

## Machine Automation Controller NJ/NX-series

### Instructions Reference Manual

NX701-1□□□

NX102-1□□□

NX102-90□□

NX1P2-1□□□□□

NX1P2-9□□□□□

NJ501-□□□□

NJ301-1□□□


NJ101-10□□

NJ101-90□□

## NOTE

1. 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.
2. 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.
3. 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. 

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.
- 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/](http://www.fa.omron.co.jp/nj_info_e/).

# Introduction

Thank you for purchasing an NJ/NX-series CPU Unit.

This manual contains information that is necessary to use the NJ/NX-series CPU Unit. Please read this manual and make sure you understand the functionality and performance of the NJ/NX-series CPU Unit 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.

- NX-series CPU Units
  - NX701-17□□
  - NX701-16□□
  - NX102-12□□
  - NX102-11□□
  - NX102-10□□
  - NX102-90□□
  - NX1P2-11□□□□
  - NX1P2-11□□□□1
  - NX1P2-10□□□□
  - NX1P2-10□□□□1
  - NX1P2-90□□□□
  - NX1P2-90□□□□1
  - NX1P2-9B□□□□
  - NX1P2-9B□□□□1
- NJ-series CPU Units
  - NJ501-□5□□
  - NJ501-□4□□
  - NJ501-□3□□
  - NJ301-12□□
  - NJ301-11□□
  - NJ101-10□□
  - NJ101-90□□

Part of the specifications and restrictions for the CPU Units are given in other manuals. Refer to *Relevant Manuals* on page 3 and *Related Manuals* on page 30.



# Relevant Manuals

The following table provides the relevant manuals for the NJ/NX-series CPU Units. Read all of the manuals that are relevant to your system configuration and application before you use the NJ/NX-series CPU Unit.

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                            |  |  |  | NJ/NX-series<br>Instructions Reference Manual | NX-series NX1P2 CPU Unit<br>Built-in I/O and Option Board User's Manual | NJ/NX-series CPU Unit<br>Software User's Manual | NJ-series CPU Unit<br>Hardware User's Manual | NX-series NX1P2 CPU Unit<br>Hardware User's Manual | NX-series NX102 CPU Unit<br>Hardware User's Manual | NX-series CPU Unit<br>Hardware User's Manual | NJ-series Robot Integrated CPU Unit<br>User's Manual | NJ-series NC Integrated Controller<br>User's Manual | NJ/NX-series Troubleshooting Manual |
|   | NJ-series CPU Unit<br>Hardware User's Manual | NX-series NX102 CPU Unit<br>Hardware User's Manual | NX-series NX1P2 CPU Unit<br>Hardware User's Manual | NJ-series CPU Unit<br>Hardware User's Manual |   |   |   |  |  |  |  |  |   |                                     |
| Introduction to NX701 CPU Units           | ○  |  |  |  |   |   |   |  |  |  |  |  |   |                                     |
| Introduction to NX102 CPU Units           |  | ○  |  |  |   |   |   |  |  |  |  |  |   |                                     |
| Introduction to NX1P2 CPU Units           |  |  | ○  |  |   |   |   |  |  |  |  |  |   |                                     |
| Introduction to NJ-series Controllers     |  |  |  | ○  |   |   |   |  |  |  |  |  |   |                                     |
| Setting devices and hardware              |  |  |  |  |   |   |   |  |  |  |  |  |   |                                     |
| Using motion control                      |  |  |  |  |   | ○   |   |  |  |  |  |  |   |                                     |
| Using EtherCAT                            | ○  | ○  | ○  | ○  |   |   |   |  |  |  |  |  |   |                                     |
| Using EtherNet/IP                         |  |  |  |  |   |   |   |  | ○  |  |  |  |   |                                     |
| Using robot control for OM-<br>RON robots |  |  |  |  |   |   |   |  |  |  | ○  |  |   |                                     |

| Purpose of use                                   | Manual                              |   |  |   |  |  |                                       |  |   |  |   |  |
|--|-------------------------------------|---|--|---|--|--|---------------------------------------|--|---|--|---|--|
|  | Basic information                   |   |  |   |  |  |                                       |  |   |  |   |  |
|  | NJ/NX-series Troubleshooting Manual | NJ/NY-series NC Integrated Controller User's Manual | NJ-series NJ Robotics CPU Unit User's Manual | NJ-series Robot Integrated CPU Unit User's Manual | NJ-series SECS/GEM CPU Units User's Manual | NJ/NX-series Database Connection CPU Units User's Manual | NX-series CPU Unit FINS User's Manual | NJ/NX-series CPU Unit OPC UA User's Manual | NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual | NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual | NJ/NX-series Motion Control Instructions Reference Manual | NJ/NX-series CPU Unit Motion Control User's Manual |
| Software settings                                |                                     |   |  |   |  |  |                                       |  |   |  |   |  |
| Using motion control                             |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Using EtherCAT                                   |                                     |   |  |   |  |  |                                       |  | ○   |  |   |  |
| Using EtherNet/IP                                |                                     |   |  |   |  |  |                                       | ○  |   |  |   |  |
| Using OPC UA                                     |                                     |   |  |   |  |  |                                       | ○  |   |  |   |  |
| Using FINS                                       |                                     |   |  |   |  |  |                                       | ○  |   |  |   |  |
| Using the database connection service            |                                     |   |  |   |  | ○  |                                       |  |   |  |   |  |
| Using the GEM Services                           |                                     |   |  |   |  |  |                                       |  |   | ○  |   |  |
| Using robot control for OM-<br>RON robots        |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Using robot control by NJ Ro-<br>botics function |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using numerical control                          |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using the NX1P2 CPU Unit<br>functions            |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Writing the user program                         |                                     |   |  |   |  |  |                                       |  |   |  |   |  |
| Using motion control                             |                                     |   |  |   |  |  |                                       |  |   | ○  | ○   |  |
| Using EtherCAT                                   |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using EtherNet/IP                                |                                     |   |  |   |  |  |                                       |  |   | ○  |   |  |
| Using OPC UA                                     |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Using FINS                                       |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Using the database connection<br>service         |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |
| Using the GEM Services                           |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using robot control for OM-<br>RON robots        |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using robot control by NJ Ro-<br>botics function |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using numerical control                          |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Programming error process-<br>ing                |                                     |   |  |   |  |  |                                       |  |   |  |   | ○  |
| Using the NX1P2 CPU Unit<br>functions            |                                     |   |  |   |  |  |                                       |  |   |  | ○   |  |

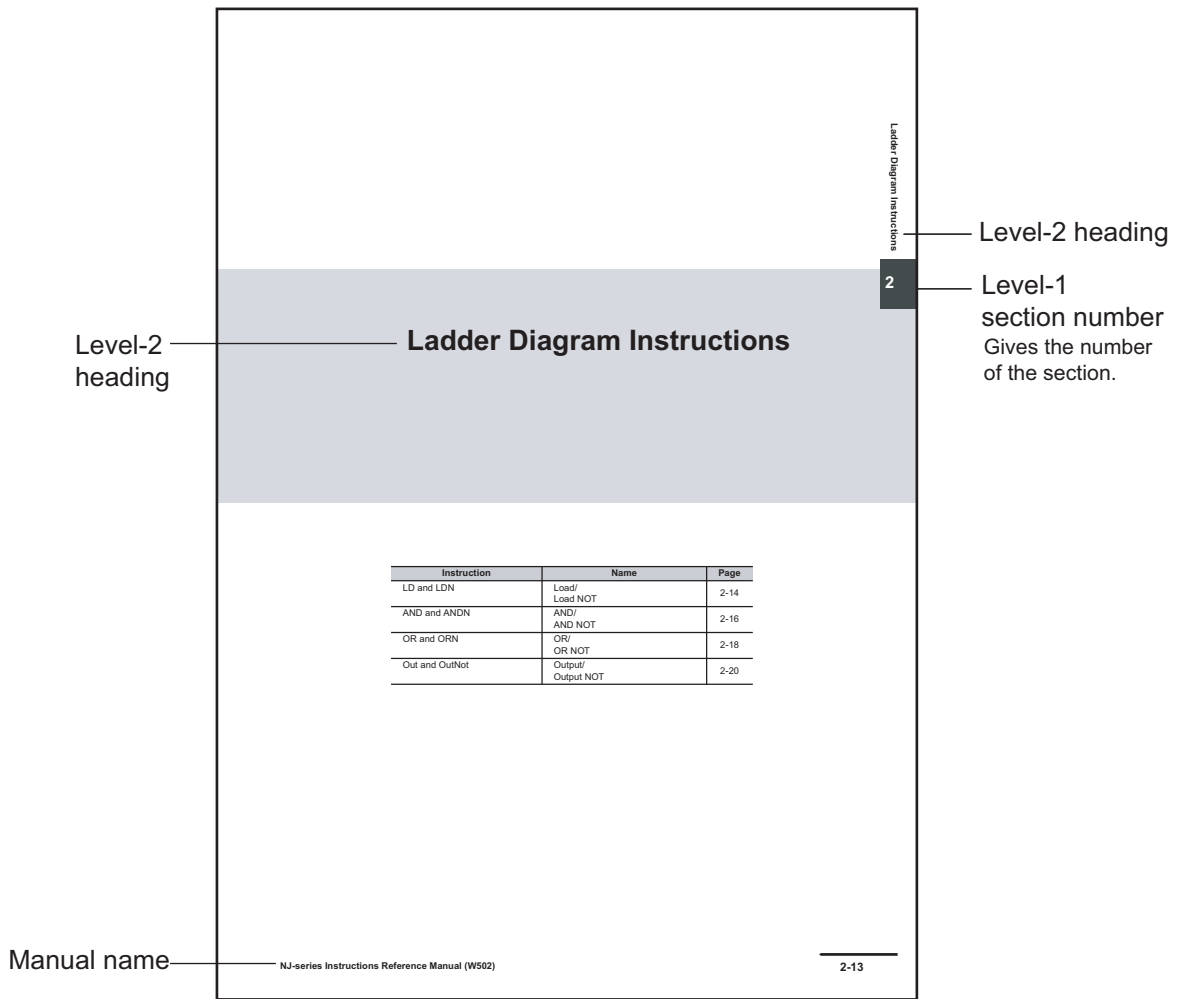
| Purpose of use   | Manual                                       |  |  |  |  |  |   |  |   |   |  |
|--|--|--|--|--|--|--|---|--|---|---|--|
|  | Basic information                            |  |  |  | NJ/NX-series<br>Troubleshooting Manual | NJ/NY-series NC Integrated Controller<br>User's Manual | NJ-series NJ Robotics CPU Unit<br>User's Manual | NJ-series Robot Integrated CPU Unit<br>User's Manual | NJ-series SECS/GEM CPU Units<br>User's Manual | NJ/NX-series Database Connection CPU Units<br>User's Manual | NX-series CPU Unit<br>FINS User's Manual |
|  | NJ-series CPU Unit<br>Hardware User's Manual | NX-series NX1P2 CPU Unit<br>Hardware User's Manual | NX-series NX102 CPU Unit<br>Hardware User's Manual | NX-series CPU Unit<br>Hardware User's Manual |  |  |   |  |   |   |  |
| Testing operation and debugging                        |  |  |  |  |  |  |   |  |   |   |  |
| Using motion control                                   |  |  |  |  |  |  |   |  |   |   |  |
| Using EtherCAT   |  |  |  |  |  |  |   |  |   |   |  |
| Using EtherNet/IP                                      |  |  |  |  |  |  |   |  |   |   |  |
| Using OPC UA   |  |  |  |  |  |  |   |  |   |   |  |
| Using FINS   |  |  |  |  |  |  |   |  |   |   |  |
| Using the database connection service                  |  |  |  |  |  |  |   |  |   |   |  |
| Using the GEM Services                                 |  |  |  |  |  |  |   |  |   |   |  |
| Using robot control for OM-<br>RON robots              |  |  |  |  |  |  |   |  |   |   |  |
| Using robot control by NJ Ro-<br>botics function       |  |  |  |  |  |  |   |  |   |   |  |
| Using numerical control                                |  |  |  |  |  |  |   |  |   |   |  |
| Using the NX1P2 CPU Unit<br>functions                  |  |  |  |  |  |  |   |  |   |   |  |
| Learning about error manage-<br>ment and corrections*1 |  |  |  |  |  |  |   |  |   |   |  |
| Maintenance  |  |  |  |  |  |  |   |  |   |   |  |
| Using motion control                                   |  |  |  |  |  |  |   |  |   |   |  |
| Using EtherCAT   |  |  |  |  |  |  |   |  |   |   |  |
| Using EtherNet/IP                                      |  |  |  |  |  |  |   |  |   |   |  |

\*1. Refer to the *NJ/NX-series Troubleshooting Manual (Cat. No. W503)* for the error management concepts and the error items. However, refer to the manuals that are indicated with triangles for details on errors corresponding to the products with the manuals that are indicated with triangles.

# Manual Structure

## Page Structure

The following page structure is used in this manual.



2 Instruction Descriptions

## 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     | ---    | <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">                     Variable<br/> <br/>                     Upward differentiation                 </div> <div style="text-align: center;">                     Variable<br/> <br/>                     Downward differentiation                 </div> </div> | result:=vBool1 OR vBool2;     |
| ORN         | OR NOT | ---    | <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">                     Variable<br/> <br/>                     Upward differentiation                 </div> <div style="text-align: center;">                     Variable<br/> <br/>                     Downward differentiation                 </div> </div> | 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.

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

Level-3 heading

Level-1 heading

Level-2 heading

Level-3 heading

Give the current headings.

Level-1 section number

2

Give the number of the section.

Manual name

**Note** These pages are for illustrative purposes only. They may not literally appear in this manual.

## Special Information

Special information in this manual is classified as follows:



### Precautions for Safe Use

Precautions on what to do and what not to do to ensure safe usage of the product.



### Precautions for Correct Use

Precautions on what to do and what not to do to ensure proper operation and performance.



### Additional Information

Additional information to read as required.

This information is provided to increase understanding or make operation easier.



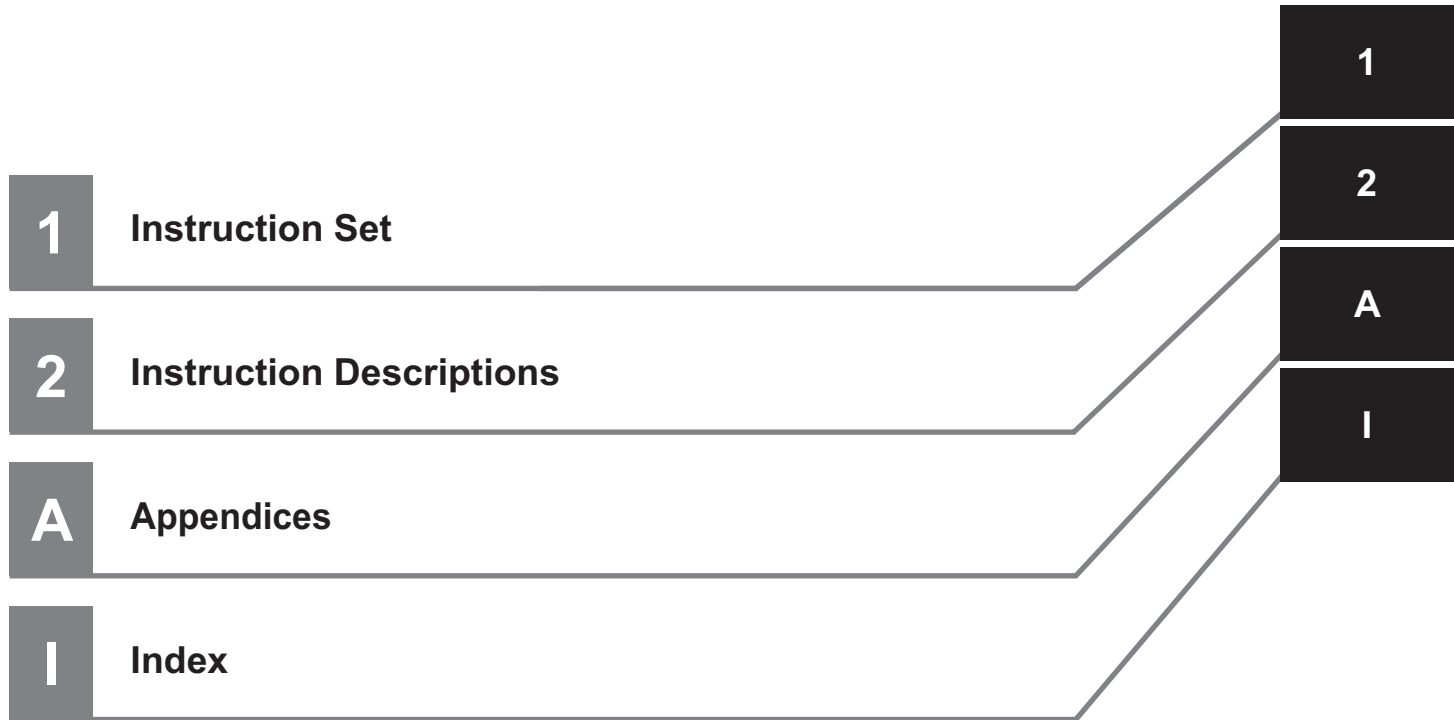
### Version Information

Information on differences in specifications and functionality for Controller with different unit versions and for different versions of the Sysmac Studio is given.



# Sections in this Manual

---



# CONTENTS

---

|   |           |
|---|-----------|
| <b>Introduction .....</b>                                   | <b>1</b>  |
| Intended Audience .....                                     | 1         |
| Applicable Products .....                                   | 1         |
| <b>Relevant Manuals.....</b>                                | <b>3</b>  |
| <b>Manual Structure.....</b>                                | <b>6</b>  |
| Page Structure .....  | 6         |
| Special Information .....                                   | 7         |
| <b>Sections in this Manual .....</b>                        | <b>9</b>  |
| <b>Terms and Conditions Agreement.....</b>                  | <b>18</b> |
| Warranty, Limitations of Liability .....                    | 18        |
| Application Considerations .....                            | 19        |
| Disclaimers .....   | 19        |
| <b>Safety Precautions.....</b>                              | <b>21</b> |
| <b>Precautions for Safe Use .....</b>                       | <b>22</b> |
| <b>Precautions for Correct Use .....</b>                    | <b>23</b> |
| <b>Regulations and Standards .....</b>                      | <b>24</b> |
| <b>Versions .....</b>                                       | <b>25</b> |
| Checking Versions .....                                     | 25        |
| Unit Versions of CPU Units and Sysmac Studio Versions ..... | 29        |
| <b>Related Manuals.....</b>                                 | <b>30</b> |
| <b>Revision History.....</b>                                | <b>35</b> |

## Section 1    Instruction Set

---

|  |            |
|--|------------|
| <b>Instruction Set.....</b>              | <b>1-2</b> |
| Ladder Diagram Instructions.....         | 1-2        |
| ST Statement Instructions .....          | 1-2        |
| Sequence Input Instructions .....        | 1-2        |
| Sequence Output Instructions .....       | 1-3        |
| Sequence Control Instructions.....       | 1-3        |
| Comparison Instructions .....            | 1-3        |
| Timer Instructions .....                 | 1-5        |
| Counter Instructions.....                | 1-5        |
| Math Instructions .....                  | 1-5        |
| BCD Conversion Instructions.....         | 1-6        |
| Data Type Conversion Instructions.....   | 1-7        |
| Bit String Processing Instructions ..... | 1-8        |
| Selection Instructions.....              | 1-9        |
| Data Movement Instructions .....         | 1-9        |
| Shift Instructions .....                 | 1-10       |
| Conversion Instructions .....            | 1-10       |
| Stack and Table Instructions.....        | 1-12       |
| FCS Instructions .....                   | 1-13       |



|  |      |
|--|------|
| Text String Instructions .....               | 1-13 |
| Time and Time of Day Instructions .....      | 1-14 |
| Analog Control Instructions.....             | 1-15 |
| System Control Instructions.....             | 1-15 |
| Program Control Instructions.....            | 1-17 |
| EtherCAT Communications Instructions .....   | 1-17 |
| IO-Link Communications Instructions .....    | 1-17 |
| EtherNet/IP Communications Instructions..... | 1-17 |
| Serial Communications Instructions .....     | 1-19 |
| SD Memory Card Instructions.....             | 1-20 |
| Time Stamp Instructions .....                | 1-20 |
| Other Instructions .....                     | 1-20 |

## Section 2 Instruction Descriptions

|  |              |
|--|--------------|
| <b>Using this Section.....</b>                                     | <b>2-3</b>   |
| Items .....  | 2-3          |
| Common Variables.....  | 2-5          |
| Valid Ranges and Default Values of Variables.....                  | 2-9          |
| Derivative Data Types (Enumerations, Structures, and Unions) ..... | 2-10         |
| Array Specifications .....   | 2-11         |
| Others .....   | 2-12         |
| <b>Ladder Diagram Instructions .....</b>                           | <b>2-13</b>  |
| LD and LDN .....   | 2-14         |
| AND and ANDN .....   | 2-17         |
| OR and ORN .....   | 2-20         |
| Out and OutNot.....  | 2-23         |
| <b>ST Statement Instructions.....</b>                              | <b>2-27</b>  |
| IF.....  | 2-28         |
| CASE .....   | 2-32         |
| WHILE .....  | 2-36         |
| REPEAT .....   | 2-39         |
| EXIT .....   | 2-42         |
| RETURN.....  | 2-45         |
| FOR .....  | 2-46         |
| <b>Sequence Input Instructions.....</b>                            | <b>2-47</b>  |
| R_TRIG (Up) and F_TRIG (Down) .....                                | 2-48         |
| TestABit and TestABitN.....  | 2-52         |
| <b>Sequence Output Instructions.....</b>                           | <b>2-55</b>  |
| RS .....   | 2-56         |
| SR.....  | 2-59         |
| Set and Reset.....   | 2-62         |
| SetBits and ResetBits.....   | 2-66         |
| SetABit and ResetABit.....   | 2-69         |
| OutABit .....  | 2-71         |
| <b>Sequence Control Instructions.....</b>                          | <b>2-73</b>  |
| End .....  | 2-74         |
| RETURN.....  | 2-75         |
| MC and MCR .....   | 2-76         |
| JMP .....  | 2-89         |
| FOR and NEXT.....  | 2-91         |
| BREAK.....   | 2-98         |
| <b>Comparison Instructions.....</b>                                | <b>2-101</b> |
| EQ (=) .....   | 2-102        |
| NE (<>) .....  | 2-105        |
| LT (<), LE (<=), GT (>), and GE (>=) .....                         | 2-108        |
| EQascii .....  | 2-111        |
| NEascii.....   | 2-113        |
| LTascii, LEascii, GTascii, and GEascii .....                       | 2-115        |
| Cmp .....  | 2-118        |

|   |              |
|---|--------------|
| ZoneCmp .....   | 2-120        |
| TableCmp.....   | 2-122        |
| AryCmpEQ and AryCmpNE.....                                  | 2-125        |
| AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE .....            | 2-127        |
| AryCmpEQV and AryCmpNEV.....                                | 2-130        |
| AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV.....         | 2-132        |
| <b>Timer Instructions .....</b>                             | <b>2-135</b> |
| TON .....   | 2-136        |
| TOF .....   | 2-142        |
| TP .....  | 2-145        |
| AccumulationTimer .....                                     | 2-148        |
| Timer.....  | 2-152        |
| <b>Counter Instructions.....</b>                            | <b>2-155</b> |
| CTD .....   | 2-156        |
| CTD_**.....   | 2-158        |
| CTU .....   | 2-161        |
| CTU_**.....   | 2-164        |
| CTUD.....   | 2-167        |
| CTUD_**.....  | 2-172        |
| <b>Math Instructions .....</b>                              | <b>2-177</b> |
| ADD (+).....  | 2-179        |
| AddOU (+OU).....  | 2-183        |
| SUB (-).....  | 2-187        |
| SubOU (-OU).....  | 2-190        |
| MUL (*).....  | 2-194        |
| MulOU (*OU).....  | 2-198        |
| DIV (/).....  | 2-202        |
| MOD.....  | 2-205        |
| ABS.....  | 2-207        |
| RadToDeg and DegToRad.....                                  | 2-209        |
| SIN, COS, and TAN.....                                      | 2-211        |
| ASIN, ACOS, and ATAN.....                                   | 2-214        |
| SQRT.....   | 2-217        |
| LN and LOG.....   | 2-220        |
| EXP.....  | 2-224        |
| EXPT (**).....  | 2-226        |
| Inc and Dec.....  | 2-232        |
| Rand.....   | 2-234        |
| AryAdd.....   | 2-236        |
| AryAddV.....  | 2-238        |
| ArySub.....   | 2-240        |
| ArySubV.....  | 2-242        |
| AryMean.....  | 2-244        |
| ArySD.....  | 2-246        |
| ModReal.....  | 2-248        |
| Fraction.....   | 2-250        |
| CheckReal.....  | 2-252        |
| <b>BCD Conversion Instructions .....</b>                    | <b>2-255</b> |
| **_BCD_TO_***.....  | 2-256        |
| **_TO_BCD_***.....  | 2-259        |
| BCD_TO_**.....  | 2-262        |
| BCDsToBin.....  | 2-265        |
| BinToBCDs_**.....   | 2-268        |
| AryToBCD.....   | 2-271        |
| AryToBin.....   | 2-273        |
| <b>Data Type Conversion Instructions.....</b>               | <b>2-275</b> |
| **_TO_*** (Integer-to-Integer Conversion Group).....        | 2-277        |
| **_TO_*** (Integer-to-Bit String Conversion Group).....     | 2-280        |
| **_TO_*** (Integer-to-Real Number Conversion Group).....    | 2-283        |
| **_TO_*** (Bit String-to-Integer Conversion Group).....     | 2-285        |
| **_TO_*** (Bit String-to-Bit String Conversion Group).....  | 2-288        |
| **_TO_*** (Bit String-to-Real Number Conversion Group)..... | 2-290        |

|   |              |
|---|--------------|
| **_TO_*** (Real Number-to-Integer Conversion Group).....        | 2-292        |
| **_TO_*** (Real Number-to-Bit String Conversion Group).....     | 2-295        |
| **_TO_*** (Real Number-to-Real Number Conversion Group).....    | 2-297        |
| **_TO_STRING (Integer-to-Text String Conversion Group).....     | 2-299        |
| **_TO_STRING (Bit String-to-Text String Conversion Group).....  | 2-301        |
| **_TO_STRING (Real Number-to-Text String Conversion Group)..... | 2-303        |
| RealToFormatString.....   | 2-305        |
| LrealToFormatString.....  | 2-311        |
| STRING_TO_** (Text String-to-Integer Conversion Group).....     | 2-317        |
| STRING_TO_** (Text String-to-Bit String Conversion Group).....  | 2-319        |
| STRING_TO_** (Text String-to-Real Number Conversion Group)..... | 2-321        |
| TO_** (Integer Conversion Group).....                           | 2-325        |
| TO_** (Bit String Conversion Group).....                        | 2-327        |
| TO_** (Real Number Conversion Group).....                       | 2-329        |
| EnumToNum.....  | 2-331        |
| NumToEnum.....  | 2-333        |
| TRUNC, Round, and RoundUp.....                                  | 2-336        |
| <b>Bit String Processing Instructions.....</b>                  | <b>2-339</b> |
| AND (&), OR, and XOR.....                                       | 2-340        |
| XORN.....   | 2-343        |
| NOT.....  | 2-345        |
| AryAnd, AryOr, AryXor, and AryXorN.....                         | 2-347        |
| <b>Selection Instructions.....</b>                              | <b>2-351</b> |
| SEL.....  | 2-352        |
| MUX.....  | 2-354        |
| LIMIT.....  | 2-357        |
| Band.....   | 2-359        |
| Zone.....   | 2-361        |
| MAX and MIN.....  | 2-363        |
| AryMax and AryMin.....  | 2-365        |
| ArySearch.....  | 2-368        |
| <b>Data Movement Instructions.....</b>                          | <b>2-371</b> |
| MOVE.....   | 2-372        |
| MoveBit.....  | 2-375        |
| MoveDigit.....  | 2-377        |
| TransBits.....  | 2-379        |
| MemCopy.....  | 2-381        |
| SetBlock.....   | 2-383        |
| Exchange.....   | 2-385        |
| AryExchange.....  | 2-387        |
| AryMove.....  | 2-389        |
| Clear.....  | 2-391        |
| Copy**ToNum (Bit String to Signed Integer).....                 | 2-393        |
| Copy**To*** (Bit String to Real Number).....                    | 2-395        |
| CopyNumTo** (Signed Integer to Bit String).....                 | 2-397        |
| CopyNumTo** (Signed Integer to Real Number).....                | 2-399        |
| Copy**To*** (Real Number to Bit String).....                    | 2-401        |
| Copy**ToNum (Real Number to Signed Integer).....                | 2-403        |
| <b>Shift Instructions.....</b>                                  | <b>2-405</b> |
| AryShiftReg.....  | 2-406        |
| AryShiftRegLR.....  | 2-408        |
| ArySHL and ArySHR.....  | 2-411        |
| SHL and SHR.....  | 2-414        |
| NSHLC and NSHRC.....  | 2-417        |
| ROL and ROR.....  | 2-420        |
| <b>Conversion Instructions.....</b>                             | <b>2-423</b> |
| Swap.....   | 2-425        |
| Neg.....  | 2-427        |
| Decoder.....  | 2-429        |
| Encoder.....  | 2-432        |
| BitCnt.....   | 2-434        |
| ColmToLine_**.....  | 2-435        |

|  |              |
|--|--------------|
| LineToColm .....                               | 2-437        |
| Gray .....                                     | 2-439        |
| UTF8ToSJIS .....                               | 2-444        |
| SJISToUTF8 .....                               | 2-446        |
| PWLApprox and PWLApproxNoLineChk .....         | 2-448        |
| PWLLineChk .....                               | 2-454        |
| MovingAverage .....                            | 2-457        |
| DispartReal .....                              | 2-464        |
| UniteReal .....                                | 2-467        |
| NumToDecString and NumToHexString .....        | 2-469        |
| HexStringToNum_** .....                        | 2-472        |
| FixNumToString .....                           | 2-474        |
| StringToFixNum .....                           | 2-476        |
| DtToString .....                               | 2-479        |
| DateToString .....                             | 2-481        |
| TodToString .....                              | 2-483        |
| GrayToBin_** and BinToGray_** .....            | 2-485        |
| StringToAry .....                              | 2-488        |
| AryToString .....                              | 2-490        |
| DispartDigit .....                             | 2-492        |
| UniteDigit_** .....                            | 2-494        |
| Dispart8Bit .....                              | 2-496        |
| Unite8Bit_** .....                             | 2-498        |
| ToAryByte .....                                | 2-500        |
| AryByteTo .....                                | 2-506        |
| SizeOfAry .....                                | 2-512        |
| PackWord .....                                 | 2-514        |
| PackDword .....                                | 2-516        |
| LOWER_BOUND and UPPER_BOUND .....              | 2-518        |
| <b>Stack and Table Instructions .....</b>      | <b>2-523</b> |
| StackPush .....                                | 2-524        |
| StackFIFO and StackLIFO .....                  | 2-533        |
| StackIns .....                                 | 2-536        |
| StackDel .....                                 | 2-539        |
| RecSearch .....                                | 2-541        |
| RecRangeSearch .....                           | 2-546        |
| RecSort .....                                  | 2-551        |
| RecNum .....                                   | 2-557        |
| RecMax and RecMin .....                        | 2-560        |
| <b>FCS Instructions .....</b>                  | <b>2-565</b> |
| StringSum .....                                | 2-566        |
| StringLRC .....                                | 2-568        |
| StringCRCCCITT .....                           | 2-570        |
| StringCRC16 .....                              | 2-572        |
| AryLRC_** .....                                | 2-574        |
| AryCRCCCITT .....                              | 2-576        |
| AryCRC16 .....                                 | 2-578        |
| <b>Text String Instructions .....</b>          | <b>2-581</b> |
| CONCAT .....                                   | 2-582        |
| LEFT and RIGHT .....                           | 2-584        |
| MID .....                                      | 2-587        |
| FIND .....                                     | 2-589        |
| LEN .....                                      | 2-591        |
| REPLACE .....                                  | 2-593        |
| DELETE .....                                   | 2-595        |
| INSERT .....                                   | 2-597        |
| GetByteLen .....                               | 2-599        |
| ClearString .....                              | 2-600        |
| ToUCase and ToLCase .....                      | 2-602        |
| TrimL and TrimR .....                          | 2-604        |
| AddDelimiter .....                             | 2-606        |
| SubDelimiter .....                             | 2-618        |
| <b>Time and Time of Day Instructions .....</b> | <b>2-631</b> |

|   |              |
|---|--------------|
| ADD_TIME.....                           | 2-633        |
| ADD_TOD_TIME.....                       | 2-635        |
| ADD_DT_TIME.....                        | 2-637        |
| SUB_TIME.....                           | 2-639        |
| SUB_TOD_TIME.....                       | 2-641        |
| SUB_TOD_TOD.....                        | 2-643        |
| SUB_DATE_DATE.....                      | 2-645        |
| SUB_DT_DT.....                          | 2-646        |
| SUB_DT_TIME.....                        | 2-648        |
| MULTIME.....                            | 2-650        |
| DIVTIME.....                            | 2-652        |
| CONCAT_DATE_TOD.....                    | 2-654        |
| DT_TO_TOD.....                          | 2-656        |
| DT_TO_DATE.....                         | 2-658        |
| SetTime.....                            | 2-660        |
| GetTime.....                            | 2-662        |
| DtToSec.....                            | 2-664        |
| DateToSec.....                          | 2-666        |
| TodToSec.....                           | 2-668        |
| SecToDt.....                            | 2-670        |
| SecToDate.....                          | 2-672        |
| SecToTod.....                           | 2-674        |
| TimeToNanoSec.....                      | 2-676        |
| TimeToSec.....                          | 2-677        |
| NanoSecToTime.....                      | 2-679        |
| SecToTime.....                          | 2-680        |
| ChkLeapYear.....                        | 2-682        |
| GetDaysOfMonth.....                     | 2-683        |
| DaysToMonth.....                        | 2-686        |
| GetDayOfWeek.....                       | 2-688        |
| GetWeekOfYear.....                      | 2-690        |
| DtToDateStruct.....                     | 2-692        |
| DateStructToDt.....                     | 2-695        |
| TruncTime.....                          | 2-698        |
| TruncDt.....                            | 2-702        |
| TruncTod.....                           | 2-706        |
| <b>Analog Control Instructions.....</b> | <b>2-711</b> |
| PIDAT.....                              | 2-712        |
| PIDAT_HeatCool.....                     | 2-742        |
| TimeProportionalOut.....                | 2-779        |
| LimitAlarm_**.....                      | 2-799        |
| LimitAlarmDv_**.....                    | 2-804        |
| LimitAlarmDvStbySeq_**.....             | 2-809        |
| ScaleTrans.....                         | 2-827        |
| AC_StepProgram.....                     | 2-830        |
| <b>System Control Instructions.....</b> | <b>2-857</b> |
| TraceSamp.....                          | 2-859        |
| TraceTrig.....                          | 2-862        |
| GetTraceStatus.....                     | 2-865        |
| SetAlarm.....                           | 2-868        |
| ResetAlarm.....                         | 2-873        |
| GetAlarm.....                           | 2-875        |
| ResetPLCError.....                      | 2-877        |
| GetPLCError.....                        | 2-880        |
| ResetCJBError.....                      | 2-882        |
| GetCJBError.....                        | 2-884        |
| GetEIPErr.....                          | 2-886        |
| ResetMCErr.....                         | 2-888        |
| GetMCErr.....                           | 2-894        |
| ResetECErr.....                         | 2-896        |
| GetECErr.....                           | 2-898        |
| ResetNXBErr.....                        | 2-901        |
| GetNXBErr.....                          | 2-903        |
| GetNXUnitError.....                     | 2-905        |

|  |               |
|--|---------------|
| SetInfo .....  | 2-912         |
| ResetUnit .....                                      | 2-914         |
| GetNTPStatus .....                                   | 2-919         |
| RestartNXUnit .....                                  | 2-921         |
| NX_ChangeWriteMode .....                             | 2-927         |
| NX_SaveParam .....                                   | 2-933         |
| PLC_ReadTotalPowerOnTime .....                       | 2-939         |
| NX_ReadTotalPowerOnTime .....                        | 2-942         |
| <b>Program Control Instructions .....</b>            | <b>2-951</b>  |
| PrgStart .....                                       | 2-952         |
| PrgStop .....  | 2-961         |
| PrgStatus .....                                      | 2-980         |
| <b>EtherCAT Communications Instructions .....</b>    | <b>2-985</b>  |
| EC_CoESDOWrite .....                                 | 2-986         |
| EC_CoESDORead .....                                  | 2-989         |
| EC_StartMon .....                                    | 2-995         |
| EC_StopMon .....                                     | 2-1001        |
| EC_SaveMon .....                                     | 2-1003        |
| EC_CopyMon .....                                     | 2-1005        |
| EC_DisconnectSlave .....                             | 2-1007        |
| EC_ConnectSlave .....                                | 2-1015        |
| EC_ChangeEnableSetting .....                         | 2-1017        |
| NX_WriteObj .....                                    | 2-1037        |
| NX_ReadObj .....                                     | 2-1053        |
| <b>IO-Link Communications Instruction .....</b>      | <b>2-1061</b> |
| IOL_ReadObj .....                                    | 2-1062        |
| IOL_WriteObj .....                                   | 2-1071        |
| <b>EtherNet/IP Communications Instructions .....</b> | <b>2-1081</b> |
| CIPOpen .....  | 2-1083        |
| CIPOpenWithDataSize .....                            | 2-1093        |
| CIPRead .....  | 2-1097        |
| CIPWrite .....                                       | 2-1103        |
| CIPSend .....  | 2-1109        |
| CIPCclose .....                                      | 2-1114        |
| CIPUCMMRead .....                                    | 2-1117        |
| CIPUCMMWrite .....                                   | 2-1123        |
| CIPUCMMSend .....                                    | 2-1130        |
| SkUDPCreate .....                                    | 2-1141        |
| SkUDPRcv .....                                       | 2-1149        |
| SkUDPSend .....                                      | 2-1152        |
| SkTCPAccept .....                                    | 2-1155        |
| SkTCPConnect .....                                   | 2-1158        |
| SkTCPRcv .....                                       | 2-1167        |
| SkTCPSend .....                                      | 2-1170        |
| SkGetTCPStatus .....                                 | 2-1173        |
| SkClose .....  | 2-1176        |
| SkClearBuf .....                                     | 2-1179        |
| SkSetOption .....                                    | 2-1182        |
| ModbusTCPcmd .....                                   | 2-1187        |
| ModbusTCPRead .....                                  | 2-1195        |
| ModbusTCPWrite .....                                 | 2-1203        |
| ChangeIPAdr .....                                    | 2-1211        |
| ChangeFTPAccount .....                               | 2-1220        |
| ChangeNTPServerAdr .....                             | 2-1224        |
| FTPGetFileList .....                                 | 2-1229        |
| FTPGetFile .....                                     | 2-1243        |
| FTPPutFile .....                                     | 2-1252        |
| FTPRemoveFile .....                                  | 2-1263        |
| FTPRemoveDir .....                                   | 2-1273        |
| <b>Serial Communications Instructions .....</b>      | <b>2-1277</b> |
| ExecPMCR .....                                       | 2-1278        |
| SerialSend .....                                     | 2-1292        |

|   |               |
|---|---------------|
| SerialRcv and SerialRcvNoClear .....    | 2-1303        |
| SendCmd .....                           | 2-1318        |
| NX_SerialSend .....                     | 2-1330        |
| NX_SerialRcv .....                      | 2-1343        |
| NX_ModbusRtuCmd .....                   | 2-1358        |
| NX_ModbusRtuRead .....                  | 2-1369        |
| NX_ModbusRtuWrite .....                 | 2-1380        |
| NX_SerialSigCtl .....                   | 2-1391        |
| NX_SerialSigRead .....                  | 2-1399        |
| NX_SerialStatusRead .....               | 2-1404        |
| NX_SerialBufClear .....                 | 2-1409        |
| NX_SerialStartMon .....                 | 2-1419        |
| NX_SerialStopMon .....                  | 2-1424        |
| <b>SD Memory Card Instructions.....</b> | <b>2-1429</b> |
| FileWriteVar .....                      | 2-1430        |
| FileReadVar .....                       | 2-1436        |
| FileOpen .....                          | 2-1441        |
| FileClose .....                         | 2-1445        |
| FileSeek .....                          | 2-1448        |
| FileRead .....                          | 2-1451        |
| FileWrite .....                         | 2-1459        |
| FileGets .....                          | 2-1467        |
| FilePuts .....                          | 2-1475        |
| FileCopy .....                          | 2-1484        |
| FileRemove .....                        | 2-1492        |
| FileRename .....                        | 2-1497        |
| DirCreate .....                         | 2-1503        |
| DirRemove .....                         | 2-1506        |
| BackupToMemoryCard .....                | 2-1509        |
| <b>Time Stamp Instructions .....</b>    | <b>2-1523</b> |
| NX_DOutTimeStamp .....                  | 2-1524        |
| NX_AryDOutTimeStamp .....               | 2-1530        |
| <b>Other Instructions .....</b>         | <b>2-1539</b> |
| ReadNbit ** .....                       | 2-1540        |
| WriteNbit ** .....                      | 2-1542        |
| ChkRange .....                          | 2-1544        |
| GetMyTaskStatus .....                   | 2-1547        |
| GetMyTaskInterval .....                 | 2-1550        |
| Task_IsActive .....                     | 2-1552        |
| Lock and Unlock .....                   | 2-1554        |
| ActEventTask .....                      | 2-1560        |
| Get**Clk .....                          | 2-1567        |
| Get**Cnt .....                          | 2-1569        |

## Appendices

---

|  |             |
|--|-------------|
| <b>A-1 Error Codes That You Can Check with ErrorID .....</b>                                   | <b>A-2</b>  |
| <b>A-2 Error Codes .....</b>   | <b>A-31</b> |
| <b>A-3 Instructions You Cannot Use in Event Tasks .....</b>                                    | <b>A-32</b> |
| <b>A-4 Instructions Related to NX Message Communications Errors .....</b>                      | <b>A-35</b> |
| <b>A-5 SDO Abort Codes .....</b>   | <b>A-36</b> |
| <b>A-6 Version Information .....</b>   | <b>A-37</b> |
| A-6-1 Instructions with Specifications Changes and New Instructions for Version Upgrades ..... | A-37        |
| A-6-2 What to Do If An Error Message Says the Instruction May Cause Unintended Operations .... | A-41        |

## 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.

- *NX-series CPU Unit Hardware User's Manual (Cat. No. W535)*
- *NX-series NX102 CPU Unit Hardware User's Manual (Cat. No. W593)*
- *NX-series NX1P2 CPU Unit Hardware User's Manual (Cat. No. W578)*
- *NJ-series CPU Unit Hardware User's Manual (Cat No. W500)*

# Precautions for Safe Use

---

Refer to the following manuals for precautions for safe use.

- *NX-series CPU Unit Hardware User's Manual (Cat. No. W535)*
- *NX-series NX102 CPU Unit Hardware User's Manual (Cat. No. W593)*
- *NX-series NX1P2 CPU Unit Hardware User's Manual (Cat. No. W578)*
- *NJ-series CPU Unit Hardware User's Manual (Cat No. W500)*

# Precautions for Correct Use

---

Refer to the following manuals for precautions for correct use.

- *NX-series CPU Unit Hardware User's Manual (Cat. No. W535)*
- *NX-series NX102 CPU Unit Hardware User's Manual (Cat. No. W593)*
- *NX-series NX1P2 CPU Unit Hardware User's Manual (Cat. No. W578)*
- *NJ-series CPU Unit Hardware User's Manual (Cat No. W500)*

# Regulations and Standards

---

Refer to the following manuals for regulations and standards.

- *NX-series CPU Unit Hardware User's Manual (Cat. No. W535)*
- *NX-series NX102 CPU Unit Hardware User's Manual (Cat. No. W593)*
- *NX-series NX1P2 CPU Unit Hardware User's Manual (Cat. No. W578)*
- *NJ-series CPU Unit Hardware User's Manual (Cat No. W500)*

# Versions

Hardware revisions and unit versions are used to manage the hardware and software in NJ/NX-series Units 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

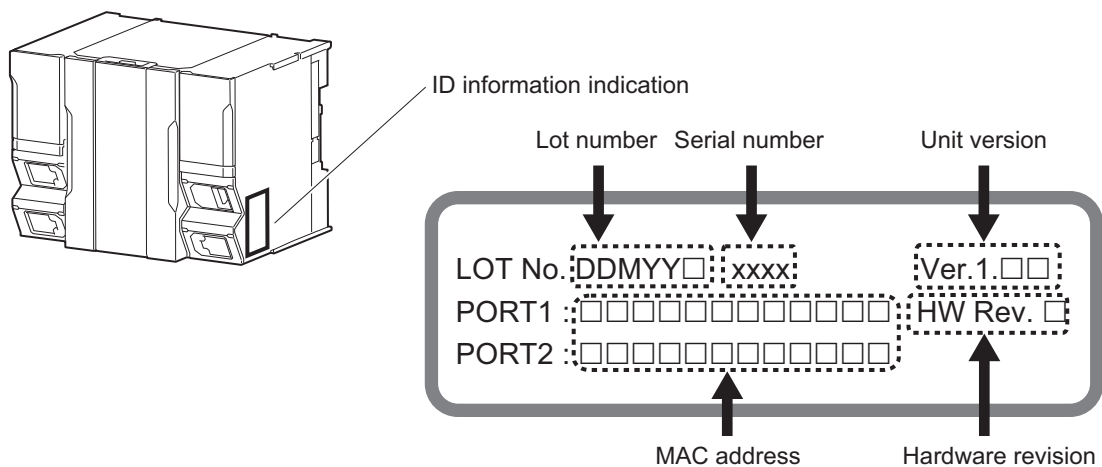
You can check versions on the ID information indications or with the Sysmac Studio.

### Checking Unit Versions on ID Information Indications

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

- **For NX701**

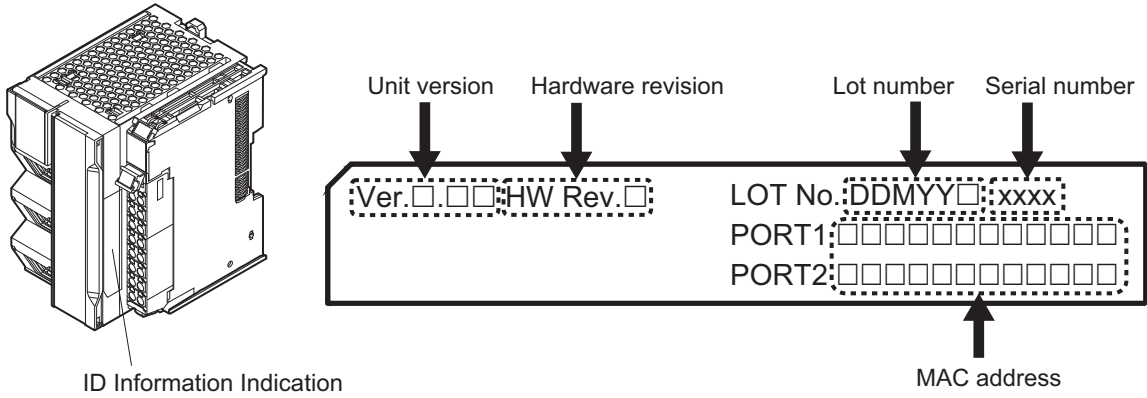
The ID information on an NX-series NX701-□□□□ CPU Unit is shown below.



**Note** The hardware revision is not displayed for the Unit whose hardware revision is blank.

- **For NX102**

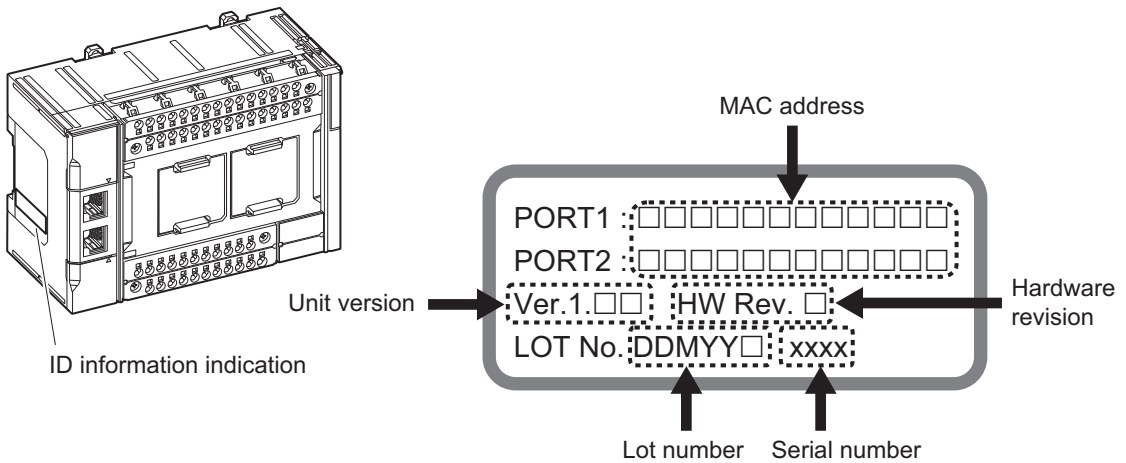
The ID information on an NX-series NX102-□□□□ CPU Unit is shown below.



**Note** The hardware revision is not displayed for the Unit whose hardware revision is blank.

● **For NX1P2**

The ID information on an NX-series NX1P2-□□□□□□ CPU Unit is shown below.

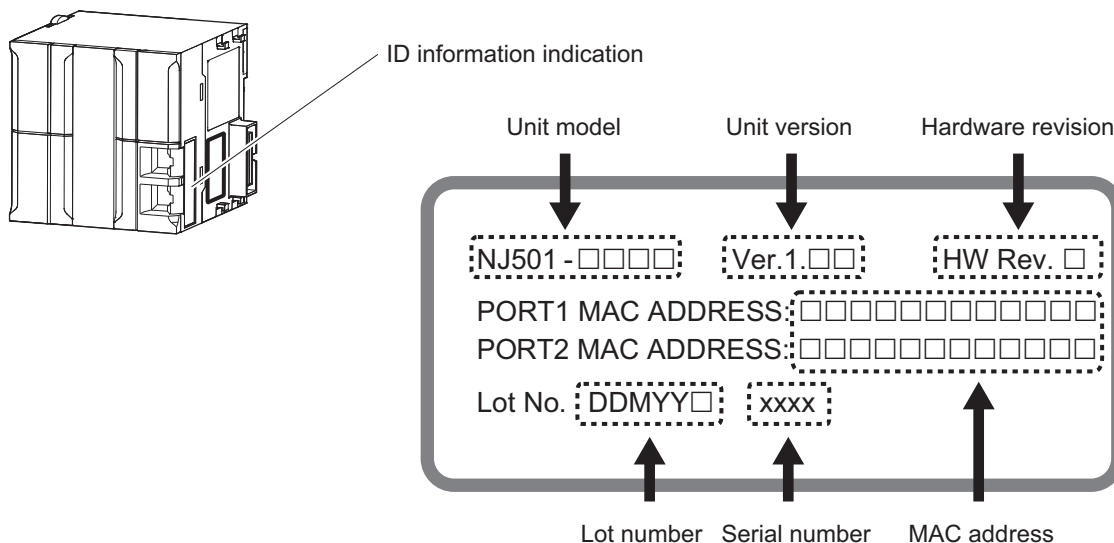


**Note** The hardware revision is not displayed for the Unit that the hardware revision is in blank.

● **For NJ-series**

The ID information on an NJ-series NJ501-□□□□ CPU Unit is shown below.





**Note** The hardware revision is not displayed for the Unit that the hardware revision is in blank.

## 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 NX-series CPU Unit

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

| Model      | Unit for which unit version can be checked       |
|------------|--|
| NX701-□□□□ | CPU Unit   |
| NX102-□□□□ | CPU Unit and NX Unit on CPU Rack                 |
| NX1P2-□□□□ | CPU Unit, NX Unit on CPU Rack, and Option Boards |

- 1 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.

### ● Checking the Unit Version of an NJ-series CPU Unit

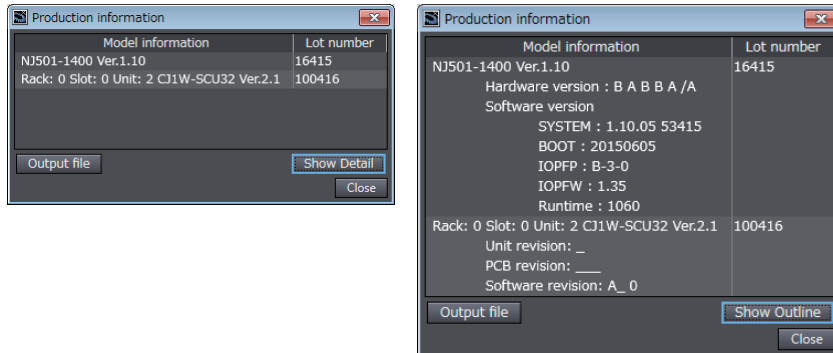
You can use the Production Information while the Sysmac Studio is online to check the unit version of a Unit. You can do this for the CPU Unit, CJ-series Special I/O Units, and CJ-series CPU Bus Units. You cannot check the unit versions of CJ-series Basic I/O Units with the Sysmac Studio.

- 1 Double-click **CPU Rack** under **Configurations and Setup - CPU/Expansion Racks** in the Multiview Explorer. Or, right-click **CPU Rack** under **Configurations and Setup - CPU/Expansion Racks** in the Multiview Explorer and select **Edit** from the menu.  
The Unit Editor is displayed.
- 2 Right-click any open space in the Unit Editor 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 various versions. The Outline View displays only the unit version.

**Note** The hardware revision is separated by “/” and displayed on the right of the hardware version. The hardware revision is not displayed for the Unit that the hardware revision is in blank.

## ● 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.

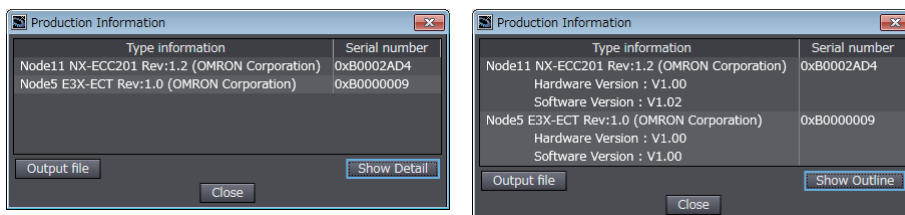
Use the following procedure to check the unit version.

- 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

## Unit Versions of CPU Units and Sysmac Studio Versions

The functions that are supported depend on the unit version of the NJ/NX-series CPU Unit. The version of Sysmac Studio that supports the functions that were added for an upgrade is also required to use those functions.

Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for the relationship between the unit versions of the CPU Units and the Sysmac Studio versions and for the functions that are supported by each unit version.

# Related Manuals

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

| Manual name  | Cat. No. | Model numbers  | Application  | Description   |
|--|----------|--|--|---|
| NX-series CPU Unit<br>Hardware User's Manual             | W535     | NX701-□□□□   | Learning the basic specifications of the NX701 CPU Units, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.     | An introduction to the entire NX701 system is provided along with the following information on the CPU Unit. <ul style="list-style-type: none"> <li>• Features and system configuration</li> <li>• Introduction</li> <li>• Part names and functions</li> <li>• General specifications</li> <li>• Installation and wiring</li> <li>• Maintenance and inspection</li> </ul>     |
| NX-series<br>NX102 CPU Unit<br>Hardware<br>User's Manual | W593     | NX102-□□□□   | Learning the basic specifications of the NX102 CPU Units, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.     | An introduction to the entire NX102 system is provided along with the following information on the CPU Unit. <ul style="list-style-type: none"> <li>• Features and system configuration</li> <li>• Introduction</li> <li>• Part names and functions</li> <li>• General specifications</li> <li>• Installation and wiring</li> <li>• Maintenance and inspection</li> </ul>     |
| NX-series<br>NX1P2 CPU Unit<br>Hardware<br>User's Manual | W578     | NX1P2-□□□□   | Learning the basic specifications of the NX1P2 CPU Units, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.     | An introduction to the entire NX1P2 system is provided along with the following information on the CPU Unit. <ul style="list-style-type: none"> <li>• Features and system configuration</li> <li>• Introduction</li> <li>• Part names and functions</li> <li>• General specifications</li> <li>• Installation and wiring</li> <li>• Maintenance and inspection</li> </ul>     |
| NJ-series CPU Unit<br>Hardware User's Manual             | W500     | NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□   | Learning the basic specifications of the NJ-series CPU Units, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided. | An introduction to the entire NJ-series system is provided along with the following information on the CPU Unit. <ul style="list-style-type: none"> <li>• Features and system configuration</li> <li>• Introduction</li> <li>• Part names and functions</li> <li>• General specifications</li> <li>• Installation and wiring</li> <li>• Maintenance and inspection</li> </ul> |
| NJ/NX-series CPU Unit<br>Software User's Manual          | W501     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning how to program and set up an NJ/NX-series CPU Unit. Mainly software information is provided.  | The following information is provided on a Controller built with an NJ/NX-series CPU Unit. <ul style="list-style-type: none"> <li>• CPU Unit operation</li> <li>• CPU Unit features</li> <li>• Initial settings</li> <li>• Programming based on IEC 61131-3 language specifications</li> </ul>  |

| Manual name  | Cat. No. | Model numbers  | Application   | Description  |
|--|----------|--|---|--|
| NX-series NX1P2 CPU Unit<br>Built-in I/O and Option Board<br>User's Manual | W579     | NX1P2-□□□□   | Learning about the details of functions only for an NX-series NX1P2 CPU Unit and an introduction of functions for an NJ/NX-series CPU Unit. | Of the functions for an NX1P2 CPU Unit, the following information is provided. <ul style="list-style-type: none"> <li>• Built-in I/O</li> <li>• Serial Communications Option Boards</li> <li>• Analog I/O Option Boards</li> </ul> An introduction of following functions for an NJ/NX-series CPU Unit is also provided. <ul style="list-style-type: none"> <li>• Motion control functions</li> <li>• EtherNet/IP communications functions</li> <li>• EtherCAT communications functions</li> </ul> |
| NJ/NX-series Instructions<br>Reference Manual                              | W502     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning detailed specifications on the basic instructions of an NJ/NX-series CPU Unit.   | The instructions in the instruction set (IEC 61131-3 specifications) are described.  |
| NJ/NX-series CPU Unit<br>Motion Control User's Manual                      | W507     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning about motion control settings and programming concepts.  | The settings and operation of the CPU Unit and programming concepts for motion control are described.  |
| NJ/NX-series<br>Motion Control Instructions<br>Reference Manual            | W508     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning about the specifications of the motion control instructions.   | The motion control instructions are described.   |
| NJ/NX-series<br>CPU Unit<br>Built-in EtherCAT® Port<br>User's Manual       | W505     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Using the built-in EtherCAT port on an NJ/NX-series CPU Unit.   | Information on the built-in EtherCAT port is provided.<br>This manual provides an introduction and provides information on the configuration, features, and setup.   |
| NJ/NX-series<br>CPU Unit<br>Built-in EtherNet/IP™ Port<br>User's Manual    | W506     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Using the built-in EtherNet/IP port on an NJ/NX-series CPU Unit.  | Information on the built-in EtherNet/IP port is provided.<br>Information is provided on the basic setup, tag data links, and other features.   |
| NJ/NX-series<br>CPU Unit<br>OPC UA<br>User's Manual                        | W588     | NX102-□□□□<br>NJ501-1□00   | Using the OPC UA.   | Describes the OPC UA.  |
| NX-series<br>CPU Unit<br>FINS Function<br>User's Manual                    | W596     | NX701-□□20<br>NX102-□□□□   | Using the FINS function of an NX-series CPU Unit.   | Describes the FINS function of an NX-series CPU Unit.  |
| NJ/NX-series<br>Database Connection CPU<br>Units<br>User's Manual          | W527     | NX701-□□20<br>NX102-□□20<br>NJ501-□□20<br>NJ101-□□20                             | Using the database connection service with NJ/NX-series Controllers.  | Describes the database connection service.   |
| NJ-series<br>SECS/GEM CPU Units<br>User's Manual                           | W528     | NJ501-1340   | Using the GEM Services with NJ-series Controllers.  | Provides information on the GEM Services.  |
| NJ-series<br>Robot Integrated CPU Unit<br>User's Manual                    | O037     | NJ501-R□□□   | Using the NJ-series Robot Integrated CPU Unit.  | Describes the settings and operation of the CPU Unit and programming concepts for OMRON robot control.   |

| Manual name  | Cat. No. | Model numbers  | Application  | Description   |
|--|----------|--|--|---|
| Sysmac Studio<br>Robot Integrated System<br>Building Function with Robot<br>Integrated CPU Unit Opera-<br>tion Manual  | W595     | SYSMAC-SE2□□<br>□<br>SYSMAC-<br>SE200D-64  | Learning about the<br>operating procedures<br>and functions of the<br>Sysmac Studio to<br>configure Robot Inte-<br>grated System using<br>Robot Integrated<br>CPU Unit.    | Describes the operating procedures of the<br>Sysmac Studio for Robot Integrated CPU<br>Unit.  |
| Sysmac Studio<br>Robot Integrated System<br>Building Function with IPC<br>Application Controller Opera-<br>tion Manual | W621     | SYSMAC-SE2□□<br>□<br>SYSMAC-<br>SE200D-64  | Learning about the<br>operating procedures<br>and functions of the<br>Sysmac Studio to<br>configure Robot Inte-<br>grated System using<br>IPC Application Con-<br>troller. | Describes the operating procedures of the<br>Sysmac Studio for IPC Application Control-<br>ler.   |
| Sysmac Studio<br>3D Simulation Function Opera-<br>tion Manual  | W618     | SYSMAC-SE2□□<br>□<br>SYSMAC-SA4□□<br>□-64  | Learning about an<br>outline of the 3D simu-<br>lation function of the<br>Sysmac Studio and<br>how to use the func-<br>tion.   | Describes an outline, execution procedures,<br>and operating procedures for the 3D simu-<br>lation function of the Sysmac Studio.   |
| NJ-series<br>NJ Robotics CPU Unit<br>User's Manual   | W539     | NJ501-4□□□<br>NJ501-R□□□   | Controlling robots<br>with NJ-series CPU<br>Units.   | Describes the functionality to control robots.  |
| NJ/NY-series<br>NC Integrated Controller<br>User's Manual  | O030     | NJ501-5300<br>NY532-5400   | Performing numerical<br>control with NJ/NY-<br>series Controllers.   | Describes the functionality to perform the<br>numerical control.  |
| NJ/NY-series<br>G code<br>Instructions Reference Man-<br>ual   | O031     | NJ501-5300<br>NY532-5400   | Learning about the<br>specifications of the<br>G code/M code in-<br>structions.  | The G code/M code instructions are descri-<br>bed.  |
| NJ/NX-series<br>Troubleshooting Manual   | W503     | NX701-□□□□<br>NX102-□□□□<br>NX1P2-□□□□<br>NJ501-□□□□<br>NJ301-□□□□<br>NJ101-□□□□ | Learning about the<br>errors that may be<br>detected in an<br>NJ/NX-series Con-<br>troller.  | Concepts on managing errors that may be<br>detected in an NJ/NX-series Controller and<br>information on individual errors are descri-<br>bed.   |
| Sysmac Studio Version 1<br>Operation Manual  | W504     | SYSMAC<br>-SE2□□□  | Learning about the<br>operating procedures<br>and functions of the<br>Sysmac Studio.   | Describes the operating procedures of the<br>Sysmac Studio.   |
| CNC Operator<br>Operation Manual   | O032     | SYSMAC-RTNC0□<br>□□D   | Learning an introduc-<br>tion of the CNC Op-<br>erator and how to<br>use it.   | An introduction of the CNC Operator, instal-<br>lation procedures, basic operations, con-<br>nection operations, and operating proce-<br>dures for main functions are described.  |
| NX-series<br>EtherCAT® Coupler Unit<br>User's Manual   | W519     | NX-ECC□□□  | Learning how to use<br>the NX-series Ether-<br>CAT Coupler Unit<br>and EtherCAT Slave<br>Terminals.  | The following items are described: the over-<br>all system and configuration methods of an<br>EtherCAT Slave Terminal (which consists of<br>an NX-series EtherCAT Coupler Unit and<br>NX Units), and information on hardware,<br>setup, and functions to set up, control, and<br>monitor NX Units through EtherCAT. |
| NX-series<br>Data Reference Manual   | W525     | NX-□□□□□□  | Referencing lists of<br>the data that is re-<br>quired to configure<br>systems with NX-ser-<br>ies Units.  | Lists of the power consumptions, weights,<br>and other NX Unit data that is required to<br>configure systems with NX-series Units are<br>provided.  |

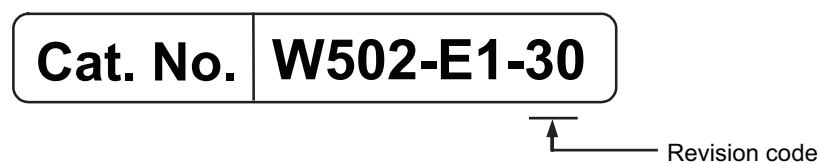
| Manual name   | Cat. No. | Model numbers   | Application   | Description   |
|---|----------|---|---|---|
| NX-series<br>NX Units<br>User's Manual  | W521     | NX-ID□□□□<br>NX-IA□□□□<br>NX-OC□□□□<br>NX-OD□□□□<br>NX-MD□□□□   | Learning how to use<br>NX Units.  | Describes the hardware, setup methods,<br>and functions of the NX Units.<br>Manuals are available for the following<br>Units.<br>Digital I/O Units, Analog I/O Units, System<br>Units, Position Interface Units, Communica-<br>tions Interface Units, Load Cell Input Unit,<br>and IO-Link Master Units.  |
|   | W522     | NX-AD□□□□<br>NX-DA□□□□  |   |   |
|   | W592     | NX-HAD□□□   |   |   |
|   | W566     | NX-TS□□□□<br>NX-HB□□□□  |   |   |
|   | W523     | NX-PD1□□□<br>NX-PF0□□□<br>NX-PC0□□□<br>NX-TBX01   |   |   |
|   | W524     | NX-EC0□□□<br>NX-ECS□□□<br>NX-PG0□□□   |   |   |
|   | W540     | NX-CIF□□□   |   |   |
|   | W565     | NX-RS□□□□   |   |   |
|   | W567     | NX-ILM□□□   |   |   |
| CJ-series<br>Special Unit Manuals<br>For NJ-series CPU Unit   | W490     | CJ1W-AD□□□<br>CJ1W-DA□□□<br>CJ1W-MAD42  | Learning how to use<br>CJ-series Units with<br>an NJ-series CPU<br>Unit.  | The methods and precautions for using CJ-<br>series Units with an NJ-series CPU Unit are<br>described, including access methods and<br>programming interfaces.<br>Manuals are available for the following<br>Units.<br>Analog I/O Units, Insulated-type Analog I/O<br>Units, Temperature Control Units, ID Sensor<br>Units, High-speed Counter Units, Serial<br>Communications Units, DeviceNet Units,<br>EtherNet/IP Units and CompoNet Master<br>Units. |
|   | W491     | CJ1W-TC□□□  |   |   |
|   | W492     | CJ1W-CT021  |   |   |
|   | W498     | CJ1W-PDC15<br>CJ1W-PH41U<br>CJ1W-AD04U  |   |   |
|   | W493     | CJ1W-CRM21  |   |   |
|   | W494     | CJ1W-SCU□□  |   |   |
|   | W495     | CJ1W-EIP21  |   |   |
|   | W497     | CJ1W-DRM21  |   |   |
|   | Z317     | CJ1W-V680□□□  |   |   |
| CX-Protocol<br>Operation Manual   | W344     | ---   | Creating data trans-<br>fer protocols for gen-<br>eral-purpose devices<br>connected to CJ-ser-<br>ies Serial Communi-<br>cations Units. | Describes operating procedures for the CX-<br>Protocol.   |
| GX-series EtherCAT Slave<br>Units<br>User's Manual  | W488     | GX-ID□□□□<br>GX-OD□□□□<br>GX-OC□□□□<br>GX-MD□□□□<br>GX-AD□□□□<br>GX-DA□□□□<br>GX-EC□□□□<br>XWT-ID□□<br>XWT-OD□□ | Learning how to use<br>the EtherCAT remote<br>I/O terminals.  | Describes the hardware, setup methods<br>and functions of the EtherCAT remote I/O<br>terminals.   |
| AC Servomotors/Servo Drives<br>1S-series with<br>Built-in EtherCAT® Communi-<br>cations User's Manual | I586     | R88M-1□<br>R88D-1SN□-ECT  | Learning how to use<br>the Servomotors/<br>Servo Drives with<br>built-in EtherCAT<br>Communications.                                    | Describes the hardware, setup methods<br>and functions of the Servomotors/Servo<br>Drives with built-in EtherCAT Communica-<br>tions.   |
|   | I621     | R88M-1AL□/-1AM<br>□<br>R88D-1SAN□-ECT   |   |   |

| Manual name  | Cat. No. | Model numbers               | Application  | Description   |
|--|----------|-----------------------------|--|---|
| AC Servomotors/Servo Drives G5 Series with Built-in EtherCAT® Communications User's Manual | I576     | R88M-K□<br>R88D-KN□-ECT     | Learning how to use the AC Servomotors/Servo Drives with built-in EtherCAT Communications. | Describes the hardware, setup methods and functions of the AC Servomotors/Servo Drives with built-in EtherCAT Communications.<br>The Linear Motor Type models and dedicated models for position control are available in G5-series. |
|  | I577     | R88L-EC-□<br>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            | July 2011      | Original production   |
| 02            | September 2011 | Corrected mistakes.   |
| 03            | March 2012     | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.01 of the CPU Unit and version 1.02 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 04            | May 2012       | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.02 of the CPU Unit and version 1.03 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 05            | August 2012    | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.03 of the CPU Unit and version 1.04 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 06            | February 2013  | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.04 of the CPU Unit and version 1.05 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 07            | April 2013     | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.05 of the CPU Unit and version 1.06 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 08            | June 2013      | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.06 of the CPU Unit and version 1.07 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 09            | September 2013 | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.07 of the CPU Unit and version 1.08 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 10            | December 2013  | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.08 of the CPU Unit and version 1.09 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 11            | July 2014      | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.09 of the CPU Unit and version 1.10 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 12            | January 2015   | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.10 of the CPU Unit and version 1.12 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 13            | April 2015     | <ul style="list-style-type: none"> <li>• Made changes accompanying the addition of NX-series NX701-□□□□ CPU Units and the release of version 1.13 of the Sysmac Studio.</li> <li>• Added information on the NJ101-□□□□ NJ-series CPU Units.</li> <li>• Corrected mistakes.</li> </ul> |
| 14            | October 2015   | <ul style="list-style-type: none"> <li>• Made changes accompanying the addition of hardware revision.</li> <li>• Corrected mistakes.</li> </ul>   |

| Revision code | Date          | Revised content   |
|---------------|---------------|---|
| 15            | April 2016    | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.11 of the CPU Unit and version 1.15 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 16            | July 2016     | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.12 of the CPU Unit and version 1.16 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 17            | October 2016  | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.13 of the CPU Unit and version 1.17 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 18            | November 2016 | Corrected mistakes.   |
| 19            | January 2017  | Corrected mistakes.   |
| 20            | April 2017    | Corrected mistakes.   |
| 21            | October 2017  | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.16 of the CPU Unit.</li> <li>• Corrected mistakes.</li> </ul>  |
| 22            | April 2018    | <ul style="list-style-type: none"> <li>• Made changes accompanying release of unit version 1.18 of the CPU Unit and version 1.22 of the Sysmac Studio.</li> <li>• Corrected mistakes.</li> </ul>  |
| 23            | April 2018    | <ul style="list-style-type: none"> <li>• Made changes accompanying the addition of NX-series NX102 CPU Units.</li> <li>• Made changes accompanying the transfer of event codes to the <i>NJ/NX-series Troubleshooting Manual (Cat. No. W503)</i>.</li> <li>• Corrected mistakes.</li> </ul>   |
| 24            | July 2018     | Corrected mistakes.   |
| 25            | January 2019  | Corrected mistakes.   |
| 26            | April 2019    | <ul style="list-style-type: none"> <li>• Added information on the functions supported by unit version 1.32 or later of NX102 CPU Units.</li> <li>• Added information on the functions supported by unit version 1.21 or later of NX1P2 CPU Units.</li> <li>• Added information on the functions supported by unit version 1.21 or later of NJ501-1□00, NJ301-□□□□, and NJ101-□□00 CPU Units.</li> </ul> |
| 27            | July 2019     | <ul style="list-style-type: none"> <li>• Made changes accompanying the upgrade of NX102-□□00, NX1P2-□□□□□□, NJ501-1□00, NJ301-□□□□, and NJ101-□□00 CPU Units to unit version 1.40.</li> <li>• Made changes accompanying the upgrade of NX701-□□□□, NJ501-4□00, NJ501-4□10, NJ501-1340 and NJ501-5300 CPU Units to unit version of 1.21.</li> </ul>  |
| 28            | July 2019     | Corrected mistakes.   |
| 29            | October 2019  | Corrected mistakes.   |
| 30            | August 2020   | <ul style="list-style-type: none"> <li>• Made changes accompanying the addition of NJ501-R□□□ CPU Units.</li> <li>• Corrected mistakes.</li> </ul>  |



# Instruction Set

This section provides a table of the instructions that you can use with NJ/NX-series CPU Unit.

---

**Instruction Set ..... 1-2**

# Instruction Set

- Refer to the *NJ/NX-series Motion Control Instructions Reference Manual (Cat. No. W508)* 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.

## Ladder Diagram Instructions

| Instruction | Name       | Function  | Page      |
|-------------|------------|---|-----------|
| LD          | Load       | Reads the value of a BOOL variable.   | page 2-14 |
| LDN         | Load NOT   | Reads the inverted value of a BOOL variable.  | page 2-14 |
| AND         | AND        | Takes the logical AND of the value of a BOOL variable and the input value.                                      | page 2-17 |
| ANDN        | AND NOT    | Takes the logical AND of the inverted value of a BOOL variable and the input value.                             | page 2-17 |
| OR          | OR         | Takes the logical OR of the value of a BOOL variable and the execution condition.                               | page 2-20 |
| ORN         | OR NOT     | Takes the logical OR of the inverted value of a BOOL variable and the execution condition.                      | page 2-20 |
| Out         | Output     | Takes the logical result from the previous instruction and outputs it to a BOOL variable.                       | page 2-23 |
| OutNot      | Output NOT | Takes the inverted value of the logical result from the previous instruction and outputs it to a BOOL variable. | page 2-23 |

## ST Statement Instructions

| Instruction | Name         | Function   | Page      |
|-------------|--------------|--|-----------|
| IF          | If           | Selects one of two statements to execute, based on the evaluation result of a specified condition expression.  | page 2-28 |
| CASE        | Case         | Selects a statement to execute, based on the value of a specified integer expression.                          | page 2-32 |
| WHILE       | While        | Repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE.  | page 2-36 |
| REPEAT      | Repeat       | Executes a statement once and then executes it repeatedly until a specified condition expression becomes TRUE. | page 2-39 |
| EXIT        | Break Loop   | Ends repeat processing for the FOR, WHILE, or REPEAT instruction of the innermost loop.                        | page 2-42 |
| RETURN      | Return       | Ends a function or function block and returns processing to the calling instruction.                           | page 2-45 |
| FOR         | Repeat Start | Specifies the condition for repeat processing, and repeatedly executes statements between FOR and END_FOR.     | page 2-46 |

## Sequence Input Instructions

| Instruction | Name       | Function   | Page      |
|-------------|------------|--|-----------|
| R_TRIG (Up) | Up Trigger | Outputs TRUE for one task period only when the input signal changes to TRUE. | page 2-48 |

| Instruction   | Name           | Function  | Page      |
|---------------|----------------|---|-----------|
| F_TRIG (Down) | Down Trigger   | Outputs TRUE for one task period only when the input signal changes to FALSE. | page 2-48 |
| TestABit      | Test A Bit     | Outputs the value of the specified bit in a bit string.                       | page 2-52 |
| TestABitN     | Test A Bit NOT | Outputs the inverted value of the specified bit in a bit string.              | page 2-52 |

## Sequence Output Instructions

| Instruction | Name                | Function  | Page      |
|-------------|---------------------|---|-----------|
| RS          | Reset-Priority Keep | Retains the value of a BOOL variable.<br>It gives priority to the Reset input if both the Set input and Reset input are TRUE. | page 2-56 |
| SR          | Set-Priority Keep   | Retains the value of a BOOL variable.<br>It gives priority to the Set input if both the Set input and Reset input are TRUE.   | page 2-59 |
| Set         | Set                 | Changes a BOOL variable to TRUE.  | page 2-62 |
| Reset       | Reset               | Changes a BOOL variable to FALSE.   | page 2-62 |
| SetBits     | Set Bits            | Changes consecutive bits in bit string data to TRUE.  | page 2-66 |
| ResetBits   | Reset Bits          | Changes consecutive bits in bit string data to FALSE.   | page 2-66 |
| SetABit     | Set A Bit           | Changes the specified bit in bit string data to TRUE.   | page 2-69 |
| ResetABit   | Reset A Bit         | Changes the specified bit in bit string data to FALSE.  | page 2-69 |
| OutABit     | Output A Bit        | Changes the specified bit in bit string data to TRUE or FALSE.  | page 2-71 |

## Sequence Control Instructions

| Instruction | Name                 | Function  | Page      |
|-------------|----------------------|---|-----------|
| End         | End                  | Ends execution of a program in the current task period.                                   | page 2-74 |
| RETURN      | Return               | Ends a function or function block and returns processing to the calling instruction.      | page 2-75 |
| MC          | Master Control Start | Marks the starting point of a master control region and resets the master control region. | page 2-76 |
| MCR         | Master Control End   | Marks the end point of a master control region.   | page 2-76 |
| JMP         | Jump                 | Moves processing to the specified jump destination.                                       | page 2-89 |
| FOR         | Repeat Start         | Marks the starting position for repeat processing and specifies the repeat condition.     | page 2-91 |
| NEXT        | Repeat End           | Marks the ending position for repeat processing.  | page 2-91 |
| BREAK       | Break Loop           | Cancels repeat processing from the innermost FOR instruction to the NEXT instruction.     | page 2-98 |

## Comparison Instructions

| Instruction | Name               | Function  | Page       |
|-------------|--------------------|---|------------|
| EQ (=)      | Equal              | Determines if the values of two or more variables are all equivalent. | page 2-102 |
| NE (<>)     | Not Equal          | Determines if the values of two variables are not equivalent.         | page 2-105 |
| LT (<)      | Less Than          | Performs a less than comparison between values.                       | page 2-108 |
| LE (<=)     | Less Than Or Equal | Performs a less than or equal comparison between values.              | page 2-108 |
| GT (>)      | Greater Than       | Performs a greater than comparison between values.                    | page 2-108 |

| Instruction | Name   | Function   | Page       |
|-------------|--|--|------------|
| GE (>=)     | Greater Than Or Equal                        | Performs a greater than or equal comparison between values.  | page 2-108 |
| EQascii     | Text String Comparison Equal                 | Determines if two or more text strings are all equivalent.   | page 2-111 |
| NEascii     | Text String Comparison Not Equal             | Determines if two text strings are not equivalent.   | page 2-113 |
| LTascii     | Text String Comparison Less Than             | Performs a less than comparison between text strings.  | page 2-115 |
| LEascii     | Text String Comparison Less Than or Equal    | Performs a less than or equal comparison between text strings.                                       | page 2-115 |
| GTascii     | Text String Comparison Greater Than          | Performs a greater than comparison between text strings.   | page 2-115 |
| GEascii     | Text String Comparison Greater Than or Equal | Performs a greater than or equal comparison between text strings.                                    | page 2-115 |
| Cmp         | Compare                                      | Compares two values.   | page 2-118 |
| ZoneCmp     | Zone Comparison                              | Determines if the comparison data is between the specified maximum and minimum values.               | page 2-120 |
| TableCmp    | Table Comparison                             | Compares the comparison data with multiple defined ranges in a comparison table.                     | page 2-122 |
| AryCmpEQ    | Array Comparison Equal                       | Determines if the corresponding elements of two arrays are equal.                                    | page 2-125 |
| AryCmpNE    | Array Comparison Not Equal                   | Determines if the corresponding elements of two arrays are not equal.                                | page 2-125 |
| AryCmpLT    | Array Comparison Less Than                   | Performs a less than comparison between the corresponding elements of two arrays.                    | page 2-127 |
| AryCmpLE    | Array Comparison Less Than Or Equal          | Performs a less than or equal comparison between the corresponding elements of two arrays.           | page 2-127 |
| AryCmpGT    | Array Comparison Greater Than                | Performs a greater than comparison between the corresponding elements of two arrays.                 | page 2-127 |
| AryCmpGE    | Array Comparison Greater Than Or Equal       | Performs a greater than or equal comparison between the corresponding elements of two arrays.        | page 2-127 |
| AryCmpEQV   | Array Value Comparison Equal                 | Determines if each element of an array is equal to a comparison value.                               | page 2-130 |
| AryCmpNEV   | Array Value Comparison Not Equal             | Determines if each element of an array is not equal to a comparison value.                           | page 2-130 |
| AryCmpLTV   | Array Value Comparison Less Than             | Performs a less than comparison between each element of an array and a comparison value.             | page 2-132 |
| AryCmpLEV   | Array Value Comparison Less Than Or Equal    | Performs a less than or equal comparison between each element of an array and a comparison value.    | page 2-132 |
| AryCmpGTV   | Array Value Comparison Greater Than          | Performs a greater than comparison between each element of an array and a comparison value.          | page 2-132 |
| AryCmpGEV   | Array Value Comparison Greater Than Or Equal | Performs a greater than or equal comparison between each element of an array and a comparison value. | page 2-132 |

## Timer Instructions

| Instruction       | Name               | Function  | Page       |
|-------------------|--------------------|---|------------|
| TON               | On-Delay Timer     | Outputs TRUE when the set time elapses after the timer starts.  | page 2-136 |
| TOF               | Off-Delay Timer    | Outputs FALSE when the set time elapses after the timer starts.   | page 2-142 |
| TP                | Timer Pulse        | Outputs TRUE for a set period of time after the timer starts.   | page 2-145 |
| AccumulationTimer | Accumulation Timer | Accumulates the period of time during which the timer input is TRUE.                                    | page 2-148 |
| Timer             | Hundred-ms Timer   | Outputs TRUE when the set time elapses after the timer starts. The time is set in increments of 100 ms. | page 2-152 |

## Counter Instructions

| Instruction | Name                  | Function   | Page       |
|-------------|-----------------------|--|------------|
| 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.   | page 2-156 |
| 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.                             | page 2-158 |
| 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.   | page 2-161 |
| 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.                             | page 2-164 |
| 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.   | page 2-167 |
| 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. | page 2-172 |

## Math Instructions

| Instruction | Name                               | Function   | Page       |
|-------------|------------------------------------|--|------------|
| ADD (+)     | Addition                           | Adds integers and real numbers. Also joins text strings.   | page 2-179 |
| AddOU (+OU) | Addition with Overflow Check       | Adds integers and real numbers. Also performs an overflow check for the integer addition result.                                     | page 2-183 |
| SUB (-)     | Subtraction                        | Subtracts integers and real numbers.   | page 2-187 |
| SubOU (-OU) | Subtraction with Overflow Check    | Subtracts integers and real numbers. Also performs an overflow check for the integer subtraction result.                             | page 2-190 |
| MUL (*)     | Multiplication                     | Multiplies integers and real numbers.  | page 2-194 |
| 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. | page 2-198 |

| Instruction | Name                                       | Function  | Page       |
|-------------|--|---|------------|
| DIV (/)     | Division                                   | Divides integers or real numbers.                                 | page 2-202 |
| MOD         | Modulo-division                            | Finds the remainder for division of integers.                     | page 2-205 |
| ABS         | Absolute value                             | Finds the absolute value of an integer or real number.            | page 2-207 |
| RadToDeg    | Radians to Degrees                         | Converts a real number from radians (rad) to degrees (°).         | page 2-209 |
| DegToRad    | Degrees to Radians                         | Converts a real number from degrees (°) to radians (rad).         | page 2-209 |
| SIN         | Sine in Radians                            | Calculates the sine of a real number.                             | page 2-211 |
| COS         | Cosine in Radians                          | Calculates the cosine of a real number.                           | page 2-211 |
| TAN         | Tangent in Radians                         | Calculates the tangent of a real number.                          | page 2-211 |
| ASIN        | Principal Arc Sine (SIN <sup>-1</sup> )    | Calculates the arcsine of a real number (sin <sup>-1</sup> ).     | page 2-214 |
| ACOS        | Principal Arc Cosine (COS <sup>-1</sup> )  | Calculates the arccosine of a real number (cos <sup>-1</sup> ).   | page 2-214 |
| ATAN        | Principal Arc Tangent (TAN <sup>-1</sup> ) | Calculates the arctangent of a real number (tan <sup>-1</sup> ).  | page 2-214 |
| SQRT        | Square Root                                | Calculates the square root of a real number.                      | page 2-217 |
| LN          | Natural Logarithm                          | Calculates the natural logarithm of a real number.                | page 2-220 |
| LOG         | Logarithm Base 10                          | Calculates the base-10 logarithm of a real number.                | page 2-220 |
| EXP         | Natural Exponential Operation              | Performs calculations for the natural exponential function.       | page 2-224 |
| EXPT (**)   | Exponentiation                             | Raises one real number to the power of another real number.       | page 2-226 |
| Inc         | Increment                                  | Increments an integer value.                                      | page 2-232 |
| Dec         | Decrement                                  | Decrements an integer value.                                      | page 2-232 |
| Rand        | Random Number                              | Generates pseudorandom numbers.                                   | page 2-234 |
| AryAdd      | Array Addition                             | Adds corresponding elements of two arrays.                        | page 2-236 |
| AryAddV     | Array Value Addition                       | Adds the same value to specified elements of an array.            | page 2-238 |
| ArySub      | Array Subtraction                          | Subtracts corresponding elements of two arrays.                   | page 2-240 |
| ArySubV     | Array Value Subtraction                    | Subtracts the same value from specified elements of an array.     | page 2-242 |
| AryMean     | Array Mean                                 | Calculates the average of the elements of an array.               | page 2-244 |
| ArySD       | Array Element Standard Deviation           | Calculates standard deviation of the elements of an array.        | page 2-246 |
| ModReal     | Real Number Modulo-division                | Calculates the remainder of real number division.                 | page 2-248 |
| Fraction    | Real Number Fraction                       | Finds the fractional part of a real number.                       | page 2-250 |
| CheckReal   | Real Number Check                          | Checks a real number to see if it is infinity or nonnumeric data. | page 2-252 |

## BCD Conversion Instructions

| Instruction  | Name                                     | Function   | Page       |
|--------------|--|--|------------|
| **_BCD_TO_** | BCD-to-Uncoded Integer Conversion Group  | Converts BCD bit strings into unsigned integers. | page 2-256 |
| **_TO_BCD_** | Unsigned Integer-to-BCD Conversion Group | Converts unsigned integers to BCD bit strings.   | page 2-259 |



| Instruction  | Name   | Function   | Page       |
|--------------|--|--|------------|
| BCD_TO_**    | BCD Data Type-to-Unsigned Integer Conversion Group | Converts BCD bit strings into unsigned integers.                             | page 2-262 |
| BCDsToBin    | Signed BCD-to-Signed Integer Conversion            | Converts signed BCD bit strings to signed integers.                          | page 2-265 |
| BinToBCDs_** | Signed Integer-to-BCD Conversion Group             | Converts signed integers to signed BCD bit strings.                          | page 2-268 |
| AryToBCD     | Array BCD Conversion                               | Converts the elements of an unsigned integer array to BCD bit strings.       | page 2-271 |
| AryToBin     | Array Unsigned Integer Conversion                  | Converts the elements of an array of BCD bit strings into unsigned integers. | page 2-273 |

## Data Type Conversion Instructions

| Instruction   | Name  | Function   | Page       |
|---|---|--|------------|
| **_TO_*** (Integer-to-Integer Conversion Group)         | Integer-to-Integer Conversion Group         | Converts integers to integers with different data types.         | page 2-277 |
| **_TO_*** (Integer-to-Bit String Conversion Group)      | Integer-to-Bit String Conversion Group      | Converts integers to bit strings.                                | page 2-280 |
| **_TO_*** (Integer-to-Real Number Conversion Group)     | Integer-to-Real Number Conversion Group     | Converts integers to real numbers.                               | page 2-283 |
| **_TO_*** (Bit String-to-Integer Conversion Group)      | Bit String-to-Integer Conversion Group      | Converts bit strings to integers.                                | page 2-285 |
| **_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.   | page 2-288 |
| **_TO_*** (Bit String-to-Real Number Conversion Group)  | Bit String-to-Real Number Conversion Group  | Converts bit strings to real numbers.                            | page 2-290 |
| **_TO_*** (Real Number-to-Integer Conversion Group)     | Real Number-to-Integer Conversion Group     | Converts real numbers to integers.                               | page 2-292 |
| **_TO_*** (Real Number-to-Bit String Conversion Group)  | Real Number-to-Bit String Conversion Group  | Converts real numbers to bit strings.                            | page 2-295 |
| **_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. | page 2-297 |
| **_TO_STRING (Integer-to-Text String Conversion Group)  | Integer-to-Text String Conversion Group     | Converts integers to text strings.                               | page 2-299 |

| Instruction  | Name  | Function  | Page       |
|--|---|---|------------|
| **_TO_STRING (Bit String-to-Text String Conversion Group)  | Bit String-to-Text String Conversion Group  | Converts bit strings to text strings.   | page 2-301 |
| **_TO_STRING (Real Number-to-Text String Conversion Group) | Real Number-to-Text String Conversion Group | Converts real numbers to text strings.  | page 2-303 |
| RealToFormatString   | REAL-to-Formatted Text String               | Converts a REAL variable to a text string with the specified format.                          | page 2-305 |
| LrealToFormatString  | LREAL-to-Formatted Text String              | Converts a LREAL variable to a text string with the specified format.                         | page 2-311 |
| STRING_TO_** (Text String-to-Integer Conversion Group)     | Text String-to-Integer Conversion Group     | Converts text strings to integers.  | page 2-317 |
| STRING_TO_** (Text String-to-Bit String Conversion Group)  | Text String-to-Bit String Conversion Group  | Converts text strings to bit strings.   | page 2-319 |
| STRING_TO_** (Text String-to-Real Number Conversion Group) | Text String-to-Real Number Conversion Group | Converts text strings to real numbers.  | page 2-321 |
| TO_** (Integer Conversion Group)                           | Integer Conversion Group                    | Converts integers, bit strings, real numbers, and text strings to integers.                   | page 2-325 |
| TO_** (Bit String Conversion Group)                        | Bit String Conversion Group                 | Converts integers, bit strings, real numbers, and text strings to bit strings.                | page 2-327 |
| TO_** (Real Number Conversion Group)                       | Real Number Conversion Group                | Converts integers, bit strings, real numbers, and text strings to real numbers.               | page 2-329 |
| EnumToNum  | Enumeration-to-Integer                      | Converts enumeration data to DINT data.   | page 2-331 |
| NumToEnum  | Integer-to-Enumeration                      | Converts DINT data to enumeration data.   | page 2-333 |
| TRUNC  | Truncate                                    | Truncates a real number to an integer.  | page 2-336 |
| Round  | Round Off Real Number                       | Rounds up or down a real number to the nearest integer, depending on the first decimal digit. | page 2-336 |
| RoundUp  | Round Up Real Number                        | Rounds up a real number to the nearest integer.   | page 2-336 |

## Bit String Processing Instructions

| Instruction | Name                  | Function   | Page       |
|-------------|-----------------------|--|------------|
| AND (&)     | Logical AND           | Performs a logical AND operation on each corresponding bit of multiple Boolean variables or bit strings.           | page 2-340 |
| OR          | Logical OR            | Performs a logical OR operation on each corresponding bit of multiple Boolean variables or bit strings.            | page 2-340 |
| XOR         | Logical Exclusive OR  | Performs a logical exclusive OR operation on each corresponding bit of multiple Boolean variables or bit strings.  | page 2-340 |
| XORN        | Logical Exclusive NOR | Performs a logical exclusive NOR operation on each corresponding bit of multiple Boolean variables or bit strings. | page 2-343 |
| NOT         | Bit Reversal          | Inverts each bit of a Boolean variable or bit string.  | page 2-345 |

| Instruction | Name                        | Function   | Page       |
|-------------|-----------------------------|--|------------|
| AryAnd      | Array Logical AND           | Performs a logical AND operation on individual bits of each corresponding Boolean or bit-string element in two arrays.           | page 2-347 |
| AryOr       | Array Logical OR            | Performs a logical OR operation on individual bits of each corresponding Boolean or bit-string element in two arrays.            | page 2-347 |
| AryXor      | Array Logical Exclusive OR  | Performs a logical exclusive OR operation on individual bits of each corresponding Boolean or bit-string element in two arrays.  | page 2-347 |
| AryXorN     | Array Logical Exclusive NOR | Performs a logical exclusive NOR operation on individual bits of each corresponding Boolean or bit-string element in two arrays. | page 2-347 |

## Selection Instructions

| Instruction | Name              | Function  | Page       |
|-------------|-------------------|---|------------|
| SEL         | Binary Selection  | Selects one of two options.   | page 2-352 |
| MUX         | Multiplexer       | Selects one of two to five options.   | page 2-354 |
| LIMIT       | Limiter           | Limits the value of an input variable between the specified minimum and maximum values. | page 2-357 |
| Band        | Deadband Control  | Performs deadband control.  | page 2-359 |
| Zone        | Dead Zone Control | Adds a bias value to the input value.   | page 2-361 |
| MAX         | Maximum           | Finds the largest of two to five values.  | page 2-363 |
| MIN         | Minimum           | Finds the smallest of two to five values.   | page 2-363 |
| AryMax      | Array Maximum     | Finds elements with the largest value in a one-dimensional array.                       | page 2-365 |
| AryMin      | Array Minimum     | Finds elements with the smallest value in a one-dimensional array.                      | page 2-365 |
| ArySearch   | Array Search      | Searches for the specified value in a one-dimensional array.                            | page 2-368 |

## Data Movement Instructions

| Instruction                                | Name  | Function   | Page       |
|--|---|--|------------|
| MOVE                                       | Move  | Moves the value of a constant or variable to another variable.   | page 2-372 |
| MoveBit                                    | Move Bit  | Moves one bit in a bit string.   | page 2-375 |
| MoveDigit                                  | Move Digit  | Moves digits (4 bits per digit) in a bit string.   | page 2-377 |
| TransBits                                  | Move Bits   | Moves one or more bits in a bit string.  | page 2-379 |
| MemCopy                                    | Memory Copy   | Moves one or more array elements. The move source and move destination must have the same data type.       | page 2-381 |
| SetBlock                                   | Block Set   | Moves the value of a variable or constant to one or more array elements.                                   | page 2-383 |
| Exchange                                   | Data Exchange   | Exchanges the values of two variables.   | page 2-385 |
| AryExchange                                | Array Data Exchange                                   | Exchanges the elements of two arrays.  | page 2-387 |
| AryMove                                    | Array Move  | Moves one or more array elements. The data types of the move source and move destination can be different. | page 2-389 |
| Clear                                      | Initialize  | Initializes a variable.  | page 2-391 |
| 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.   | page 2-393 |

| Instruction                                 | Name   | Function  | Page       |
|---|--|---|------------|
| 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.     | page 2-395 |
| 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.  | page 2-397 |
| CopyNumTo** (Signed Integer to Real Number) | Bit Pattern Copy (Signed Integer to Real Number) Group | Copies the content of a signed integer directly to a real number. | page 2-399 |
| 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.     | page 2-401 |
| 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. | page 2-403 |

## Shift Instructions

| Instruction   | Name                          | Function  | Page       |
|---------------|-------------------------------|---|------------|
| AryShiftReg   | Shift Register                | Shifts an array of bit strings by one bit to the left and inserts an input value to the least-significant bit.                              | page 2-406 |
| AryShiftRegLR | Reversible Shift Register     | Shifts an array of bit strings by one bit to the left or right and inserts an input value to the least-significant or most-significant bit. | page 2-408 |
| ArySHL        | Array N-element Left Shift    | Shifts array elements by one or more elements to the left (toward the higher elements).   | page 2-411 |
| ArySHR        | Array N-element Right Shift   | Shifts array elements by one or more elements to the right (toward the lower elements).   | page 2-411 |
| SHL           | N-bit Left Shift              | Shifts a bit string by one or more bits to the left (toward the higher bits).   | page 2-414 |
| SHR           | N-bit Right Shift             | Shifts a bit string by one or more bits to the right (toward the lower bits).   | page 2-414 |
| NSHLC         | Shift N-bits Left with Carry  | Shifts an array of bit strings by one or more bits to the left (toward the higher elements), with the Carry (CY) Flag available.            | page 2-417 |
| NSHRC         | Shift N-bits Right with Carry | Shifts an array of bit strings by one or more bits to the right (toward the lower elements), with the Carry (CY) Flag available.            | page 2-417 |
| ROL           | Rotate N-bits Left            | Rotates a bit string by one or more bits to the left (toward the higher bits).  | page 2-420 |
| ROR           | Rotate N-bits Right           | Rotates a bit string by one or more bits to the right (toward the lower bits).  | page 2-420 |

## Conversion Instructions

| Instruction | Name         | Function   | Page       |
|-------------|--------------|--|------------|
| Swap        | Swap Bytes   | Swaps the upper byte and lower byte of a 16-bit value. | page 2-425 |
| Neg         | Reverse Sign | Reverses the sign of a number.                         | page 2-427 |

| Instruction        | Name   | Function   | Page       |
|--------------------|--|--|------------|
| 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.                      | page 2-429 |
| Encoder            | Bit Encoder  | Finds the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.                                      | page 2-432 |
| BitCnt             | Bit Counter  | Counts the number of TRUE bits in a bit string.  | page 2-434 |
| ColmToLine_**      | Column to Line Conversion Group                          | Extracts bit values from the specified position of array elements and outputs them as a bit string.                                      | page 2-435 |
| LineToColm         | Line to Column Conversion                                | Takes the bits from a bit string and outputs them to the specified bit position in array elements.                                       | page 2-437 |
| Gray               | Gray Code Conversion                                     | Converts a gray code into an angle.  | page 2-439 |
| UTF8ToSJIS         | UTF-8 to SJIS Character Code Conversion                  | Converts a UTF-8 text string to a SJIS BYTE array.   | page 2-444 |
| SJISToUTF8         | SJIS to UTF-8 Character Code Conversion                  | Converts a SJIS BYTE array to a UTF-8 text string.   | page 2-446 |
| PWLApprox          | Broken Line Approximation with Broken Line Data Check    | Performs broken line approximations for integers or real numbers with a check of the validity of the broken line data.                   | page 2-448 |
| PWLApproxNoLineChk | Broken Line Approximation without Broken Line Data Check | Performs broken line approximations for integers or real numbers without a check of the validity of the broken line data.                | page 2-448 |
| PWLLineChk         | Broken Line Data Check                                   | Checks whether broken line data to be used for the PWLApproxNoLineCheck instruction is sorted in ascending order of X-coordinate values. | page 2-454 |
| MovingAverage      | Moving Average   | Calculates a moving average.   | page 2-457 |
| DispartReal        | Separate Mantissa and Exponent                           | Separates a real number into the signed mantissa and the exponent.   | page 2-464 |
| UniteReal          | Combine Real Number Mantissa and Exponent                | Combines a signed mantissa and exponent to make a real number.   | page 2-467 |
| NumToDecString     | Fixed-length Decimal Text String Conversion              | Converts an integer to a fixed-length decimal text string.   | page 2-469 |
| NumToHexString     | Fixed-length Hexadecimal Text String Conversion          | Converts an integer to a fixed-length hexadecimal text string.   | page 2-469 |
| HexStringToNum_**  | Hexadecimal Text String-to-Number Conversion Group       | Converts a hexadecimal text string to an integer.  | page 2-472 |
| FixNumToString     | Fixed-decimal Number-to-Text String Conversion           | Converts a signed fixed-decimal number to a decimal text string.   | page 2-474 |
| StringToFixNum     | Text String-to-Fixed-decimal Conversion                  | Converts a decimal text string to a signed fixed-decimal number.   | page 2-476 |
| DtToString         | Date and Time-to-Text String Conversion                  | Converts a date and time to a text string.   | page 2-479 |
| DateToString       | Date-to-Text String Conversion                           | Converts a date to a text string.  | page 2-481 |

| Instruction   | Name                                      | Function  | Page       |
|---------------|---|---|------------|
| TodToString   | Time of Day-to-Text String Conversion     | Converts a time of day to a text string.                              | page 2-483 |
| GrayToBin_**  | Gray Code-to-Binary Code Conversion Group | Converts a gray code to a bit string.                                 | page 2-485 |
| BinToGray_**  | Binary Code-to-Gray Code Conversion       | Converts a bit string to a gray code.                                 | page 2-485 |
| StringToAry   | Text String-to-Array Conversion           | Converts a text string to a BYTE array.                               | page 2-488 |
| AryToString   | Array-to-Text String Conversion           | Converts a BYTE array to a text string.                               | page 2-490 |
| DispartDigit  | Four-bit Separation                       | Separates a bit string into 4-bit units.                              | page 2-492 |
| UniteDigit_** | Four-bit Join Group                       | Joins 4-bit units of data into a bit string.                          | page 2-494 |
| Dispart8Bit   | Byte Data Separation                      | Separates a bit string into individual bytes.                         | page 2-496 |
| Unite8Bit_**  | Byte Data Join Group                      | Joins bytes of data into a bit string.                                | page 2-498 |
| ToAryByte     | Conversion to Byte Array                  | Separates a variable into bytes and stores the bytes in a BYTE array. | page 2-500 |
| AryByteTo     | Conversion from Byte Array                | Joins BYTE array elements and stores in a variable.                   | page 2-506 |
| SizeOfAry     | Get Number of Array Elements              | Gets the number of elements in an array.                              | page 2-512 |
| PackWord      | 2-byte Join                               | Joins two 1-byte data into a 2-byte data.                             | page 2-514 |
| PackDword     | 4-byte Join                               | Joins four 1-byte data into a 4-byte data.                            | page 2-516 |
| LOWER_BOUND   | Get First Number of Array                 | Gets the first number of array dimensions.                            | page 2-518 |
| UPPER_BOUND   | Get Last Number of Array                  | Gets the last number of array dimensions.                             | page 2-518 |

## Stack and Table Instructions

| Instruction    | Name                  | Function  | Page       |
|----------------|-----------------------|---|------------|
| StackPush      | Push onto Stack       | Stores a value into the top of a stack.   | page 2-524 |
| StackFIFO      | First In First Out    | Removes the bottom value from a stack.  | page 2-533 |
| StackLIFO      | Last In First Out     | Removes the top value from a stack.   | page 2-533 |
| StackIns       | Insert into Stack     | Inserts a value at a specified position in a stack.   | page 2-536 |
| StackDel       | Delete from Stack     | Deletes a value from a specified position in a stack.   | page 2-539 |
| RecSearch      | Record Search         | Searches an array of structures for elements that match the search key with the specified method.             | page 2-541 |
| RecRangeSearch | Range Record Search   | Searches an array of structures for elements that match the search condition range with the specified method. | page 2-546 |
| RecSort        | Record Sort           | Sorts the elements of an array of structures.   | page 2-551 |
| RecNum         | Get Number of Records | Finds the number of records in an array of structures to the end data.  | page 2-557 |
| RecMax         | Maximum Record Search | Searches an array of structures for the maximum value of a specified member.                                  | page 2-560 |
| RecMin         | Minimum Record Search | Searches an array of structures for the minimum value of a specified member.                                  | page 2-560 |

## FCS Instructions

| Instruction   | Name                            | Function  | Page       |
|---------------|---------------------------------|---|------------|
| StringSum     | Checksum Calculation            | Calculates the checksum for a text string.              | page 2-566 |
| StringLRC     | Calculate Text String LRC       | Calculates the LRC value (horizontal parity).           | page 2-568 |
| StringCRCCITT | Calculate Text String CRC-CCITT | Calculates the CRC-CCITT value using the XMODEM method. | page 2-570 |
| StringCRC16   | Calculate Text String CRC-16    | Calculates the CRC-16 value using the MODBUS method.    | page 2-572 |
| AryLRC_**     | Calculate Array LRC Group       | Calculates the LRC value for an array.                  | page 2-574 |
| AryCRCCITT    | Calculate Array CRC-CCITT       | Calculates the CRC-CCITT value using the XMODEM method. | page 2-576 |
| AryCRC16      | Calculate Array CRC-16          | Calculates the CRC-16 value using the MODBUS method.    | page 2-578 |

## Text String Instructions

| Instruction  | Name                              | Function   | Page       |
|--------------|-----------------------------------|--|------------|
| CONCAT       | Concatenate String                | Joins two to five text strings.  | page 2-582 |
| LEFT         | Get String Left                   | Extracts a substring with a specified number of characters from the start (left) of a text string.     | page 2-584 |
| RIGHT        | Get String Right                  | Extracts a substring with a specified number of characters from the end (right) of a text string.      | page 2-584 |
| MID          | Get String Any                    | Extracts a substring with a specified number of characters from a specified position of a text string. | page 2-587 |
| FIND         | Find String                       | Searches for the position of a specified substring in a text string.                                   | page 2-589 |
| LEN          | String Length                     | Finds the number of characters in a text string.   | page 2-591 |
| REPLACE      | Replace String                    | Replaces part of a text string with another text string.   | page 2-593 |
| DELETE       | Delete String                     | Deletes all or part of a text string.  | page 2-595 |
| INSERT       | Insert String                     | Inserts a text string into another text string.  | page 2-597 |
| GetByteLen   | Get Byte Length                   | Counts the number of bytes in a text string.   | page 2-599 |
| ClearString  | Clear String                      | Clears a text string.  | page 2-600 |
| ToUCase      | Convert to Uppercase              | Converts all single-byte letters in a text string to uppercase.  | page 2-602 |
| ToLCase      | Convert to Lowercase              | Converts all single-byte letters in a text string to lowercase.  | page 2-602 |
| TrimL        | Trim String Left                  | Removes blank space from the beginning of a text string.   | page 2-604 |
| TrimR        | Trim String Right                 | Removes blank space from the end of a text string.   | page 2-604 |
| AddDelimiter | Put Text Strings with Delimiters  | Converts the values of all the members in a structure into a text string with delimiters.              | page 2-606 |
| SubDelimiter | Get Text Strings Minus Delimiters | Reads out delimited part of a text string and stores as the value of the members of a structure.       | page 2-618 |



## Time and Time of Day Instructions

| Instruction      | Name                                   | Function  | Page       |
|------------------|--|---|------------|
| ADD_TIME         | Add Time                               | Adds two times.   | page 2-633 |
| ADD_TOD_TIME     | Add Time to Time of Day                | Adds a time to a time of day.   | page 2-635 |
| ADD_DT_TIME      | Add Time to Date and Time              | Adds a time to a date and time.   | page 2-637 |
| SUB_TIME         | Subtract Time                          | Subtracts a time from another time.   | page 2-639 |
| SUB_TOD_TIME     | Subtract Time from Time of Day         | Subtracts a time from a time of day.  | page 2-641 |
| SUB_TOD_TOD      | Subtract Time of Day                   | Subtracts a time of day from another time of day.                                   | page 2-643 |
| SUB_DATE_DATE    | Subtract Date                          | Subtracts a date from another date.   | page 2-645 |
| SUB_DT_DT        | Subtract Date and Time                 | Subtracts a date and time from another date and time.                               | page 2-646 |
| SUB_DT_TIME      | Subtract Time from Date and Time       | Subtracts a time from a date and time.  | page 2-648 |
| MULTIME          | Multiply Time                          | Multiplies a time by a specified number.  | page 2-650 |
| DIVTIME          | Divide Time                            | Divides a time by a specified number.   | page 2-652 |
| CON-CAT_DATE_TOD | Concatenate Date and Time of Day       | Combines a date and a time of day.  | page 2-654 |
| DT_TO_TOD        | Extract Time of Day from Date and Time | Extracts the time of day from a date and time.                                      | page 2-656 |
| DT_TO_DATE       | Extract Date from Date and Time        | Extracts the date from a date and time.   | page 2-658 |
| SetTime          | Set Time                               | Sets the system time.   | page 2-660 |
| GetTime          | Get Time of Day                        | Reads the current time.   | page 2-662 |
| 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. | page 2-664 |
| DateToSec        | Convert Date to Seconds                | Converts a date to the number of seconds from 00:00:00 on January 1, 1970.          | page 2-666 |
| TodToSec         | Convert Time of Day to Seconds         | Converts a time of day to the number of seconds from 00:00:00.                      | page 2-668 |
| 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. | page 2-670 |
| SecToDate        | Convert Seconds to Date                | Converts the number of seconds from 00:00:00 on January 1, 1970 to a date.          | page 2-672 |
| SecToTod         | Convert Seconds to Time of Day         | Converts the number of seconds from 00:00:00 to a time of day.                      | page 2-674 |
| TimeToNanoSec    | Convert Time to Nanoseconds            | Converts a time to nanoseconds.   | page 2-676 |
| TimeToSec        | Convert Time to Seconds                | Converts a time to seconds.   | page 2-677 |
| NanoSecToTime    | Convert Nanoseconds to Time            | Converts nanoseconds to a time.   | page 2-679 |
| SecToTime        | Convert Seconds to Time                | Converts seconds to a time.   | page 2-680 |
| ChkLeapYear      | Check for Leap Year                    | Checks if a specified year is a leap year.  | page 2-682 |
| GetDaysOfMonth   | Get Days in Month                      | Gets the number of days in a specified month.                                       | page 2-683 |



| Instruction    | Name                     | Function   | Page       |
|----------------|--------------------------|--|------------|
| DaysToMonth    | Convert Days to Month    | Calculates the month based on the number of days from January 1.                           | page 2-686 |
| GetDayOfWeek   | Get Day of Week          | Gets the day of the week for a specified date (year, month, and day).                      | page 2-688 |
| GetWeekOfYear  | Get Week Number          | Gets the week number for a specified date (year, month, and day).                          | page 2-690 |
| DtToDateStruct | Break Down Date and Time | Converts a date and time to the year, month, day, hour, minutes, seconds, and nanoseconds. | page 2-692 |
| DateStructToDt | Join Time                | Joins a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.    | page 2-695 |
| TruncTime      | Truncate Time            | Truncates a TIME variable to a specified time unit.  | page 2-698 |
| TruncDt        | Truncate Date and Time   | Truncates a DT variable to a specified time unit.  | page 2-702 |
| TruncTod       | Truncate Time of Day     | Truncates a TOD variable to a specified time unit.   | page 2-706 |

## Analog Control Instructions

| Instruction             | Name  | Function  | Page       |
|-------------------------|---|---|------------|
| PIDAT                   | PID Control with Autotuning                             | Performs PID control with autotuning (2-PID control with set point filter).   | page 2-712 |
| PIDAT_HeatCool          | Heating/Cooling PID with Autotuning                     | Performs heating/cooling PID control with autotuning (2-PID control with set point filter).   | page 2-742 |
| TimeProportionalOut     | Time-proportional output                                | Converts a manipulated variable to a time-proportional output.  | page 2-779 |
| 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.  | page 2-799 |
| 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. | page 2-804 |
| LimitAlarmDvStby-Seq_** | Upper/Lower Deviation Alarm with Standby Sequence Group | Outputs upper and lower deviation alarms with a standby sequence.   | page 2-809 |
| ScaleTrans              | Scale Transformation                                    | Converts input values from an input range to an output range.   | page 2-827 |
| AC_StepProgram          | Step Program  | Calculates the present set point and the predicted set point every task period according to the specified program pattern.                            | page 2-830 |

## System Control Instructions

| Instruction    | Name                      | Function                                    | Page       |
|----------------|---------------------------|---|------------|
| TraceSamp      | Data Trace Sampling       | Performs sampling for a data trace.         | page 2-859 |
| TraceTrig      | Data Trace Trigger        | Generates a trigger for data tracing.       | page 2-862 |
| GetTraceStatus | Read Data Trace Status    | Reads the execution status of a data trace. | page 2-865 |
| SetAlarm       | Create User-defined Error | Creates a user-defined error.               | page 2-868 |

| Instruction              | Name                             | Function  | Page       |
|--------------------------|----------------------------------|---|------------|
| ResetAlarm               | Reset User-defined Error         | Resets a user-defined error.  | page 2-873 |
| 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.                           | page 2-875 |
| ResetPLCError            | Reset PLC Controller Error       | Resets errors in the PLC Function Module.   | page 2-877 |
| 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.            | page 2-880 |
| ResetCJBError            | Reset CJ Bus Controller Error    | Resets Controller errors in the I/O bus.  | page 2-882 |
| GetCJBError              | Get I/O Bus Error Status         | Gets the highest level status and highest level event code of the current Controller errors in the I/O bus of the NJ-series CPU Unit.                             | page 2-884 |
| 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.    | page 2-886 |
| ResetMCErr               | Reset Motion Control Error       | Resets Controller errors in the Motion Control Function Module.   | page 2-888 |
| GetMCErr                 | 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. | page 2-894 |
| ResetECErr               | Reset EtherCAT Error             | Resets Controller errors in the EtherCAT Master Function Module.  | page 2-896 |
| GetECErr                 | Get EtherCAT Error Status        | Detects errors in the EtherCAT Master Function Module.  | page 2-898 |
| ResetNXBError            | Reset NX Bus Error               | Resets Controller errors in the NX Bus Function Module.   | page 2-901 |
| GetNXBError              | Get NX Bus Error Status          | Gets the highest level status of the current Controller errors in the NX Bus Function Module of the NX-series CPU Unit.   | page 2-903 |
| GetNXUnitError           | Get NX Unit Error Status         | Gets the highest level status and highest level event code of the current Controller errors in the NX Bus Function Module of the NX-series CPU Unit or NX Units.  | page 2-905 |
| SetInfo                  | Create User-defined Information  | Creates user-defined information.   | page 2-912 |
| ResetUnit                | Restart Unit                     | Restarts a CPU Bus Unit or Special I/O Unit.  | page 2-914 |
| GetNTPStatus             | Read NTP Status                  | Reads the NTP status.   | page 2-919 |
| RestartNXUnit            | Restart NX Units                 | Restarts an EtherCAT Coupler Unit or NX Units.  | page 2-921 |
| NX_ChangeWrite-Mode      | Change to NX Unit Write Mode     | Changes an EtherCAT Coupler Unit or NX Unit to a mode that allows writing data.   | page 2-927 |
| NX_SaveParam             | Save NX Unit Parameters          | Saves the data that was written to an EtherCAT Coupler Unit or NX Unit.   | page 2-933 |
| PLC_ReadTotalPowerOnTime | Read PLC Total Power ON Time     | Reads the total power ON time from a specified CPU Unit.  | page 2-939 |
| NX_ReadTotalPowerOnTime  | Read NX Unit Total Power ON Time | Reads the total power ON time from a Communications Coupler Unit or NX Unit.  | page 2-942 |

## Program Control Instructions

| Instruction | Name                | Function  | Page       |
|-------------|---------------------|---|------------|
| PrgStart    | Enable Program      | Enables the execution of the specified program. | page 2-952 |
| PrgStop     | Disable Program     | Disables execution of the specified program.    | page 2-961 |
| PrgStatus   | Read Program Status | Reads the status of the specified program.      | page 2-980 |

## EtherCAT Communications Instructions

| Instruction             | Name                          | Function  | Page        |
|-------------------------|-------------------------------|---|-------------|
| EC_CoESDOWrite          | Write EtherCAT CoE SDO        | Writes a value to a CoE object of a specified slave on the EtherCAT network.                        | page 2-986  |
| EC_CoESDORead           | Read EtherCAT CoE SDO         | Reads a value from a CoE object of a specified slave on the EtherCAT network.                       | page 2-989  |
| EC_StartMon             | Start EtherCAT Packet Monitor | Starts packet monitoring for EtherCAT communications.   | page 2-995  |
| EC_StopMon              | Stop EtherCAT Packet Monitor  | Stops execution of packet monitoring for EtherCAT communications.                                   | page 2-1001 |
| EC_SaveMon              | Save EtherCAT Packets         | Saves EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.   | page 2-1003 |
| EC_CopyMon              | Transfer EtherCAT Packets     | Transfers packet data in an internal file in the main memory of the CPU Unit to the SD Memory Card. | page 2-1005 |
| EC_Disconnect-Slave     | Disconnect EtherCAT Slave     | Disconnects the specified slave from the EtherCAT network.  | page 2-1007 |
| EC_ConnectSlave         | Connect EtherCAT Slave        | Connects the specified slave to the EtherCAT network.   | page 2-1015 |
| EC_ChangeEnable-Setting | Enable/Disable EtherCAT Slave | Enables or disables an EtherCAT slave.  | page 2-1017 |
| NX_WriteObj             | Write NX Unit Object          | Writes data to an NX object in an EtherCAT Coupler Unit or NX Unit.                                 | page 2-1037 |
| NX_ReadObj              | Read NX Unit Object           | Reads data from an NX object in an EtherCAT Coupler Unit or NX Unit.                                | page 2-1053 |

## IO-Link Communications Instructions

| Instruction  | Name                        | Function                                | Page        |
|--------------|-----------------------------|---|-------------|
| IOL_ReadObj  | Read IO-Link Device Object  | Reads data from IO-Link device objects. | page 2-1062 |
| IOL_WriteObj | Write IO-Link Device Object | Writes data to IO-Link device objects.  | page 2-1071 |

## EtherNet/IP Communications Instructions

| Instruction | Name   | Function   | Page        |
|-------------|--|--|-------------|
| 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. | page 2-1083 |

| Instruction          | Name   | Function   | Page        |
|----------------------|--|--|-------------|
| CIPOpenWithData-Size | 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. | page 2-1093 |
| 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.  | page 2-1097 |
| 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.   | page 2-1103 |
| CIPSend              | Send Explicit Message Class 3                        | Sends a class 3 CIP message to a specified device on a CIP network.  | page 2-1109 |
| CIPClose             | Close CIP Class 3 Connection                         | Closes the CIP class 3 connection to the specified handle.   | page 2-1114 |
| 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.   | page 2-1117 |
| CIPUCMMWrite         | Write Variable UCMM Explicit                         | Uses a UCMM explicit message to write the value of a variable in another Controller on a CIP network.  | page 2-1123 |
| CIPUCMMSend          | Send Explicit Message UCMM                           | Sends a UCMM CIP message to a specified device on a CIP network.   | page 2-1130 |
| SktUDPCreate         | Create UDP Socket                                    | Creates a UDP socket request to open a servo port for the built-in EtherNet/IP.  | page 2-1141 |
| SktUDPRcv            | UDP Socket Receive                                   | Reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.  | page 2-1149 |
| SktUDPSend           | UDP Socket Send                                      | Sends data from a UDP port for the built-in EtherNet/IP.   | page 2-1152 |
| SktTCPAccept         | Accept TCP Socket                                    | Requests accepting of a TCP socket for the built-in EtherNet/IP.   | page 2-1155 |
| SktTCPConnect        | Connect TCP Socket                                   | Connects to a remote TCP port from the built-in EtherNet/IP.   | page 2-1158 |
| SktTCPRcv            | TCP Socket Receive                                   | Reads the data from the receive buffer for a TCP socket for the built-in EtherNet/IP.  | page 2-1167 |
| SktTCPSend           | TCP Socket Send                                      | Sends data from a TCP port for the built-in EtherNet/IP.   | page 2-1170 |
| SktGetTCPStatus      | Read TCP Socket Status                               | Reads the status of a TCP socket.  | page 2-1173 |
| SktClose             | Close TCP/UDP Socket                                 | Closes the specified TCP or UDP socket for the built-in EtherNet/IP.   | page 2-1176 |
| SktClearBuf          | Clear TCP/UDP Socket Receive Buffer                  | Clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.  | page 2-1179 |
| SktSetOption         | Set TCP Socket Option                                | Sets the option for TCP socket specified for the built-in EtherNet/IP.   | page 2-1182 |
| ModbusTCPcmd         | Send Modbus TCP General Command                      | Sends general commands using Modbus-TCP protocol.  | page 2-1187 |
| ModbusTCPRead        | Send Modbus TCP Read Command                         | Reads data that is requested by sending read commands using Modbus-TCP protocol.   | page 2-1195 |
| ModbusTCPWrite       | Send Modbus TCP Write Command                        | Sends write commands using Modbus-TCP protocol.  | page 2-1203 |
| ChangeIPAdr          | Change IP Address                                    | Changes the IP address of the built-in EtherNet/IP port on a CPU Unit, or the IP address of an EtherNet/IP Unit.   | page 2-1211 |
| ChangeFTPAccount     | Change FTP Account                                   | 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.   | page 2-1220 |

| Instruction         | Name                        | Function   | Page        |
|---------------------|-----------------------------|--|-------------|
| ChangeNTPServer-Adr | Change NTP Server Address   | Changes the NTP server address of the built-in EtherNet/IP port on a CPU Unit, or the NTP server address of an EtherNet/IP Unit. | page 2-1224 |
| FTPGetFileList      | Get FTP Server File List    | Gets a list of the files in the FTP server.  | page 2-1229 |
| FTPGetFile          | Get File from FTP Server    | Downloads a file from the FTP server.  | page 2-1243 |
| FTPPutFile          | Put File onto FTP Server    | Uploads a file to the FTP server.  | page 2-1252 |
| FTPRemoveFile       | Delete FTP Server File      | Deletes a file from the FTP server.  | page 2-1263 |
| FTPRemoveDir        | Delete FTP Server Directory | Deletes a directory from the FTP server.   | page 2-1273 |

## Serial Communications Instructions

| Instruction          | Name  | Function  | Page        |
|----------------------|---|---|-------------|
| ExecPMCR             | Protocol Macro                                  | Requests execution of a communications sequence (protocol data) registered in a Serial Communications Unit.   | page 2-1278 |
| SerialSend           | SCU Send Serial                                 | Sends data in No-protocol Mode from a serial port on a Serial Communications Unit.  | page 2-1292 |
| SerialRcv            | SCU Receive Serial                              | Receives data in No-protocol Mode from a serial port on a Serial Communications Unit. It then clears the receive buffer after reading the data.         | page 2-1303 |
| SerialRcvNoClear     | SCU Receive Serial without Receive Buffer Clear | Receives data in No-protocol Mode from a serial port on a Serial Communications Unit. It does not clear the receive buffer after reading the data.      | page 2-1303 |
| SendCmd              | Send Command                                    | Uses a serial gateway and sends a command to a Serial Communications Unit. Or, sends an explicit command to a DeviceNet Unit or CompoNet Master Unit.   | page 2-1318 |
| 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.  | page 2-1330 |
| 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.  | page 2-1343 |
| 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. | page 2-1358 |
| NX_ModbusRtu-Read    | 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.    | page 2-1369 |
| NX_ModbusRtu-Write   | 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.   | page 2-1380 |
| 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.                                     | page 2-1391 |
| NX_SerialSigRead     | Read Serial Control Signal                      | Reads the CS or DR signal of a serial port on an Option Board.  | page 2-1399 |
| NX_SerialStatus-Read | Read Serial Port Status                         | Reads the status of a serial port on an Option Board.   | page 2-1404 |
| NX_SerialBufClear    | Clear Buffer                                    | Clears the send or receive buffer.  | page 2-1409 |

| Instruction       | Name                         | Function   | Page        |
|-------------------|------------------------------|--|-------------|
| NX_SerialStartMon | Start Serial Line Monitoring | Starts serial line monitoring of an NX-series Communications Interface Unit. | page 2-1419 |
| NX_SerialStopMon  | Stop Serial Line Monitoring  | Stops serial line monitoring of an NX-series Communications Interface Unit.  | page 2-1424 |

## SD Memory Card Instructions

| Instruction         | Name                    | Function   | Page        |
|---------------------|-------------------------|--|-------------|
| 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. | page 2-1430 |
| 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.         | page 2-1436 |
| FileOpen            | Open File               | Opens the specified file in the SD Memory Card.  | page 2-1441 |
| FileClose           | Close File              | Closes the specified file in the SD Memory Card.   | page 2-1445 |
| FileSeek            | Seek File               | Sets a file position indicator in the specified file in the SD Memory Card.  | page 2-1448 |
| FileRead            | Read File               | Reads the data from the specified file in the SD Memory Card.  | page 2-1451 |
| FileWrite           | Write File              | Writes data to the specified file in the SD Memory Card.   | page 2-1459 |
| FileGets            | Get Text String         | Reads a text string of one line from the specified file in the SD Memory Card.                                     | page 2-1467 |
| FilePuts            | Put Text String         | Writes a text string to the specified file in the SD Memory Card.  | page 2-1475 |
| FileCopy            | Copy File               | Copies the specified file in the SD Memory Card.   | page 2-1484 |
| FileRemove          | Delete File             | Deletes the specified file from the SD Memory Card.  | page 2-1492 |
| FileRename          | Change File Name        | Changes the name of the specified file or directory in the SD Memory Card.   | page 2-1497 |
| DirCreate           | Create Directory        | Creates a directory with the specified name in the SD Memory Card.   | page 2-1503 |
| DirRemove           | Delete Directory        | Deletes the specified directory from the SD Memory Card.   | page 2-1506 |
| BackupToMemory-Card | SD Memory Card Backup   | Backs up data to the SD Memory Card.   | page 2-1509 |

## Time Stamp Instructions

| Instruction         | Name   | Function   | Page        |
|---------------------|--|--|-------------|
| 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. | page 2-1524 |
| NX_AryDOutTimeStamp | Write Digital Output Array with Specified Time Stamp | Outputs pulses from a Digital Output Unit that supports time stamp refreshing.                 | page 2-1530 |

## Other Instructions

| Instruction  | Name              | Function                                   | Page        |
|--------------|-------------------|--|-------------|
| ReadNbit_**  | N-bit Read Group  | Reads zero or more bits from a bit string. | page 2-1540 |
| WriteNbit_** | N-bit Write Group | Writes zero or more bits to a bit string.  | page 2-1542 |

| Instruction       | Name  | Function  | Page        |
|-------------------|---|---|-------------|
| ChkRange          | Check Subrange Variable                     | Determines if the value of a variable is within the valid range of the range specification.                                   | page 2-1544 |
| GetMyTaskStatus   | Read Current Task Status                    | Reads the status of the current task.   | page 2-1547 |
| GetMyTaskInterval | Read Current Task Period                    | Reads the task period of the current task.  | page 2-1550 |
| Task_IsActive     | Determine Task Status                       | Determines if the specified task is currently in execution.   | page 2-1552 |
| 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. | page 2-1554 |
| Unlock            | Unlock Tasks                                | Stops an exclusive lock between tasks.  | page 2-1554 |
| ActEventTask      | Activate Event Task                         | Activates an event task.  | page 2-1560 |
| Get**Clk          | Get Clock Pulse Group                       | Outputs a clock pulse at the specified cycle.   | page 2-1567 |
| Get**Cnt          | Get Incrementing Free-running Counter Group | Gets free-running counter values at the specified cycle.  | page 2-1569 |





# 2

## Instruction Descriptions

This section describes the specifications of the instructions that you can use with NJ/NX-series CPU Unit.

---

|  |       |
|--|-------|
| Using this Section .....                   | 2-3   |
| Ladder Diagram Instructions.....           | 2-13  |
| ST Statement Instructions .....            | 2-27  |
| Sequence Input Instructions .....          | 2-47  |
| Sequence Output Instructions .....         | 2-55  |
| Sequence Control Instructions .....        | 2-73  |
| Comparison Instructions .....              | 2-101 |
| Timer Instructions .....                   | 2-135 |
| Counter Instructions .....                 | 2-155 |
| Math Instructions.....                     | 2-177 |
| BCD Conversion Instructions .....          | 2-255 |
| Data Type Conversion Instructions .....    | 2-275 |
| Bit String Processing Instructions .....   | 2-339 |
| Selection Instructions .....               | 2-351 |
| Data Movement Instructions .....           | 2-371 |
| Shift Instructions.....                    | 2-405 |
| Conversion Instructions .....              | 2-423 |
| Stack and Table Instructions.....          | 2-523 |
| FCS Instructions.....                      | 2-565 |
| Text String Instructions .....             | 2-581 |
| Time and Time of Day Instructions.....     | 2-631 |
| Analog Control Instructions .....          | 2-711 |
| System Control Instructions .....          | 2-857 |
| Program Control Instructions .....         | 2-951 |
| EtherCAT Communications Instructions ..... | 2-985 |

|  |               |
|--|---------------|
| <b>IO-Link Communications Instruction .....</b>      | <b>2-1061</b> |
| <b>EtherNet/IP Communications Instructions .....</b> | <b>2-1081</b> |
| <b>Serial Communications Instructions .....</b>      | <b>2-1277</b> |
| <b>SD Memory Card Instructions .....</b>             | <b>2-1429</b> |
| <b>Time Stamp Instructions.....</b>                  | <b>2-1523</b> |
| <b>Other Instructions .....</b>                      | <b>2-1539</b> |

# 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 | <p>The figure that represents the instruction in a ladder diagram is given.</p> <p>●Example for a FUN Instruction ●Example for a FB Instruction</p> <p>The <i>instruction option</i>, <i>upward differentiation specification</i>, and <i>instance specification</i> are described below.</p> <p><b>Instruction option</b> : 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.</p> <p><b>Upward differentiation specification</b> : 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.</p> <p><b>Instance specification</b> : 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.</p> |

| Item          | Description  |
|---------------|--|
| ST expression | <p>The notation that represents the instruction in ST is given.</p> <p>There are two ways that you can use to code an instruction in ST. These are described below.</p> <ol style="list-style-type: none"> <li>1. Directly specifying the Correspondence between the Parameters and the Input, Output, and In-Out Variables<br/>Example: MoveBit(In:=abc, InPos:=def, InOut:=ghi, InOutPos:=jkl);</li> <li>2. Specifying Only the Parameters and Omitting the Input, Output, and In-Out Variables<br/>Example: MoveBit(In, InPos, InOut, InOutPos);</li> </ol> <p>Method 2 is used in this section.</p> <p>You must assign an instance name to any instruction that is given as “XX_instance(variable_name).”</p> <p>Example: TON_instance (In, PT, Q, ET);</p>  |
| Variables     | <ul style="list-style-type: none"> <li>• Name<br/>The input variables, output variables, and in-out variables are given. Example: In1<br/>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.<br/>(EN, ENO, Execute, Done, Busy, Error, ErrorID, ErrorIDEx)</li> <li>• Meaning<br/>The name of the variable is given. Example: Up-counter</li> <li>• I/O<br/>Whether the variable is an input variable, output variable, or in-out variable is given.</li> <li>• Description<br/>The meaning of the variable and any restrictions are given.</li> <li>• Valid range<br/>The range that the variable can take is given.<br/><i>Depends on data type</i> 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.</li> <li>• Unit<br/>The unit of the value that is specified with the variable is given. --- indicates that no unit is required. Example: Bytes</li> <li>• Default<br/>The specified default value is automatically used for the variable if you do not assign a parameter to the instruction before it is executed.<br/>--- indicates the following: <ul style="list-style-type: none"> <li>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.<br/>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.</li> <li>Output variable : Default values are not set.</li> <li>In-out variables : Default values are not set.</li> </ul> </li> <li>• Data Types<br/>The data type of the variable is given.<br/>The use of enumerations, arrays, structures, and unions is also given.</li> </ul> |
| Function      | <p>The function of the instruction is described.</p> <p>Variable names are given in italic text. Example: <i>In1</i></p> <p>Array names are followed by “[]”. Example: InOut[]</p>   |

| Item                                | Description   |
|-------------------------------------|---|
| Related System-defined Variables    | The system-defined variables that are related to the instruction are given. Refer to the <i>NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)</i> 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 the 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*.

|    | Meaning                      | I/O   | Description   | Data type | Valid range   | Default |
|----|------------------------------|-------|---|-----------|---------------|---------|
| EN | Enable (Execution Condition) | Input | TRUE: Instruction is executed.<br>*1<br>FALSE: Instruction is not executed. | BOOL      | TRUE or FALSE | TRUE    |

\*1. If upward differentiation (@) is specified as an instruction option, the instruction is executed when the value of *EN* changes from FALSE to TRUE.

- FB instructions do not have *EN* input variables.
- 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

*ENO* is an output variable which passes the result of instruction execution.

The output variable can be used to pass execution conditions to the next instruction in a ladder diagram.

Normally, when the instruction execution is completed, the value of *ENO* changes to TRUE. Execution of the next instruction is then started.

|     | Meaning       | I/O    | Description   | Data type | Valid range   | Default |
|-----|---------------|--------|---|-----------|---------------|---------|
| ENO | Enable Output | Output | TRUE: Normal end.* <sup>1</sup><br>FALSE: Error end, execution in progress, or execution condition not met. | BOOL      | TRUE or FALSE | ---     |

\*1. The value 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.

## 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 the processing is completed even if the value 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.

|         | Meaning | I/O    | Description  | Data type | Valid range   | Default |
|---------|---------|--------|--|-----------|---------------|---------|
| Execute | Execute | Input  | TRUE: Instruction is executed.<br>* <sup>1</sup><br>FALSE: Instruction is not executed.* <sup>2</sup>                      | BOOL      | TRUE or FALSE | FALSE   |
| Done    | Done    | Output | TRUE: Normal end.* <sup>3</sup> * <sup>4</sup><br>FALSE: Error end, execution in progress, or execution condition not met. | BOOL      | TRUE or FALSE | ---     |
| Busy    | Busy    |        | TRUE: Execution processing is in progress.<br>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, change the value of *Execute* to FALSE.

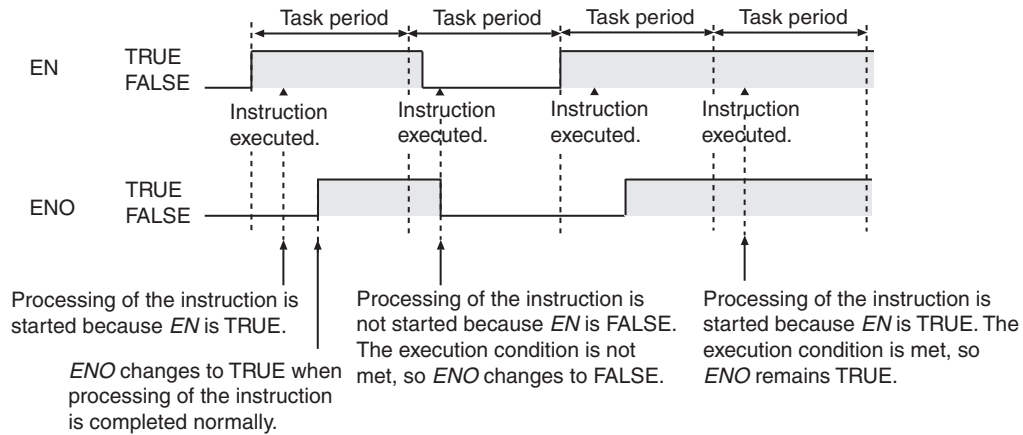
\*2. Processing is continued to the end even if the value changes to FALSE during execution.

\*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.

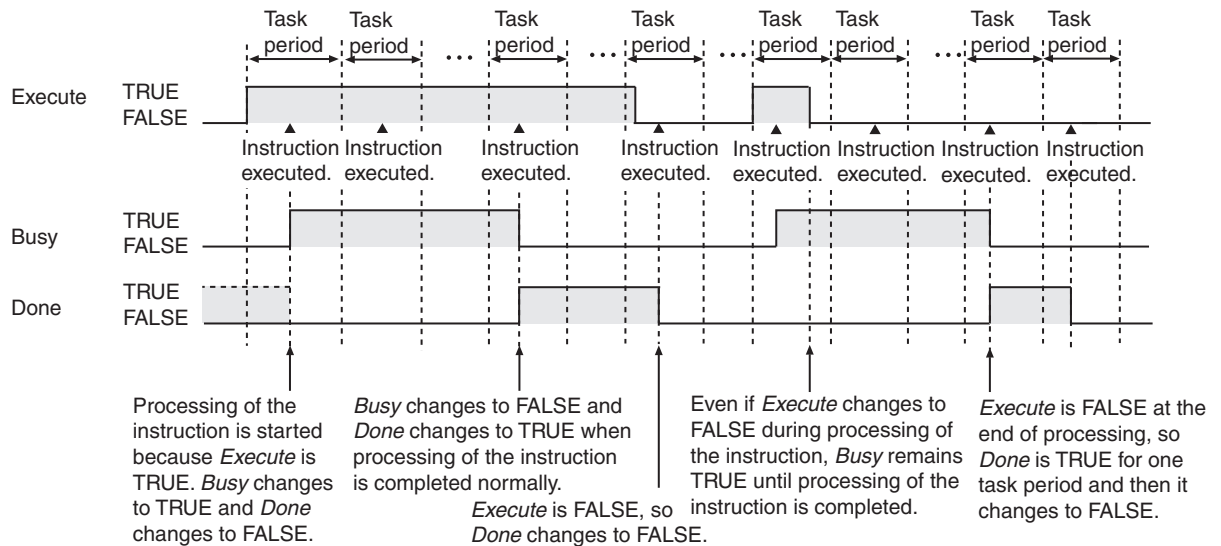
### ● Instructions Completed in One Task Period

Below is a timing chart for an instruction that has *EN* and *ENO* variables (i.e., an instruction to be completed in one task period).



● Instructions Processed over More Than One Task Period

Below is a timing chart for an instruction that has *Execute* and *Busy* variables (i.e., an instruction to be processed over more than one task period).



**Error, ErrorID, and ErrorIDex**

*Error*, *ErrorID*, and *ErrorIDex* are output variables used by some FB instructions, and indicate error end of the instructions.

|           | Meaning              | I/O    | Description   | Data type | Valid range                 | Default |
|-----------|----------------------|--------|---|-----------|-----------------------------|---------|
| Error     | Error                | Output | TRUE: Error end.*1 *2<br>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.<br>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.<br>The value is DWORD#16#0 for a normal end. | DWORD     |                             |         |

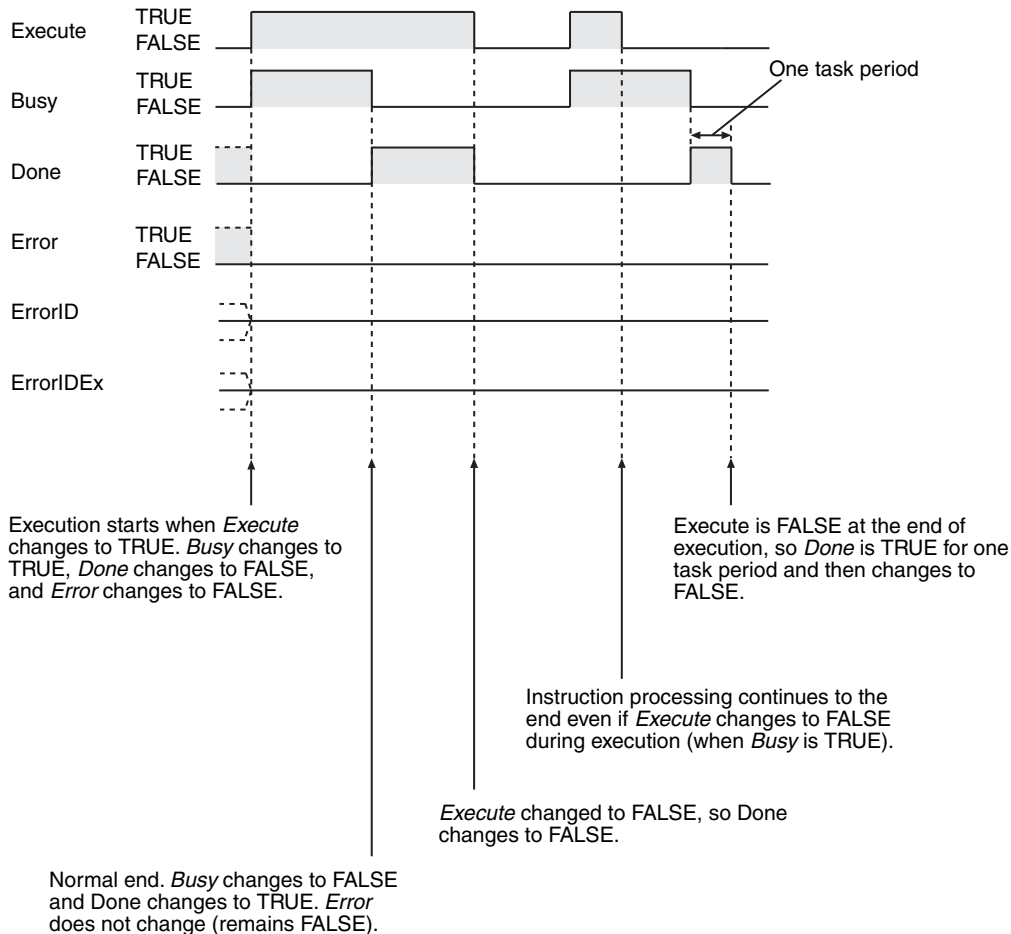
- \*1. The value of *Error* changes to FALSE when the execution condition is no longer met after the error end.
- \*2. If the execution condition is no longer met when an error end occurs, the value of *Error* is TRUE for one task period and it then changes to FALSE.

Refer to *A-1 Error Codes That You Can Check with ErrorID* on page A-2 for a list of error codes, which can be identified with *ErrorID*.

Refer to the *NJ/NX-series Troubleshooting Manual (Cat. No. W503)* for the meanings of the error codes.

### ● Normal End

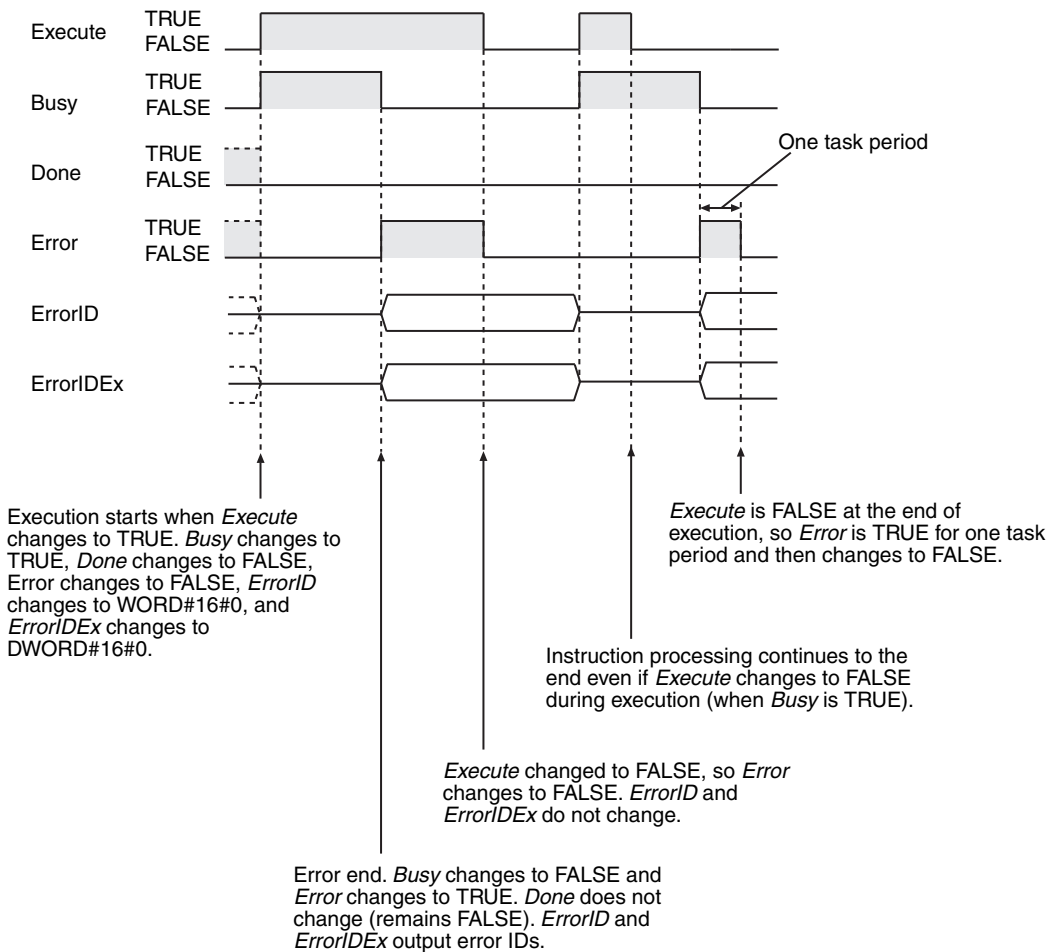
Below is a timing chart for *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*.





● Error End

Below is a timing chart for *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*.



**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 with *Depends on the data type* in the valid range column and with --- 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#00000000         |
|                | LWORD     | LWORD#16#0000000000000000 to FFFFFFFFFFFFFFFF | LWORD#16#0000000000000000 |

| 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                           |
| Real numbers  | REAL      | REAL#-3.402823e+38 to -1.175495e-38,<br>0,<br>+1.175494e-38 to +3.402823e+38,<br>+∞/-∞   | REAL#0                           |
|   | LREAL     | LREAL#-1.79769313486231e+308 to<br>-2.22507385850721e-308,<br>0,<br>+2.22507385850721e-308 to +1.79769313486231e<br>+308,<br>+∞/-∞   | LREAL#0                          |
| Times,<br>durations,<br>dates,<br>and text<br>strings | TIME      | T#-9223372036854.775808ms<br>(T#-106751d_23h_47m_16s_854.775808ms) to<br>T#9223372036854.775807ms<br>(T#+106751d_23h_47m_16s_854.775807ms)                                     | T#0s                             |
|   | DATE      | D#1970-01-01 to D#2106-02-06<br>(January 1, 1970 to February 6, 2106)  | D#1970-01-01                     |
|   | TOD       | TOD#00:00:00.000000000 to<br>TOD#23:59:59.999999999<br>(00:00 and 0.000000000 to 23:59 and 59.999999999<br>seconds)  | TOD#00:00:00.000000000           |
|   | DT        | DT#1970-01-01-00:00:00.000000000 to<br>DT#2106-02-06-23:59:59.999999999<br>(00:00 and 0.000000000 on January 1, 1970 to 23:59<br>and 59.999999999 seconds on February 6, 2106) | DT#1970-01-01-00:00:00.000000000 |
|   | STRING    | Character code: UTF-8<br>0 to 1,986 bytes<br>(1,985 single-byte alphanumeric characters plus the<br>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 in the table.

The following is an example. Here, the data type of the *Out* variable is enumerated type `_eDAYOF-WEEK`. The enumerators are described in the description of the function of the instruction.

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <code>_eDAYOFWEEK</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

## Structures and Unions

The data type for a structure or union variable is given in 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.

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <code>_sPORT</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

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.

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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.

|                            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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>In1[]</code> (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.

|               | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 basic data type for the *In1* variable, or you can specify an entire array or an array element, as well. To specify an entire array, pass its array name as a parameter. To specify an element of an array, pass its array name with its subscript as a parameter.

|     | Boo<br>lean                                      | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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     |
|     | 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 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 array 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.


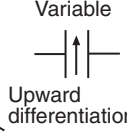
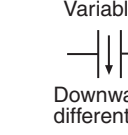

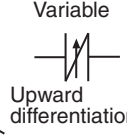
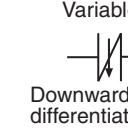
# Ladder Diagram Instructions

| Instruction    | Name              | Page      |
|----------------|-------------------|-----------|
| LD and LDN     | Load/Load NOT     | page 2-14 |
| AND and ANDN   | AND/AND NOT       | page 2-17 |
| OR and ORN     | OR/OR NOT         | page 2-20 |
| Out and OutNot | Output/Output NOT | page 2-23 |

# LD and LDN

LD : Reads the value of a BOOL variable.

LDN : Reads the inverted value of a BOOL variable.

| Instruction | Name     | FB/<br>FUN | Graphic expression   | ST expression |
|-------------|----------|------------|--|---------------|
| LD          | Load     | ---        | Variable  Variable  Variable  | None          |
| LDN         | Load NOT | ---        | Variable  Variable  Variable  | 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.

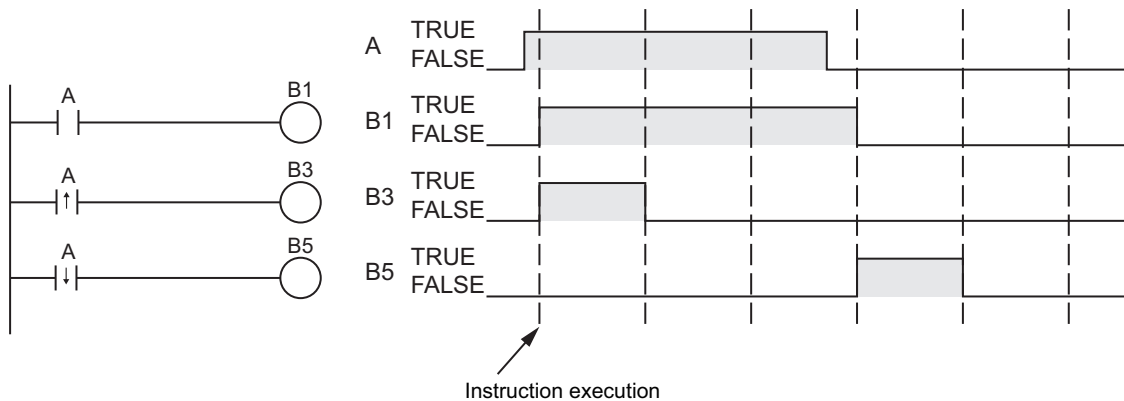
The operation is as shown below if you do not specify upward or downward differentiation.

| Instruction | Value of variable | Output value |
|-------------|-------------------|--------------|
| LD          | TRUE              | TRUE         |
|             | FALSE             | FALSE        |

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        |

The following figure shows a programming example and timing chart.



## 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.

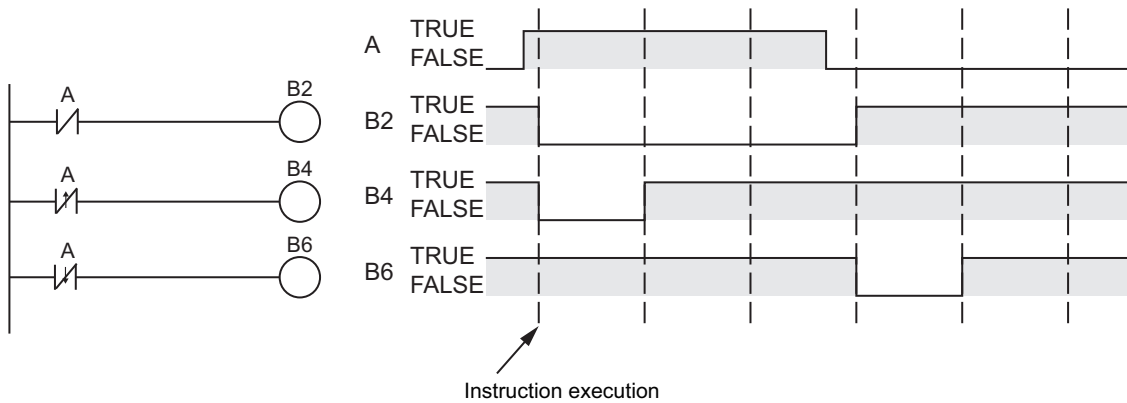
The operation is as shown below if you do not specify upward or downward differentiation.

| Instruction | Value of variable | Output value |
|-------------|-------------------|--------------|
| 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 |
|-------------|-------------------------------|---|--------------|
| 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.
  - a) 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 input value.

**ANDN** : Takes the logical AND of the inverted value of a BOOL variable and the input value.

| Instruction | Name    | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|---------|------------|--------------------|---|
| AND         | AND     | ---        |                    | $\text{result} := \text{vBool1 AND vBool2};$<br>$\text{result} := \text{vBool1 \& vBool2};$ |
| ANDN        | AND NOT | ---        |                    | $\text{result} := \text{vBool1 AND NOT vBool2};$  |

## Variables

None

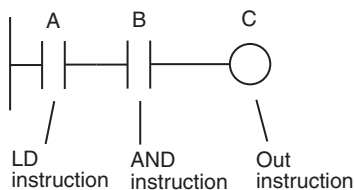
## Function

### 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.

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<br>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<br>Execution condition: TRUE              | TRUE         |
|             |                               | Other than the above.  | FALSE        |
|             | Downward differentiation      | Variable value: TRUE at the last execution → Currently FALSE<br>Execution condition: TRUE              | TRUE         |
|             |                               | Other than the above.  | FALSE        |

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

| Instruction | Combination of variable value and execution condition | Output value |
|-------------|---|--------------|
| ANDN        | Variable value: FALSE<br>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 |
|-------------|-------------------------------|--|--------------|
| ANDN        | Upward differentiation        | Variable value: FALSE at the last execution → Currently TRUE<br>Execution condition: TRUE              | FALSE        |
|             |                               | Variable value: Ignored<br>Execution condition: FALSE  |              |
|             |                               | Other than the above.  |              |
|             | Downward differentiation      | Variable value: TRUE at the last execution → Currently FALSE<br>Execution condition: TRUE              | FALSE        |
|             |                               | Variable value: Ignored<br>Execution condition: FALSE  |              |
|             |                               | Other than the above.  |              |

## Precautions for Correct Use

- An error occurs in the following case and the output value from the last execution is retained.
  - a) 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 inverted value of a BOOL variable and the execution condition.

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

## Variables

None

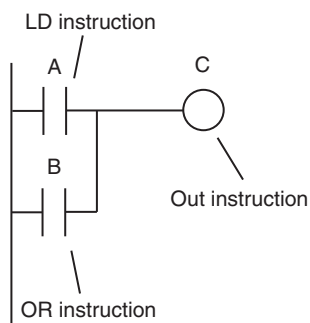
## Function

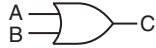
### OR

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.

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<br>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<br>Execution condition: Ignored.          | TRUE         |
|             |                               | Variable value: Ignored<br>Execution condition: TRUE   |              |
|             |                               | Other than the above.  | FALSE        |
|             | Downward differentiation      | Variable value: TRUE at the last execution → Currently FALSE<br>Execution condition: Ignored.          | TRUE         |
|             |                               | Variable value: Ignored<br>Execution condition: TRUE   |              |
|             |                               | Other than the above.  | FALSE        |

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

| Instruction | Combination of variable value and execution condition | Output value |
|-------------|---|--------------|
| ORN         | Variable value: TRUE<br>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 |
|-------------|-------------------------------|--|--------------|
| ORN         | Upward differentiation        | Variable value: FALSE at the last execution → Currently TRUE<br>Execution condition: FALSE             | FALSE        |
|             |                               | Other than the above.  | TRUE         |
|             | Downward differentiation      | Variable value: TRUE at the last execution → Currently FALSE<br>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.
  - a) 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 inverted value of the logical result from the previous instruction and outputs it to a BOOL variable.

| Instruction | Name       | FB/<br>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.

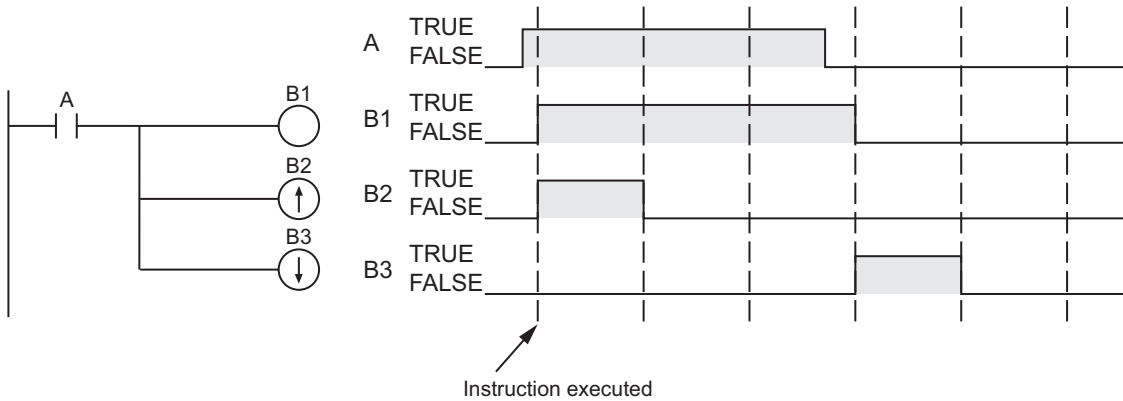
| Logic processing result from previous instruction | Output |
|---|--------|
| TRUE  | TRUE   |
| FALSE   | FALSE  |

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  |

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

The following figure shows a programming example and timing chart.

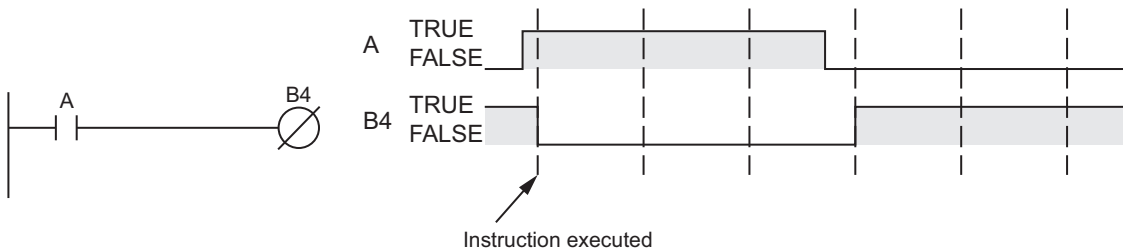


## OutNot

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

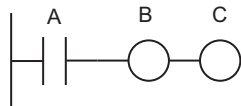
### Differences between the Set and Reset Instructions and the Out and OutNot Instructions

- 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.

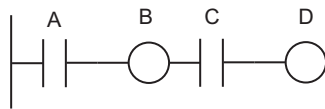


## Precautions for Correct Use

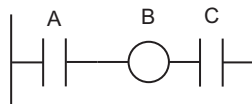
- In the following case, an error occurs and nothing is output.
  - You specify an array element for the variable value and the element does not exist.  
Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- The following connections are possible.
  - You can connect another Out instruction after the Out instruction.



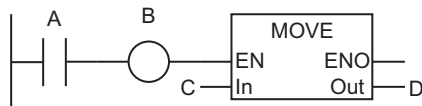
- You can connect the LD instruction and Out instruction after the Out instruction.



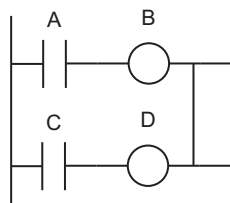
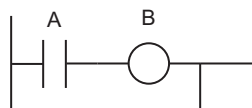
- The following connections are not possible.
  - You cannot connect only the LD instruction after the Out instruction.



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



- Branches and joins cannot be used after Out instructions.





# ST Statement Instructions

| Instruction | Name         | Page      |
|-------------|--------------|-----------|
| IF          | If           | page 2-28 |
| CASE        | Case         | page 2-32 |
| WHILE       | While        | page 2-36 |
| REPEAT      | Repeat       | page 2-39 |
| EXIT        | Break Loop   | page 2-42 |
| RETURN      | Return       | page 2-45 |
| FOR         | Repeat Start | page 2-46 |

# IF

The IF construct selects one of two statements to execute, based on the evaluation result of a specified condition expression.

| Instruction | Name | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|------|------------|--------------------|--|
| IF          | If   | ---        | None               | IF condition expression THEN<br>statement;<br>ELSIF condition expression THEN<br>statement;<br>ELSE<br>statement;<br>END_IF; |

## Variables

None

## Function

The IF construct selects one of two statements to execute, based on the evaluation result of a specified condition expression. Use a condition expression that evaluates to TRUE or FALSE as shown in the table below.

| Item used for condition expression | Example    | Evaluation result   |
|------------------------------------|------------|---|
| Logic expression                   | $a > 3$    | If the value of variable <i>a</i> is greater than 3, the result is TRUE. Otherwise, the result is FALSE.        |
|                                    | $a = b$    | If the values of variables <i>a</i> and <i>b</i> are equal, the result is TRUE. Otherwise, the result is FALSE. |
| BOOL variable                      | <i>abc</i> | If the value of variable <i>abc</i> 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 <i>a</i> and <i>b</i> are equal, the result is TRUE. Otherwise, the result is FALSE.     |
| <>       | Not equals | $a <> b$ | If the values of variables <i>a</i> and <i>b</i> are not equal, the result is TRUE. Otherwise, the result is FALSE. |

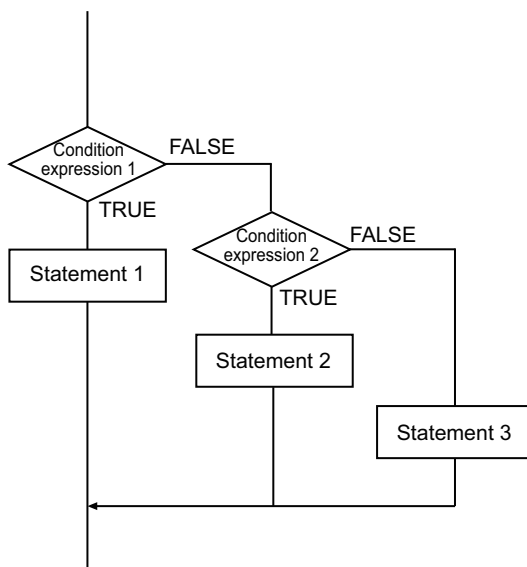
| Operator | Meaning      | Example                 | Evaluation result  |
|----------|--------------|-------------------------|--|
| <        | Comparison   | $a < b$                 | If the value of variable $a$ is less than the value of variable $b$ , the result is TRUE. Otherwise, the result is 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 $b$<br>$a \& b$ | 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 $a$ .   |

In the following flowchart, the IF construct is executed based on the evaluation results of condition expressions 1 and 2. More than one statement can be used in a IF construct, as shown below.

```

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

```



## Additional Information

- IF statements can be nested. 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;

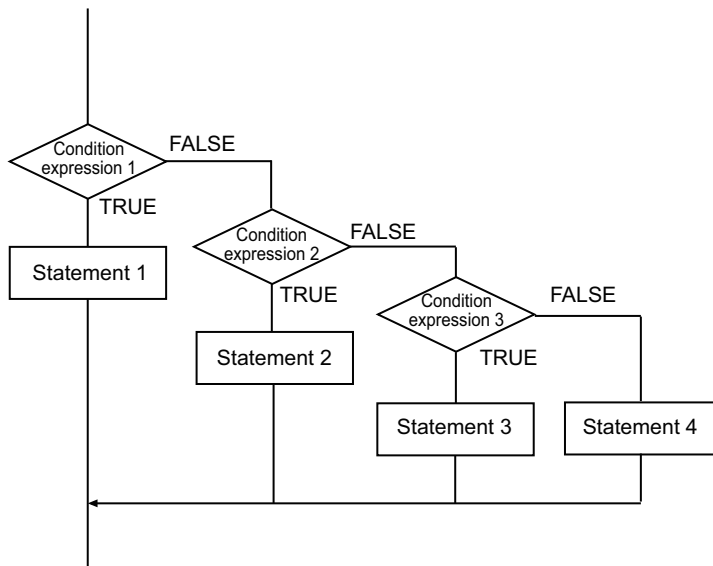
```

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;

```

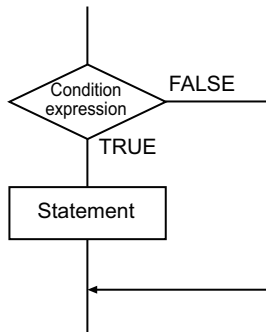


- ELSIF is not needed if the IF construct has only one condition expression. ELSE is not needed either if no processing is required when none of the condition expressions evaluate to TRUE. The following processing flow is for this example.

```

IF condition expression THEN
    statement;
END_IF;

```



- There are no restrictions on the statements that you can use. You can use any statements for the IF construct as you do for other instructions, such as function block calls and FOR statements.

## Precautions for Correct Use

- You must always use IF and END\_IF. IF and END\_IF must be paired.
- You can nest statement constructs up to a maximum of 15 levels, including IF, CASE, FOR, WHILE, and REPEAT statements.

## Sample Programming

In the following example, INT#0 is assigned to variable *def* if the value of variable *abc* is less than INT#0. INT#1 is assigned to variable *def* and INT#2 to variable *ghi* if the value of variable *abc* is INT#0. INT#3 is assigned 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

The CASE construct selects a statement to execute, based on the value of a specified integer expression.

| Instruction | Name | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|------|------------|--------------------|--|
| CASE        | Case | ---        | None               | <pre> CASE integer expression O F     value:         statement;     value:         statement;     :     ELSE         statement; END_CASE; </pre> |

## Variables

None

## Function

The CASE construct selects a statement to execute, based on the value of a specified integer expression.

The following integer expressions and values can be used.

|                         | Allowed notation  |
|-------------------------|---|
| Integer expres-<br>sion | Integer variable, integer constant, integer expression, or a function that returns an integer return value, enumeration variable, enumeration expression, or enumerator |
| Values                  | Integer constants   |

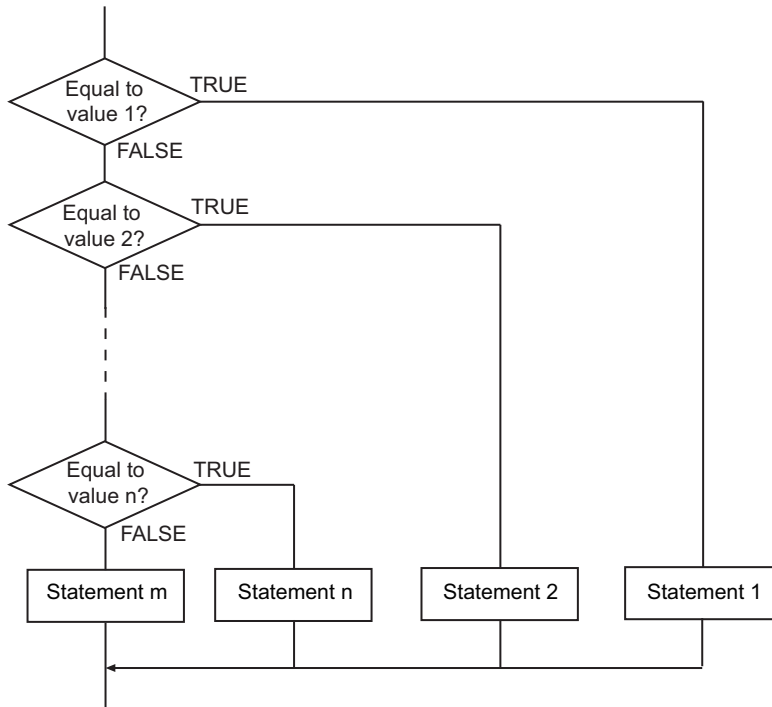
The flowchart in the following example shows the processing flow for an integer expression. You can use more than one statement as shown below.

```

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

```





## Additional Information

- CASE statements can be nested. 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. In the following example, statement 1 is executed 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. In the following example, statement 1 is executed 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 statements that you can use. You can use any statements for the CASE construct as you do for other instructions, such as function block calls and FOR statements.
- The CASE statement behaves differently from the *switch* case statement in C programming. With a *switch* case statement in C programming, all statements after a value that matches the integer expression are executed unless a *break* statement is used. With a CASE statement, only statements selected based on a value that matches the integer expression are executed. In the following example, statements 1 to 3 are executed for the *switch* statement in C programming. On the other hand, only statement 1 is executed for the CASE instruction.

| C Language switch Statement  | CASE Instruction  |
|--|---|
| <pre> val=1; switch val {     case 1:         <b>statement 1;</b>     case 2:         <b>statement 2;</b>     case 3:         <b>statement 3;</b> } </pre> | <pre> val:=1; CASE val OF     1:         <b>statement 1;</b>     2:         <b>statement 2;</b>     3:         <b>statement 3;</b> END_CASE; </pre> |

## Precautions for Correct Use

- You must always use CASE and END\_CASE. CASE and END\_CASE must be paired.
- The data types of the integer expression and values can be different.
- Each value can be given only once.

- You can nest statement constructs up to a maximum of 15 levels, including IF, CASE, FOR, WHILE, and REPEAT statements.

## Sample Programming

If the value of variable *abc* is INT#1, INT#10 is assigned to variable *def*. Similarly, INT#20 is assigned for INT#2, and INT#30 for INT#3. For any other value, the value of variable *ghi* is assigned.

| 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;

```

If the value of variable *abc* is INT#1, INT#10 is assigned to variable *def*. Similarly, INT#20 is assigned for either INT#2 or INT#5, and INT#30 for a value between INT#6 and INT#10, inclusive. For any other value, no value is assigned.

| 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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------|------------|--------------------|---|
| WHILE       | While | ---        | None               | WHILE condition expression DO<br>statement;<br>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 as shown in the table below.

| 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.           |

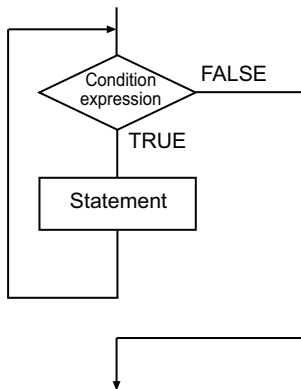
You can use the following operators in the logic expression.

| 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. |

| Operator | Meaning      | Example          | Evaluation result  |
|----------|--------------|------------------|--|
| AND (&)  | Logical AND  | a AND b<br>a & b | The result is the logical AND of BOOL variables <i>a</i> and <i>b</i> .          |
| OR       | Logical OR   | a OR b           | The result is the logical OR of BOOL variables <i>a</i> and <i>b</i> .           |
| XOR      | Exclusive OR | a XOR b          | The result is the logical exclusive OR of BOOL variables <i>a</i> and <i>b</i> . |
| NOT      | NOT          | NOT a            | The result is the NOT of BOOL variable <i>a</i>                                  |

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

```
WHILE condition expression DO
    statement;
END_WHILE;
```



## Additional Information

- If the first condition expression evaluates to FALSE, the following statement will not be executed.
- There are no restrictions on statements that you can use. You can use any statements for the WHILE construct as you do for other instructions, such as function block calls and FOR statements.
- Execute the EXIT instruction to cancel repeat processing. The processing between the EXIT instruction and the END\_WHILE instruction will not be executed.

## Precautions for Correct Use

- You must always use WHILE and END\_WHILE. WHILE and END\_WHILE must be paired.
- You can nest statement constructs up to a maximum of 15 levels, including IF, CASE, FOR, WHILE, and REPEAT statements.

## Sample Programming

INT#7 is repeatedly added 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 becomes TRUE.

| Instruction | Name   | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|--------|------------|--------------------|---|
| REPEAT      | Repeat | ---        | None               | REPEAT<br>statement;<br>UNTIL condition expression<br>END_REPEAT; |

## Variables

None

## Function

The REPEAT construct executes a statement once, and then executes it repeatedly until a specified condition expression becomes TRUE. Use a condition expression that evaluates to TRUE or FALSE as shown in the table below.

| 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.           |

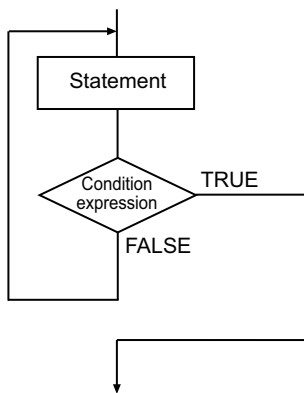
You can use the following operators in the logic expression.

| 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. |

| Operator | Meaning      | Example          | Evaluation result  |
|----------|--------------|------------------|--|
| AND (&)  | Logical AND  | a AND b<br>a & b | The result is the logical AND of BOOL variables <i>a</i> and <i>b</i> .          |
| OR       | Logical OR   | a OR b           | The result is the logical OR of BOOL variables <i>a</i> and <i>b</i> .           |
| XOR      | Exclusive OR | a XOR b          | The result is the logical exclusive OR of BOOL variables <i>a</i> and <i>b</i> . |
| NOT      | NOT          | NOT a            | The result is the NOT of BOOL variable <i>a</i>                                  |

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 statements that you can use. You can use any statements for the REPEAT construct as you do for other instructions, such as function block calls and FOR statements.
- Execute the EXIT instruction to cancel repeat processing. The processing between the EXIT instruction and the END\_REPEAT instruction will not be executed.

### Precautions for Correct Use

- You must always use REPEAT, UNTIL, and END\_REPEAT. REPEAT, UNTIL, and END\_REPEAT must be used as a set.
- You can nest statement constructs up to a maximum of 15 levels, including IF, CASE, FOR, WHILE, and REPEAT statements.

### Sample Programming

INT#1 is repeatedly added 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 ends repeat processing for the FOR, WHILE, or REPEAT instruction of the innermost loop.

| Instruction | Name       | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|------------|------------|--------------------|--|
| EXIT        | Break Loop | ---        | None               | <pre>FOR Index:=0 TO 9 BY 1 DO   IF Error[Index] THEN     EXIT;   END_IF; END_FOR;</pre> |

## Variables

None

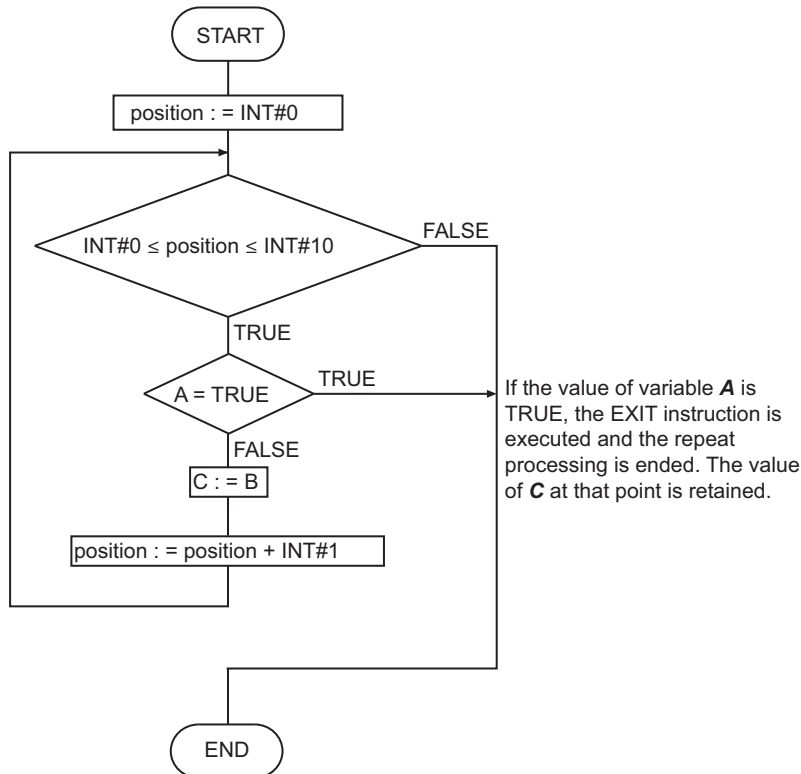
## Function

The EXIT instruction ends repeat processing for the FOR, WHILE, or REPEAT instruction of the innermost loop. Processing moves to the next instruction after the repeat processing.

In the following programming, the value of variable *A* is checked every time the FOR instruction is processed for repeat processing. 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 previous 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.



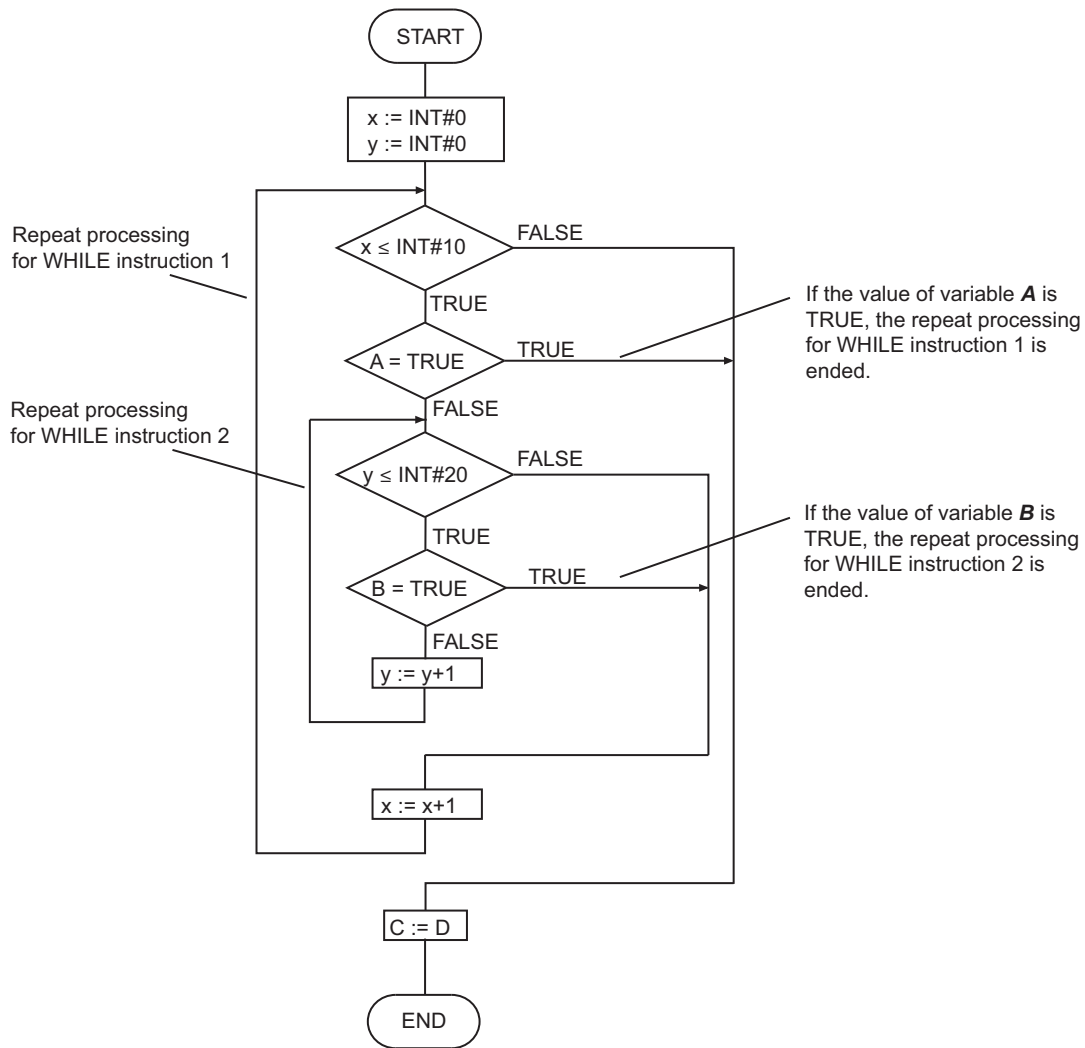
When the EXIT instruction is executed, only the innermost repeat processing is ended. In the following programming, when the value of variable *B* is TRUE, EXIT instruction 2 is executed and the repeat processing for WHILE instruction 2 is ended. And processing moves to  $x:=x+1$ ;. In this case, repeat 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. And 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 the instruction, *RETURN* on page 2-75, in the Sequence Control Instructions for a description of this instruction.

# FOR

---

Refer to the instructions, *FOR* and *NEXT* on page 2-91, 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   | page 2-48 |
| TestABit and TestABitN        | Test A Bit/ Test A Bit NOT | page 2-52 |

# 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/<br>FUN | Graphic expression | ST expression            |
|-------------|--------------|------------|--------------------|--------------------------|
| R_TRIG      | Up Trigger   | FB         |                    | R_TRIG_instance(Clk, Q); |
| Up          |              | FUN        |                    | None                     |
| F_TRIG      | Down Trigger | FB         |                    | F_TRIG_instance(Clk, Q); |
| Down        |              | FUN        |                    | None                     |

## Variables

|         | 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. | ---  | ---     |

|         | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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

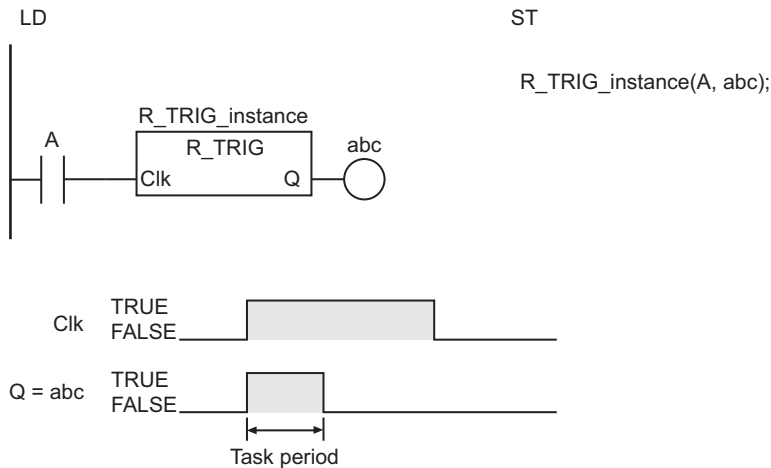
### R\_TRIG

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.

The functions of the R\_TRIG instruction and the Up instruction are the same.

The following figure shows a programming example and timing chart.





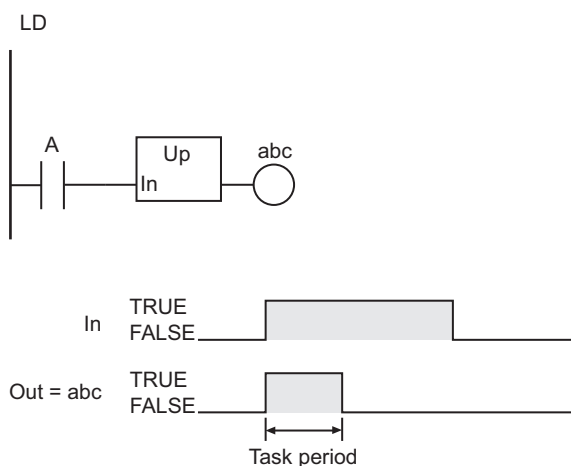
## Up

The Up instruction assigns TRUE to output signal *Out* for one task period only when input signal *In* changes to TRUE. Otherwise, the value of *Out* is FALSE.

The functions of the R\_TRIG instruction and the Up instruction are the same.

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 *Precautions for Correct Use* on page 2-50 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.

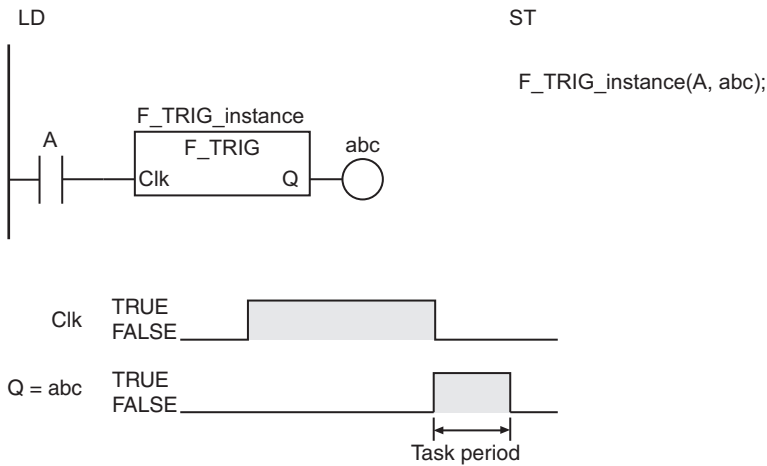


## F\_TRIG

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.

The functions of the F\_TRIG instruction and the Down instruction are exactly the same.

The following figure shows a programming example and timing chart.

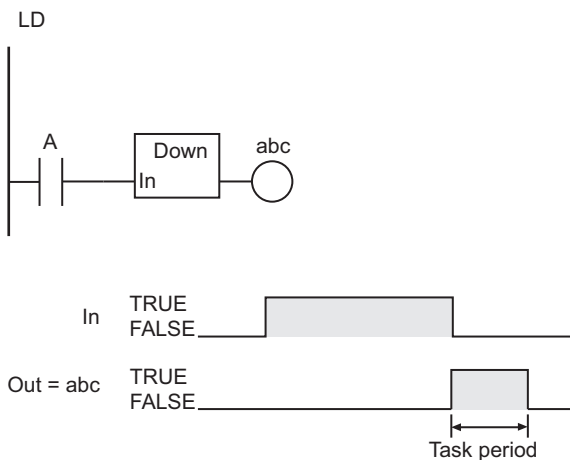


## Down

The Down instruction assigns TRUE to output signal *Out* for one task period only when input signal *In* changes to FALSE. Otherwise, the value of *Out* is FALSE.

The functions of the F\_TRIG instruction and the Down instruction are exactly the same.

The following figure shows a programming example and timing chart.



## 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 if this instruction is not executed every task period as the JMP instruction or any other instruction is used.
- 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 always 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 always 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 always 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

If the value of *Clk* is TRUE, the value of *Q* varies depending on the unit version of the CPU Unit when the R\_TRIG instruction is executed at the timing described in the table below.

| Timing of execution of R_TRIG when <i>Clk</i> is TRUE | Value of <i>Q</i>                        |  |
|---|--|--|
|   | 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         | TRUE                                     | Always TRUE  |
| When the power supply is turned ON                    | TRUE                                     | The value of <i>Q</i> remains FALSE until the value of <i>Clk</i> 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 inverted value of the specified bit in a bit string.

| Instruction | Name           | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|----------------|------------|--------------------|---------------------------|
| TestABit    | Test A Bit     | FUN        |                    | Out:=TestABit (In, Pos);  |
| TestABitN   | Test A Bit NOT | FUN        |                    | Out:=TestABitN (In, Pos); |

## Variables

|     | Meaning      | I/O    | Description  | Valid range                         | Unit | Default |
|-----|--------------|--------|--|-------------------------------------|------|---------|
| In  | Bit string   | Input  | Bit string   | Depends on data type.               | ---  | *1      |
| Pos | Bit position |        | Specified bit position   | 0 to the number of bits in $In - 1$ |      | 0       |
| Out | Bit value    | Output | <ul style="list-style-type: none"> <li>TestABit<br/>Value of specified bit</li> <li>TestABitN<br/>Inverse of value of specified bit</li> </ul> | Depends on data type.               | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

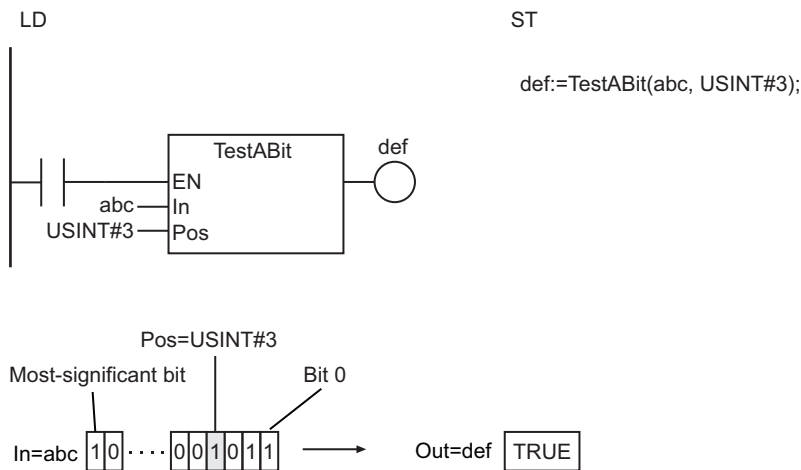
## Function

### 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.

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



## TestABitN

The TestABitN instruction assigns the inverted 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.

## Precautions for Correct Use

- If the TestABit instruction and the TestABitN instruction are used in a ladder diagram, the value of *Out* changes to FALSE when an error occurs in the previous instruction on the rung.
- An error occurs in the following case. *Out* will be FALSE.
  - a) The value of *Pos* is greater than the number of bits in *In* - 1.



## Sequence Output Instructions

| Instruction           | Name                  | Page      |
|-----------------------|-----------------------|-----------|
| RS                    | Reset-Priority Keep   | page 2-56 |
| SR                    | Set-Priority Keep     | page 2-59 |
| Set and Reset         | Set/Reset             | page 2-62 |
| SetBits and ResetBits | Reset Bits/Reset Bits | page 2-66 |
| SetABit and ResetABit | Set A Bit/Reset A Bit | page 2-69 |
| OutABit               | Output A Bit          | page 2-71 |

# RS

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

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

## Variables

|           | Meaning | I/O    | Description | Valid range           | Unit | Default |
|-----------|---------|--------|-------------|-----------------------|------|---------|
| Set *1    | Set     | Input  | Set input   | Depends on data type. | ---  | 0       |
| Reset1 *1 | Reset   |        | Reset input |                       |      |         |
| Q1        | Keep    | Output | Keep output | Depends on data type. | ---  | ---     |

- \*1. On Sysmac Studio version 1.03 or higher, 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);.

|        | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

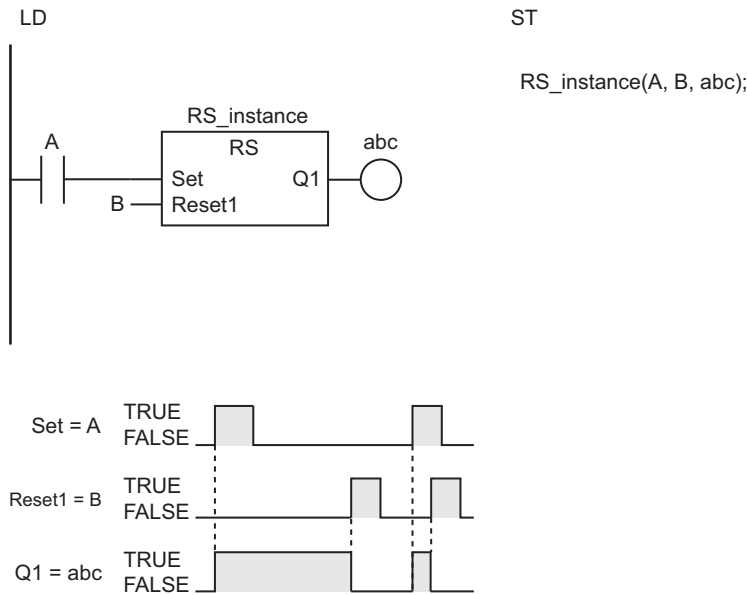
## Function

The RS instruction forms a self-holding output that gives priority to resetting. The following table shows input values and resulting output values.

| Value of Set | Value of Reset1 | Value of Q1  |
|--------------|-----------------|--------------|
| TRUE         | TRUE            | FALSE        |
| TRUE         | FALSE           | TRUE         |
| FALSE        | TRUE            | FALSE        |
| FALSE        | FALSE           | Not changed. |

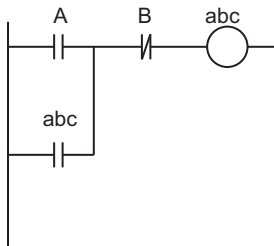
The following figure shows a programming example and timing chart.





## 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 when 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:
  - a) 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.
  - b) 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 change to FALSE when the operating mode is changed to RUN mode and the instruction is executed. If the self-holding rung given in *Additional Information* on page 2-57 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/<br>FUN | Graphic expression | ST expression                 |
|-------------|----------------------|------------|--------------------|-------------------------------|
| SR          | Set-Priority<br>Keep | FB         |                    | SR_instance(Set1, Reset, Q1); |

## Variables

|         | Meaning | I/O    | Description | Valid range           | Unit | Default |
|---------|---------|--------|-------------|-----------------------|------|---------|
| Set1 *1 | Set     | Input  | Set input   | Depends on data type. | ---  | 0       |
| Reset   | Reset   |        | Reset input |                       |      |         |
| Q1      | Keep    | Output | Keep output | Depends on data type. | ---  | ---     |

- \*1. On Sysmac Studio version 1.03 or higher, 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=>abc);.

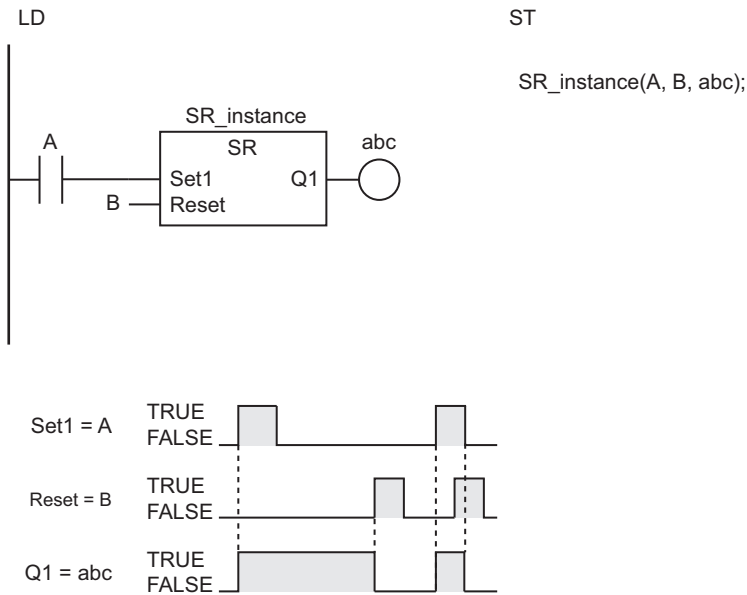
|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |
|-------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|
|       | 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 input values and resulting output values.

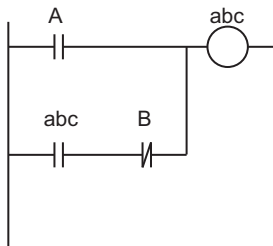
| Value of <i>Set1</i> | Value of <i>Reset</i> | Value of <i>Q1</i> |
|----------------------|-----------------------|--------------------|
| TRUE                 | TRUE                  | TRUE               |
| TRUE                 | FALSE                 | TRUE               |
| FALSE                | TRUE                  | FALSE              |
| FALSE                | FALSE                 | Not changed.       |

The following figure shows a programming example and timing chart.



### Additional Information

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



- However, if the SR instruction is in a master control region and if 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      | 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 when an error occurs in the previous instruction on the rung.
- If this instruction is not executed due to 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:
  - a) If the value of *Reset* is TRUE, the value of *Q1* is retained. If it is FALSE, the value of *Q1* changes to FALSE.
  - b) 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 change to 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/<br>FUN | Graphic expression | ST expression |
|-------------|-------|------------|--------------------|---------------|
| Set         | Set   | ---        |                    | None          |
| Reset       | Reset | ---        |                    | None          |

## Variables

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

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 *Out* to FALSE even if the input changes to FALSE.

Use the Reset instruction to change *Out* to FALSE.

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

| Instruction | Input | Output       |
|-------------|-------|--------------|
| Set         | TRUE  | TRUE         |
|             | 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

The Reset instruction changes *Out* to FALSE if the input is TRUE.

If *Out* is FALSE, the Reset instruction will not change *Out* 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.

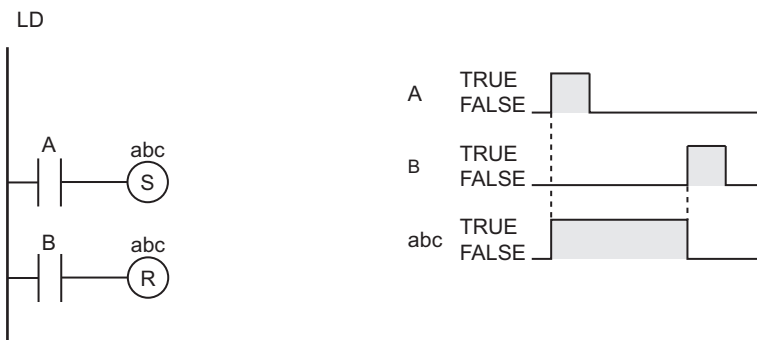
| Instruction | Input | Output       |
|-------------|-------|--------------|
| Reset       | TRUE  | FALSE        |
|             | FALSE | Not changed. |

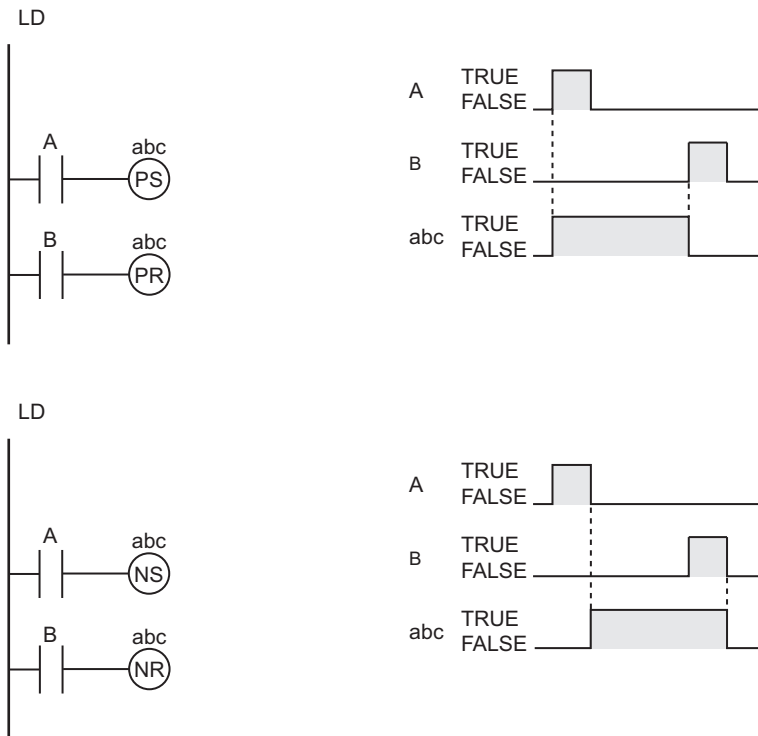
If you specify upward or downward differentiation, the operation depends on the following: the value of the input for the last execution and the current value of the input. This is shown below.

| Instruction | Differentiation specification | Value of input at last execution and current value | Output value |
|-------------|-------------------------------|--|--------------|
| 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. |

## Programming Example and Timing Chart

The following figures show programming examples and timing charts.





## 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 the Set instruction and the Reset instruction are in a master control region and if the master control region is reset, the value of *Out* is retained.
- If these instructions are not executed due to execution of a jump instruction (e.g., the JMP instruction), the value of *Out* is retained.



- These instructions will not be executed if you specify upward differentiation and if the input is TRUE at the time of power-on. They will be executed only when the input changes to FALSE and then back to TRUE.
- These instructions will be executed if you do not specify upward differentiation and if the input is TRUE at the time of power-on. In this case, it is not necessary to change the input to FALSE before the execution.

# 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/<br>FUN | Graphic expression | ST expression                |
|-------------|------------|------------|--------------------|------------------------------|
| SetBits     | Set Bits   | FUN        |                    | SetBits(InOut, Pos, Size);   |
| ResetBits   | Reset Bits | FUN        |                    | ResetBits(InOut, Pos, Size); |

## Variables

|       | 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 the number of bits in <i>InOut</i><br>-1 | ---  | 0       |
| Size  | Number of bits |        | Number of bits         | 0 to the number of bits in <i>InOut</i>       |      |         |
| Out   | Return value   | Output | Always TRUE            | TRUE only                                     | ---  | ---     |

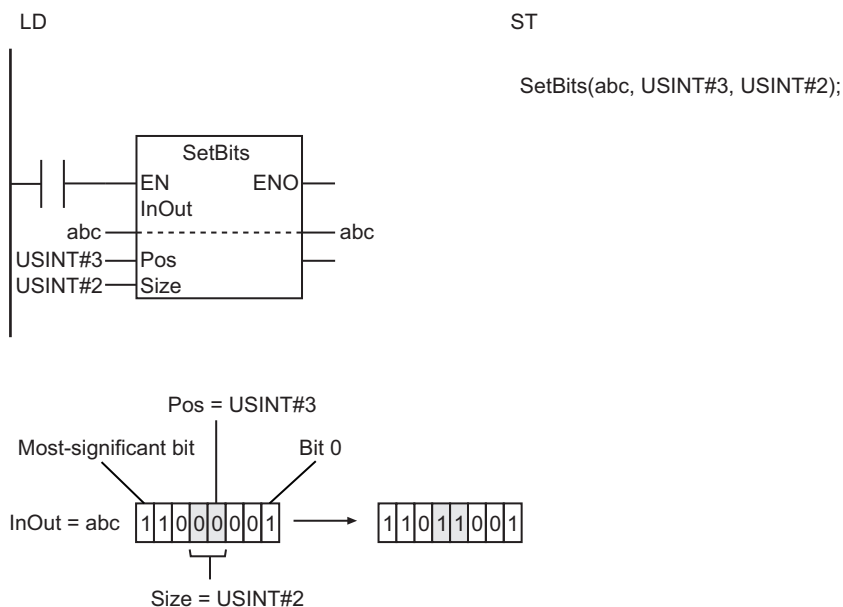
|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|       | 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.

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



### 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.

## 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 the SetBits instruction and the RestBits instruction are in a master control region and the master control region is reset, the value of *InOut* is retained.
- If these instructions are not executed due to 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 these instructions are used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - a) The value of *Pos* is greater than the number of bits in *InOut* - 1.
  - b) The value of *Size* is outside the valid range.

- c) 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/<br>FUN | Graphic expression | ST expression          |
|-------------|-------------|------------|--------------------|------------------------|
| SetABit     | Set A Bit   | FUN        |                    | SetABit(InOut, Pos);   |
| ResetABit   | Reset A Bit | FUN        |                    | ResetABit(InOut, Pos); |

## Variables

|       | 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 the number of bits in <i>InOut</i><br>-1 | ---  | 0       |
| Out   | Return value | Output | Always TRUE            | TRUE only                                     | ---  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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

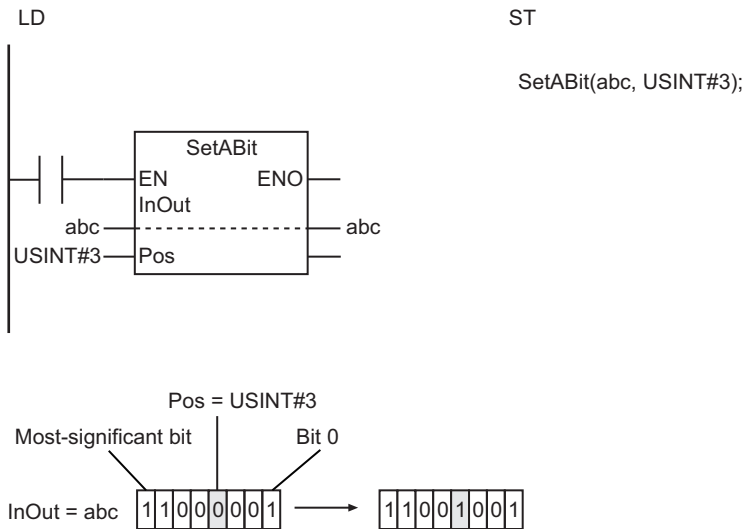
### SetABit

The SetBits 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.

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



## 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.

## 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 the SetABit instruction and the ResetABit instruction are in a master control region and the master control region is reset, the value of *InOut* is retained.
- If these instructions are not executed due to execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when these instructions are used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - a) The value of *Pos* is greater than the number of bits in *In* - 1.

# OutABit

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

| Instruction | Name         | FB/<br>FUN | Graphic expression | ST expression                |
|-------------|--------------|------------|--------------------|------------------------------|
| OutABit     | Output A Bit | FUN        |                    | OutABit(InOut, Pos, BitVal); |

## Variables

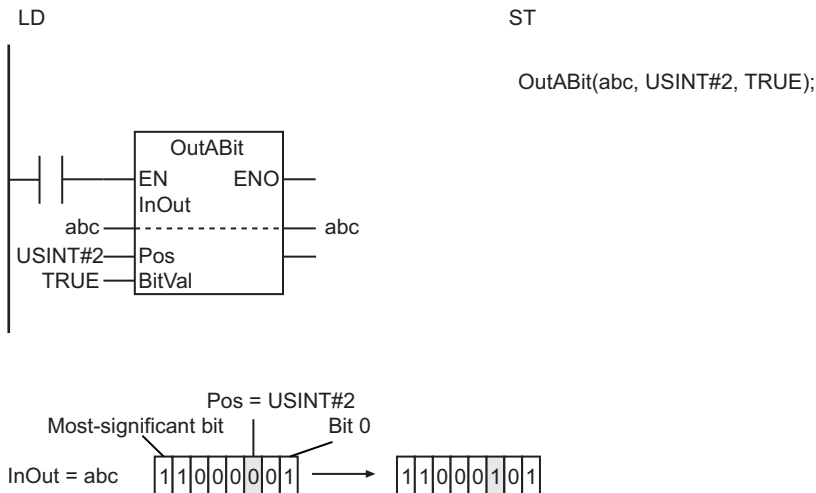
|        | Meaning      | I/O    | Description            | Valid range                                | Unit | Default |
|--------|--------------|--------|------------------------|--|------|---------|
| InOut  | Bit string   | In-out | Bit string             | Depends on data type.                      | ---  | ---     |
| Pos    | Bit position | Input  | Specified bit position | 0 to the number of bits in <i>InOut</i> -1 | ---  | 0       |
| BitVal | Set value    |        | Value to set           | Depends on data type.                      | ---  | TRUE    |
| Out    | Return value | Output | Always TRUE            | TRUE only                                  | ---  | ---     |

|        | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 shows an example where *Pos* is *USINT#2* and *BitVal* is 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 of the specified bit by changing 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 execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - a) The value of *Pos* is greater than the number of bits in *InOut* - 1.

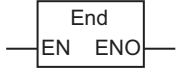


# Sequence Control Instructions

| Instruction  | Name                                    | Page      |
|--------------|---|-----------|
| End          | End                                     | page 2-74 |
| RETURN       | Return                                  | page 2-75 |
| MC and MCR   | Master Control Start/Master Control End | page 2-76 |
| JMP          | Jump                                    | page 2-89 |
| FOR and NEXT | Repeat Start/Repeat End                 | page 2-91 |
| BREAK        | Break Loop                              | page 2-98 |

# End

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

| Instruction | Name | FB/<br>FUN | Graphic expression  | ST expression |
|-------------|------|------------|---|---------------|
| End         | End  | FUN        |  | None          |

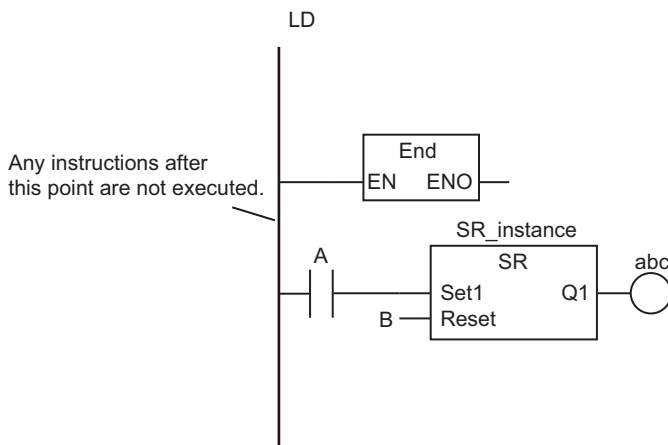
## Variables

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/<br>FUN | Graphic expression  | ST expression |
|-------------|--------|------------|---|---------------|
| RETURN      | Return | FUN        |  | RETURN;       |

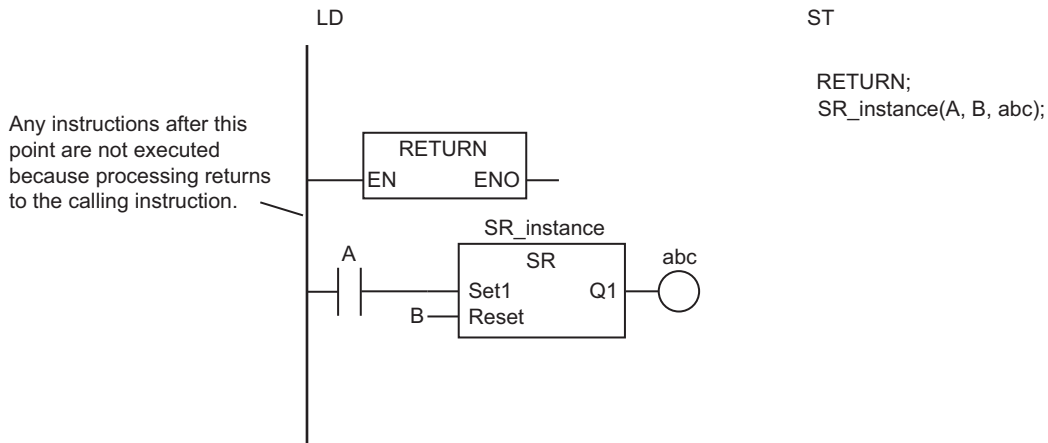
## Variables

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.



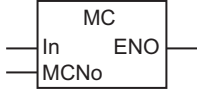
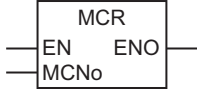
## Precautions for Correct Use

- Observe the following precautions if you use this instruction in a ladder diagram.
  - a) Use this instruction only in functions and function blocks. If you use it in a program, a building error will occur.
  - b) Always connect this instruction directly to the left bus bar.
- Before you execute this instruction, set the return value, 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/<br>FUN | Graphic expression  | ST expression |
|-------------|----------------------|------------|---|---------------|
| MC          | Master Control Start | ---        |  | None          |
| MCR         | Master Control End   | ---        |  | None          |

## Variables

|                             | Meaning               | I/O   | Description                              | Valid range           | Unit | Default |
|-----------------------------|-----------------------|-------|--|-----------------------|------|---------|
| In<br>(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 <sup>*1</sup> |      |         |

\*1. 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<br>(MC instruction only) | OK      |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| MCNo                        |         |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |

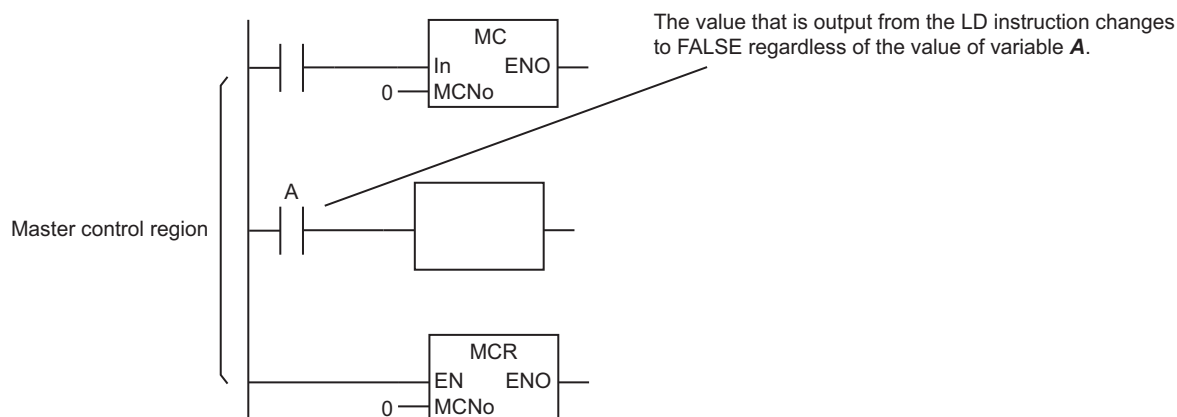
## Function

Master control is used to stop processing or place in an equivalent status all POU's 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 the 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 POU's that follow the LD instructions, as a rule, operate as if the execution condition is FALSE. There are, however, some POU's that operate differently. This is explained later.



If the value of *In* is TRUE, a master control reset is not performed. The POU's in the master control region operate normally.

## POU Operation during a Master Control Reset

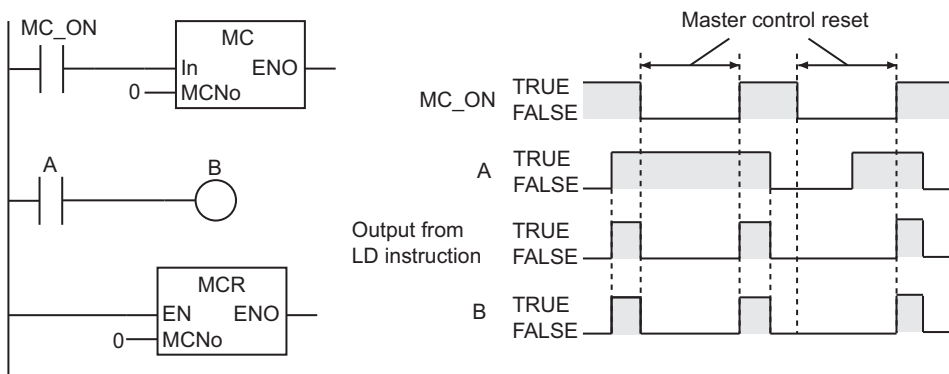
The operation of the POU's when master control is reset depends on the POU as described in the following table.

| 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 FALSE for timer input <i>In</i> . That means that the timer is reset.<br>The value of elapsed time <i>ET</i> changes to 0 and the value of timer output <i>Q</i> changes to FALSE.  |
| TOF instruction              | The instruction operates with TRUE for timer input <i>In</i> . That means that the timer is reset.<br>The value of elapsed time <i>ET</i> changes to 0 and the value of timer output <i>Q</i> changes to TRUE.<br>However, if the Out instruction is connected to <i>Q</i> , the execution condition to the Out instruction is FALSE.   |
| TP instruction               | The instruction operates with FALSE for timer input <i>In</i> . That means that the timer is reset.<br>Timing active : The value of elapsed time <i>ET</i> is incremented to the end and then returns to 0. The value of timer output <i>Q</i> is TRUE until the end of timing, and then it changes to FALSE.<br>Timing not active : The value of <i>ET</i> changes to 0 and the value of <i>Q</i> changes to FALSE.<br>However, if the Out instruction is connected to <i>Q</i> , the execution condition to the Out instruction is FALSE even while timing is active. |

| POU  | Operation   |
|--|---|
| AccumulationTimer instruction  | The instruction operates with FALSE for timer input <i>In</i> . That means that the timer stops.<br>The values of elapsed time <i>ET</i> and timer output <i>Q</i> are retained.<br>However, if the Out instruction is connected to <i>Q</i> , the execution condition to the Out instruction is FALSE even if the value of <i>Q</i> is TRUE.<br>However, reset <i>Reset</i> is enabled.  |
| Timer instruction  | The instruction operates with FALSE for timer input <i>In</i> . That means that the timer is reset.<br>Remaining time <i>ET</i> is set to the value of set time <i>PT</i> , and the value of timer output <i>Q</i> 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.<br>If an Out instruction is connected to the Counter Completion Flag, <i>Q</i> , the execution condition to the Out instruction is FALSE.  |
| JMP instruction  | This instruction is not executed.   |
| 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 <i>Done</i> , <i>Busy</i> , and <i>Error</i> output variables) | The power flow from the left bus bar changes to FALSE.<br>If this instruction is being executed when the master control reset is attempted, the execution is continued until completed. <i>Busy</i> , <i>Done</i> , and <i>Error</i> 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 <i>Busy</i> , <i>Done</i> , or <i>Error</i> , the variable will be assigned a proper value as specified in the instruction specifications.<br>You can also get the value of <i>Busy</i> , <i>Done</i> , or <i>Error</i> in the form of <i>instance_name.output_variable</i> . |
| Other functions  | These are not executed.   |
| Other function blocks  | The power flow from the left bus bar changes to FALSE.  |

● **Out**

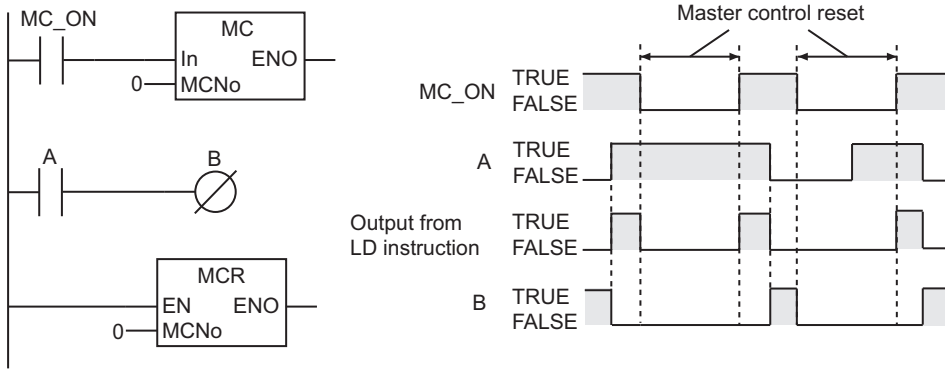
FALSE is output while the master control is reset.



● **OutNot**

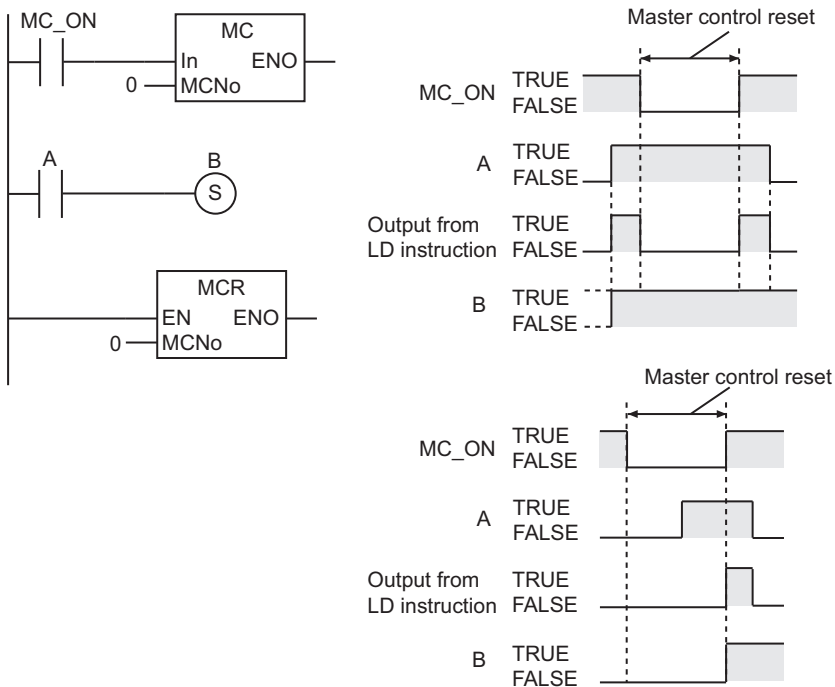
FALSE is output while the master control is reset.

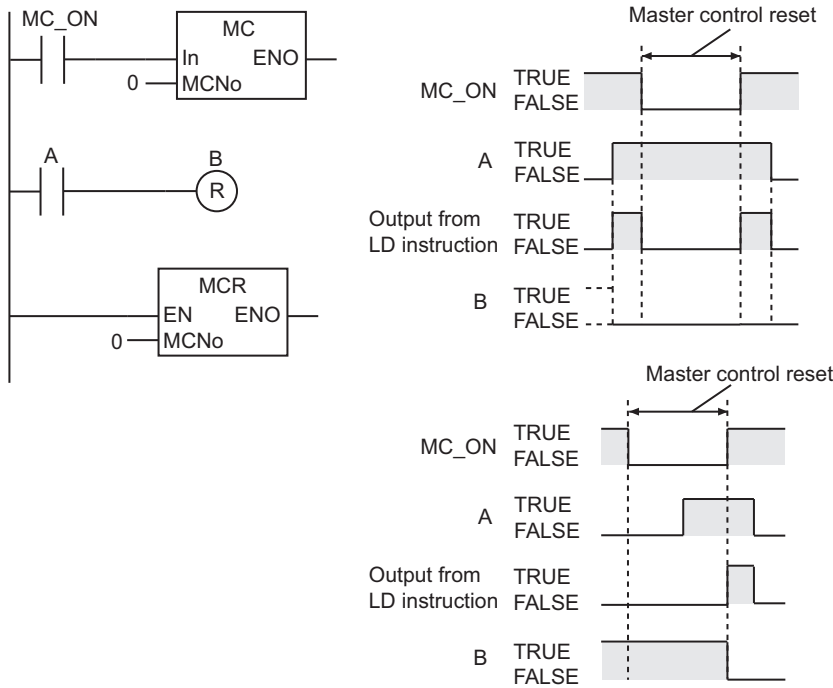
Caution is required because this operation of the **OutNot** instruction is different from when the output of the previous **LD** instruction is FALSE.



● Set and Reset

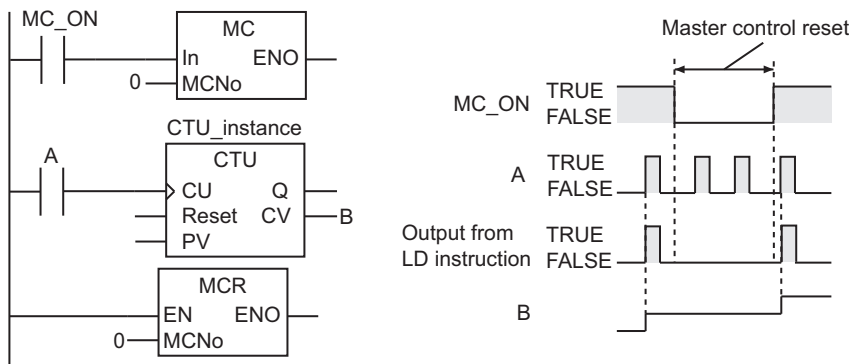
The previous value of the output is retained while the master control is reset.





● **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.

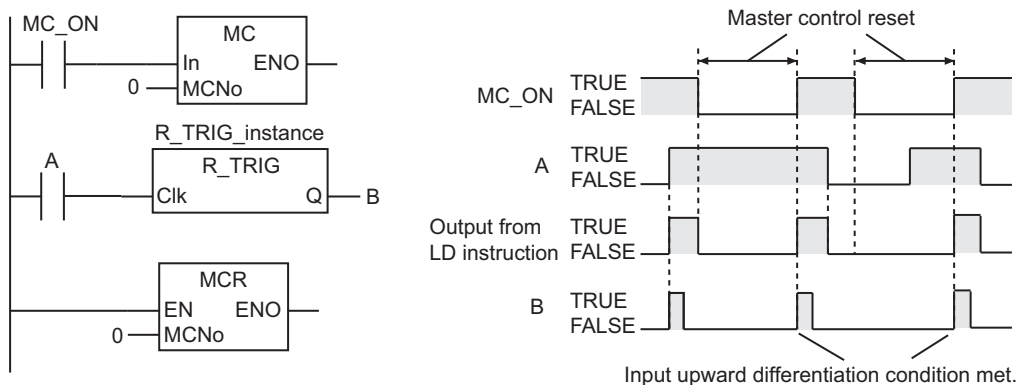
| Differentiation                | Instructions  |
|--------------------------------|---|
| Input upward differentiation   | <ul style="list-style-type: none"> <li>LD, LDN, AND, ANDN, OR, ORN, and Out with upward differentiation specifications</li> <li>R_TRIG (Up)</li> <li>Functions with an @ input upward differentiation option</li> <li>Functions blocks (e.g., counter instructions) with input upward differentiation specifications</li> </ul> |
| Input downward differentiation | <ul style="list-style-type: none"> <li>LD, LDN, AND, ANDN, OR, ORN, and Out with downward differentiation specifications</li> <li>F_TRIG (Down)</li> </ul>  |



When the master control is reset or the reset is cleared, the execution conditions for these POU's change. That means that the upward or downward differentiation conditions for these POU's may be met. If the upward or downward differentiation conditions are met, then the instructions are executed accordingly.

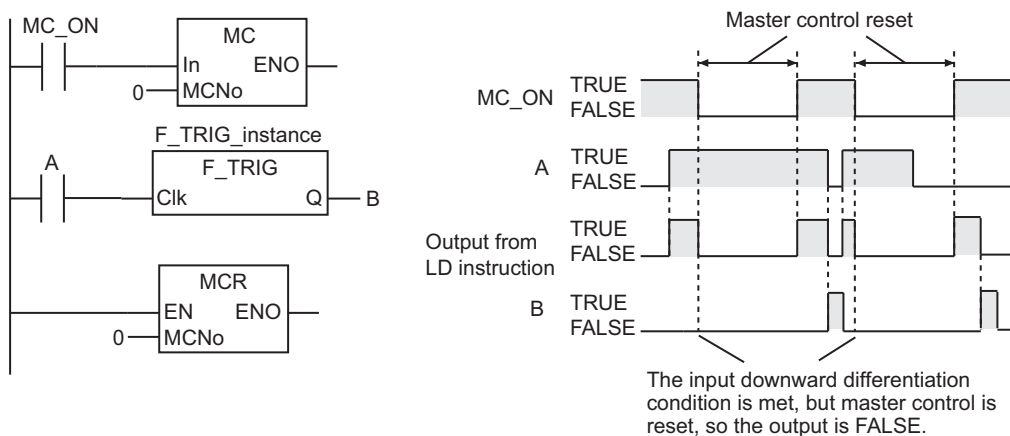
### ● R\_TRIG (Up)

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)

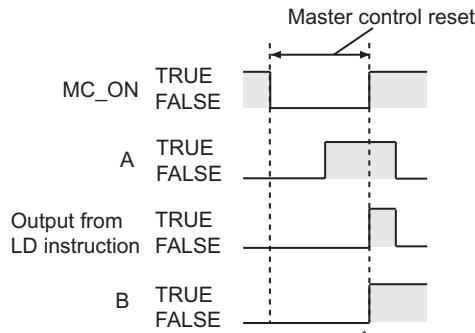
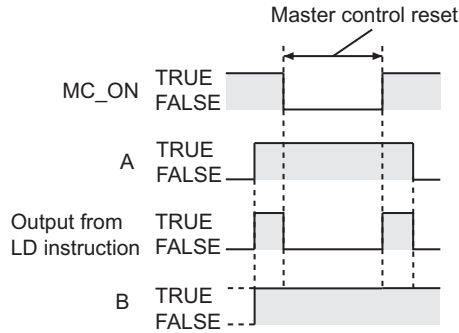
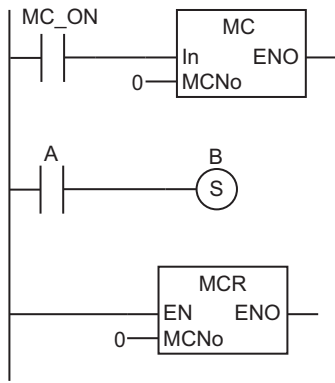
When the master control is reset, the execution condition changes to FALSE. If the previous execution condition was TRUE, then the input downward differentiation condition is met. However, the value of the output from the F\_TRIG (Down) instruction during the master control reset is forced to change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



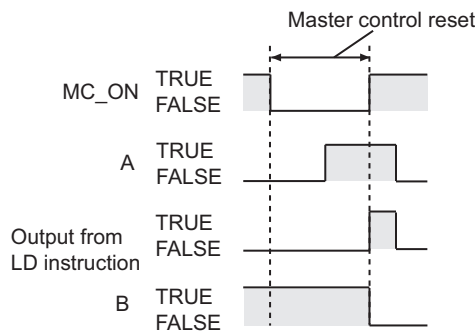
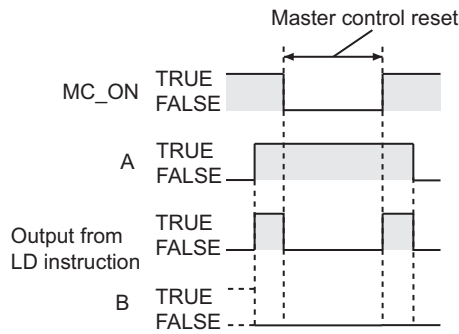
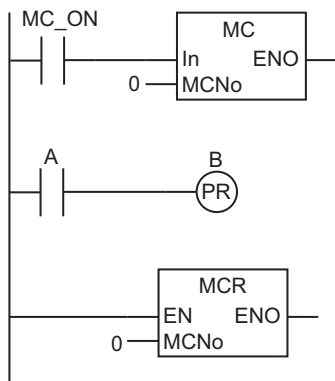
### ● Set and Reset with Input Upward Differentiation Specification

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.



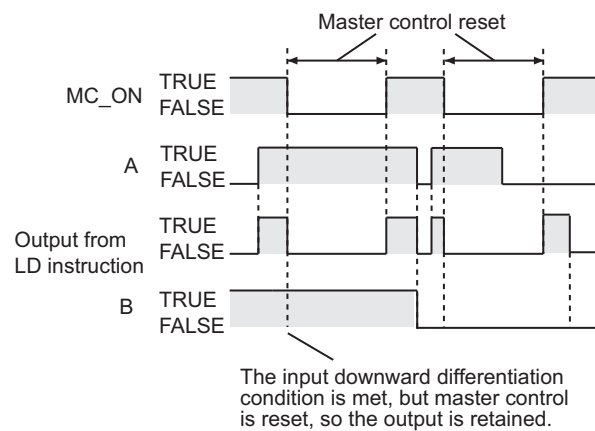
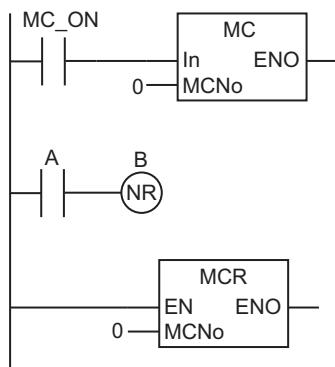
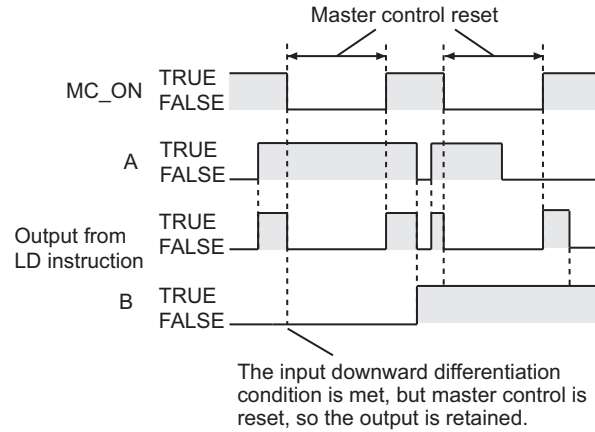
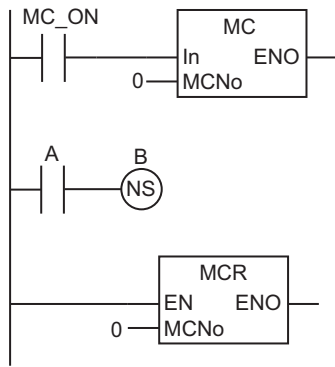
Here, the input upward differentiation condition is met and the output value changes to TRUE.



Here, the input upward differentiation condition is met and the output value changes to FALSE.

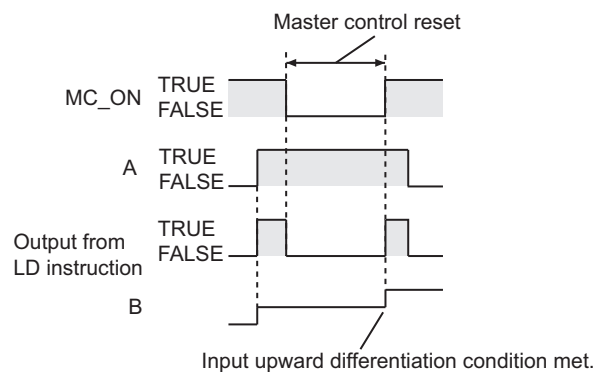
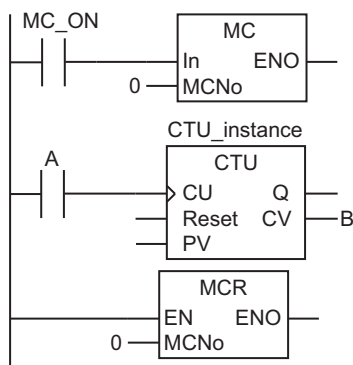
### ● Set and Reset with Input Downward Differentiation Specification

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.



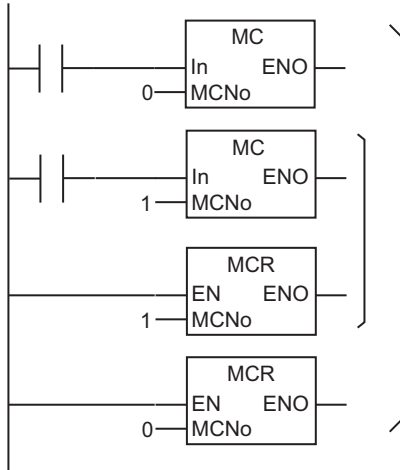
● CTU, CTD, and CTUD

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.



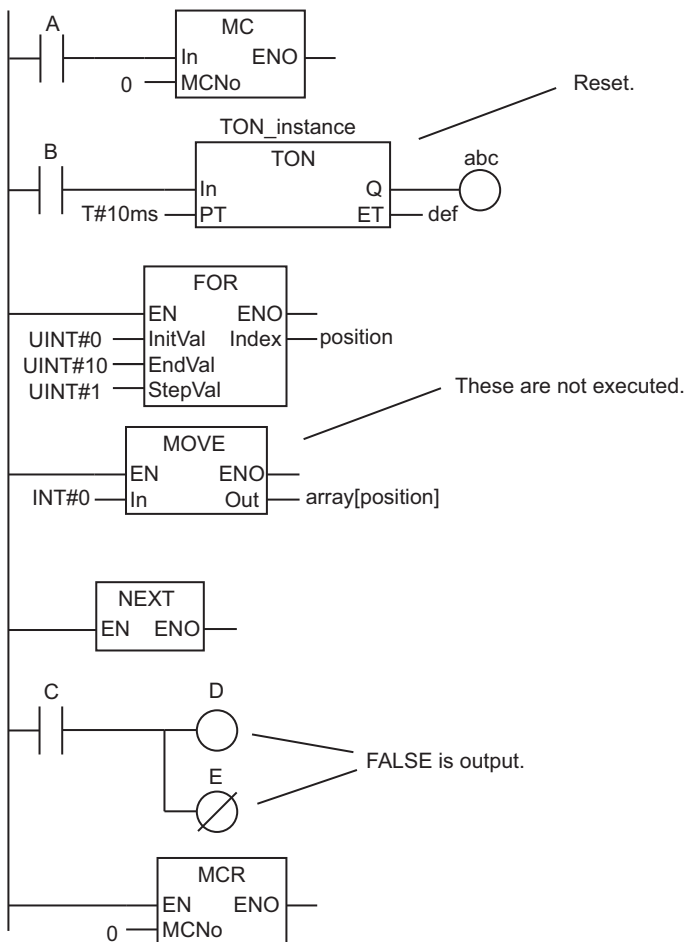
Always use the MC and MCR instructions as a pair in the same POU. The same value is used for master control number *MCNo* for both of the paired MC and MCR instructions. The user does not set the value of *MCNo*. It is automatically registered by the Sysmac Studio.

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



The following figure shows a programming example.

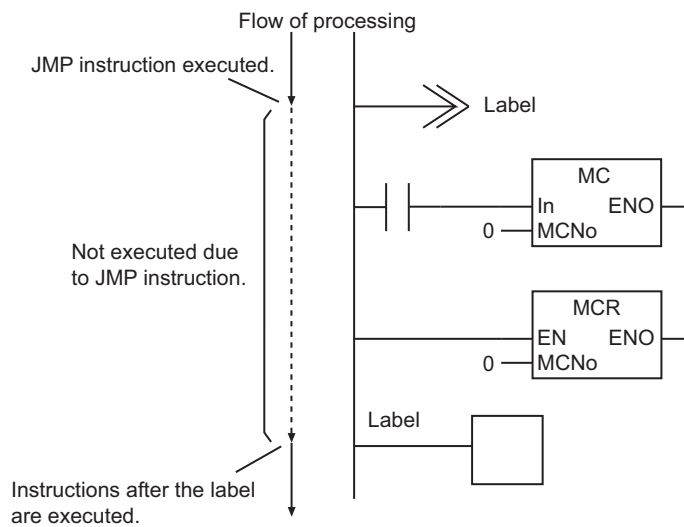
If the value of bit A is FALSE, the master control region is reset. While the master control region is in a reset state, the TON instruction is reset. The MOVE instruction is not executed. Also the Out instruction and OutNot instruction will output FALSE to bits D and E.



### Precautions for Correct Use

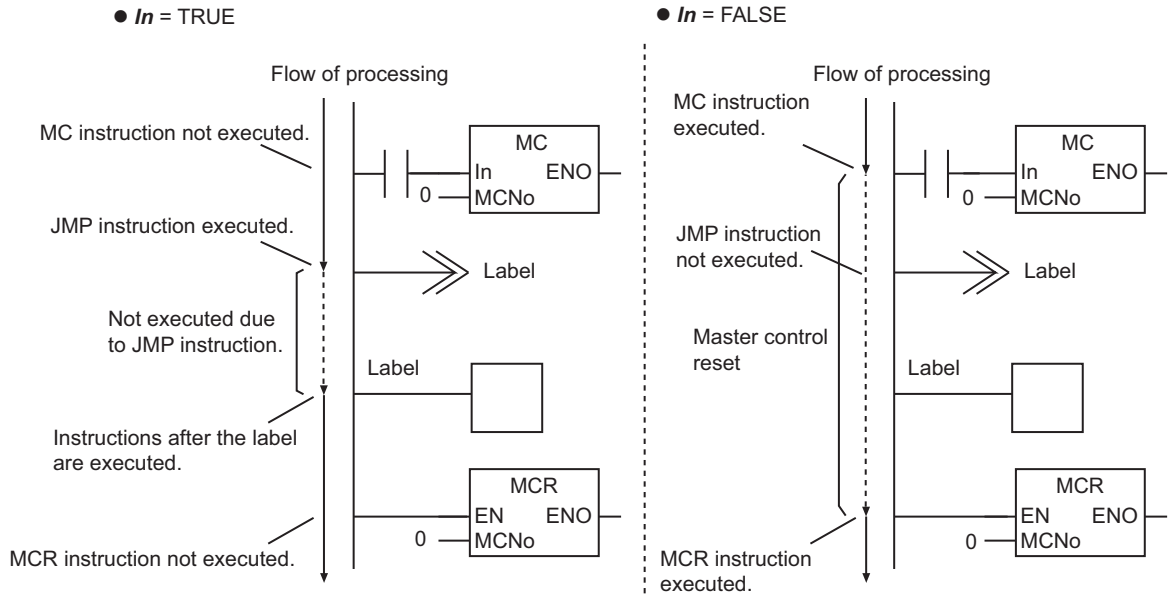
- These instructions must be used in a ladder diagram. They cannot be used in ST. They also cannot be used in inline ST in a ladder diagram.

- 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:
  - a) The following figure shows an MC-MCR pair inside a JMP-Label pair. Here, the jump is executed regardless of the value of *In*.

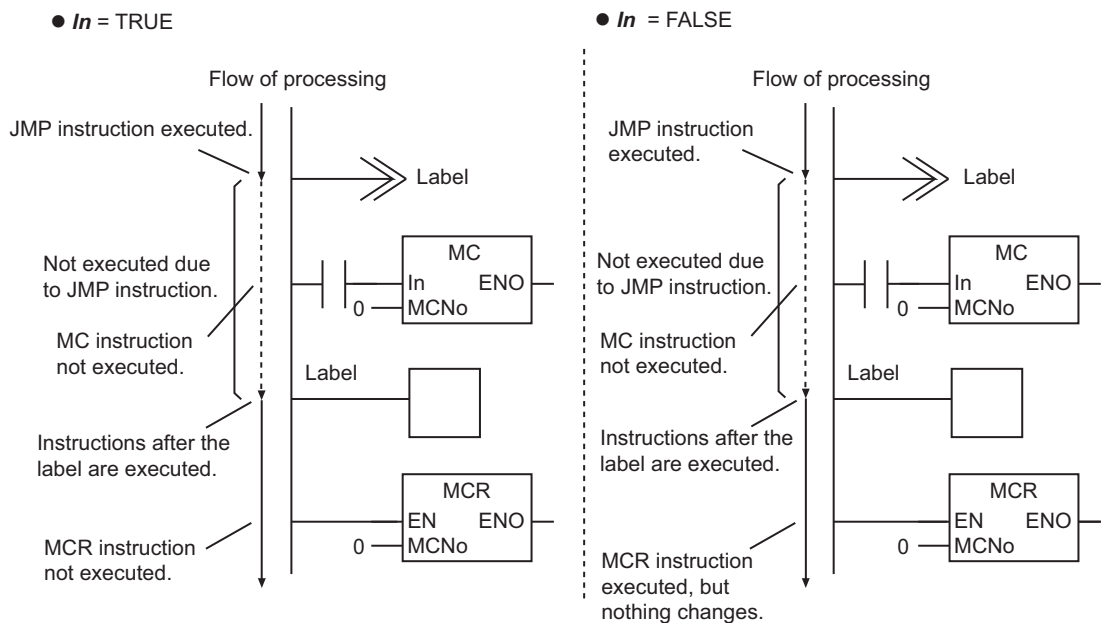


- b) The following figure shows a JMP-Label pair inside an MC-MCR pair. Here, operation is as given in the following table.

| Value of <i>In</i> | Operation  |
|--------------------|--|
| TRUE               | Master control region is not reset.<br>The jump is made. |
| FALSE              | Master control region is reset.<br>The jump is not made. |

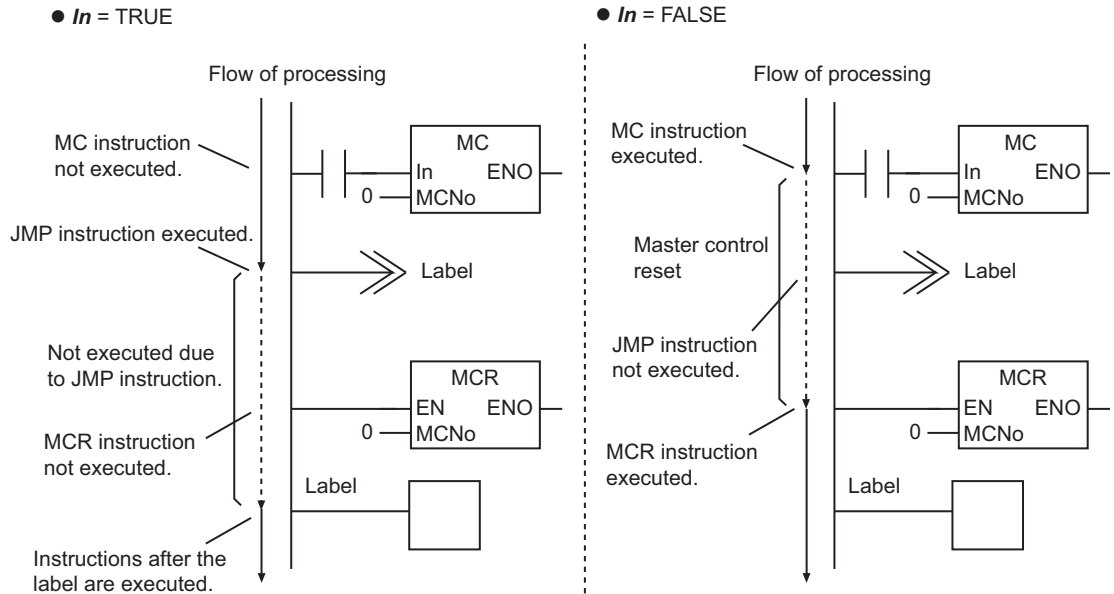


c) In the following figure, the JMP instruction, the MC instruction, a Label, and the MCR instruction are arranged in the stated order. First, the JMP instruction is executed. As a result, the MC instruction is not executed. Therefore, instructions can be executed after the Label. If the value of *In* is FALSE, the MCR instruction is executed, but nothing changes.



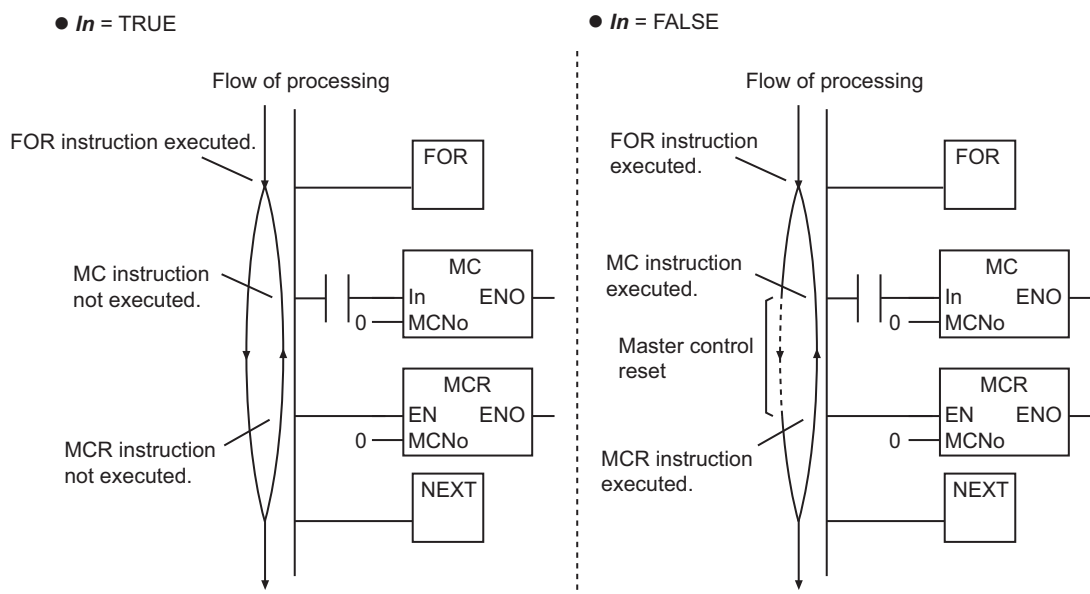
d) The instructions are in the following order in the following figure: MC instruction, JMP instruction, MCR instruction, and Label. Here, operation is as given in the following table.

| Value of <i>In</i> | Operation  |
|--------------------|--|
| TRUE               | Master control region is not reset.<br>The jump is made. |
| FALSE              | Master control region is reset.<br>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:
  - a) The following figure shows an MC-MCR pair inside a FOR-NEXT pair. Here, operation is as given in the following table.

| Value of <i>In</i> | Operation   |
|--------------------|---|
| TRUE               | Master control region is not reset.<br>The FOR loop is executed.  |
| FALSE              | Master control region is reset.<br>The FOR loop is executed, but the instructions between the MC and MCR instructions are not executed. |

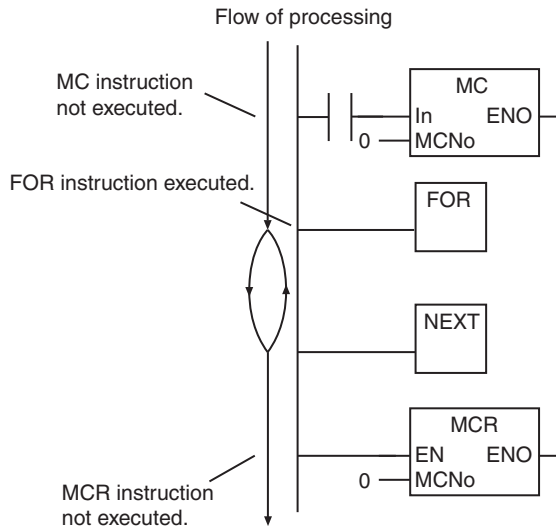


- b) The following figure shows a FOR-NEXT pair inside an MC-MCR pair. Here, operation is as given in the following table.

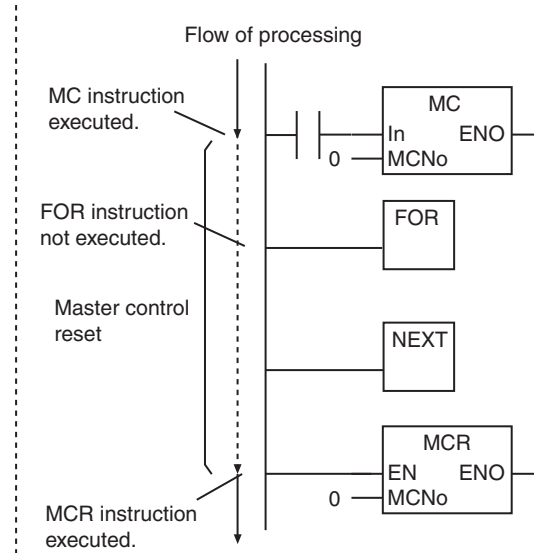
| Value of <i>In</i> | Operation  |
|--------------------|--|
| TRUE               | Master control region is not reset.<br>The FOR loop is executed. |

| Value of <i>In</i> | Operation  |
|--------------------|--|
| FALSE              | Master control region is reset.<br>The FOR loop is not executed. |

● *In* = TRUE



● *In* = FALSE




c) 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/<br>FUN | Graphic expression  | ST expression |
|-------------|------|------------|---|---------------|
| JMP         | Jump | FUN        |  Label | None          |

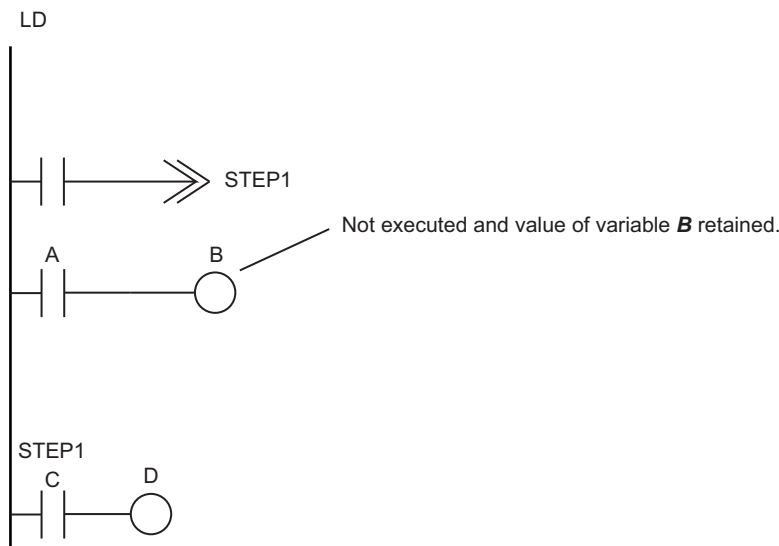
## Variables

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.



## 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<br>127 characters when converted to ACSII<br>31 characters when converted to Japanese characters (including single-byte kana)  |
| Character code          | UTF-8  |
| Applicable characters   | Not case sensitive.<br>English alphanumeric characters and other language characters.<br>Symbols: _ (underbar) and ~ (tilde)   |
| Prohibited text strings | <ul style="list-style-type: none"> <li>• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39)</li> <li>• A text string that consists of only a single _ (underbar) ASCII character</li> <li>• Any text string that includes two or more consecutive _ (underbar) ASCII characters</li> <li>• Any text string that starts with an _ (underbar) ASCII character</li> <li>• Any text string that ends with an _ (underbar) ASCII character</li> <li>• Any text string that starts with P_</li> </ul> |
| 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/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------|------------|--------------------|--|
| FOR         | Repeat Start | FUN        |                    | <pre>FOR Index:=InitVal TO EndVal BY StepVal DO   expression END_FOR*;</pre>   |
| NEXT        | Repeat End   | FUN        |                    | <p>* In ST, do not use <i>NEXT</i> to mark the ending position of repeat processing. Use <i>END_FOR</i> instead.</p> |

## Variables

|         | Meaning          | I/O    | Description   | Valid range                         | Unit | Default |
|---------|------------------|--------|---|-------------------------------------|------|---------|
| InitVal | Initial value    | Input  | Value to set the <i>Index</i> to when repetition is started.  | Depends on data type.* <sup>1</sup> | ---  | *2      |
| EndVal  | End value        |        | Value of <i>Index</i> where repetition is stopped             |                                     |      |         |
| StepVal | Increment        |        | Value to add to <i>Index</i> each time processing is repeated | Depends on data type.* <sup>3</sup> |      | *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.

|         | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.* <sup>1</sup> |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| 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.                            |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

|   | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   |      |  |      |      |     |    |        |
| An array element or structure member can also be specified. |             |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

\*1. 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).

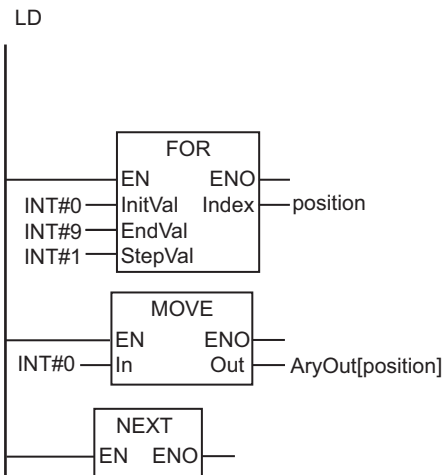
| Programming language | Conditions to start repeat processing    |
|----------------------|--|
| Ladder diagram       | $StepVal \geq 0$ and $Index \leq EndVal$ |
| ST                   | $StepVal \geq 0$ and $Index \leq EndVal$ |
|                      | $StepVal < 0$ and $Index \geq EndVal$    |

- 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       | $Index \leq EndVal$                                 |
| ST                   | If $StepVal \geq 0$ , $Index$ must be $\leq EndVal$ |
|                      | If $StepVal < 0$ , $Index$ must be $\geq EndVal$    |

- 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.

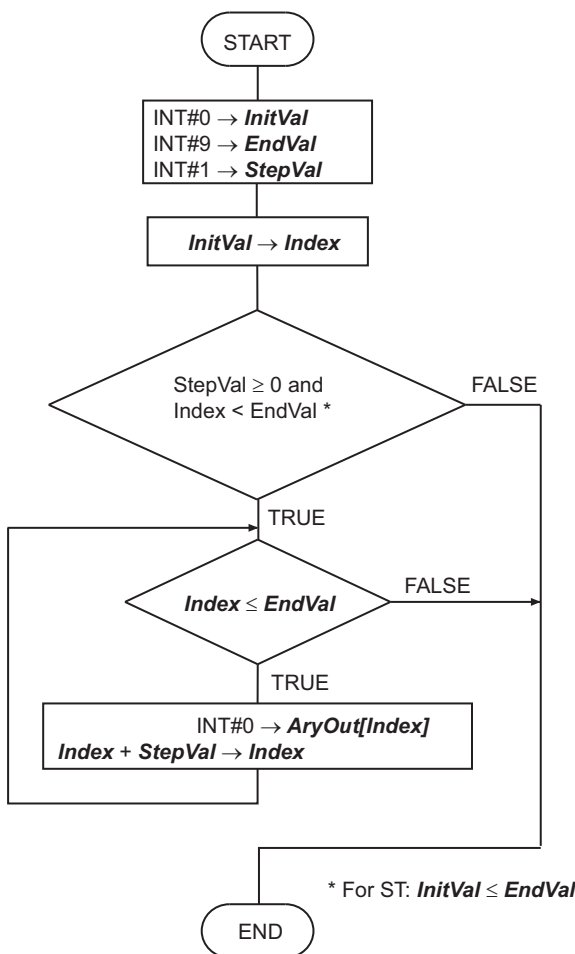
The following example is for when *InitVal* is INT#0, *EndVal* is INT#9, and *StepVal* is INT#1. The MOVE instruction is executed 10 times and INT#0 is assigned to array variables AryOut[0] to AryOut[9].



ST

```

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



INT#0 is assigned in order to **AryOut[0]** to **AryOut[9]**.

|           |       |
|-----------|-------|
| AryOut[0] | INT#0 |
| AryOut[1] | INT#0 |
| AryOut[2] | INT#0 |
| AryOut[3] | INT#0 |
| AryOut[4] | INT#0 |
| AryOut[5] | INT#0 |
| AryOut[6] | INT#0 |
| AryOut[7] | INT#0 |
| AryOut[8] | INT#0 |
| AryOut[9] | INT#0 |

## 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 shows an example with *EndVal* for the function and *StepVal* for the expression.

```
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;
```



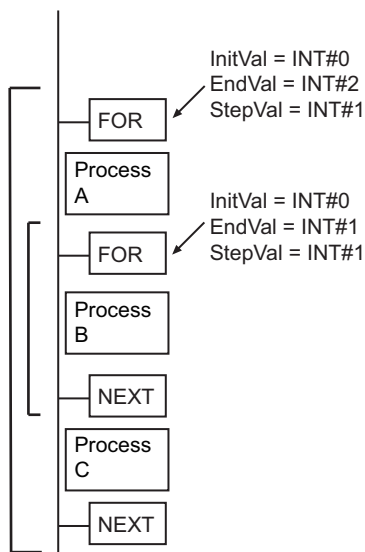
### Version Information

SyMac Studio version 1.08 or higher is required to use expressions for *EndVal* and *StepVal*. You can use an expression for *InitVal* even with SyMac Studio version 1.07 or lower.

## Additional Information

- Execute the BREAK instruction (or the 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.

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 <i>StepVal</i> | Values of <i>InitVal</i> and <i>EndVal</i> | 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.<br>An error does not occur.  |
|                      | StepVal < 0             | InitVal < EndVal                           | The processing between the FOR and NEXT instructions is executed an indeterminate number of times.<br>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.<br>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.<br>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.<br>An error does not occur.   |
|                      | StepVal < 0             | InitVal < EndVal                           | The processing between the FOR and END_FOR instructions is not executed even once.<br>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.<br>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 the BREAK instruction (or the 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.

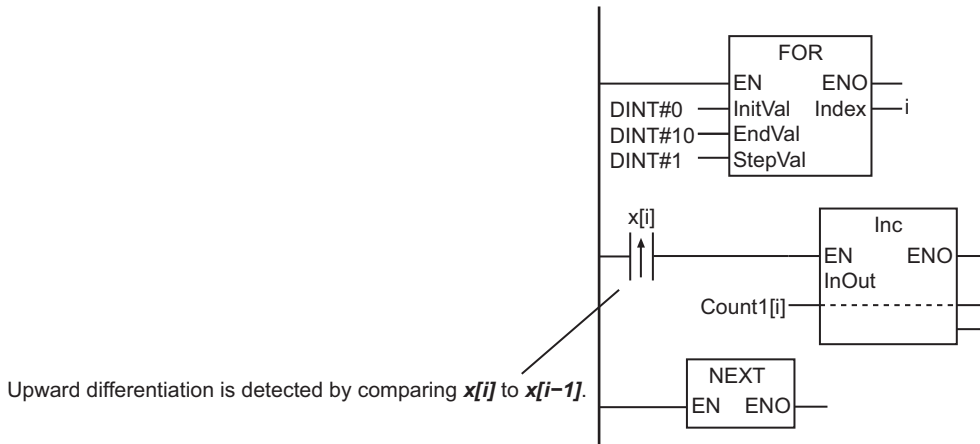
ST : The value of *Index* will be 101 after 100 repetitions.

- 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  $\text{Count1}[i]$ .

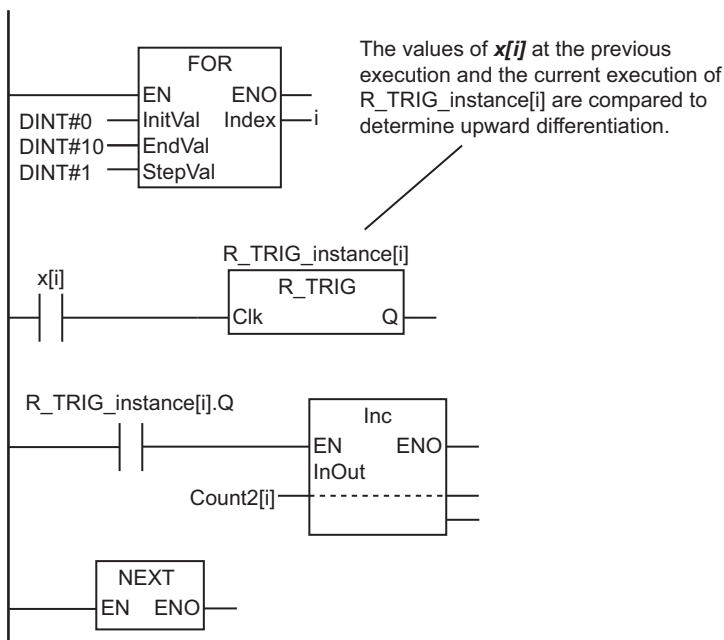
| Value of $x[i-1]$ | Value of $x[i]$ | Increment processing for $\text{Count1}[i]$ |
|-------------------|-----------------|---|
| TRUE              | TRUE            | Not executed.                               |
| TRUE              | FALSE           | Not executed.                               |
| FALSE             | TRUE            | Executed.                                   |
| FALSE             | FALSE           | Not executed.                               |





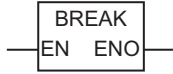
- In the following programming, upward differentiation of  $x[i]$  is detected by the R\_TRIG instruction. An instance of the R\_TRIG instruction is provided for each element of  $x[i]$ , so it is possible to detect the elements of  $x[i]$  for which there was upward differentiation. The following table 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.                      |



# BREAK

The BREAK instruction cancels repeat processing from the innermost FOR instruction to the NEXT instruction.

| Instruction | Name       | FB/<br>FUN | Graphic expression  | ST expression |
|-------------|------------|------------|---|---------------|
| BREAK       | Break Loop | FUN        |  | None          |

## Variables

None

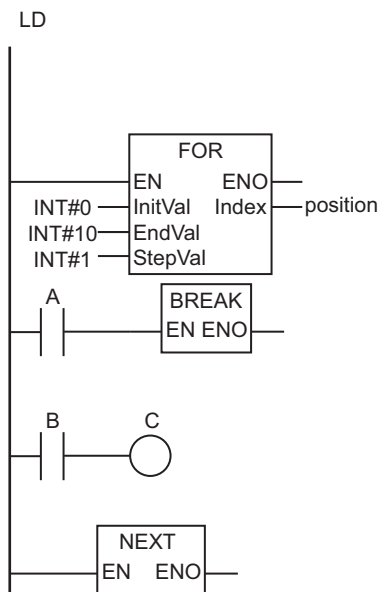
## Function

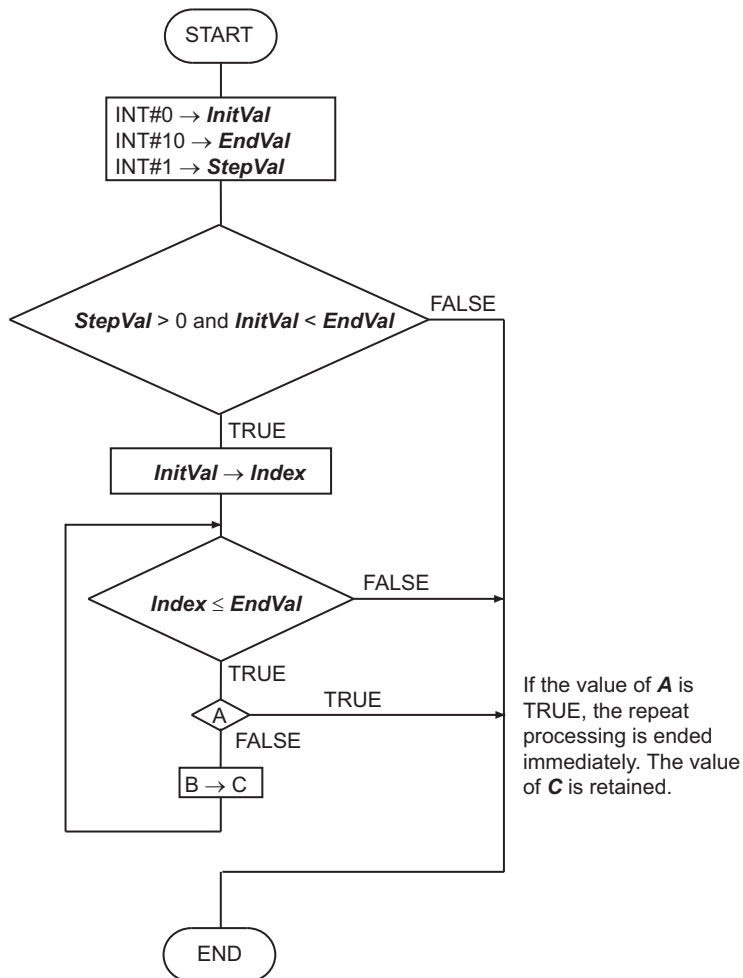
The BREAK instruction cancels the repeat processing from the lowest level FOR instruction to the NEXT instruction. It moves processing to the next instruction after the NEXT instruction.

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.





## 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 the BREAK instruction to cancel repeat processing.



# Comparison Instructions

| Instruction                                    | Name  | Page       |
|--|---|------------|
| EQ (=)   | Equal   | page 2-102 |
| NE (<>)  | Not Equal   | page 2-105 |
| LT (<), LE (<=), GT (>), and GE (>=)           | Less Than/Less Than Or Equal/Greater Than/Greater Than Or Equal   | page 2-108 |
| EQascii  | Text String Comparison Equal  | page 2-111 |
| NEascii  | Text String Comparison Not Equal  | page 2-113 |
| 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 | page 2-115 |
| Cmp  | Compare   | page 2-118 |
| ZoneCmp  | Zone Comparison   | page 2-120 |
| TableCmp                                       | Table Comparison  | page 2-122 |
| AryCmpEQ and AryCmpNE                          | Array Comparison Equal/ Array Comparison Not Equal  | page 2-125 |
| 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                        | page 2-127 |
| AryCmpEQV and AryCmpNEV                        | Array Value Comparison Equal/Array Value Comparison Not Equal   | page 2-130 |
| 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 | page 2-132 |

# EQ (=)

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

| Instruction | Name  | FB/<br>FUN | Graphic expression | ST expression                                   |
|-------------|-------|------------|--------------------|---|
| EQ(=)       | Equal | FUN        |                    | Out:=(In1=In2) & (In2=In3) & ... & (InN-1=InN); |

## Variables

|            | Meaning           | I/O    | Description                     | Valid range           | Unit | Default |
|------------|-------------------|--------|---------------------------------|-----------------------|------|---------|
| In1 to InN | Comparison data   | Input  | Values to compare<br>N = 2 to 5 | Depends on data type. | ---  | 0*1     |
| Out        | Comparison result | Output | Comparison result               | Depends on data type. | ---  | ---     |

- \*1. 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.

|            | Boo<br>lean                         | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
|            | Enumerations can also be specified. |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |          |          |          |          |          |
| Out        | OK                                  |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |          |          |          |          |          |

- \*1. 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.

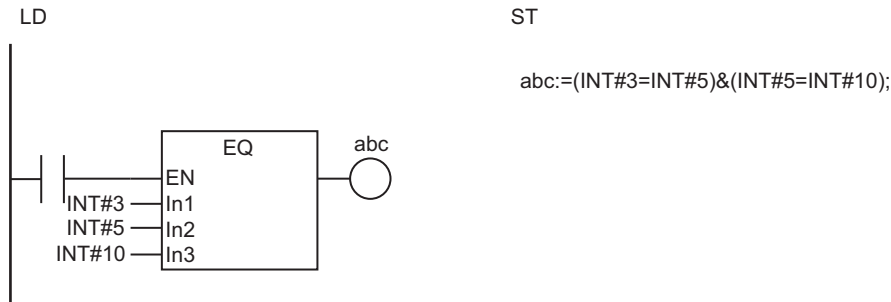
## Function

The EQ (=) instruction determines if the contents of from two to five variables *In1* to *InN* are all equivalent.

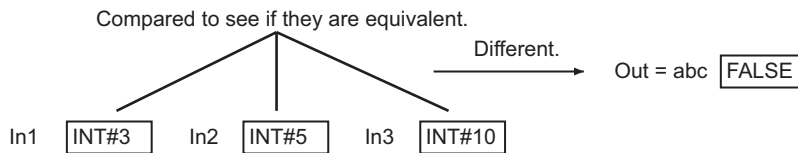
The comparison result *Out* is TRUE only when all values are equivalent. Otherwise, the value of *Out* is FALSE.

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.



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 accuracy of their values so that the comparison can be based on the same accuracy. Use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## 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 (SINT, INT, DINT, LINT, USINT, UINT, UDINT, and ULINT). You cannot compare bit string data to 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 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 values of two variables are not equivalent.

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

## Variables

|             | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-------------|-------------------|--------|-------------------|-----------------------|------|---------|
| In1 and In2 | Comparison data   | Input  | Values to compare | Depends on data type. | ---  | *1      |
| Out         | Comparison result | Output | Comparison result | Depends on data type. | ---  | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

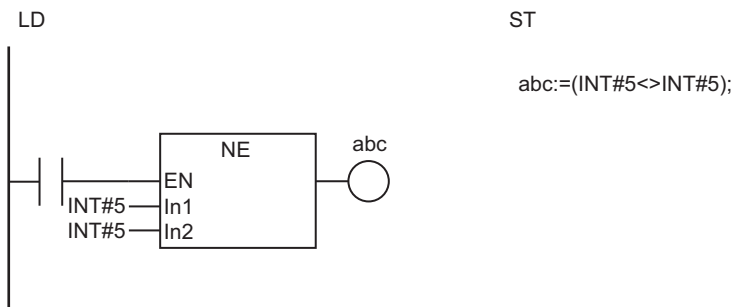
|             | Boo<br>lean                         | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |    |
|-------------|-------------------------------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|---------------------|------|--|------|------|-----|----|--------|----|
|             | 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     | OK |
|             | Enumerations can also be specified. |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |    |
| Out         | OK                                  |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |    |

\*1. 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.

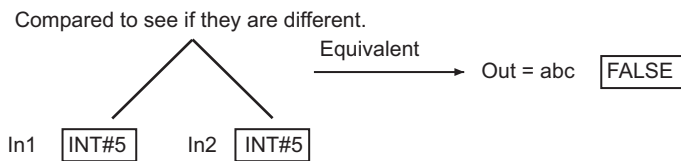
## Function

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, it 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 accuracy of their values so that the comparison can be based on the same accuracy. Use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## 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/<br>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

|            | Meaning           | I/O    | Description                     | Valid range           | Unit | Default |
|------------|-------------------|--------|---------------------------------|-----------------------|------|---------|
| In1 to InN | Comparison data   | Input  | Values to compare<br>N = 2 to 5 | Depends on data type. | ---  | *1      |
| Out        | Comparison result | Output | Comparison result               | Depends on data type. | ---  | ---     |

- \*1. 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.

|            | Boo<br>lean | Bit strings |          |          |          |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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<br>*1    | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK                   | OK    | OK<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
| Out        | OK          |             |          |          |          |       |          |       |       |      |     |      |      |                      |       |  |          |          |          |          |

- \*1. 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

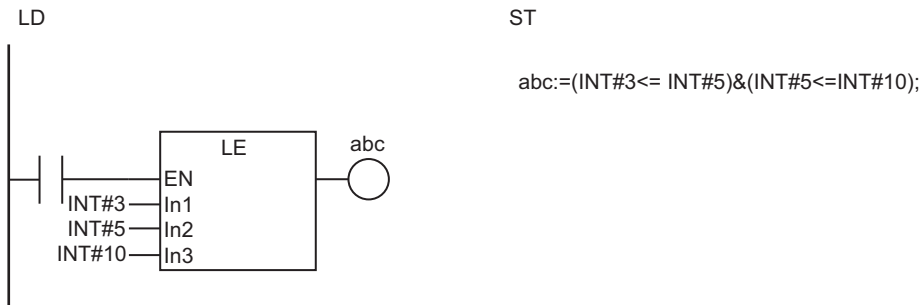
These instructions compare the data in *In1* to *InN* (N = 2 to 5).  
The output value *Out* is shown below for each instruction.

| Instruction | Value of <i>Out</i>   |
|-------------|---|
| LT (<)      | If $In1 < In2 < \dots < InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE.          |
| LE (<=)     | If $In1 \leq In2 \leq \dots \leq InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE. |
| GT (>)      | If $In1 > In2 > \dots > InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE.          |
| GE (>=)     | If $In1 \geq In2 \geq \dots \geq InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE. |

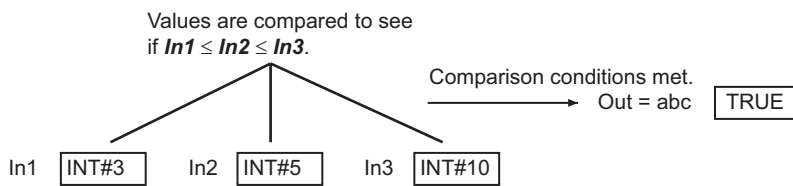
The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

| 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 instructions, <i>LTascii</i> , <i>LEascii</i> , <i>GTascii</i> , and <i>GEascii</i> on page 2-115.<br>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.



The LE instruction determines if  $In1 \leq In2 \leq In3$ .  
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 accuracy of their values so that the comparison can be based on the same accuracy. You can use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## Precautions for Correct Use

- If the data types of  $In1$  to  $InN$  are different, they will be cast to a data type which can accommodate every possible value in all the types before comparison.
- 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  $InN$  are real numbers and include any non-terminating decimal numbers, error may cause unexpected processing results.
- 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 when 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/<br>FUN | Graphic expression  | ST expression               |
|-------------|------------------------------------|------------|---|-----------------------------|
| EQascii     | Text String<br>Comparison<br>Equal | FUN        | <pre> graph LR     subgraph EQascii ["(@)EQascii"]         EN[EN]         In1[In1]         colon[:]         InN[InN]     end     EQascii --&gt; Out[Out] </pre> | Out:=EQascii(In1, ··, InN); |

## Variables

|            | Meaning                 | I/O    | Description                           | Valid range           | Unit | Default |
|------------|-------------------------|--------|---------------------------------------|-----------------------|------|---------|
| In1 to InN | Comparison text strings | Input  | Text strings to compare<br>N = 2 to 5 | Depends on data type. | ---  | "*1     |
| Out        | Comparison result       | Output | Comparison result                     | Depends on data type. | ---  | ---     |

- \*1. 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.

|            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

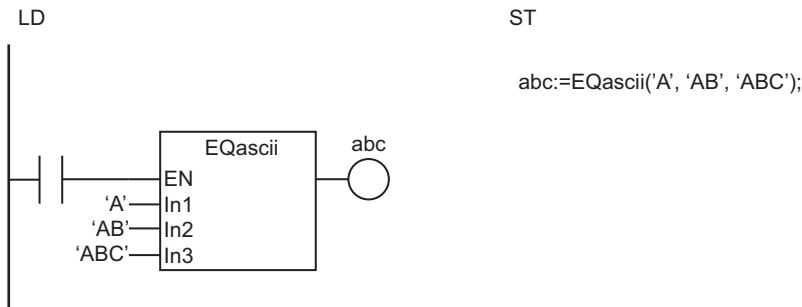
## Function

The EQascii instruction determines if two to five text strings *In1* to *InN* are all equivalent.

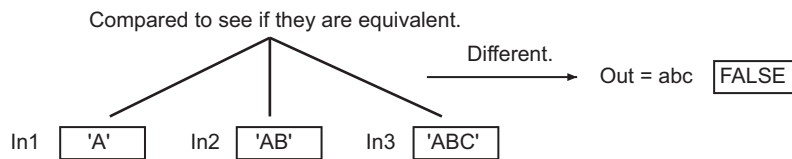
If they are all equivalent, comparison result *Out* changes to TRUE. Otherwise, the value of *Out* is FALSE.

"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.



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



### 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 instruction, *EQ (=)* on page 2-102, can also be used to compare text strings. The specifications of the EQ (=) instruction for comparing text strings are the same as those of the EQascii instruction.

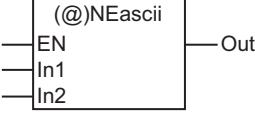
### Precautions for Correct Use

- 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 when 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/<br>FUN | Graphic expression  | ST expression           |
|-------------|--|------------|---|-------------------------|
| NEascii     | Text String<br>Comparison<br>Not Equal | FUN        |  | Out:=NEascii(In1, In2); |

## Variables

|             | Meaning                 | I/O    | Description             | Valid range           | Unit | Default |
|-------------|-------------------------|--------|-------------------------|-----------------------|------|---------|
| In1 and In2 | Comparison text strings | Input  | Text strings to compare | Depends on data type. | ---  | *1      |
| Out         | Comparison result       | Output | Comparison result       | Depends on data type. | ---  | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|             | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

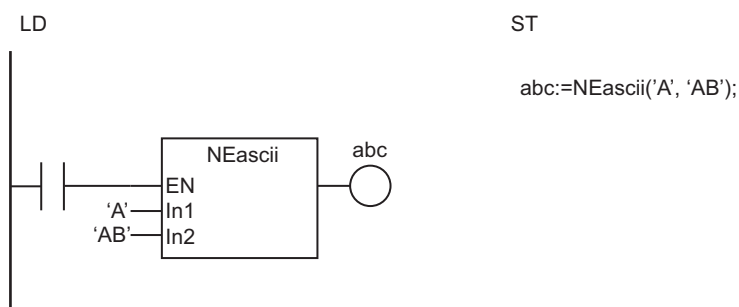
## Function

The NEascii instruction determines if two text strings *In1* and *In2* are not equivalent.

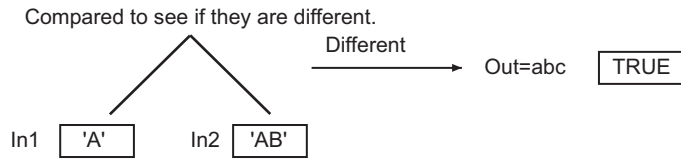
Comparison result *Out* will be TRUE if they are different, and will be FALSE if they are the same.

"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.



The NEascii instruction determines if *In1* and *In2* are different. If they are different, 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 instruction, *NE (<>)* on page 2-105, can also be used to compare text strings. The specifications of the *NE (<>)* instruction for comparing text strings are the same as those of 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 when 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/<br>FUN | Graphic expression | ST expression                |
|-------------|---|------------|--------------------|------------------------------|
| LTascii     | Text String<br>Comparison<br>Less Than                | FUN        |                    | Out:=LTascii(In1, ..., InN); |
| LEascii     | Text String<br>Comparison<br>Less Than or<br>Equal    | FUN        |                    | Out:=LEascii(In1, ..., InN); |
| GTascii     | Text String<br>Comparison<br>Greater Than             | FUN        |                    | Out:=GTascii(In1, ..., InN); |
| GEascii     | Text String<br>Comparison<br>Greater Than or<br>Equal | FUN        |                    | Out:=GEascii(In1, ..., InN); |

## Variables

|            | Meaning                 | I/O    | Description                           | Valid range           | Unit | Default |
|------------|-------------------------|--------|---------------------------------------|-----------------------|------|---------|
| In1 to InN | Comparison text strings | Input  | Text strings to compare<br>N = 2 to 5 | Depends on data type. | ---  | "*1     |
| Out        | Comparison result       | Output | Comparison result                     | Depends on data type. | ---  | ---     |

- \*1. 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.

|            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 $In1 < In2 < \dots < InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE.  |
| LEascii     | If $In1 \leq In2 \leq \dots \leq InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE.   |
| GTascii     | If $In1 > In2 > \dots > InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE.  |
| GEascii     | If $In1 \geq In2 \geq \dots \geq InN$ , <i>Out</i> is TRUE. Otherwise, it is FALSE. The sizes of the character codes are compared. The comparison procedure |

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.

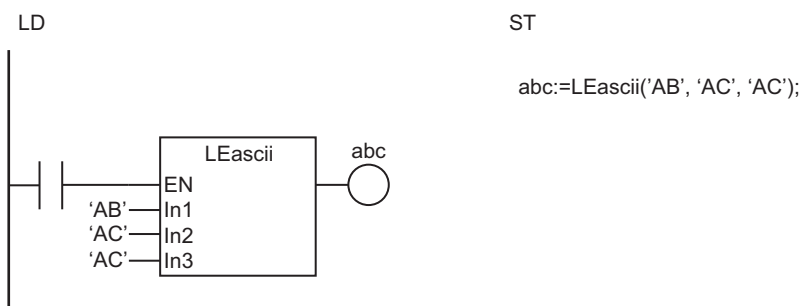
If the lengths of the text strings are different, NULL characters (16#00) are added to the shorter text string to complete the comparison.

The relationships between various text strings are as follows:

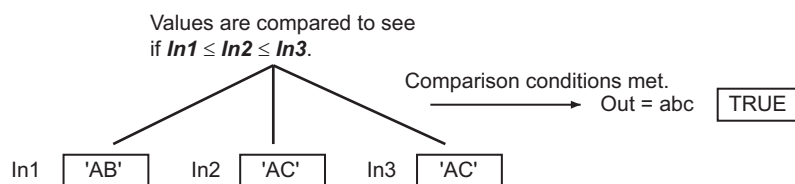
- 'AD'(16#414400) < 'BC'(16#424400)
- 'ADC' (16#41444300) < 'B' (16#42000000)
- 'ABC' (16#41424300) < 'ABD' (16#41424400)
- 'ABC' (16#41424300) > 'AB' (16#41420000)
- 'AB' (16#414200) = 'AB' (16#414200)

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.



The LEascii instruction determines if  $In1 \leq In2 \leq In3$ .  
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 instructions, *LT* (<), *LE* (<=), *GT* (>), and *GE* (>=) on page 2-108, 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 those of 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 when 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/<br>FUN | Graphic expression | ST expression   |
|-------------|---------|------------|--------------------|---|
| Cmp         | Compare | FUN        |                    | <pre>Out:=Cmp(In1, In2, OutEQ, OutGT, OutGE, OutNE, OutLT, OutLE);</pre> <p>You can omit Out.</p> |

## Variables

|             | Meaning                    | I/O    | Description                | Valid range           | Unit | Default |
|-------------|----------------------------|--------|----------------------------|-----------------------|------|---------|
| In1 and In2 | Comparison data            | Input  | Values to compare          | Depends on data type. | ---  | *1      |
| 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    |                       |      |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|             | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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   |  |      |      |     |    |        |
| Out         | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutEQ       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutGT       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutGE       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutNE       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutLT       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OutLE       | OK          |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

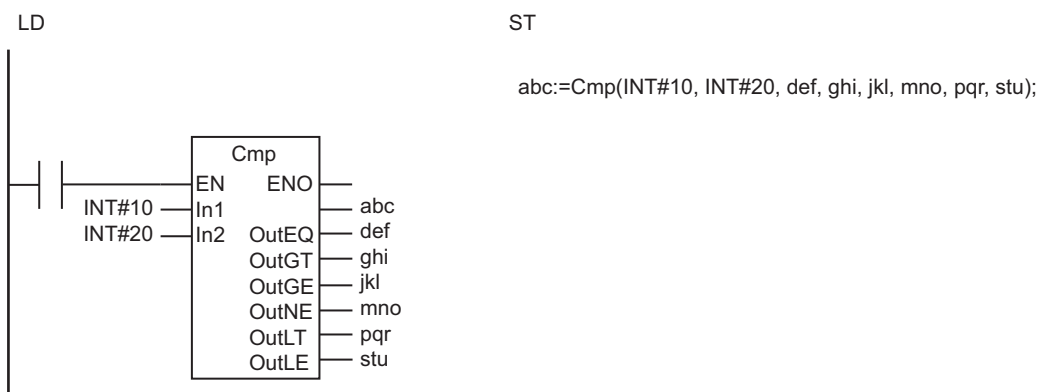
## Function

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

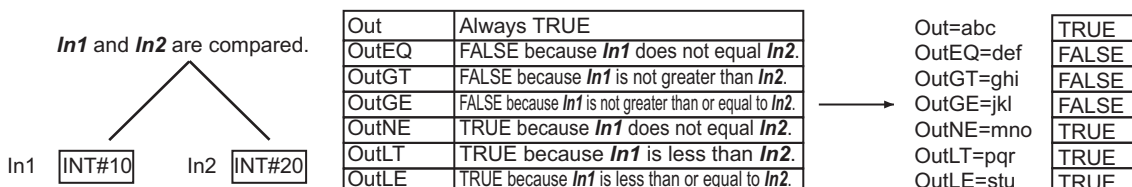
The values of the flags are as follows:

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



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, one will be cast to the other data type which can accommodate every possible value in both of the data types before comparison.
- If *In1* and *In2* are real numbers and include any non-terminating decimal numbers, error may cause unexpected processing results.
- 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 between the specified maximum and minimum values.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|-----------------|------------|--------------------|---------------------------|
| ZoneCmp     | Zone Comparison | FUN        |                    | Out:=ZoneCmp(MN, In, MX); |

## Variables

|     | 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  |                       |      | *1      |
| MX  | Maximum value     |        | Maximum value     |                       |      | 0       |
| Out | Comparison result | Output | Comparison result | Depends on data type. | ---  | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 |        |
| In  |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK   | OK                   | OK    | OK<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 |        |
| MX  |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK   | OK                   | OK    | OK<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 |        |
| 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.

## Function

The ZoneCmp instruction determines if comparison data *In* is between maximum value *MX* and minimum value *MN*.

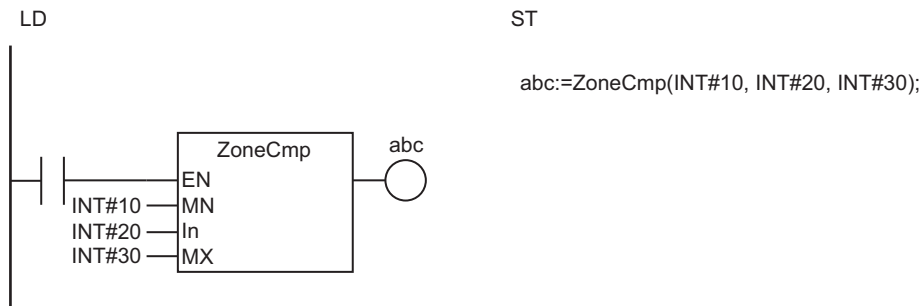
If  $MX \geq In \geq MN$ , *Out* will be TRUE. Otherwise, it will be FALSE.

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



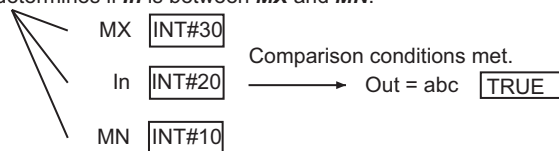
| Data type        | Relationship   |
|------------------|--|
| TIME             | The numerically larger value is considered to be larger. |
| DATE, TOD, or DT | Later dates or times of day are considered to be larger. |

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.



The ZoneCmp instruction determines if  $MX \geq In \geq MN$ .  
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 accuracy of their values so that the comparison can be based on the same accuracy. You can use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## Precautions for Correct Use

- If the data types of *In*, *MX*, and *MN* are different, they will be cast to a data type which can accommodate every possible value in all the types before comparison.
- If *In*, *MX*, and *MN* are real numbers and include any non-terminating decimal numbers, error may cause unexpected processing results.
- 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 when an error occurs in the previous instruction on the rung.
- An error occurs in the following cases. *Out* will be FALSE.
  - a) The value of *MN* is greater than the value of *MX*.
  - b) 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/<br>FUN | Graphic expression | ST expression                           |
|-------------|------------------|------------|--------------------|---|
| TableCmp    | Table Comparison | FUN        |                    | Out:=TableCmp(In, Table, Size, AryOut); |

## Variables

|                                    | Meaning                             | I/O    | Description   | Valid range           | Unit | Default |
|------------------------------------|-------------------------------------|--------|---|-----------------------|------|---------|
| In                                 | Comparison data                     | Input  | Value to compare  | Depends on data type. | ---  | *1      |
| Table[]<br>(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 <i>In</i>   |                       |      |         |
| AryOut[] (array)                   | Individual comparison results array | In-out | Comparison results for Table[] elements<br>TRUE: Condition met.<br>FALSE: Condition not met.  | Depends on data type. | ---  | ---     |
| Out                                | Comparison result                   | Output | TRUE: <i>In</i> meets all comparison conditions for elements of Table[]<br>FALSE: The comparison condition is not met for one or more sets of elements. | Depends on data type. | ---  | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

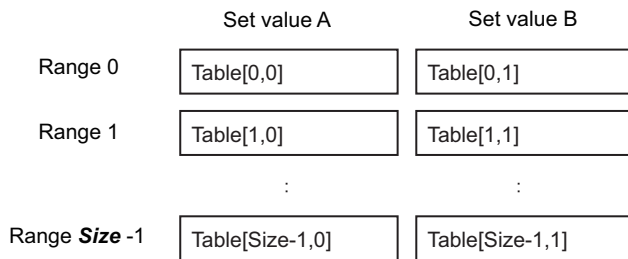
|                                 | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|---------------------------------|---|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|---------------------|-------|--|------|-----|----|--------|
|                                 | 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                  |       |  |      |     |    |        |
| Table[] (two-dimensional array) | Must be a two-dimensional array with elements that have the same data type as <i>In</i> . |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Size                            |   |             |      |       |       | OK       |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |

|                       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----------------------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|-------------------|-------|--|------|-----|----|--------|
|                       | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL              | LREAL | TIME   | DATE | TOD | DT | STRING |
| AryOut[] (ar-<br>ray) | OK          |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| Out                   | OK          |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

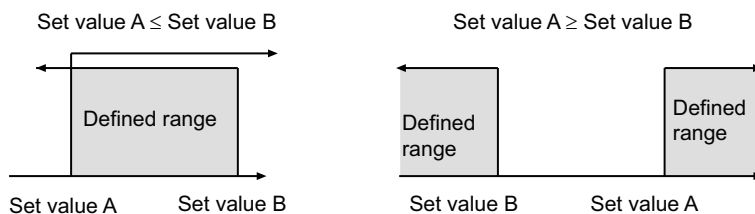
## Function

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.

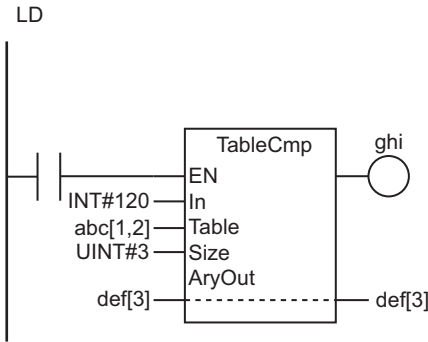


Set value A and set value B define range as shown below. Set value A and set value B are always included in the range.



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.



ST

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

In=INT#120

|             |   |   |
|-------------|---|---|
| Size=UINT#3 | { | Table[0,0]=abc[1,2] <span style="border: 1px solid black; padding: 2px;">0</span> Table[0,1]=abc[1,3] <span style="border: 1px solid black; padding: 2px;">99</span> →    AryOut[0]=def[3] <span style="border: 1px solid black; padding: 2px;">FALSE</span>    |
|             |   | Table[1,0]=abc[2,2] <span style="border: 1px solid black; padding: 2px;">100</span> Table[1,1]=abc[2,3] <span style="border: 1px solid black; padding: 2px;">199</span> →    AryOut[1]=def[4] <span style="border: 1px solid black; padding: 2px;">TRUE</span>  |
|             |   | Table[2,0]=abc[3,2] <span style="border: 1px solid black; padding: 2px;">200</span> Table[2,1]=abc[3,3] <span style="border: 1px solid black; padding: 2px;">299</span> →    AryOut[2]=def[5] <span style="border: 1px solid black; padding: 2px;">FALSE</span> |
|             |   | Out=ghi <span style="border: 1px solid black; padding: 2px;">FALSE</span>   |

## 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.
  - a) If the value of *Size* exceeds the size of the *AryOut*[] array.
  - b) If the value of *Size* exceeds the size of the first dimension of the *Table*[] array.
  - c) The size of the second dimension of *Table*[] is 1.

# AryCmpEQ and AryCmpNE

These instructions compare the corresponding elements of two arrays.

AryCmpEQ : Determines if the corresponding elements of two arrays are equal.

AryCmpNE : Determines if the corresponding elements of two arrays are not equal.

| Instruction | Name                       | FB/<br>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

|                          | 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. | ---  | *1      |
| Size                     | Number of comparison elements |        | Number of elements to compare             |                       |      | 1       |
| AryOut[] (array)         | Comparison results array      | In-out | Comparison results array                  | 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.

|                  | Boo<br>lean  | Bit strings |          |           |           |           | Integers |           |           |          |         |          |          | Real<br>num<br>bers |           | Times, durations,<br>dates, and text strings |          |         |    |            |
|------------------|--|-------------|----------|-----------|-----------|-----------|----------|-----------|-----------|----------|---------|----------|----------|---------------------|-----------|--|----------|---------|----|------------|
|                  | BOO<br>L   | BY<br>TE    | WO<br>RD | DWO<br>RD | LWO<br>RD | US<br>INT | UI<br>NT | UD<br>INT | UL<br>INT | SI<br>NT | IN<br>T | DI<br>NT | LI<br>NT | RE<br>AL            | LR<br>EAL | TI<br>ME                                     | DA<br>TE | TO<br>D | DT | STR<br>ING |
| In1[] (array)    | 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   |             |          |           |           |           |          |           |           |          |         |          |          |                     |           |  |          |         |    |            |

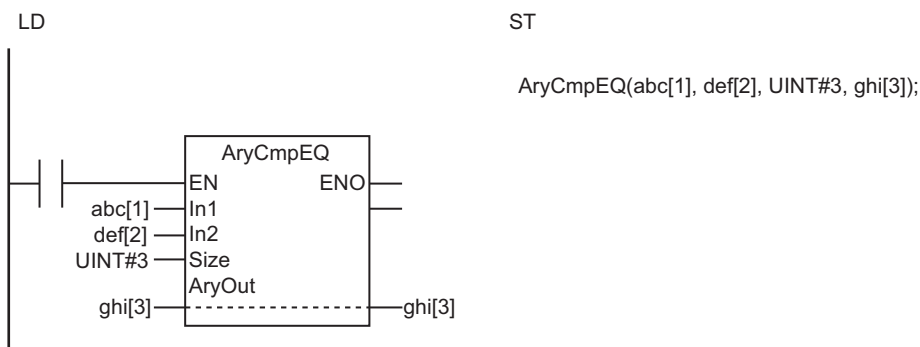
## Function

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.



|             |               |    |            |   |                  |       |
|-------------|---------------|----|------------|---|------------------|-------|
| Size=UINT#3 | In1[0]=abc[1] | 10 | In2=INT#10 | → | AryOut[0]=def[2] | TRUE  |
|             | In1[1]=abc[2] | 20 | In2=INT#10 | → | AryOut[1]=def[3] | FALSE |
|             | In1[2]=abc[3] | 30 | In2=INT#10 | → | AryOut[2]=def[4] | FALSE |

## 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 case. ENO will be FALSE, and AryOut[] will not change.
  - a) If the In1[], In2[], or AryOut[] array is smaller than the value of Size.

# AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE

These instructions compare the corresponding elements of two arrays as below.

- 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/<br>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

|                          | 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. | ---  | *1      |
| Size                     | Number of comparison elements |        | Number of elements to compare             |                       |      | 1       |
| AryOut[] (array)         | Comparison results array      | In-out | Comparison results array                  | Depends on data type. | ---  | ---     |

|     | Meaning      | I/O    | Description | Valid range | Unit | Default |
|-----|--------------|--------|-------------|-------------|------|---------|
| 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.

|                       | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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[] (ar-<br>ray) | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out                   | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

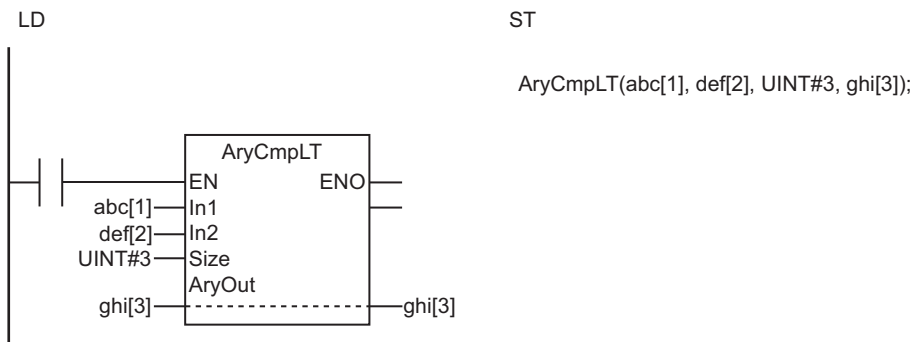
## Function

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.



|             |               |     |               |     |                    |       |
|-------------|---------------|-----|---------------|-----|--------------------|-------|
| Size=UINT#3 | In1[0]=abc[1] | 110 | In2[0]=def[2] | 100 | → AryOut[0]=ghi[3] | FALSE |
|             | In1[1]=abc[2] | 120 | In2[1]=def[3] | 130 | → AryOut[1]=ghi[4] | TRUE  |
|             | In1[2]=abc[3] | 140 | In2[2]=def[4] | 160 | → AryOut[2]=ghi[5] | TRUE  |

## 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 case. `ENO` will be FALSE, and `AryOut[]` will not change.
  - a) If the `In1[]`, `In2[]`, or `AryOut[]` array is smaller than the value of `Size`.

# AryCmpEQV and AryCmpNEV

These instructions compare each element of an array with a comparison value.

AryCmpEQV : Determines if each element of the array is equal to the comparison value.

AryCmpNEV : Determines if each element of the array is not equal to the comparison value.

| Instruction | Name                                   | FB/<br>FUN | Graphic expression | ST expression                           |
|-------------|--|------------|--------------------|---|
| AryCmpEQV   | Array Value<br>Comparison<br>Equal     | FUN        |                    | AryCmpEQV(In1, In2, Size, Ary-<br>Out); |
| AryCmpNEV   | Array Value<br>Comparison<br>Not Equal | FUN        |                    | AryCmpNEV(In1, In2, Size, Ary-<br>Out); |

## Variables

|                       | Meaning                          | I/O    | Description                                   | Valid range                | Unit | Default |
|-----------------------|----------------------------------|--------|---|----------------------------|------|---------|
| In1[] (array)         | Comparison array                 | Input  | Array containing the el-<br>ements to compare | Depends on da-<br>ta type. | ---  | *1      |
| In2                   | Comparison value                 |        | Value to compare                              |                            |      |         |
| Size                  | Number of comparison<br>elements |        | Number of elements to<br>compare              |                            |      | 1       |
| AryOut[] (ar-<br>ray) | Comparison results ar-<br>ray    | In-out | Comparison results ar-<br>ray                 | Depends on da-<br>ta 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.

|                       | Boo<br>lean                                      | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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[] (ar-<br>ray) | OK   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out                   | OK   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

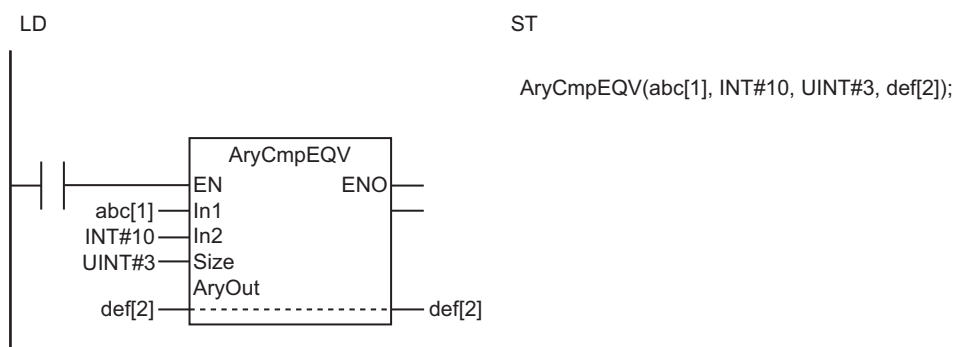
These instructions compare comparison value  $In2$  with the specified elements in an array ( $In1[0]$  to  $In1[Size-1]$ ).

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 $In1[i] \neq In2$ , 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.



|             |                 |    |                           |                    |       |
|-------------|-----------------|----|---------------------------|--------------------|-------|
| Size=UINT#3 | $In1[0]=abc[1]$ | 10 | $In2=INT\#10 \rightarrow$ | $AryOut[0]=def[2]$ | TRUE  |
|             | $In1[1]=abc[2]$ | 20 | $In2=INT\#10 \rightarrow$ | $AryOut[1]=def[3]$ | FALSE |
|             | $In1[2]=abc[3]$ | 30 | $In2=INT\#10 \rightarrow$ | $AryOut[2]=def[4]$ | FALSE |

## 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.
  - a) If the  $In1[]$  or  $AryOut[]$  array is smaller than the value of  $Size$ .

# AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV

These instructions compare each element of an array with a comparison value as below.

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/<br>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

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

|     | Meaning      | I/O    | Description | Valid range | Unit | Default |
|-----|--------------|--------|-------------|-------------|------|---------|
| 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.

|                       | Boo<br>lean                                      | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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[] (ar-<br>ray) | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out                   | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

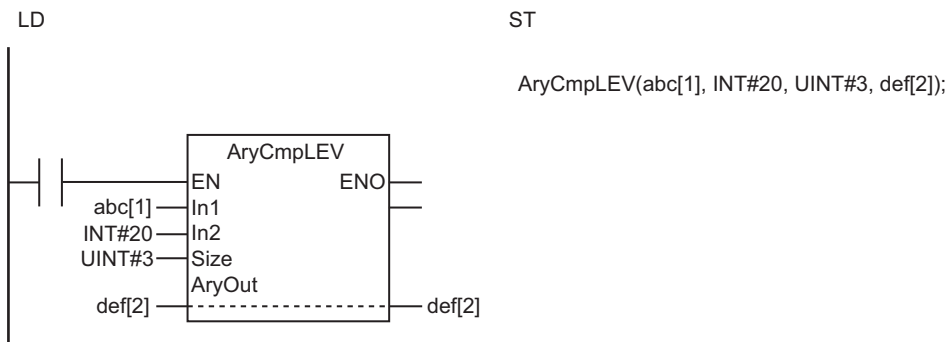
These instructions compare comparison value *In2* with the specified elements in an array (*In1*[0] to *In1*[*Size*-1]).

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 <i>AryOut</i> [ <i>i</i> ]  |
|-------------|--|
| AnyCmpLTV   | If <i>In1</i> [ <i>i</i> ] < <i>In2</i> , the result is TRUE. Otherwise, it is FALSE.  |
| AnyCmpLEV   | If <i>In1</i> [ <i>i</i> ] <= <i>In2</i> , the result is TRUE. Otherwise, it is FALSE. |
| AnyCmpGTV   | If <i>In1</i> [ <i>i</i> ] > <i>In2</i> , the result is TRUE. Otherwise, it is FALSE.  |
| AnyCmpGEV   | If <i>In1</i> [ <i>i</i> ] >= <i>In2</i> , the result is TRUE. Otherwise, it is FALSE. |

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



|             |                        |    |            |   |                           |       |
|-------------|------------------------|----|------------|---|---------------------------|-------|
| Size=UINT#3 | In1[0]= <i>abc</i> [1] | 10 | In2=INT#20 | → | AryOut[0]= <i>def</i> [2] | TRUE  |
|             | In1[1]= <i>abc</i> [2] | 20 | In2=INT#20 | → | AryOut[1]= <i>def</i> [3] | TRUE  |
|             | In1[2]= <i>abc</i> [3] | 30 | In2=INT#20 | → | AryOut[2]= <i>def</i> [4] | FALSE |

## Precautions for Correct Use

- Use the same data type for *In1*[] and *In2*. If they are different, a building error will occur.

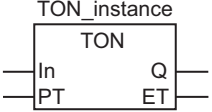
- Use an `AryOut[]` array that is at least as large as the value of `Size`.
- If `In1[]` contains real numbers and `In2` is a real number, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of `Size` is 0, the 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.
  - a) If the `In1[]` or `AryOut[]` array is smaller than the value of `Size`.

# Timer Instructions

| Instruction       | Name               | Page       |
|-------------------|--------------------|------------|
| TON               | On-Delay Timer     | page 2-136 |
| TOF               | Off-Delay Timer    | page 2-142 |
| TP                | Timer Pulse        | page 2-145 |
| AccumulationTimer | Accumulation Timer | page 2-148 |
| Timer             | Hundred-ms Timer   | page 2-152 |

# TON

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

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

## Variables

|    | Meaning      | I/O    | Description   | Valid range           | Unit | Default |
|----|--------------|--------|---|-----------------------|------|---------|
| In | Timer input  | Input  | TRUE: Timer start signal<br>FALSE: Timer reset signal | Depends on data type. | ---  | FALSE   |
| PT | Set time     |        | Time from when timer starts until Q changes to TRUE   | *1                    | ms   | 0       |
| Q  | Timer output | Output | TRUE: Timer output ON<br>FALSE: Timer output OFF      | Depends on data type. | ---  | ---     |
| ET | Elapsed time |        | Elapsed time since timer started                      | *1                    | ms   |         |

\*1. T#0 ms to T#106751d\_23h\_47m\_16s\_854.775807ms

|    | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.

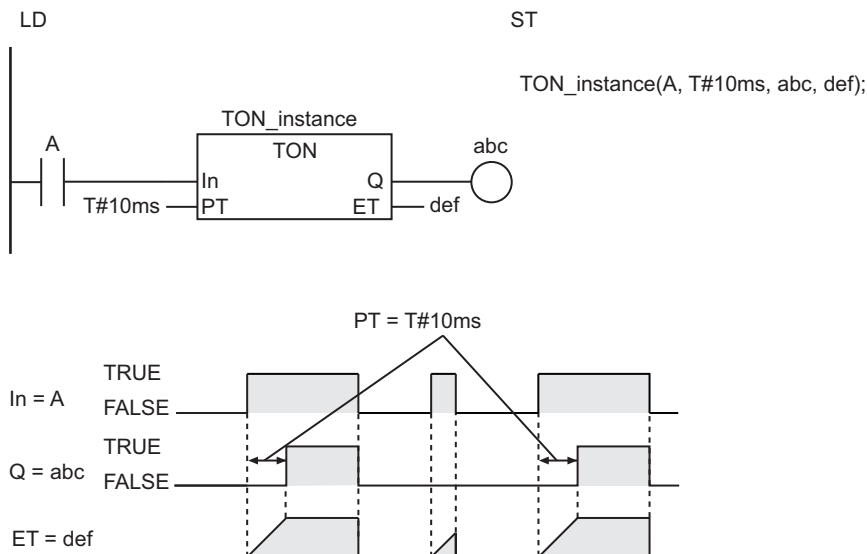
When *ET* reaches set time *PT*, timer output *Q* changes to TRUE. *ET* is not incremented after that.

The timer is reset when *In* changes to FALSE. *ET* changes to 0, and *Q* changes to FALSE.



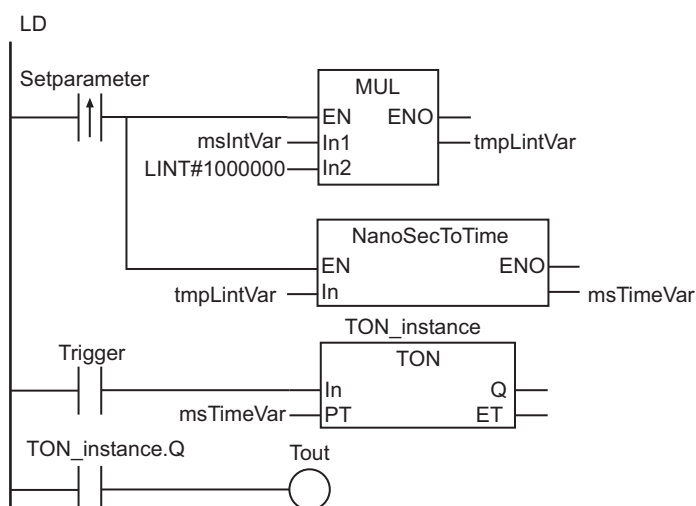
If *In* changes to FALSE after the timer is started, the timer is reset even before *ET* reaches *PT*.

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



## Additional Information

- Use the instruction, *TP* on page 2-145, 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 instruction, *TOF* on page 2-142, 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 instruction, *Timer* on page 2-152, which measures time 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 instruction, *NanoSecToTime* on page 2-679, to convert integer data to TIME data. Use the instruction, *TimeToNanoSec* on page 2-676, to convert TIME data to integer data. Both instructions express the time in nanoseconds. The following shows a user programming example where the INT variable, *msIntVar*, is the set time in milliseconds.



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:
  - a) The  $\pm 100$  ns is the timing error of ET.
  - b) Time ET is judged to check if it reaches PT every task period. If time ET reaches PT immediately after the judgment 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#0 ms 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.<br>The value of ET also does not change. It remains at the same value of PT as before it is changed.  |
| Timing in progress         | FALSE      | $PT \geq ET$                    | 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.<br>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 a correct value the next time this instruction is executed.
- If this instruction is used in a ladder diagram, the value of Q changes to FALSE when an error occurs in the previous instruction on the rung.

## Sample Programming

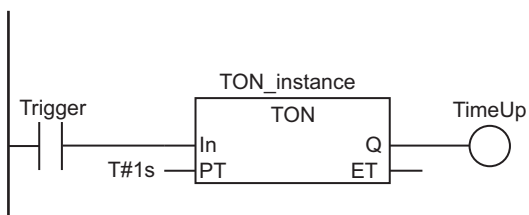
### Measuring Time with One On-Delay Timer

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

#### ● LD

| Variable | Data type | Initial value | Comment             |
|----------|-----------|---------------|---------------------|
| Trigger  | BOOL      | FALSE         | Execution condition |

| Variable     | Data type | Initial value | Comment      |
|--------------|-----------|---------------|--------------|
| TimeUp       | BOOL      | FALSE         | Timer output |
| TON_instance | TON       |               |              |



### ● ST (Example 1)

| 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;

```

### ● ST (Example 2)

| 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);

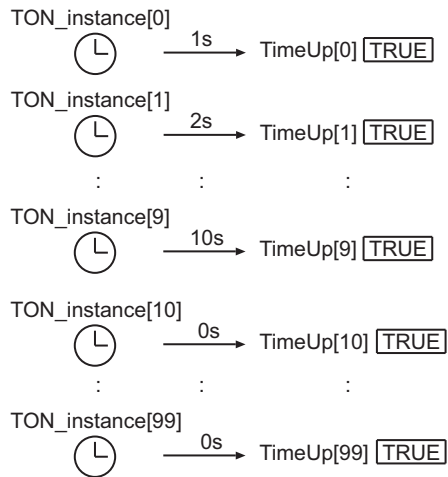
```

## 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.

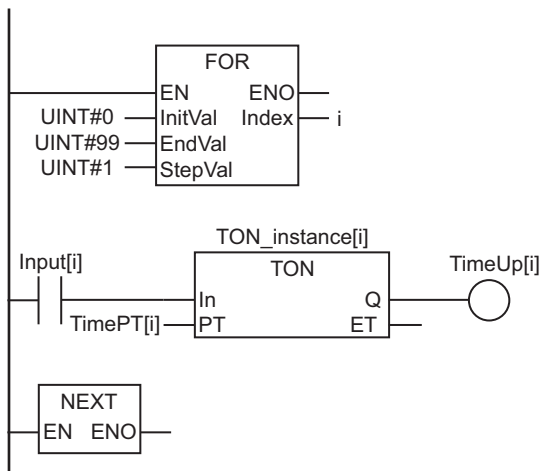
The timers for the first 10 instances, TON\_instance[0] to TON\_instance[9], change the corresponding values in TimeUp[i] to TRUE i+1 seconds (i = 0 to 9) after execution is started.

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        |

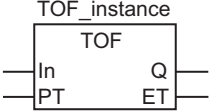
```

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/<br>FUN | Graphic expression  | ST expression                |
|-------------|-----------------|------------|---|------------------------------|
| TOF         | Off-Delay Timer | FB         |  | TOF_instance(In, PT, Q, ET); |

## Variables

|    | Meaning      | I/O    | Description   | Valid range           | Unit | Default |
|----|--------------|--------|---|-----------------------|------|---------|
| In | Timer input  | Input  | TRUE: Timer reset signal<br>FALSE: Timer start signal | Depends on data type. | ---  | FALSE   |
| PT | Set time     |        | Time from when timer starts until Q changes to TRUE   | *1                    | ms   | 0       |
| Q  | Timer output | Output | TRUE: Timer output ON<br>FALSE: Timer output OFF      | Depends on data type. | ---  | ---     |
| ET | Elapsed time |        | Elapsed time since timer started                      | *1                    | ms   |         |

\*1. T#0 ms to T#106751d\_23h\_47m\_16s\_854.775807ms

|    | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.

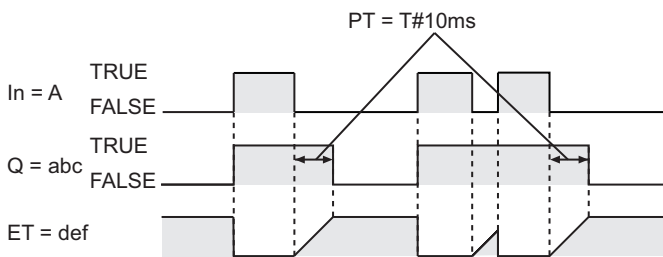
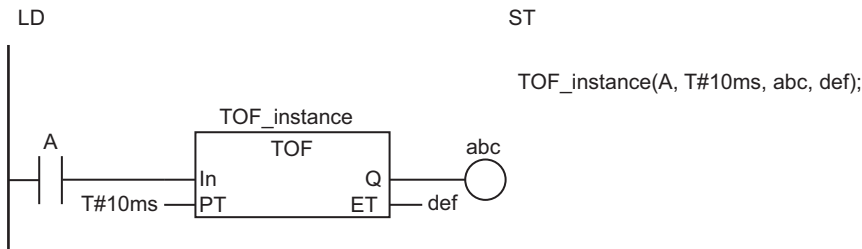
The timer starts when timer input *In* changes to FALSE. Elapsed time *ET* is incremented as time elapses.

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.

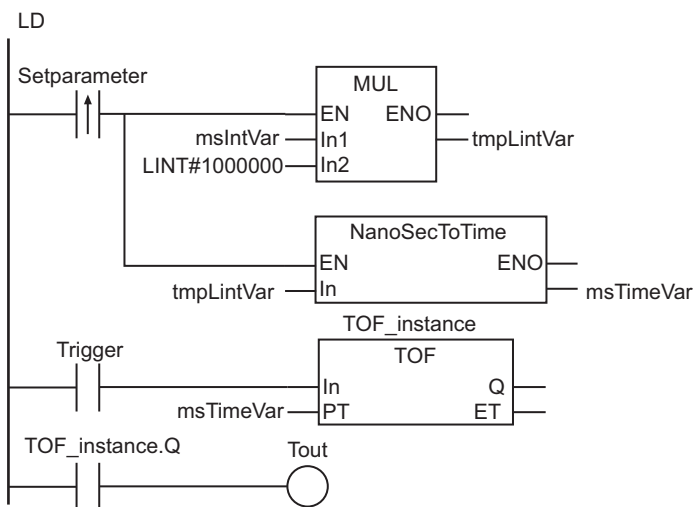
If *In* changes to FALSE after the timer is started, the timer is reset even before *ET* reaches *PT*.

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



### Additional Information

- Use the instruction, *TP* on page 2-145, 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 instruction, *TON* on page 2-136, 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 instruction, *NanoSecToTime* on page 2-679, to convert integer data to TIME data. Use the instruction, *TimeToNanoSec* on page 2-676, to convert TIME data to integer data. Both instructions express the time in nanoseconds. The following shows a user programming example where the INT variable, *msIntVar*, is the set time in milliseconds.



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:
  - a) The ±100 ns is the timing error of ET.
  - b) Time ET is judged to check if it reaches PT every task period. If time ET reaches PT immediately after the judgment 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#0 ms or a negative number is set for PT, Q will change to FALSE as soon as the value of In changes to FALSE.
- After this instruction is executed, the value of Q changes to TRUE if the value of In is TRUE. The value of Q changes to FALSE when the set time PT elapses after the timer is started.
- 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.<br>The value of ET also does not change. It remains at the same value of PT as before it is 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.<br>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:
  - a) The value of ET changes to 0, and the value of Q changes to TRUE.
  - b) If an Out instruction is connected to Q, the execution condition to the Out instruction is FALSE.
  - c) 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 a correct value the next time this instruction is executed.
- If this instruction is used in a ladder diagram, the value of Q changes to FALSE when an error occurs in the previous instruction on the rung.



# TP

The TP instruction outputs TRUE for a set period of time after the timer starts.

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

## Variables

|    | Meaning      | I/O    | Description   | Valid range           | Unit | Default |
|----|--------------|--------|---|-----------------------|------|---------|
| In | Timer input  | Input  | TRUE: Timer start signal<br>FALSE: Timer reset signal | Depends on data type. | ---  | FALSE   |
| PT | Set time     |        | Time during which Q remains at TRUE                   | *1                    | ms   | 0       |
| Q  | Timer output | Output | TRUE: Timer output ON<br>FALSE: Timer output OFF      | Depends on data type. | ---  | ---     |
| ET | Elapsed time |        | Elapsed time since timer started                      | *1                    | ms   |         |

\*1. T#0 ms to T#106751d\_23h\_47m\_16s\_854.775807ms

|    | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.

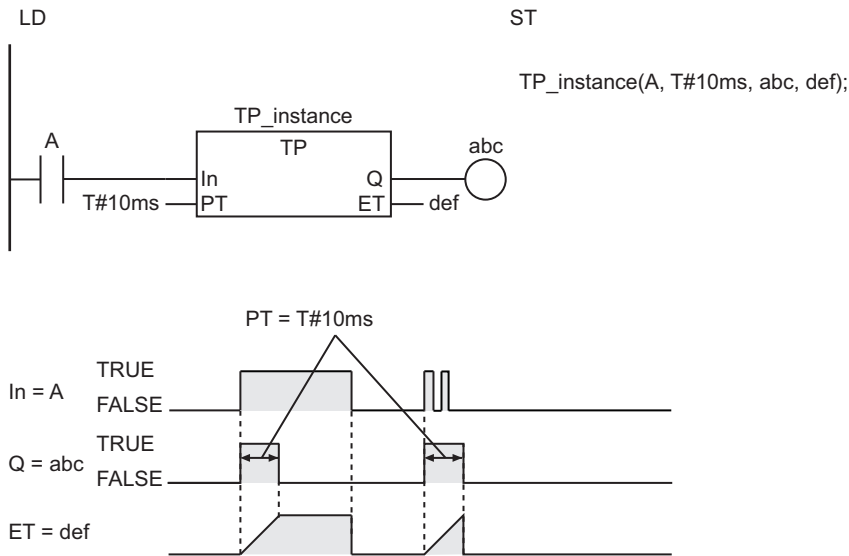
The timer starts when timer input *In* changes to TRUE, and timer output *Q* changes to TRUE. Elapsed time *ET* is incremented as time elapses.

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.

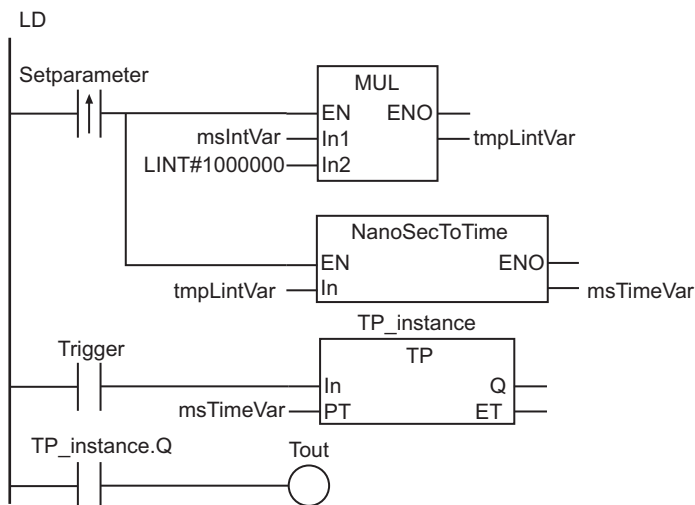
The timer is not reset if *In* changes to FALSE before *ET* reaches *PT*.

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



### Additional Information

- Use the instruction, *TON* on page 2-136, 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 instruction, *TOF* on page 2-142, 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 instruction, *NanoSecToTime* on page 2-679, to convert integer data to TIME data. Use the instruction, *TimeToNanoSec* on page 2-676, to convert TIME data to integer data. Both instructions express the time in nanoseconds. The following shows a user programming example where the INT variable, *msIntVar*, is the set time in milliseconds.



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:
  - a) The  $\pm 100$  ns is the timing error of *ET*.
  - b) Time *ET* is judged to check if it reaches *PT* every task period. If time *ET* reaches *PT* immediately after the judgment 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#0 ms or a negative number is set for *PT*, Q will not change to TRUE even when 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 <i>PT</i> after it is changed | Operation   |
|----------------------------|------------|--|---|
| After completion of timing | FALSE      | ---                                    | The value of Q remains FALSE.<br>The value of <i>ET</i> also does not change. It remains at the same value of <i>PT</i> as before it is changed.          |
| Timing in progress         | TRUE       | $PT \geq ET$                           | Timing is continued. When the value of <i>ET</i> reaches the value of <i>PT</i> , the value of Q changes to FALSE and <i>ET</i> is no longer incremented. |
|                            |            | $PT < ET$                              | The value of Q changes to FALSE immediately.<br>Incrementing <i>ET</i> 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 when 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 this instruction is executed again.
- If this instruction is used in a ladder diagram, the value of Q changes to FALSE when an error occurs in the previous instruction on the rung.

# AccumulationTimer

The AccumulationTimer instruction accumulates the period of time during which the timer input is TRUE.

| Instruction        | Name               | FB/<br>FUN | Graphic expression | ST expression                                     |
|--------------------|--------------------|------------|--------------------|---|
| Accumulation-Timer | Accumulation Timer | FB         |                    | AccumulationTimer_instance(In, PT, Reset, Q, ET); |

## Variables

|       | Meaning      | I/O    | Description   | Valid range           | Unit | Default |
|-------|--------------|--------|---|-----------------------|------|---------|
| In    | Timer input  | Input  | TRUE: Timer operates<br>FALSE: Timer stops  | Depends on data type. | ---  | FALSE   |
| PT    | Set time     |        | Maximum time  | *1                    | ms   | 0       |
| Reset | Reset        |        | TRUE: Timer reset<br>FALSE: Timer not reset   | Depends on data type. | ---  | FALSE   |
| Q     | Timer output | Output | TRUE: <i>ET</i> reached <i>PT</i> .<br>FALSE: <i>ET</i> has not reached <i>PT</i> . | Depends on data type. | ---  | ---     |
| ET    | Total time   |        | Total time  | *1                    | ms   |         |

\*1. T#0 ms to T#106751d\_23h\_47m\_16s\_854.775807ms

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

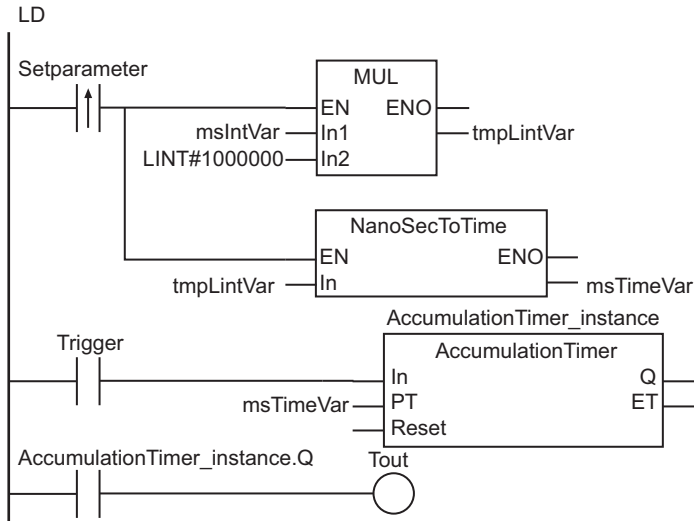
The AccumulationTimer instruction accumulates time during which the timer input *In* is TRUE. The time is set in nanoseconds.

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. The value of *ET* is held.

When *In* changes to TRUE again, the timer starts again. *ET* is incremented from the value that was previously held.





```

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:
  - a) The ±100 ns is the timing error of ET.
  - b) Time ET is judged to check if it reaches PT every task period. If time ET reaches PT immediately after the judgment 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#0 ms 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.<br>The value of ET also does not change. It remains at the same value of PT as before it is 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.<br>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:
  - a) The timer stops. The values of *ET* and *Q* at that time are retained.
  - b) When the master control reset is cleared, *ET* is incremented again from the value that was retained.
  - c) If an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even when the value of *Q* is TRUE.
  - d) *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 a correct value the next time this instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE when 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 | Meaning             | FB/<br>FUN | Graphic expression | ST expression                        |
|-------------|---------------------|------------|--------------------|--------------------------------------|
| Timer       | Hundred-ms<br>Timer | FUN        |                    | Out:=Timer(In, PT, TimerDat, Q, ET); |

## Variables

|          | Meaning        | I/O    | Description   | Valid range           | Unit | Default |
|----------|----------------|--------|---|-----------------------|------|---------|
| In       | Timer input    | Input  | TRUE: Timer start signal<br>FALSE: Timer reset signal | Depends on data type. | ---  | FALSE   |
| PT       | Set time       |        | Time from when timer starts until Q changes to TRUE   |                       | ms   | *1      |
| TimerDat | Timer status   | In-out | Current status of timer                               | ---                   | ---  | ---     |
| Out      | Return value   | Output | TRUE: Timer output ON<br>FALSE: Timer output OFF      | Depends on data type. | ---  | ---     |
| Q        | Timer output   |        | Same meaning as <i>Out</i> .                          |                       | ---  |         |
| ET       | Remaining time |        | Remaining time  |                       | ms   |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|          | Boo<br>lean | Bit strings                    |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <code>_sTimer</code> |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out      | OK          |                                |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Q        | OK          |                                |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ET       |             |                                |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |





- 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*.
- If *In* is TRUE, the timer starts as soon as operation starts.
- If the value of *PT* changes, the new value is reflected the next time 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 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 a 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 when an error occurs in the previous instruction on the rung.

# Counter Instructions

| Instruction | Name                  | Page       |
|-------------|-----------------------|------------|
| CTD         | Down-counter          | page 2-156 |
| CTD_**      | Down-counter Group    | page 2-158 |
| CTU         | Up-counter            | page 2-161 |
| CTU_**      | Up-counter Group      | page 2-164 |
| CTUD        | Up-down Counter       | page 2-167 |
| CTUD_**     | Up-down Counter Group | page 2-172 |

# 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/<br>FUN | Graphic expression | ST expression                       |
|-------------|--------------|------------|--------------------|-------------------------------------|
| CTD         | Down-counter | FB         |                    | CTD_instance (CD, Load, PV, Q, CV); |

## Variables

|        | 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                                 |                       |      |         |
| Q      | Counter output | Output | TRUE: Counter output ON<br>FALSE: Counter output OFF | Depends on data type. | ---  | ---     |
| CV     | Counter value  |        | Counter present value                                |                       |      |         |

\*1. On Sysmac Studio version 1.03 or higher, 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);.

|      | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

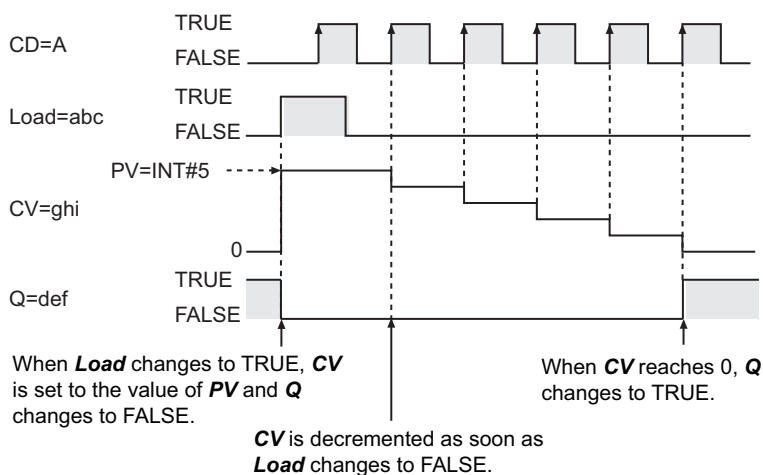
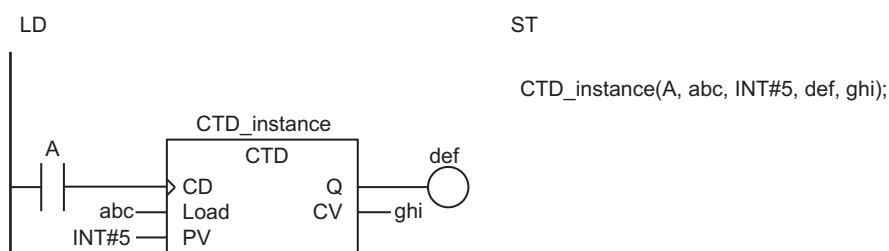
The CTD instruction creates a down counter. The preset value and counter value must have an INT data type.

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.



## Additional Information

- Use the instruction, *CTU* on page 2-161, to create a counter that increments the counter value each time the counter input signal is received.
- Use the instruction, *CTUD* on page 2-167, to create a counter that can be 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* is 0 or less, so the value of *Q* will change to TRUE immediately. After that, the value of *CV* will not be 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 when this instruction is restarted while the value of *CD* is TRUE.
- 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/<br>FUN | Graphic expression                               | ST expression   |
|-------------|--------------------|------------|--|---|
| CTD_**      | Down-counter Group | FB         | <p>**** must be DINT, LINT, UDINT, or ULINT.</p> | CTD_**_instance (CD, Load, PV, Q, CV);<br><br>**** must be DINT, LINT, UDINT, or ULINT. |

## Variables

|        | 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<br>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 or higher, 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.

|      | Boo lean                         | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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

The 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.

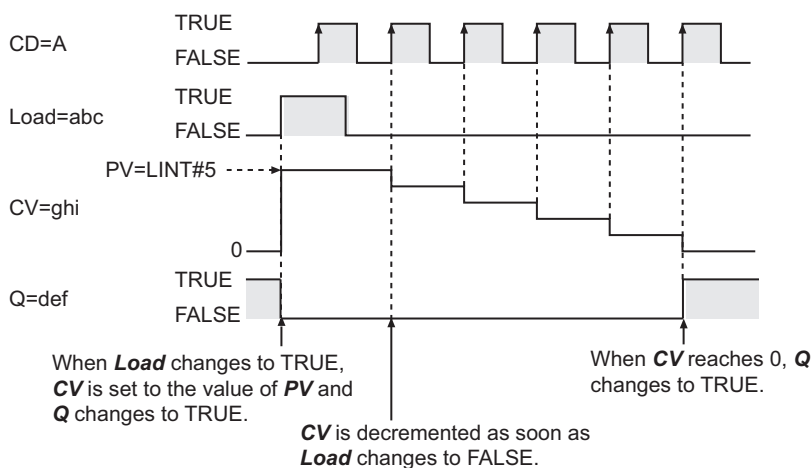
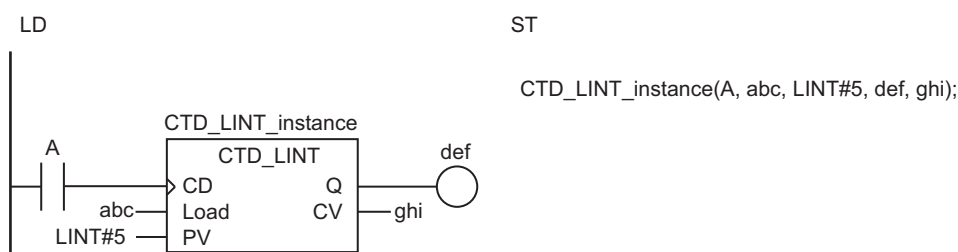
When load signal *Load* changes to TRUE, counter value *CV* is set to the value of preset value *PV* and counter output *Q* changes to FALSE.

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.



## Additional Information

- Use the instruction, *CTU* on page 2-161, to create a counter that increments the counter value each time the counter input signal is received.
- Use the instruction, *CTUD* on page 2-167, to create a counter that can be 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* is 0 or less, so the value of *Q* will change to TRUE immediately. After that, the value of *CV* will not be 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 when this instruction is restarted while the value of *CD* is TRUE.
- 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/<br>FUN | Graphic expression | ST expression                        |
|-------------|------------|------------|--------------------|--------------------------------------|
| CTU         | Up-counter | FB         |                    | CTU_instance (CU, Reset, PV, Q, CV); |

## Variables

|                     | Meaning        | I/O    | Description  | Valid range           | Unit | Default |
|---------------------|----------------|--------|--|-----------------------|------|---------|
| CU                  | Counter input  | Input  | Counter input  | Depends on data type. | ---  | FALSE   |
| Reset* <sup>1</sup> | 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<br>FALSE: Counter output OFF | Depends on data type. | ---  | ---     |
| CV                  | Counter value  |        | Counter present value                                |                       |      |         |

\*1. On Sysmac Studio version 1.03 or higher, 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);.

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.

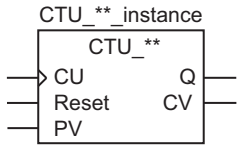


| Value of <i>PV</i>                                      | Meaning  |
|---|--|
| Equal to or smaller than the current value of <i>CV</i> | The count operation is ended. The value of <i>Q</i> changes to TRUE. The current value of <i>CV</i> is retained and will not change. |

- 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 when this instruction is restarted while the value of *CU* is TRUE.
- 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/<br>FUN | Graphic expression  | ST expression  |
|-------------|------------------|------------|---|--|
| CTU_**      | Up-counter Group | FB         |  <p>CTU_**_instance<br/>                 CTU_**<br/>                 CU    Q<br/>                 Reset    CV<br/>                 PV</p> <p>**** must be DINT, LINT, UDINT, or ULINT.</p> | CTU_**_instance (CU, Reset, PV, Q, CV);<br><br>**** must be DINT, LINT, UDINT, or ULINT. |

## Variables

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

\*1. On Sysmac Studio version 1.03 or higher, 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.

|       | Boo lean                         | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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

The 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.

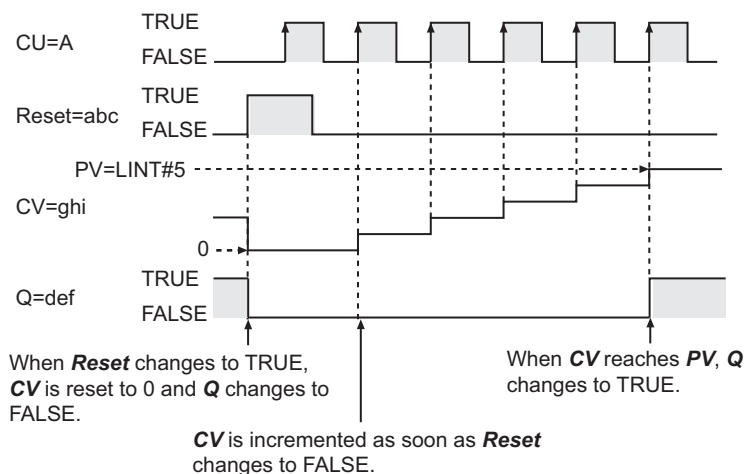
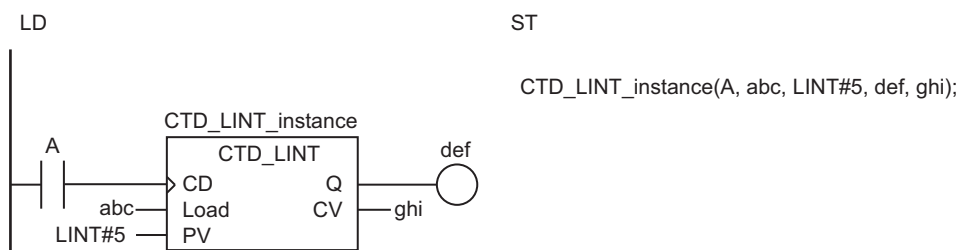
When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and counter output *Q* changes to FALSE.

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 instruction, *CTD* on page 2-156, to create a counter that decrements the counter value each time the counter input signal is received.
- Use the instruction, *CTUD* on page 2-167, to create a counter that can be 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 that 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* is changed while the value of *Reset* is FALSE.

| Value of <i>PV</i>                                      | Meaning  |
|---|--|
| Larger than the current value of <i>CV</i>              | The count operation is continued.  |
| Equal to or smaller than the current value of <i>CV</i> | The count operation is ended. The value of <i>Q</i> changes to TRUE. The current value of <i>CV</i> is retained and will 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 when this instruction is restarted while the value of *CU* is TRUE.
- 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/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------|------------|--------------------|--|
| CTUD        | Up-down Counter | FB         |                    | CTUD_instance (CU, CD, Reset, Load, PV, QU, QD, CV); |

## Variables

|                     | 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* <sup>1</sup> | Reset signal        |        | TRUE: Reset CV to 0   |                       |      |         |
| Load* <sup>1</sup>  | Load signal         |        | TRUE: CV set to PV  |                       |      |         |
| PV                  | Preset value        |        | The final counter value when operating as an up counter<br>The initial counter value when operating as a down counter | 0 to 32767            |      | 0       |
| QU                  | Up-counter output   | Output | TRUE: up-counter output ON<br>FALSE: up-counter output OFF  | Depends on data type. | ---  | ---     |
| QD                  | Down-counter output |        | TRUE: down-counter output ON<br>FALSE: down-counter output OFF  |                       |      |         |
| CV                  | Counter value       |        | Counter present value   |                       |      |         |

\*1. On Sysmac Studio version 1.03 or higher, 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 strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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      |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|       | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| 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.

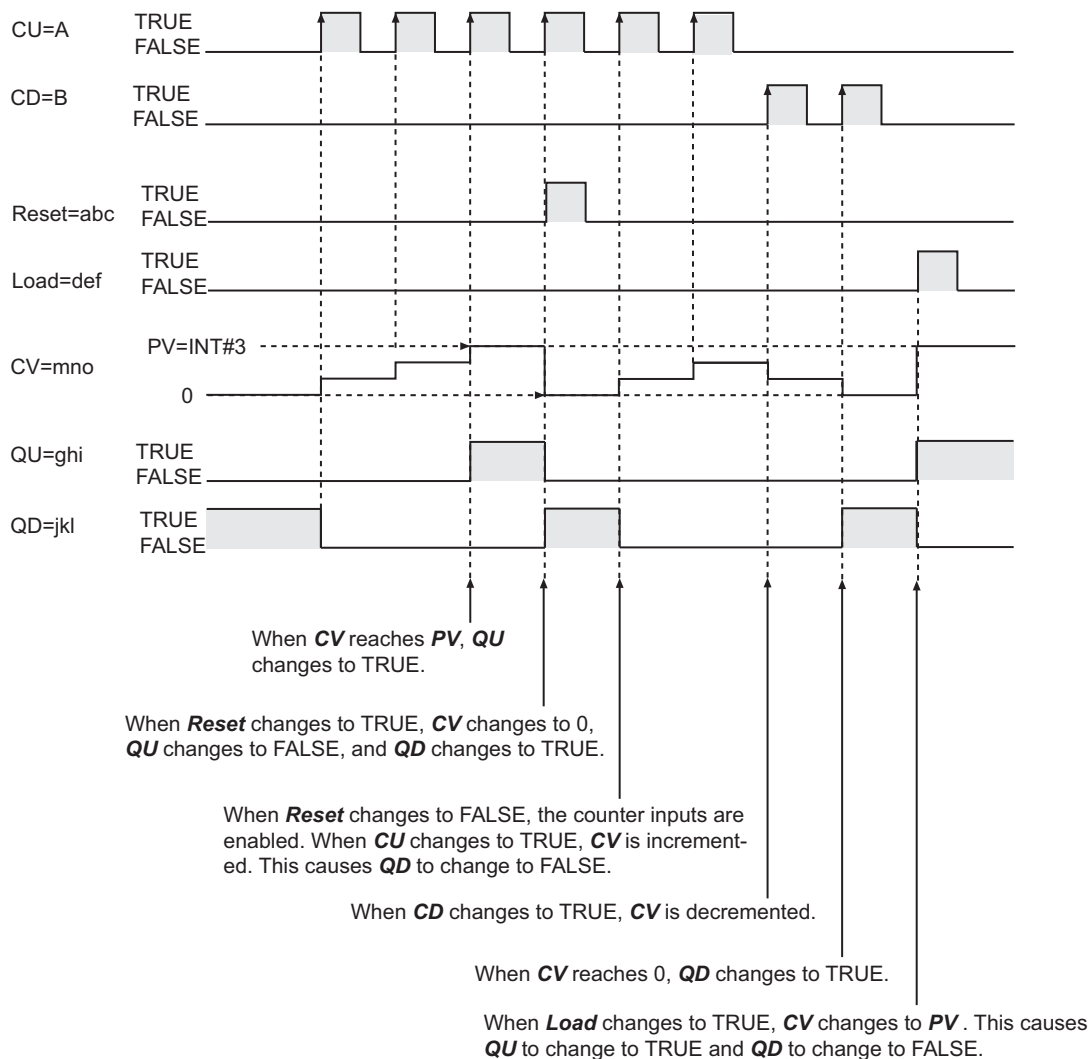
If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE.

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.







## Additional Information

Use *CTD* on page 2-156 or *CTU* on page 2-161 to create either a down counter or up counter alone.

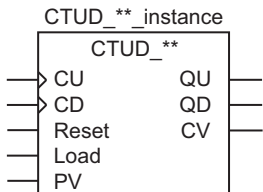
## Precautions for Correct Use

- 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* is 0 or less, so the value of *QD* will change to TRUE immediately. After that, the value of *CV* will not be 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* is equal to or higher than *PV*, so the value of *QU* changes to TRUE immediately. After that, the value of *CV* will not be 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 when this instruction is restarted while the value of *CU* or *CD* is TRUE.

# 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/<br>FUN | Graphic expression  | ST expression   |
|-------------|-----------------------|------------|---|---|
| CTUD_**     | Up-down Counter Group | FB         |  <p>*** must be DINT, LINT, UDINT, or ULINT.</p> | CTUD_**_instance (CU, CD, Reset, Load, PV, QU, QD, CV);<br><br>*** must be DINT, LINT, UDINT, or ULINT. |

## Variables

|                     | 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 <sup>*1</sup> | Reset signal        |        | TRUE: Reset CV to 0   |                                     |      |         |
| Load <sup>*1</sup>  | Load signal         |        | TRUE: CV set to PV  |                                     |      |         |
| PV                  | Preset value        | Output | The final counter value when operating as an up counter<br>The initial counter value when operating as a down counter | Depends on data type. <sup>*2</sup> | ---  | 0       |
| QU                  | Up-counter output   |        | TRUE: up-counter output ON<br>FALSE: up-counter output OFF  | Depends on data type.               | ---  | ---     |
| QD                  | Down-counter output |        | TRUE: down-counter output ON<br>FALSE: down-counter output OFF  |                                     |      |         |
| CV                  | Counter value       |        | Counter present value   | Depends on data type. <sup>*2</sup> |      |         |

\*1. On Sysmac Studio version 1.03 or higher, 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.

|       | Boo<br>lean                             | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>PV</i> |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

## Function

The 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 name of 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.

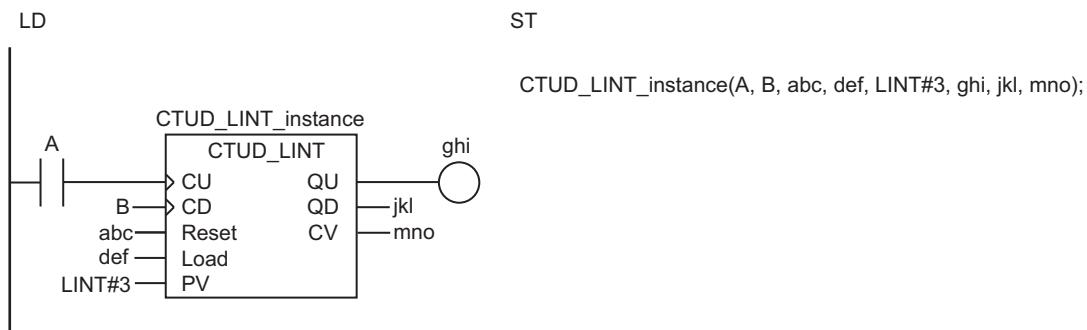
If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE.

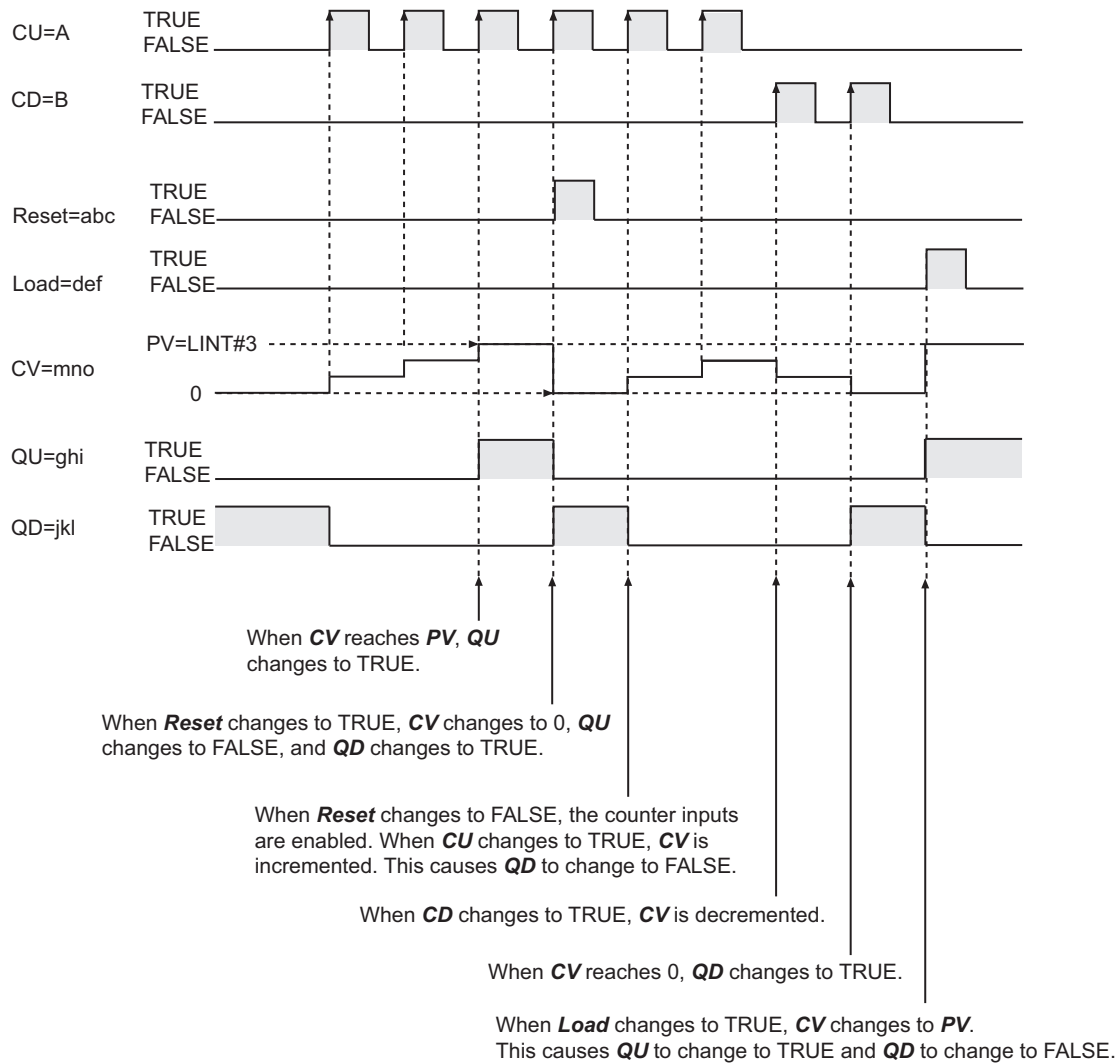
If *Load* changes to TRUE, the value of *CV* changes to *PV*, and so *QU* changes to TRUE.

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.<br>• <i>CV</i> is incremented when <i>CU</i> changes to TRUE. It is not decremented when <i>CD</i> changes to TRUE.   |
|       |       | Between 0 and <i>PV</i> | FALSE | FALSE | Both up and down counter operation is performed.<br>• <i>CV</i> is incremented when <i>CU</i> changes to TRUE and decremented when <i>CD</i> changes to TRUE. |
|       |       | <i>PV</i> or higher     | TRUE  | FALSE | Only down counter operation is performed.<br>• <i>CV</i> is decremented when <i>CD</i> changes to TRUE. It is not incremented when <i>CU</i> changes to TRUE. |
| TRUE  | FALSE | 0                       | FALSE | TRUE  | The up counter is reset.<br>• The value of <i>CV</i> is set to 0.   |
| FALSE | TRUE  | <i>PV</i>               | TRUE  | FALSE | The down counter is reset.<br>• The value of <i>CV</i> is set to <i>PV</i> .  |
| TRUE  | TRUE  | 0                       | FALSE | TRUE  | The up counter is reset. <i>Reset</i> take priority over <i>Load</i> .<br>• The value of <i>CV</i> is set to 0.   |

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





## Additional Information

Use `CTD_**` on page 2-158 or `CTU_**` on page 2-164 to create either a down counter or up counter alone.

## 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* is 0 or less, so the value of *QD* will change to TRUE immediately. After that, the value of *CV* will not be 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* will not be 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 when this instruction is restarted while the value of *CU* or *CD* is *TRUE*.



# Math Instructions

| Instruction           | Name  | Page       |
|-----------------------|---|------------|
| ADD (+)               | Addition  | page 2-179 |
| AddOU (+OU)           | Addition with Overflow Check                                  | page 2-183 |
| SUB (-)               | Subtraction   | page 2-187 |
| SubOU (-OU)           | Subtraction with Overflow Check                               | page 2-190 |
| MUL (*)               | Multiplication  | page 2-194 |
| MulOU (*OU)           | Multiplication with Overflow Check                            | page 2-198 |
| DIV (/)               | Division  | page 2-202 |
| MOD                   | Modulo-division   | page 2-205 |
| ABS                   | Absolute Value  | page 2-207 |
| RadToDeg and DegToRad | Radians to Degrees/Degrees to Radians                         | page 2-209 |
| SIN, COS, and TAN     | Sine in Radians/Cosine in Radians/Tangent in Radians          | page 2-211 |
| ASIN, ACOS, and ATAN  | Principal Arc Sine/Principal Arc Cosine/Principal Arc Tangent | page 2-214 |
| SQRT                  | Square Root   | page 2-217 |
| LN and LOG            | Natural Logarithm/Logarithm Base 10                           | page 2-220 |
| EXP                   | Natural Exponential Operation                                 | page 2-224 |
| EXPT (**)             | Exponentiation  | page 2-226 |
| Inc and Dec           | Increment/Decrement   | page 2-232 |
| Rand                  | Random Number   | page 2-234 |
| AryAdd                | Array Addition  | page 2-236 |
| AryAddV               | Array Value Addition  | page 2-238 |
| ArySub                | Array Subtraction   | page 2-240 |

| Instruction | Name                             | Page       |
|-------------|----------------------------------|------------|
| ArySubV     | Array Value Subtraction          | page 2-242 |
| AryMean     | Array Mean                       | page 2-244 |
| ArySD       | Array Element Standard Deviation | page 2-246 |
| ModReal     | Real Number Modulo-division      | page 2-248 |
| Fraction    | Real Number Fraction             | page 2-250 |
| CheckReal   | Real Number Check                | page 2-252 |

# ADD (+)

The ADD (+) instruction adds integers or real numbers. It also joins text strings.

| Instruction | Name     | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|----------|------------|--------------------|-----------------|
| ADD (+)     | Addition | FUN        |                    | Out:=In1 + In2; |

## Variables

|            | Meaning      | I/O    | Description  | Valid range           | Unit | Default |
|------------|--------------|--------|--|-----------------------|------|---------|
| In1 to InN | Add values   | Input  | Numbers to add<br>Ladder diagram: N = 2<br>to 5<br>ST: N = 2*1 | Depends on data type. | ---  | 0*2     |
| Out        | Output value | Output | Output value   | Depends on data type. | ---  | ---     |

\*1. You can use more than one instruction as operators in one 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, where N is 3, if 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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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 up two or more and five or less 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*.

The variables, *In1* to *InN* (Add values) can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other



The ADD instruction adds *In1* through *InN*.  
The calculation is  $10 + 20 + 30 = 60$ , so the value of *abc* will be INT#60.

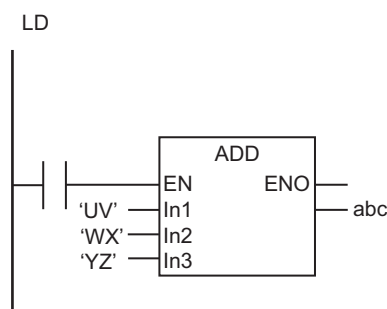
$$\text{In1 } \boxed{\text{INT\#10}} + \text{In2 } \boxed{\text{INT\#20}} + \text{In3 } \boxed{\text{INT\#30}} \xrightarrow{\text{Added.}} \text{Out=abc } \boxed{\text{INT\#60}}$$

## 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 shows an example where *In1*, *In2*, and *In3* are UV, WX, and YZ, respectively. The value of STRING variable *abc* will be 'UVWXYZ'.



Text strings of *In1* to *InN* are joined to make one text string.  
The calculation is 'UV' + 'WX' + 'YZ' = 'UVWXYZ', so the value of *abc* will be 'UVWXYZ'.

$$\text{In1 } \boxed{\text{'UV'}} + \text{In2 } \boxed{\text{'WX'}} + \text{In3 } \boxed{\text{'YZ'}} \xrightarrow{\text{Text strings joined.}} \text{Out = abc } \boxed{\text{'UVWXYZ'}}$$

## 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 <sup>*1</sup>                       |
| Omitting input parameters for values to add                              | You can omit parameters except for the input parameters connected to <i>InN</i> .       | You cannot omit any input parameters. |
| Existence of <i>EN</i> and <i>ENO</i>                                    | Present   | None                                  |
| Number of data processing bits if the values to add are all integer data | 8, 16, 32, or 64 <sup>*2</sup>  | 32 or 64 <sup>*3</sup>                |
| 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. You can use more than one Add instruction as operators in one 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 up, 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 instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.
- Use the instruction, *CONCAT* on page 2-582, to join text strings in structured text.

### Precautions for Correct Use

- *Out* can have a different data type than the addition result. However, it should be able to accommodate the valid value range of the data type of the addition result. Otherwise, a building error will occur.  
For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.
- 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 will not occur.
- Addition of real number values with positive or negative infinity is handled as follows.

| 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 variables from *In1* to *InN* is nonnumeric data, the value of the addition result is nonnumeric data.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The size of the joined text string exceeds 1,986 bytes.

# AddOU (+OU)

The AddOU (+OU) instruction adds integers and real numbers. It also performs an overflow check for the integer addition result.

| Instruction | Name                            | FB/<br>FUN | Graphic expression | ST expression              |
|-------------|---------------------------------|------------|--------------------|----------------------------|
| AddOU (+OU) | Addition with<br>Overflow Check | FUN        |                    | Out:=AddOU(In1, ..., InN); |

## Variables

|            | Meaning      | I/O    | Description                  | Valid range           | Unit | Default |
|------------|--------------|--------|------------------------------|-----------------------|------|---------|
| In1 to InN | Add values   | Input  | Numbers to add<br>N = 2 to 5 | Depends on data type. | ---  | 0*1     |
| Out        | Output Value | Output | Output Value                 | Depends on data type. | ---  | ---     |

- \*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, where N is 3, if 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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>number |          | Times, durations,<br>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<br>*1 | OK<br>*1                                     |      |      |     |    |        |
| Out        |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK             | OK       | OK   |      |      |     |    |        |

- \*1. If any of the values from *In1* to *InN* is REAL data, an overflow check is not performed.

## Function

The AddOU (+OU) instruction adds up two or more and five or less integers or real numbers, and outputs the result to output value *Out*.

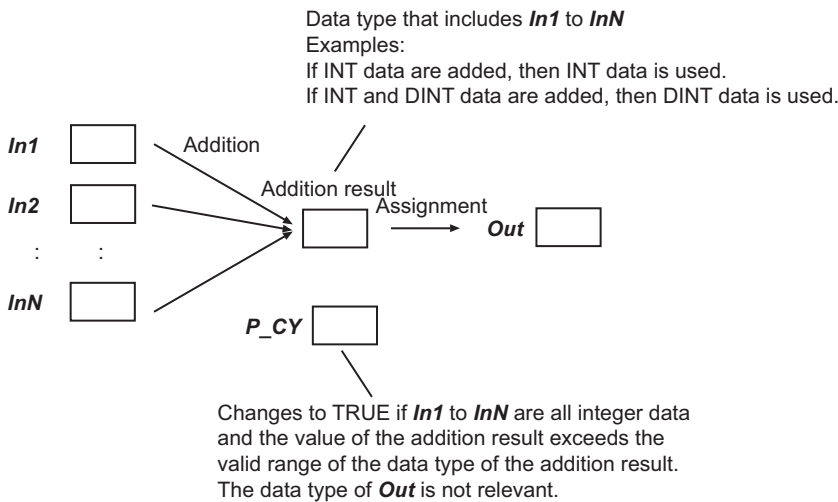
The variables, *In1* to *InN* (Add values), can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data types. Calculations are performed based on the data type that accommodates every possible value held by the other existing data types. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the addition result is DINT data.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

## Processing for Overflows

An overflow occurs if the sum of the variables from *In1* to *InN* exceeds the valid range of the data type of the addition result. If all of the variables from *In1* to *InN* are integer data, the value of the *P\_CY* system-defined variable (Carry Flag) changes to TRUE when an overflow occurs.

If any of the variables from *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 <i>In1</i> to <i>InN</i> | Data type of addition result | Value of addition result   | Value of <i>P_CY</i> |
|--|------------------------------|--|----------------------|
| All integer data                       | Integer data                 | Of the sum of the variables from <i>In1</i> to <i>InN</i> , 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 values of *In1* and *In2* are INT#32767 and INT#3, respectively, 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.

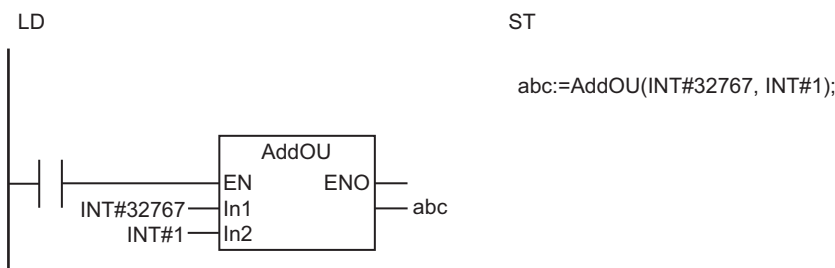
\*2. If the sum of the variables from *In1* to *InN* is positive, the addition result will be positive infinity. If the sum is negative, the addition result will be negative infinity.

## Notation Examples

The following shows an example where *In1* and *In2* are INT#32767 and INT#1, respectively, and variable *abc* is INT data.

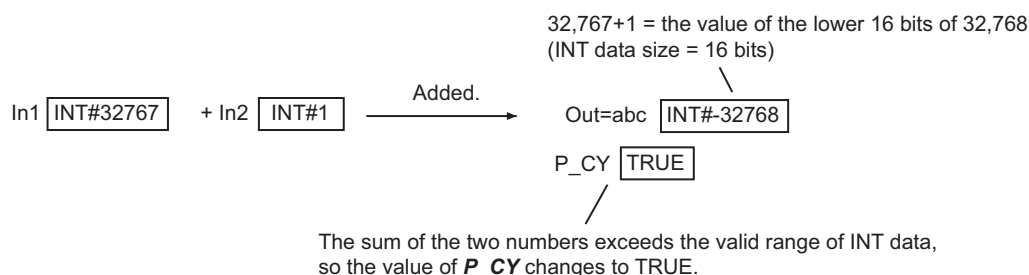
*In1* and *In2* are both INT data, so the addition result is INT data. The sum of the two values (32,768) exceeds the valid range of INT data, so the value of *P\_CY* changes to TRUE. The value of INT variable *abc* will be INT#-32768 (the lower 16 bits of 32768).





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.<br>FALSE: Overflow did not occur for integer calculations. |

### Additional Information

- If *Out* is REAL data, use the instruction, *CheckReal* on page 2-252, to check if it is positive infinity, negative infinity, or non-numeric data.
- Use the instruction, *ADD (+)* on page 2-179, 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, it should be able to accommodate the valid value range of the data type of the addition result. Otherwise, a building error will occur.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

- 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 will not occur.
- Addition of real number values with positive or negative infinity is handled as follows.

| 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 variables from *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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------------|------------|--------------------|-----------------|
| SUB (-)     | Subtraction | FUN        |                    | Out:=In1 - In2; |

## Variables

|     | Meaning      | I/O    | Description  | Valid range           | Unit | Default |
|-----|--------------|--------|--------------|-----------------------|------|---------|
| In1 | Minuend      | Input  | Minuend      | Depends on data type. | ---  | 0*1     |
| In2 | Subtrahend   |        | Subtrahend   |                       |      |         |
| Out | Output value | Output | Output Value | Depends on data type. | ---  | ---     |

\*1. 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    | 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. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data type. Calculations are performed based on the data type that accommodates every possible value held by the other existing data type. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the subtraction result is DINT data.



## 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 <i>EN</i> and <i>ENO</i>   | Present                        | None                   |
| Number of data processing bits if the minuend and subtrahend are integer data | 8, 16, 32, or 64* <sup>1</sup> | 32 or 64* <sup>2</sup> |

- \*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 subtract one SINT value from another SINT value, 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 instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

### Precautions for Correct Use

- *Out* can have a different data type than the subtraction result. However, it should be able to accommodate the valid value range of the data type of the subtraction result. Otherwise, a building error will occur.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

- 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 will not occur.
- Subtraction of real number values with positive or negative infinity is handled as follows.

| 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 either *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 the integer subtraction result.

| Instruction | Name                            | FB/<br>FUN | Graphic expression | ST expression         |
|-------------|---------------------------------|------------|--------------------|-----------------------|
| SubOU (-OU) | Subtraction with Overflow Check | FUN        |                    | Out:=SubOU(In1, In2); |

## Variables

|     | Meaning      | I/O    | Description  | Valid range           | Unit | Default |
|-----|--------------|--------|--------------|-----------------------|------|---------|
| In1 | Minuend      | Input  | Minuend      | Depends on data type. | ---  | 0*1     |
| In2 | Subtrahend   |        | Subtrahend   |                       |      |         |
| Out | Output value | Output | Output Value | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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*1 | OK*1   |      |      |     |    |        |
| In2 |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   | OK*1 | OK*1   |      |      |     |    |        |
| Out |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   | OK   | OK   |      |      |     |    |        |

\*1. If either *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*.

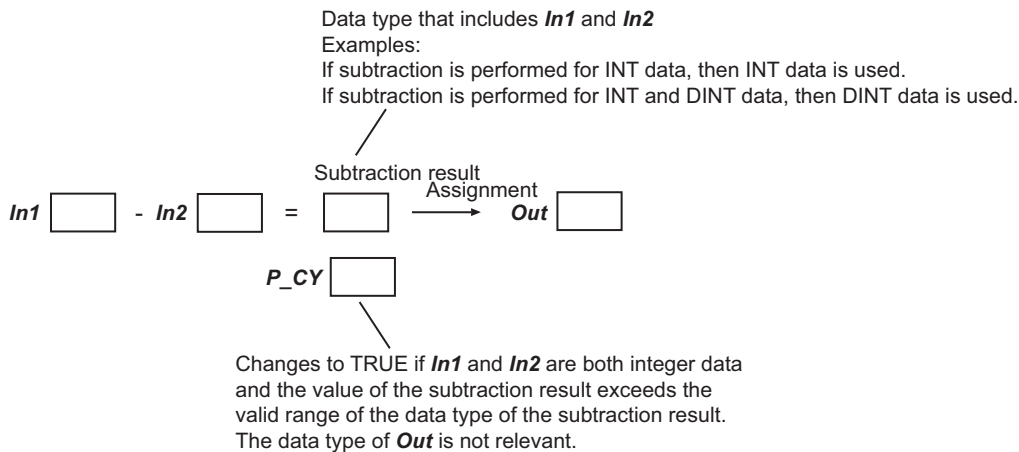
*In1* and *In2* can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data type. Calculations are performed based on the data type that accommodates every possible value held by the other existing data type. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the subtraction result is DINT data.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

## 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, the value of the *P\_CY* system-defined variable (Carry Flag) changes to TRUE when an overflow occurs.

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 <i>In1</i> and <i>In2</i> | Data type of subtraction result | Value of subtraction result  | Value of <i>P_CY</i> |
|---|---------------------------------|--|----------------------|
| All integer data                        | Integer data                    | Of the difference between <i>In1</i> and <i>In2</i> , 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                | $\pm\infty$ *2   | Does not change.     |

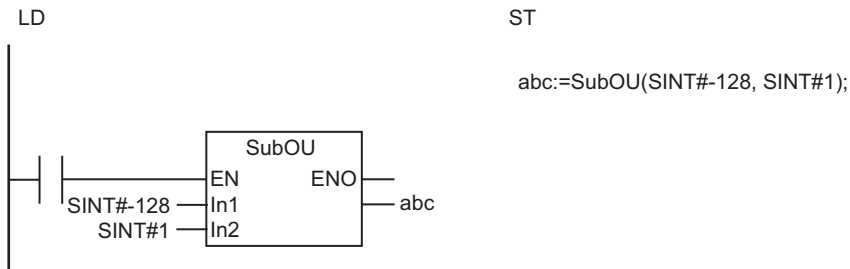
\*1. For example, if the values of *In1* and *In2* are INT#32767 and INT#-3, respectively, the subtraction result will be INT data. The value of the subtraction result will be the lower 16 bits of the difference (32,770), i.e., INT#-32766.

\*2. If the difference between *In1* and *In2* is positive, the subtraction result will be positive infinity. If the difference is negative, the subtraction result will be negative infinity.

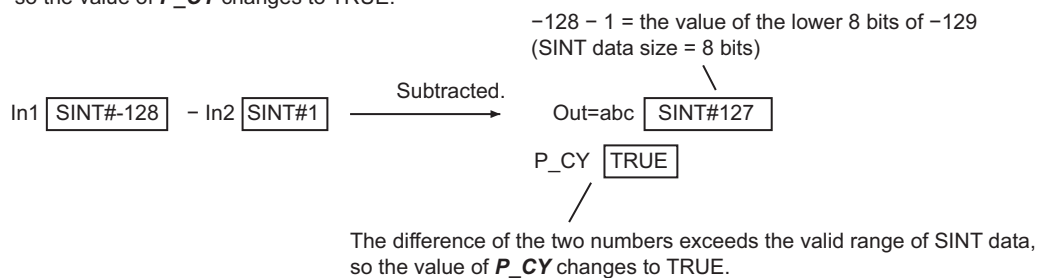
## Notation Examples

The following shows an example where *In1* and *In2* are SINT#-128 and INT#1, respectively, 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).



The SubOU instruction subtracts *In2* from *In1*. The difference between the two values ( $-129$ ) exceeds the valid range of SINT 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 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.<br>FALSE: Overflow did not occur for integer calculations. |

## Additional Information

- When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.
- Use the instruction, *SUB (-)* on page 2-187, 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, it should be able to accommodate the valid value range of the data type of the subtraction result. Otherwise, a building error will occur.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.



- 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 will not occur.
- Subtraction of real number values with positive or negative infinity is handled as follows.

| 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 either  $In1$  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/<br>FUN | Graphic expression | ST expression   |
|-------------|----------------|------------|--------------------|-----------------|
| MUL (*)     | Multiplication | FUN        |                    | Out:=In1 * In2; |

## Variables

|            | Meaning            | I/O    | Description   | Valid range           | Unit | Default |
|------------|--------------------|--------|---|-----------------------|------|---------|
| In1 to InN | Values to multiply | Input  | Numbers to multiply<br>Ladder diagram: N = 2<br>to 5<br>ST: N = 2*1 | Depends on data type. | ---  | 1*2     |
| Out        | Output value       | Output | Output Value  | Depends on data type. | ---  | ---     |

- \*1. You can use more than one MUL instruction as operators in one 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, where N is 3, if 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.

|            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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                   |       |  |      |     |    |        |
| Out        |             |             |      |       |       | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK                   |       |  |      |     |    |        |

## Function

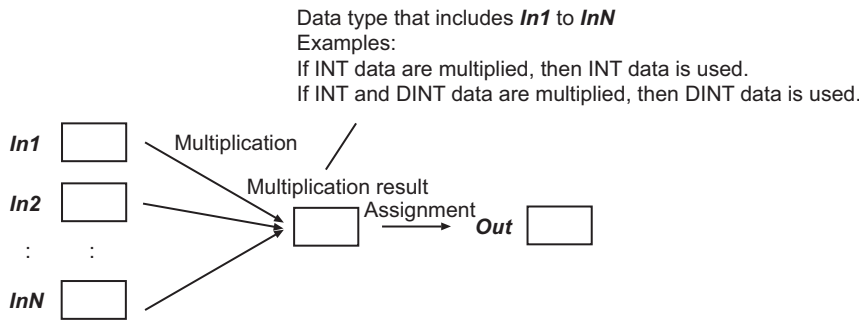
In a ladder diagram, the MUL (\*) instruction multiplies two or more and five or less integers or real numbers, and outputs the result to output value *Out*.

In ST, the MUL instruction multiplies two integers or real numbers, and outputs the result to output value *Out*.

The variables, *In1* to *InN* (Values to multiply), can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by

the other data types. Calculations are performed based on the data type that accommodates every possible value held by the other existing data types. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the multiplication result is DINT data.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.



## Processing for Overflows

An overflow occurs if the product of the variables from *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.

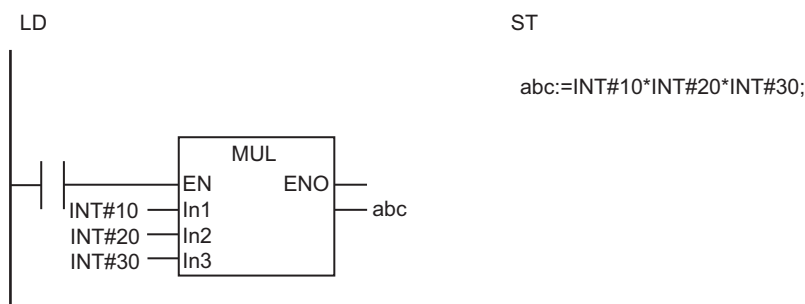
| Data types of <i>In1</i> to <i>InN</i> | Data type of multiplication result | Value of multiplication result  |
|--|------------------------------------|---|
| All integer data                       | Integer data                       | Of the product of the variables from <i>In1</i> to <i>InN</i> , 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                   | $\pm\infty$ *2  |

\*1. For example, if the values of *In1* and *In2* are INT#16384 and INT#2, respectively, 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 the variables from *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 shows an example where *In1*, *In2*, and *In3* are INT#10, INT#20, and INT#30, respectively. The value of INT variable *abc* will be INT#6000.



The MUL instruction multiplies *In1* to *InN*.

The calculation is  $10 \times 20 \times 30 = 6,000$ , 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 parameters except for the input parameters connected to InN. | You cannot omit any input parameters. |
| Existence of <i>EN</i> and <i>ENO</i>   | Present   | None                                  |
| Number of data processing bits if the values to multiply are all integer data | 8, 16, 32, or $64^{*2}$   | 32 or $64^{*3}$                       |

- \*1. You can use more than one MUL instruction as operators in one 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 instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

## Precautions for Correct Use

- *Out* can have a different data type than the multiplication result. However, it should be able to accommodate the valid value range of the data type of the multiplication result. Otherwise, a building error will occur.  
For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.
- 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 will not occur.
- Multiplication of real number values with positive or negative infinity is handled as follows.

| Multiplication                  | Multiplication result |
|---------------------------------|-----------------------|
| $+\infty$ times positive number | $+\infty$             |
| $+\infty$ times negative number | $-\infty$             |
| $-\infty$ times positive number | $-\infty$             |
| $-\infty$ times negative number | $+\infty$             |
| $+\infty$ times $+\infty$       | $+\infty$             |
| $-\infty$ times $-\infty$       | $+\infty$             |
| $+\infty$ times $-\infty$       | $-\infty$             |
| $+\infty$ times 0               | Nonnumeric data       |
| $-\infty$ times 0               | Nonnumeric data       |

- If any of the variables from  $ln1$  to  $lnN$  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 the integer multiplication result.

| Instruction | Name                               | FB/<br>FUN | Graphic expression | ST expression              |
|-------------|------------------------------------|------------|--------------------|----------------------------|
| MulOU (*OU) | Multiplication with Overflow Check | FUN        |                    | Out:=MulOU(In1, ..., InN); |

## Variables

|            | Meaning            | I/O    | Description                       | Valid range           | Unit | Default |
|------------|--------------------|--------|-----------------------------------|-----------------------|------|---------|
| In1 to InN | Values to multiply | Input  | Numbers to multiply<br>N = 2 to 5 | Depends on data type. | ---  | 1*1     |
| Out        | Output value       | Output | Output Value                      | Depends on data type. | ---  | ---     |

- \*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, where N is 3, if 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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |          | Times, durations,<br>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<br>*1 | OK<br>*1                                     |      |      |     |    |        |
| Out        |             |             |      |       |       | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   | OK       | OK   |      |      |     |    |        |

- \*1. If any of the values from *In1* to *InN* is REAL data, an overflow check is not performed.

## Function

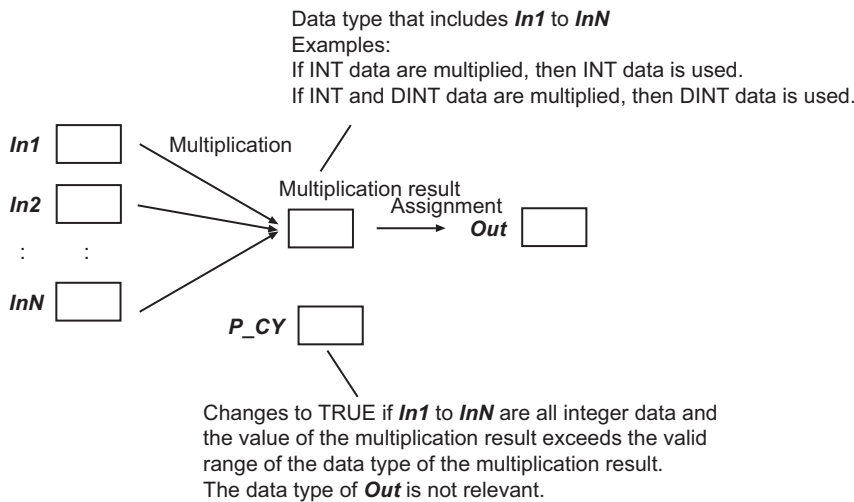
The MulOU (\*OU) instruction multiplies two or more and five or less integers or real numbers, and outputs the result to output value *Out*.

Values to multiply, *In1* to *InN*, can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data types. Calculations are performed based on the data type that accommodates every possible value held by the other existing data types. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the multiplication result is DINT data.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

## Processing for Overflows

An overflow occurs if the product of the variables from *In1* to *InN* exceeds the valid range of the data type of the multiplication result. If all of the variables from *In1* to *InN* are integer data, the value of the *P\_CY* system-defined variable (Carry Flag) changes to TRUE when an overflow occurs. If any of the variables from *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 <i>In1</i> to <i>InN</i> | Data type of multiplication result | Value of multiplication result   | Value of <i>P_CY</i> |
|--|------------------------------------|--|----------------------|
| All integer data                       | Integer data                       | Of the product of the variables from <i>In1</i> to <i>InN</i> , 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                   | $\pm\infty$ *2   | Does not change.     |

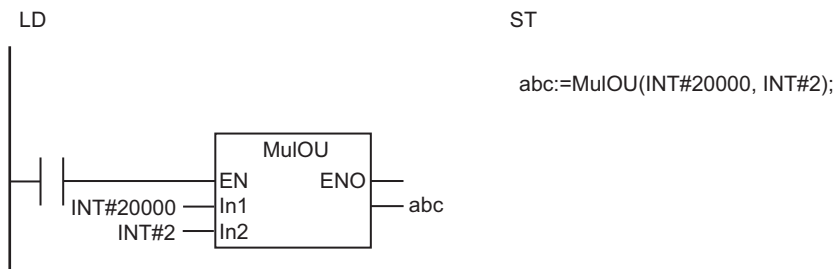
\*1. For example, if the values of *In1* and *In2* are INT#16384 and INT#2, respectively, the multiplication result will be INT data. The value of the multiplication result will be the lower 16 bits of the product (32,768), i.e., INT#-32768.

\*2. If the product of the variables from *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

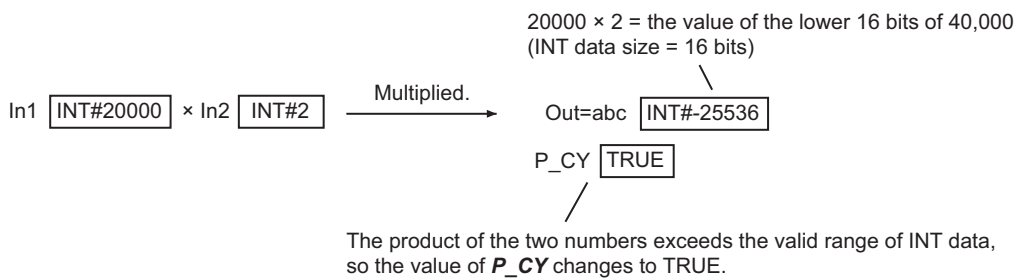
The following shows an example where *In1* and *In2* are INT#20000 and INT#2, respectively, and variable *abc* is INT data.

*In1* and *In2* are both INT data, so the multiplication 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 40000).



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.<br>FALSE: Overflow did not occur for integer calculations. |

## Additional Information

- When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.
- Use the instruction, *MUL* (\*) on page 2-194, 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, it should be able to accommodate the valid value range of the data type of the multiplication result. Otherwise, a building error will occur.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

- 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 will not occur.
- Multiplication of real number values with positive or negative infinity is handled as follows.

| Multiplication                  | Multiplication result |
|---------------------------------|-----------------------|
| $+\infty$ times positive number | $+\infty$             |
| $+\infty$ times negative number | $-\infty$             |
| $-\infty$ times positive number | $-\infty$             |
| $-\infty$ times negative number | $+\infty$             |
| $+\infty$ times $+\infty$       | $+\infty$             |
| $-\infty$ times $-\infty$       | $+\infty$             |
| $+\infty$ times $-\infty$       | $-\infty$             |
| $+\infty$ times 0               | Nonnumeric data       |
| $-\infty$ times 0               | Nonnumeric data       |

- If any of the variables from *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/<br>FUN | Graphic expression | ST expression  |
|-------------|----------|------------|--------------------|----------------|
| DIV (/)     | Division | FUN        |                    | Out:=In1/ In2; |

## Variables

|     | Meaning      | I/O    | Description  | Valid range           | Unit | Default |
|-----|--------------|--------|--------------|-----------------------|------|---------|
| In1 | Dividend     | Input  | Dividend     | Depends on data type. | ---  | *1      |
| In2 | Divisor      |        | Divisor      |                       |      |         |
| Out | Output value | Output | Output Value | Depends on data type. | ---  | ---     |

\*1. 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    | 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    |   |      |     |    |        |

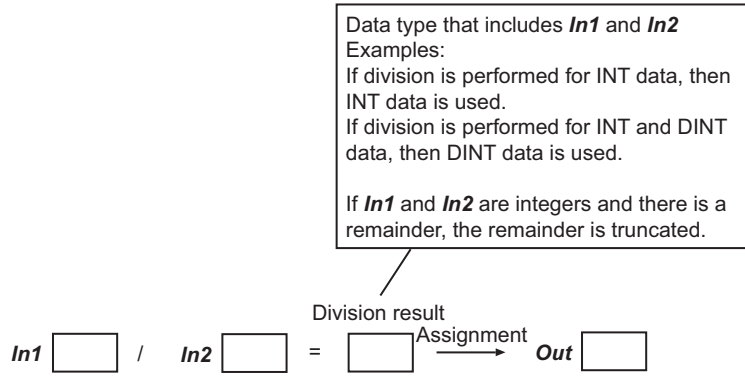
## Function

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. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data type. Calculations are performed based on the data type that accommodates every possible value held by the other existing data type. For example, if *In1* and *In2* are INT data and DINT data, respectively, calculations are performed with DINT data. Here, the division result is DINT data.

If *In1* and *In2* are both integers, the remainder is truncated, if any.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.



## Processing for Overflows

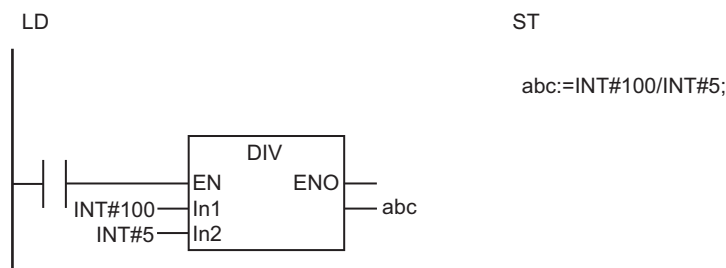
An overflow occurs if the quotient of *In1* and *In2* exceeds the valid range of the data type of the division result. If an overflow occurs, the data types of *In1* and *In2*, the data type of the division result, and the value of the division result will be as shown in the following table.

| Data types of <i>In1</i> and <i>In2</i> | Data type of division result | Value of division result |
|---|------------------------------|--------------------------|
| At least one real number                | Real number data             | $\pm\infty^{*1}$         |

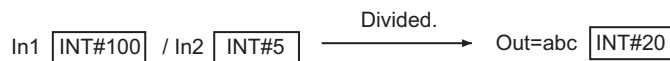
\*1. If the quotient of *In1* and *In2* is positive, the division result will be positive infinity. If the quotient is negative, the division result will be negative infinity.

## Notation Examples

The following shows an example where *In1* and *In2* are INT#100 and INT#5, respectively. The value of INT variable *abc* will be INT#20.



The DIV instruction divides *In1* by *In2*.  
 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 <i>EN</i> and <i>ENO</i>                                       | 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 instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

## Precautions for Correct Use

- *Out* can have a different data type than the division result. However, it should be able to accommodate the valid value range of the data type of the division result. Otherwise, a building error will occur.  
For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.
- 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 will not occur.
- Division of real number values with positive infinity, negative infinity, or 0 is handled as follows.

|     |                 | 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 either *In1* or *In2* is nonnumeric data, the value of the division result is nonnumeric data.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *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/<br>FUN | Graphic expression | ST expression     |
|-------------|-----------------|------------|--------------------|-------------------|
| MOD         | Modulo-division | FUN        |                    | Out:=In1 MOD In2; |

## Variables

|     | Meaning   | I/O    | Description | Valid range           | Unit | Default |
|-----|-----------|--------|-------------|-----------------------|------|---------|
| In1 | Dividend  | Input  | Dividend    | Depends on data type. | ---  | *1      |
| In2 | Divisor   |        | Divisor     |                       |      |         |
| Out | Remainder | Output | Remainder   | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 return the remainder.

*In1*, *In2*, and *Out* can have different data types. In the combination of the different data types, however, one of the data types should be able to accommodate values held by the other data types.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.

This instruction performs the calculation with the following formula.

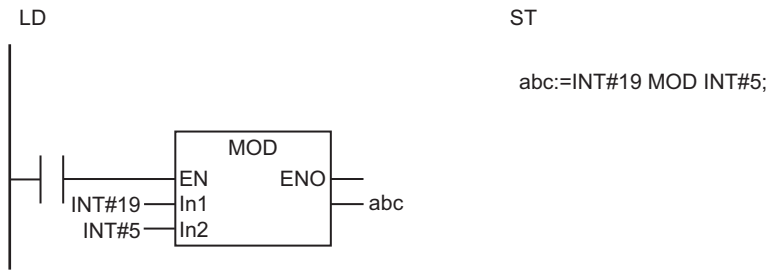
$$Out = In1 - (In1/In2)*In2$$

Decimal places are truncated in the division operation.

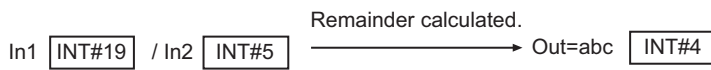
Examples with the values of *In1*, *In2* and *Out* are given in the following table.

| Value of <i>In1</i> | Value of <i>In2</i> | Value of <i>Out</i> |
|---------------------|---------------------|---------------------|
| 5                   | 3                   | 2                   |
| 5                   | -3                  | 2                   |
| -5                  | 3                   | -2                  |
| -5                  | -3                  | -2                  |

The following shows an example where *In1* and *In2* are INT#19 and INT#5, respectively. The value of variable *abc* will be INT#4.



The MOD instruction divides *In1* by *In2* to find the remainder.  
The remainder of 19/5 is 4, so the value of *abc* will be INT#4.



### Precautions for Correct Use

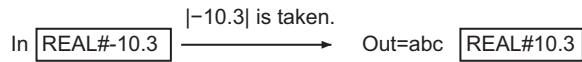
Set the data type of *Out* to accommodate the valid value ranges of *In1* and *In2*.

For casting between data types, refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)*.



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 instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

### Precautions for Correct Use

- Set the data type of *Out* to accommodate 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 <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| $+\infty$          | $+\infty$           |
| $-\infty$          | $+\infty$           |
| 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/<br>FUN | Graphic expression | ST expression      |
|-------------|--------------------|------------|--------------------|--------------------|
| RadToDeg    | Radians to Degrees | FUN        |                    | Out:=RadToDeg(In); |
| DegToRad    | Degrees to Radians | FUN        |                    | Out:=DegToRad(In); |

## Variables

|     | Meaning           | I/O    | Description       | Valid range           | Unit                                   | Default |
|-----|-------------------|--------|-------------------|-----------------------|--|---------|
| In  | Data to convert   | Input  | Data to convert   | Depends on data type. | RadToDeg: Radians<br>DegToRad: Degrees | *1      |
| Out | Conversion result | Output | Conversion result | Depends on data type. | RadToDeg: Degrees<br>DegToRad: Radians | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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.

$$Out = In * 180 / \pi$$

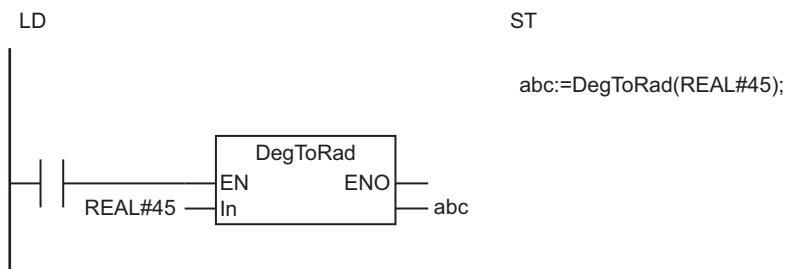
## DegToRad

The DegToRad instruction converts the data to convert *In* from degrees (°) to radians (rad).

The following conversion is used.

$$Out = In * \pi / 180$$

The following shows an example where *In* is REAL#45 for the DegToRad instruction. The value of the REAL variable *abc* will be REAL#0.785398.



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

When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric 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 <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| + ∞                | + ∞                 |
| -∞                 | -∞                  |
| 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 <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| 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 : Calculates the sine of a number.
- COS : Calculates the cosine of a number.
- TAN : Calculates the tangent of a number.

| Instruction | Name               | FB/<br>FUN | Graphic expression | ST expression |
|-------------|--------------------|------------|--------------------|---------------|
| SIN         | Sine in Radians    | FUN        |                    | Out:=SIN(In); |
| COS         | Cosine in Radians  | FUN        |                    | Out:=COS(In); |
| TAN         | Tangent in Radians | FUN        |                    | Out:=TAN(In); |

## Variables

|     | Meaning            | I/O    | Description        | Valid range                                      | Unit | Default |
|-----|--------------------|--------|--------------------|--|------|---------|
| In  | Number to process  | Input  | Number to process  | Depends on data type.                            | rad  | *1      |
| Out | Calculation result | Output | Calculation result | SIN: *2<br>COS: *2<br>TAN: Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

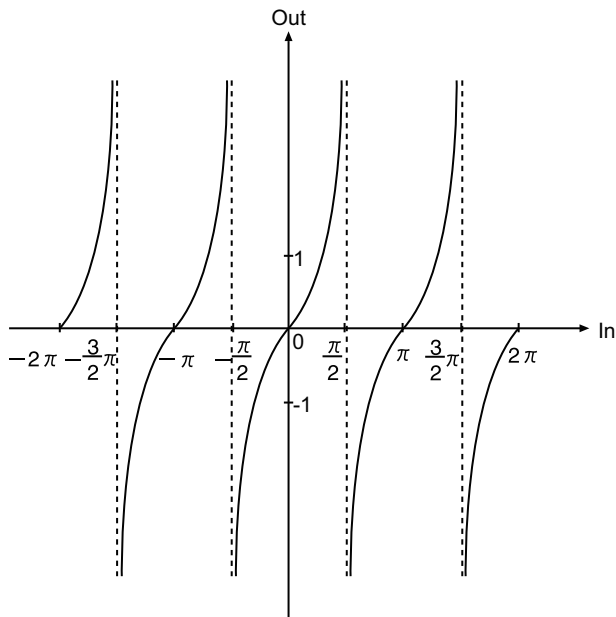
\*2. The valid value range for REAL data is from -1.000000e+0 to 1.000000e+0. The valid value range for LREAL data is from -1.0000000000000000e+0 to 1.0000000000000000e+0.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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).





### Additional Information

- Use the instructions, *RadToDeg* and *DegToRad* on page 2-209, to convert data between degrees and radians.
- If  $In$  for the TAN instruction is  $n\pi/2$  ( $n$  is an integer), the value of  $Out$  is positive infinity or negative infinity.  
Use the instruction, *CheckReal* on page 2-252, to check if the value of  $Out$  is positive infinity, negative infinity, or non-numeric data.

### 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. |

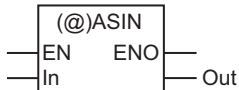
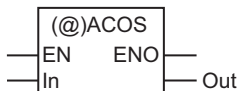

# ASIN, ACOS, and ATAN

These instructions perform inverse trigonometric calculations on real numbers.

ASIN : Calculates the arc sine of a number ( $\sin^{-1}$ )

ACOS : Calculates the arc cosine of a number ( $\cos^{-1}$ )

ATAN : Calculates the arc tangent of a number ( $\tan^{-1}$ )

| Instruction | Name  | FB/<br>FUN | Graphic expression   | ST expression  |
|-------------|---|------------|--|----------------|
| ASIN        | Principal Arc Sine ( $\text{SIN}^{-1}$ )    | FUN        |   | Out:=ASIN(In); |
| ACOS        | Principal Arc Cosine ( $\text{COS}^{-1}$ )  | FUN        |   | Out:=ACOS(In); |
| ATAN        | Principal Arc Tangent ( $\text{TAN}^{-1}$ ) | FUN        |  | Out:=ATAN(In); |

## Variables

|     | Meaning            | I/O    | Description        | Valid range  | Unit | Default |
|-----|--------------------|--------|--------------------|--|------|---------|
| In  | Number to process  | Input  | Number to process  | Depends on data type.  | ---  | *1      |
| Out | Calculation result | Output | Calculation result | ASIN: $-\pi/2$ to $\pi/2$<br>ACOS: 0 to $\pi$<br>ATAN: $-\pi/2$ to $\pi/2$ | rad  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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    |  |      |     |    |        |

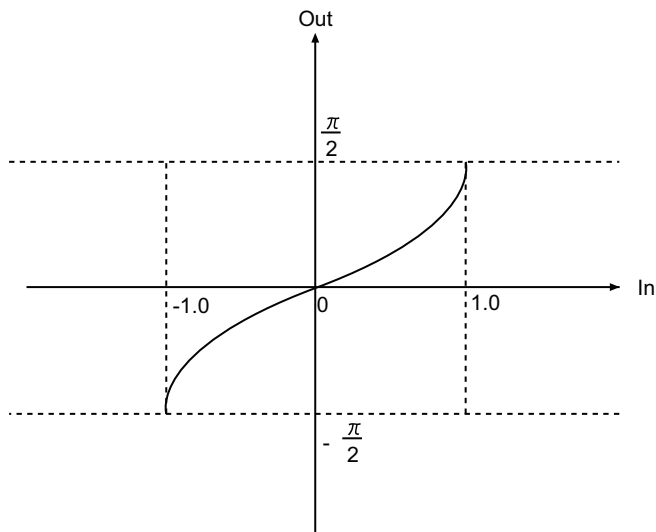
## Function

These instructions perform inverse trigonometric calculations on real numbers.

The calculation result *Out* is an angle in radians (rad).

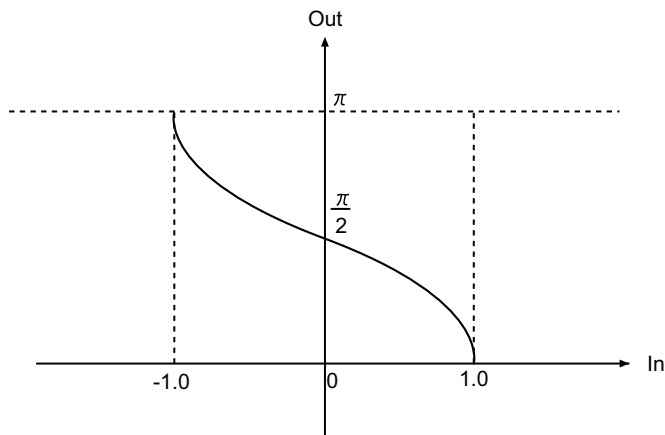
## ASIN

The ASIN instruction finds the arc sine ( $\sin^{-1}$ ) of *In*. *Out* is between  $-\pi/2$  and  $\pi/2$ .



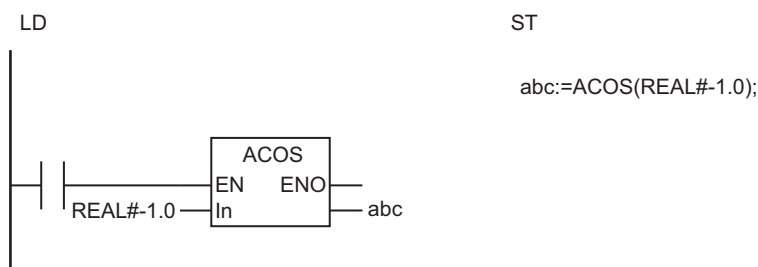
## ACOS

The ACOS instruction finds the arc cosine ( $\cos^{-1}$ ) of *In*. *Out* is between 0 and  $\pi$ .



### ● Example for ACOS Instruction

The following shows an example where *In* is REAL#-1.0 for the ACOS instruction. The value of variable *abc* will be REAL#3.141592.

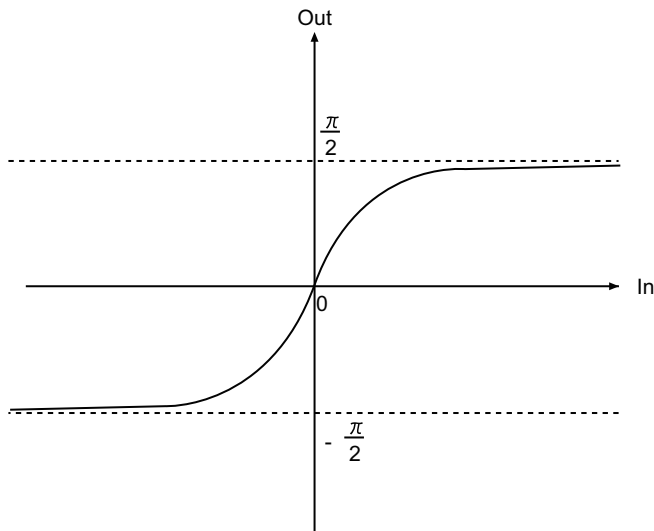


The ACOS instruction finds the arccosine of *In*.  
The arccosine of -1.0 is 3.141592, so the value of *abc* will be REAL#3.141592.

In REAL#-1.0  $\xrightarrow{\text{Arccosine found.}}$  Out=abc REAL#3.141592

### ATAN

The ATAN instruction finds the arc tangent ( $\tan^{-1}$ ) of *In*. *Out* is between  $-\pi/2$  and  $\pi/2$ .  
If the value of *In* is positive infinity, the value of *Out* is  $\pi/2$ . If the value of *In* is negative infinity, the value of *Out* is  $-\pi/2$ .



### Additional Information

Use the instructions, *RadToDeg* and *DegToRad* on page 2-209, to convert data 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 positive infinity, negative 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 <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| USINT, UINT, SINT, or INT                          | REAL                         |
| UDINT or DINT                                      | LREAL                        |
| ULINT or LINT                                      | A building error will occur. |



# SQRT

The SQRT instruction calculates the square root of a number.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|-------------|------------|--------------------|----------------|
| SQRT        | Square Root | FUN        |                    | Out:=SQRT(In); |

## Variables

|     | Name              | 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 the input parameter, the default value is not applied. A building error will occur.

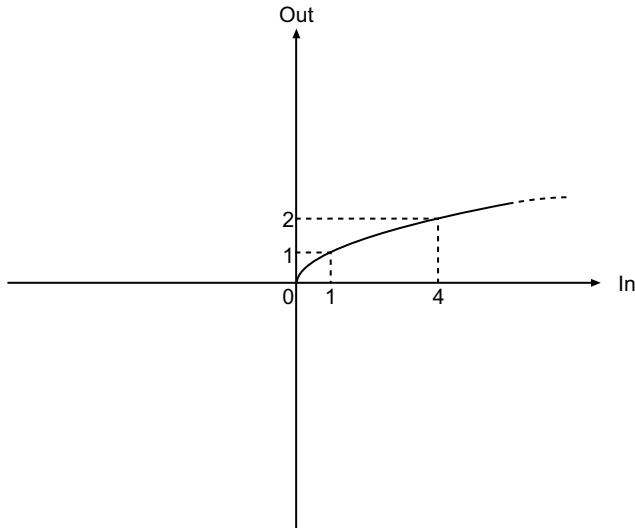
\*3. The valid value range for REAL data is from 0.000000e+00 to 1.844674e+19, or positive infinity. The valid value range for LREAL data is from 0.000000000000000e+000 to 1.34078079299425e+154, or positive infinity.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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    |  |      |     |    |        |

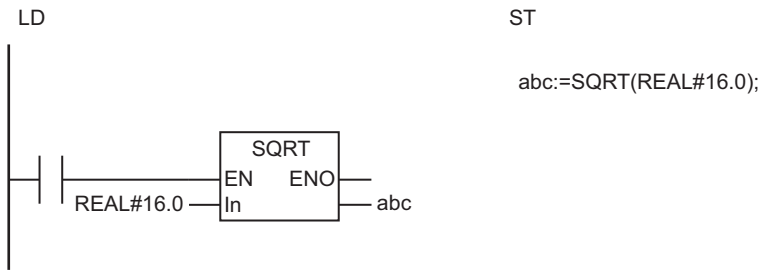
## Function

The SQRT instruction finds the square root of number to process *In*.

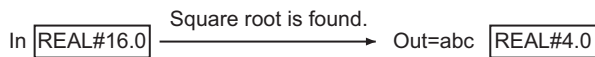
Number to process *In* and square root *Out* can have different data types.



The following shows an example where *In* is REAL#16.0. The value of variable *abc* will be REAL#4.0.



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.



## Additional Information

Use the instruction, *CheckReal* on page 2-252, to check 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 <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| 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 <i>In</i> | Data type of <i>In</i> |
|--|------------------------|
| USINT, UINT, SINT, or INT                          | REAL                   |

| Data type of parameter that is passed to <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| UDINT or DINT                                      | LREAL                        |
| ULINT or LINT                                      | A building error will occur. |

# LN and LOG

These instructions calculate the logarithm of a real number.

LN : Calculates the natural logarithm of a number.

LOG : Calculates the base-10 logarithm of a number.

| Instruction | Name              | FB/<br>FUN | Graphic expression | ST expression |
|-------------|-------------------|------------|--------------------|---------------|
| LN          | Natural Logarithm | FUN        |                    | Out:=LN(In);  |
| LOG         | Logarithm Base 10 | FUN        |                    | Out:=LOG(In); |

## Variables

|     | 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 the input parameter, the default value is not applied. A building error will occur.

\*3. The valid value range depends on the data types of *In* and *Out*. Refer to *Valid Value Range* on page 2-222 for details.

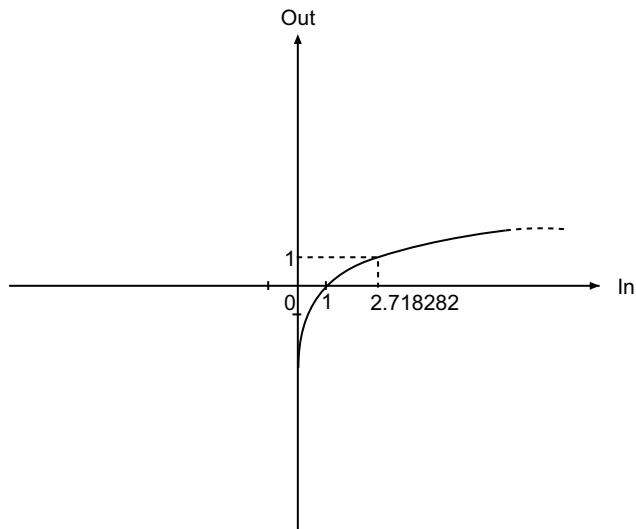
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|     | 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.

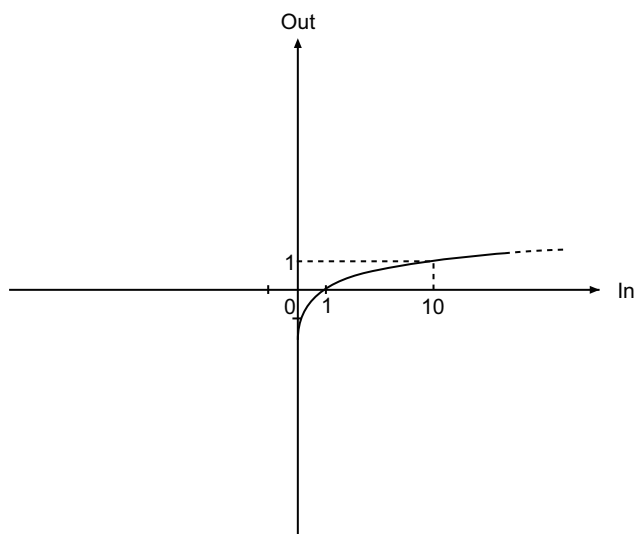
### LN

The LN instruction finds the natural logarithm (logarithm to base e, where e = 2.718282).



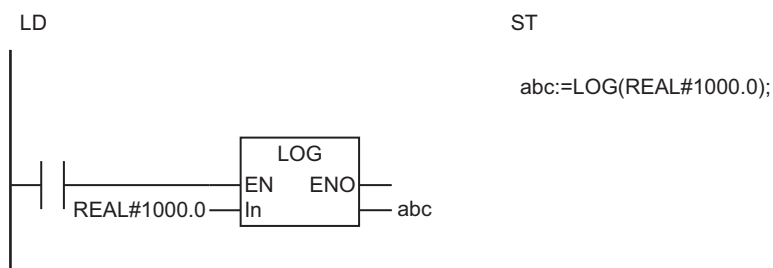
## LOG

The LOG instruction finds the base-10 logarithm.



### ● Example for the LOG Instruction

The following shows an example where *In* is REAL#1000.0 for the LOG instruction. The value of variable *abc* will be REAL#3.0.



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 REAL#1000.0  $\xrightarrow{\text{Common logarithm is taken.}}$  Out=abc REAL#3.0

## Valid Value Range

The following tables show the valid value ranges for LN and LOG.

### ● Valid Value Ranges for LN

| Data type of In | Data type of Out | Valid Value Range   |
|-----------------|------------------|---|
| REAL            | REAL             | -8.73365448e+1 to 8.87228390e+1<br>or $-\infty/+\infty$                 |
| REAL            | LREAL            | -8.7336544750000000e+1 to 8.8722839050000000e+1<br>or $-\infty/+\infty$ |
| LREAL           | REAL             | -7.08384950e+2 to 7.09782712e+2<br>or $-\infty/+\infty$                 |
| LREAL           | LREAL            | -7.0838495021978327e+1 to 7.0978271289338399e+2<br>or $-\infty/+\infty$ |

### ● Valid Value Ranges for LOG

| Data type of In | Data type of Out | Valid Value Range   |
|-----------------|------------------|---|
| REAL            | REAL             | -3.79297795e+1 to 3.85318394e+1<br>or $-\infty/+\infty$                 |
| REAL            | LREAL            | -3.7929779453965430e+1 to 3.8531839419564961e+1<br>or $-\infty/+\infty$ |
| LREAL           | REAL             | -3.07652656e+2 to 3.08254716e+2<br>or $-\infty/+\infty$                 |
| LREAL           | LREAL            | -3.0765265556858878e+2 to 3.0825471555991674e+2<br>or $-\infty/+\infty$ |

## Additional Information

When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

## Precautions for Correct Use

- If the value of *In* is not a positive number, the value of *Out* is as shown below.

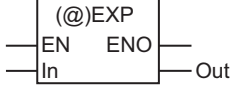
| Value of In     | Value of Out    |
|-----------------|-----------------|
| Negative number | Nonnumeric data |
| 0               | $-\infty$       |
| $+\infty$       | $+\infty$       |
| $-\infty$       | Nonnumeric data |
| Nonnumeric data | Nonnumeric data |

- If you pass an integer parameter to *In*, the data type is converted as follows:

| Data type of parameter that is passed to <i>ln</i> | Data type of <i>ln</i>       |
|--|------------------------------|
| 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/<br>FUN | Graphic expression  | ST expression |
|-------------|-------------------------------|------------|---|---------------|
| EXP         | Natural Exponential Operation | FUN        |  | Out:=EXP(In); |

## Variables

|     | 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 | ---  | ---     |

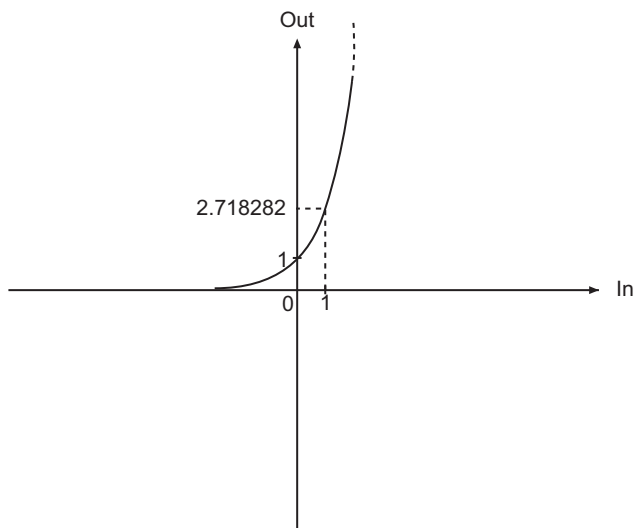
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

\*2. Negative numbers are excluded.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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    |  |      |     |    |        |

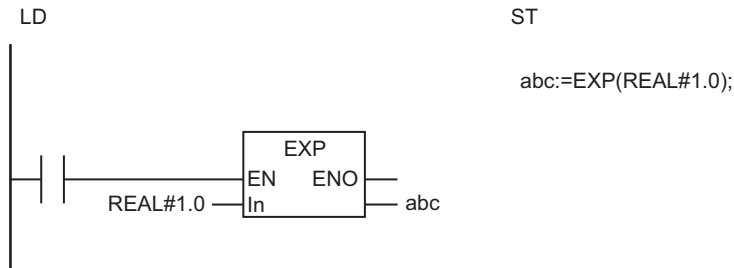
## Function

The EXP instruction returns the natural logarithm of  $In$  to the base  $e$ .

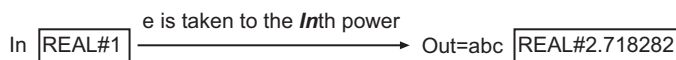




The following shows an example where *In* is REAL#1.0. The value of variable *abc* will be REAL#2.718282.



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 instruction, *EXPT (\*\*)* on page 2-226, to find powers of numbers with bases other than e.
- When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric 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 <i>In</i> | Value of <i>Out</i> |                 |
|--------------------|---------------------|-----------------|
|                    | NX1P2               | Other models    |
| 0.0                | 1.0                 | 1.0             |
| $+\infty$          | $+\infty$           | Nonnumeric data |
| $-\infty$          | 0.0                 | Nonnumeric data |
| 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 <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| 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/<br>FUN | Graphic expression | ST expression                          |
|-------------|----------------|------------|--------------------|--|
| EXPT (**)   | Exponentiation | FUN        |                    | Out:=EXPT(In, Pwr);<br>Out:=In ** Pwr; |

## Variables

|     | Meaning            | I/O    | Description                                  | Valid range           | Unit | Default |
|-----|--------------------|--------|--|-----------------------|------|---------|
| In  | Base number        | Input  | Base number<br>(e.g., 5 for 5 <sup>2</sup> ) | Depends on data type. | ---  | *1      |
| Pwr | Exponent           |        | Exponent<br>(e.g., 2 for 5 <sup>2</sup> )    |                       |      |         |
| Out | Calculation result | Output | Calculation result                           | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

## The Instruction of LD and the EXPT Instruction in ST

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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

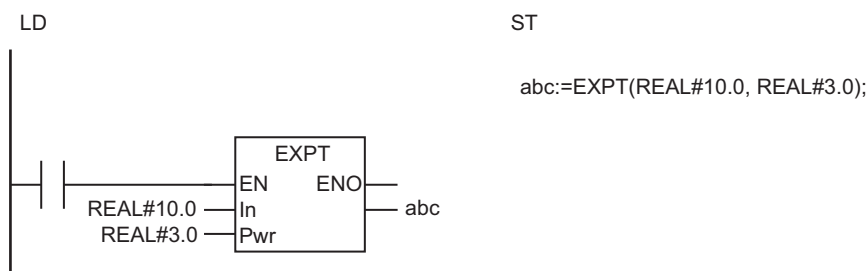
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|     | 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   |      |      |     |    |        |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   | OK   | OK   |      |      |     |    |        |

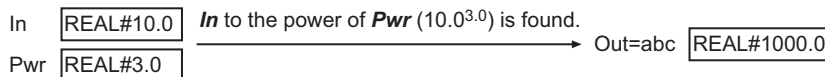
## Function

The EXPT (\*\*) instruction raises base number *In* to exponent *Pwr* to find  $In^{Pwr}$ .

The following shows an example where *In* and *Pwr* are REAL#10.0 and REAL#3.0, respectively. The value of variable *abc* will be REAL#1000.0.



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.



## Differences in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in ladder diagram programming, or the \*\* operator is used in ST. The following table gives the differences in specifications. The specifications of the EXPT instruction and the \*\* instruction in ladder diagram programming and the EXPT function in ST programming are exactly the same.

| Item  | EXPT functions in ladder dia-<br>gram and ST | ** operator in ST |
|---|--|-------------------|
| Presence of <i>EN</i> and <i>ENO</i>  | Present                                      | None              |
| Number of data processing bits if <i>In</i> and <i>Pwr</i> are integer data | 32 or 64 <sup>*1</sup>                       | 64 <sup>*2</sup>  |

\*1. Operations are performed with REAL or LREAL data type, whichever is smaller. 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 one SINT value is raised to the power of another SINT value, 64-bit processing is performed.

## Additional Information

- Use the instruction, *EXP* on page 2-224, to find powers for base e.

- When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

## Precautions for Correct Use

- If the absolute value of a 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 a calculation with the **\*\*** operator.
- If an underflow or overflow occurs in the calculation with the **\*\*** operator, 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 will not occur.
- For the EXPT instruction and **\*\*** instruction in ladder programming and the EXPT function 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 <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| USINT, UINT, SINT, or INT                          | REAL                         |
| UDINT or DINT                                      | LREAL                        |
| ULINT or LINT                                      | A building error will occur. |

- With the **\*\*** operator, integer variables are calculated as real number variables, even when they are set as operands, if you select **1.15** or earlier version for **Version** in the Select Device Area of the Project Properties Dialog Box on the Sysmac Studio for an NX701 CPU Unit or an NJ-series CPU Unit.

If a rounding error is included in a calculation result, the result value may not be an intended value because it is rounded to the integer.

Use the EXPT and TO\_\*\* (Integer Conversion Group) instructions together to round the value to an integer.

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

|                 |                         | In              |             |                 |             |                 |                 |    |             |                      | Nonnumeric data |
|-----------------|-------------------------|-----------------|-------------|-----------------|-------------|-----------------|-----------------|----|-------------|----------------------|-----------------|
|                 |                         | + ∞             | 1 to +∞     | 1               | 0 to 1      | 0               | -1 to 0         | -1 | -1 to -∞    | -∞                   |                 |
| Pwr             | + ∞                     | + ∞             | + ∞         | 1               | 0           | 1               | 0               | 1  | + ∞         | 1                    | Nonnumeric data |
|                 | Positive even number    |                 |             |                 |             |                 | Number *1*2     | 1  | Number *1*2 |                      | Nonnumeric data |
|                 | Positive odd number     | + ∞             | Number *1*2 | 1               | Number *1*2 | 0               | Number *2*3     | -1 | Number *2*3 | + ∞                  |                 |
|                 | Positive decimal number |                 |             |                 |             |                 | Nonnumeric data |    |             |                      |                 |
|                 | 0                       | 1               | 1           |                 |             | 1               | 1               |    |             | 1                    | 1               |
|                 | Negative even number    |                 |             |                 |             |                 | Number *1*2     | 1  | Number *1*2 |                      | Nonnumeric data |
|                 | Negative odd number     | 0               | Number *1*2 | 1               | Number *1*2 | + ∞             | Number *2*3     | -1 | Number *2*3 | 0                    |                 |
|                 | Negative decimal number |                 |             |                 |             |                 | Nonnumeric data |    |             |                      |                 |
|                 | -∞                      | 0               | 0           | 1               | + ∞         | + ∞             | + ∞             | 1  | 0           | 0                    | Nonnumeric data |
| Nonnumeric data | 1                       | Nonnumeric data | 1           | Nonnumeric data | 1           | Nonnumeric data |                 |    | 1           | Nonnumeric data<br>1 |                 |

- \*1. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be +∞.
- \*2. If the calculation result is too close to 0 to express with the data type of *Out*, or if it is a subnormal number, the value of *Out* will be 0.
- \*3. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be -∞.

● The *EXPT* Function for the NX1P2 CPU Unit

|                 |                         | In              |             |                 |             |                 |                 |    |             |                      |                 |   |
|-----------------|-------------------------|-----------------|-------------|-----------------|-------------|-----------------|-----------------|----|-------------|----------------------|-----------------|---|
|                 |                         | +∞              | 1 to +∞     | 1               | 0 to 1      | 0               | -1 to 0         | -1 | -1 to -∞    | -∞                   | Nonnumeric data |   |
| Pwr             | +∞                      | +∞              | +∞          | 1               | 0           | 1               | 0               | 1  | +∞          | 1                    | Nonnumeric data |   |
|                 | Positive even number    |                 |             |                 |             |                 | Number *1*2     | 1  | Number *1*2 | +∞                   | Nonnumeric data |   |
|                 | Positive odd number     | +∞              | Number *1*2 | 1               | Number *1*2 | 0               | Number *2*3     | -1 | Number *2*3 | -∞                   |                 |   |
|                 | Positive decimal number |                 |             |                 |             |                 | Nonnumeric data |    |             | +∞                   |                 |   |
|                 | 0                       | 1               | 1           |                 |             |                 | 1               | 1  |             |                      | 1               | 1 |
|                 | Negative even number    |                 |             |                 |             |                 | Number *1*2     | 1  | Number *1*2 | 0                    | Nonnumeric data |   |
|                 | Negative odd number     | 0               | Number *1*2 | 1               | Number *1*2 | +∞              | Number *2*3     | -1 | Number *2*3 | -0                   |                 |   |
|                 | Negative decimal number |                 |             |                 |             |                 | Nonnumeric data |    |             | 0                    |                 |   |
|                 | -∞                      | 0               | 0           | 1               | +∞          | +∞              | +∞              | 1  | 0           | 0                    | Nonnumeric data |   |
| Nonnumeric data | 1                       | Nonnumeric data | 1           | Nonnumeric data | 1           | Nonnumeric data |                 |    | 1           | Nonnumeric data<br>1 |                 |   |

\*1. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be +∞.

\*2. If the calculation result is too close to 0 to express with the data type of *Out*, or if it is a subnormal number, the value of *Out* will be 0.

\*3. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be -∞.

● The \*\* Operator

|                 |                         | In              |               |   |             |                 |                 |    |               |                      |                 |
|-----------------|-------------------------|-----------------|---------------|---|-------------|-----------------|-----------------|----|---------------|----------------------|-----------------|
|                 |                         | +∞              | 1 to +∞       | 1 | 0 to 1      | 0               | -1 to 0         | -1 | -1 to -∞      | -∞                   | Nonnumeric data |
| Pwr             | +∞                      | +∞              | +∞            | 1 | 0           | 1               | 0               | 1  | +∞            | 1                    | Nonnumeric data |
|                 | Positive even number    |                 |               |   |             |                 | Number *1*2     | 1  | Number *1*2*3 | +∞                   | Nonnumeric data |
|                 | Positive odd number     | +∞              | Number *1*2*3 | 1 | Number *1*2 | 0               | Number *2*4     | -1 | Number *2*3*4 | -∞                   |                 |
|                 | Positive decimal number |                 |               |   |             |                 | Nonnumeric data |    |               | +∞                   |                 |
|                 | 0                       | 1               | 1             |   |             | 1               | 1               |    |               | 1                    | 1               |
|                 | Negative even number    |                 |               |   |             |                 | Number *1*2     | 1  | Number *1*2   | 0                    | Nonnumeric data |
|                 | Negative odd number     | 0               | Number *1*2   | 1 | Number *1*2 | +∞*5            | Number *2*4     | -1 | Number *2*4   | -0                   |                 |
|                 | Negative decimal number |                 |               |   |             |                 | Nonnumeric data |    |               | 0                    |                 |
|                 | -∞                      | 0               | 0             | 1 | +∞          | +∞              | +∞              | 1  | 0             | 0                    | Nonnumeric data |
| Nonnumeric data | 1                       | Nonnumeric data |               |   | 1           | Nonnumeric data |                 |    | 1             | Nonnumeric data<br>1 |                 |

- \*1. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be +∞.
- \*2. If the calculation result is too close to 0 to express with the data type of *Out*, or if it is a subnormal number, the value of *Out* will be 0.
- \*3. If both *In* and *Pwr* are integer data, *Out* will contain an undefined value when the calculation result exceeds the valid value range of the data type of *Out*.
- \*4. If the calculation result exceeds the valid value range of the data type of *Out*, the value of *Out* will be -∞.
- \*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/<br>FUN | Graphic expression | ST expression |
|-------------|-----------|------------|--------------------|---------------|
| Inc         | Increment | FUN        |                    | Inc(InOut);   |
| Dec         | Decrement | FUN        |                    | Dec(InOut);   |

## Variables

|       | 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             | ---  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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

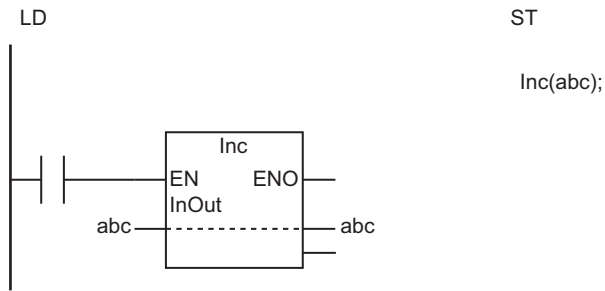
### Inc

The Inc instruction increments target data *InOut*. If the result value exceeds the maximum value of *InOut*, it returns to the minimum value.

#### ● Example for the Inc Instruction

The following shows an example where variable *abc* is passed to *InOut* for the Inc instruction. If the value of variable *abc* is INT#4, it will change to INT#5 after the instruction is executed.





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.



## Dec

The Dec instruction decrements target data *InOut*. If the result value exceeds the minimum value of *InOut*, it returns to the maximum value.

### Precautions for Correct Use

Return value *Out* is not used when these instructions are used in ST.

# Rand

The Rand instruction generates pseudorandom numbers.

| Instruction | Name          | FB/<br>FUN | Graphic expression | ST expression                      |
|-------------|---------------|------------|--------------------|------------------------------------|
| Rand        | Random Number | FB         |                    | Rand_instance(Execute, Seed, Rnd); |

## Variables

|      | Meaning               | I/O    | Description                                | Valid range           | Unit | Default |
|------|-----------------------|--------|--|-----------------------|------|---------|
| Seed | Random number pattern | Input  | Random number pattern<br>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. 0.0000000000000000e+0 to 1.0000000000000000e+0

|      | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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                   |       |  |      |     |    |        |

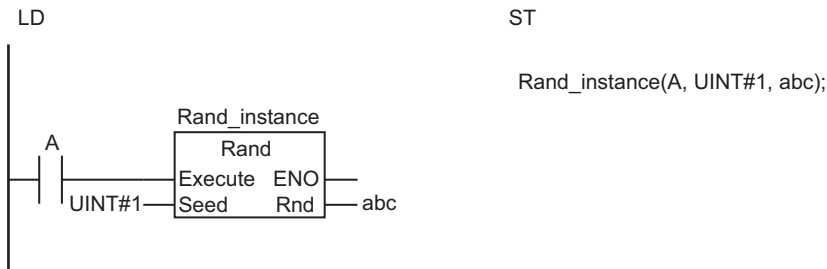
## Function

The Rand instruction specifies random number *Rnd*. The value of *Rnd* is different each time the instruction is executed.

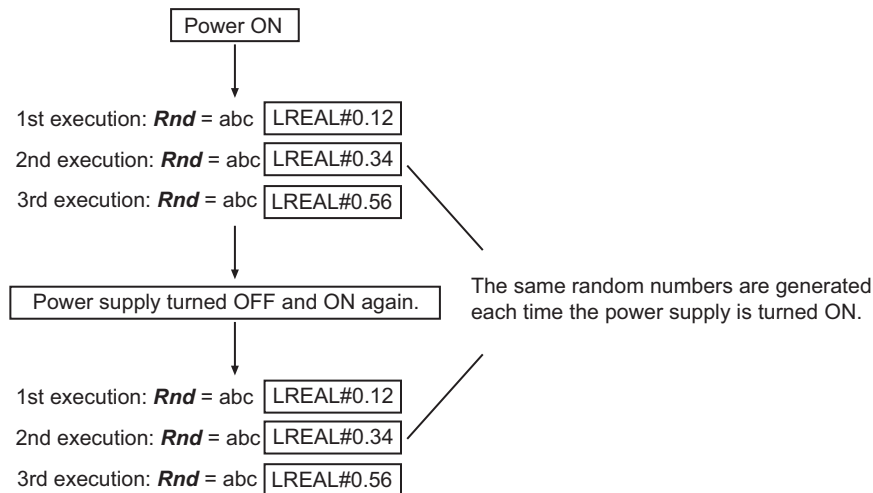
Random number pattern *Seed* specifies the random number series. If the value of *Seed* is the same, the same random number series is generated each time the power supply is turned ON. This allows you to generate a reproducible series of random numbers.

If the value of *Seed* is 0, irreproducible random numbers 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 shows a programming example where *Seed* is UINT#1. The value of *Seed* is not 0, so reproducible random numbers are generated.



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. Perform 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/<br>FUN | Graphic expression | ST expression                   |
|-------------|----------------|------------|--------------------|---------------------------------|
| AryAdd      | Array Addition | FUN        |                    | AryAdd(In1, In2, Size, AryOut); |

## Variables

|                                 | Meaning                       | I/O    | Description                   | Valid range           | Unit | Default |
|---------------------------------|-------------------------------|--------|-------------------------------|-----------------------|------|---------|
| In1[] (array),<br>In2[] (array) | Array to process              | Input  | Array to process              | Depends on data type. | ---  | *1      |
| Size                            | Number of elements to process |        | Number of elements to process |                       |      | 1       |
| AryOut[] (array)                | Calculation results array     | In-out | Calculation results array     | Depends on data type. | ---  | ---     |
| Out                             | Return value                  | Output | Always TRUE                   | TRUE only             | ---  | ---     |

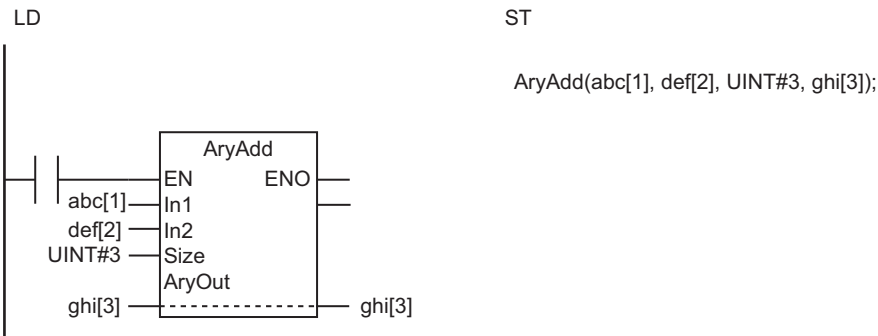
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 each pair of corresponding *Size* elements in In1[] and In2[], which start from In1[0] and In2[0], respectively. It outputs each addition result to the corresponding element of AryOut[] (calculation results array).

The following shows an example where *Size* is UINT#3.



|             |               |      |               |      |                  |
|-------------|---------------|------|---------------|------|------------------|
| Size=UINT#3 | In1[0]=abc[1] | +    | In2[0]=def[2] | →    | AryOut[0]=ghi[3] |
|             | 1234          | +    | 2345          | →    | 3579             |
|             | 2345          | +    | 3456          | →    | 5801             |
| 3456        | +             | 4567 | →             | 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 calculation results exceed the valid value range of AryOut[], the results will be illegal values. This will not result in an error. Data in the memory area adjacent to those elements will not be corrupted.
- The values in AryOut[] do not change if the value of Size is 0.
- Return value Out is not used when this instruction is used in ST.
- An error will occur in the following case. ENO will be FALSE, and AryOut[] will not change.
  - a) The value of Size exceeds the array range of either In1[], In2[], or AryOut[].

# AryAddV

The AryAddV instruction adds the same value to specified elements of an array.

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression                    |
|-------------|----------------------|------------|--------------------|----------------------------------|
| AryAddV     | Array Value Addition | FUN        |                    | AryAddV(In1, In2, Size, AryOut); |

## Variables

|                  | Meaning                | I/O    | Description                              | Valid range           | Unit | Default |
|------------------|------------------------|--------|--|-----------------------|------|---------|
| In1[] (array)    | Addition array         | Input  | Addition array                           | Depends on data type. | ---  | *1      |
| 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             | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

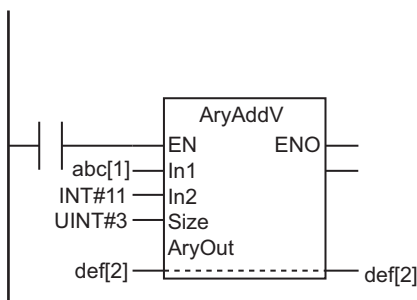
|                  | Boo<br>lean                          | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|------------------|--------------------------------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|                  | 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 the same data type as In1[]. |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Size             |                                      |             |      |       |       | OK       |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| AryOut[] (array) | Must be the same data type as In1[]  |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out              | OK                                   |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

## Function

The AryAddV instruction adds value to add *In2* to each of *Size* elements of addition array *In1*[], which starts from *In1*[0]. It outputs the addition result to each corresponding element of *AryOut*[] (addition results array).

The following shows an example where *In2* and *Size* are INT#11 and UINT#3, respectively.

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*[]. Otherwise, a building error will occur.
- If addition results exceed the valid value range of *AryOut*[], the elements of *AryOut*[] will contain illegal values. This will not result in an error. Data in the memory area adjacent to those elements will not be corrupted.
- The values in *AryOut*[] do not change if the value of *Size* is 0.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *AryOut*[] will not change.
  - a) The value of *Size* exceeds the array range of *In1*[] or *AryOut*[].

# ArySub

The ArySub instruction subtracts corresponding elements of two arrays.

| Instruction | Name              | FB/<br>FUN | Graphic expression | ST expression                   |
|-------------|-------------------|------------|--------------------|---------------------------------|
| ArySub      | Array Subtraction | FUN        |                    | ArySub(In1, In2, Size, AryOut); |

## Variables

|                  | Meaning                   | I/O    | Description                        | Valid range           | Unit | Default |
|------------------|---------------------------|--------|------------------------------------|-----------------------|------|---------|
| In1[] (array)    | Minuend array             | Input  | Minuend array                      | Depends on data type. | ---  | *1      |
| In2[] (array)    | Subtrahend array          |        | Subtrahend array                   |                       |      |         |
| Size             | Number of elements        |        | Number of elements 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             | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean                         | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 the same data type as In1[] |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Size             |                                     |             |      |       |       | OK       |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| AryOut[] (array) | Must be the same data type as In1[] |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out              | OK                                  |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

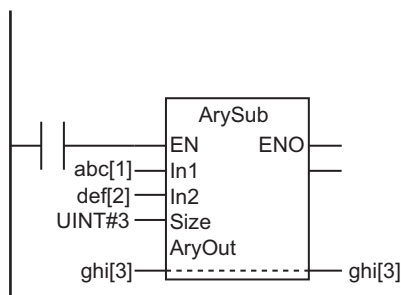
## Function

The ArySub instruction subtracts each element value of *Size* elements in subtrahend array In2[] from the corresponding element of minuend array In1[]. It outputs each subtraction result to the corresponding element of AryOut[] (subtraction results array).

The following shows an example where *Size* is UINT#3.



LD



ST

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

|             |               |    |   |               |   |   |                  |    |
|-------------|---------------|----|---|---------------|---|---|------------------|----|
| Size=UINT#3 | In1[0]=abc[1] | 12 | - | In2[0]=def[2] | 1 | → | AryOut[0]=ghi[3] | 11 |
|             | In1[1]=abc[2] | 23 | - | In2[1]=def[3] | 2 | → | AryOut[1]=ghi[4] | 21 |
|             | In1[2]=abc[3] | 34 | - | In2[2]=def[4] | 3 | → | AryOut[2]=ghi[5] | 31 |

## Precautions for Correct Use

- Use the same data type for In1[], In2[], and AryOut[]. If they are different, a building error will occur.
- If subtraction results exceed the valid value range of AryOut[], the elements of AryOut[] will contain illegal values. This will not result in an error. Data in the memory area adjacent to those elements will not be corrupted.
- The values in AryOut[] do not change if the value of Size is 0.
- Return value Out is not used when this instruction is used in ST.
- An error will occur in the following case. ENO will be FALSE, and AryOut[] will not change.
  - a) The value of Size exceeds the array range of either In1[], In2[], or AryOut[].

# ArySubV

The ArySubV instruction subtracts the same value from specified elements of an array.

| Instruction | Name                    | FB/<br>FUN | Graphic expression | ST expression                    |
|-------------|-------------------------|------------|--------------------|----------------------------------|
| ArySubV     | Array Value Subtraction | FUN        |                    | ArySubV(In1, In2, Size, AryOut); |

## Variables

|                  | Meaning                   | I/O    | Description                                 | Valid range           | Unit | Default |
|------------------|---------------------------|--------|---|-----------------------|------|---------|
| In1[] (array)    | Minuend array             | Input  | Minuend array                               | Depends on data type. | ---  | *1      |
| 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             | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

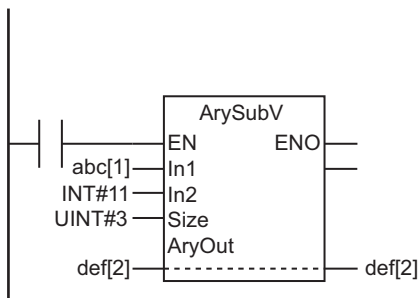
|                  | Boo<br>lean                          | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------------------|--------------------------------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                  | 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 the same data type as In1[]. |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Size             |                                      |             |      |       |       | OK       |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| AryOut[] (array) | Must be the same data type as In1[]  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out              | OK                                   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The ArySubV instruction subtracts subtrahend *In2* from each element value of *Size* elements of minuend array *In1*[], which starts from *In1*[0]. It outputs each subtraction result to the corresponding element of *AryOut*[] (subtraction results array).

The following shows an example where *In2* and *Size* are INT#11 and UINT#3, respectively.

LD



ST

ArySubV(abc[1], INT#11, UINT#3, def[2]);

|             |   |    |   |            |   |                  |    |
|-------------|---|----|---|------------|---|------------------|----|
| Size=UINT#3 | $\left[ \begin{array}{l} \text{In1}[0]=\text{abc}[1] \\ \text{In1}[1]=\text{abc}[2] \\ \text{In1}[2]=\text{abc}[3] \end{array} \right.$ | 22 | - | In2=INT#11 | → | AryOut[0]=def[2] | 11 |
|             |   | 33 | - | In2=INT#11 | → | AryOut[1]=def[3] | 22 |
|             |   | 44 | - | In2=INT#11 | → | AryOut[2]=def[4] | 33 |

## Precautions for Correct Use

- Use the same data type for *In1*[], *In2*, and *AryOut*[]. Otherwise, a building error will occur.
- If subtraction results exceed the valid value range of *AryOut*[], the elements of *AryOut*[] will contain illegal values. This will not result in an error. Data in the memory area adjacent to those elements will not be corrupted.
- The values in *AryOut*[] do not change if the value of *Size* is 0.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *AryOut*[] will not change.
  - a) The value of *Size* exceeds the array range of *In1*[] or *AryOut*[].

# AryMean

The AryMean instruction calculates the average of the elements of an array.

| Instruction | Name       | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|------------|------------|--------------------|---------------------------|
| AryMean     | Array Mean | FUN        |                    | Out := AryMean(In, Size); |

## Variables

|              | Meaning                       | I/O    | Description             | Valid range           | Unit | Default |
|--------------|-------------------------------|--------|-------------------------|-----------------------|------|---------|
| In[] (array) | Array to process              | Input  | Array to process        | Depends on data type. | ---  | *1      |
| Size         | Number of elements to process |        | Number of In[] elements |                       |      | 1       |
| Out          | Calculation result            | Output | Calculation result      | Depends on data type. | ---  | ---     |

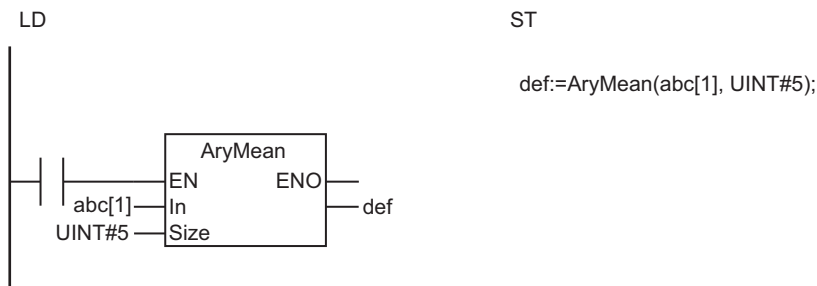
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

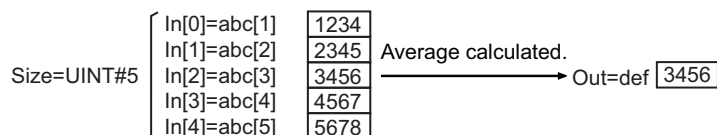
|              | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|--------------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|              | 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[], which starts from In[0].

The following shows an example where *Size* is UINT#5.





## Precautions for Correct Use

- Refer to the descriptions of the instructions, *ADD (+)* on page 2-179, *SUB (-)* on page 2-187, *MUL (\*)* on page 2-194, and *DIV (/)* on page 2-202, for calculation results when the value of *In[]* is positive infinity, negative infinity, or nonnumeric data.
- If *In[]* and *Out* are integers, the average value is truncated to an integer.
- If the data types of *In[]* and *Out* are different, make sure that the valid value range of *Out* accommodates the valid value range of *In[]*.
- If a calculation result exceeds the valid value range of *Out*, *Out* will contain an illegal value. This will not result in an error.
- If an intermediate value in the calculation process exceeds the valid value range of *In[]*, *Out* will contain an illegal value. This will not result in an error.
- If the value of *Size* is 0, the value of *Out* is 0.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* exceeds the array range of *In[]*.

# ArySD

The ArySD instruction calculates standard deviation of the elements of an array.

| Instruction | Name                                     | FB/<br>FUN | Graphic expression | ST expression         |
|-------------|--|------------|--------------------|-----------------------|
| ArySD       | Array Element<br>Standard Devi-<br>ation | FUN        |                    | Out:=ArySD(In, Size); |

## Variables

|              | Meaning            | I/O    | Description                               | Valid range           | Unit | Default |
|--------------|--------------------|--------|---|-----------------------|------|---------|
| In[] (array) | Array to process   | Input  | Array to process                          | Depends on data type. | ---  | *1      |
| Size         | Number of elements |        | Number of elements of In[] for conversion |                       |      | 2       |
| Out          | Standard deviation | Output | Standard deviation                        | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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    |  |      |     |    |        |

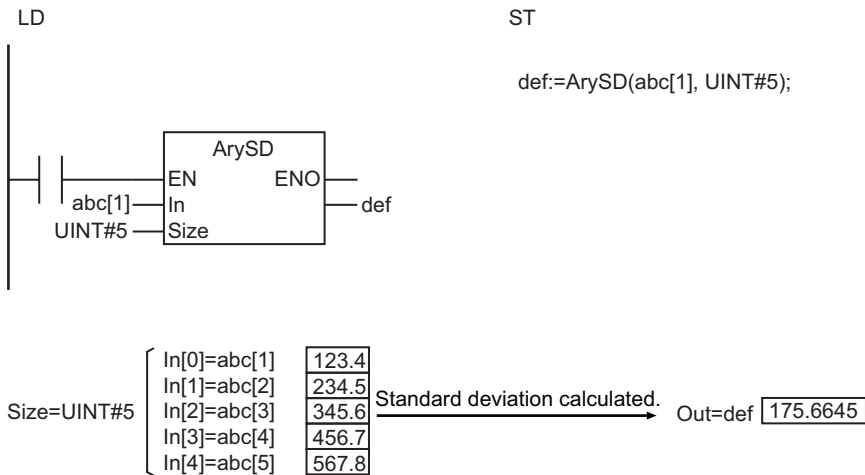
## Function

The ArySD instruction calculates the standard deviation of *Size* elements of array to process In[], which starts from In[0].

$$\text{Standard deviation} = \sqrt{\frac{\sum_i (\text{In}[i] - \text{InM})^2}{\text{Size} - 1}}$$

*i*: Subscript of In[], 0 to *Size* - 1  
InM: Average value of In[0] to In[*Size* - 1]

The following shows an example where *Size* is 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 value range of *In*[], *Out* will contain an illegal value. This will not result in an error.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* exceeds the array range of *In*[].

# ModReal

The ModReal instruction calculates the remainder of real number division.

| Instruction | Name                        | FB/<br>FUN | Graphic expression | ST expression           |
|-------------|-----------------------------|------------|--------------------|-------------------------|
| ModReal     | Real Number Modulo-division | FUN        |                    | Out:=ModReal(In1, In2); |

## Variables

|     | Meaning   | I/O    | Description | Valid range           | Unit | Default |
|-----|-----------|--------|-------------|-----------------------|------|---------|
| In1 | Dividend  | Input  | Dividend    | Depends on data type. | ---  | *1      |
| In2 | Divisor   |        | Divisor     |                       |      |         |
| Out | Remainder | Output | Remainder   | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.

This instruction performs the calculation with the following formula.

$$Out = In1 - (In1/In2)*In2$$

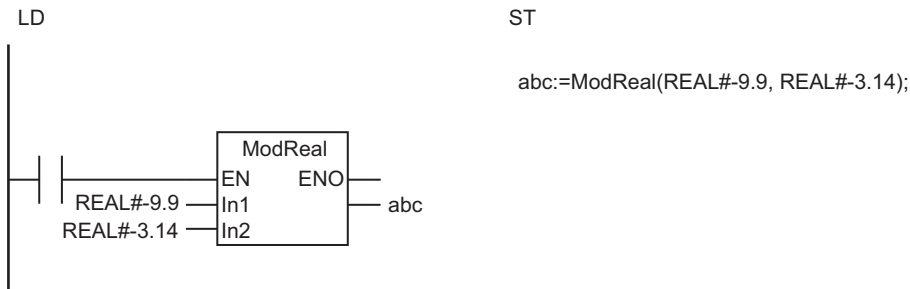
Decimal places are truncated in the division operation.

Examples with the values of *In1*, *In2* and *Out* are given in the following table.

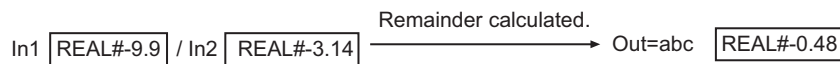
| Value of <i>In1</i> | Value of <i>In2</i> | Value of <i>Out</i> |
|---------------------|---------------------|---------------------|
| 9.9                 | 3.14                | 0.48                |
| 9.9                 | -3.14               | 0.48                |
| -9.9                | 3.14                | -0.48               |
| -9.9                | -3.14               | -0.48               |

The following shows an example where *In1* and *In2* are REAL#-9.9 and REAL#-3.14, respectively. The value of variable *abc* will be REAL#-0.48.





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

When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.

## Precautions for Correct Use

- The following table shows the values of *Out* for different combinations of *In1* and *In2* values.

|     |                 | In1             |                        |                 |                 |                 |
|-----|-----------------|-----------------|------------------------|-----------------|-----------------|-----------------|
|     |                 | 0               | Number                 | $+\infty$       | $-\infty$       | 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 |
|     | $+\infty$       | 0               | Value of $In1$         | Nonnumeric data | Nonnumeric data | Nonnumeric data |
|     | $-\infty$       | 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 <i>In1</i> or <i>In2</i> | Data type of <i>In1</i> or <i>In2</i> |
|---|---------------------------------------|
| USINT, UINT, SINT, or INT   | REAL                                  |
| UDINT or DINT   | LREAL                                 |
| ULINT or LINT   | A building error will occur.          |



## Additional Information

- When you calculate real numbers, use the instruction, *CheckReal* on page 2-252, to check if *Out* is positive infinity, negative infinity, or non-numeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

| Data type of parameter that is passed to <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| 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/<br>FUN | Graphic expression | ST expression                                 |
|-------------|-------------------|------------|--------------------|---|
| CheckReal   | Real Number Check | FUN        |                    | CheckReal(In, Nan, PosInfinite, NegInfinite); |

## Variables

|             | Meaning                        | I/O    | Description   | Valid range           | Unit | Default |
|-------------|--------------------------------|--------|---|-----------------------|------|---------|
| In          | Real number                    | Input  | Real number   | Depends on data type. | ---  | *1      |
| Out         | Return value                   | Output | Always TRUE   | TRUE only             | ---  | ---     |
| Nan         | Nonnumeric data check result   |        | TRUE: Nonnumeric data<br>FALSE: Not nonnumeric data     | Depends on data type. |      |         |
| PosInfinite | Positive infinity check result |        | TRUE: Positive infinity<br>FALSE: Not positive infinity |                       |      |         |
| NegInfinite | Negative infinity check result |        | TRUE: Negative infinity<br>FALSE: Not negative infinity |                       |      |         |

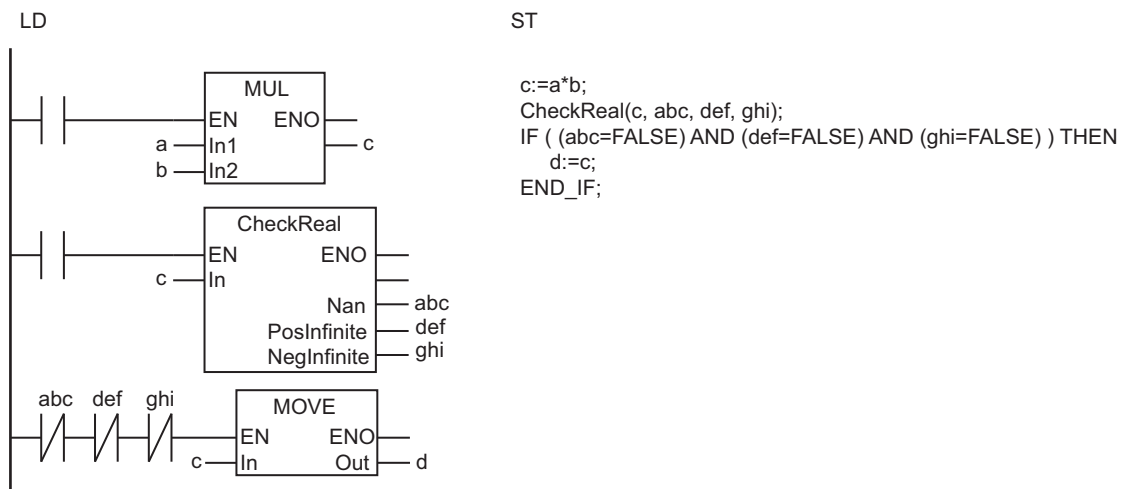
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|             | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

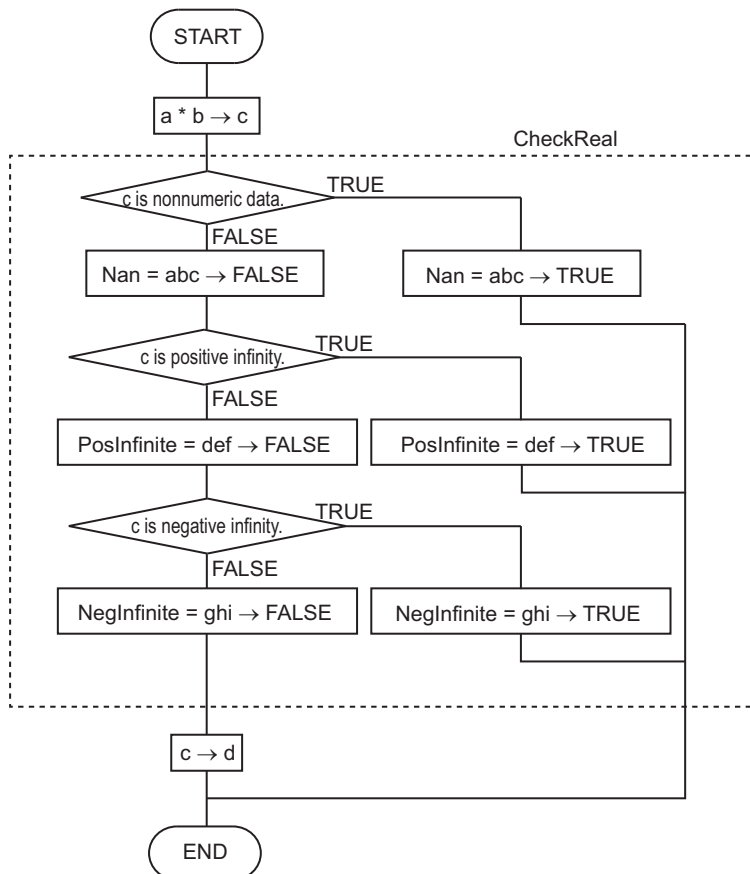
## Function

The CheckReal instruction checks a real number *In* to see if it is nonnumeric data, positive infinity, or negative infinity. It outputs the result 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*.



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 check if the result is nonnumeric data, positive infinity, or negative infinity.

## Precautions for Correct Use

- Return value *Out* is not used when this 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 <i>In</i> | Data type of <i>In</i>       |
|--|------------------------------|
| USINT, UINT, SINT, or INT                          | REAL                         |
| UDINT or DINT                                      | LREAL                        |
| ULINT or LINT                                      | A building error will occur. |

## BCD Conversion Instructions

| Instruction   | Name   | Page       |
|---------------|--|------------|
| **_BCD_TO_*** | BCD-to-Unsigned Integer Conversion Group           | page 2-256 |
| **_TO_BCD_*** | Unsigned Integer-to-BCD Conversion Group           | page 2-259 |
| BCD_TO_**     | BCD Data Type-to-Unsigned Integer Conversion Group | page 2-262 |
| BCDsToBin     | Signed BCD-to-Signed Integer Conversion            | page 2-265 |
| BinToBCDs_**  | Signed Integer-to-BCD Conversion Group             | page 2-268 |
| AryToBCD      | Array BCD Conversion                               | page 2-271 |
| AryToBin      | Array Unsigned Integer Conversion                  | page 2-273 |

# \*\* \_BCD\_TO\_ \*\*

These instructions convert BCD bit strings into unsigned integers.

| Instruction | Name                                     | FB/<br>FUN | Graphic expression   | ST expression  |
|-------------|--|------------|--|--|
| **_BCD_TO_* | BCD-to-Unsigned Integer Conversion Group | FUN        | <p>*** must be a bit string data type. ***** must be an integer data type.</p> | Out:=**_BCD_TO_*** (In);<br>*** must be a bit string data type.<br>***** must be an integer data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-257, below, for details.

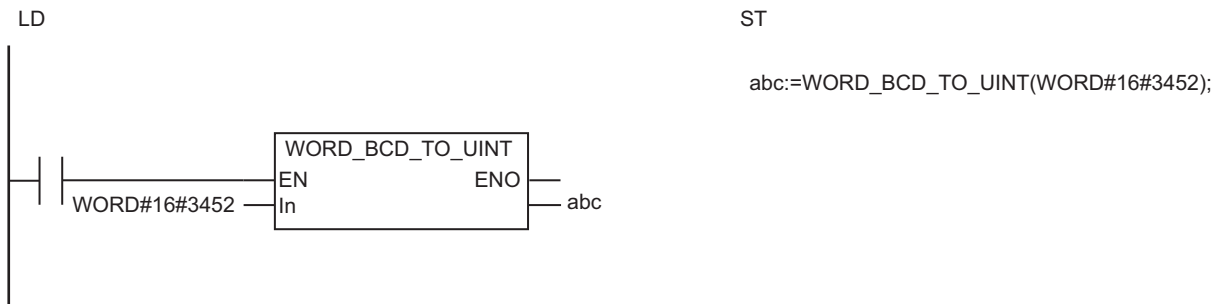
|     | Boo<br>lean | Bit strings |      |      |       | Integers |       |      |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|------|-------|----------|-------|------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|
|     |             | 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

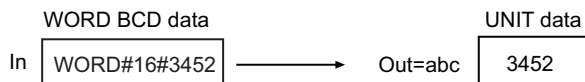
These instructions convert data to convert *In* (which must be a BCD bit string) into an unsigned integer.

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.







## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>          | Valid range for <i>Out</i> |
|------------------------|-------------------------|------------------------------------|----------------------------|
| 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                   | 16#0000 to 16#0127 (BCD)           | 0 to 127                   |
|                        | SINT                    | 16#0000 to 16#9999 (BCD)           | 0 to 9999                  |
|                        | INT                     |                                    |                            |
|                        | 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                    |                                    |                            |

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                                 | Valid range for <i>Out</i> |
|------------------------|-------------------------|---|----------------------------|
| LWORD                  | USINT                   | 16#0000_0000_0000_0000 to<br>16#0000_0000_0000_0255 (BCD) | 0 to 255                   |
|                        | UINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0000_0006_5535 (BCD) | 0 to 65535                 |
|                        | UDINT                   | 16#0000_0000_0000_0000 to<br>16#0000_0042_9496_7295 (BCD) | 0 to 4294967295            |
|                        | ULINT                   | 16#0000_0000_0000_0000 to<br>16#9999_9999_9999_9999 (BCD) | 0 to 9999999999999999      |
|                        | SINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0000_0000_0127 (BCD) | 0 to 127                   |
|                        | INT                     | 16#0000_0000_0000_0000 to<br>16#0000_0000_0003_2767 (BCD) | 0 to 32767                 |
|                        | DINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0021_4748_3647 (BCD) | 0 to 2147483647            |
|                        | LINT                    | 16#0000_0000_0000_0000 to<br>16#9999_9999_9999_9999 (BCD) | 0 to 9999999999999999      |

### Additional Information

- To convert a BCD bit string to an integer, use the instruction, *BCD\_TO\_\*\** on page 2-262.
- To convert an integer to a BCD bit string, use the instruction, *\*\*\_TO\_BCD\_\*\*\** on page 2-259.

### 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *In* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression  | ST expression  |
|--------------|--|------------|---|--|
| **_TO_BCD_** | Unsigned Integer-to-BCD Conversion Group | FUN        | <p>**** must be an integer data type. ***** must be a bit string data type.</p> | Out:=**_TO_BCD_** (In);<br><br>**** must be an integer data type.<br>***** must be a bit string data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

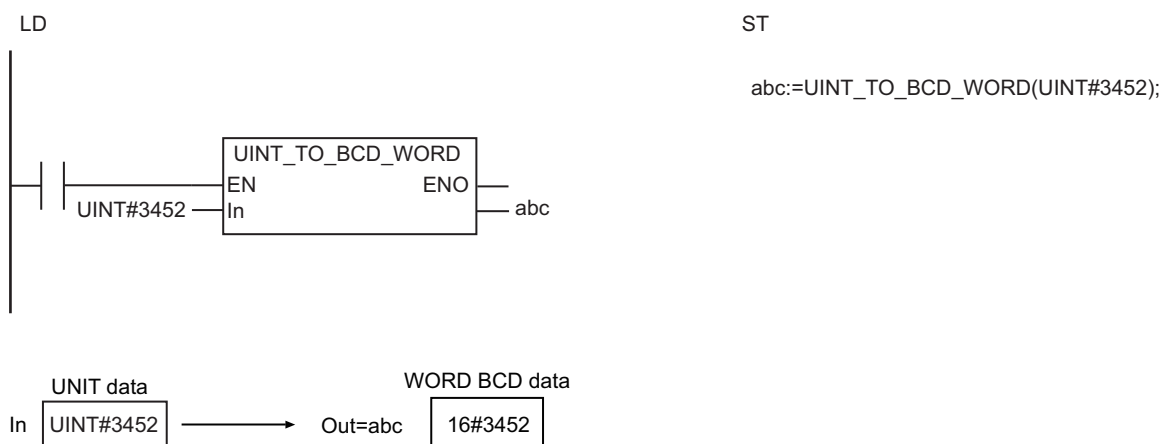
\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-260, below, for details.

|     | 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*.



## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> | Valid range for <i>Out</i>                                |                          |
|------------------------|-------------------------|---------------------------|---|--------------------------|
| 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#0000_0255 (BCD)                        |                          |
|                        | LWORD                   |                           | 16#0000_0000_0000_0000 to<br>16#0000_0000_0000_0255 (BCD) |                          |
| UINT                   | BYTE                    |                           | 0 to 99   | 16#00 to 16#99 (BCD)     |
| UINT                   | 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<br>16#0000_0000_0006_5535 (BCD) |                          |
|                        | UDINT                   |                           | BYTE  | 0 to 99                  |
| UDINT                  | 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<br>16#0000_0042_9496_7295 (BCD) |                          |
|                        | ULINT                   | BYTE                      | 0 to 99   | 16#00 to 16#99 (BCD)     |
| ULINT                  | 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<br>16#9999_9999_9999_9999 (BCD) |                          |
|                        | SINT                    | BYTE                      | 0 to 99   | 16#00 to 16#99 (BCD)     |
| SINT                   | 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<br>16#0000_0000_0000_0127 (BCD) |                          |
|                        | INT                     |                           | BYTE  | 0 to 99                  |
| INT                    | 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<br>16#0000_0000_0003_2767 (BCD) |                          |
|                        | DINT                    |                           | BYTE  | 0 to 99                  |
| DINT                   | 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<br>16#0000_0021_4748_3647 (BCD) |                          |
|                        | LINT                    | BYTE                      | 0 to 99   | 16#00 to 16#99 (BCD)     |
| LINT                   | 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<br>16#9999_9999_9999_9999 (BCD) |                          |

## Additional Information

- To convert a specific BCD bit string to an integer, use the instruction, `**_BCD_TO_**` on page 2-256.
- To convert a BCD bit string to an integer, use the instruction, `BCD_TO_**` on page 2-262.

## 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *In* is outside the valid range.

# BCD\_TO\_\*\*

The BCD\_TO\_\*\* instruction converts BCD bit strings into unsigned integers.

| Instruction | Name   | FB/<br>FUN | Graphic expression                        | ST expression  |
|-------------|--|------------|---|--|
| BCD_TO_**   | BCD Data Type-to-Unsigned Integer Conversion Group | FUN        | <p>**** must be an integer data type.</p> | Out:=BCD_TO_** (In);<br>**** must be an integer data type. |

## Variables

|     | 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 *Valid Range* on page 2-263, below, for details.
- \*2. If you omit the input parameter, the default value is not applied. A building error will occur.

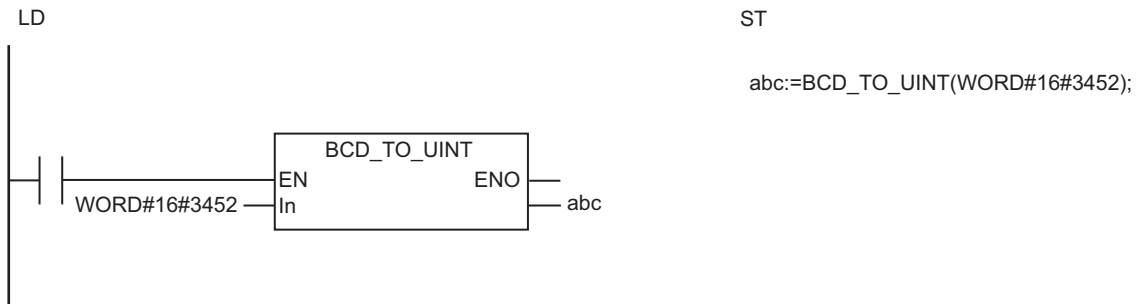
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|     | 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   |                      |      |  |      |      |     |    |        |

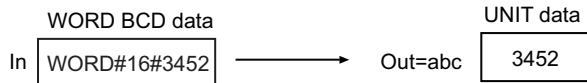
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>          | Valid range for <i>Out</i> |
|------------------------|-------------------------|------------------------------------|----------------------------|
| BYTE                   | USINT                   | 16#00 to 16#99 (BCD)               | 0 to 99                    |
|                        | UINT                    |                                    |                            |
|                        | UDINT                   |                                    |                            |
|                        | ULINT                   |                                    |                            |
|                        | SINT                    |                                    |                            |
|                        | INT                     |                                    |                            |
|                        | DINT                    |                                    |                            |
| WORD                   | LINT                    |                                    |                            |
|                        | 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                   |                         |                                    |                            |

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                                 | Valid range for <i>Out</i> |
|------------------------|-------------------------|---|----------------------------|
| LWORD                  | USINT                   | 16#0000_0000_0000_0000 to<br>16#0000_0000_0000_0255 (BCD) | 0 to 255                   |
|                        | UINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0000_0006_5535 (BCD) | 0 to 65535                 |
|                        | UDINT                   | 16#0000_0000_0000_0000 to<br>16#0000_0042_9496_7295 (BCD) | 0 to 4294967295            |
|                        | ULINT                   | 16#0000_0000_0000_0000 to<br>16#9999_9999_9999_9999 (BCD) | 0 to 9999999999999999      |
|                        | SINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0000_0000_0127 (BCD) | 0 to 127                   |
|                        | INT                     | 16#0000_0000_0000_0000 to<br>16#0000_0000_0003_2767 (BCD) | 0 to 32767                 |
|                        | DINT                    | 16#0000_0000_0000_0000 to<br>16#0000_0021_4748_3647 (BCD) | 0 to 2147483647            |
|                        | LINT                    | 16#0000_0000_0000_0000 to<br>16#9999_9999_9999_9999 (BCD) | 0 to 9999999999999999      |

### Additional Information

- To convert a specific BCD bit string to an integer, use the instruction, **\*\*\_BCD\_TO\_\*\*\*** on page 2-256.
- To convert an integer to a BCD bit string, use the instruction, **\*\*\_TO\_BCD\_\*\*\*** on page 2-259.

### 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *In* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression               |
|-------------|---|------------|--------------------|-----------------------------|
| BCDsToBin   | Signed BCD-to-Signed Integer Conversion | FUN        |                    | Out:=BCDsToBin(In, Format); |

## Variables

|        | Meaning            | I/O    | Description              | Valid range    | Unit | Default |
|--------|--------------------|--------|--------------------------|----------------|------|---------|
| In     | Data to convert    | Input  | Data to convert          | *1             | ---  | *2      |
| Format | Data format number |        | 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 *Valid Range* on page 2-266, below, for details.

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

|        | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-265 for the enumerators of the enumerated type <code>_eBCD_FORMAT</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out    | Must be a signed integer data type that is the same size as <i>In</i> .                                       |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

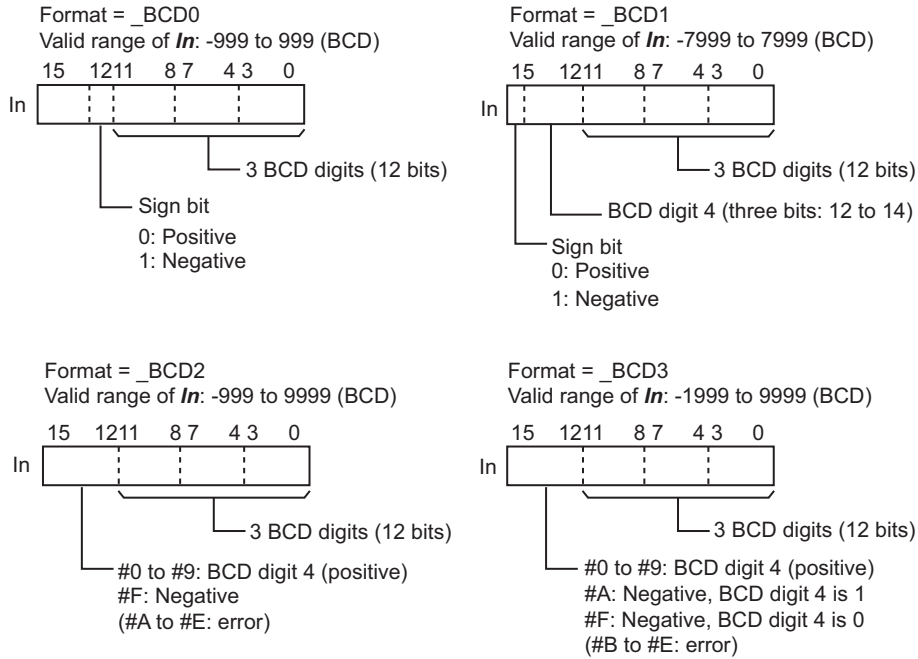
## Function

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*.



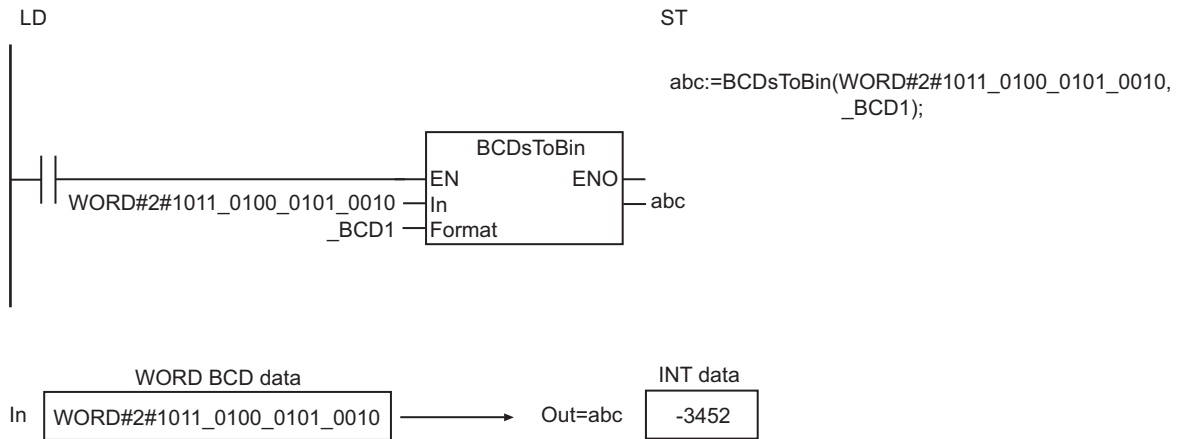
## Valid Range

The data types of *In* and *Out* should be of the same size. The valid ranges depend on the value of *Format*, as shown below.

|                         |                    | Value of <i>Format</i>                |                                       |                                       |                                       |
|-------------------------|--------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|
|                         |                    | <code>_BCD0</code>                    | <code>_BCD1</code>                    | <code>_BCD2</code>                    | <code>_BCD3</code>                    |
| Data type of <i>In</i>  | BYTE<br>↓<br>SINT  | -9 to 9                               | -79 to 79                             | -9 to 99                              | -19 to 99                             |
|                         | WORD<br>↓<br>INT   | -999 to 999                           | -7999 to 7999                         | -999 to 9999                          | -1999 to 9999                         |
| Data type of <i>Out</i> | DWORD<br>↓<br>DINT | -9999999 to 9999999                   | -79999999 to 79999999                 | -9999999 to 99999999                  | -19999999 to 99999999                 |
|                         | LWORD<br>↓<br>LINT | -9999999999999999 to 9999999999999999 | -7999999999999999 to 7999999999999999 | -9999999999999999 to 9999999999999999 | -1999999999999999 to 9999999999999999 |

## Notation Example

The following example is for when *In* is `WORD#2#1011_0100_0101_0010` and *Format* is `_BCD1`.



### Precautions for Correct Use

- The data types of *In* and *Out* should be of the same size.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Format* is *\_BCD0*, and the upper digit of *In* is 2 to F.
  - b) The value of *Format* is *\_BCD2*, and the upper digit of *In* is A to E.
  - c) The value of *Format* is *\_BCD3*, and the upper digit of *In* is B to E.
  - d) Except for the above conditions, any digit in *In* is A to F.
  - e) The value of *Format* is outside the valid range.

# BinToBCDs\_\*\*

These instructions convert signed integers to signed BCD bit strings.

| Instruction  | Name                                   | FB/<br>FUN | Graphic expression                          | ST expression   |
|--------------|--|------------|---|---|
| BinToBCDs_** | Signed Integer-to-BCD Conversion Group | FUN        | <p>**** must be a bit string data type.</p> | Out:=BinToBCDs(In, Format);<br>**** must be a bit string data type. |

## Variables

|        | Meaning            | I/O    | Description              | Valid range    | Unit | Default |
|--------|--------------------|--------|--------------------------|----------------|------|---------|
| In     | Data to convert    | Input  | Data to convert          | *1             | ---  | 0       |
| Format | Data format number |        | 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 *Valid Range* on page 2-269, below, for details.

|        | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-268 for the enumerators of the enumerated type <code>_eBCD_FORMAT</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out    | Must be same size of data type as <i>In</i>   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

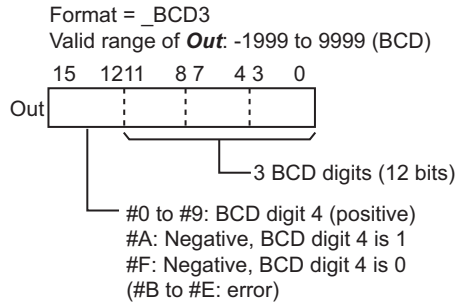
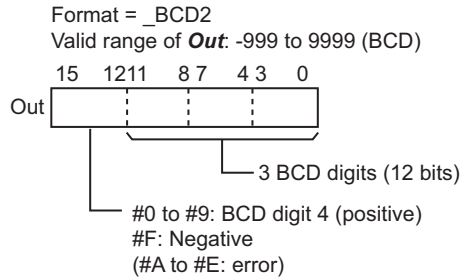
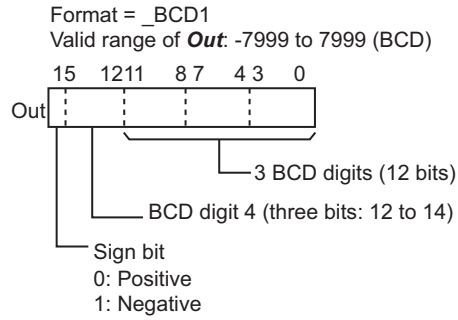
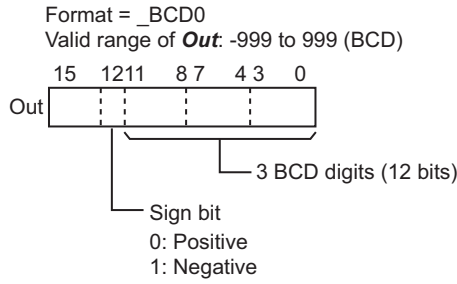
These instructions convert signed integer *In* to a signed BCD bit string.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the name of 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*.



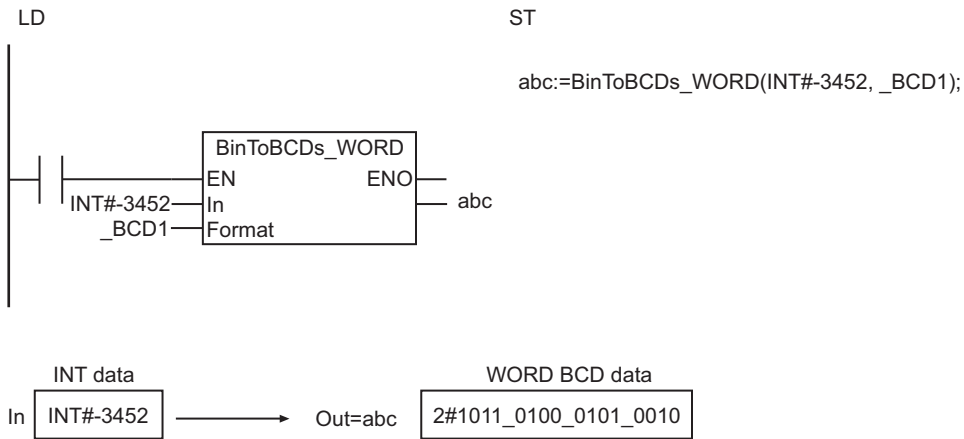
## Valid Range

The data types of *In* and *Out* should be of the same size. The valid ranges depend on the value of *Format*, as shown below.

|  |                    | Value of <i>Format</i>                |                                       |                                       |  |
|--|--------------------|---------------------------------------|---------------------------------------|---------------------------------------|--|
|  |                    | <code>_BCD0</code>                    | <code>_BCD1</code>                    | <code>_BCD2</code>                    | <code>_BCD3</code>                     |
| Data type of <i>In</i><br>↓<br>Data type of <i>Out</i> | SINT<br>↓<br>BYTE  | -9 to 9                               | -79 to 79                             | -9 to 99                              | -19 to 99                              |
|  | INT<br>↓<br>WORD   | -999 to 999                           | -7999 to 7999                         | -999 to 9999                          | -1999 to 9999                          |
|  | DINT<br>↓<br>DWORD | -9999999 to 9999999                   | -79999999 to 79999999                 | -99999999 to 99999999                 | -199999999 to 99999999                 |
|  | LINT<br>↓<br>LWORD | -9999999999999999 to 9999999999999999 | -7999999999999999 to 7999999999999999 | -9999999999999999 to 9999999999999999 | -19999999999999999 to 9999999999999999 |

## Notation Example

The following example shows the `BinToBCDs_WORD` instruction when *In* is `INT#-3452` and *Format* is `_BCD1`.



### Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error will occur in the following cases. *ENO* will be *FALSE*, and *Out* will not change.
  - a) The value of *In* is outside the valid range.
  - b) The value of *Format* is outside the valid range.

# AryToBCD

The AryToBCD instruction converts the elements of an unsigned integer array to BCD bit strings.

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression               |
|-------------|----------------------|------------|--------------------|-----------------------------|
| AryToBCD    | Array BCD Conversion | FUN        |                    | AryToBCD(In, Size, AryOut); |

## Variables

|                  | Meaning                | I/O    | Description                               | Valid range           | Unit | Default |
|------------------|------------------------|--------|---|-----------------------|------|---------|
| In[] (array)     | Unsigned integer array | Input  | Unsigned integer array                    | *1                    | ---  | *2      |
| Size             | Number of elements     |        | 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 *Valid Range* on page 2-272 for details.

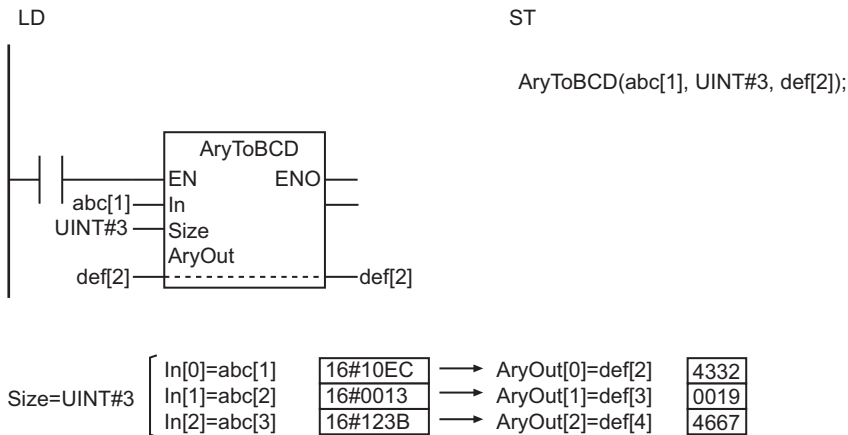
\*2. If you omit an input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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.



## Valid Range

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 Ary-Out[] | 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 this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and AryOut[] will not change.
  - a) The value of In[] is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression               |
|-------------|-----------------------------------|------------|--------------------|-----------------------------|
| AryToBin    | Array Unsigned Integer Conversion | FUN        |                    | AryToBin(In, Size, AryOut); |

## Variables

|                  | 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 *Valid Range* on page 2-274 for details.

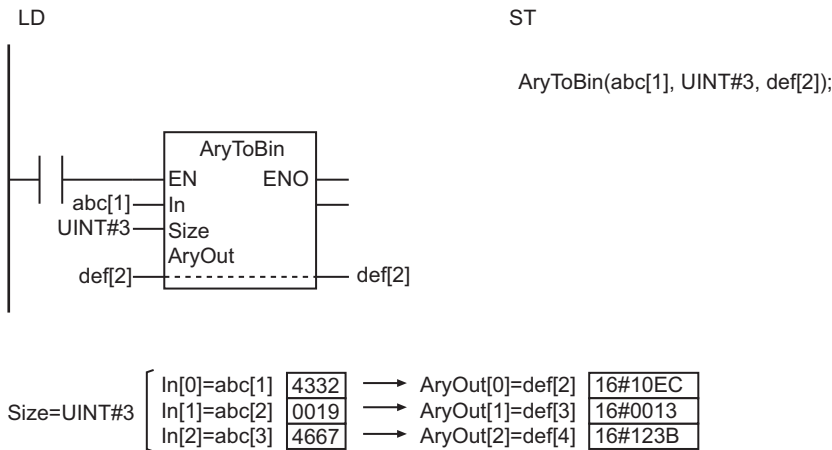
\*2. If you omit an input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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.



## Valid Range

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 Ary-Out[] | 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 this instruction is used in ST.
- An error will occur in the following cases. ENO will be FALSE, and AryOut[] will not change.
  - a) The value of Size exceeds the array area of In[] or AryOut[].
  - b) A value in In[] is not a BCD bit string (i.e., contains A, B, C, D, E, or F hexadecimal).

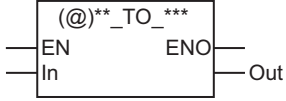
# Data Type Conversion Instructions

| Instruction  | Name  | Page       |
|--|---|------------|
| **_TO_*** (Integer-to-Integer Conversion Group)            | Integer-to-Integer Conversion Group         | page 2-277 |
| **_TO_*** (Integer-to-Bit String Conversion Group)         | Integer-to-Bit String Conversion Group      | page 2-280 |
| **_TO_*** (Integer-to-Real Number Conversion Group)        | Integer-to-Real Number Conversion Group     | page 2-283 |
| **_TO_*** (Bit String-to-Integer Conversion Group)         | Bit String-to-Integer Conversion Group      | page 2-285 |
| **_TO_*** (Bit String-to-Bit String Conversion Group)      | Bit String-to-Bit String Conversion Group   | page 2-288 |
| **_TO_*** (Bit String-to-Real Number Conversion Group)     | Bit String-to-Real Number Conversion Group  | page 2-290 |
| **_TO_*** (Real Number-to-Integer Conversion Group)        | Real Number-to-Integer Conversion Group     | page 2-292 |
| **_TO_*** (Real Number-to-Bit String Conversion Group)     | Real Number-to-Bit String Conversion Group  | page 2-295 |
| **_TO_*** (Real Number-to-Real Number Conversion Group)    | Real Number-to-Real Number Conversion Group | page 2-297 |
| **_TO_STRING (Integer-to-Text String Conversion Group)     | Integer-to-Text String Conversion Group     | page 2-299 |
| **_TO_STRING (Bit String-to-Text String Conversion Group)  | Bit String-to-Text String Conversion Group  | page 2-301 |
| **_TO_STRING (Real Number-to-Text String Conversion Group) | Real Number-to-Text String Conversion Group | page 2-303 |
| RealToFormatString   | REAL-to-Formatted Text String               | page 2-305 |
| LrealToFormatString  | LREAL-to-Formatted Text String              | page 2-311 |
| STRING_TO_** (Text String-to-Integer Conversion Group)     | Text String-to-Integer Conversion Group     | page 2-317 |
| STRING_TO_** (Text String-to-Bit String Conversion Group)  | Text String-to-Bit String Conversion Group  | page 2-319 |

| Instruction  | Name  | Page       |
|--|---|------------|
| STRING_TO_** (Text String-to-Real Number Conversion Group) | Text String-to-Real Number Conversion Group         | page 2-321 |
| TO_** (Integer Conversion Group)                           | Integer Conversion Group                            | page 2-325 |
| TO_** (Bit String Conversion Group)                        | Bit String Conversion Group                         | page 2-327 |
| TO_** (Real Number Conversion Group)                       | Real Number Conversion Group                        | page 2-329 |
| EnumToNum  | Enumeration-to-Integer                              | page 2-331 |
| NumToEnum  | Integer-to-Enumeration                              | page 2-333 |
| TRUNC, Round, and RoundUp                                  | Truncate/Round Off Real Number/Round Up Real Number | page 2-336 |

# \*\*\_TO\_\*\*\* (Integer-to-Integer Conversion Group)

These instructions convert integers to integers with different data types.

| Instruction | Name                                | FB/<br>FUN | Graphic expression   | ST expression   |
|-------------|-------------------------------------|------------|--|---|
| **_TO_***   | Integer-to-Integer Conversion Group | FUN        |  <p>*** and ***** must be different integer data types.</p> | Out:=**_TO_*** (In);<br><br>*** and ***** must be different integer data types. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-278, below, for details.

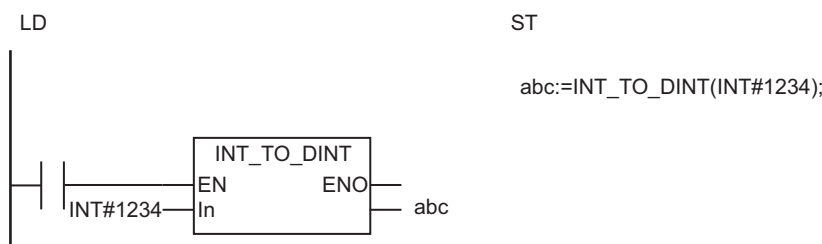
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|-------------------|------|--|------|------|-----|----|--------|
|     | 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                |      |  |      |      |     |    |        |

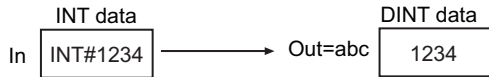
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> and <i>Out</i> |
|------------------------|-------------------------|--|
| 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                    |  |

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> and <i>Out</i> |
|------------------------|-------------------------|--|
| 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                |
| 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

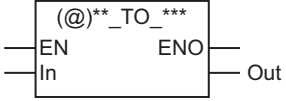
To convert data with any data type to integer data, use the instruction, *TO\_\*\** (*Integer Conversion Group*) on page 2-325.

## 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 will be truncated.

# \*\*\_TO\_\*\*\* (Integer-to-Bit String Conversion Group)

These instructions convert integers to bit strings.

| Instruction | Name                                   | FB/<br>FUN | Graphic expression  | ST expression   |
|-------------|--|------------|---|---|
| **_TO_***   | Integer-to-Bit String Conversion Group | FUN        |  <p>*** must be an integer data type. **** must be a bit string data type.</p> | Out:=**_TO_*** (In);<br>*** must be an integer data type.<br>**** must be a bit string data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-281, below, for details.

|     | Boo<br>lean | Bit strings |          |       |       |       | Integers |       |       |      |     |      |          | Real<br>num-<br>bers |       | Times, durations,<br>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<br>*1 |                      |       |  |      |     |    |        |
| Out |             | OK          | OK<br>*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.

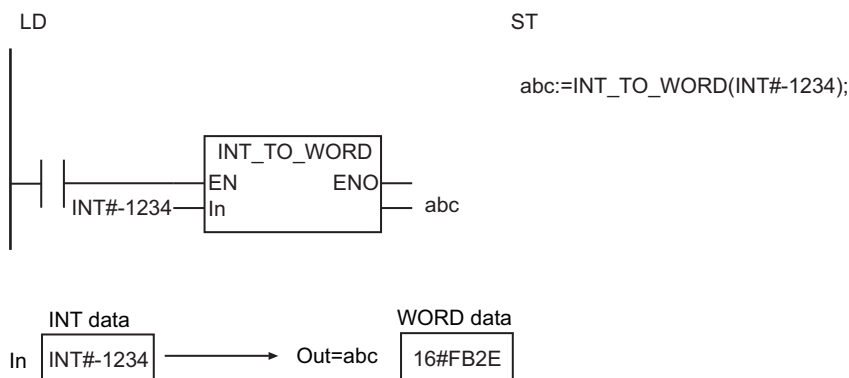
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> | Valid range for <i>Out</i>                       |
|------------------------|-------------------------|---------------------------|--|
| 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                   |                           |  |

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                   | Valid range for <i>Out</i>                       |
|------------------------|-------------------------|---|--|
| 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

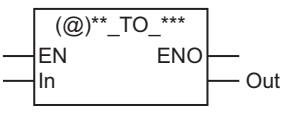
- To convert a bit string to an integer, use the instruction, **\*\*\_TO\_\*\*** (*Bit String-to-Integer Conversion Group*) on page 2-285.
- To convert data with any data type to a bit string, use the instruction, **TO\_\*\*** (*Bit String Conversion Group*) on page 2-327.

## 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 will be truncated.

# \*\*\_TO\_\*\*\* (Integer-to-Real Number Conversion Group)

These instructions convert integers to real numbers.

| Instruction | Name                                    | FB/<br>FUN | Graphic expression  | ST expression   |
|-------------|---|------------|---|---|
| **_TO_***   | Integer-to-Real Number Conversion Group | FUN        |  <p>*** must be an integer data type. ***** must be a real number data type.</p> | Out:=**_TO_*** (In);<br><br>*** must be an integer data type.<br>***** must be a real number data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-284, below, for details.

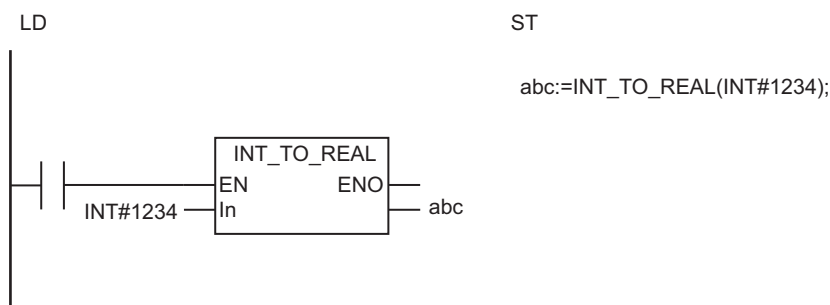
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | 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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                   | Valid range for <i>Out</i>                    |
|------------------------|-------------------------|---|---|
| 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 the instruction, **\*\*\_TO\_\*\*** (*Real Number-to-Integer Conversion Group*) on page 2-292.
- To convert data with any data type to a real number, use the instruction, **TO\_\*\*** (*Real Number Conversion Group*) on page 2-329.

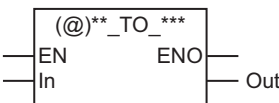
## Precautions for Correct Use

- 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 may cause error between the values before and after conversion. The following table lists the data types that result in error.

| Data type of <i>In</i> | Data type of <i>Out</i> | 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/<br>FUN | Graphic expression   | ST expression  |
|-------------|--|------------|--|--|
| **_TO_***   | Bit String-to-Integer Conversion Group | FUN        |  <p>*** must be a bit string data type. ***** must be an integer data type.</p> | Out:=**_TO_*** (In);<br><br>***** must be a bit string data type.<br>***** must be an integer data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-286, below, for details.

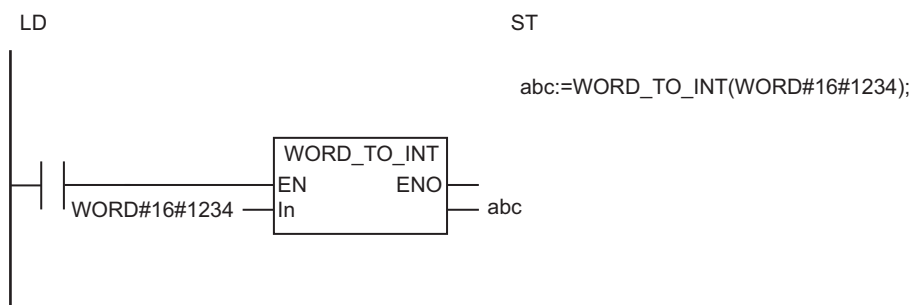
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|     | 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                   |      |  |      |      |     |    |        |

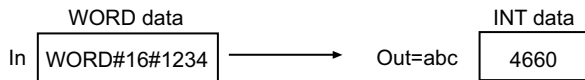
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                        | Valid range for <i>Out</i>                  |
|------------------------|-------------------------|--|---|
| 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                   | 16#00 to 16#FF                                   | -128 to 127                                 |
|                        | SINT                    |  |   |
|                        | INT                     |  |   |
|                        | 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

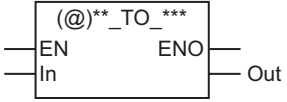
- To convert an integer to a bit string, use the instruction, **\*\*\_TO\_\*\*** (*Integer-to-Bit String Conversion Group*) on page 2-280.
- To convert data with any data type to a bit string, use the instruction, **TO\_\*\*** (*Bit String Conversion Group*) on page 2-327.

## 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.

# \*\*\_TO\_\*\*\* (Bit String-to-Bit String Conversion Group)

These instructions convert bit strings to bit strings with different data types.

| Instruction | Name                                      | FB/<br>FUN | Graphic expression  | ST expression  |
|-------------|---|------------|---|--|
| **_TO_***   | Bit String-to-Bit String Conversion Group | FUN        |  <p>*** and ***** must be different bit string data types.</p> | Out:=**_TO_*** (In);<br>***** and ***** must be different bit string data types. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-289, below, for details.

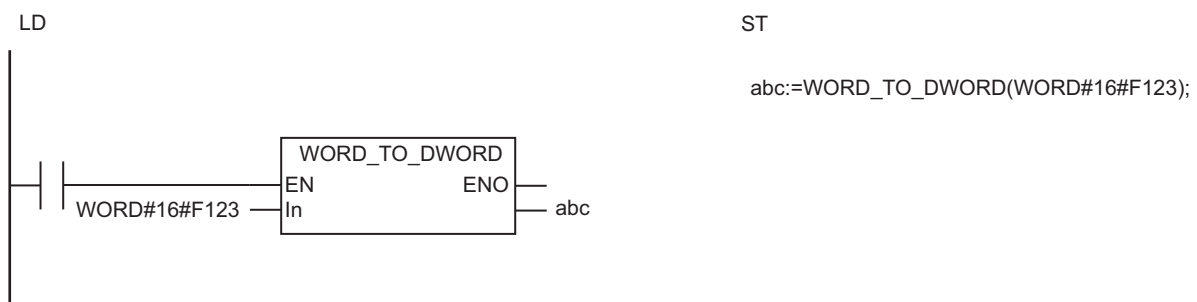
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|     | 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.







## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> and <i>Out</i> |
|------------------------|-------------------------|--|
| 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

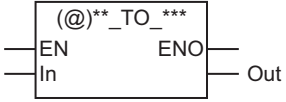
To convert data with any data type to a bit string, use the instruction, *TO\_\*\** (*Bit String Conversion Group*) on page 2-327.

## 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.

# \*\*\_TO\_\*\*\* (Bit String-to-Real Number Conversion Group)

These instructions convert bit strings to real numbers.

| Instruction | Name                                       | FB/<br>FUN | Graphic expression  | ST expression   |
|-------------|--|------------|---|---|
| **_TO_***   | Bit String-to-Real Number Conversion Group | FUN        |  <p>*** must be a bit string data type. ***** must be a real number data type.</p> | Out:=**_TO_*** (In);<br>*** must be a bit string data type.<br>***** must be a real number data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-291, below, for details.

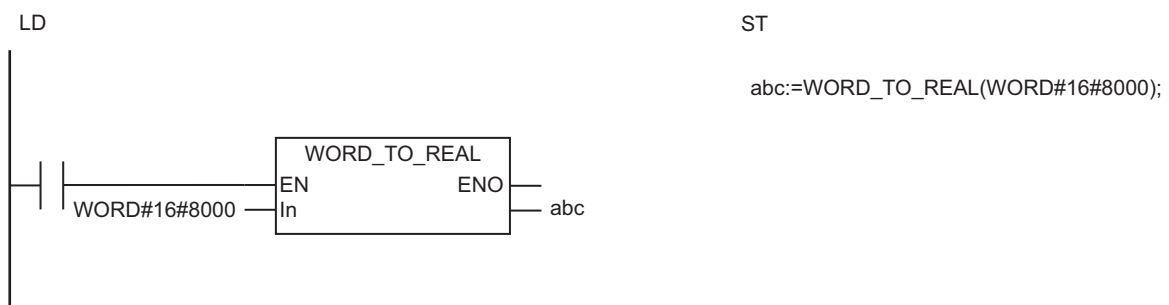
|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |  |
|-----|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|--|
|     | 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   |      |      |     |    |        |  |

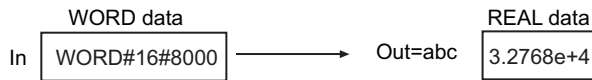
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                        | Valid range for <i>Out</i> |
|------------------------|-------------------------|--|----------------------------|
| 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 the instruction, **\*\*\_TO\_\*\*** (*Real Number-to-Bit String Conversion Group*) on page 2-295.
- To convert data with any data type to a real number, use the instruction, **TO\_\*\*** (*Real Number Conversion Group*) on page 2-329.

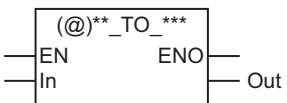
## Precautions for Correct Use

- 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 may cause error between the values before and after conversion. The following table lists the data types that result in error.

| Data type of <i>In</i> | Data type of <i>Out</i> | 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|---|------------|--|---|
| **_TO_***   | Real Number-to-Integer Conversion Group | FUN        |  <p>**** must be a real number data type.<br/>***** must be an integer data type.</p> | Out:=**_TO_*** (In);<br><br>**** must be a real number data type. ***** must be an integer data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-293, below, for details.

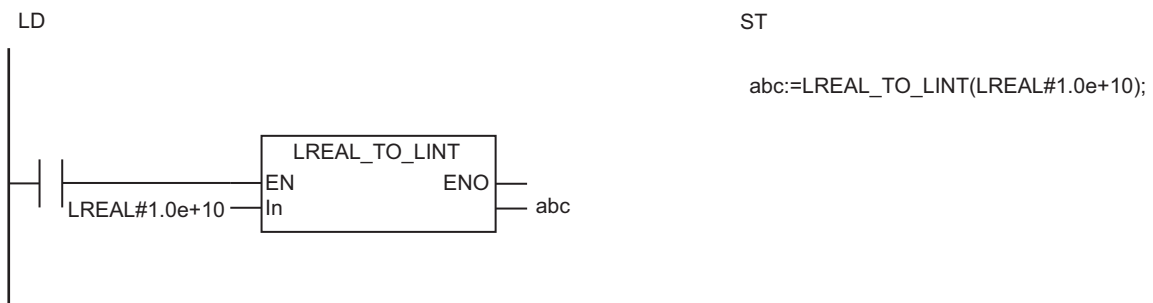
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | 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   |                      |       |  |      |     |    |        |

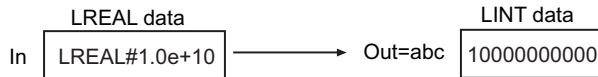
## Function

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.





## Fractional Part of the Value of *In*

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 | Description   | Examples   |
|--------------------------|---|--|
| Less than 0.5            | The fractional part is truncated.   | 1.49 → 1<br>-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<br>2.50 → 2<br>-1.50 → -2<br>-2.50 → -2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2<br>-1.51 → -2                           |

## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i>                     | Valid range for <i>Out</i>                  |
|------------------------|-------------------------|---|---|
| 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 2.55e+2                                  | 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 the instruction, **\*\*\_TO\_\*\*\*** (*Integer-to-Real Number Conversion Group*) on page 2-283.

- To convert data with any data type to integer data, use the instruction, *TO\_\*\* (Integer Conversion Group)* on page 2-325.
- 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.

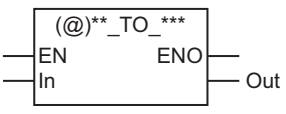
| Input value | Output value |         |         |         |
|-------------|--------------|---------|---------|---------|
|             | REAL_TO_INT  | TRUNC   | Round   | RoundUp |
| REAL#1.6    | INT#2        | DINT#1  | DINT#2  | DINT#2  |
| REAL#1.5    | INT#2        | DINT#1  | DINT#2  | DINT#2  |
| 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|--|------------|--|---|
| **_TO_***   | Real Number-to-Bit String Conversion Group | FUN        |  <p>*** must be a real number data type.<br/>**** must be a bit string data type.</p> | Out:=**_TO_*** (In);<br><br>*** must be a real number data type. **** must be a bit string data type. |

## Variables

|     | 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. | ---  | ---     |

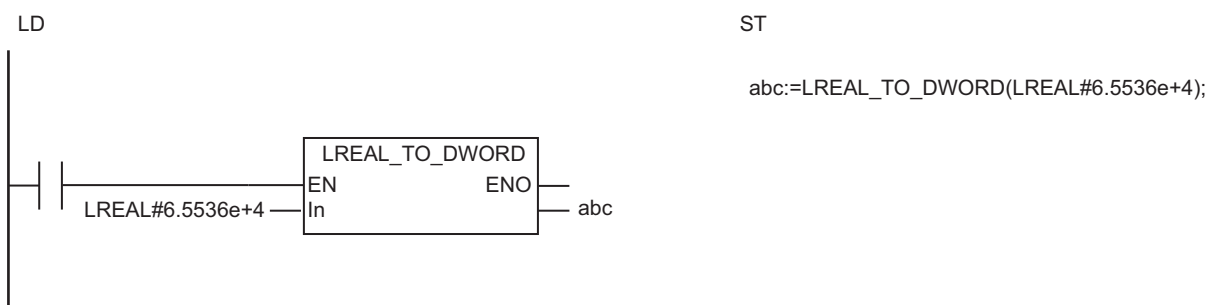
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | 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    |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

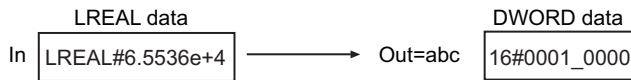
## Function

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.





The following table gives some conversion examples.

| Value of <i>In</i> | Integer | Value of <i>Out</i> |
|--------------------|---------|---------------------|
| 1.6                | 2       | 16#0002             |
| 3.5                | 4       | 16#0004             |

## Conversion Procedure

Conversion is performed using the following procedure.

- 1** 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.

## Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | 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<br>2.50 → 2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2             |

## Additional Information

To convert a bit string to a real number, use the instruction, **\*\*\_TO\_\*\*\*** (*Bit String-to-Real Number Conversion Group*) on page 2-290.

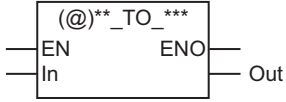
## Precautions for Correct Use

- 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*.
- When you input a negative value, the conversion result depends on the CPU Unit model. If you input a negative value, sufficiently debug before use.



# \*\*\_TO\_\*\*\* (Real Number-to-Real Number Conversion Group)

These instructions convert real numbers to real numbers with different data types.

| Instruction | Name  | FB/<br>FUN | Graphic expression   | ST expression   |
|-------------|---|------------|--|---|
| **_TO_***   | Real Number-to-Real Number Conversion Group | FUN        |  <p>*** and ***** must be different real number data types.</p> | Out:=**_TO_*** (In);<br><br>***** and ***** must be different real number data types. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range | Unit | Default |
|-----|-------------------|--------|-------------------|-------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1          | ---  | 0       |
| Out | Conversion result | Output | Conversion result | *1          | ---  | ---     |

\*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-298, below, for details.

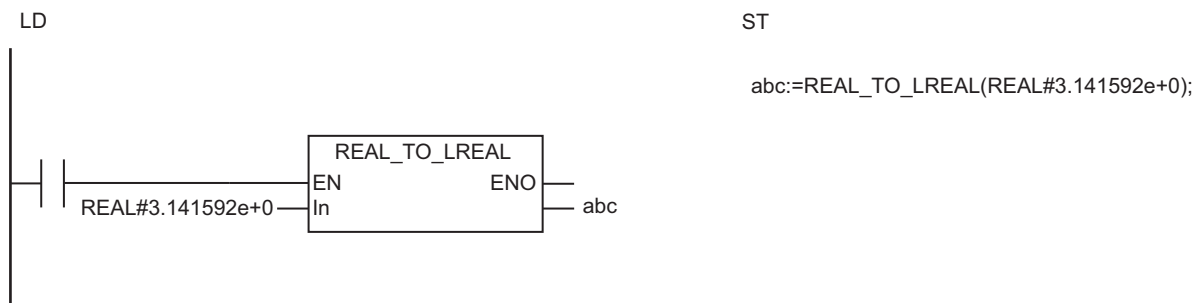
|     | Boolean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|---------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|     | 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    |   |      |     |    |        |

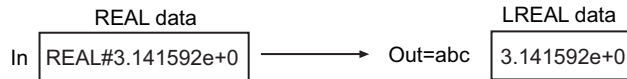
## Function

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.





## Valid Range

The following table shows the valid ranges for *In* and *Out* according to their data types.

| Data type of <i>In</i> | Data type of <i>Out</i> | Valid range for <i>In</i> and <i>Out</i> |
|------------------------|-------------------------|--|
| REAL                   | LREAL                   | -3.402823e+38 to 3.402823e+38            |
| LREAL                  | REAL                    | or +∞/-∞                                 |

## Additional Information

To convert data with any data type to a real number, use the instruction, *TO\_\*\** (*Real Number Conversion Group*) on page 2-329.

## 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  $\pm 1.175494e-38$ , 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/<br>FUN | Graphic expression                        | ST expression  |
|------------------|---|------------|---|--|
| **_TO_STRIN<br>G | Integer-to-Text<br>String Conver-<br>sion Group | FUN        | <p>**** must be an integer data type.</p> | Out:=**_TO_STRING(In);<br><br>**** must be an integer data type. |

## Variables

|     | 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 | *1                    | ---  | ---     |

\*1. The valid range depends on the data type of *In*. Refer to *Valid Range* on page 2-300 for details.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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*.

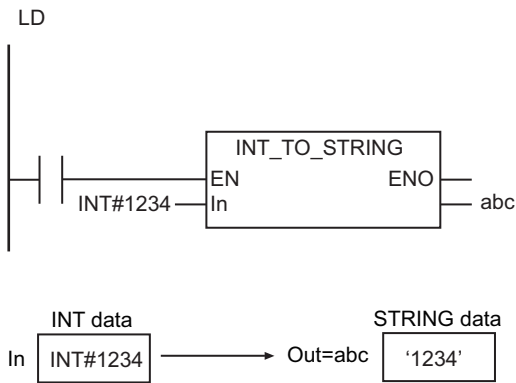
The text in *Out* is left-aligned.

If the number of significant digits in *In* is less than the digits provided by the data type of *In*, its leading zeros will not be output to *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.



ST

abc:=INT\_TO\_STRING(INT#1234);

## Valid Range

The valid range of *Out* depends on the data type of *In* as shown below:

| Data type of <i>In</i> | Valid range of <i>Out</i> (maximum number of bytes)                             |
|------------------------|---|
| USINT                  | 4 bytes (3 single-byte alphanumeric characters plus the final NULL character)   |
| UINT                   | 6 bytes (5 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 (4 single-byte alphanumeric characters plus the final NULL character)   |
| INT                    | 7 bytes (6 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

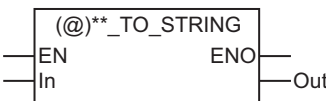
To convert a text string number to an integer, use the instruction, *STRING\_TO\_\*\** (*Text String-to-Integer Conversion Group*) on page 2-317.

## 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/<br>FUN | Graphic expression   | ST expression  |
|--------------|--|------------|--|--|
| **_TO_STRING | Bit String-to-Text String Conversion Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=**_TO_STRING(In);<br>**** must be a bit string data type. |

## Variables

|     | 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 | *1                    | ---  | ---     |

\*1. The valid range depends on the data type of *In*. Refer to *Valid Range* on page 2-302 for details.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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.

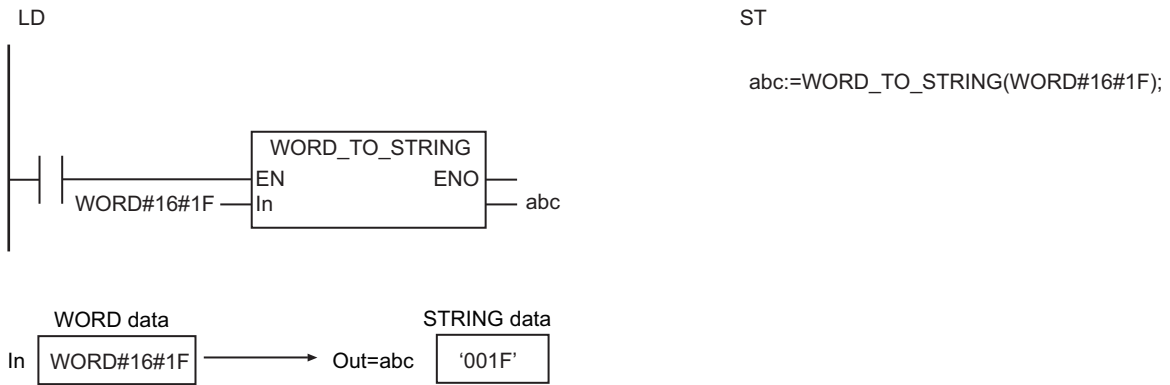
The hexadecimal number given in *In* is output to conversion result *Out* as a text string. The "#16" prefix of the hexadecimal number is not output to *Out*. A NULL character (16#00) is placed at the end of *Out*.

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.



## Valid Range

The valid range of *Out* depends on the data type of *In* as shown below:

| Data type of <i>In</i> | Valid range of <i>Out</i> (maximum number of bytes)                             |
|------------------------|---|
| BYTE                   | 3 bytes (2 single-byte alphanumeric characters plus the final NULL character)   |
| WORD                   | 5 bytes (4 single-byte alphanumeric characters plus the final NULL character)   |
| DWORD                  | 9 bytes (8 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 the instruction, **\*\*\_TO\_\*\*\*** (*Bit String-to-Integer Conversion Group*) on page 2-285, and then use the instruction, **\*\*\_TO\_STRING** (*Integer-to-Text String Conversion Group*) on page 2-299.

## 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/<br>FUN | Graphic expression  | ST expression   |
|------------------|---|------------|---|---|
| **_TO_STRIN<br>G | Real Number-to-Text String Conversion Group | FUN        | <p>Graphic expression showing a box labeled (@)**_TO_STRING with EN and In on the left and Out on the right.</p> <p>**** must be a real number data type.</p> | Out:=**_TO_STRING(In);<br>**** must be a real number data type. |

## Variables

|     | 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 | *1                    | ---  | ---     |

\*1. The valid range depends on the data type of *In*. Refer to *Valid Range* on page 2-304 for details.

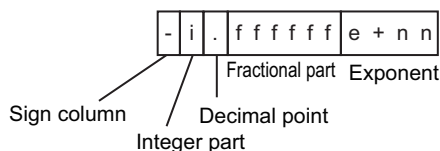
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|-------------------|-------|--|------|-----|----|--------|
|     | 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:



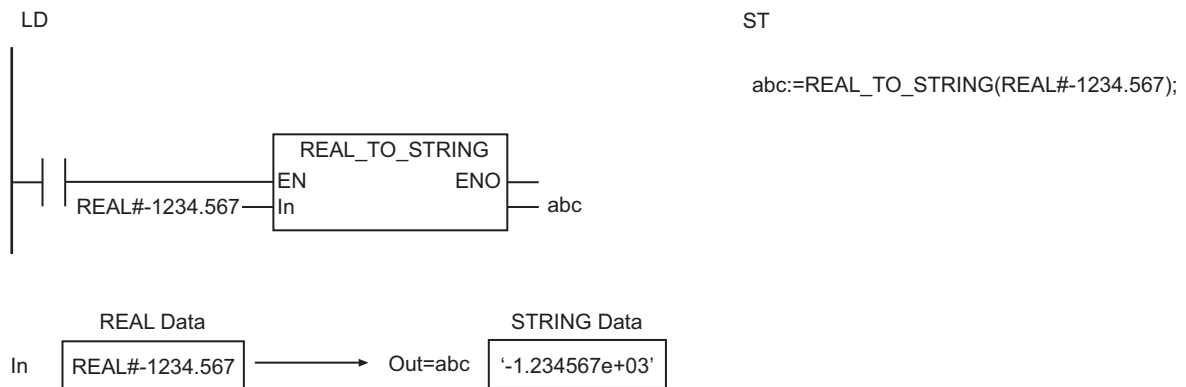
| Item            | Description   |
|-----------------|---|
| Sign column     | If <i>In</i> contains a negative value, a minus sign (-) is added.<br>If <i>In</i> contains a positive value, a plus sign (+) is not added. |
| Integer part    | The integer part is always only one digit.  |
| Decimal point   | The decimal point is always given even if <i>In</i> is not a decimal number.  |
| Fractional part | If <i>In</i> is REAL data, 6 digits are given, and if it is LREAL data, 14 digits are given.  |

| Item     | Description  |
|----------|--|
| Exponent | The exponent is always given.<br>"nn" is 2 or 3 digits.<br>The sign of "nn" is positive (+) if the absolute value of <i>In</i> 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.



### Valid Range

If the value of *In* is 0, infinity, or nonnumeric data, the value of *Out* is as shown below.

| Value of <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| 0                  | '0'                 |
| +∞                 | 'inf'               |
| -∞                 | '-inf'              |
| Nonnumeric data    | 'nan' or '-nan'     |

### Additional Information

- To convert a text string to a real number, use the instruction, *STRING\_TO\_\*\** (*Text String-to-Real Number Conversion Group*) on page 2-321.
- To specify the format when you convert a real number to a text string, use the instruction, *RealToFormatString* on page 2-305 or *LrealToFormatString* on page 2-311.

### 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/<br>FUN | Graphic expression | ST expression  |
|--------------------|-------------------------------|------------|--------------------|--|
| RealToFormatString | REAL-to-Formatted Text String | FUN        |                    | Out:=RealToFormatString(In, Exponent, Sign, MinLen, DecPlace); |

## Variables

|          | 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<br>FALSE: No exponent       |  |      | FALSE   |   |
| Sign     | Sign column              |        | TRUE: Sign column<br>FALSE: No sign column |  |      |         |   |
| MinLen   | Minimum number of digits |        | Minimum number of digits in <i>Out</i>     |  |      |         | 6 |
| DecPlace | Precision                |        | Number of decimal digits in <i>Out</i>     |  |      |         |   |
| Out      | Conversion result        | Output | Conversion result                          | 327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character) | ---  | ---     |   |

|          | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     |

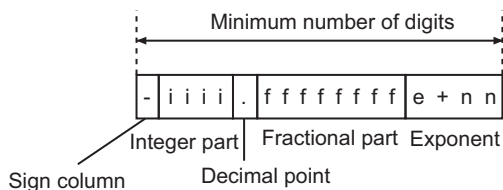
## Function

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       | <i>Exponent</i> specifies whether an exponent is given.<br>TRUE: Exponent<br>FALSE: No exponent   |
| Sign           | <i>Sign</i> specifies whether there is a sign column.<br>TRUE: Sign column<br>FALSE: No sign column<br><br>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.<br>If the number is negative when no sign column is specified, a minus sign (-) will be added to the front of the integer part.<br><br>However, if the number of digits in the conversion result exceeds the value of <i>MinLen</i> and the conversion result is positive, the highest digit is placed in the sign column. |
| MinLen         | <i>MinLen</i> is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent.<br><br>If the conversion result has fewer digits than the value of <i>MinLen</i> , the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters.<br>If the number of digits in the conversion result exceeds the value of <i>MinLen</i> , the text string is left-aligned and the text string for the digits that exceed the value of <i>MinLen</i> is assigned to <i>Out</i> .                         |
| DecPlace       | <i>DecPlace</i> is the number of digits in the fractional part.<br><br>If the number of digits exceeds the value of <i>DecPlace</i> , the extra digits in the fractional portion are rounded off as described below.<br>If the value of <i>DecPlace</i> is 0, the fractional part and decimal point are not given.  |

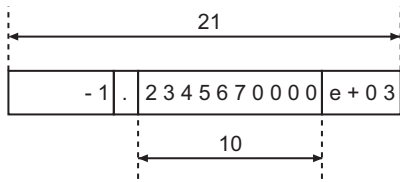
If the value of *In* is infinity, or nonnumeric data, the value of *Out* is as shown below.

| Value of <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| +∞                 | 'inf'               |
| -∞                 | '-inf'              |
| Nonnumeric data    | 'nan' or '-nan'     |

## Rounding Off

The following table shows how values are rounded.

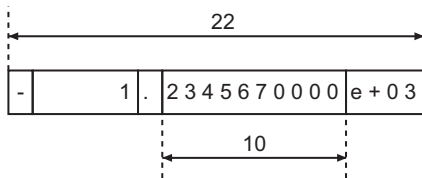




● Example 3

| Variables | Settings       |
|-----------|----------------|
| In        | REAL#-1234.567 |
| Exponent  | TRUE           |
| Sign      | TRUE           |
| MinLen    | USINT#22       |
| DecPlace  | USINT#10       |

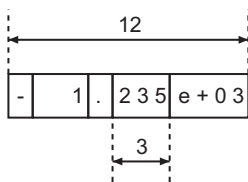
The sign column is always on the left. Blank characters are added to the front of the integer part.



● Example 4

| Variables | Settings       |
|-----------|----------------|
| In        | REAL#-1234.567 |
| Exponent  | TRUE           |
| Sign      | TRUE           |
| MinLen    | USINT#12       |
| DecPlace  | USINT#3        |

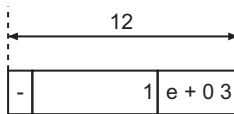
The fourth decimal place is rounded off because *DecPlace* is USINT#3.



● Example 5

| Variables | Settings       |
|-----------|----------------|
| In        | REAL#-1234.567 |
| Exponent  | TRUE           |
| Sign      | TRUE           |
| MinLen    | USINT#12       |
| DecPlace  | USINT#0        |

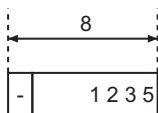
The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



### ● Example 6

| Variables | Settings       |
|-----------|----------------|
| In        | REAL#-1234.567 |
| Exponent  | FALSE          |
| Sign      | TRUE           |
| MinLen    | USINT#8        |
| DecPlace  | USINT#0        |

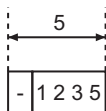
Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



### ● Example 7

| Variables | Settings       |
|-----------|----------------|
| In        | REAL#-1234.567 |
| Exponent  | FALSE          |
| Sign      | TRUE           |
| MinLen    | USINT#2        |
| DecPlace  | USINT#0        |

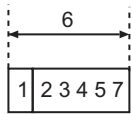
Here, the number of digits in the integer part of *In* (four digits) is larger than the value of *MinLen* (USINT#2). The four digits of the integer part are given.



### ● Example 8

| Variables | Settings      |
|-----------|---------------|
| In        | REAL#123456.7 |
| 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.



## 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 instruction, *LrealToFormatString* on page 2-311.
- To convert a text string to a real number, use the instruction, *STRING\_TO\_\*\** (*Text String-to-Real Number Conversion Group*) on page 2-321.

## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside 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/<br>FUN | Graphic expression | ST expression  |
|---------------------|--------------------------------|------------|--------------------|--|
| LrealToFormatString | LREAL-to-Formatted Text String | FUN        |                    | Out:=LrealToFormatString (In, Exponent, Sign, MinLen, DecPlace); |

## Variables

|          | 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<br>FALSE: No exponent       |  |      | FALSE   |
| Sign     | Sign column              |        | TRUE: Sign column<br>FALSE: No sign column |  |      |         |
| MinLen   | Minimum number of digits |        | Minimum number of digits in <i>Out</i>     |  |      | 6       |
| DecPlace | Precision                |        | Number of decimal digits in <i>Out</i>     |  |      |         |
| Out      | Conversion result        | Output | Conversion result                          | 327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character) | ---  | ---     |

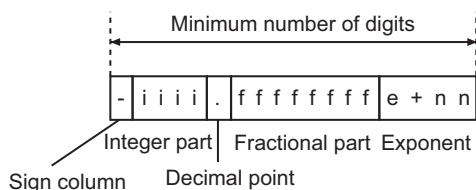
|          | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|----------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|          | 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       | <i>Exponent</i> specifies whether an exponent is given.<br>TRUE: Exponent<br>FALSE: No exponent   |
| Sign           | <i>Sign</i> specifies whether there is a sign column.<br>TRUE: Sign column<br>FALSE: No sign column<br><br>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.<br>If the number is negative when no sign column is specified, a minus sign (-) will be added to the front of the integer part.<br><br>However, if the number of digits in the conversion result exceeds the value of <i>MinLen</i> and the conversion result is positive, the highest digit is placed in the sign column. |
| MinLen         | <i>MinLen</i> is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent.<br><br>If the conversion result has fewer digits than the value of <i>MinLen</i> , the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters.<br>If the number of digits in the conversion result exceeds the value of <i>MinLen</i> , the text string is left-aligned and the text string for the digits that exceed the value of <i>MinLen</i> is assigned to <i>Out</i> .                         |
| DecPlace       | <i>DecPlace</i> is the number of digits in the fractional part.<br><br>If the number of digits exceeds the value of <i>DecPlace</i> , the extra digits in the fractional portion are rounded off as described below.<br>If the value of <i>DecPlace</i> is 0, the fractional part and decimal point are not given.  |

If the value of *In* is infinity, or nonnumeric data, the value of *Out* is as shown below.

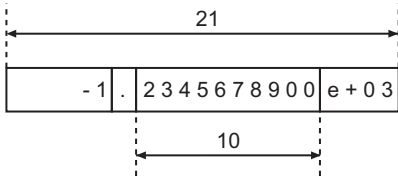
| Value of <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| $+\infty$          | 'inf'               |
| $-\infty$          | '-inf'              |
| Nonnumeric data    | 'nan' or '-nan'     |





| Variables | Settings |
|-----------|----------|
| MinLen    | USINT#21 |
| DecPlace  | USINT#10 |

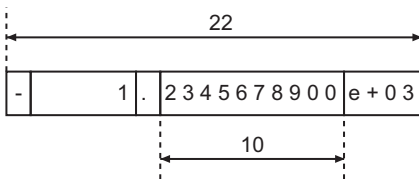
Here, the value of *MinLen* exceeds the number of digits in the text string, so the text string is right-aligned and blank characters are added before it.



● Example 3

| Variables | Settings          |
|-----------|-------------------|
| In        | LREAL#-1234.56789 |
| Exponent  | TRUE              |
| Sign      | TRUE              |
| MinLen    | USINT#22          |
| DecPlace  | USINT#10          |

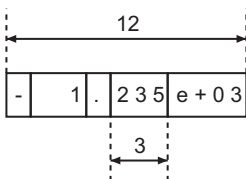
The sign column is always on the left. Blank characters are added to the front of the integer part.



● Example 4

| Variables | Settings          |
|-----------|-------------------|
| In        | LREAL#-1234.56789 |
| Exponent  | TRUE              |
| Sign      | TRUE              |
| MinLen    | USINT#12          |
| DecPlace  | USINT#3           |

The fourth decimal place is rounded off because *DecPlace* is USINT#3.

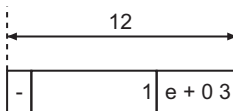


● Example 5

| Variables | Settings          |
|-----------|-------------------|
| In        | LREAL#-1234.56789 |

| Variables | Settings |
|-----------|----------|
| Exponent  | TRUE     |
| Sign      | TRUE     |
| MinLen    | USINT#12 |
| DecPlace  | USINT#0  |

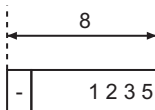
The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



### ● Example 6

| Variables | Settings          |
|-----------|-------------------|
| In        | LREAL#-1234.56789 |
| Exponent  | FALSE             |
| Sign      | TRUE              |
| MinLen    | USINT#8           |
| DecPlace  | USINT#0           |

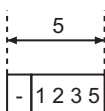
Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



### ● Example 7

| Variables | Settings          |
|-----------|-------------------|
| In        | LREAL#-1234.56789 |
| 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.

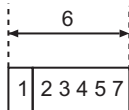


### ● Example 8

| Variables | Settings         |
|-----------|------------------|
| In        | LREAL#123456.789 |

| Variables | Settings |
|-----------|----------|
| 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.



## 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 instruction, *RealToFormatString* on page 2-305.
- To convert a text string to a real number, use the instruction, *STRING\_TO\_\*\** (*Text String-to-Real Number Conversion Group*) on page 2-321.

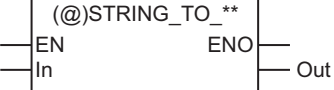
## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside 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/<br>FUN | Graphic expression  | ST expression  |
|--------------|---|------------|---|--|
| STRING_TO_** | Text String-to-Integer Conversion Group | FUN        |  <p>*** must be an integer data type.</p> | Out:=STRING_TO_** (In);<br>*** must be an integer data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-----|-------------------|--------|-------------------|-----------------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1                    | ---  | "       |
| Out | Conversion result | Output | Conversion result | Depends on data type. | ---  | ---     |

\*1. The valid range depends on the data type of *Out*. Refer to *Valid Range* on page 2-318 for details.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num<br>bers |       | Times, durations,<br>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

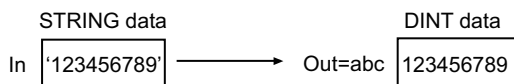
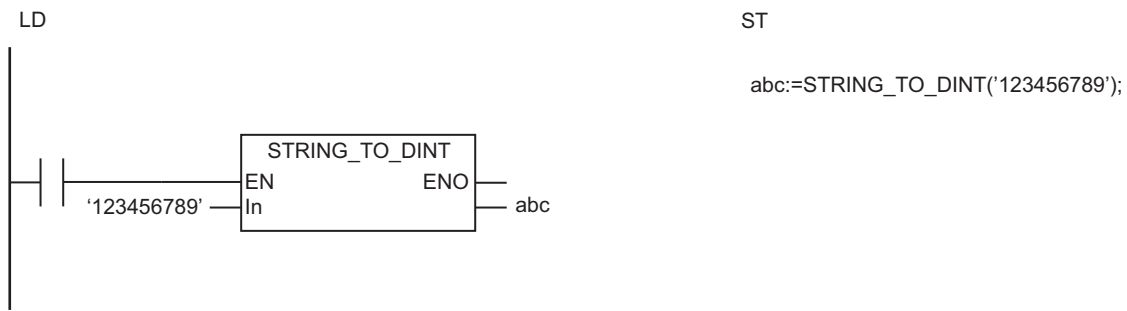
These instructions convert a text string, *In*, to an integer.

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'.



## Valid Range

The valid range of *In* depends on the data type of *Out* as shown below:

| Data type of <i>Out</i> | Valid range of <i>In</i> (maximum number of bytes)*1                            |
|-------------------------|---|
| USINT                   | 4 bytes (3 single-byte alphanumeric characters plus the final NULL character)   |
| UINT                    | 6 bytes (5 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 (4 single-byte alphanumeric characters plus the final NULL character)   |
| INT                     | 7 bytes (6 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) |

\*1. 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 the instruction, `STRING_TO_**` (*Text String-to-Bit String Conversion Group*) on page 2-319.
- To convert an integer to a text string, use the instruction, `**_TO_STRING` (*Integer-to-Text String Conversion Group*) on page 2-299.

## 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/<br>FUN | Graphic expression                         | ST expression  |
|--------------|--|------------|--|--|
| STRING_TO_** | Text String-to-Bit String Conversion Group | FUN        | <p>*** must be a bit string data type.</p> | Out:=STRING_TO_** (In);<br>*** must be a bit string data type. |

## Variables

|     | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-----|-------------------|--------|-------------------|-----------------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | *1                    | ---  | "       |
| Out | Conversion result | Output | Conversion result | Depends on data type. | ---  | ---     |

\*1. The valid range depends on the data type of *Out*. Refer to *Valid Range* on page 2-320 for details.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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

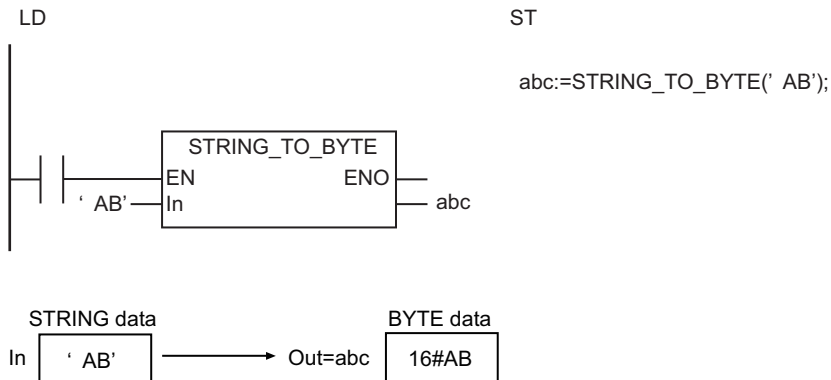
These instructions interpret the content of a text string, *In*, as a hexadecimal number and convert it to a bit string.

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.



## Valid Range

The valid range of *In* depends on the data type of *Out* as shown below:

| Data type of <i>Out</i> | Valid range of <i>In</i> (maximum number of bytes) <sup>*1</sup>                |
|-------------------------|---|
| BYTE                    | 3 bytes (2 single-byte alphanumeric characters plus the final NULL character)   |
| WORD                    | 5 bytes (4 single-byte alphanumeric characters plus the final NULL character)   |
| DWORD                   | 9 bytes (8 single-byte alphanumeric characters plus the final NULL character)   |
| LWORD                   | 17 bytes (16 single-byte alphanumeric characters plus the final NULL character) |

\*1. 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 the instruction, `STRING_TO_**` (*Text String-to-Integer Conversion Group*) on page 2-317.
- To convert a bit string to a text string, use the instruction, `**_TO_STRING` (*Bit String-to-Text String Conversion Group*) on page 2-301.


## Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error will occur 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/<br>FUN | Graphic expression   | ST expression   |
|--------------|---|------------|--|---|
| STRING_TO_** | Text String-to-Real Number Conversion Group | FUN        |  <p>*** must be a real number data type.</p> | Out:=STRING_TO_** (In);<br>*** must be a real number data type. |

## Variables

|     | 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.  | ---  | ---     |

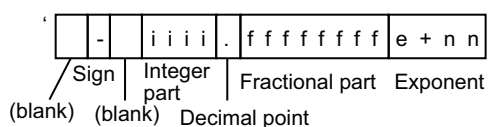
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|---------------------|-------|--|------|-----|----|--------|
|     | 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.



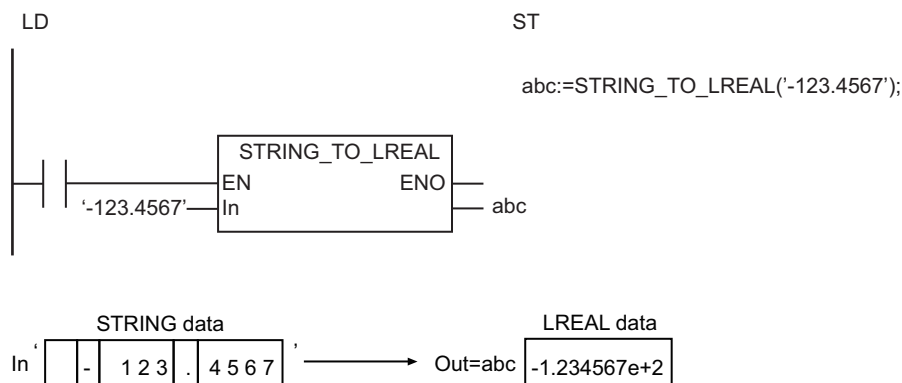
| Name            | Format  |
|-----------------|---|
| Sign            | <ul style="list-style-type: none"> <li>Any consecutive blank characters at the beginning of the text string are ignored. Any following single plus sign (+) or minus sign (-) is treated as the sign.</li> <li>The plus sign (+) can be omitted.</li> <li>Any consecutive blank characters after the sign are ignored.</li> </ul>   |
| Integer part    | <ul style="list-style-type: none"> <li>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.</li> <li>If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part.</li> <li>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.</li> <li>The integer part consists of '0' to '9'.</li> <li>The integer part cannot be omitted.</li> <li>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.</li> </ul> |
| Decimal point   | <ul style="list-style-type: none"> <li>A single period (.) following the integer part is taken as the decimal point.</li> <li>Omit the decimal point if there is no fractional part.</li> </ul>   |
| Fractional part | <ul style="list-style-type: none"> <li>The characters after the decimal point and up to the exponent are taken as the fractional part.</li> <li>If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part.</li> <li>The fractional part consists of '0' to '9'.</li> <li>The fractional part can be omitted.</li> <li>The fractional part can consist of a maximum of 15 digits.</li> <li>If there is no decimal point, then there is no fractional part.</li> </ul>   |
| Exponent        | <ul style="list-style-type: none"> <li>The exponent consists of a single 'e' or 'E' after the fractional part, a following single plus sign (+) or minus sign (-), and the remaining characters to the end of the text string.</li> <li>If there is no fractional part, then the above text string after the decimal point is taken as the exponent.</li> <li>If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent.</li> <li>The numeric part of the exponent consists of '0' to '9'.</li> <li>The exponent can be omitted.</li> <li>The numeric part of the exponent can consist of a maximum of three digits.</li> </ul>   |

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.

## Notation Example

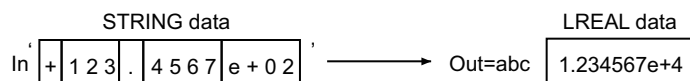
### ● Example 1:

The following example uses the sign, decimal point, and fractional part, but does not use an exponent.



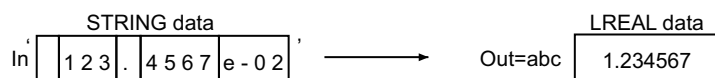
### ● Example 2:

The following example uses the sign, decimal point, fractional part, and exponent.



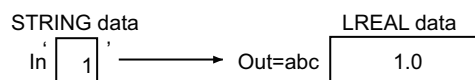
### ● Example 3:

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.



## Additional Information

To convert a real number to a text string, use the instruction, *\*\*\_TO\_STRING* (*Real Number-to-Text String Conversion Group*) on page 2-303.

## Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If there is a single underbar ( ) at any location in *In*, it is ignored.
- An error occurs if there is an underbar ( ) at the beginning or the end of *In*.
- An error occurs if there are two or more consecutive underbars ( ) at any location in *In*.
- An error occurs if there is an underbar ( ) between the minus (-) or plus (+) sign and the number of *In*.
- If the value of *In* exceeds the accuracy of the data type of *Out*, the value is rounded.

- If the value 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 value 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The text string in *In* does not express a number.
  - b) The text string in *In* has a decimal point but not a fractional part.



- 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.

## Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | 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<br>2.50 → 2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2             |

## Valid Range

The valid ranges for *In* and *Out* depend on their data types. For the valid value range for each data type, refer to *Valid Range* on page 2-278 for **\*\*\_TO\_\*\*** (Integer-to-Integer Conversion Group), *Valid Range* on page 2-286 for **\*\*\_TO\_\*\*** (Bit String-to-Integer Conversion Group), and *Valid Range* on page 2-293 for **\*\*\_TO\_\*\*** (Real Number-to-Integer Conversion Group).

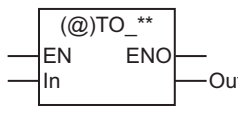
For detailed specifications when *In* is STRING data, refer to *Valid Range* on page 2-318 for **STRING\_TO\_\*\*** (Text String-to-Integer Conversion Group).

## 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.
  - a) If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
  - b) If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated.
- Observe the following precautions if *In* is STRING data.
  - a) If the first character in *In* is a minus sign (-) or a plus sign (+), it is processed as the sign.
  - b) 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.
  - a) *In* is STRING data, but the text string 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/<br>FUN | Graphic expression  | ST expression   |
|-------------|-----------------------------|------------|---|---|
| TO_**       | Bit String Conversion Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=TO_**(In);<br><br>**** must be a bit string data type. |

## Variables

|     | 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 *Valid Range* on page 2-328 for details.

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

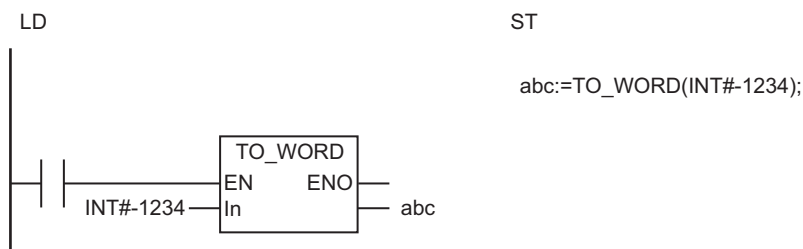
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | 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.





## Valid Range

The valid ranges for *In* and *Out* depend on their data types. For the valid value range for each data type, refer to *Valid Range* on page 2-281 for **\*\*\_TO\_\*\*\*** (Integer-to-Bit String Conversion Group), and *Valid Range* on page 2-289 for **\*\*\_TO\_\*\*\*** (Bit String-to-Bit String Conversion Group).

For detailed specifications when *In* is STRING data, refer to *Valid Range* on page 2-320 for **STRING\_TO\_\*\*** (Text String-to-Bit String Conversion Group).

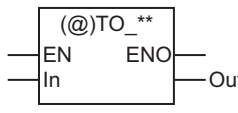
## Precautions for Correct Use

- 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* is STRING data, but the text string 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/<br>FUN | Graphic expression  | ST expression   |
|-------------|------------------------------|------------|---|---|
| TO_**       | Real Number Conversion Group | FUN        |  <p>*** must be a real number data type.</p> | Out:=TO_**(In);<br><br>*** must be a real number data type. |

## Variables

|     | 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          | ---  | ---     |

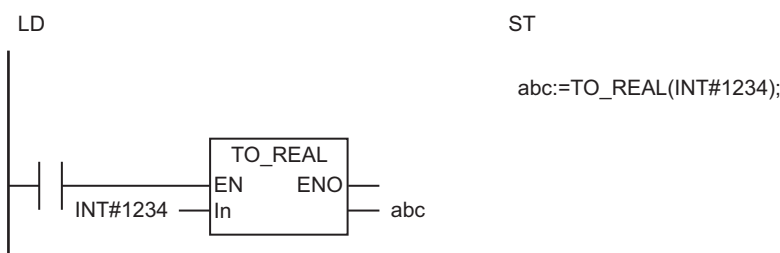
- \*1. The valid ranges depend on the data types of *In* and *Out*. Refer to *Valid Range* on page 2-330 for details.
- \*2. For STRING data, the valid range is 311 bytes max. (310 single-byte alphanumeric characters plus the final NULL character).
- \*3. If you omit the input parameter, the default value is not applied. A building error will occur.

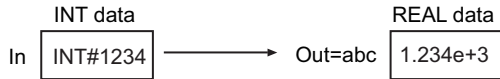
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|------------------|-------|--|------|-----|----|--------|
|     | 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.





## Valid Range

The valid ranges for *In* and *Out* depend on their data types. For the valid value range for each data type, refer to *Valid Range* on page 2-284 for **\*\*\_TO\_\*\*** (Integer-to-Real Number Conversion Group), *Valid Range* on page 2-291 for **\*\*\_TO\_\*\*** (Bit String-to-Real Number Conversion Group), and *Valid Range* on page 2-298 for **\*\*\_TO\_\*\*** (Real Number-to-Real Number Conversion Group).

For detailed specifications when *In* is STRING data, refer to *Function* on page 2-321 for **STRING\_TO\_\*\*** (Text String-to-Real Number Conversion Group).

## Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* is STRING data, but the text string does not express a number.

# EnumToNum

The EnumToNum instruction converts enumeration data to DINT data.

| Instruction | Name                   | FB/<br>FUN | Graphic expression | ST expression       |
|-------------|------------------------|------------|--------------------|---------------------|
| EnumToNum   | Enumeration-to-Integer | FUN        |                    | Out:=EnumToNum(In); |



## 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.

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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  |             |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |
| Out |             |             |      |       |       |          |      |       |       |      |     | OK   |                   |      |  |      |      |     |    |        |  |

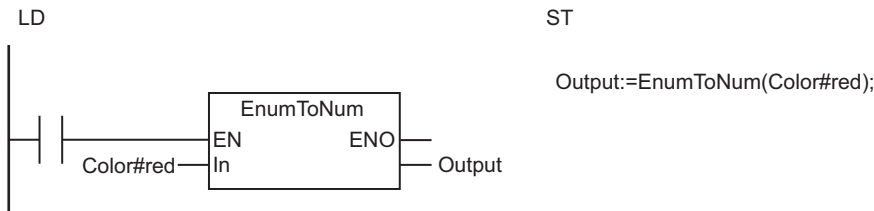
## Function

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.



## Sample Programming

In this sample, the operating mode of the user program is defined with enumerated data type Enum-Mode.

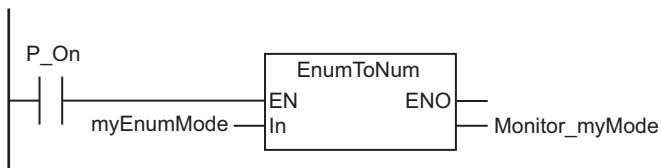
To monitor the operating mode on the HMI, the value of variable *myEnumMode* (an enumeration with a data type of EnumMode) is converted and the converted value is output to DINT variable *Monitor\_myMode*. For example, if the value of *myEnumMode* is mode2, the value of *Monitor\_myMode* will be 2.

## 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/<br>FUN | Graphic expression | ST expression         |
|-------------|------------------------|------------|--------------------|-----------------------|
| NumToEnum   | Integer-to-Enumeration | FUN        |                    | NumToEnum(In, InOut); |



## 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.

## Variables

|       | 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.<br>FALSE: Instruction was not executed or an error occurred. | Depends on data type. | ---  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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          |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |  |

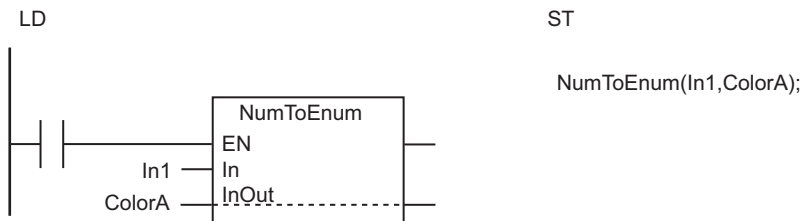
## Function

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*.



## 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.

## 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.

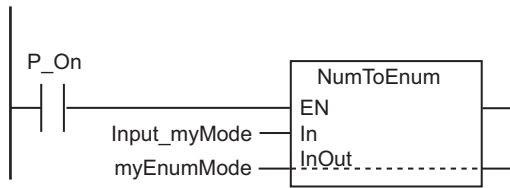
In the user program, the value of *Input\_myMode* is converted and the converted value is output to variable *myEnumMode* (an enumeration with a data type of EnumMode). For example, if the value of *Input\_myMode* is 1, the value of *myEnumMode* will be mode1.

## 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 convert real numbers to integers.

TRUNC : Truncates a real number to an integer.

Round : Rounds up or down a real number to the nearest integer.

RoundUp : Rounds up a real number to the nearest integer.

| Instruction | Name                  | FB/<br>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

|     | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-----|-------------------|--------|-------------------|-----------------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | Depends on data type. | ---  | *1      |
| Out | Conversion result | Output | Conversion result | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.

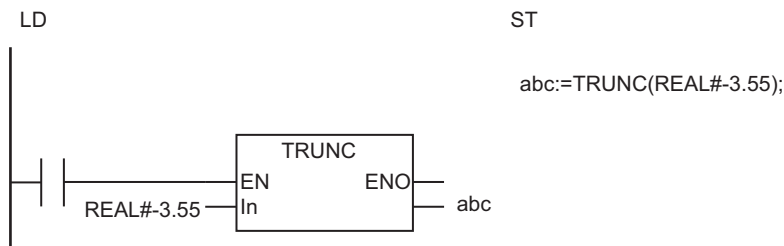
### TRUNC

The TRUNC instruction truncates the number at the first decimal digit.

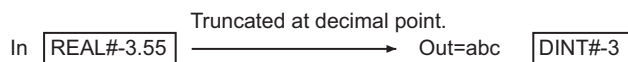
The following example for the TRUNC instruction is for when *In* is REAL#-3.55.



The value of variable *abc* will be DINT#-3.



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.



## Round

The Round instruction rounds the number at the first decimal digit.

### ● Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | Examples   |
|--------------------------|---|--|
| Less than 0.5            | The fractional part is truncated.   | 1.49 → 1<br>-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<br>2.50 → 2<br>-1.50 → -2<br>-2.50 → -2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2<br>-1.51 → -2                           |

## RoundUp

The RoundUp instruction rounds up the number at the first decimal digit.

## Differences in Operation

The following table shows differences in operation between these three instructions.

| 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 |

| Input value | Output value |         |         |
|-------------|--------------|---------|---------|
|             | TRUNC        | Round   | RoundUp |
| REAL#-1.4   | DINT#-1      | DINT#-1 | DINT#-2 |
| REAL#-2.5   | DINT#-2      | DINT#-2 | 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

| Instruction                        | Name  | Page       |
|------------------------------------|---|------------|
| AND (&), OR, and XOR               | Logical AND/Logical OR/Logical Exclusive OR   | page 2-340 |
| XORN                               | Logical Exclusive NOR   | page 2-343 |
| NOT                                | Bit Reversal  | page 2-345 |
| AryAnd, AryOr, AryXor, and AryXorN | Array Logical AND/Array Logical OR/Array Logical Exclusive OR/Array Logical Exclusive NOR | page 2-347 |

# AND (&), OR, and XOR

These instructions perform the following operations on each corresponding bit of multiple Boolean variables or bit strings.

AND (&) : Logical AND  
 OR : Logical OR  
 XOR : Logical Exclusive OR

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression                                  |
|-------------|----------------------|------------|--------------------|--|
| AND (&)     | Logical AND          | FUN        |                    | Out:=In1 AND ..AND InN;<br>Out:=In1 & ..& InN; |
| OR          | Logical OR           | FUN        |                    | Out:=In1 OR ..OR InN;                          |
| XOR         | Logical Exclusive OR | FUN        |                    | Out:=In1 XOR ..XOR InN;                        |

## Variables

|            | Meaning           | I/O    | Description                   | Valid range           | Unit | Default |
|------------|-------------------|--------|-------------------------------|-----------------------|------|---------|
| In1 to InN | Data to process   | Input  | Data to process<br>N = 2 to 5 | Depends on data type. | ---  | 0*1     |
| Out        | Processing result | Output | Processing result             | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, where N is 3, if 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.

|            | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 the same data type as <i>In1</i> to <i>InN</i> |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |



## OR

If both bits are FALSE, the processing result is FALSE. Otherwise, the processing result is TRUE.

| <i>In1 bit</i> | <i>In2 bit</i> | <i>Out bit</i> |
|----------------|----------------|----------------|
| FALSE          | FALSE          | FALSE          |
| FALSE          | TRUE           | TRUE           |
| TRUE           | FALSE          | TRUE           |
| TRUE           | TRUE           | TRUE           |

## XOR

If both bits are the same, the processing result is FALSE. If one bit is TRUE and the other is FALSE, the processing result is TRUE.

| <i>In1 bit</i> | <i>In2 bit</i> | <i>Out bit</i> |
|----------------|----------------|----------------|
| FALSE          | FALSE          | FALSE          |
| FALSE          | TRUE           | TRUE           |
| TRUE           | FALSE          | TRUE           |
| TRUE           | TRUE           | FALSE          |

## 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 same data type should be used for *In1* to *InN* and *Out*.  
Otherwise, a building error will occur.

# XORN

The XORN instruction performs a logical exclusive NOR operation on each corresponding bit of multiple Boolean variables or bit strings.

| Instruction | Name                  | FB/<br>FUN | Graphic expression | ST expression                    |
|-------------|-----------------------|------------|--------------------|----------------------------------|
| XORN        | Logical Exclusive NOR | FUN        |                    | Out:=In1 XOR NOT .. XOR NOT InN; |

## Variables

|            | Meaning           | I/O    | Description                   | Valid range           | Unit | Default |
|------------|-------------------|--------|-------------------------------|-----------------------|------|---------|
| In1 to InN | Data to process   | Input  | Data to process<br>N = 2 to 5 | Depends on data type. | ---  | 0*1     |
| Out        | Processing result | Output | Processing result             | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, where N is 3, if 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.

|            | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 the same data type as <i>In1</i> to <i>InN</i> |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |

## Function

These instructions perform bitwise operations on data to process, *In1* to *InN*, which are multiple Boolean variables or bit strings.

The same data type should be used for *In1* to *InN* and *Out*.

If there are three or more data to process, operations are performed as below.

- 1** Perform operations on *In1* and *In2*.
- 2** Perform operations on the result of step 1 and *In3*.
- 3** Perform operations on the result of step 2 and *In4*.

Continue operations as above.

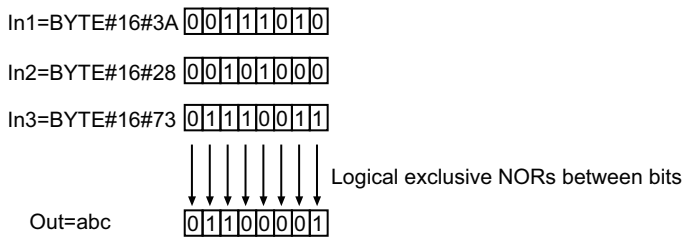
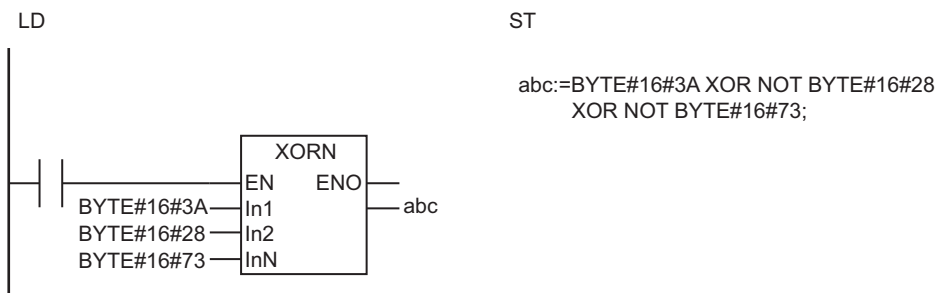
## XORN

This instruction outputs operation results as below:

If both bits 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 shows an example where *In1*, *In2* and *In3* are BYTE#16#3A, BYTE#16#28 and BYTE#16#73, respectively.

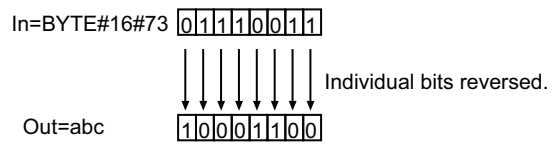


### Precautions for Correct Use

The same data type should be used for *In1* to *InN* and *Out*. Otherwise, a building error will occur.







### 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 perform the following operations on individual bits of each corresponding Boolean or bit-string element in two arrays.

- AryAnd : Logical AND
- AryOr : Logical OR
- AryXor : Logical Exclusive OR
- AryXorN : Logical Exclusive NOR

| Instruction | Name                        | FB/<br>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

|                          | Meaning            | I/O   | Description                   | Valid range           | Unit | Default |
|--------------------------|--------------------|-------|-------------------------------|-----------------------|------|---------|
| In1[] and In2[] (arrays) | Array to process   | Input | Array to process              | Depends on data type. | ---  | *1      |
| Size                     | Number of elements |       | Number of elements to process |                       |      | 1       |

|                  | Meaning                  | I/O    | Description              | Valid range           | Unit | Default |
|------------------|--------------------------|--------|--------------------------|-----------------------|------|---------|
| AryOut[] (array) | Processing results array | In-out | Processing results array | 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.

|                  | Boolean                         | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 perform bitwise operations on the specified number of elements, *Size*, from the beginning of respective arrays to process, In1[] and In2[]. The operation results are stored in the corresponding elements of AryOut[].

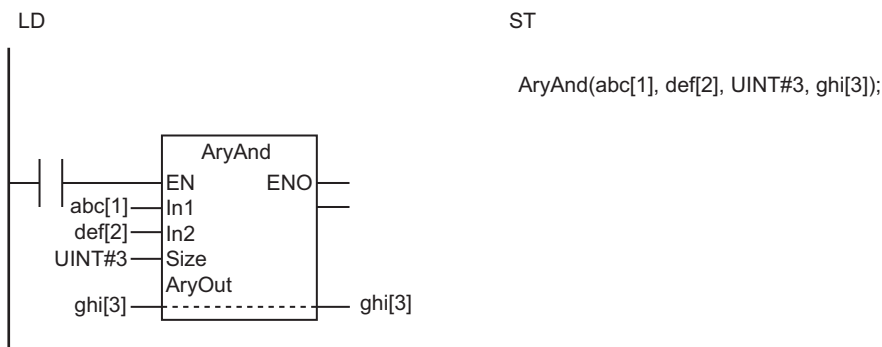
In1[], In2[] and AryOut[] must be of the same data type.

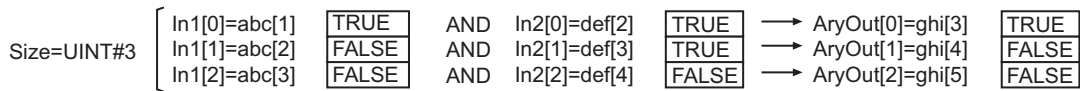
## AryAnd

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            |

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





## 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            |

## 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.
- The number of elements in AryOut[] should be equal to or more than the value of *Size*.
- The value of AryOut[] will not change if the value of *Size* is 0.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and AryOut[] will not change.
  - a) The value of *Size* exceeds the number of elements in In1[], In2[], or AryOut[].

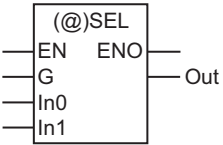


## Selection Instructions

| Instruction       | Name                        | Page       |
|-------------------|-----------------------------|------------|
| SEL               | Binary Selection            | page 2-352 |
| MUX               | Multiplexer                 | page 2-354 |
| LIMIT             | Limiter                     | page 2-357 |
| Band              | Deadband Control            | page 2-359 |
| Zone              | Dead Zone Control           | page 2-361 |
| MAX and MIN       | Maximum/Minimum             | page 2-363 |
| AryMax and AryMin | Array Maximum/Array Minimum | page 2-365 |
| ArySearch         | Array Search                | page 2-368 |

# SEL

The SEL instruction selects one of two options.

| Instruction | Name             | FB/<br>FUN | Graphic expression  | ST expression          |
|-------------|------------------|------------|---|------------------------|
| SEL         | Binary Selection | FUN        |  | Out:=SEL(G, In0, In1); |

## Variables

|             | Meaning          | I/O    | Description   | Valid range           | Unit | Default |
|-------------|------------------|--------|---|-----------------------|------|---------|
| G           | Gate             | Input  | FALSE: Selects <i>In0</i><br>TRUE: Selects <i>In1</i> | Depends on data type. | ---  | FALSE   |
| In0 and In1 | Selections       |        | Selections  |                       |      | *1      |
| Out         | Selection result | Output | Selection result                                      | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|             | Boo<br>lean                           | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.*1 |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| 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.*1 |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

\*1. A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to specify enumerations.

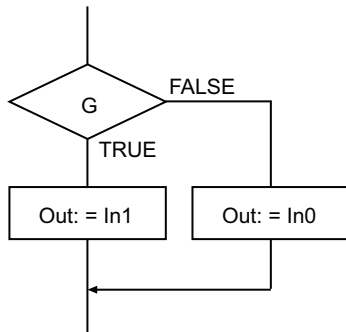
## Function

The SEL instruction selects one of two options, *In0* and *In1* (Selections).

Gate *G* specifies which of *In0* and *In1* to select.

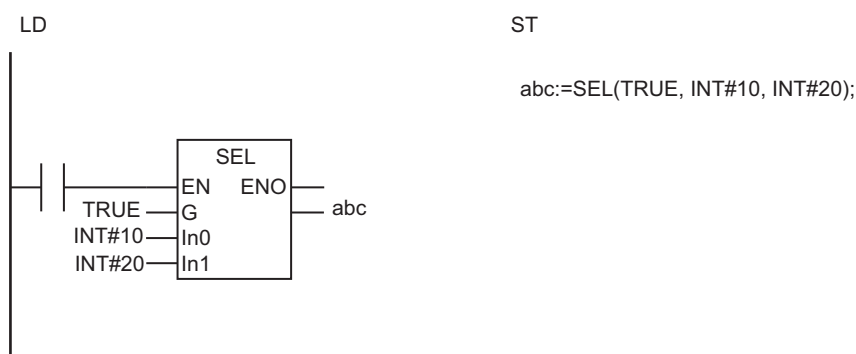
If *G* is FALSE, *In0* is assigned to *Out*. If it is TRUE, *In1* is assigned.





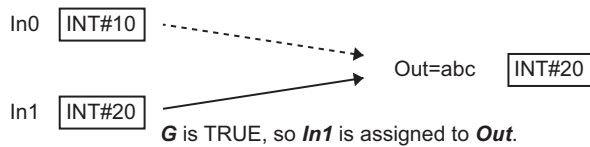
The following shows an example where *In0*, *In1*, and *G* are INT#10, INT#20, and TRUE, respectively.

The value of variable *abc* will be INT#20.



The SEL instruction selects *In0* or *In1*.

*G* is TRUE, so *In1* (INT#20) is selected and assigned to *abc*.



## Additional Information



### Version Information

With a CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher, the instruction, *MUX* on page 2-354, can also be used.

The *MUX* instruction selects one of two to five options.

## Precautions for Correct Use

- The data types of *In0*, *In1*, and *Out* may be different, but observe the following precautions.
  - a) The valid value range of *Out* should accommodate the valid value ranges of *In0* and *In1*.
  - b) The data types of *In0*, *In1*, and *Out* should be in the same data type category. (i.e., they should not be in different categories, such as bit string and an integer, or an integer and a text string).

# MUX

The MUX instruction selects one of two to five options.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression                     |
|-------------|-------------|------------|--------------------|-----------------------------------|
| MUX         | Multiplexer | FUN        |                    | Out:=MUX(K, In0, In1, ... , InN); |

## Variables

|            | Meaning          | I/O    | Description   | Valid range           | Unit | Default |
|------------|------------------|--------|---|-----------------------|------|---------|
| K          | Selector         | Input  | 0: Selects <i>In0</i><br>1: Selects <i>In1</i><br>2: Selects <i>In2</i><br>3: Selects <i>In3</i><br>4: Selects <i>In4</i> | 0 to N                | ---  | *1      |
| In0 to In1 | Selections       |        | Selections<br>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 the 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, where N is 2, if 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.

|            | Boo<br>lean                           | Bit strings |      |       |       |          | Integers |       |          |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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<br>*1 |          |       | OK<br>*1 |      |     |      |      |                     |       |  |      |     |    |        |    |
| In0 to InN | OK                                    | 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     | 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.

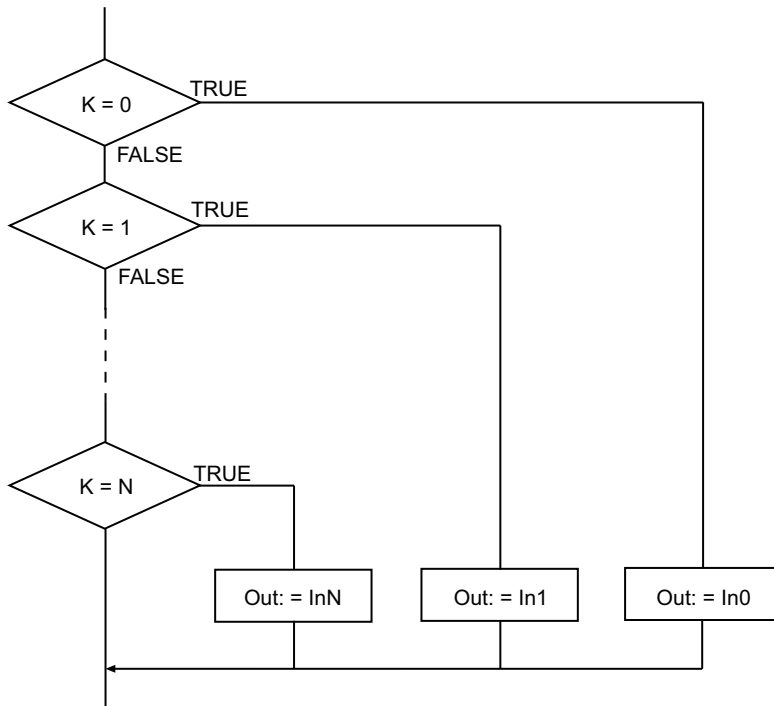
\*2. A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to specify enumerations.

## Function

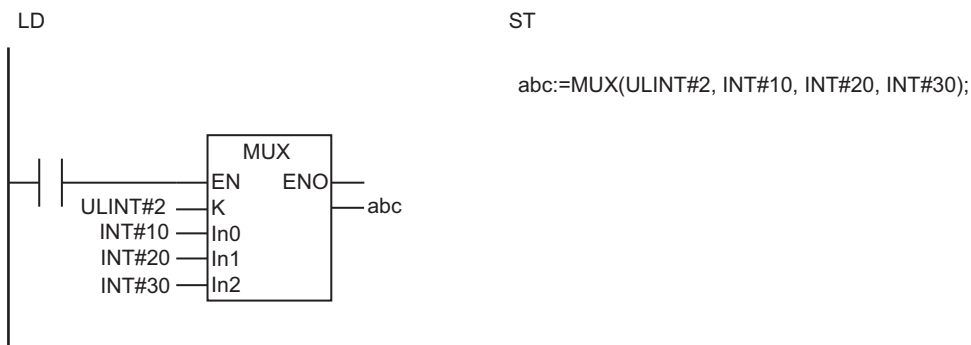
The MUX instruction selects one of two to five options, *In0* to *InN* (Selections).

Selector *K* specifies which of *In0* to *InN* to select.

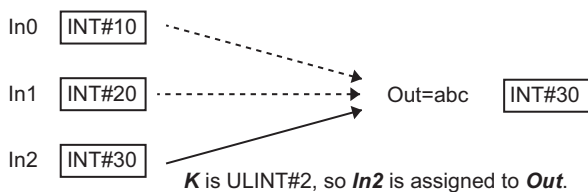
According to the value of *K*, a value is assigned to *Out*. If *K* is 0, *In0* is assigned. If it is 1, *In1* is assigned, etc.



The following shows an example where *In0*, *In1*, *In2*, and *K* are INT#10, INT#20, INT#30, and ULINT#2, respectively. The value of variable *abc* will be 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*.



### Precautions for Correct Use

- The data types of *In0* to *InN*, and *Out* may be different, but observe the following precautions.
  - a) The valid value range of *Out* should accommodate the valid value ranges of *In0* to *InN*.
  - b) The data types of *In0* to *InN*, and *Out* should be in the same data type category (i.e., they should not be in different categories such as a bit string and an integer, or an integer and a text string).
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) 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 an input variable between the specified minimum and maximum values.

| Instruction | Name    | FB/<br>FUN | Graphic expression | ST expression           |
|-------------|---------|------------|--------------------|-------------------------|
| LIMIT       | Limiter | FUN        |                    | Out:=LIMIT(MN, In, MX); |

## Variables

|     | Meaning           | I/O    | Description              | Valid range           | Unit | Default |
|-----|-------------------|--------|--------------------------|-----------------------|------|---------|
| MN  | Minimum value     | Input  | Minimum value of limiter | Depends on data type. | ---  | *1      |
| In  | Data to limit     |        | Data to limit            |                       |      |         |
| MX  | Maximum value     |        | Maximum value of limiter |                       |      |         |
| 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.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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    |   |      |     |    |        |

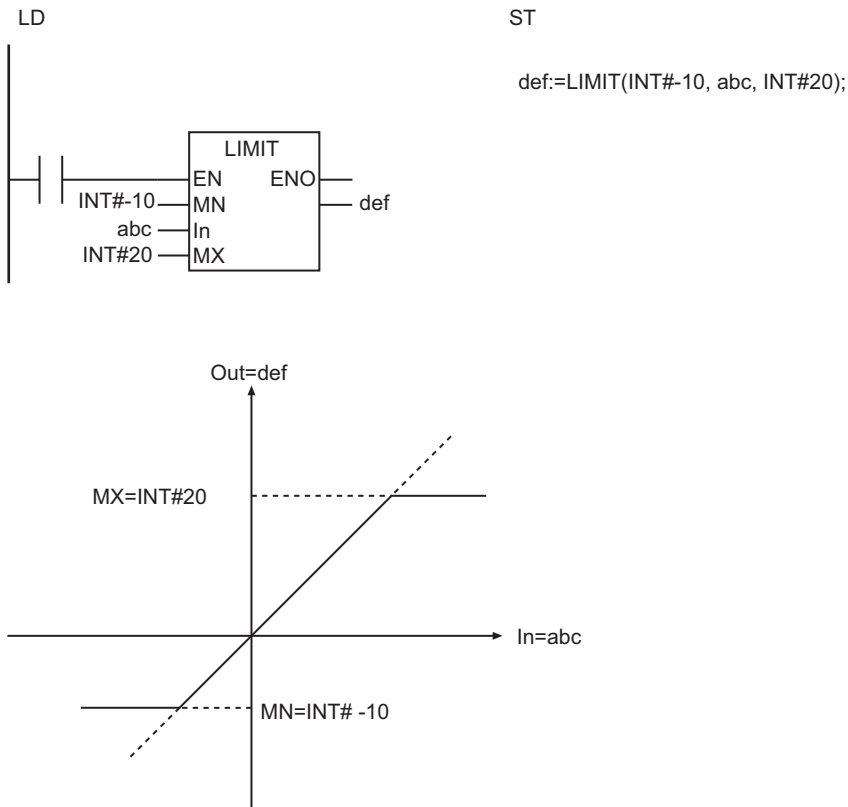
## Function

The LIMIT instruction limits the value of data to limit *In* between the maximum value *MX* and the minimum value *MN*.

The value of processing result *Out* is as shown below.

| Value of <i>In</i>   | Value of <i>Out</i> |
|----------------------|---------------------|
| $In < MN$            | <i>MN</i>           |
| $MN \leq In \leq MX$ | <i>In</i>           |
| $MX < In$            | <i>MX</i>           |

The following shows an example where *MN* and *MX* are INT#-10 and INT#20, respectively.



### Precautions for Correct Use

- The data types of *In*, *MN*, *MX*, and *Out* may be different, but observe the following precautions.
  - a) The valid value range of *Out* should accommodate the valid value ranges of *In*, *MN*, and *MX*.
  - b) Do not combine signed integers (SINT, INT, DINT, and LINT) and unsigned integers (USINT, UINT, UDINT, and ULINT) together for *In*, *MN*, and *MX*.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *MX* is smaller than the value of *MN*.

# Band

The Band instruction performs deadband control.

| Instruction | Name             | FB/<br>FUN | Graphic expression | ST expression          |
|-------------|------------------|------------|--------------------|------------------------|
| Band        | Deadband Control | FUN        |                    | Out:=Band(MN, In, MX); |

## Variables

|     | Meaning           | I/O    | Description               | Valid range           | Unit | Default |
|-----|-------------------|--------|---------------------------|-----------------------|------|---------|
| MN  | Minimum value     | Input  | Minimum value of deadband | Depends on data type. | ---  | *1      |
| In  | Data to control   |        | Data to control           |                       |      |         |
| MX  | Maximum value     |        | Maximum value of deadband |                       |      |         |
| 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.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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   |      |      |     |    |        |

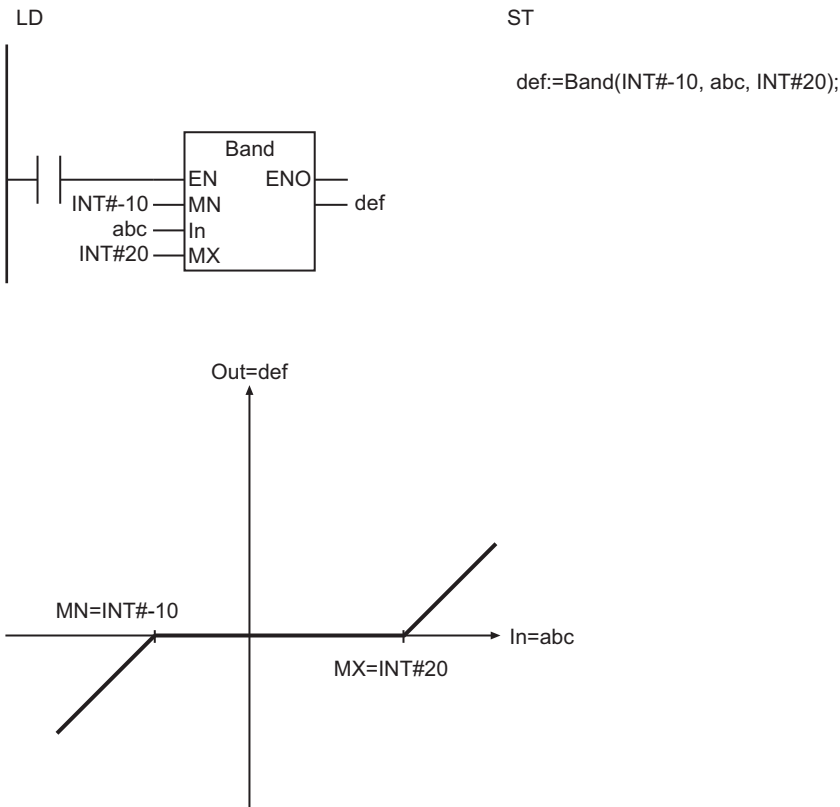
## Function

The Band instruction limits the value of data to control *In* with the maximum value *MX* and the minimum value *MN*.

The value of processing result *Out* is as shown below.

| Value of <i>In</i>   | Value of <i>Out</i> |
|----------------------|---------------------|
| $In < MN$            | $In - MN$           |
| $MN \leq In \leq MX$ | 0                   |
| $MX < In$            | $In - MX$           |

The following shows an example where *MN* and *MX* are INT#-10 and INT#20, respectively.



### Precautions for Correct Use

- The data types of *In*, *MN*, *MX*, and *Out* may be different, but observe the following precautions.
  - The valid value range of *Out* should accommodate the valid value ranges of *In*, *MN*, and *MX*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the values of *In*, *MN*, and *MX* are positive infinity or negative infinity, the value of *Out* is as shown below.

| Value of <i>In</i> | Value of <i>MN</i> | Value of <i>MX</i> | Value of <i>Out</i> |
|--------------------|--------------------|--------------------|---------------------|
| + ∞                | + ∞                | + ∞                | 0                   |
|                    |                    | - ∞                | Error               |
|                    | - ∞                | + ∞                | 0                   |
|                    |                    | - ∞                | + ∞                 |
| - ∞                | + ∞                | + ∞                | - ∞                 |
|                    |                    | - ∞                | Error               |
|                    | - ∞                | + ∞                | 0                   |
|                    |                    | - ∞                | 0                   |

- An error will occur 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 result exceeds the valid range of *Out*.



# Zone

The Zone instruction adds a bias value to the input value.

| Instruction | Name              | FB/<br>FUN | Graphic expression | ST expression                |
|-------------|-------------------|------------|--------------------|------------------------------|
| Zone        | Dead Zone Control | FUN        |                    | Out:=Zone(BiasN, In, BiasP); |

## Variables

|       | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-------|-------------------|--------|-------------------|-----------------------|------|---------|
| BiasN | Negative bias     | Input  | Negative bias     | Depends on data type. | ---  | *1      |
| In    | Data to control   |        | Data to control   |                       |      |         |
| BiasP | Positive bias     |        | Positive bias     |                       |      |         |
| 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.

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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

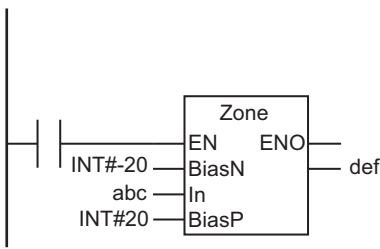
The Zone instruction controls the value of data to control *In* according to the positive bias *BiasP* and the negative bias *BiasN*.

The value of processing result *Out* is as shown below.

| Value of <i>In</i> | Value of <i>Out</i> |
|--------------------|---------------------|
| $In < 0$           | $In + BiasN$        |
| $In = 0$           | 0                   |
| $0 < In$           | $In + BiasP$        |

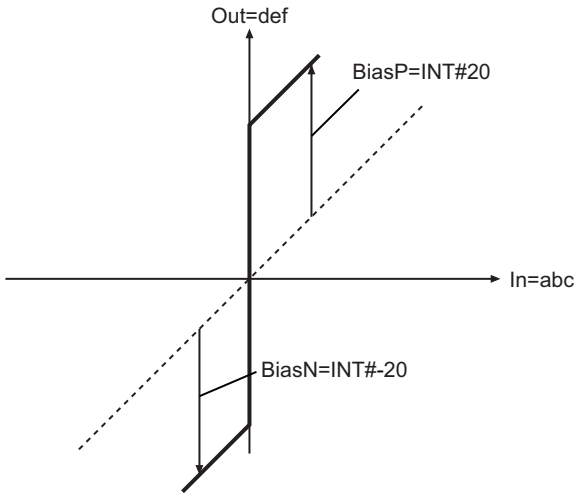
The following shows an example where *BiasP* and *BiasN* are INT#20 and INT#-20, respectively.

LD



ST

def:=Zone(INT#-20, abc, INT#20);



### Precautions for Correct Use

- The data types of *In*, *BiasP*, *BiasN*, and *Out* may be different, but observe the following precautions.
  - The valid value range of *Out* should accommodate the valid value ranges of *In*, *BiasP*, and *BiasN*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the values of *In*, *BiasP*, and *BiasN* are positive infinity or negative infinity, the value of *Out* is as shown below.

| Value of <i>In</i> | Value of <i>BiasP</i> | Value of <i>BiasN</i> | Value of <i>Out</i> |
|--------------------|-----------------------|-----------------------|---------------------|
| + ∞                | + ∞                   | + ∞                   | + ∞                 |
|                    |                       | - ∞                   | + ∞                 |
|                    | - ∞                   | + ∞                   | Error               |
|                    |                       | - ∞                   | 0                   |
| - ∞                | + ∞                   | + ∞                   | 0                   |
|                    |                       | - ∞                   | - ∞                 |
|                    | - ∞                   | + ∞                   | Error               |
|                    |                       | - ∞                   | - ∞                 |

- An error will occur 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 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/<br>FUN | Graphic expression | ST expression                  |
|-------------|---------|------------|--------------------|--------------------------------|
| MAX         | Maximum | FUN        |                    | Out:=MAX(In1, In2, ... , InN); |
| MIN         | Minimum | FUN        |                    | Out:=MIN(In1, In2, ... , InN); |

## Variables

|            | 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*1     |
| Out        | Search result   | Output | Search result                      | Depends on data type. | ---  | ---     |

- \*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, where N is 3, if 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.

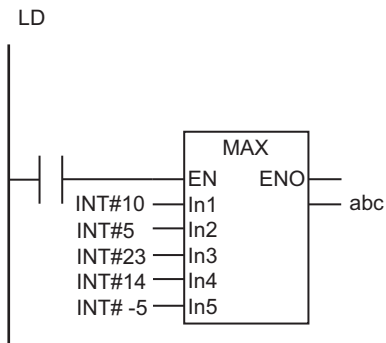
|            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|---------------------|-------|--|------|-----|----|--------|
|            | 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

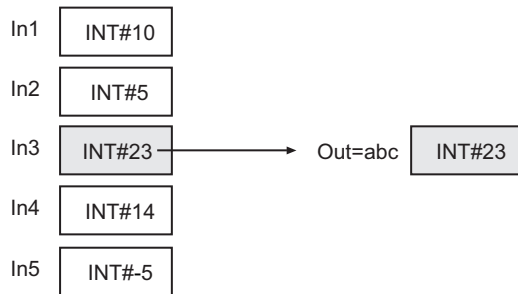
The MAX instruction finds the largest value of two to five data to process, *In1* to *InN*.

The following shows an example where *In1*, *In2*, *In3*, *In4*, and *In5* are INT#10, INT#5, INT#23, INT#14, and INT#-5, respectively.



ST

```
abc:=MAX(INT#10, INT#5, INT#23, INT#14, INT#-5);
```



## MIN

The MIN instruction finds the smallest value of two to five data to process, *In1* to *InN*.

### Additional Information

To find the largest or smallest of six or more values, use *AryMax* and *AryMin* on page 2-365.

### Precautions for Correct Use

- The data types of *In1* to *InN*, and *Out* may be different, but observe the following precautions.
  - a) The valid value range of *Out* should accommodate the valid value ranges of *In1* to *InN*.
  - b) Do not combine signed integers (SINT, INT, DINT, and LINT) and unsigned integers (USINT, UINT, UDINT, and ULINT) together for *In1* to *InN*.
- If *In1* to *InN* contain real numbers, desired results may not be returned due to error.

# AryMax and AryMin

AryMax : Finds elements with the largest value in a one-dimensional array.

AryMin : Finds elements with the smallest value in a one-dimensional array.

| Instruction | Name          | FB/<br>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

|              | Meaning                      | I/O    | Description                                | Valid range           | Unit | Default |
|--------------|------------------------------|--------|--|-----------------------|------|---------|
| In[] (array) | Array to search              | Input  | Array to search                            | Depends on data type. | ---  | *1      |
| 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                               |                       |      |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
| Size         |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |          |          |          |          |
| InOutPos     |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |          |          |          |          |
| Out          |             |             |      |       |       | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK                  | OK    | OK<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
| Num          |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |          |          |          |          |

\*1. 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*[], which starts from *In*[0].

The search result value is assigned to *Out*, the element number where the value is found is assigned to *InOutPos*, and the number of times that the value is found is assigned to *Num*.

If *Num* is greater than 1, the lowest number of the elements where the search result value is found is assigned to *InOutPos*.

Other data types than integer and real number are handled as below.

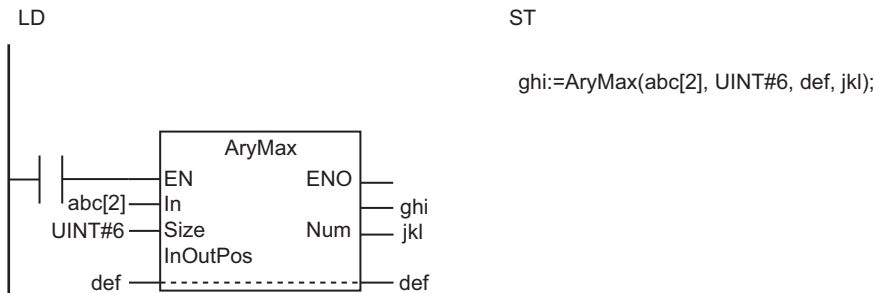
| Data type        | Description  |
|------------------|--|
| TIME             | The numerically larger value is considered to be larger.   |
| DATE, TOD, or DT | Later date or time of day is considered to be larger.  |
| STRING           | The specifications are the same as for the instructions, <i>LTascii</i> , <i>LEascii</i> , <i>GTascii</i> , and <i>GEascii</i> on page 2-115. Refer to the specified page for details. |

## AryMax

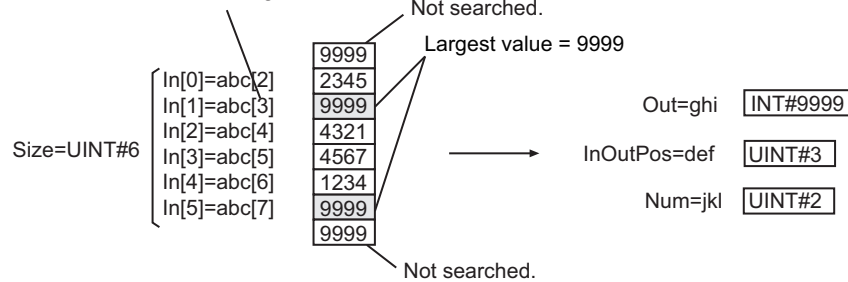
The AryMax instruction finds the largest value.

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].



The lowest element number that contains the largest value is 3.



## AryMin

The AryMin instruction finds the smallest value.

## Additional Information

When you compare TIME, DT, or TOD data, adjust the value precision so that values of those data types can be compared based on the same precision.

You can use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## Precautions for Correct Use

- If the data types of *In[]* and *Out* are different, make sure that the valid value range of *Out* accommodates the valid value range of *In[]*.
- If *In[]* contains a real number, a desired result may not be returned due to error.
- Always use 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* contains only null characters.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* is outside the valid range.
  - b) The value of *Size* exceeds the array area of *In[]*.
  - c) *In[]* is not a one-dimensional array.
  - d) *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/<br>FUN | Graphic expression | ST expression                                  |
|-------------|--------------|------------|--------------------|--|
| ArySearch   | Array Search | FUN        |                    | Out:=ArySearch(In, Size, Key, In-OutPos, Num); |

## Variables

|              | Meaning                      | I/O    | Description                                    | Valid range           | Unit | Default |
|--------------|------------------------------|--------|--|-----------------------|------|---------|
| In[] (array) | Array to search              | Input  | Array to search                                | Depends on data type. | ---  | *1      |
| Size         | Number of elements to search |        | Number of elements in In[] to search           |                       |      | 1       |
| Key          | Search key                   |        | Value to search for                            | Depends on data type. |      | *1      |
| InOutPos     | Found element number         | In-out | Array element number where the value was found | Depends on data type. | ---  | ---     |
| Out          | Search result                | Output | Search result                                  | Depends on data type. | ---  | ---     |
| Num          | Number found                 |        | Number found                                   |                       |      |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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<br>*1                                  | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK     |
|              | Arrays of enumerations can also be specified.       |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |          |          |          |        |
| Size         |   |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |          |          |          |        |
| Key          | Must be the same data type as the elements of In[]. |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |          |          |          |        |
| InOutPos     |   |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |          |          |          |        |
| Out          | OK  |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |          |          |          |        |
| Num          |   |             |      |       |       |       | 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.



## 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.

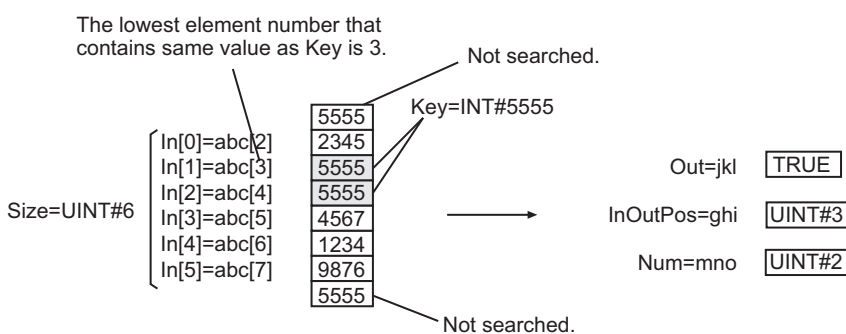
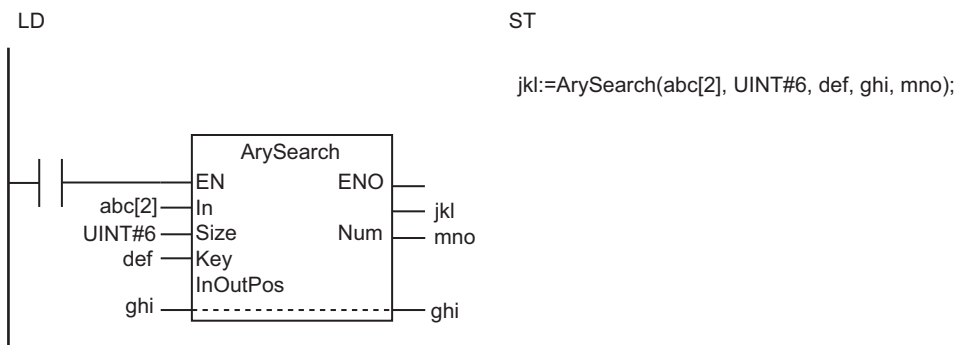
| Element with the same value as Key | Out   | InOutPos  | Num   |
|------------------------------------|-------|---|---|
| Exists.                            | TRUE  | Lowest element number that contains the same value as Key | Number of elements with the same value as Key |
| Does not exist.                    | FALSE | Does not change.  | 0   |

Other data types than integer and real number are handled as below.

| Data type        | Description  |
|------------------|--|
| TIME             | The numerically larger value is considered to be larger. |
| DATE, TOD, or DT | Later date or time of day is considered to be larger.    |

The following shows an example where *Size* is UINT#6.

The input parameter that is passed to In[] is abc[2], so the search starts from abc[2].



## Additional Information

When you compare TIME, DT, or TOD data, adjust the value precision so that values of those data types can be compared based on the same precision.

You can use the following instructions to adjust the accuracy: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

### 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 directly.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out*, *Num*, and *InOutPos* will not change.
  - a) The value of *Size* exceeds the array area of *In[]*.
  - b) *In[]* is STRING data and it does not end in a NULL character.
  - c) *In[]* is not a one-dimensional array.

# Data Movement Instructions

| Instruction                                 | Name   | Page       |
|---|--|------------|
| MOVE  | Move   | page 2-372 |
| MoveBit                                     | Move Bit   | page 2-375 |
| MoveDigit                                   | Move Digit   | page 2-377 |
| TransBits                                   | Move Bits  | page 2-379 |
| MemCopy                                     | Memory Copy  | page 2-381 |
| SetBlock                                    | Block Set  | page 2-383 |
| Exchange                                    | Data Exchange  | page 2-385 |
| AryExchange                                 | Array Data Exchange                                    | page 2-387 |
| AryMove                                     | Array Move   | page 2-389 |
| Clear                                       | Initialize   | page 2-391 |
| Copy**ToNum (Bit String to Signed Integer)  | Bit Pattern Copy (Bit String to Signed Integer) Group  | page 2-393 |
| Copy**To*** (Bit String to Real Number)     | Bit Pattern Copy (Bit String to Real Number) Group     | page 2-395 |
| CopyNumTo** (Signed Integer to Bit String)  | Bit Pattern Copy (Signed Integer to Bit String) Group  | page 2-397 |
| CopyNumTo** (Signed Integer to Real Number) | Bit Pattern Copy (Signed Integer to Real Number) Group | page 2-399 |
| Copy**To*** (Real Number to Bit String)     | Bit Pattern Copy (Real Number to Bit String) Group     | page 2-401 |
| Copy**ToNum (Real Number to Signed Integer) | Bit Pattern Copy (Real Number to Signed Integer) Group | page 2-403 |

# MOVE

The MOVE instruction moves the value of a constant or variable to another variable.

| Instruction | Name | FB/<br>FUN | Graphic expression | ST expression |
|-------------|------|------------|--------------------|---------------|
| MOVE        | Move | FUN        |                    | Out:=In;      |

## Variables

|     | Meaning          | I/O    | Description      | Valid range           | Unit | Default |
|-----|------------------|--------|------------------|-----------------------|------|---------|
| In  | Move source      | Input  | Move source      | Depends on data type. | ---  | *1      |
| Out | Move destination | Output | Move destination | Depends on data type. | ---  | *1      |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

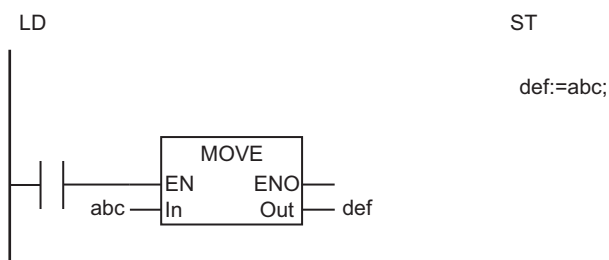
|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |    |
|-----|--|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|----|
|     | 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     | 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     | OK |
|     | Must be the same data type as <i>In</i> if <i>In</i> is an enumeration, array element, structure, or structure member.<br>Must be an array with the same data type, size and subscripts as <i>In</i> if <i>In</i> is an array. |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |    |

## Function

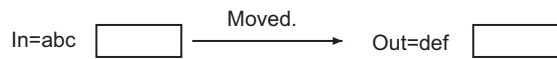
The MOVE instruction moves the value in move source *In* to move destination *Out*.

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*.



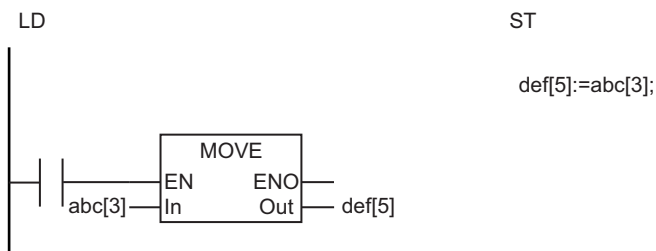
The MOVE instruction moves the value of *In* to *Out*.



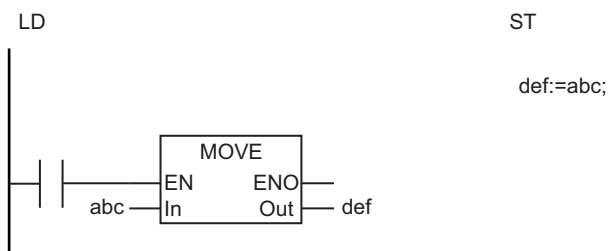
## Additional Information

- 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

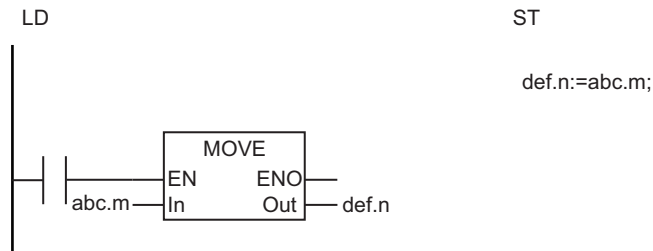


### Moving All Array Elements

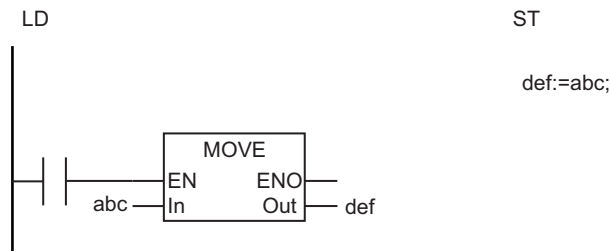


- When moving a structure, you can move either one member or all of the members in the structure. To move only one member, specify the member. To move the entire structure, give only the structure name.

### Moving One Member of a Structure



### Moving the Entire Structure



- 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 in the same data type group as shown below. 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, *Out* must have the same data type as *In*.
- If *In* is an array, an array of the same data type, size, and subscripts as *In* must be used for *Out*.

# MoveBit

The MoveBit instruction moves one bit in a bit string.

| Instruction | Name     | FB/<br>FUN | Graphic expression | ST expression                        |
|-------------|----------|------------|--------------------|--------------------------------------|
| MoveBit     | Move Bit | FUN        |                    | MoveBit(In, InPos, InOut, InOutPos); |

## Variables

|          | Meaning              | I/O    | Description                              | Valid range                                 | Unit | Default |
|----------|----------------------|--------|--|---|------|---------|
| In       | Move source          | Input  | Move source                              | Depends on data type.                       | ---  | *1      |
| InPos    | Move source bit      |        | Source bit position in <i>In</i>         | 0 to the number of bits in <i>In</i> - 1    |      | 0       |
| InOutPos | Move destination bit |        | Destination bit position in <i>InOut</i> | 0 to the number of bits in <i>InOut</i> - 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.

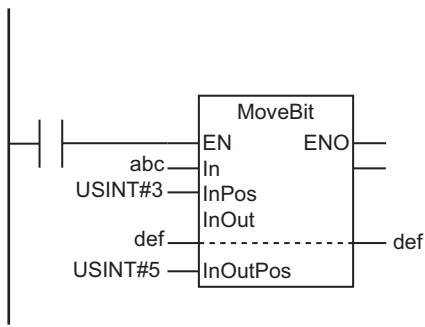
|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|----------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|          | 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 source bit position *InPos* in move source *In* to the destination bit position *InOutPos* in move destination *InOut*.

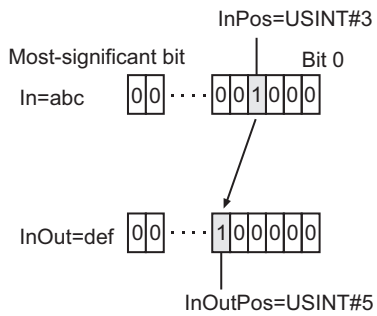
The following shows an example where *InPos* is USINT#3 and *InOutPos* is USINT#5.

LD



ST

MoveBit(abc, USINT#3, def, USINT#5);



### Precautions for Correct Use

- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - a) The value of *InPos* is outside the valid range.
  - b) The value of *InOutPos* is outside the valid range.



# MoveDigit

The MoveDigit instruction moves digits (4 bits per digit) in a bit string.

| Instruction | Name       | FB/<br>FUN | Graphic expression | ST expression                                |
|-------------|------------|------------|--------------------|--|
| MoveDigit   | Move Digit | FUN        |                    | MoveDigit(In, InPos, InOut, InOutPos, Size); |

## Variables

|          | Meaning                | I/O    | Description                                | Valid range                                    | Unit | Default |
|----------|------------------------|--------|--|--|------|---------|
| In       | Move source            | Input  | Move source                                | Depends on data type.                          | ---  | *1      |
| InPos    | Move source digit      |        | Source digit position in <i>In</i>         | 0 to the number of bits in <i>In</i> /4 - 1    |      | 0       |
| InOutPos | Move destination digit |        | Destination digit position in <i>InOut</i> | 0 to the number of bits in <i>InOut</i> /4 - 1 |      |         |
| Size     | Number of digits       |        | Number of digits to move                   | 0 to the number of bits in <i>In</i> /4        |      |         |
| InOut    | Move destination       | In-out | Move destination                           | Depends on data type.                          | ---  | ---     |
| Out      | Return value           | Output | Always TRUE                                | TRUE only                                      | ---  | ---     |

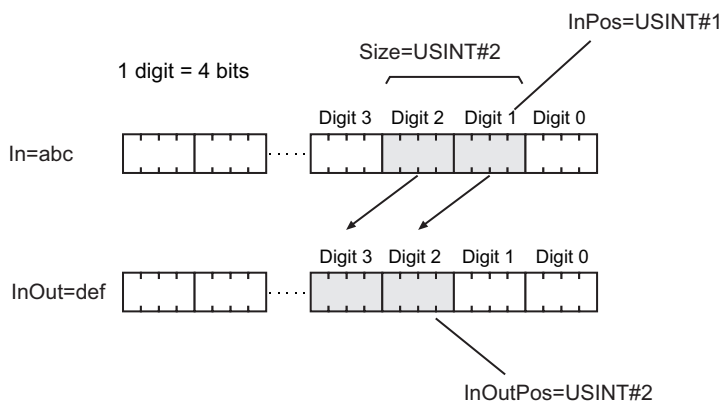
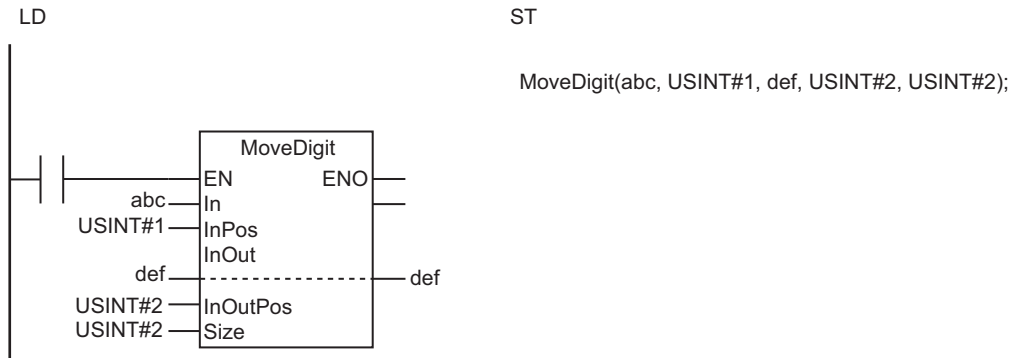
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 source digit position *InPos* in move source *In* to the destination digit position *InOutPos* in move destination *InOut*. One digit represents four bits.

The following shows an example where *InPos* is USINT#1, *InOutPos* is USINT#2, and *Size* is 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 in 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 this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - a) The value of *InPos* is outside the valid range.
  - b) The value of *InOutPos* is outside the valid range.
  - c) The value of *Size* is outside the valid range.

# TransBits

The TransBits instruction moves one or more bits in a bit string.

| Instruction | Name      | FB/<br>FUN | Graphic expression | ST expression                                |
|-------------|-----------|------------|--------------------|--|
| TransBits   | Move Bits | FUN        |                    | TransBits(In, InPos, InOut, InOutPos, Size); |

## Variables

|          | Meaning              | I/O    | Description                              | Valid range                                 | Unit | Default |
|----------|----------------------|--------|--|---|------|---------|
| In       | Move source          | Input  | Move source                              | Depends on data type.                       | ---  | *1      |
| InPos    | Move source bit      |        | Source bit position in <i>In</i>         | 0 to the number of bits in <i>In</i> - 1    |      | 0       |
| InOutPos | Move destination bit |        | Destination bit position in <i>InOut</i> | 0 to the number of bits in <i>InOut</i> - 1 |      |         |
| Size     | Number of bits       |        | Number of bits to move                   | 0 to the number of bits in <i>In</i>        |      |         |
| InOut    | Move destination     | In-out | Move destination                         | Depends on data type.                       | ---  | ---     |
| Out      | Return value         | Output | Always TRUE                              | TRUE only                                   | ---  | ---     |

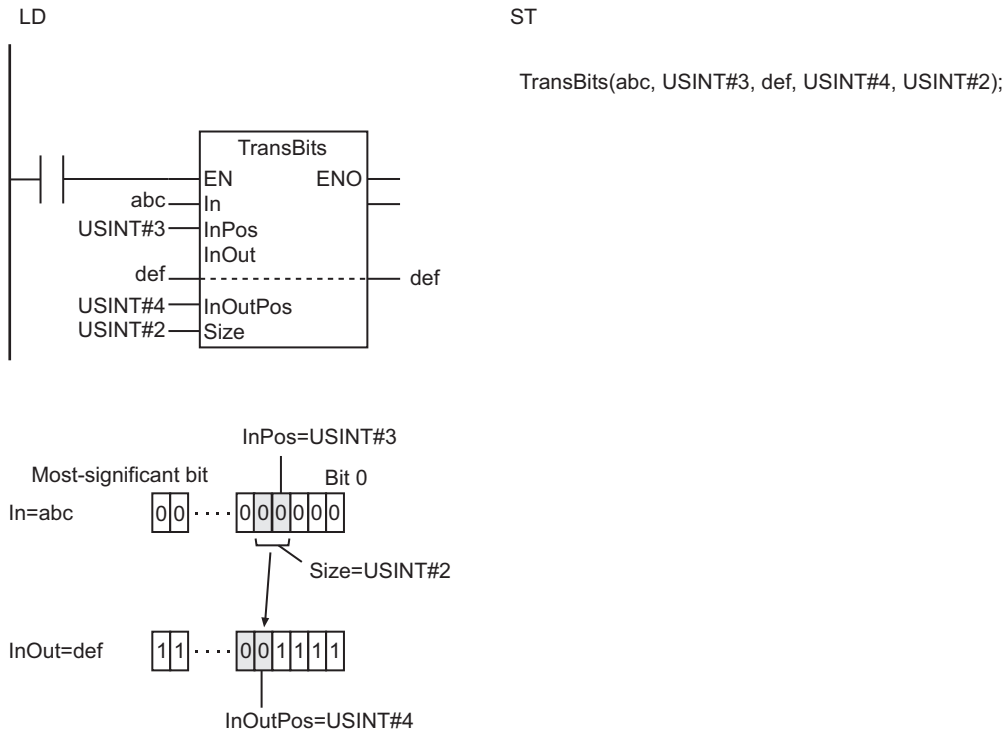
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 TransBits instruction moves *Size* bits from the source bit position *InPos* in move source *In* to the destination bit position *InOutPos* in move destination *InOut*.

The following shows an example where *InPos* is USINT#3, *InOutPos* is USINT#4, and *Size* is 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*. Otherwise 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 this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - a) The value of *InPos* is outside the valid range.
  - b) The value of *InOutPos* is outside the valid range.
  - c) The value of *Size* is outside the valid range.
  - d) The value of *InPos* or *Size* exceeds the number of bits in *In*.
  - e) 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/<br>FUN | Graphic expression | ST expression              |
|-------------|-------------|------------|--------------------|----------------------------|
| MemCopy     | Memory Copy | FUN        |                    | MemCopy(In, AryOut, Size); |

## Variables

|                  | Meaning                | I/O    | Description                      | Valid range           | Unit | Default |
|------------------|------------------------|--------|----------------------------------|-----------------------|------|---------|
| In[] (array)     | Move source array      | Input  | Move source array                | Depends on data type. | ---  | *1      |
| 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             | ---  | ---     |

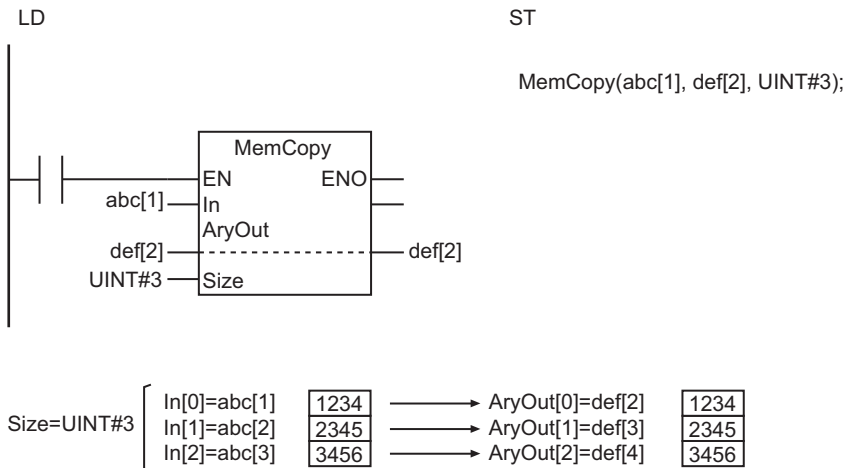
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean                                       | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     |
| 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].

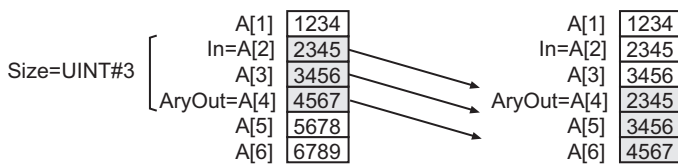
The following shows an example where *Size* is UINT#3.



### 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 instruction, *AryMove* on page 2-389, if the source and destination have different data types.
- If the source and destination have the same data type, this instruction is faster than the *AryMove* instruction.
- Use the instruction, *MOVE* on page 2-372, 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 change to TRUE and AryOut[] will not change.
- Return value Out is not used when this instruction is used in ST.
- An error will occur 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/<br>FUN | Graphic expression | ST expression               |
|-------------|-----------|------------|--------------------|-----------------------------|
| SetBlock    | Block Set | FUN        |                    | SetBlock(In, AryOut, Size); |

## Variables

|                  | Meaning                | I/O    | Description                      | Valid range           | Unit | Default |
|------------------|------------------------|--------|----------------------------------|-----------------------|------|---------|
| In               | Move source            | Input  | Move source                      | Depends on data type. | ---  | *1      |
| 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             | ---  | ---     |

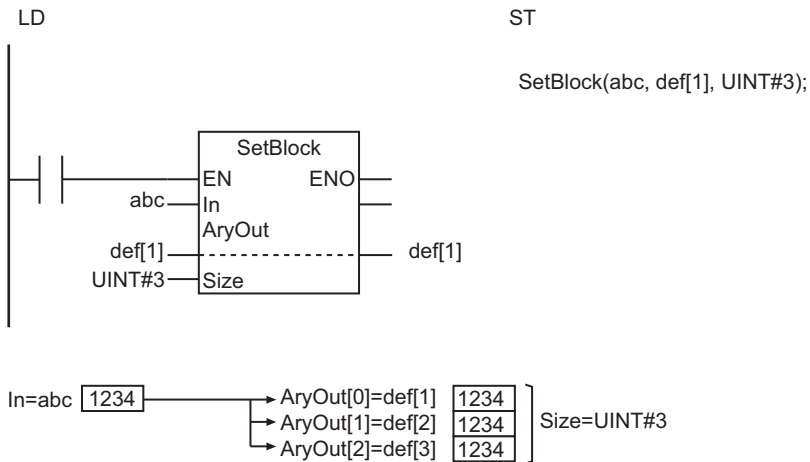
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     | 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 <i>In</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |    |
| Out              | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |    |

## Function

The SetBlock instruction moves the value of move source *In* to *Size* elements in move destination array *AryOut*[] starting from *AryOut*[0].

The following shows an example where *Size* is *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 this instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *AryOut*[] will not change.
  - a) The value of *Size* exceeds the array area of *AryOut*[].



# Exchange

The Exchange instruction exchanges the values of two variables.

| Instruction | Name          | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|---------------|------------|--------------------|---------------------------|
| Exchange    | Data Exchange | FUN        |                    | Exchange(InOut1, InOut2); |

## Variables

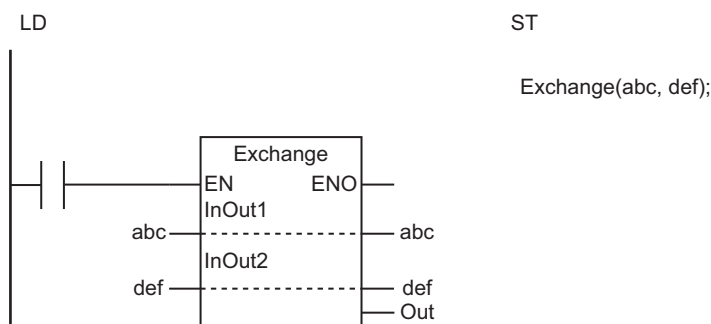
|                   | Meaning          | I/O    | Description      | Valid range           | Unit | Default |
|-------------------|------------------|--------|------------------|-----------------------|------|---------|
| InOut1 and InOut2 | Data to exchange | In-out | Data to exchange | Depends on data 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 the same data type as <i>InOut1</i> .                         |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |  |
| Out    | OK  |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |  |

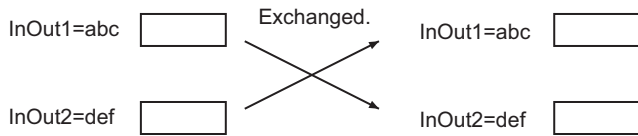
## Function

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 of variables *abc* and *def* are exchanged.



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 this instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut1* and *InOut2* will not change.
  - a) Both *InOut1* and *InOut2* are STRING data, and the string length of one of them cannot accommodate the other.

# AryExchange

The AryExchange instruction exchanges the elements of two arrays.

| Instruction | Name                | FB/<br>FUN | Graphic expression | ST expression                      |
|-------------|---------------------|------------|--------------------|------------------------------------|
| AryExchange | Array Data Exchange | FUN        |                    | AryExchange(InOut1, InOut2, Size); |

## Variables

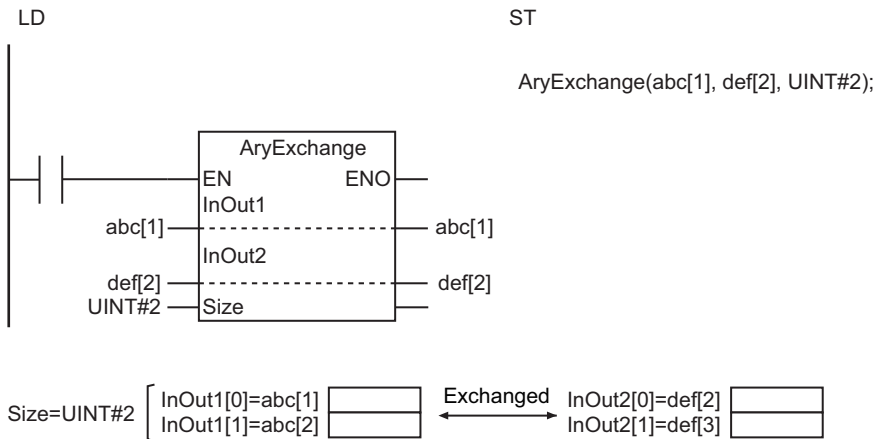
|                                | Meaning            | I/O    | Description                    | Valid range           | Unit | Default |
|--------------------------------|--------------------|--------|--------------------------------|-----------------------|------|---------|
| Size                           | 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             | ---  | ---     |

|                  | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     |
| 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 shows an example where *Size* is UINT#2.



## Additional Information

- Use the instruction, *MOVE* on page 2-372, to assign constants to variables.
- Use the instruction, *MemCopy* on page 2-381, 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, *Out* will be TRUE, and InOut1[] and InOut2[] will not change.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and InOut1[] and InOut2[] will not change.
  - a) The value of *Size* exceeds the array range of InOut1[] or InOut2[].
  - b) InOut1[] and InOut2[] are STRING arrays, and the string length of an element in one array exceeds that of the corresponding element in the other array.
  - c) InOut1[] and InOut2[] are STRING arrays, and an element does not end with 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/<br>FUN | Graphic expression | ST expression              |
|-------------|------------|------------|--------------------|----------------------------|
| AryMove     | Array Move | FUN        |                    | AryMove(In, AryOut, Size); |

## Variables

|                  | Meaning            | I/O    | Description                | Valid range           | Unit | Default |
|------------------|--------------------|--------|----------------------------|-----------------------|------|---------|
| In[] (array)     | Move source array  | Input  | Array to move              | Depends on data type. | ---  | *1      |
| 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             | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

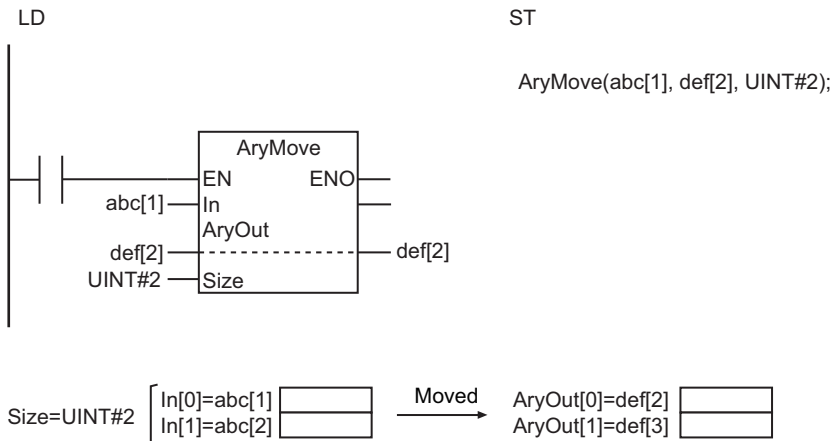
|                  | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------------------|---|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                  | 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].

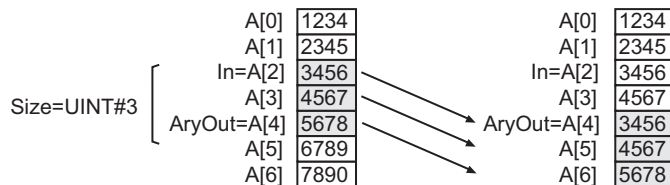
The data types of In[] and AryOut[] can be different.

The following shows an example where *Size* is 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 this instruction is used in ST.
- An error will occur in the following cases. ENO will be FALSE, and AryOut[] will not change.
  - The value of Size exceeds the size of In[] or AryOut[].
  - In[] and AryOut[] are STRING arrays, and the string length of any In[] element to move exceeds the size of the corresponding element in AryOut[].

# Clear

The Clear instruction initializes a variable.

| Instruction | Name       | FB/<br>FUN | Graphic expression | ST expression |
|-------------|------------|------------|--------------------|---------------|
| Clear       | Initialize | FUN        |                    | Clear(InOut); |

## Variables

|       | 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             | ---  | ---     |

|       | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 *InOut* (Data to initialize).

If an initial value attribute is specified for the variable, the initial value is applied. If an initial value attribute is not specified, the default initial value for the data type of *InOut* is applied.

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 corresponding global variable.

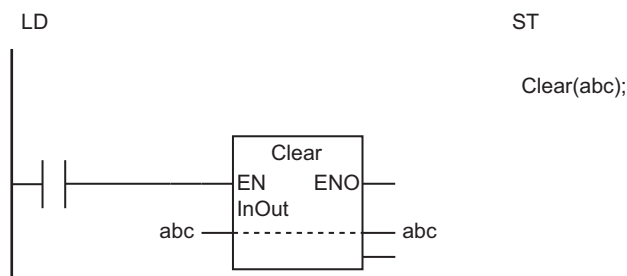
The default values for the data types are given below.

| 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   | "                     |

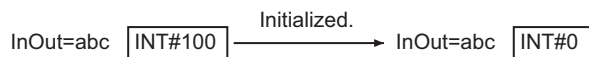
If *InOut* is an array, array element, structure, or structure member, the following processing is performed.

| 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. For example, if the value of variable *abc* is INT#100, it is initialized to INT#0.



The Clear instruction initializes the value of *InOut*.  
The data type of *abc* is INT, so the value of *abc* will be INT#0.



### Additional Information

- If *InOut* is an array that is used as a stack, execute this instruction, and also assign 0 to the variable that manages the number of items stored in the stack.
- 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.

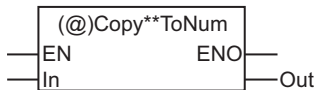
### Precautions for Correct Use

- Return value *Out* is not used when this 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.
  - a) Pass one element of a BOOL array as an in-out variable to a function or function block.
  - b) Execute the Clear instruction in the function or function block.
  - c) 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/<br>FUN | Graphic expression  | ST expression  |
|-------------|--|------------|---|--|
| Copy**ToNum | Bit Pattern<br>Copy (Bit String to Signed Integer) Group | FUN        |  <p>*** must be a bit string data type.</p> | Out:=Copy**ToNum(In);<br>*** must be a bit string data type. |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>In</i> . |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

## Function

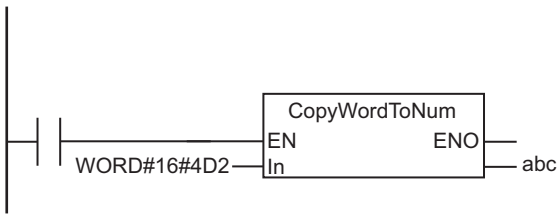
The Copy\*\*ToNum instruction group copies the content of copy source *In* directly to copy destination *Out*.

Four instructions are provided for the following data type combinations of *In* and *Out*.

| In    | Out  | Instruction    |
|-------|------|----------------|
| BYTE  | SINT | CopyByteToNum  |
| WORD  | INT  | CopyWordToNum  |
| DWORD | DINT | CopyDwordToNum |
| LWORD | LINT | CopyLwordToNum |

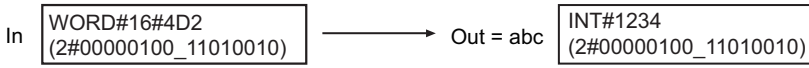
The following shows an example where *In* is WORD#16#4D2 for the CopyWordToNum instruction.

LD



ST

abc:=CopyWordToNum(WORD#16#4D2);



# 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/<br>FUN | Graphic expression | ST expression   |
|-------------|--|------------|--------------------|---|
| Copy**To*** | Bit Pattern Copy (Bit String to Real Number) Group | FUN        |                    | Out:=CopyDwordToReal(In);<br>or<br>Out:=CopyLwordToLreal(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>In</i> is DWORD, and LREAL if it is LWORD. |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

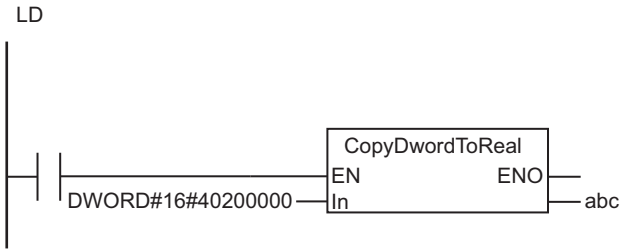
## Function

The Copy\*\*To\*\*\* instruction group copies the content of copy source *In* directly to copy destination *Out*.

Two instructions are provided for the following data type combinations of *In* and *Out*.

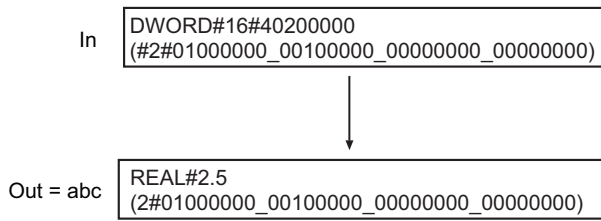
| In    | Out   | Instruction      |
|-------|-------|------------------|
| DWORD | REAL  | CopyDwordToReal  |
| LWORD | LREAL | CopyLwordToLreal |

The following shows an example where *In* is DWORD#16#40200000 for the CopyDwordToReal instruction.



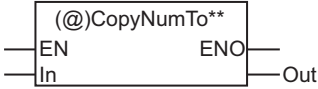
ST

```
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/<br>FUN | Graphic expression   | ST expression   |
|-------------|--|------------|--|---|
| CopyNumTo** | Bit Pattern<br>Copy (Signed Integer to Bit String) Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=CopyNumTo**(In);<br>**** must be a bit string data type. |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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. |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

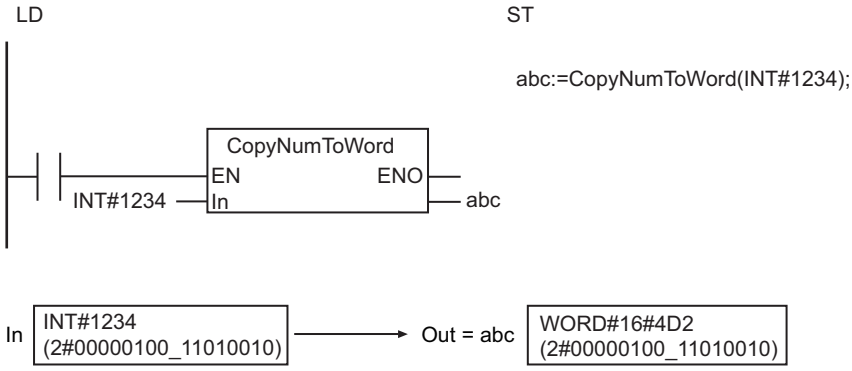
## Function

The CopyNumTo\*\* instruction group copies the content of copy source *In* directly to copy destination *Out*.

Four instructions are provided for the following data type combinations of *In* and *Out*.

| In   | Out   | Instruction    |
|------|-------|----------------|
| SINT | BYTE  | CopyNumToByte  |
| INT  | WORD  | CopyNumToWord  |
| DINT | DWORD | CopyNumToDword |
| LINT | LWORD | CopyNumToLword |

The following shows an example where *In* is INT#1234 for the CopyNumToWord instruction.



# CopyNumTo\*\* (Signed Integer to Real Number)

The CopyNumTo\*\* instruction copies the content of a signed integer directly to a real number.

| Instruction | Name  | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|---|------------|--------------------|---|
| CopyNumTo** | Bit Pattern<br>Copy (Signed<br>Integer to Real<br>Number) Group | FUN        |                    | Out:=CopyNumToReal(In);<br>or<br>Out:=CopyNumToLreal(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>In</i> is DINT, and LREAL if it is LINT. |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

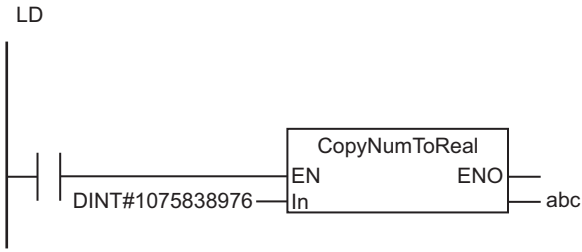
## Function

The CopyNumTo\*\* instruction group copies the content of copy source *In* directly to copy destination *Out*.

Two instructions are provided for the following data type combinations of *In* and *Out*.

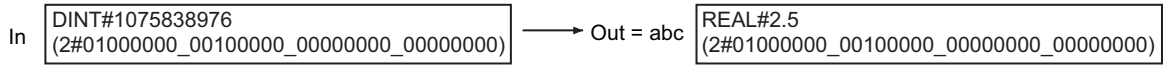
| In   | Out   | Instruction    |
|------|-------|----------------|
| DINT | REAL  | CopyNumToReal  |
| LINT | LREAL | CopyNumToLreal |

The following shows an example where *In* is DINT#1075838976 for the CopyNumToReal instruction.



ST

abc:=CopyNumToReal(DINT#1075838976);





# 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/<br>FUN | Graphic expression | ST expression   |
|-------------|--|------------|--------------------|---|
| Copy**To*** | Bit Pattern Copy (Real Number to Bit String) Group | FUN        |                    | Out:=CopyRealToDword(In);<br>or<br>Out:=CopyLrealToLword(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>In</i> is REAL, and LWORD if it is LREAL. |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

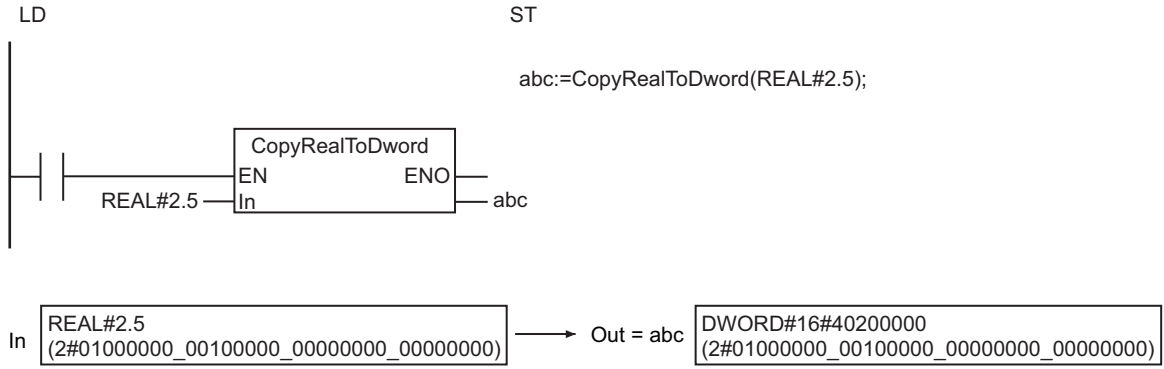
## Function

The Copy\*\*To\*\*\* instruction group copies the content of copy source *In* directly to copy destination *Out*.

Two instructions are provided for the following data type combinations of *In* and *Out*.

| In    | Out   | Instruction      |
|-------|-------|------------------|
| REAL  | DWORD | CopyRealToDword  |
| LREAL | LWORD | CopyLrealToLword |

The following shows an example where *In* is REAL#2.5 for the CopyRealToDword instruction.



# 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/<br>FUN | Graphic expression | ST expression   |
|-------------|--|------------|--------------------|---|
| Copy**ToNum | Bit Pattern Copy (Real Number to Signed Integer) Group | FUN        |                    | Out:=CopyRealToNum(In);<br>or<br>Out:=CopyLrealToNum(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>In</i> is REAL, and LINT if it is LREAL. |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

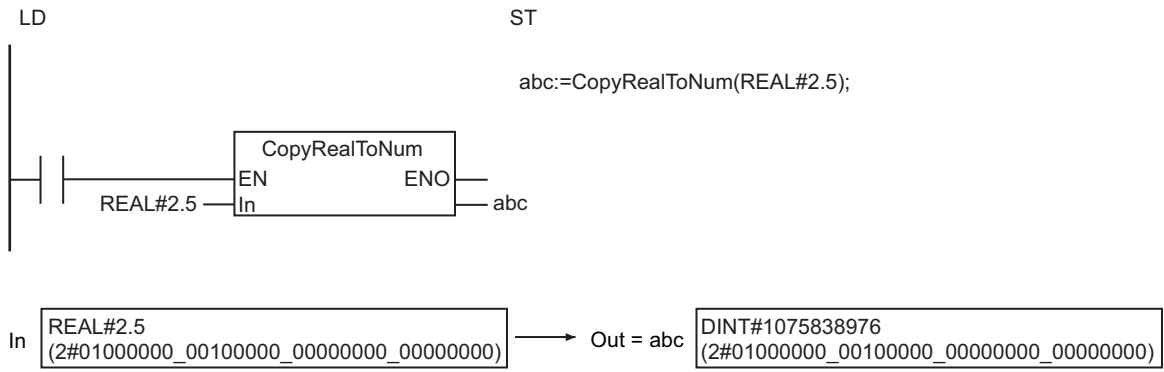
## Function

The Copy\*\*ToNum instruction group copies the content of copy source *In* directly to copy destination *Out*.

Two instructions are provided for the following data type combinations of *In* and *Out*.

| In    | Out  | Instruction    |
|-------|------|----------------|
| REAL  | DINT | CopyRealToNum  |
| LREAL | LINT | CopyLrealToNum |

The following shows an example where *In* is REAL#2.5 for the CopyRealToNum instruction.



# Shift Instructions

| Instruction       | Name   | Page       |
|-------------------|--|------------|
| AryShiftReg       | Shift Register   | page 2-406 |
| AryShiftRegLR     | Reversible Shift Register                                  | page 2-408 |
| ArySHL and ArySHR | Array N-element Left Shift/Array N-element Right Shift     | page 2-411 |
| SHL and SHR       | N-bit Left Shift/N-bit Right Shift                         | page 2-414 |
| NSHLC and NSHRC   | Shift N-bits Left with Carry/Shift N-bits Right with Carry | page 2-417 |
| ROL and ROR       | Rotate N-bits Left/Rotate N-bits Right                     | page 2-420 |

# AryShiftReg

The AryShiftReg instruction shifts an array of bit strings by one bit to the left and inserts an input value to the least-significant bit.

| Instruction | Name           | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|----------------|------------|--------------------|--|
| AryShiftReg | Shift Register | FB         |                    | AryShiftReg_instance(Shift, Reset, In, InOut, Size); |

## Variables

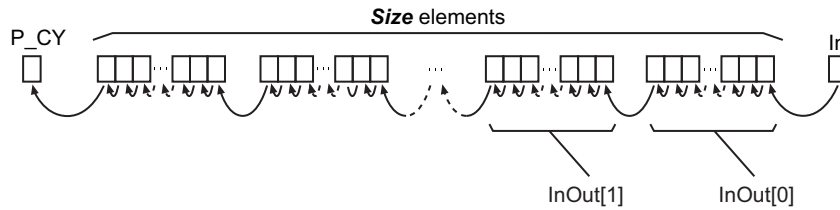
|                 | Meaning                                    | I/O    | Description   | Valid range           | Unit | Default |
|-----------------|--|--------|---|-----------------------|------|---------|
| Shift           | Shift                                      | Input  | Shifted when signal changes to TRUE.                      | Depends on data type. | ---  | FALSE   |
| Reset           | Reset                                      |        | TRUE: Register is reset.                                  |                       |      |         |
| In              | Input value                                |        | Value to insert to least-significant bit of InOut[].      |                       |      |         |
| Size            | Number of elements in array of bit strings |        | Number of elements to use as a shift register in InOut[]. |                       |      |         |
| InOut[] (array) | Array of bit strings                       | In-out | Array of bit strings                                      | Depends on data type. | ---  | ---     |

|                 | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num<br>bers |      | Times, durations,<br>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

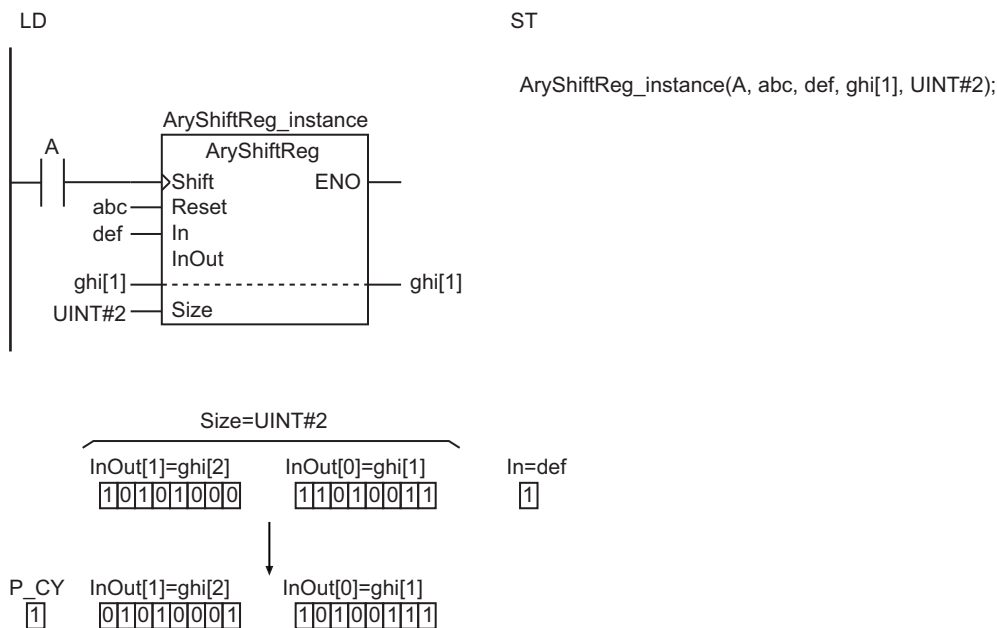
The AryShiftReg instruction shifts *Size* elements from InOut[0] of the array of bit strings InOut[] by one bit to the left (i.e., toward the most-significant bit) when *Shift* changes to TRUE.

Input value *In* is inserted to the least-significant bit. The most-significant bit, which is shifted out 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 shows an example where *InOut[]* is a BYTE array and *Size* is *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 normally performed, or when *Reset* is TRUE and the reset operation is normally performed.
- The *InOut[]* does not change if the value of *Size* is 0.
- An error will occur in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - a) The value of *Size* exceeds the array area of *InOut[]*.

# AryShiftRegLR

The AryShiftRegLR instruction shifts an array of bit strings by one bit to the left or right and inserts an input value to the least-significant or most-significant bit.

| Instruction   | Name                      | FB/<br>FUN | Graphic expression | ST expression   |
|---------------|---------------------------|------------|--------------------|---|
| AryShiftRegLR | Reversible Shift Register | FB         |                    | AryShiftRegLR_instance(ShiftL, ShiftR, Reset, In, InOut, Size); |

## Variables

|                 | 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[].               |                       |      |         |
| InOut[] (array) | Array of bit strings                       | In-out | Array of bit strings  | Depends on data type. | ---  | ---     |

|                 | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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    |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

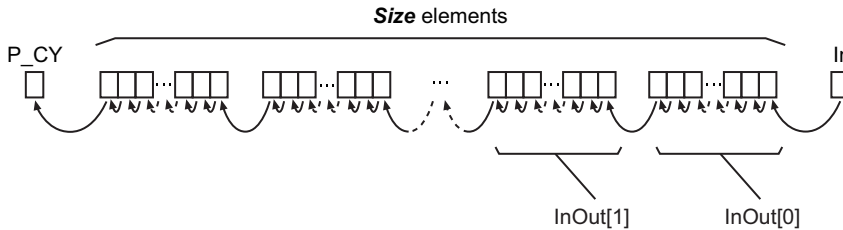


**Function**

The AryShiftRegLR instruction shifts *Size* elements from InOut[0] of the array of bit strings InOut[] by one bit to the left when *ShiftL* changes to TRUE.

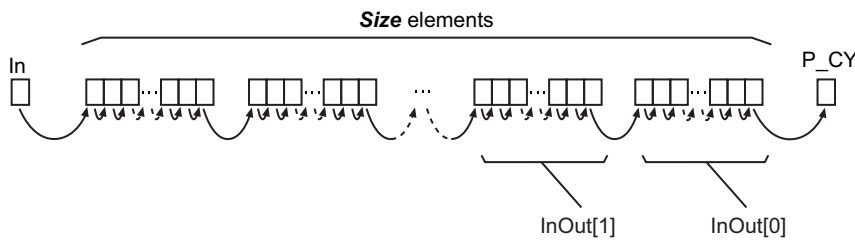
Input value *In* is inserted to the least-significant bit.

The most-significant bit, which is shifted out of the array of bit strings, is output to the Carry (CY) Flag (*P\_CY*).



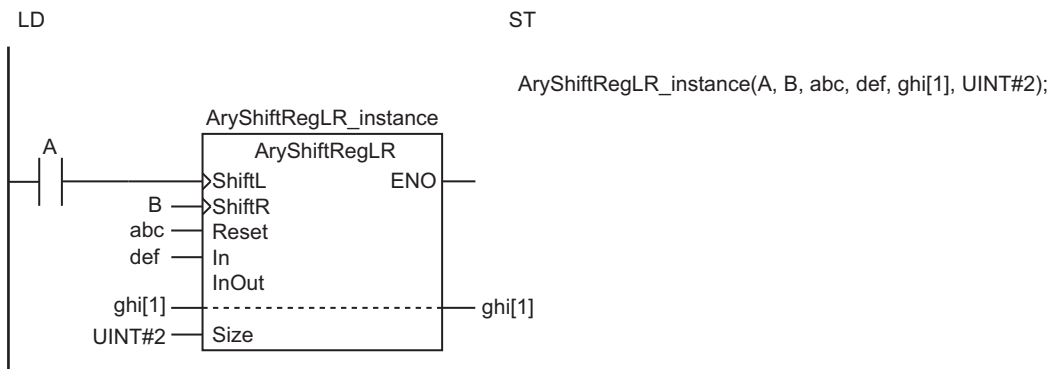
When *ShiftR* changes to TRUE, the bits are shifted by one bit to the right, and *In* is inserted to the most-significant bit.

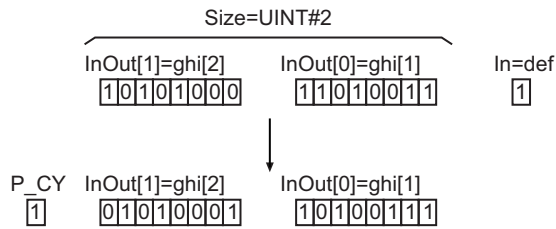
The least-significant bit, which is shifted out of the array of bit strings, is output to the Carry (CY) Flag (*P\_CY*).



When *Reset* is TRUE, *P\_CY* and all of the bits in *Size* elements starting from InOut[0] are set to FALSE.

The following shows an example where *InOut* is a BYTE array, *Size* is UINT#2 and *ShiftL* changes to TRUE.





## 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 normally performed, or when *Reset* is TRUE and the reset operation is normally performed.
- The InOut[] does not change if the value of *Size* is 0.
- An error will occur in the following case. *ENO* will be FALSE, and InOut[] will not change.
  - a) 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/<br>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

|                 | 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             | ---  | ---     |

|                 | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real num<br>bers |      | Times, durations,<br>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     |  |
| Out             | OK          |             |      |       |       |       |          |       |       |      |     |      |      |      |                  |      |  |     |    |        |  |

Arrays of structures can also be specified.

## 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, all members in the structures are initialized.

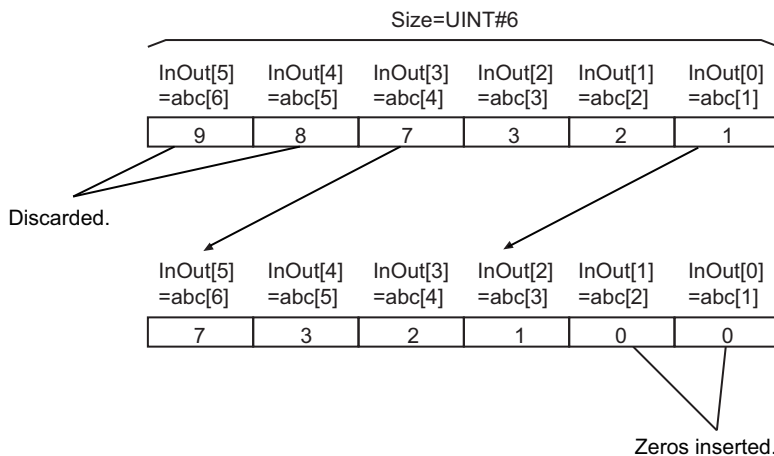
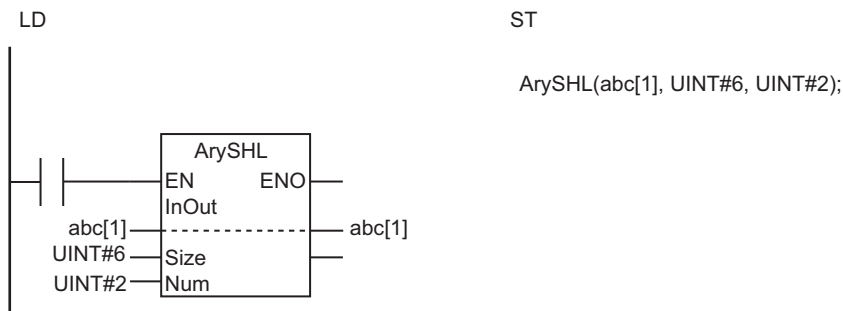
The default values for the data types are given below.

| Data type  | 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).

The following example shows the ArySHL instruction when *Size* is UINT#6 and *Num* is UINT#2.



## ArySHR

The ArySHR instruction shifts the array to the right (toward the lower elements of the array).

## Additional Information

If `InOut[]` is BOOL data, the result 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 these instructions are used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and `InOut[]` will not change.
  - a) 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/<br>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

|        | 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 the number of bits in <i>In</i> | 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 or higher, 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);.

|     | Boo<br>lean                             | Bit strings |      |       |       | Integers |      |       |          |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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<br>*1 |      |       | OK<br>*1 |      |     |      |                     |      |  |      |      |     |    |        |
| Out | Must be the same data type as <i>In</i> |             |      |       |       |          |      |       |          |      |     |      |                     |      |  |      |      |     |    |        |

\*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.

## Function

These instructions shift bit string data, *In* (Data to shift), by the number of bits specified in *Num* (Number to shift).



- 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, with the Carry (CY) Flag available.

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/<br>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

|                 | 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             | ---  | ---     |

|                 | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real num<br>bers |      | Times, durations,<br>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

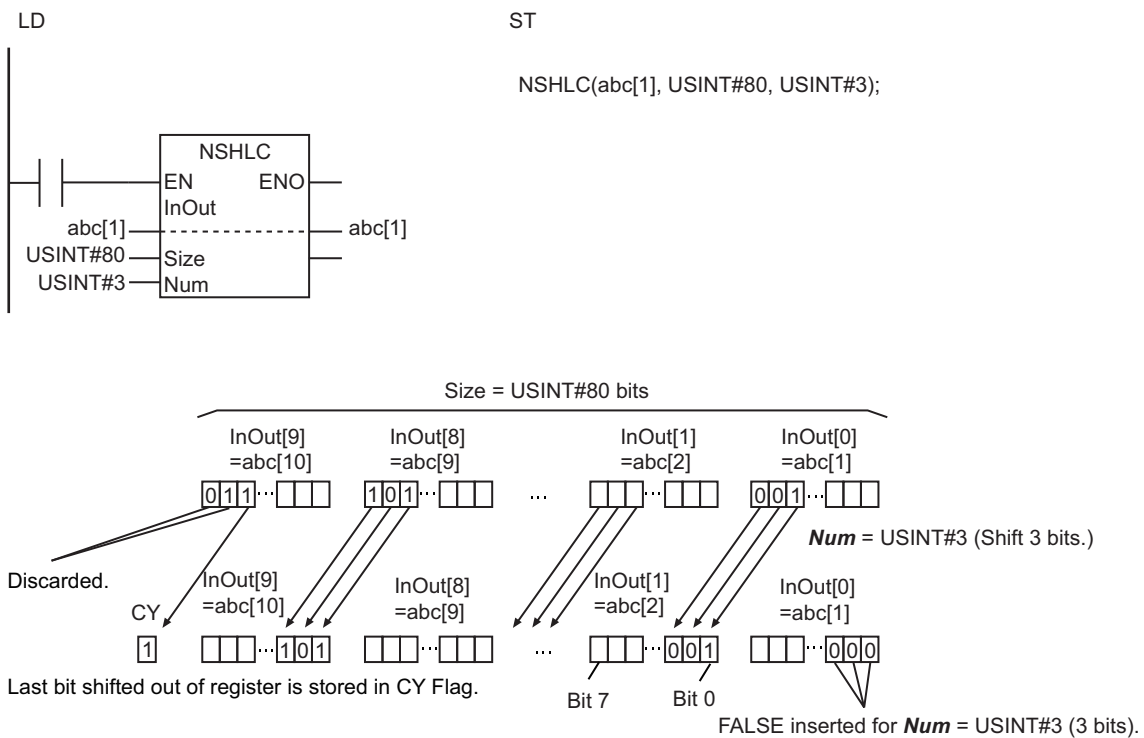
These instructions shift *Size* array elements in InOut[] (Shift register array) by the number of bits specified in *Num*. The shift register starts at InOut[0].

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.

The following example shows the NSHLC instruction when `InOut[]` is a BYTE array, `Size` is `USINT#80` and `Num` is `USINT#3`.



## 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.

### 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 these instructions are used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - a) 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/<br>FUN | Graphic expression | ST expression      |
|-------------|------------------------|------------|--------------------|--------------------|
| ROL         | Rotate N-bits<br>Left  | FUN        |                    | Out:=ROL(In, Num); |
| ROR         | Rotate N-bits<br>Right | FUN        |                    | Out:=ROR(In, Num); |

## Variables

|       | 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    |        | Number of bits to rotate | 0 to the number of bits in <i>In</i> | 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 or higher, 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);

|     | Boo<br>lean                             | Bit strings |      |       |       | Integers |      |       |          |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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<br>*1 |      |       | OK<br>*1 |      |     |      |                      |      |  |      |      |     |    |        |
| Out | Must be the same data type as <i>In</i> |             |      |       |       |          |      |       |          |      |     |      |                      |      |  |      |      |     |    |        |

\*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.

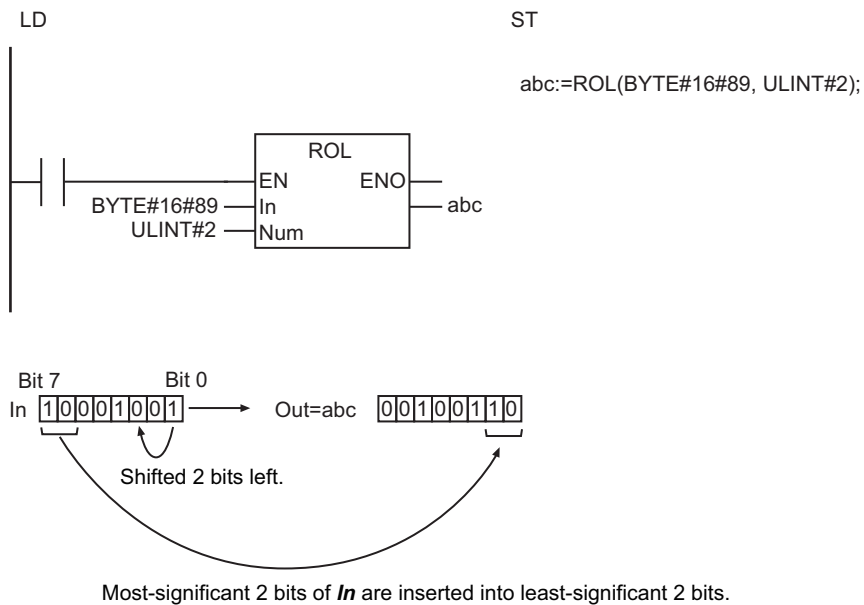
## Function

These instructions rotate bit string data, *In* (Data to rotate), by the number of bits specified in *Num* (Number of bits). 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).

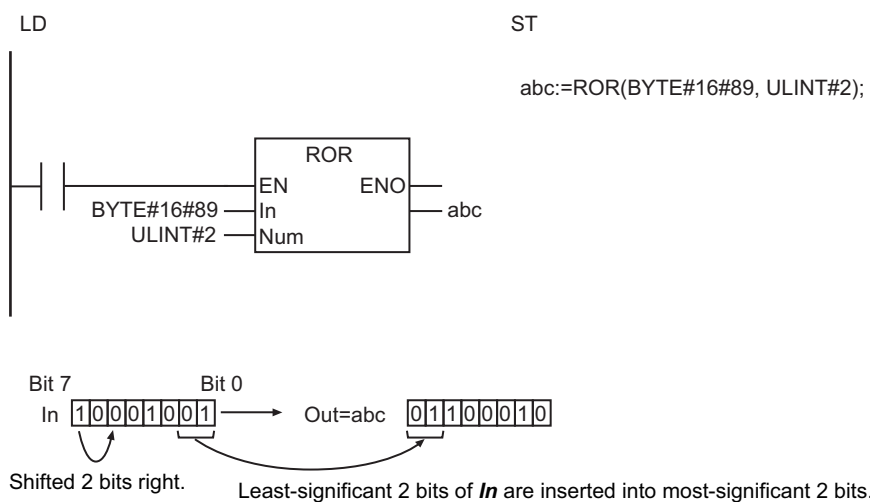
The following shows an example where *In* is BYTE#16#89 and *Num* is ULINT#2.



## ROR

The ROR instruction rotates bits from left to right (from most-significant to least-significant bits).

The following shows an example where *In* is BYTE#16#89 and *Num* is ULINT#2.



## 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.

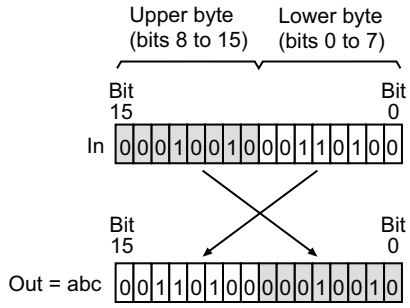
# Conversion Instructions

| Instruction                       | Name  | Page       |
|-----------------------------------|---|------------|
| Swap                              | Swap Bytes  | page 2-425 |
| Neg                               | Reverse Sign  | page 2-427 |
| Decoder                           | Bit Decoder   | page 2-429 |
| Encoder                           | Bit Encoder   | page 2-432 |
| BitCnt                            | Bit Counter   | page 2-434 |
| ColmToLine_**                     | Column to Line Conversion Group   | page 2-435 |
| LineToColm                        | Line to Column Conversion   | page 2-437 |
| Gray                              | Gray Code Conversion  | page 2-439 |
| UTF8ToSJIS                        | Character Code Conversion (UTF-8 to SJIS)   | page 2-444 |
| SJISToUTF8                        | Character Code Conversion (SJIS to UTF-8)   | page 2-446 |
| PWLApprox and PWLApproxNoLineChk  | Broken Line Approximation with Broken Line Data Check/ Broken Line Approximation without Broken Line Data Check | page 2-448 |
| PWLLineChk                        | Broken Line Data Check  | page 2-454 |
| MovingAverage                     | Moving Average  | page 2-457 |
| DispartReal                       | Separate Mantissa and Exponent  | page 2-464 |
| UniteReal                         | Combine Real Number Mantissa and Exponent   | page 2-467 |
| NumToDecString and NumToHexString | Fixed-length Decimal Text String Conversion/Fixed-length Hexadecimal Text String Conversion                     | page 2-469 |
| HexStringToNum_**                 | Hexadecimal Text String-to-Number Conversion Group  | page 2-472 |
| FixNumToString                    | Fixed-decimal Number-to-Text String Conversion  | page 2-474 |
| StringToFixNum                    | Text String-to-Fixed-decimal Conversion   | page 2-476 |
| DtToString                        | Date and Time-to-Text String Conversion   | page 2-479 |

| Instruction                   | Name   | Page       |
|-------------------------------|--|------------|
| DateToString                  | Date-to-Text String Conversion   | page 2-481 |
| TodToString                   | Time of Day-to-Text String Conversion  | page 2-483 |
| GrayToBin_** and BinToGray_** | Gray Code-to-Binary Code Conversion Group/ Binary Code-to-Gray Code Conversion | page 2-485 |
| StringToAry                   | Text String-to-Array Conversion  | page 2-488 |
| AryToString                   | Array-to-Text String Conversion  | page 2-490 |
| DispartDigit                  | Four-bit Separation  | page 2-492 |
| UniteDigit_**                 | Four-bit Join Group  | page 2-494 |
| Dispart8Bit                   | Byte Data Separation   | page 2-496 |
| Unite8Bit_**                  | Byte Data Join Group   | page 2-498 |
| ToAryByte                     | Conversion to Byte Array   | page 2-500 |
| AryByteTo                     | Conversion from Byte Array   | page 2-506 |
| SizeOfAry                     | Get Number of Array Elements   | page 2-512 |
| PackWord                      | 2-byte Join  | page 2-514 |
| PackDword                     | 4-byte Join  | page 2-516 |
| LOWER_BOUND and UPPER_BOUND   | Get First Number of Array/ Get Last Number of Array                            | page 2-518 |

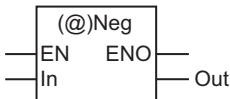






# Neg

The Neg instruction reverses the sign of a number.

| Instruction | Name         | FB/<br>FUN | Graphic expression  | ST expression |
|-------------|--------------|------------|---|---------------|
| Neg         | Reverse Sign | FUN        |  | Out:=Neg(In); |

## Variables

|     | Meaning           | I/O    | Description       | Valid range           | Unit | Default |
|-----|-------------------|--------|-------------------|-----------------------|------|---------|
| In  | Data to convert   | Input  | Data to convert   | Depends on data type. | ---  | *1      |
| Out | Conversion result | Output | Conversion result | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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    |  |      |     |    |        |

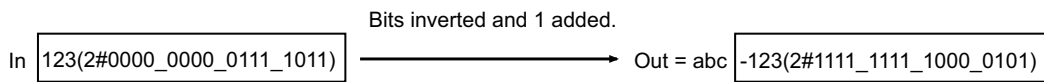
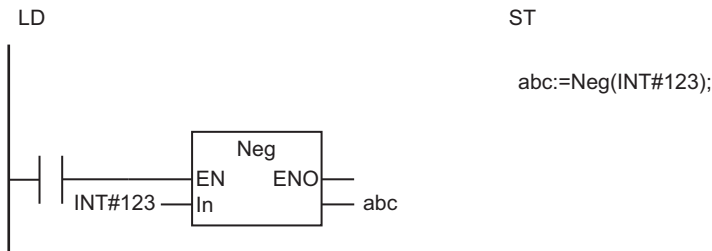
## Function

The Neg instruction reverses the sign of data to convert *In*.

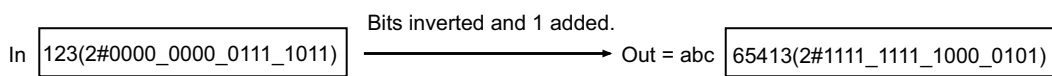
The conversion processing depends on the data type of *In*, as shown below.

| Data type of <i>In</i>                             | Value of <i>Out</i>  |
|--|--|
| Signed integer:<br>SINT, INT, DINT, or LINT        | All bits in <i>In</i> are inverted and then 1 is added.<br>(This is equal to the result of multiplying <i>In</i> by -1.) |
| Unsigned integers:<br>USINT, UNIT, UDINT, or ULINT | All bits in <i>In</i> are inverted and then 1 is added.  |
| Real numbers:<br>REAL or LREAL                     | $In \times (-1)$   |

The following shows an example where *In* is INT#123.

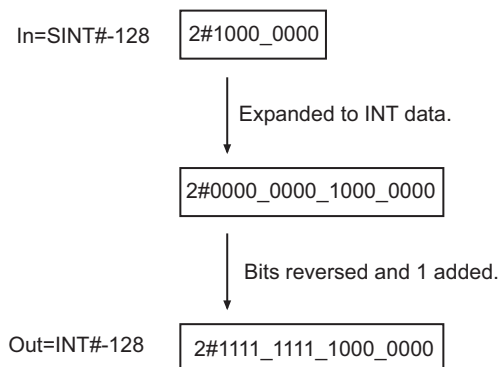


The following shows an example where *In* is UINT#123.



### Precautions for Correct Use

If you use different data types for *In* and *Out*, make sure the valid range of *Out* accommodates the valid range of *In*. Otherwise, an error will not occur, but 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/<br>FUN | Graphic expression | ST expression             |
|-------------|-------------|------------|--------------------|---------------------------|
| Decoder     | Bit Decoder | FUN        |                    | Decoder(In, Size, InOut); |

## Variables

|                 | 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             | ---  | ---     |

|                 | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

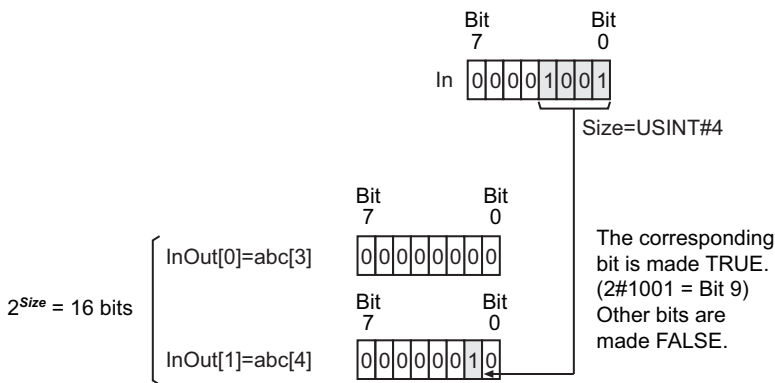
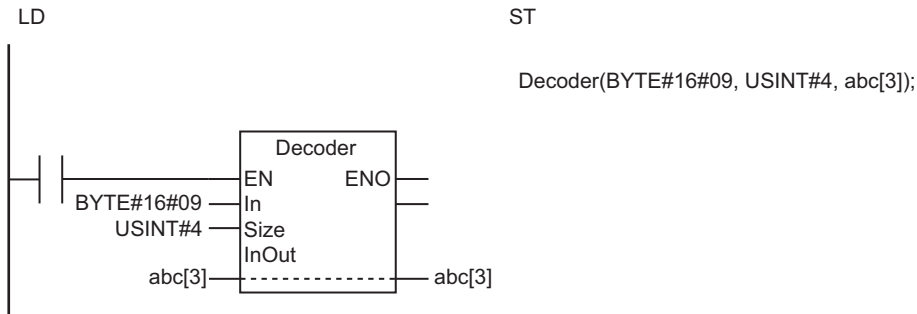
## Function

The Decoder instruction accesses  $2^{Size}$  bits in InOut[], which begins with InOut[0], and sets a specified bit to TRUE. The other bits are set 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].

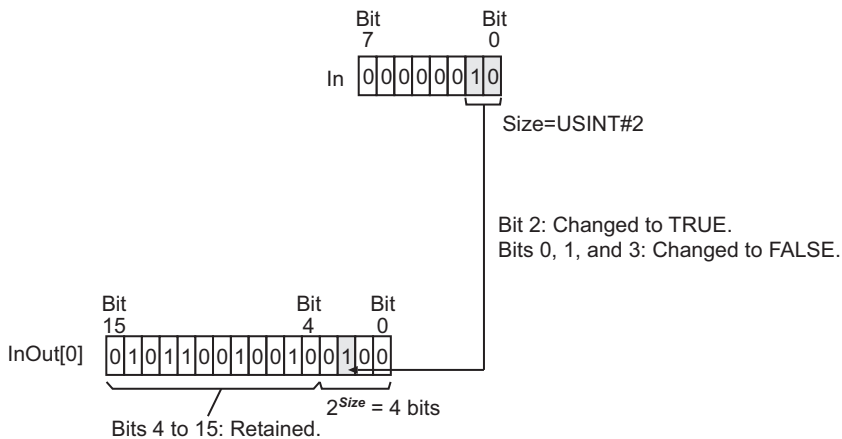
Consider an example where *In* is BYTE#16#09, *Size* is USINT#4, and InOut[] is a BYTE array. The value of *In* (conversion bit position) is 16#09, which is 9 in decimal notation. Accordingly, the ninth lowest bit in InOut[] is set to TRUE, and the other bits are set to FALSE.

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.



If the number of bits in the elements of InOut[] is larger than the number of bits specified with 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 lower 4 bits of InOut[0] are set. The values of the remaining bits in InOut[0] (bits 4 to 15) are retained.



### Additional Information

Use the instruction, *Encoder* on page 2-432, 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 the bits in `InOut[]` change to FALSE.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and `InOut[]` will not change.
  - a) The value of *Size* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression           |
|-------------|-------------|------------|--------------------|-------------------------|
| Encoder     | Bit Encoder | FUN        |                    | Out:=Encoder(In, Size); |

## Variables

|              | Meaning           | I/O    | Description               | Valid range           | Unit | Default |
|--------------|-------------------|--------|---------------------------|-----------------------|------|---------|
| In[] (array) | Array to convert  | Input  | Array to convert          | Depends on data type. | ---  | *1      |
| Size         | Bits to convert   |        | Number of bits to convert | 0 to 8                | Bits | 1       |
| Out          | Conversion result | Output | Conversion result         | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 searches for a TRUE bit in the range of  $2^{Size}$  bits of In[], which starts from In[0]. The position of the TRUE bit in this range is expressed in binary and stored in the lower Size bits of conversion result Out. The remaining bits of Out is set to FALSE.

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 where Size is USINT#4 and In[] is a BYTE array.

Size is USINT#4, so a TRUE bit is searched for in the range of  $2^4$ , or 16 bits, starting from In[0]. In the following figure, the ninth bit in the range is TRUE.







# ColmToLine\_\*\*

The ColmToLine\_\*\* instruction extracts bit values from the specified position of array elements and outputs them as a bit string.

| Instruction   | Name                            | FB/<br>FUN | Graphic expression                          | ST expression  |
|---------------|---------------------------------|------------|---|--|
| ColmToLine_** | Column to Line Conversion Group | FUN        | <p>**** must be a bit string data type.</p> | Out:=ColmToLine_**(In, Size, Pos);<br>**** must be a bit string data type. |

## Variables

|              | Meaning                       | I/O    | Description                           | Valid range                    | Unit | Default |
|--------------|-------------------------------|--------|---------------------------------------|--------------------------------|------|---------|
| In[] (array) | Array to convert              | Input  | Array to convert                      | Depends on data type.          | ---  | *1      |
| Size         | Number of elements to convert |        | Number of elements in In[] to convert | 0 to the number 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.          | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|--------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|---------------------|------|--|------|------|-----|----|--------|
|              | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT                | REAL | REAL   | 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 of In[] (array to convert) are extracted, starting from In[0].

Next, the value of the *Pos*-th bit of each element is extracted.

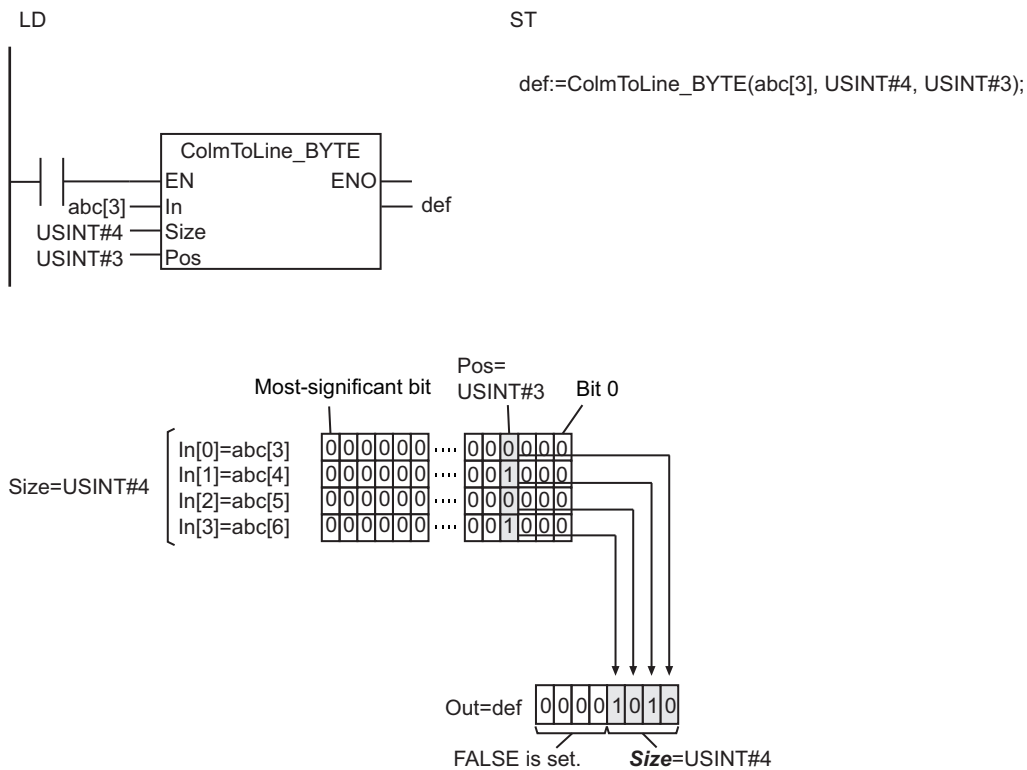
The extracted values are converted into a bit string of *Size* bits and stored in the lower bits of *Out* (conversion result).

The remaining bits of *Out* are set to FALSE.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is BYTE data, the name of 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*.



### Additional Information

Use the instruction, *LineToColm* on page 2-437, 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 the bits in *Out* change to FALSE.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* is outside the valid range.
  - b) The value of *Pos* is outside the valid range.
  - c) 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/<br>FUN | Graphic expression | ST expression                     |
|-------------|---------------------------|------------|--------------------|-----------------------------------|
| LineToColm  | Line to Column Conversion | FUN        |                    | LineToColm(In, InOut, Size, Pos); |

## Variables

|                 | Meaning                      | I/O    | Description                            | Valid range                          | Unit | Default |
|-----------------|------------------------------|--------|--|--------------------------------------|------|---------|
| In              | Data to convert              | Input  | Data to convert                        | Depends on data type.                | ---  | *1      |
| Size            | Number of elements in result |        | Number of elements in result           | 0 to the number of bits in <i>In</i> |      | 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                            | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                 | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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

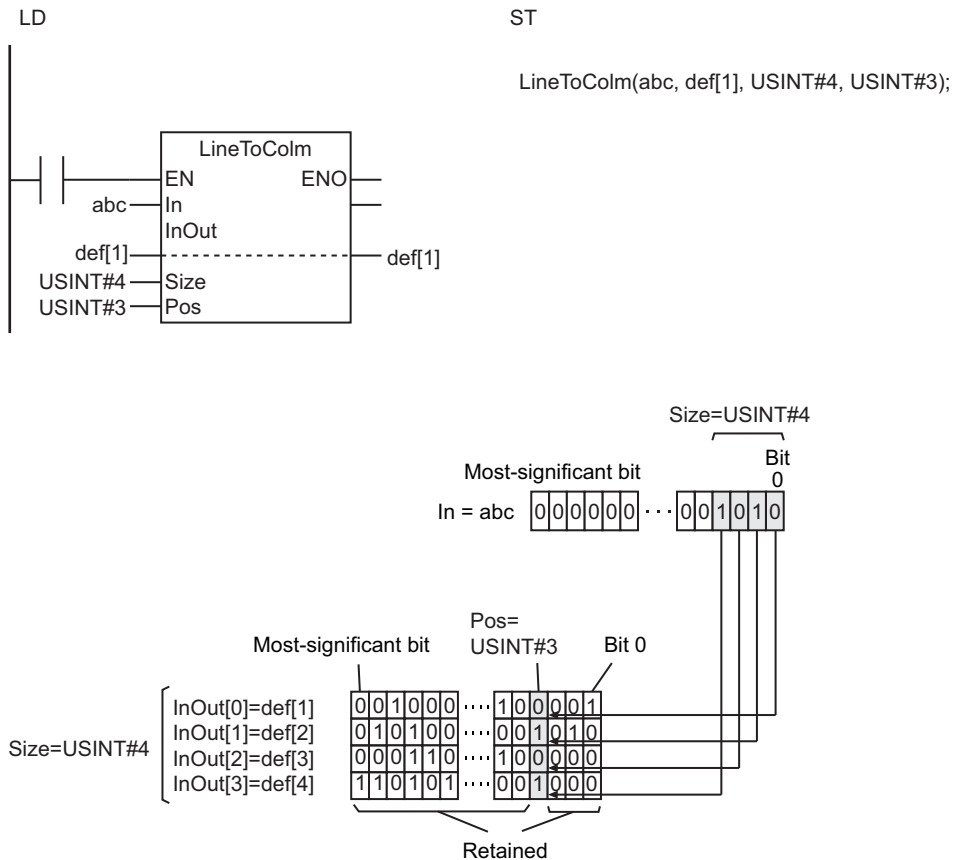
The LineToColm instruction takes the bits from a bit string and outputs them to the specified bit position in array elements.

First, the lower *Size* bits are extracted from *In* (data to convert), and handled as individual bits.

Next, each extracted bit is stored in the *Pos*-th bit of the corresponding element of *InOut*[], which begins with *InOut*[0]. The value of *Size* is equal to the number of array elements to which the extracted bits are assigned.

The values of all bits for which values are not stored are retained.

The following shows an example where *Pos* is *USINT#3* and *Size* is *USINT#4*.



### Additional Information

Use the instruction, *ColmToLine\_\*\** on page 2-435, 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*[] will not change.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *InOut*[] will not change.
  - a) The value of *Size* is outside the valid range.
  - b) The value of *Pos* is outside the valid range.
  - c) The value of *Size* exceeds the array area of *InOut*[].

# Gray

The Gray instruction converts a gray code into an angle.

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression                        |
|-------------|----------------------|------------|--------------------|--------------------------------------|
| Gray        | Gray Code Conversion | FUN        |                    | Out:=Gray(In, Resolution, ERC, ZPC); |

## Variables

|            | 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 <i>Resolution</i>                        |      | 0       |
| ZPC        | Zero point correction        |        | Zero point correction        |   |      |         |
| Out        | Conversion result            | Output | Conversion result            | *1  | °    | ---     |

\*1. 0 to 3.5999999999999999e+2

|            | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-439 for the enumerators of the enumerated type <code>_eGRY_RESOLUTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| 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 meanings of the enumerators are as follows:

| Enumerator         | Meaning |
|--------------------|---------|
| <code>_R256</code> | 256     |

| Enumerator | Meaning        |
|------------|----------------|
| _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.

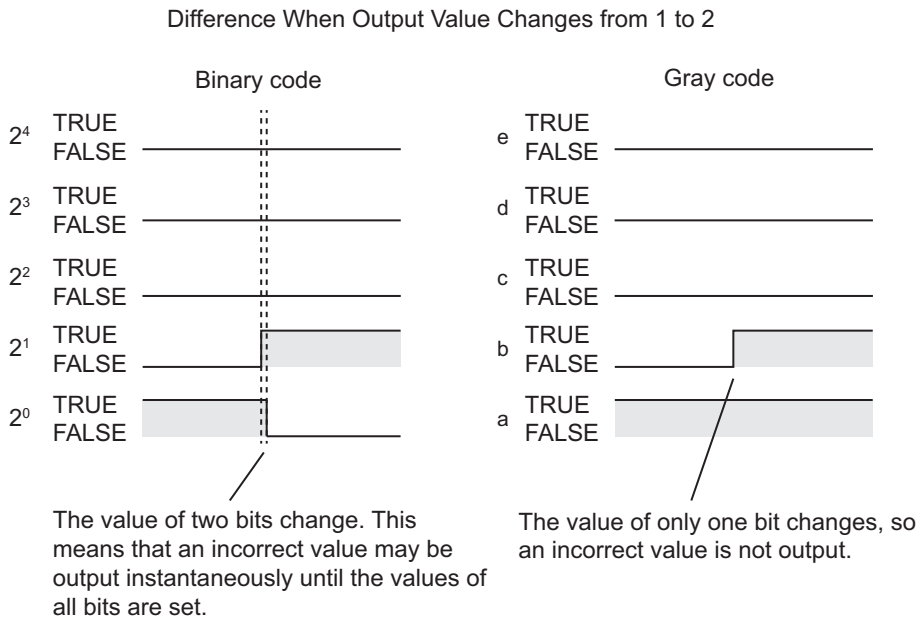
The following tables shows the 4-bit Binary code and Gray code.

| 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.



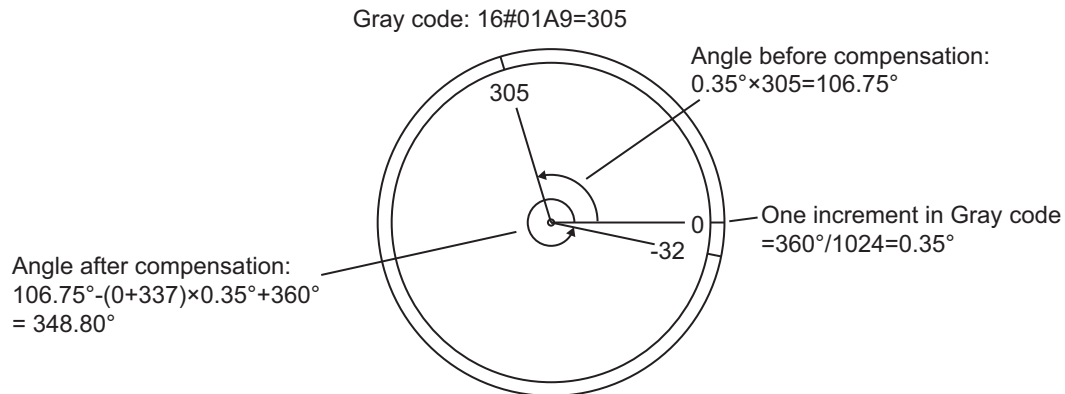
## ERC: Encoder Remainder Correction

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.



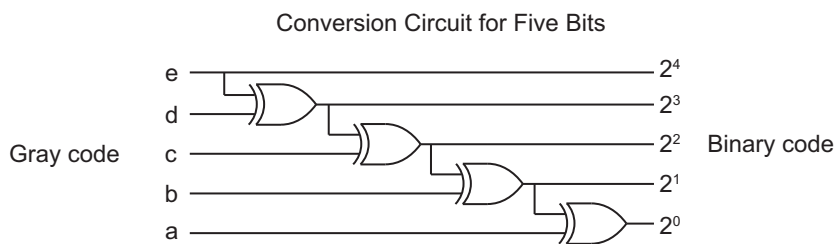


## Additional Information

Refer to the user documentation for your rotary encoder for 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.



## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *Resolution* is outside the valid range.
- The value of *ERC* exceeds the resolution that is specified with *Resolution*.
- The value of *ZPC* exceeds the resolution that is specified with *Resolution*.
- When converted to a bit string, *In* is smaller than the value of *ERC*.
- The value of the bit string corrected with *ERC* exceeds the resolution that is specified with *Resolution*.

# UTF8ToSJIS

The UTF8ToSJIS instruction converts a UTF-8 text string to a SJIS BYTE array.

| Instruction | Name  | FB/<br>FUN | Graphic expression | ST expression                  |
|-------------|---|------------|--------------------|--------------------------------|
| UTF8ToSJIS  | UTF-8 to SJIS<br>Character Code<br>Conversion | FUN        |                    | Out:=UTF8ToSJIS(In, SJISCode); |



## 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.

## Variables

|                       | Meaning                      | I/O    | Description                             | Valid range           | Unit | Default |
|-----------------------|------------------------------|--------|---|-----------------------|------|---------|
| In                    | Text string to convert       | Input  | Text string to convert                  | Depends on data type. | ---  | "       |
| SJISCode[]<br>(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             | ---  | ---     |

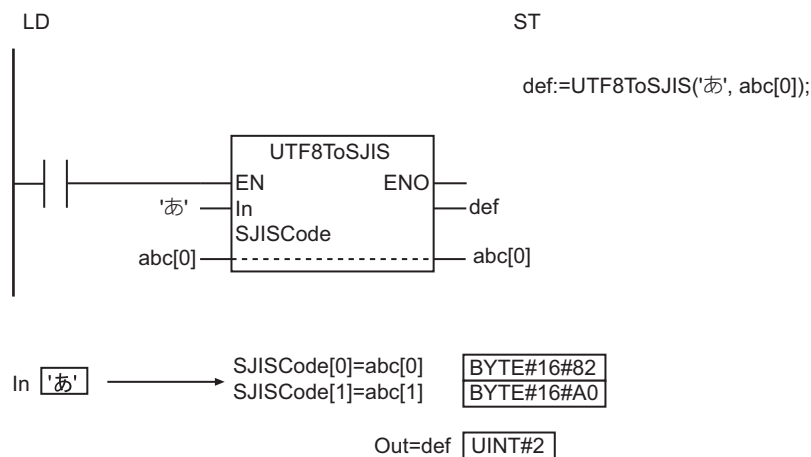
|                       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |    |
|-----------------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|-------------------|-------|--|------|-----|----|--------|----|
|                       | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT | REAL              | LREAL | TIME   | DATE | TOD | DT | STRING |    |
| In                    |             |             |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        | OK |
| SJISCode[]<br>(array) |             | OK          |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |    |
| Out                   |             |             |      |       |       |          | OK   |       |       |      |     |      |      |                   |       |  |      |     |    |        |    |

## Function

The UTF8ToSJIS instruction converts an UTF-8 text string, *In*, to a BYTE array encoded in SJIS, SJIS-Code[]. The converted data is separated into bytes, and each is stored in the corresponding element of SJISCode[] in order from SJISCode[0].

The number of SJISCode[] elements, where the converted data is stored, is assigned to *Out* (number of converted elements).

The following shows an example where *In* is 'あ'.



## Precautions for Correct Use

- NULL characters at the end of *In* are not converted. They are not counted for the number of converted elements, either.
- If the *In* text string contains only NULL characters, the value of *Out* will be 0 and SJISCode[] will not change.
- In the SJISCode[] array, subsequent elements after *Out* elements do not change. For example, if the number of converted elements is 5, SJISCode[5] and subsequent elements do not change.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* and SJISCode[] will not change.
  - a) The number of converted elements exceeds the range of the output parameter for SJISCode[].
  - b) *In* includes characters that cannot be converted.

# SJISToUTF8

The SJISToUTF8 instruction converts a SJIS BYTE array to a UTF-8 text string.

| Instruction | Name  | FB/<br>FUN | Graphic expression | ST expression              |
|-------------|---|------------|--------------------|----------------------------|
| SJISToUTF8  | SJIS to UTF-8<br>Character Code<br>Conversion | FUN        |                    | Out:=SJISToUTF8(In, Size); |



## 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.

## Variables

|              | Meaning                       | I/O    | Description                           | Valid range           | Unit | Default |
|--------------|-------------------------------|--------|---------------------------------------|-----------------------|------|---------|
| In[] (array) | SJIS array to convert         | Input  | Array encoded in SJIS 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.

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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     |

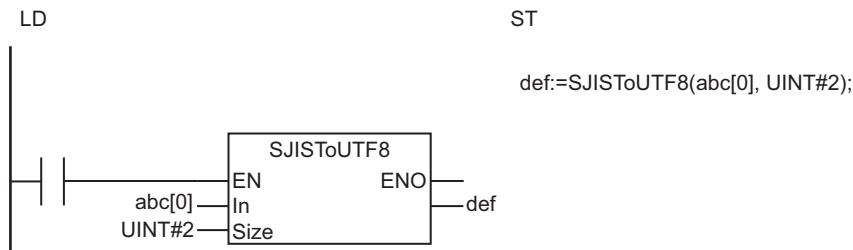
## Function

The SJISToUTF8 instruction converts the elements in a SJIS array to convert in In[] (a BYTE array) to a UTF-8 text string.

Size elements of In[], which begins with In[0], are converted. However, if a NULL character (BYTE#16#0) is included somewhere in the elements, the conversion is terminated at the point.

The converted text string is stored in Out (resulting text string). A NULL character is placed at the end of Out.

The following shows an example where  $In[0]$  is BYTE#16#82,  $In[1]$  is BYTE#16#A0, and  $Size$  is UINT#2.



## Precautions for Correct Use

- If the value of  $Size$  is 0,  $Out$  is a text string containing only NULL characters.
- An error will occur in the following cases.  $ENO$  will be FALSE, and  $Out$  will not change.
  - a) The value of  $Size$  exceeds the number of elements in  $In[]$ .
  - b) The contents of  $In[]$  includes characters that cannot be converted.

# PWLApprox and PWLApproxNoLineChk

The PWLApprox and PWLApproxNoLineChk instructions perform broken line approximations for integers or real numbers.

- PWLApprox : Checks the validity of the broken line data.  
 PWLApproxNoLineChk : Does not check the validity of the broken line data.

| Instruction         | Name   | FB/<br>FUN | Graphic expression | ST expression                           |
|---------------------|--|------------|--------------------|---|
| PWLApprox           | Broken Line Approximation with Broken Line Data Check    | FUN        |                    | Out:=PWLApprox(In, Line, Num);          |
| PWLApprox-NoLineChk | Broken Line Approximation without Broken Line Data Check | FUN        |                    | Out:=PWLApproxNoLineChk(In, Line, Num); |



## 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.

## Variables

|                | Meaning                    | I/O    | Description                | Valid range           | Unit | Default |
|----------------|----------------------------|--------|----------------------------|-----------------------|------|---------|
| In             | Data to convert            | Input  | Data to convert            | Depends on data type. | ---  | *1      |
| Line[] (array) | Broken line data array     |        | Broken line data array     |                       |      |         |
| Num            | Number of broken line data |        | Number of broken line data |                       |      |         |
| Out            | Conversion result          | Output | Conversion result          | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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   |  |      |      |     |    |        |
| Line[] (array) | Must be an array with elements that have the same data type as In. |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |
| Num            |  |             |      |       |       | OK       |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |

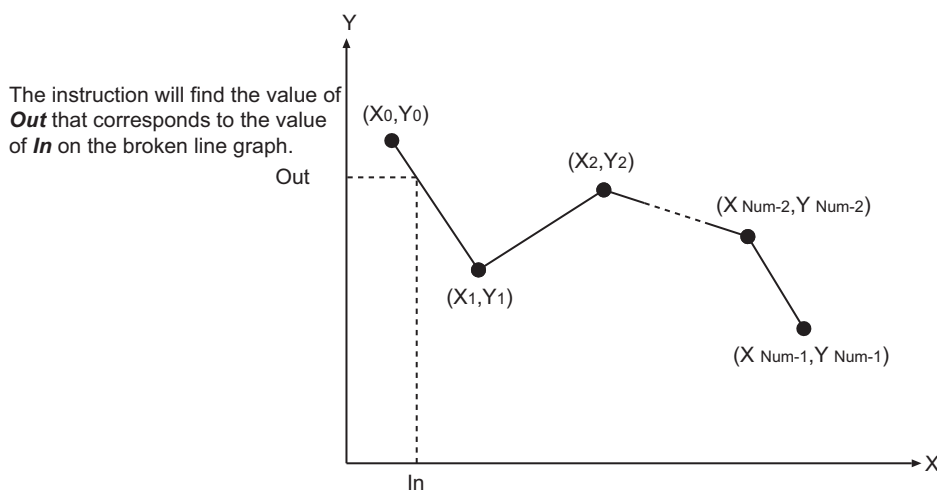


|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| Out |             |             |      |       |       | OK    | 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*.

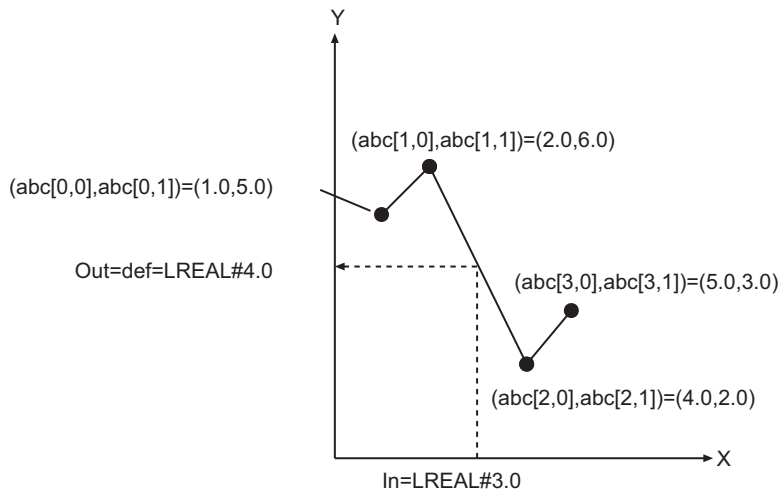


## Elements of Broken Line Data Array Line[] and Number of Broken Line Data *Num*

Line[] must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2. 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.





## Difference between the PWLApprox and PWLApproxNoLineChk Instructions

The PWLApprox and PWLApproxNoLineChk instructions are different in the following points: the validity check of *In* and *Line[]*, and processing time. The specifications of both instructions are given in the following table.

| Instruction        | Checks   | Processing when the data is not valid   | Processing time |
|--------------------|--|---|-----------------|
| PWLApprox          | <ul style="list-style-type: none"> <li>The contents of <i>Line[]</i> are checked to make sure the elements are in ascending order of the X coordinates.</li> <li>If <i>In</i> and <i>Line[]</i> are integers, <i>In</i> and the elements of <i>Line[]</i> are checked to make sure they are not nonnumeric data, positive infinity, or negative infinity.</li> </ul> | <ul style="list-style-type: none"> <li>An error occurs.</li> <li>The value of <i>ENO</i> will be FALSE.</li> <li>The value of <i>Out</i> will not change.</li> </ul>              | Long            |
| PWLApproxNoLineChk | No checks are performed.   | <ul style="list-style-type: none"> <li>An error will not occur.</li> <li>The value of <i>ENO</i> will be TRUE.</li> <li>A valid value may not be output to <i>Out</i>.</li> </ul> | Short           |

## PWLApproxNoLineChk and PWLLineChk Instructions

As the PWLApproxNoLineChk instruction does not check the validity of *In* and *Line[]*, the processing time is short. Therefore, if you are sure that the input variables are valid, it is better to use the PWLApproxNoLineChk instruction rather than the PWLApprox instruction.

*PWLLineChk* on page 2-454 checks the contents of *Line[]* to see if X coordinates are in ascending order. You can shorten the processing time by using the PWLApproxNoLineChk instruction for normal operation, and use the *PWLLineChk* instruction only when you need to check if *Line[]* data is sorted in ascending order of X coordinates.

## 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.

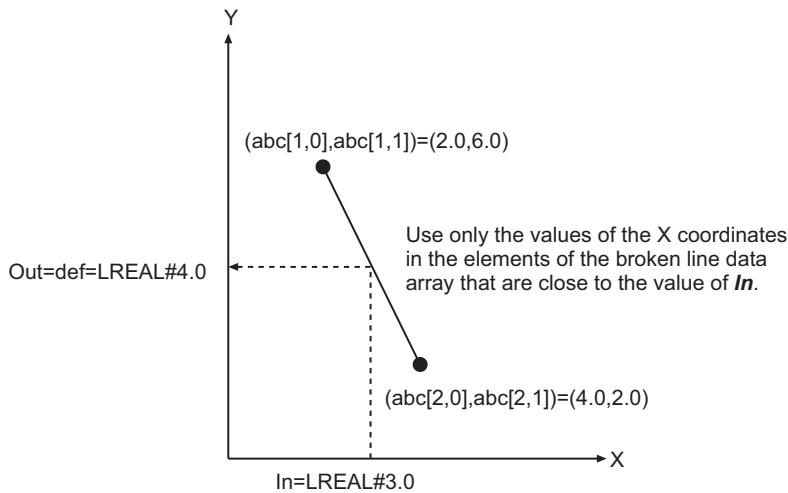
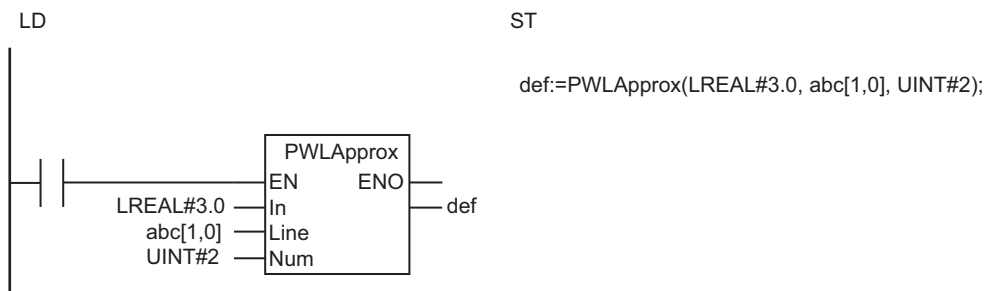
In the previous example, the processing time will be shorter for the value of *In* (LREAL#3.0) if the approximation is performed only with the following four elements, whose x-coordinate values are close to 3.0.

(abc[1,0], abc[1,1]) = (2.0, 6.0)

(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.



## 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 as below:

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. The error will not occur for the PWLApproxNoLineChk instruction in the cases, though.
  - a) The X coordinates of the broken line data are not in ascending order; the condition  $X_1 < X_2 < \dots < X_{Num}$  is not met.
  - b) *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.
  - a) The value of *Num* exceeds the array area of *Line[]*.
  - b) The value of *In* exceeds the domain of X coordinates of the broken line data that is specified with *Line[]*.

# PWLLineChk

The PWLLineChk instruction checks whether broken line data to be used for the PWLApproxNoLine-Check instruction is sorted in ascending order of X-coordinate values.

| Instruction | Name                   | FB/<br>FUN | Graphic expression | ST expression               |
|-------------|------------------------|------------|--------------------|-----------------------------|
| PWLLineChk  | Broken Line Data Check | FUN        |                    | Out:=PWLLineChk(Line, Num); |



## 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.

## Variables

|                | Meaning                    | I/O    | Description                | Valid range           | Unit | Default |
|----------------|----------------------------|--------|----------------------------|-----------------------|------|---------|
| Line[] (array) | Broken line data array     | Input  | Broken line data array     | Depends on data type. | ---  | *1      |
| Num            | Number of broken line data |        | Number of broken line data |                       |      | 1       |
| Out            | Result                     | Output | Result                     | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                | Boo lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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           |       |   |      |     |    |        |
| Num            |          |             |      |       |       |          | OK   |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Out            | OK       |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |

## Function

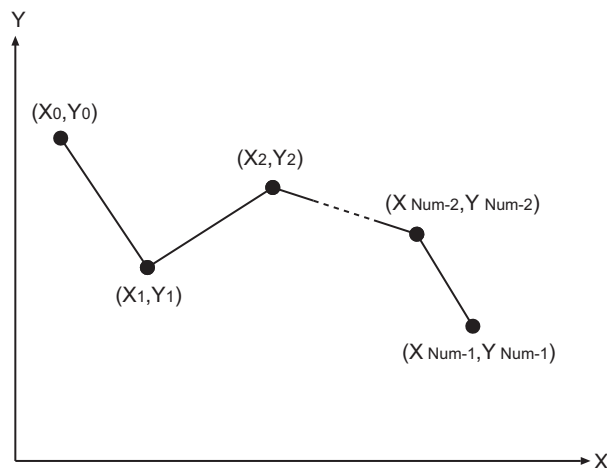
The PWLLineChk instruction is used to check whether the X coordinates in the broken line data array Line[] that is used for a Broken Line Approximation without Broken Line Data Check (PWLApproxNoLineChk) instruction are in ascending order.

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. 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[]*      Using a Three-dimensional Array for *Line[]*

|               |                    |
|---------------|--------------------|
| Line[0,0]     | X <sub>0</sub>     |
| Line[0,1]     | Y <sub>0</sub>     |
| Line[1,0]     | X <sub>1</sub>     |
| Line[1,1]     | Y <sub>1</sub>     |
| Line[2,0]     | X <sub>2</sub>     |
| Line[2,1]     | Y <sub>2</sub>     |
| :             | :                  |
| Line[Num-1,0] | X <sub>Num-1</sub> |
| Line[Num-1,1] | Y <sub>Num-1</sub> |

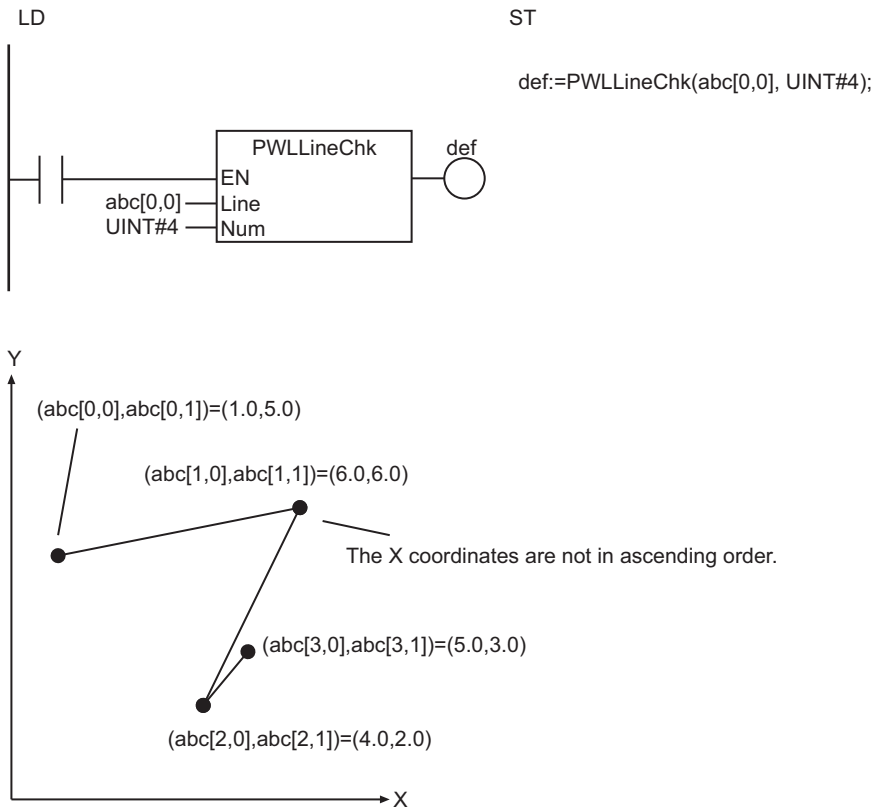
|                  |                    |
|------------------|--------------------|
| Line[0,0,0]      | X <sub>0</sub>     |
| Line[0,0,1]      | Y <sub>0</sub>     |
| Line[0,1,0]      | X <sub>1</sub>     |
| Line[0,1,1]      | Y <sub>1</sub>     |
| Line[0,2,0]      | X <sub>2</sub>     |
| Line[0,2,1]      | Y <sub>2</sub>     |
| :                | :                  |
| Line[0, Num-1,0] | X <sub>Num-1</sub> |
| Line[0, Num-1,1] | Y <sub>Num-1</sub> |

## Notation Example

In the following example, check whether the four elements of *abc[]* (broken line data array) are sorted in ascending order of the X-coordinate values. In this example, *Num* is UINT#4, and the elements of *abc[]* are as below.

- $abc[0,0] = X_0 = \text{LREAL}\#1.0$ ,  $abc[0,1] = Y_0 = \text{LREAL}\#5.0$ ,
- $abc[1,0] = X_1 = \text{LREAL}\#6.0$ ,  $abc[1,1] = Y_1 = \text{LREAL}\#6.0$ ,
- $abc[2,0] = X_2 = \text{LREAL}\#4.0$ ,  $abc[2,1] = Y_2 = \text{LREAL}\#2.0$ ,
- $abc[3,0] = X_3 = \text{LREAL}\#5.0$ ,  $abc[3,1] = Y_3 = \text{LREAL}\#3.0$

The X-coordinate values are not sorted in ascending order, so the value of *Out* is FALSE.



## Additional Information

- Use this instruction in combination with the *PWLApproxNoLineChk* instruction. Refer to *PWLApprox* and *PWLApproxNoLineChk* on page 2-448 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* on page 2-448 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.
  - a) The value of *Num* exceeds the array area of Line[].
  - b) Line[] is REAL data, and its elements are nonnumeric data, positive infinity, or negative infinity.



# MovingAverage

The MovingAverage instruction calculates a moving average.

| Instruction   | Name           | FB/<br>FUN | Graphic expression | ST expression                                      |
|---------------|----------------|------------|--------------------|--|
| MovingAverage | Moving Average | FUN        |                    | Out:=MovingAverage(In, CurIndex, Buf, BufSize, Q); |

## Variables

|               | Meaning                      | I/O    | Description   | Valid range           | Unit | Default |
|---------------|------------------------------|--------|---|-----------------------|------|---------|
| In            | Input value                  | Input  | Number to include in average  | Depends on data type. | ---  | *1      |
| BufSize       | Maximum number stored        |        | Maximum number of elements to include in average  |                       |      | 1       |
| CurIndex      | Input value storage position | In-out | Position in Buf[] to store <i>In</i>  | Depends on data type. | ---  | ---     |
| Buf[] (array) | Input value storage array    |        | Array to store <i>In</i> values   |                       |      |         |
| Q             | Calculation completed flag   |        | TRUE: The number of values stored in Buf[] has reached or exceeded <i>BufSize</i> .<br>FALSE: The number of values stored in Buf[] has not reached <i>BufSize</i> . |                       |      |         |
| Out           | Calculation result           | Output | Calculation result  | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

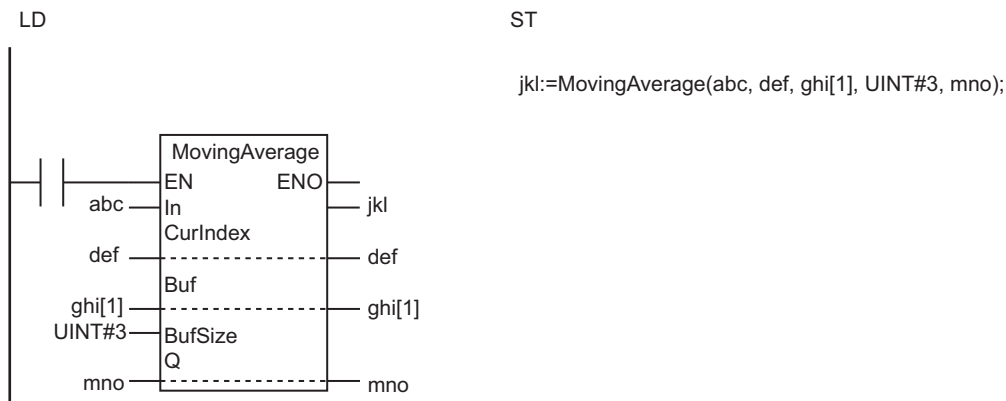
|               | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|---------------|--|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|---------------------|-------|--|------|-----|----|--------|
|               | 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                  |       |  |      |     |    |        |
| BufSize       |  |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| CurIndex      |  |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Buf[] (array) | Must be an array with elements that have the same data type as <i>In</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Q             | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num-<br>bers |      | Times, durations,<br>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    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK   | OK                   |      |  |     |    |        |  |

## Function

The `MovingAverage` instruction stores the value of *In* in `Buf[]` (input value storage array) each time it is executed. And then, it calculates the average of the input values and stores the result in *Out* (calculation result). *BufSize* specifies the maximum number of elements to be included in the average calculation.

The processing procedure is described in the following example, where *BufSize* is `UINT#3`. The instruction is executed as below.



### First Time a Number Is Input

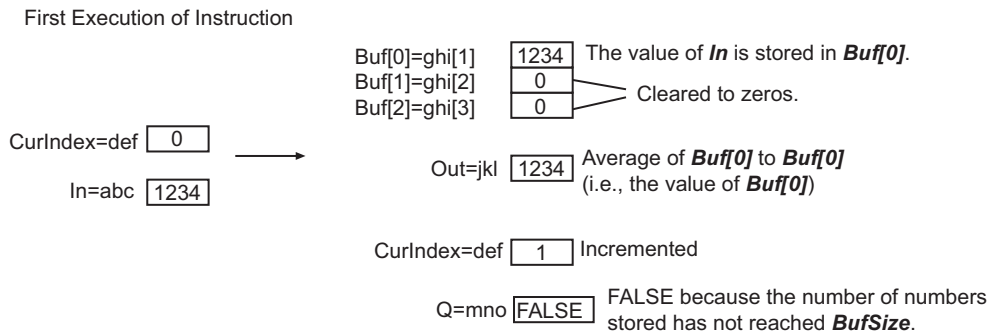
The input value storage position *CurIndex* is set to 0, and the instruction is executed.

`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*.

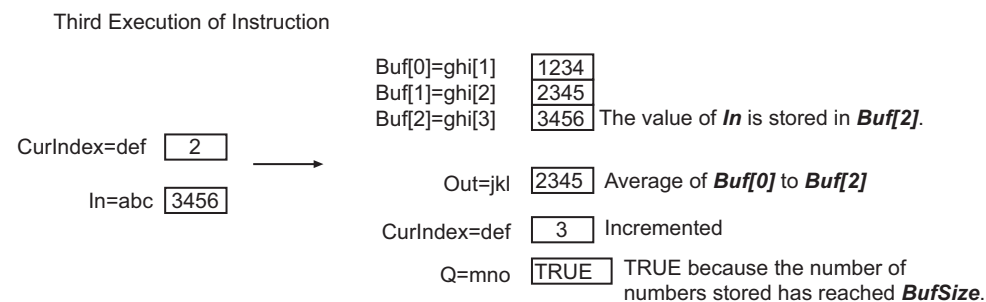
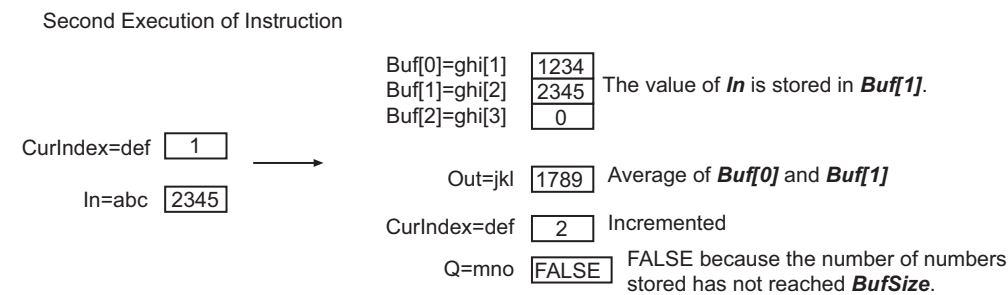
Finally, the value of *CurIndex* is incremented.



## Inputting Numbers Up to *BufSize*

Each time the instruction is executed, the value of *In* is stored in `Buf[CurIndex]`, which starts from `Buf[0]`. The instruction calculates the average of input values as many as `CurIndex + 1`, and stores the result in *Out*.

When the number of instruction executions reaches *BufSize*, the value of *Q* changes to TRUE.

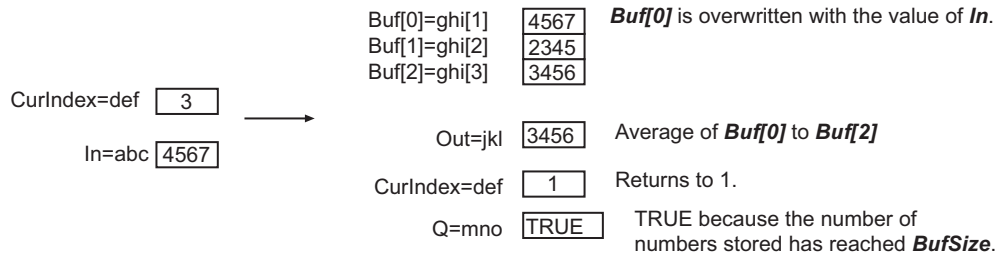


## Inputting Numbers after Reaching *BufSize*

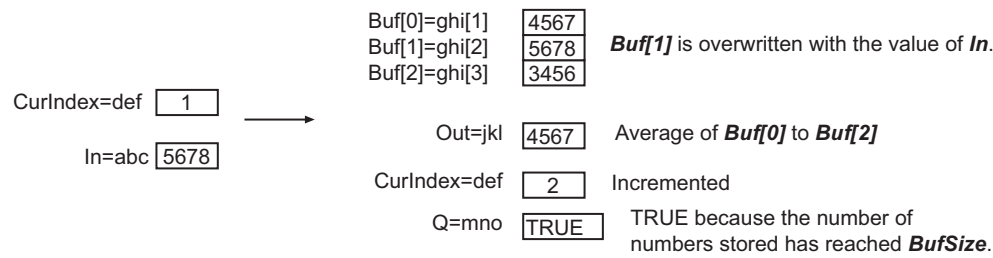
Each time the instruction is executed, `Buf[0]` to `Buf[BufSize-1]` are overwritten with the value of *In* in cyclic fashion. The average of `Buf[0]` to `Buf[BufSize-1]` is calculated and stored in *Out*.

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



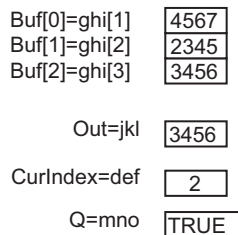
## Initializing the Stored Values

If the value of *CurIndex* is set to 0 before the instruction is executed, the values in *Buf[0]* to *Buf[BufSize-1]* are once set to 0, and then the current value of *In* is stored in *Buf[0]*. The value of *CurIndex* changes to 1, and the value of *Q* changes to FALSE.

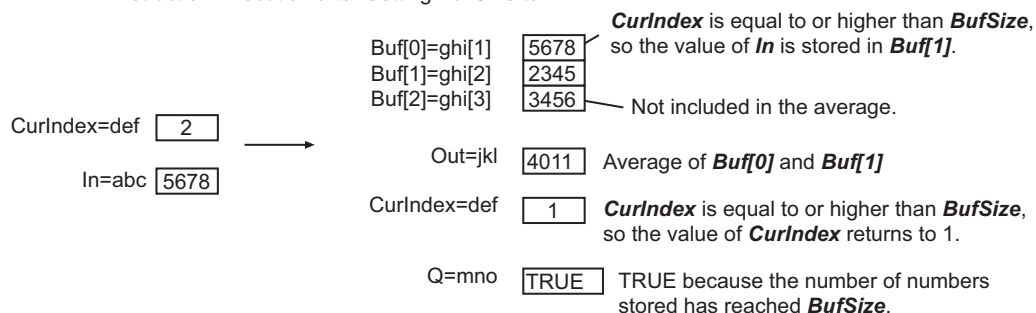
## Changing the Value of *BufSize*

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*.

Status before Instruction Execution BufSize=3



Instruction Execution after Setting **BufSize** to 2



## 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. *Out* will contain an illegal value.
- If the value of *BufSize* is 0, the values of *Out* and *CurlIndex* change to 0. The value of *Q* changes to TRUE.
- If you change the value of *BufSize*, always set the value of *CurlIndex* to 0 and initialize the stored values.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) 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.

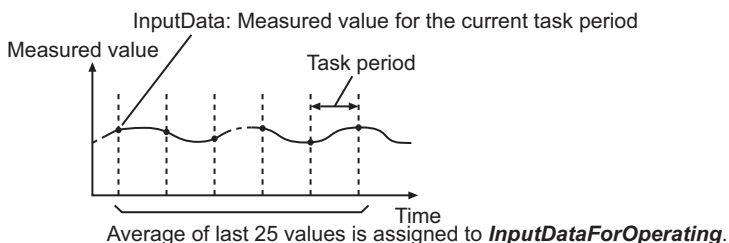
The average of the last 25 values of *InputData* (input data) is calculated as *DataAve*, which is assigned to *InputDataForOperating* as the input data for the next process.

*InputData* is input every task period as long as the value of *Trigger* (execution condition) is TRUE.

The most recent value of *InputData*, instead of the average value, is assigned to

*InputDataForOperating* until 25 values of *InputData* are input for calculating the average.

When the value of *Trigger* changes to TRUE, the calculated average is cleared and input of *InputData* is started again from the beginning.

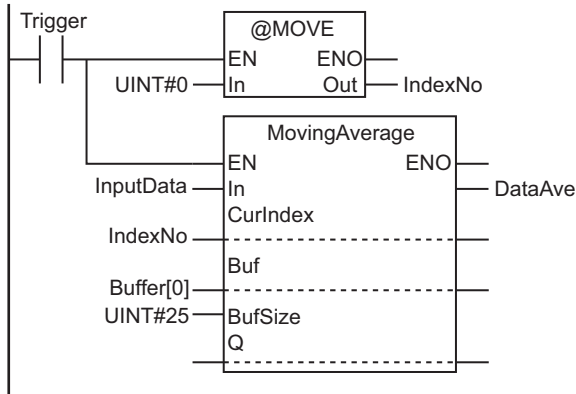


## LD

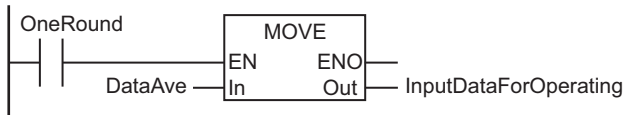
| Variable              | Data type           | Initial value | Comment                       |
|-----------------------|---------------------|---------------|-------------------------------|
| Trigger               | BOOL                | FALSE         | Execution condition           |
| InputData             | 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       |

When **Trigger** changes to TRUE, 0 is assigned to **IndexNo**.

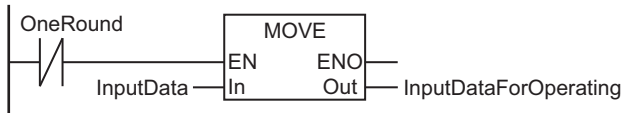
While **Trigger** is TRUE, the value of **InputData** is input every task period and the average is calculated.



When there are 25 or more input values for **InputData**, **DataAve** is assigned to **InputDataForOperating**.



Until there are 25 or more input values for **InputData**, **InputData** is assigned to **InputDataForOperating**.



### ST

| Variable              | Data type           | Initial value | Comment   |
|-----------------------|---------------------|---------------|---|
| Trigger               | BOOL                | FALSE         | Execution condition                                   |
| LastTrigger           | BOOL                | FALSE         | Value of <i>Trigger</i> from the 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                          |
| InputDataForOperating | 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/<br>FUN | Graphic expression | ST expression                             |
|-------------|--------------------------------|------------|--------------------|---|
| DispartReal | Separate Mantissa and Exponent | FUN        |                    | Out:=DispartReal(In, Fraction, Exponent); |

## Variables

|          | 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 *Valid Range of Fraction* on page 2-465 for details.
- \*3. The valid range is from -44 to 32 if the data type of *In* is REAL data, and from -322 to 294 if it is LREAL data.

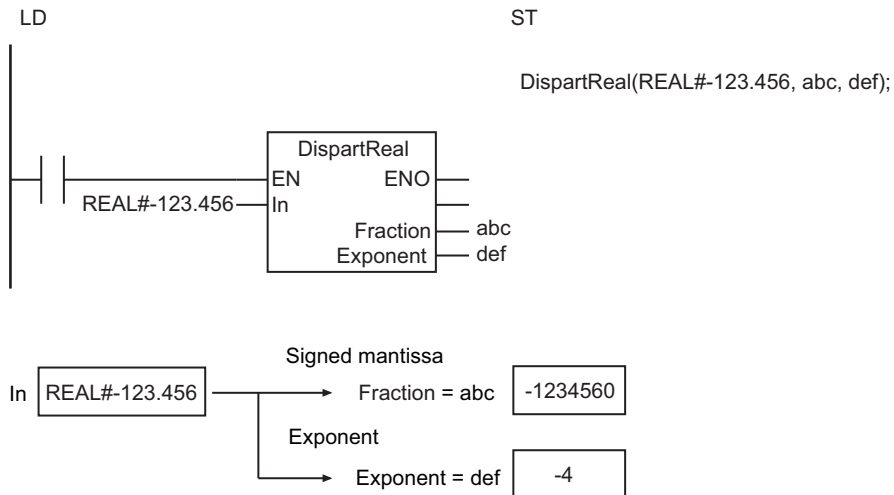
|          | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|----------|--|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|          | 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 <i>In</i> is REAL, and LINT if it is LREAL. |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Exponent |  |             |      |       |       |          |      |       |       |      | OK  |      |                      |      |  |      |      |     |    |        |

## Function

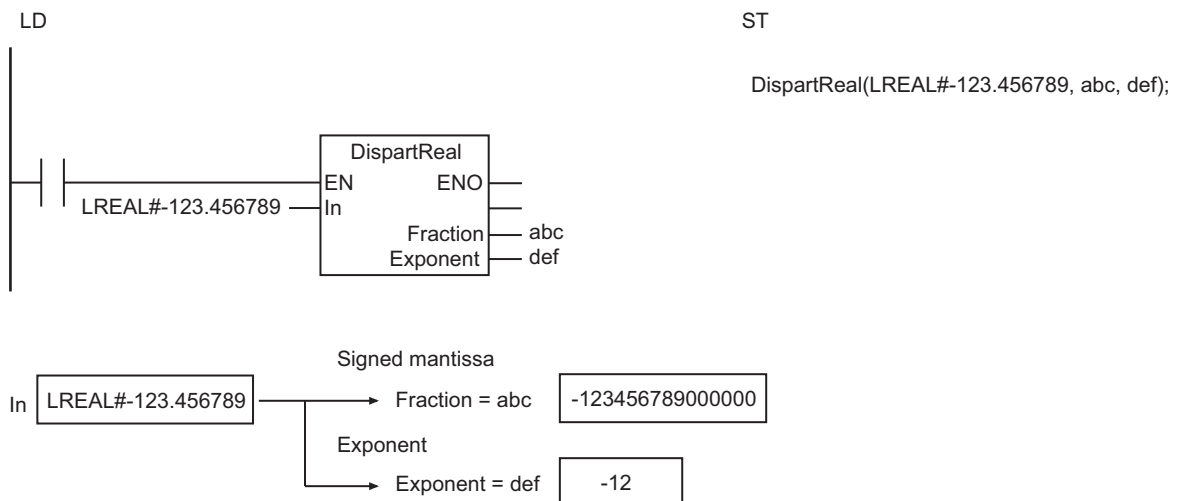
The DispartReal instruction separates *In* (real number) into the signed mantissa and exponent *Exponent*.

If *In* is REAL data, *Fraction* is a 7-digit integer. If *In* is LREAL data, *Fraction* is a 15-digit integer. The following shows an example where *In* is REAL#-123.456.





The following shows an example where *In* is LREAL#-123.456789.



## Valid Range of *Fraction*

The following table shows the valid value range of *Fraction* according to the data types of *In* and *Fraction*.

| Data type of <i>In</i> | Data type of <i>Fraction</i> | Valid value range of <i>Fraction</i>  |
|------------------------|------------------------------|---------------------------------------|
| REAL                   | DINT                         | -9999999 to 9999999                   |
| LREAL                  | LINT                         | -9999999999999999 to 9999999999999999 |

## Additional Information

Use the instruction, *UniteReal* on page 2-467, to combine a signed mantissa and an 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.

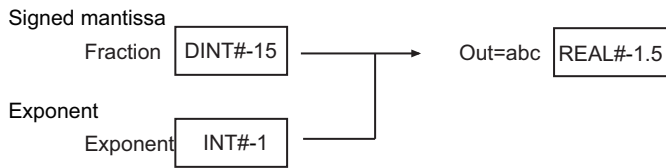
- If the number of valid digits in *In* exceeds the number of valid digits of *Fraction*, the value is rounded to fit in the valid range of *Fraction*.
- An error will occur in the following case. *ENO* will be FALSE, and *Fraction* and *Exponent* will not change.
  - a) The value of *In* is nonnumeric or infinity.

## Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | Examples   |
|--------------------------|---|--|
| Less than 0.5            | The fractional part is truncated.   | 1.49 → 1<br>-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<br>2.50 → 2<br>-1.50 → -2<br>-2.50 → -2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2<br>-1.51 → -2                           |





### Additional Information

Use the instruction, *DispartReal* on page 2-464, to separate a real number into the signed mantissa and the 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/<br>FUN | Graphic expression | ST expression                     |
|----------------|--|------------|--------------------|-----------------------------------|
| NumToDecString | Fixed-length<br>Decimal Text<br>String Conver-<br>sion     | FUN        |                    | Out:=NumToDecString(In, L, Fill); |
| NumToHexString | Fixed-length<br>Hexadecimal<br>Text String Con-<br>version | FUN        |                    | Out:=NumToHexString(In, L, Fill); |

## Variables

|      | Meaning              | I/O    | Description                        | Valid range           | Unit | Default |
|------|----------------------|--------|------------------------------------|-----------------------|------|---------|
| In   | Integer              | Input  | Integer                            | Depends on data type. | ---  | *1      |
| L    | Number of characters |        | Number of characters in <i>Out</i> | 0 to 1985             |      | 1       |
| Fill | Fill character       |        | Fill character                     | _BLANK or _ZERO       |      | _BLANK  |
| Out  | Text string          | Output | Text string                        | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|      | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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   |      |              |       |   |      |     |    |        |
| L    |   |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Fill | Refer to <i>Function</i> on page 2-469 for the enumerators of the enumerated type <code>_eFILL_CHR</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Out  |   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    | OK     |

## Function

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*.





# HexStringToNum\_\*\*

The HexStringToNum\_\*\* instruction converts a hexadecimal text string to an integer.

| Instruction       | Name   | FB/<br>FUN | Graphic expression | ST expression   |
|-------------------|--|------------|--------------------|---|
| HexStringToNum_** | Hexadecimal Text String-to-Number Conversion Group | FUN        |                    | Out:=HexStringToNum_**(In);<br>"****" must be an integer data type. |

## Variables

|     | 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. | ---  | ---     |

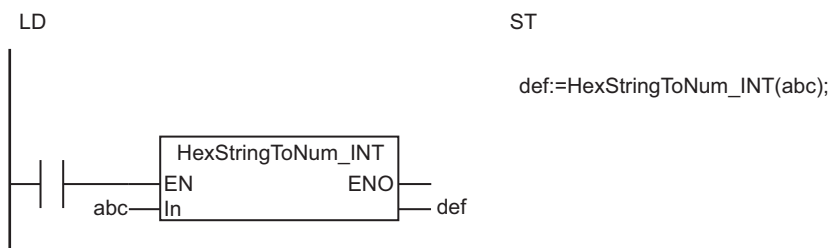
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|     | 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 the data type of *Out* is INT, the instruction name is HexStringToNum\_INT.

A few examples are given below.





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. *Out* will contain an illegal value.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *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/<br>FUN | Graphic expression | ST expression                  |
|----------------|--|------------|--------------------|--------------------------------|
| FixNumToString | Fixed-decimal Number-to-Text String Conversion | FUN        |                    | Out:=FixNumToString(In, Zero); |

## Variables

|      | Meaning              | I/O    | Description   | Valid range           | Unit | Default |
|------|----------------------|--------|---|-----------------------|------|---------|
| In   | Fixed-decimal number | Input  | Signed fixed-decimal number   | Depends on data type. | ---  | 0       |
| Zero | Zero augmentation    |        | Augmentation of zeros if there are less than 3 decimal digits<br>TRUE: Add '0'<br>FALSE: Do not add '0' |                       |      | TRUE    |
| Out  | Decimal text string  | Output | Decimal text string   | Depends on data type. | ---  | ---     |

|      | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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

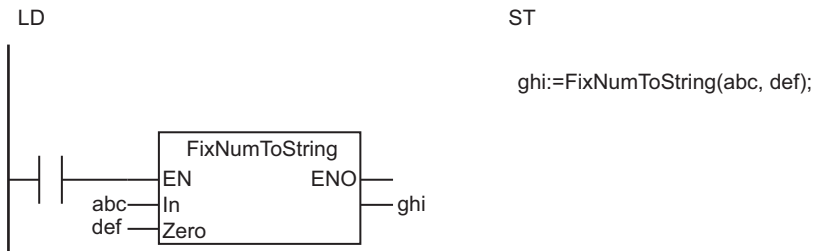
The FixNumToString instruction converts signed fixed-decimal number *In* to a decimal text string. The following conversion is performed.

- 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 pad decimal places of *Out* with '0' to make the value with three decimal digits when *In* has two or less decimal digits. If the value of *Zero* is TRUE, zero padding takes place.

A NULL character is placed at the end of *Out*.

A few examples are given below.



| In = abc                   | Out = ghi         |                    |
|----------------------------|-------------------|--------------------|
|                            | Zero = def = TRUE | Zero = def = FALSE |
| 16#0001462C<br>(10#83500)  | '83.500'          | '83.5'             |
| 16#00051AA4<br>(10#334500) | '334.500'         | '334.5'            |
| 16#0003BEFC<br>(10#245500) | '245.500'         | '245.5'            |

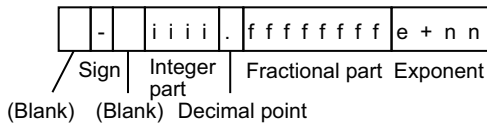
## Additional Information

The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OM-  
RON FZ-series Vision Sensors.



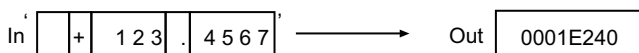
| In = abc | Out = def                  |
|----------|----------------------------|
| '83.5'   | 16#0001462C<br>(10#83500)  |
| '334.5'  | 16#00051AA4<br>(10#334500) |
| '245.5'  | 16#0003BEFC<br>(10#245500) |

The text sting format of *In* is given below.



| Name            | Format  |
|-----------------|---|
| Sign            | <ul style="list-style-type: none"> <li>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.</li> <li>The sign can be omitted.</li> <li>Any consecutive blank characters after the sign are ignored.</li> </ul>  |
| Integer part    | <ul style="list-style-type: none"> <li>Numbers ('0' to '9') placed between the sign and the decimal point are taken as the integer part. The sign may be omitted. There may be blank characters between the sign and the integer part.</li> <li>If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part.</li> <li>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.</li> <li>The integer part cannot be omitted.</li> <li>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.</li> </ul> |
| Decimal point   | <ul style="list-style-type: none"> <li>A single dot ('.') following the integer part is taken as the decimal point.</li> <li>Omit the decimal point if there is no fractional part.</li> </ul>  |
| Fractional part | <ul style="list-style-type: none"> <li>Numbers ('0' to '9') placed between the decimal point and the exponent are taken as the fractional part.</li> <li>If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part.</li> <li>The fractional part can be omitted. If there is no decimal point, then there is no fractional part.</li> <li>The fractional part can consist of a maximum of 15 digits.</li> </ul>  |
| Exponent        | <ul style="list-style-type: none"> <li>The exponent consists of a single 'e' or 'E' after the fractional part, a following single plus or minus sign, and the remaining numbers ('0' to '9') to the end of the text string.</li> <li>If there is no fractional part, then the above text string after the decimal point is taken as the exponent.</li> <li>If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent.</li> <li>The exponent can be omitted.</li> <li>The numeric part of the exponent can consist of a maximum of three digits.</li> </ul>  |

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

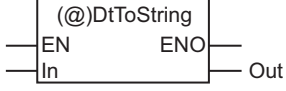
The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OM-RON FZ-series Vision Sensors.

### Precautions for Correct Use

- The value of *In* is truncated to three decimal places.
- Underbars (16#5F) in the text string in *In* are ignored.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* includes characters that cannot be converted to numbers.
  - b) *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/<br>FUN | Graphic expression  | ST expression        |
|-------------|---|------------|---|----------------------|
| DtToString  | Date and Time-to-Text String Conversion | FUN        |  | Out:=DtToString(In); |

## Variables

|     | 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) | ---                                      | ---               |

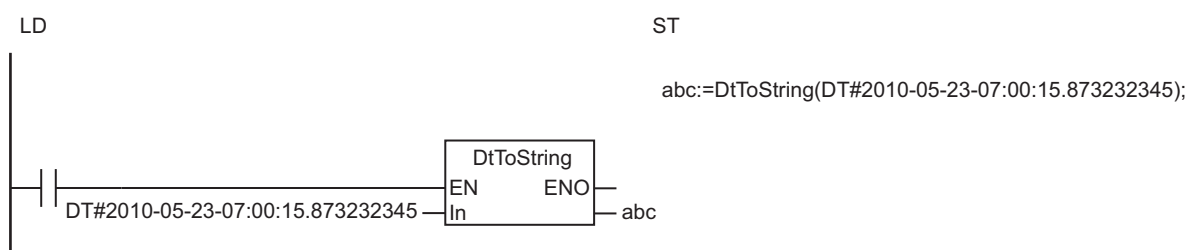
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|     | 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*.

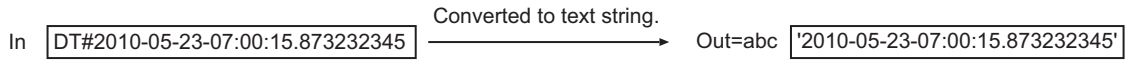
The following shows an example where *In* is 2010-5-23-07:00:15.873232345 (7:00 am and 15.873232345 seconds on May 23, 2010).

The value of variable *abc* will be '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'.

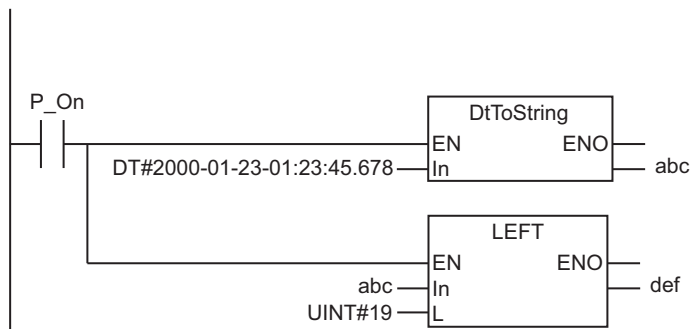


### Additional Information

*Out* is represented in nanoseconds. To get a text string in seconds or milliseconds, combine this instruction with the instructions, *LEFT* and *RIGHT* on page 2-584.

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);
```



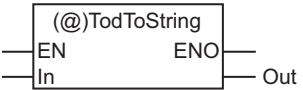


The `DateToString` instruction converts date *ln* to a text string.  
The value of *ln* is May 23, 2010, so the value of *abc* will be '2010-05-23'.

In D#2010-05-23  $\xrightarrow{\text{Converted to text string.}}$  Out=abc '2010-05-23'

# TodToString

The TodToString instruction converts a time of day to a text string.

| Instruction | Name                                  | FB/<br>FUN | Graphic expression  | ST expression         |
|-------------|---------------------------------------|------------|---|-----------------------|
| TodToString | Time of Day-to-Text String Conversion | FUN        |  | Out:=TodToString(In); |

## Variables

|     | 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 alphanumeric characters plus the final NULL character) | ---                    | ---       |

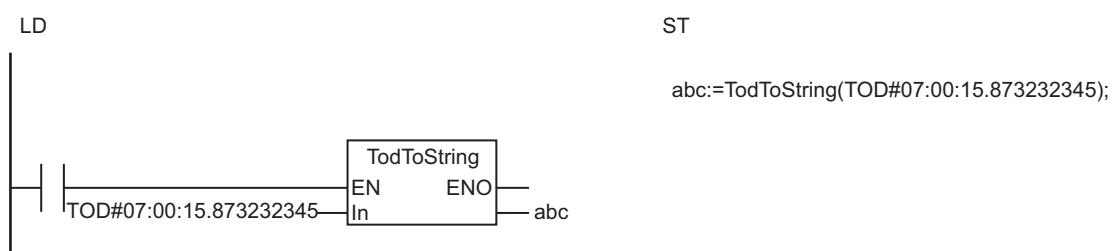
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|     | 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*.

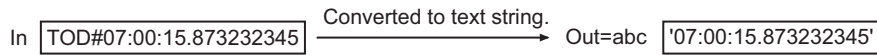
The following shows an example where *In* is 07:00:15.873232345 (7:00 am and 15.873232345 seconds).

The value of variable *abc* will be '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'.



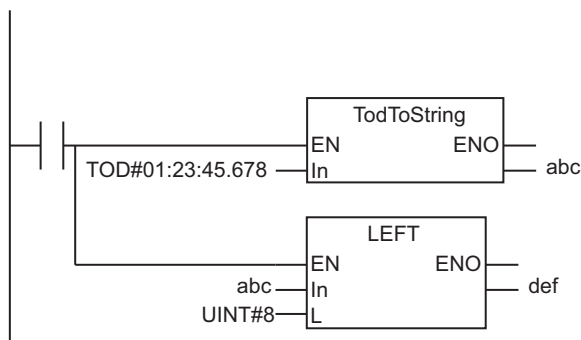
### Additional Information

*Out* is represented in nanoseconds.

To get a text string in seconds or milliseconds, combine this instruction with the instructions, *LEFT* and *RIGHT* on page 2-584.

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/<br>FUN | Graphic expression                          | ST expression  |
|--------------|---|------------|---|--|
| GrayToBin_** | Gray Code-to-Binary Code Conversion Group | FUN        | <p>**** must be a bit string data type.</p> | Out:=GrayToBin_**(In);<br>**** must be a bit string data type. |
| BinToGray_** | Binary Code-to-Gray Code Conversion       | FUN        | <p>**** must be a bit string data type.</p> | Out:=BinToGray_**(In);<br>**** must be a bit string data type. |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean                             | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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 the same data type as <i>In</i> |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |

## Function

The names of the instructions are determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the names of the instructions are GrayToBin\_WORD and BinToGray\_WORD.

### GrayToBin\_\*\*

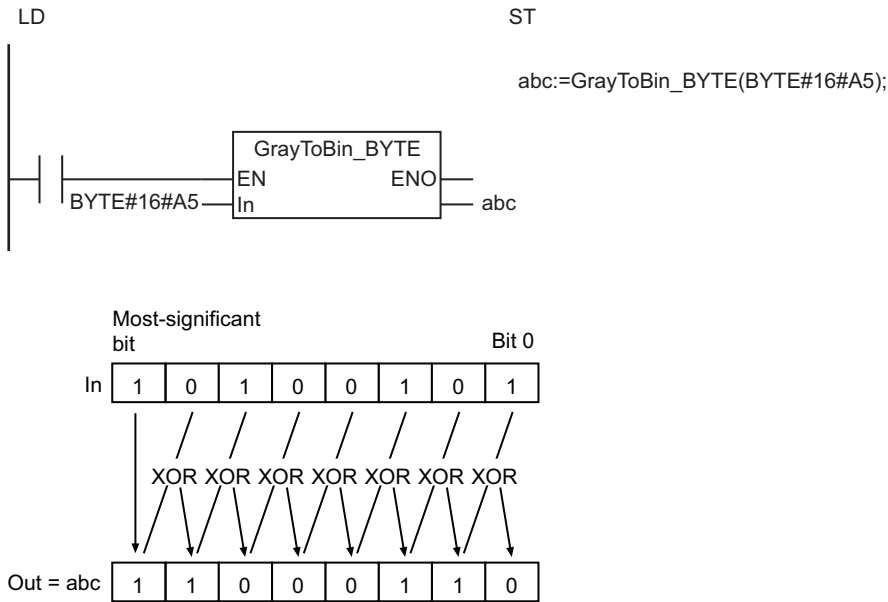
The GrayToBin\_\*\* instructions convert the gray code in date to convert *In* to a bit string.

If *In* and *Out* are BYTE data, the conversion procedure is as follows.

- 1 The most-significant bit (bit 7) of *In* is assigned to the most-significant bit (bit 7) of *Out*.

- 2** The result of an exclusive logical OR operation on bit 6 of *In* and bit 7 of *Out* is assigned to bit 6 of *Out*.
- 3** This process is repeated through the least-significant bit (bit 0) of *Out*.

The following shows an example where *In* is BYTE#16#A5 for the GrayToBin\_BYTE instruction.



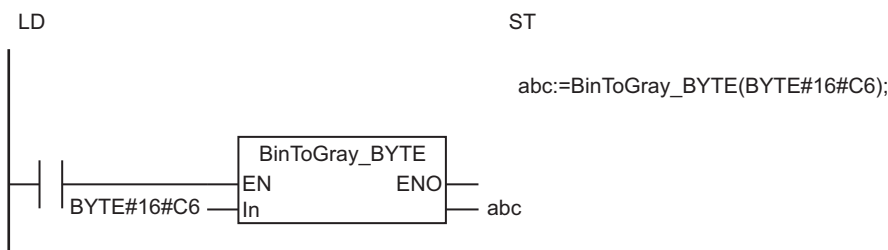
## BinToGray\_\*\*

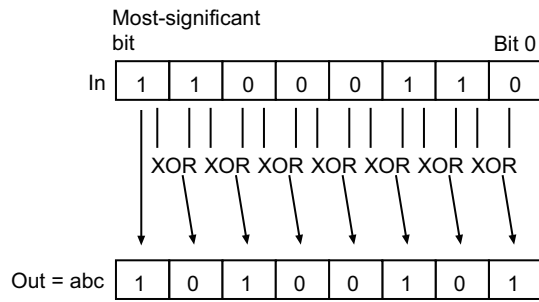
The BinToGray\_\*\* instructions convert the bit string in data to convert *In* to a gray code.

If *In* and *Out* are BYTE data, the conversion procedure is as follows.

- 1** The most-significant bit (bit 7) of *In* is assigned to the most-significant bit (bit 7) of *Out*.
- 2** The result of an exclusive logical OR operation on bit 7 of *In* and bit 6 of *In* is assigned to bit 6 of *Out*.
- 3** This process is repeated through the least-significant bit (bit 0) of *Out*.

The following shows an example where *In* is BYTE#16#C6 for the BinToGray\_BYTE instruction.





## 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/<br>FUN | Graphic expression | ST expression                 |
|-------------|---------------------------------|------------|--------------------|-------------------------------|
| StringToAry | Text String-to-Array Conversion | FUN        |                    | Out:=StringToAry(In, AryOut); |

## Variables

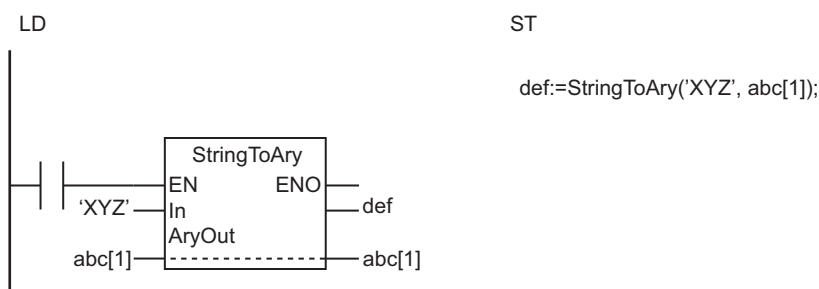
|                  | Meaning                    | I/O    | Description                | Valid range           | Unit  | Default |
|------------------|----------------------------|--------|----------------------------|-----------------------|-------|---------|
| In               | Text string                | 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 | ---     |

|                  | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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       |       |       |      |     |      |      |              |       |   |      |     |    |        |

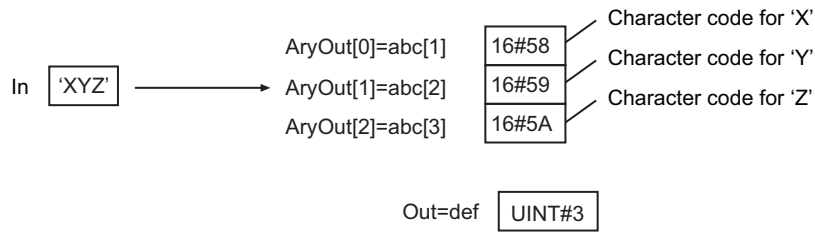
## Function

The StringToAry instruction processes the character codes in text string *In* as numeric values, and assigns each numeric value to the corresponding element of the BYTE array, AryOut[]. The number of converted bytes is stored in *Out*.

The following shows an example where *In* is 'XYZ'.







## Precautions for Correct Use

- The NULL character at the end of *In* is not stored in *AryOut*].
- If the *In* text string contains only NULL characters, the value of *Out* will be 0 and *AryOut*[] will not change.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* and *AryOut*[] will not change.
  - a) 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/<br>FUN | Graphic expression | ST expression               |
|-------------|---------------------------------|------------|--------------------|-----------------------------|
| AryToString | Array-to-Text String Conversion | FUN        |                    | Out:=AryToString(In, Size); |

## Variables

|              | Meaning                       | I/O    | Description                                    | Valid range           | Unit | Default |
|--------------|-------------------------------|--------|--|-----------------------|------|---------|
| In[] (array) | BYTE array                    | Input  | BYTE array<br>Maximum number of elements: 1985 | Depends on data type. | ---  | *1      |
| Size         | Number of elements to convert |        | Number of elements of In[] for conversion      | 0 to 1985             |      | 1       |
| Out          | Text string                   | Output | Text string                                    | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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     |

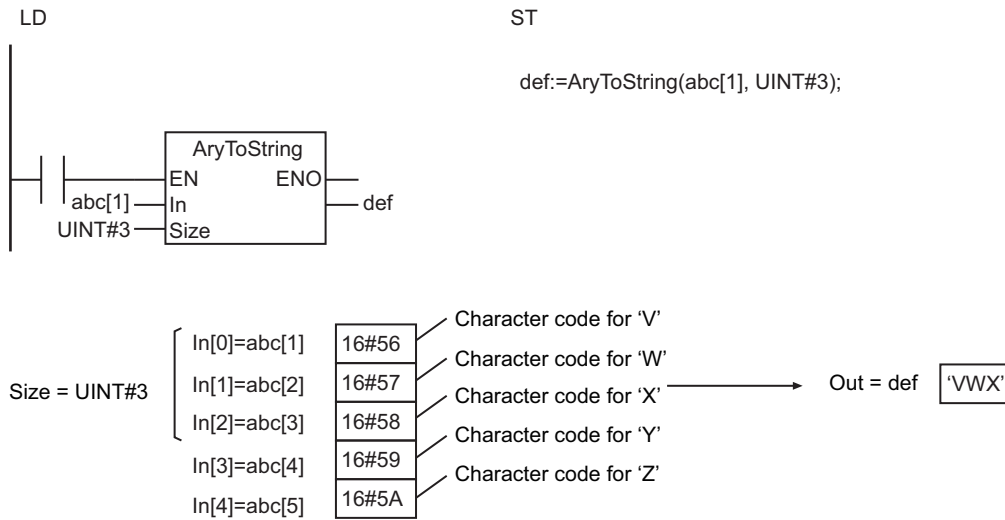
## Function

The AryToString instruction processes the elements of In[] (BYTE array), which begin with In[0], as character codes, and converts them into a text string to be stored in Out.

A NULL character is placed at the end of Out.

Size specifies the number of elements of In[] to convert. If a NULL character is included between In[0] and In[Size-1], only character codes before the Null character are stored in Out.

The following shows an example where Size is UINT#3.



## Precautions for Correct Use

- If the value of *Size* is 0, *Out* is a text string containing only NULL characters.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) 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/<br>FUN | Graphic expression | ST expression                  |
|--------------|---------------------|------------|--------------------|--------------------------------|
| DispartDigit | Four-bit Separation | FUN        |                    | DispartDigit(In, Num, AryOut); |

## Variables

|                  | Meaning                      | I/O    | Description                  | Valid range                          | Unit | Default |
|------------------|------------------------------|--------|------------------------------|--------------------------------------|------|---------|
| In               | Data to separate             | Input  | Bit string to separate       | Depends on data type.                | ---  | *1      |
| Num              | Number of digits to separate |        | Number of digits to separate | 0 to the number of bits in <i>In</i> |      | 1       |
| AryOut[] (array) | Separation results array     | In-out | Separation results array     | 16#00 to 16#0F                       | ---  | ---     |
| 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.

|                  | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

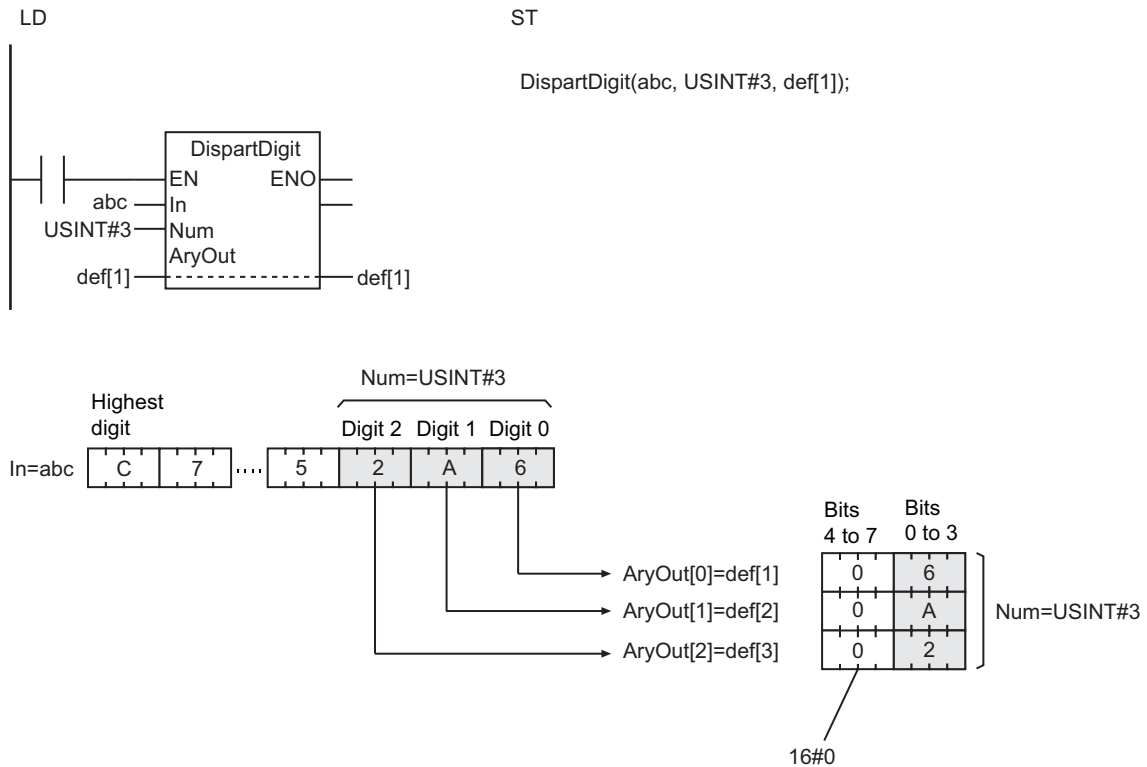
## Function

The DispartDigit instruction separates *In* (data to separate) into 4-bit units (digit-based), and stores them in separation results array *AryOut[]*.

First, *In* is separated into 4-bit units. Then, the four lowest bits are stored in *AryOut[0]*. *AryOut[0]* is BYTE data, and 16#0 is assigned to bits 4 to 7.

This process is repeated for the number of digits to separate, *Num*.

The following shows an example where *Num* is USINT#3.



## Additional Information

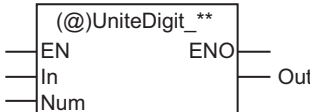
Use the instruction, *UniteDigit\_\*\** on page 2-494, to join four bits of each element together into a single bit string.

## 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 will occur in the following cases. *ENO* will be FALSE, and *AryOut*[] will not change.
  - a) The value of *Num* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression  | ST expression  |
|---------------|---------------------|------------|---|--|
| UniteDigit_** | Four-bit Join Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=UniteDigit_**(In, Num);<br>**** must be a bit string data type. |

## Variables

|              | Meaning                  | I/O    | Description                   | Valid range                           | Unit | Default |
|--------------|--------------------------|--------|-------------------------------|---------------------------------------|------|---------|
| In[] (array) | Array to join            | Input  | Array to join                 | Depends on data type.                 | ---  | *1      |
| Num          | Number of digits to join |        | Number of digits to join      | 0 to the number of bits in <i>Out</i> |      | 1       |
| Out          | Joined result            | Output | Bit string with joined result | Depends on data type.                 | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 joins the lower four bits (digit-based) of each element of In[] (array to join) and creates a bit string for Out (joined result).

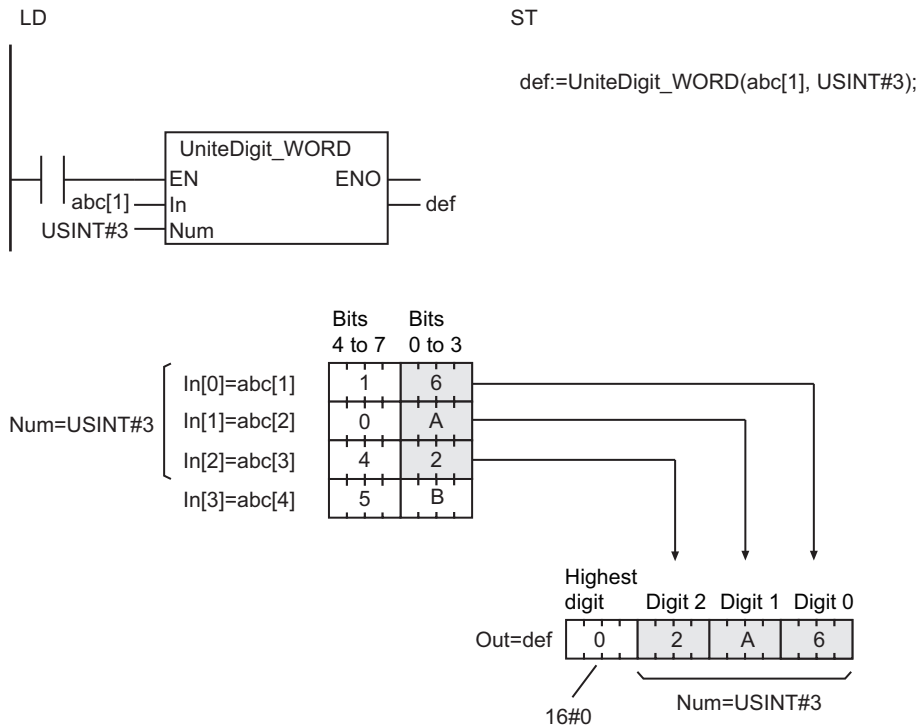
*Num* (number of digits to join) specifies the number of array elements for the joining.

First, the lower four bits of each element of In[], from In[0] to In[Num-1], are joined to create a bit string with *Num* digits.

16#0, which the number of digits in *Out* minus *Num* equals, is joined to the bit string as its upper digit, and the joined string is assigned to *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the name of the instruction is UniteDigit\_WORD.

The following shows an example where *Num* is USINT#3 for the UniteDigit\_WORD instruction.



## Additional Information

Use the instruction, *DispartDigit* on page 2-492, 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Num* is outside the valid range.
  - b) The value of *Num* exceeds the array area of *In*[],

# Dispart8Bit

The Dispart8Bit instruction separates a bit string into individual bytes.

| Instruction | Name                 | FB/<br>FUN | Graphic expression   | ST expression                 |
|-------------|----------------------|------------|--|-------------------------------|
| Dispart8Bit | Byte Data Separation | FUN        | <pre> graph LR     subgraph Dispart8Bit         EN[EN]         In[In]         Num[Num]         AryOut[AryOut]     end     Out[Out]     EN --- Dispart8Bit     In --- Dispart8Bit     Num --- Dispart8Bit     AryOut --- Dispart8Bit     Dispart8Bit --- Out   </pre> | Dispart8Bit(In, Num, AryOut); |

## Variables

|                  | Meaning                     | I/O    | Description                 | Valid range                           | Unit | Default |
|------------------|-----------------------------|--------|-----------------------------|---------------------------------------|------|---------|
| In               | Data to separate            | Input  | Bit string to separate      | Depends on data type.                 | ---  | *1      |
| Num              | Number of bytes to separate |        | Number of bytes to separate | 0 to the number of bytes in <i>In</i> |      | 1       |
| AryOut[] (array) | Separation results array    | In-out | Separation results array    | 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.

|                  | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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

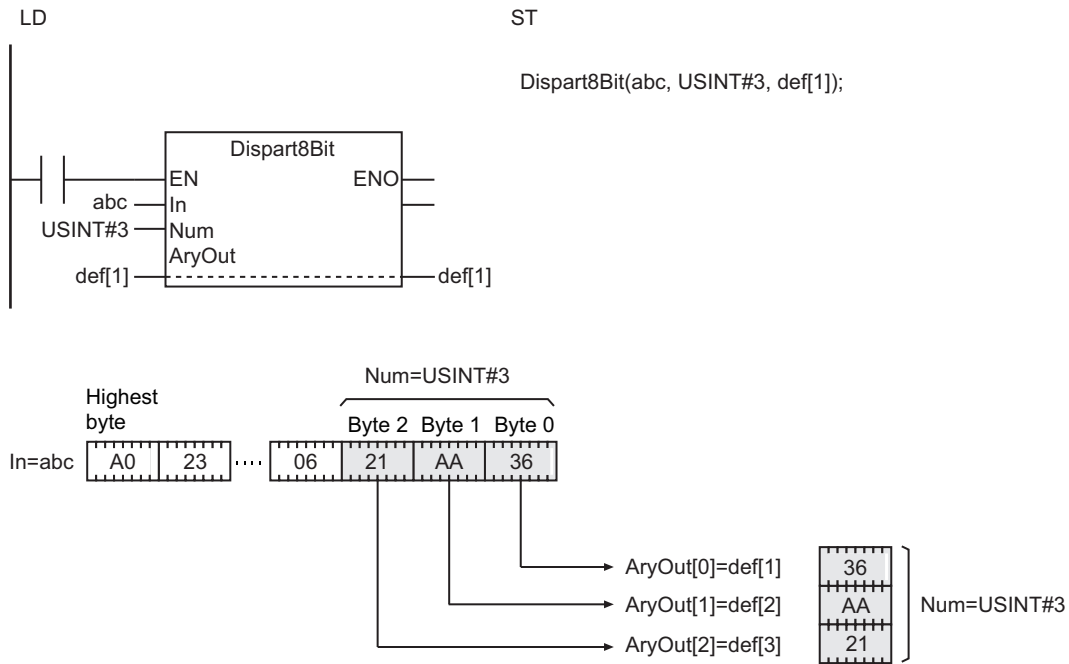
The Dispart8Bit instruction separates *In* (data to separate) into individual bytes, and stores them in separation results array *AryOut[]*.

First, *In* is separated into bytes. The lowest byte is stored in *AryOut[0]*.

The second lowest byte is stored in *AryOut[1]*. This process is repeated for the number of bytes to separate, *Num*.

The following shows an example where *Num* is USINT#3.





## Additional Information

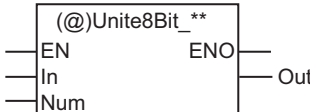
Use the instruction, *Unite8Bit\_\*\** on page 2-498, to join one byte of each element together into a single bit string.

## Precautions for Correct Use

- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - a) The value of *Num* is outside the valid range.
  - b) The value of *Num* exceeds the array area of *AryOut[]*.

# Unite8Bit\_\*\*

The Unite8Bit\_\*\* instructions join bytes of data into a bit string.

| Instruction  | Name                 | FB/<br>FUN | Graphic expression  | ST expression   |
|--------------|----------------------|------------|---|---|
| Unite8Bit_** | Byte Data Join Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=Unite8Bit_**(In, Num);<br>**** must be a bit string data type. |

## Variables

|              | Meaning                 | I/O    | Description                   | Valid range                            | Unit | Default |
|--------------|-------------------------|--------|-------------------------------|--|------|---------|
| In[] (array) | Array to join           | Input  | Array to join                 | Depends on data type.                  | ---  | *1      |
| Num          | Number of bytes to join |        | Number of bytes to join       | 0 to the number of bytes in <i>Out</i> |      | 1       |
| Out          | Joined result           | Output | Bit string with joined result | Depends on data type.                  | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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

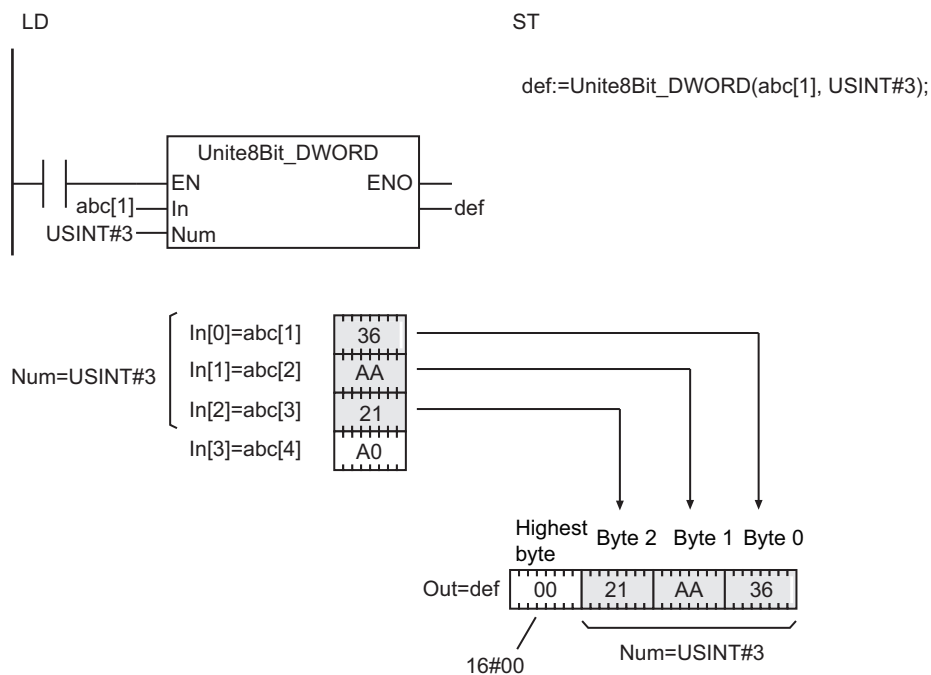
The Unite8Bit\_\*\* instructions join elements of In[] (array to join) to create a bit string in *Out* (joined result).

*Num* (number of bytes to join) specifies the number of array elements to join. First, elements from In[0] to In[Num-1] are joined to create a bit string with *Num* bytes.

16#0, which the number of bytes in *Out* minus *Num* equals, is joined to the bit string as its upper byte, and the joined string is assigned to *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the DWORD data type, the name of the instruction is Unite8Bit\_DWORD.

The following example shows the Unite8Bit\_DWORD instruction when *Num* is USINT#3.



## Additional Information

Use the instruction, *Dispart8Bit* on page 2-496, to separate a bit string into 1-byte units.

## Precautions for Correct Use

- If the value of *Num* is 0, the value of *Out* will be 0.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Num* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression                      |
|-------------|--------------------------|------------|--------------------|------------------------------------|
| ToAryByte   | Conversion to Byte Array | FUN        |                    | Out:=ToAryByte(In, Order, AryOut); |

## Variables

|                  | Meaning                      | I/O    | Description                  | Valid range            | Unit | Default   |
|------------------|------------------------------|--------|------------------------------|------------------------|------|-----------|
| In               | Data to convert              | Input  | Data to convert              | Depends on data type.  | ---  | *1        |
| 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.  | ---  | ---       |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-500 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| AryOut[] (array) |   | OK          |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| Out              |   |             |      |       |       |          | OK   |       |       |      |     |      |              |      |   |      |      |     |    |        |

## Function

The ToAryByte instruction separates the value of *In* into individual bytes and stores them in order in AryOut[] (conversion results array) 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 meanings of the enumerators are as follows:

| Enumerator | Meaning                            |
|------------|------------------------------------|
| _LOW_HIGH  | Lower byte first, higher byte last |

| Enumerator | Meaning                            |
|------------|------------------------------------|
| _HIGH_LOW  | Higher byte first, lower byte last |

## When the Data Type of *In* Is Two Bytes or Larger

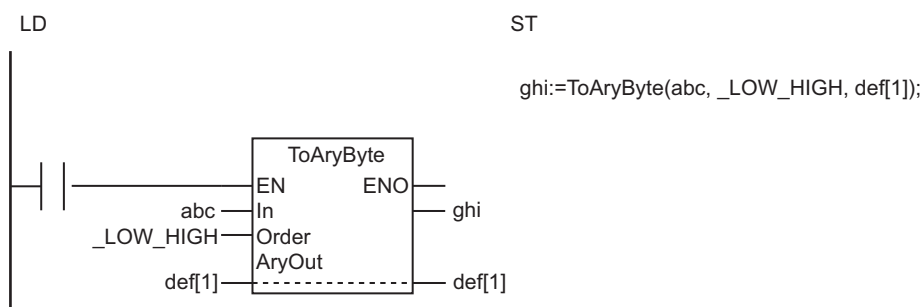
If the data type of *In* is two bytes or larger, *In* is separated into bytes and stored in *AryOut*[]. The following data types have two bytes or more.

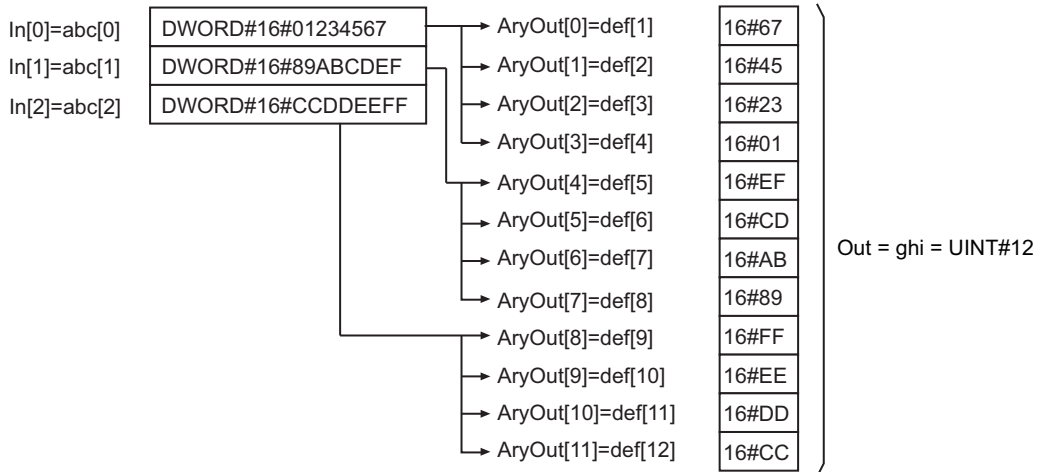
| Classification                            | Data type  |
|---|--|
| Bit strings                               | WORD, DWORD, and LWORD   |
| Integers                                  | UINT, UDINT, ULINT, INT, DINT, and LINT  |
| Real numbers                              | REAL and LREAL   |
| Times, durations, dates, and text strings | TIME, DATE, TOD, DT, and STRING types of two bytes or more   |
| Others                                    | <ul style="list-style-type: none"> <li>An enumeration</li> <li>An array for which the total for all elements is 2 bytes or more</li> <li>An array element that is 2 bytes or more</li> <li>A structure for which the total for all members is 2 bytes or more</li> <li>A structure member that is 2 bytes or more</li> </ul> |

The processing procedure is as follows:

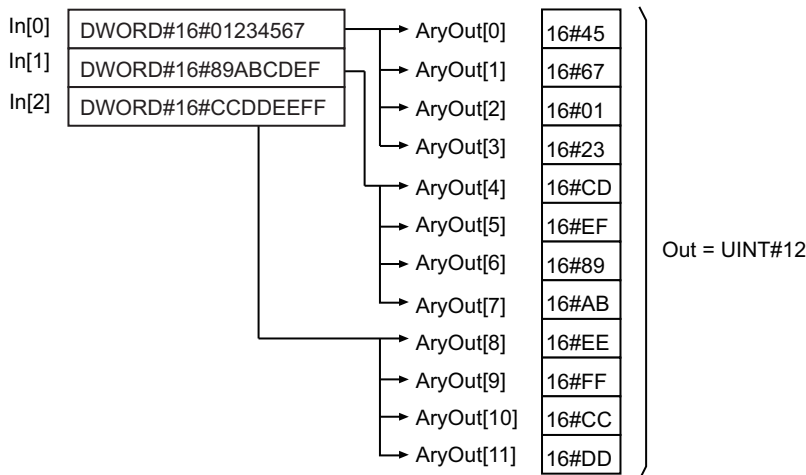
- 1 First, the value in *In* is separated into words (two bytes).
- 2 The lowest word is separated into bytes.
- 3 If *Order* is `_LOW_HIGH`, the lower byte is stored in *AryOut*[0] and the upper byte is stored in *AryOut*[1]. If *Order* is `_HIGH_LOW`, the upper byte is stored in *AryOut*[0] and the 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 of *In*.

The following shows an example where *In* is a DWORD array with three elements and *Order* is `_LOW_HIGH`.





The following shows an example where *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.

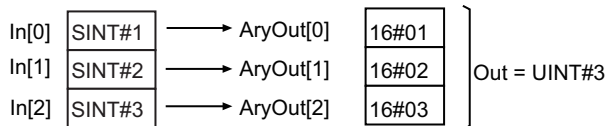
| Classification                            | Data type  |
|---|--|
| Bit strings                               | BYTE   |
| Integers                                  | USINT and SINT   |
| Real numbers                              | None   |
| Times, durations, dates, and text strings | STRING types with one byte   |
| Others                                    | <ul style="list-style-type: none"> <li>An array for which the total for all elements is 1 byte</li> <li>An array element that is 1 byte</li> <li>A structure for which the total for all members is 1 byte</li> <li>A structure member that is 1 byte</li> </ul> |

The following storage method is used.

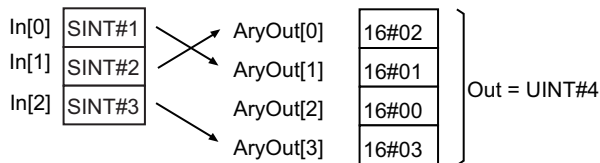
| Value of <i>Order</i>  | <i>In</i> (array or not) | Storage method in <code>AryOut[]</code>                           |
|------------------------|--------------------------|---|
| <code>_LOW_HIGH</code> | Not an array             | Value of <i>In</i> is stored in <code>AryOut[0]</code> .          |
|                        | Array                    | Value of <code>In[i]</code> is stored in <code>AryOut[i]</code> . |

| Value of Order | In (array or not) | Storage method in AryOut[]   |
|----------------|-------------------|--|
| _HIGH_LOW      | Not an array      | Value of <i>In</i> is stored in AryOut[1].<br>16#00 is stored in AryOut[0].  |
|                | Array             | In[ <i>i</i> ] (where <i>i</i> is even) is stored in AryOut[ <i>i</i> +1].<br>In[ <i>i</i> ] (where <i>i</i> is odd) is stored in AryOut[ <i>i</i> -1].<br>If the number of elements in In[] is odd, 16#00 is stored last in AryOut[ <i>n</i> -1]. |

The following shows an example where *In* is a SINT array with three elements and *Order* is \_LOW\_HIGH.



The following shows an example where *In* is the same as above and *Order* is \_HIGH\_LOW.

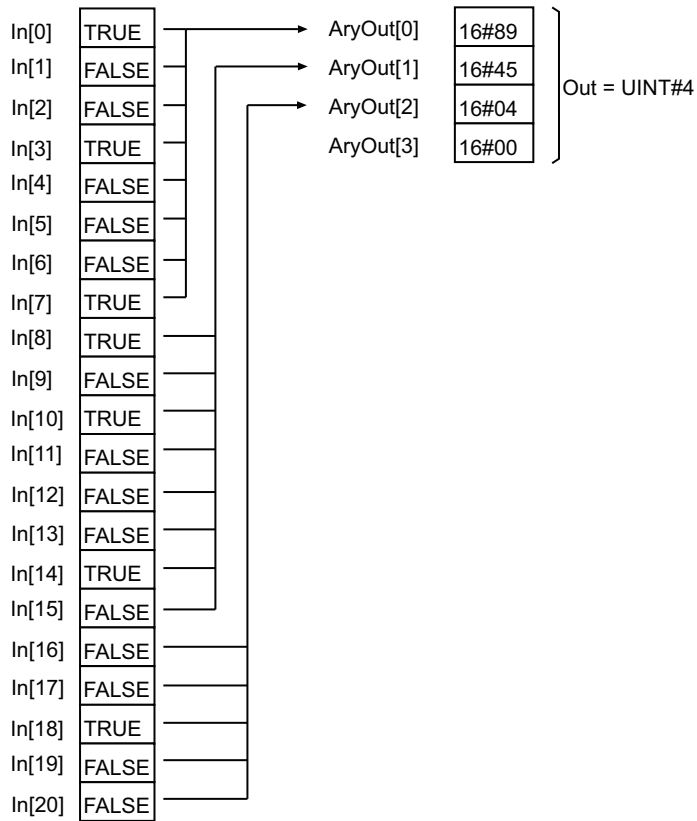


## When *In* Is BOOL Data

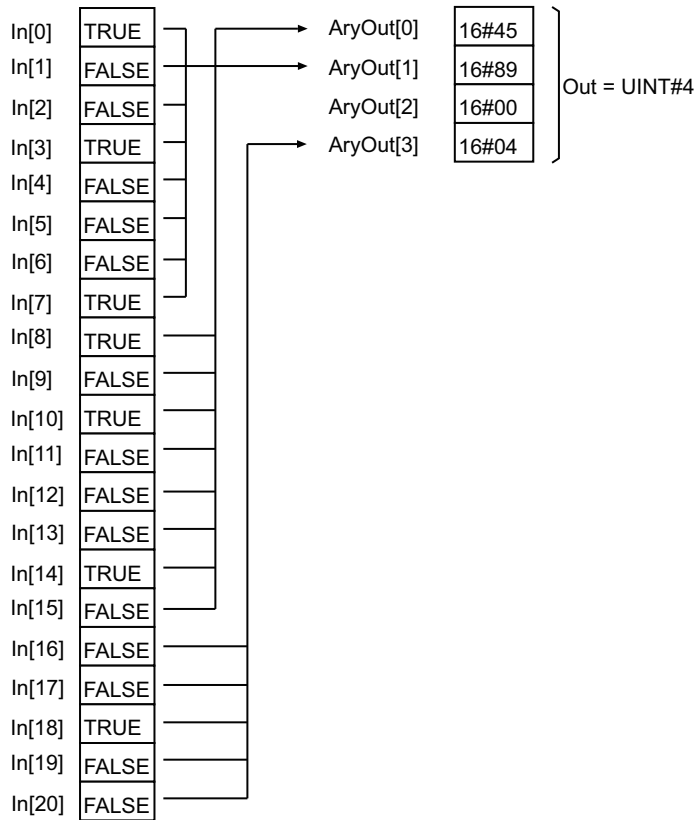
If the data type of *In* is BOOL (one bit), data is stored in AryOut[] as described below.

| Value of Order | In (array or not) | Storage method in AryOut[]  |
|----------------|-------------------|---|
| _LOW_HIGH      | Not an array      | The logical OR of the value of <i>In</i> and 16#00 is stored in AryOut[0].  |
|                | Array             | Values of In[0] to In[7] are joined and stored in AryOut[0].<br>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.<br>If there is not sufficient data in In[] for 8 values, FALSE is added to the most-significant bit.<br>The value of <i>Out</i> is always even. If there are not sufficient bit values, the remaining values will be all FALSE. |
| _HIGH_LOW      | Not an array      | The logical OR of the value of <i>In</i> and 16#00 is stored in AryOut[1].<br>16#00 is stored in AryOut[0].   |
|                | Array             | Values of In[0] to In[7] are joined and stored in AryOut[1].<br>Values of In[8] to In[15] are joined and stored in AryOut[0].<br>The same process is repeated to store the rest of the data.<br>The value of <i>Out</i> is always even. If there are not sufficient bit values, the remaining values will be all 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. A building error will occur if any enumerator is passed 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 will occur in the following cases. *ENO* will be FALSE, and *Out* and *AryOut*[] will not change.
  - a) The value of *Order* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression                       |
|-------------|----------------------------|------------|--------------------|-------------------------------------|
| AryByteTo   | Conversion from Byte Array | FUN        |                    | AryByteTo(In, Size, Order, OutVal); |

## Variables

|              | Meaning                       | I/O    | Description                           | Valid range           | Unit | Default                |
|--------------|-------------------------------|--------|---------------------------------------|-----------------------|------|------------------------|
| In[] (array) | Array to convert              | Input  | Array to convert                      | Depends on data type. | ---  | *1                     |
| Size         | Number of elements to convert |        | Number of elements in In[] to convert |                       |      | 1                      |
| Order        | Conversion order              |        | Conversion order                      |                       |      | _LOW_HIGH or _HIGH_LOW |
| OutVal       | Conversion result             | In-out | Conversion result                     | 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.

|              | Boo lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-506 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| 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 In[] (array to convert) and joins them to match the size of the data type of *OutVal* (conversion result). 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 meanings of the enumerators are as follows:

| Enumerators | Meaning                            |
|-------------|------------------------------------|
| _LOW_HIGH   | Lower byte first, higher byte last |

| Enumerators | Meaning                            |
|-------------|------------------------------------|
| _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 of *In[]* are joined to be equivalent to the data size of *OutVal*, and the joined data is stored in *OutVal*.

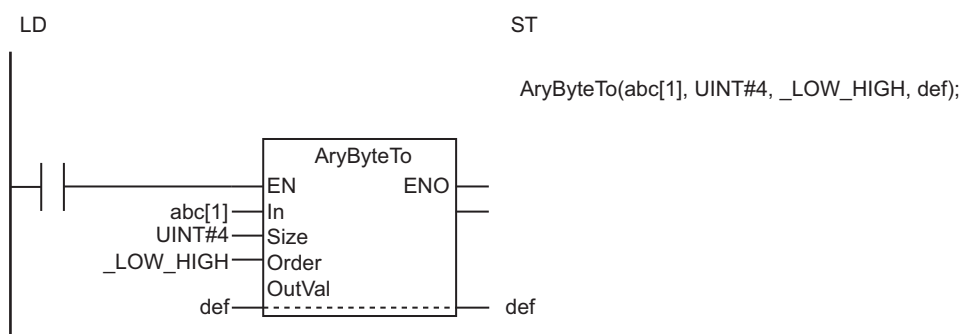
The following data types have two bytes or more.

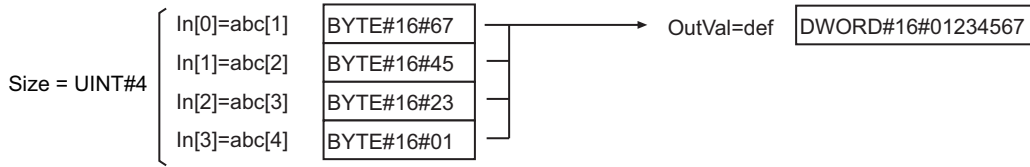
| Classification                            | Data type  |
|---|--|
| Bit strings                               | WORD, DWORD, and LWORD   |
| Integers                                  | UINT, UDINT, ULINT, INT, DINT, and LINT  |
| Real numbers                              | REAL and LREAL   |
| Times, durations, dates, and text strings | TIME, DATE, TOD, DT, and STRING types of two bytes or more   |
| Others                                    | <ul style="list-style-type: none"> <li>An enumeration</li> <li>An array for which the total for all elements is 2 bytes or more</li> <li>An array element that is 2 bytes or more</li> <li>A structure for which the total for all members is 2 bytes or more</li> <li>A structure member that is 2 bytes or more</li> </ul> |

The processing procedure is as follows:

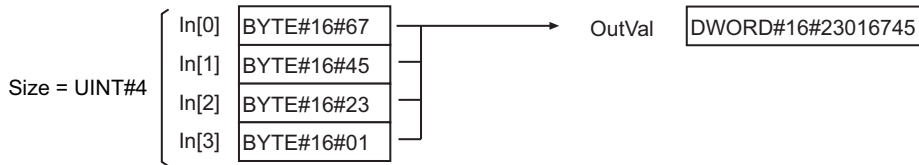
- 1** *In[0]* and *In[1]* are joined according to the value of *Order* to create one word (two bytes) of data. If *Order* is *\_LOW\_HIGH*, the upper byte is stored in *In[1]* and the lower byte is stored in *In[0]*. If *Order* is *\_HIGH\_LOW*, the upper byte is stored in *In[0]* and the lower byte 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 *OutVal*. For example, if *OutVal* is *DWORD* data, four words of data are joined.
- 4** The joined data is stored in *OutVal*.

The following shows an example where *OutVal* is *DWORD* data, *Size* is *UINT#4*, and *Order* is *\_LOW\_HIGH*.





The following shows an example where *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*.

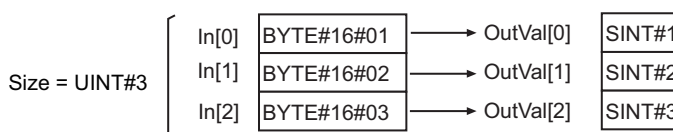
The following data types have one byte.

| 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                                    | <ul style="list-style-type: none"> <li>An array for which the total for all elements is 1 byte</li> <li>An array element that is 1 byte</li> <li>A structure for which the total for all members is 1 byte</li> <li>A structure member that is 1 byte</li> </ul> |

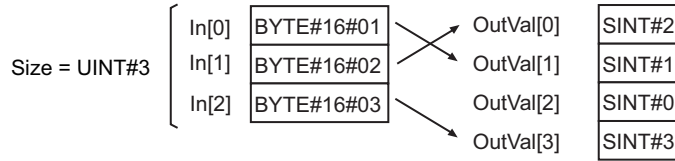
The following storage method is used.

| Value of <i>Order</i> | <i>OutVal</i> (array or not) | Storage method in <i>OutVal</i>   |
|-----------------------|------------------------------|---|
| _LOW_HIGH             | Not an array                 | Value of <i>In</i> [0] is stored in <i>OutVal</i>   |
|                       | Array                        | Value of <i>In</i> [ <i>i</i> ] is stored in <i>OutVal</i> [ <i>i</i> ]   |
| _HIGH_LOW             | Not an array                 | Value of <i>In</i> [1] is stored in <i>OutVal</i>   |
|                       | Array                        | <i>In</i> [ <i>i</i> ] (where <i>i</i> is even) is stored in <i>OutVal</i> [ <i>i</i> +1].<br><i>In</i> [ <i>i</i> ] (where <i>i</i> is odd) is stored in <i>OutVal</i> [ <i>i</i> -1].<br>If the value of <i>Size</i> is odd, data is stored up to <i>OutVal</i> [ <i>Size</i> ] and 16#00 is stored in <i>OutVal</i> [ <i>Size</i> -1]. |

The following shows an example where *OutVal* is a SINT array with three elements, *Size* is *UINT#3*, and *Order* is *\_LOW\_HIGH*.



The following shows an example where *OutVal* and *Size* are the same as above and *Order* is *\_HIGH\_LOW*.

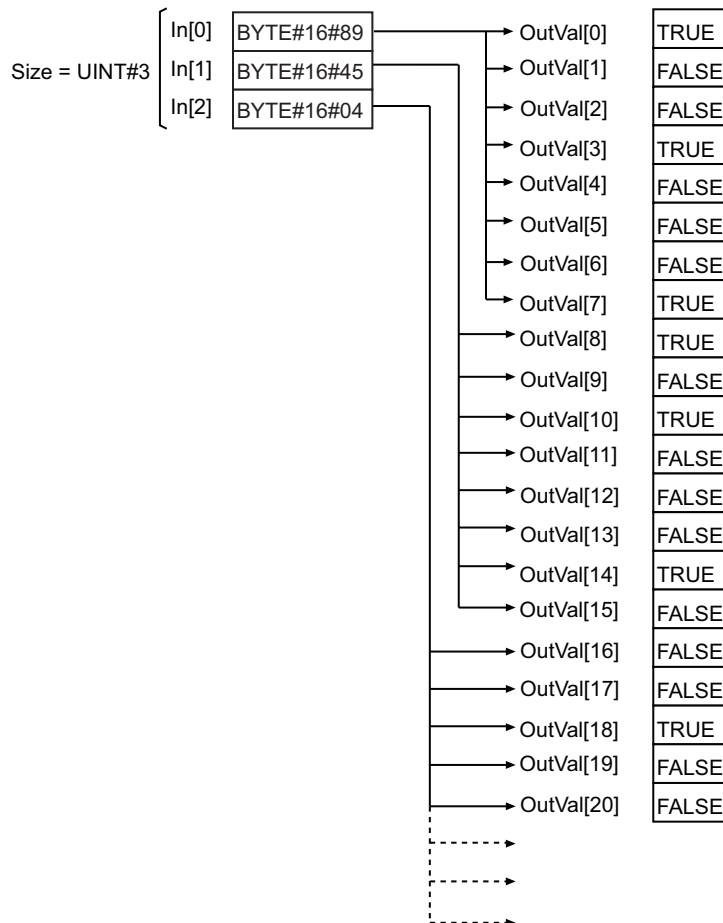


## When *OutVal* Is BOOL Data

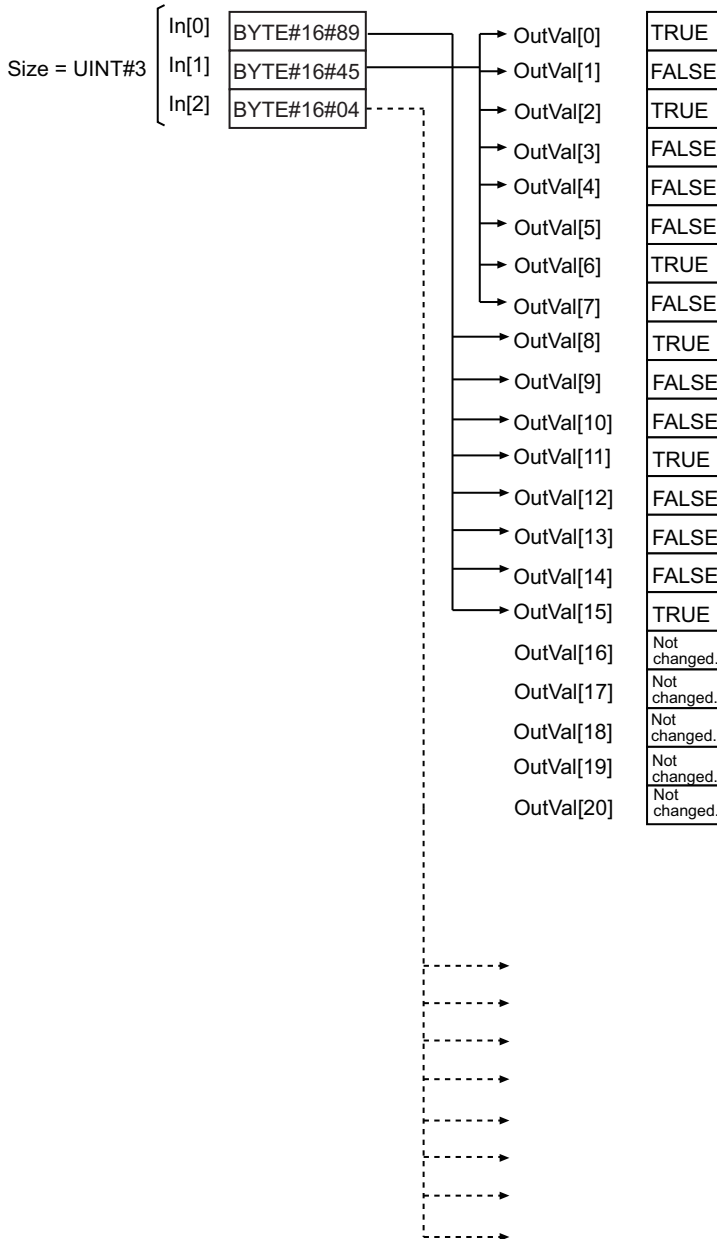
If the data type of *OutVal* is BOOL (one bit), data is stored in *OutVal* as described below.

| Value of Order | <i>OutVal</i> (array or not) | Storage method in <i>OutVal</i>  |
|----------------|------------------------------|--|
| _LOW_HIGH      | Not an array                 | Value of bit 0 of In[0] is stored in <i>OutVal</i> .   |
|                | 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 <i>OutVal</i> .   |
|                | 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*.



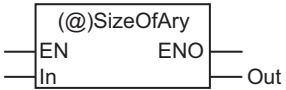
### Precautions for Correct Use

- If *OutVal* is a structure, some of the values of *In[]* may be inserted in adjustment areas between members depending on the composition.
- If *Size* is less than the data size of *OutVal*, an error does not occur, and the specified byte data is stored in *OutVal*. If the byte data is insufficient, the values before the instruction was executed will be held.  
If the size is smaller than the previous execution, use the instruction after the variables are cleared with the Clear (Initialize) instruction in advance.
- If the value of *Size* is 0, the value of *Out* will change to TRUE and *OutVal* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *OutVal* will not change.

- a) The value of *Order* is outside the valid range.
- b) 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/<br>FUN | Graphic expression  | ST expression       |
|-------------|------------------------------|------------|---|---------------------|
| SizeOfAry   | Get Number of Array Elements | FUN        |  | Out:=SizeOfAry(In); |

## Variables

|              | Meaning            | I/O    | Description        | Valid range           | Unit | Default |
|--------------|--------------------|--------|--------------------|-----------------------|------|---------|
| In[] (array) | Array              | Input  | Array              | Depends on data type. | ---  | *1      |
| Out          | Number of elements | Output | Number of elements | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

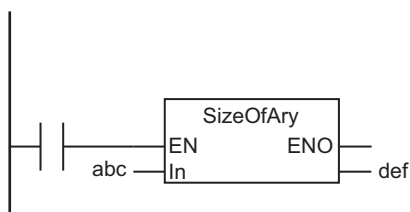
The SizeOfAry instruction gets the number of elements in array In[].

For the input parameter, use an array name, such as *array*, instead of an array element name, such as *array[0]*.

The following figure shows a programming example.

| Name | Data Type          |
|------|--------------------|
| abc  | ARRAY[0..3] OF INT |

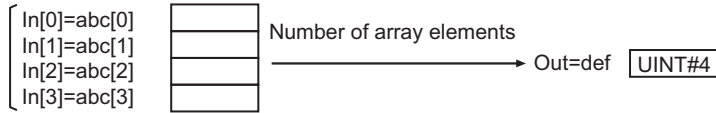
LD



ST

```
def:=SizeOfAry(abc);
```

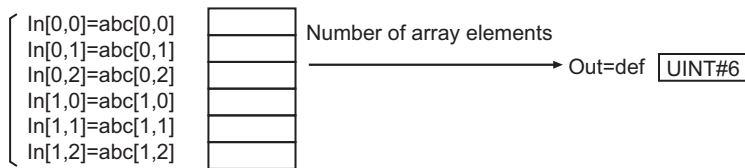
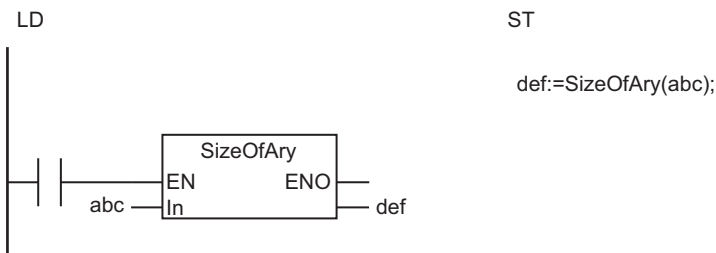




### Additional Information

In[] can be an array with two or more dimensions. In that case, Out will contain all the elements 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.

| Name | Data Type                |
|------|--------------------------|
| abc  | ARRAY[0..1,0..2] OF BOOL |



# PackWord

The PackWord instruction joins two 1-byte data into a 2-byte data.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|-------------|------------|--------------------|---------------------------|
| PackWord    | 2-byte Join | FUN        |                    | 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

|      | 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. | ---  | ---     |

|      | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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

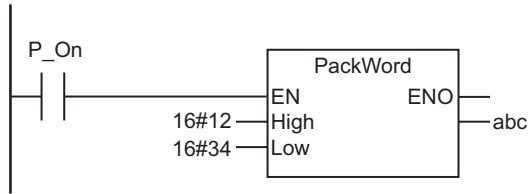
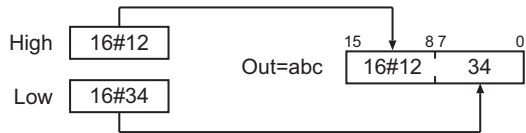
The PackWord instruction joins two 1-byte data into a 2-byte data.

The data specified in *High* is stored in bits 15 to 8, and the data specified in *Low* is stored in bits 7 to 0.

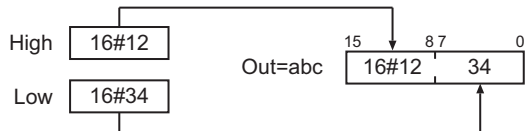
## LD

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.

**ST**

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.



```
abc:=PackWord(16#12, 16#34);
```

# PackDword

The PackDword instruction joins four 1-byte data into a 4-byte data.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression                                       |
|-------------|-------------|------------|--------------------|---|
| PackDword   | 4-byte Join | FUN        |                    | 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

|          | 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. | ---  | ---     |

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

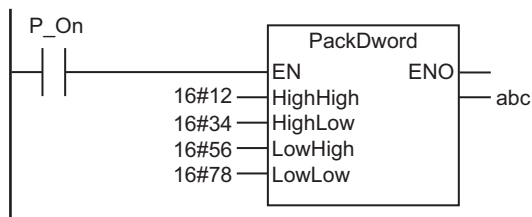
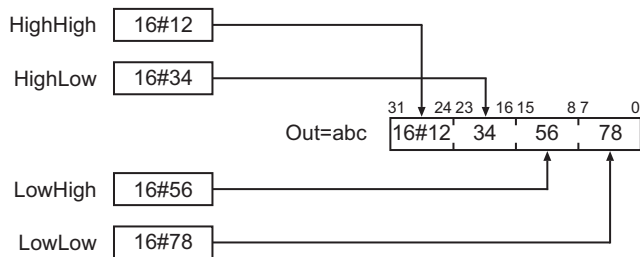
The PackDword instruction joins four 1-byte data into a 4-byte data.

The data specified with *HighHigh* is stored in bits 31 to 24, the data specified with *HighLow* in bits 23 to 16, the data specified with *LowHigh* in bits 15 to 8, and the data specified with *LowLow* in bits 7 to 0.

## LD

The following example shows the instruction when *HighHigh* is 16#12, *HighLow* is 16#34, *LowHigh* is 16#56, and *LowLow* is 16#78.

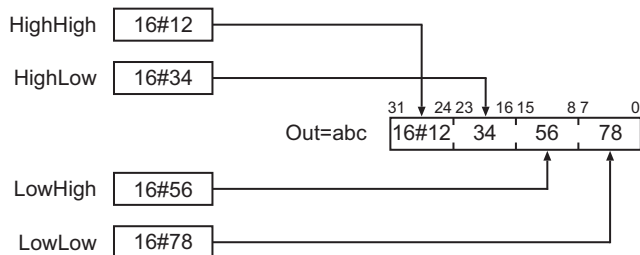
The value of variable *abc* will be 16#12345678.



## ST

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.



```
abc:=PackDword(16#12, 16#34, 16#56, 16#78);
```

# LOWER\_BOUND and UPPER\_BOUND

LOWER\_BOUND : Gets the first number of a specified array dimension.

UPPER\_BOUND : Gets the last number of a specified array dimension.

| Instruction | Name                      | FB/<br>FUN | Graphic expression | ST expression               |
|-------------|---------------------------|------------|--------------------|-----------------------------|
| LOWER_BOUND | Get First Number of Array | FUN        |                    | Out:=LOWER_BOUND(ARR, DIM); |
| UPPER_BOUND | Get Last Number of Array  | FUN        |                    | 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 these instructions.

## Variables

|     | 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 an array dimension. *1 | ---                   | ---  | ---     |
| DIM | Dimension        |        | Specifies the dimension. *2   | ---                   | ---  | 1       |
| Out | Return value     | Output | LOWER_BOUND: First number<br>UPPER_BOUND: Last number   | Depends on data type. | ---  | ---     |

\*1. Use an array name, such as *array*, instead of an array element name, such as *array[0]*.

\*2. For the first dimension of the array, specify 1.

|   | Boo lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real number |      | Times, durations, 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. |          |             |      |       |       |          |      |       |       |      |     |      |             |      |   |      |      |     |    |        |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>number |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------|-------|--|------|-----|----|--------|
|     | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL           | LREAL | TIME   | DATE | TOD | DT | STRING |
| 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 set to FALSE.<br>FALSE: Set to FALSE by the user program. |

## Precautions for Correct Use

An error occurs 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 number of dimensions that *ARR* has.

## 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 variable | Name | Data Type | Default | Comment |
|-------------------|------|-----------|---------|---------|
|                   | i    | DINT      |         |         |

| Input/output variables | Name | I/O    | Data Type       | Comment |
|------------------------|------|--------|-----------------|---------|
|                        | EN   | Input  | BOOL            |         |
|                        | ENO  | Output | BOOL            |         |
|                        | a    | In-out | ARRAY[*] OF INT |         |

| Return value | Name | Data Type | Default | Comment |
|--------------|------|-----------|---------|---------|
|              | Sum  | INT       |         |         |

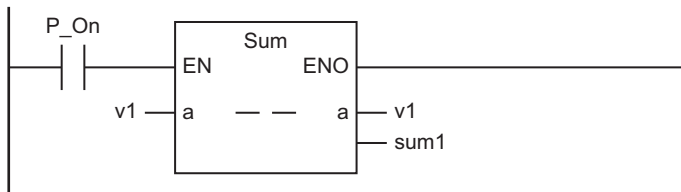
```

Sum := 0;
FOR i := LOWER_BOUND(a,1) TO UPPER_BOUND(a,1) DO
    Sum := Sum + a[i];
END_FOR;
    
```

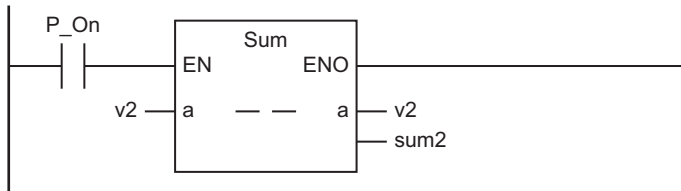
● Calling Program

| Internal variables | Name | Data Type          | Default                | 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 2x2 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 | Name | Data type | Default | Comment |
|--------------------|------|-----------|---------|---------|
|                    | i    | DINT      |         |         |
|                    | j    | DINT      |         |         |
|                    | m1   | DINT      |         |         |
|                    | m2   | DINT      |         |         |
|                    | n1   | DINT      |         |         |
|                    | n2   | DINT      |         |         |



| Input/<br>output variables | Name | 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 | Name       | Data type | Default | 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 | Name | Data type                | Default      | 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

| Instruction             | Name  | Page       |
|-------------------------|---|------------|
| StackPush               | Push onto Stack                             | page 2-524 |
| StackFIFO and StackLIFO | First In First Out/Last In First Out        | page 2-533 |
| StackIns                | Insert into Stack                           | page 2-536 |
| StackDel                | Delete from Stack                           | page 2-539 |
| RecSearch               | Record Search                               | page 2-541 |
| RecRangeSearch          | Range Record Search                         | page 2-546 |
| RecSort                 | Record Sort                                 | page 2-551 |
| RecNum                  | Get Number of Records                       | page 2-557 |
| RecMax and RecMin       | Maximum Record Search/Minimum Record Search | page 2-560 |

# StackPush

The StackPush instruction stores a value in the top of a stack.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression                    |
|-------------|-----------------|------------|--------------------|----------------------------------|
| StackPush   | Push onto Stack | FUN        |                    | StackPush(In, InOut, Size, Num); |

## Variables

|                 | 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             | ---  | ---     |

|                 | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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 <i>In</i> . |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |
| Num             |  |             |      |       |       | OK       |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |
| Out             | OK   |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |

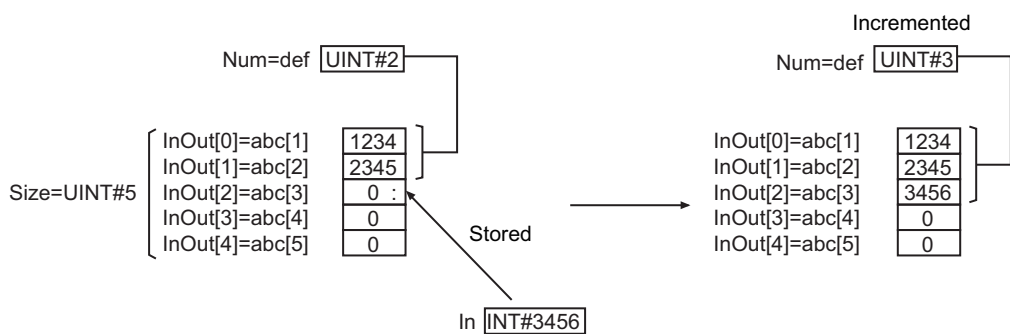
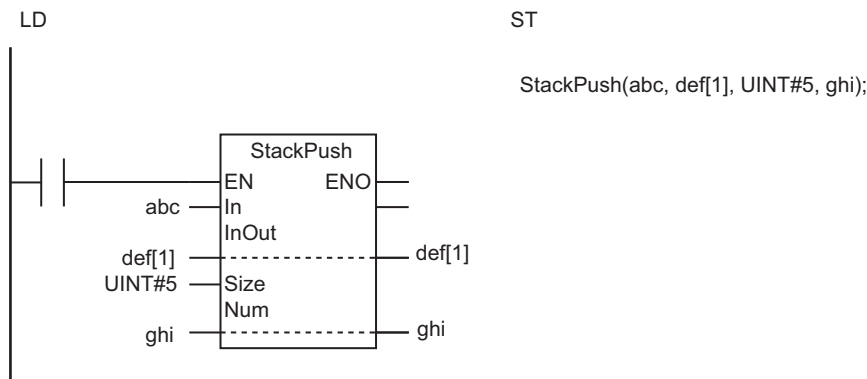
## Function

The StackPush instruction assumes that there are *Num* elements stored in stack array InOut[]. *In* (input value) is written to the next element, InOut[*Num*].

And then, *Num* is incremented.

For *Size* (the number of stack elements), specify the number of InOut[] elements to be used for the stack.

The following shows an example where *Size* is UINT#5 and *Num* is UINT#2.



## Additional Information

Use the instruction, *StackFIFO* and *StackLIFO* on page 2-533, to remove the bottom or top value that was stored in the stack.

## Precautions for Correct Use

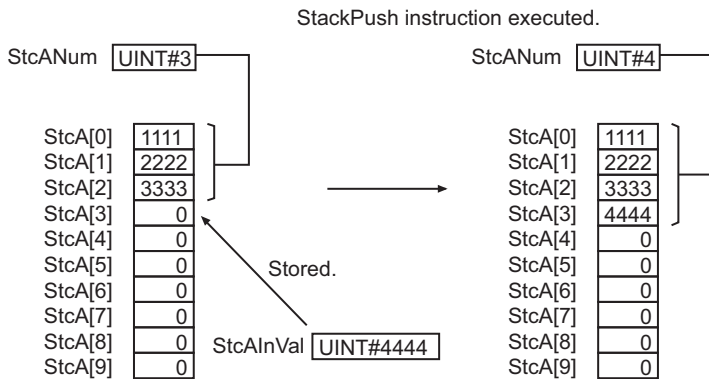
- Use the same data type for *In* and the elements of *InOut[]*. If they are different, a building error will occur.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]* and *Num* do not change if the value of *Size* is 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. A building error will occur if any enumerator is passed directly.
- Return value *Out* is not used when the instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - a) The value of *Size* is not 0, and *Num* is greater than or equal to *Size*.
  - b) The value of *Size* exceeds the array area of *InOut[]*.
  - c) *In* and *InOut[]* are STRING data and the number of bytes in *In* exceeds the size of *InOut[]*.

## Sample Programming

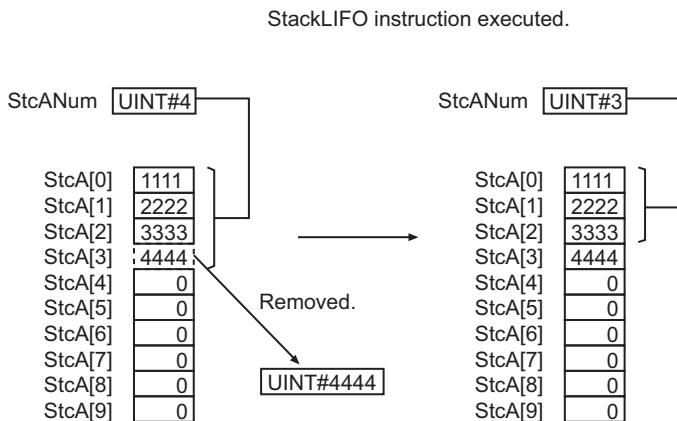
The array variable StcA[0..9] is used as a stack. As preparations, three values (UINT#1111, UINT#2222, and UINT#3333) are stored in the stack.

|         |      |
|---------|------|
| StcA[0] | 1111 |
| StcA[1] | 2222 |
| StcA[2] | 3333 |
| StcA[3] | 0    |
| StcA[4] | 0    |
| StcA[5] | 0    |
| StcA[6] | 0    |
| StcA[7] | 0    |
| StcA[8] | 0    |
| StcA[9] | 0    |

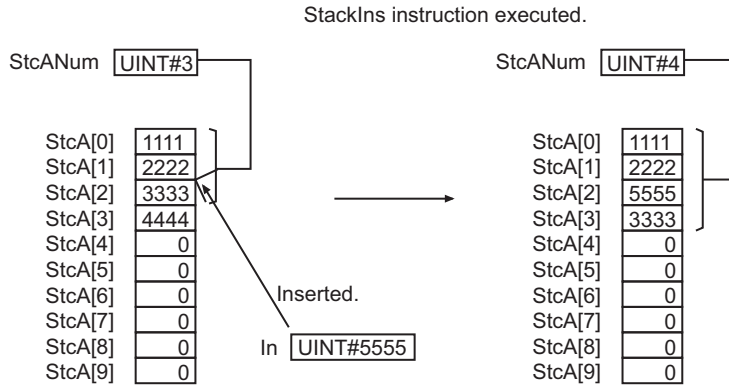
The StackPush instruction is used to store a new value (UINT#4444) at the top of the stack StcA[3]. That means there will be four values in the stack.



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.



And last, 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                                  |

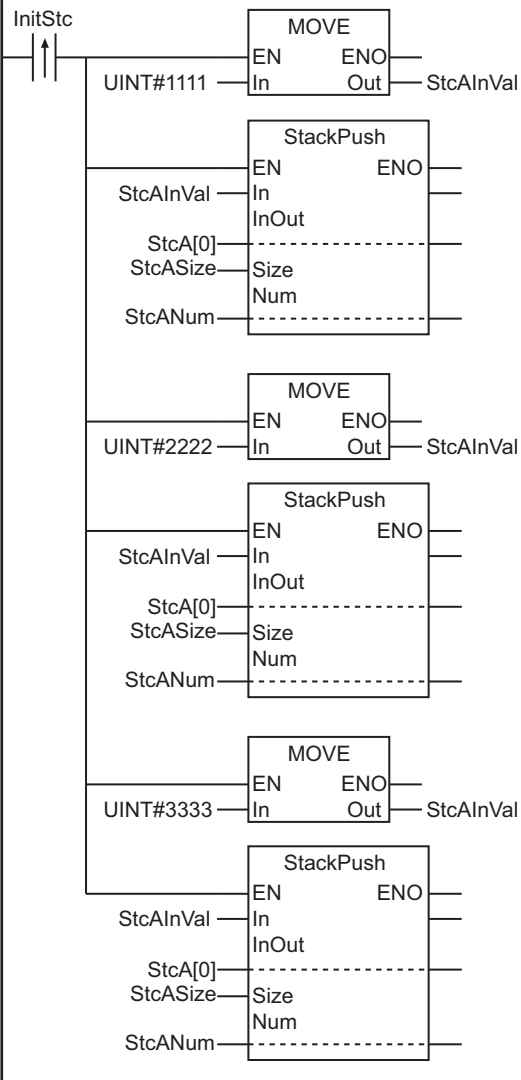
Initialize stack.



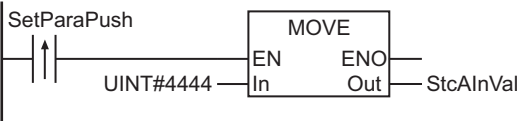
Inline ST

**Note:** The contents of the inline ST are given below.

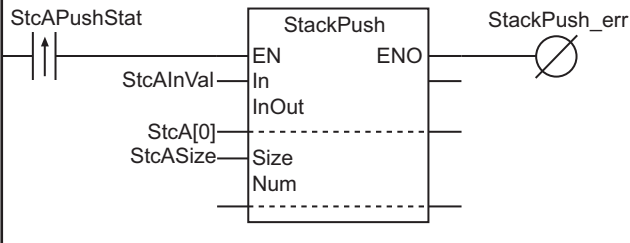
Store three values in stack.



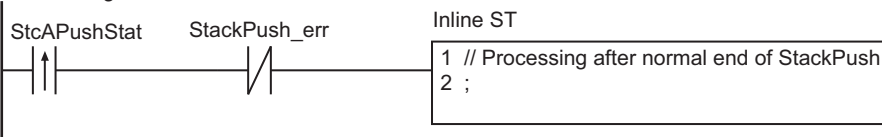
Set the value to add with StackPush.



Add data with StackPush instruction.

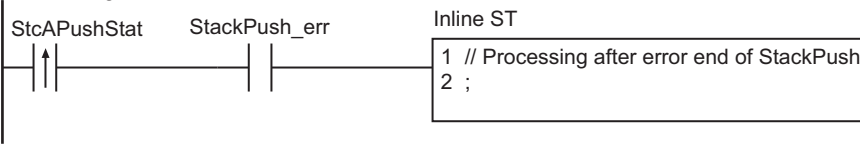


Processing after normal end of StackPush

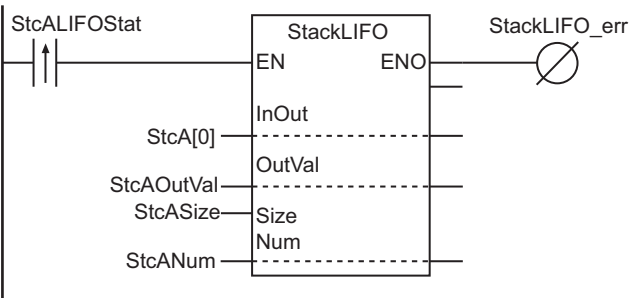




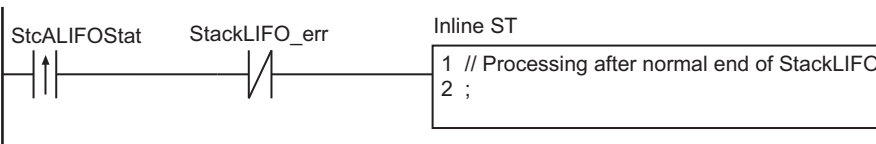
Processing after error end of StackPush



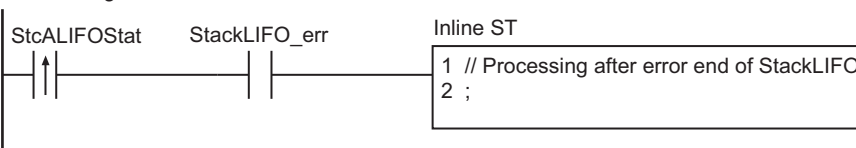
Remove data with StackLIFO instruction.



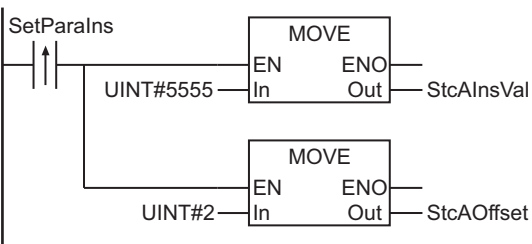
Processing after normal end of StackLIFO



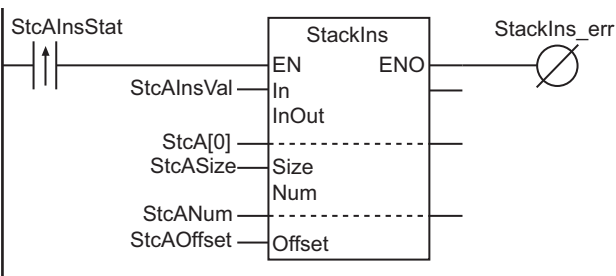
Processing after error end of StackLIFO

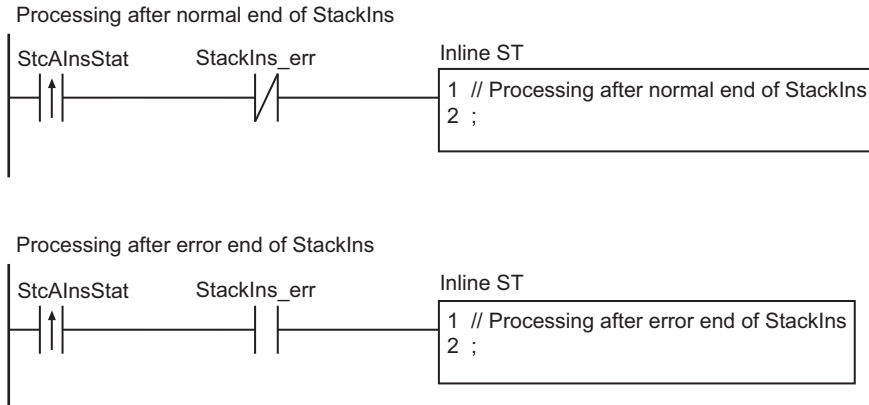


Set the insert value and offset with StackInsh.



Insert data with StackIns instruction.





### ● 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 |
| StcAlnVal         | 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         |
| StcAlnsStat       | BOOL                | FALSE         | StackIns execution condition                    |
| preStcAlnsStat    | BOOL                | FALSE         | Value of StcAlnsStat from previous task period  |
| StcAlnsVal        | UINT                | 0             | Value inserted by StackIns                      |
| StcAOffset        | UINT                | 0             | Offset for StackIns                             |
| StcAlns_OK        | BOOL                | FALSE         | StackIns normal end flag                        |
| StcAlnsNormalEnd  | BOOL                | FALSE         | Processing after normal end of StackIns         |
| StcAlnsErrorEnd   | 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/<br>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

|                 | 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               | In-out | Array that functions as stack        | Depends on data type. | ---  | ---     |
| OutVal          | Output value              |        | Value or structure output from stack |                       |      |         |
| Num             | Number of stored elements |        | Number of elements stored in stack   |                       |      |         |
| Out             | Return value              | Output | Always TRUE                          | TRUE only             | ---  | ---     |

|                 | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     |  |
| OutVal          | Arrays of enumerations or structures can also be specified. |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
| Num             | Must be the same data type as the elements of InOut[].      |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
|                 |   |             |      |       |       |          | OK   |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 instruction assumes that there are *Num* elements stored in stack array *InOut*[]. The instruction removes a value from the stack and assigns it to output value *OutVal*.

For *Size* as the number of stack elements, specify the number of elements in *InOut*[] as a stack.

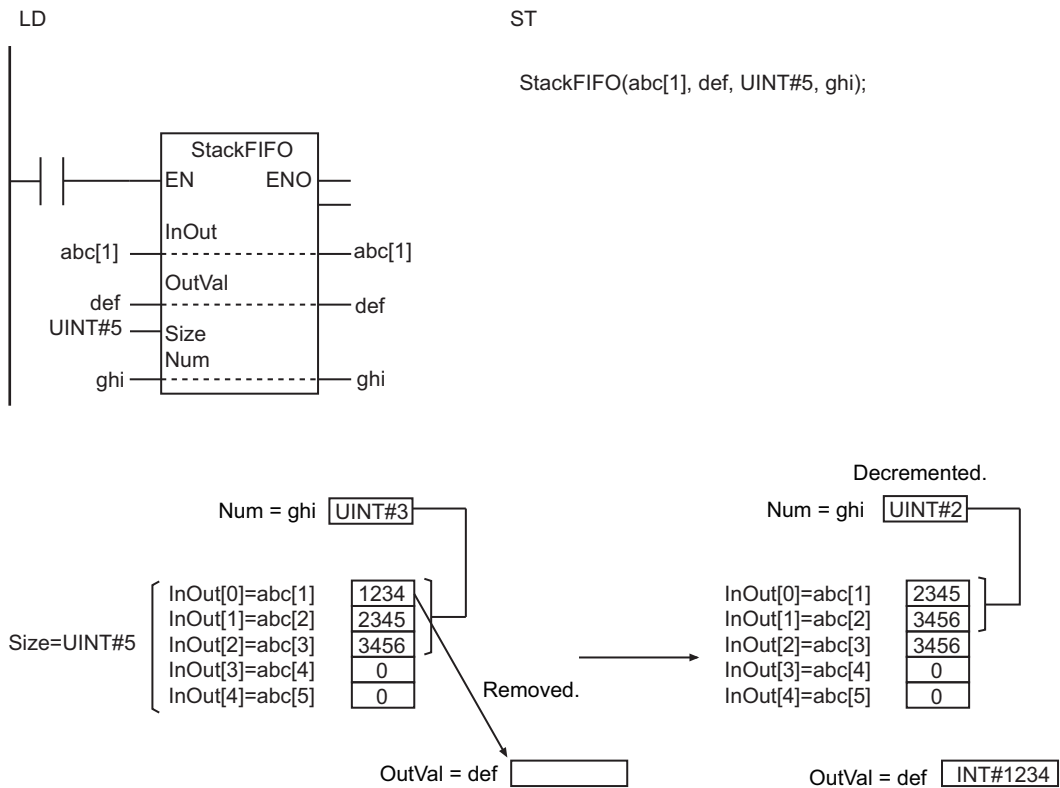
## StackFIFO

The StackFIFO instruction retrieves the value stored at the bottom of the stack. The value of *InOut*[0] is assigned to *OutVal*.

And then, each of *Num*-1 elements, which begins with *InOut*[1], is shifted to the next lower element in the stack array.

And last, *Num* is decremented.

The following shows an example where *Size* is *UINT*#5 and *Num* is *UINT*#3.

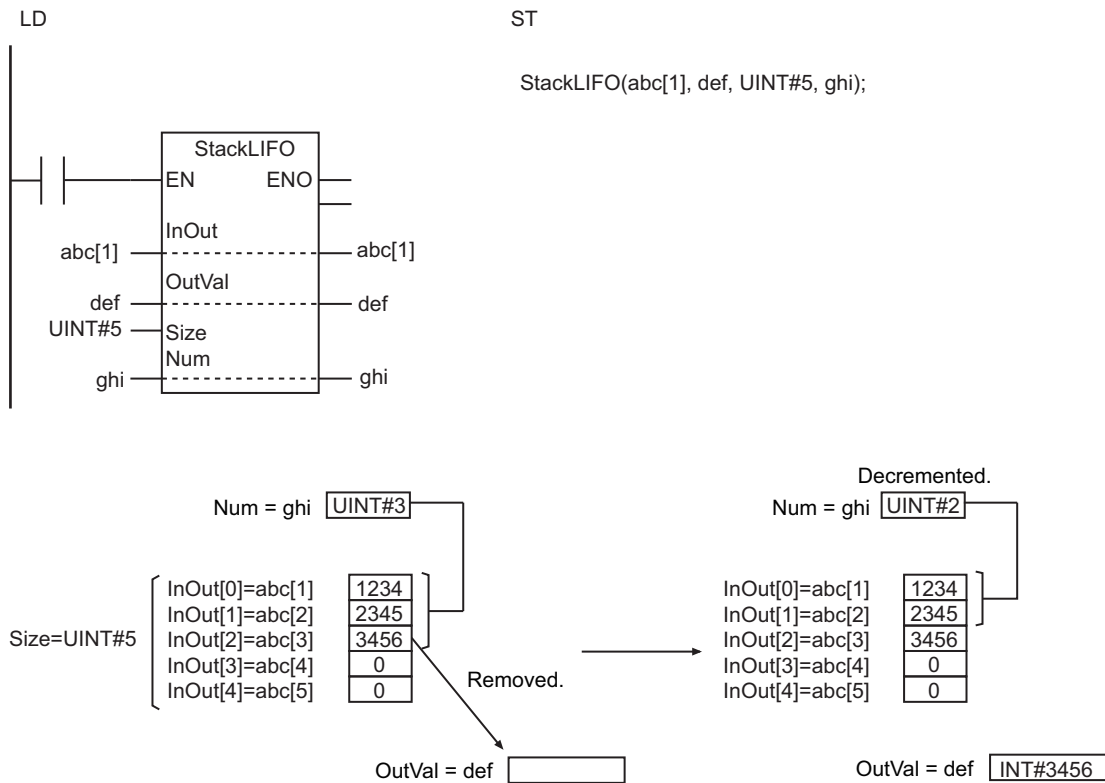


## StackLIFO

The StackLIFO instruction retrieves the value stored at the top of the stack. The value of  $InOut[Num-1]$  is assigned to  $OutVal$ .

And then,  $Num$  is decremented.

The following shows an example where  $Size$  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 will occur in the following cases.  $ENO$  will be FALSE, and  $OutVal$  will not change.
  - a) The values of  $Num$  and  $Size$  are not 0, and  $Num$  is greater than  $Size$ .
  - b) The value of  $Size$  exceeds the array area of  $InOut[]$ .
  - c)  $InOut[]$  is a STRING array and any of the elements does not end in a NULL character.
  - d)  $InOut[]$  is a STRING array and the number of bytes in the elements exceeds the size of  $OutVal$ .

## Sample Programming

Refer to *Sample Programming* on page 2-526 for the StackPush instruction.

# StackIns

The StackIns instruction inserts a value at a specified position in a stack.

| Instruction | Name              | FB/<br>FUN | Graphic expression | ST expression                           |
|-------------|-------------------|------------|--------------------|---|
| StackIns    | Insert into Stack | FUN        |                    | StackIns(In, InOut, Size, Num, Offset); |

## Variables

|                 | 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. | ---  | *1      |
| Size            | Number of stack elements  |        | Number of stack array elements                                 |                       |      | 1       |
| Offset          | Offset                    |        | Position in stack at which to insert <i>In</i>                 |                       |      | 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             | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                 | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>In</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Num             |  |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out             | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |



## Function

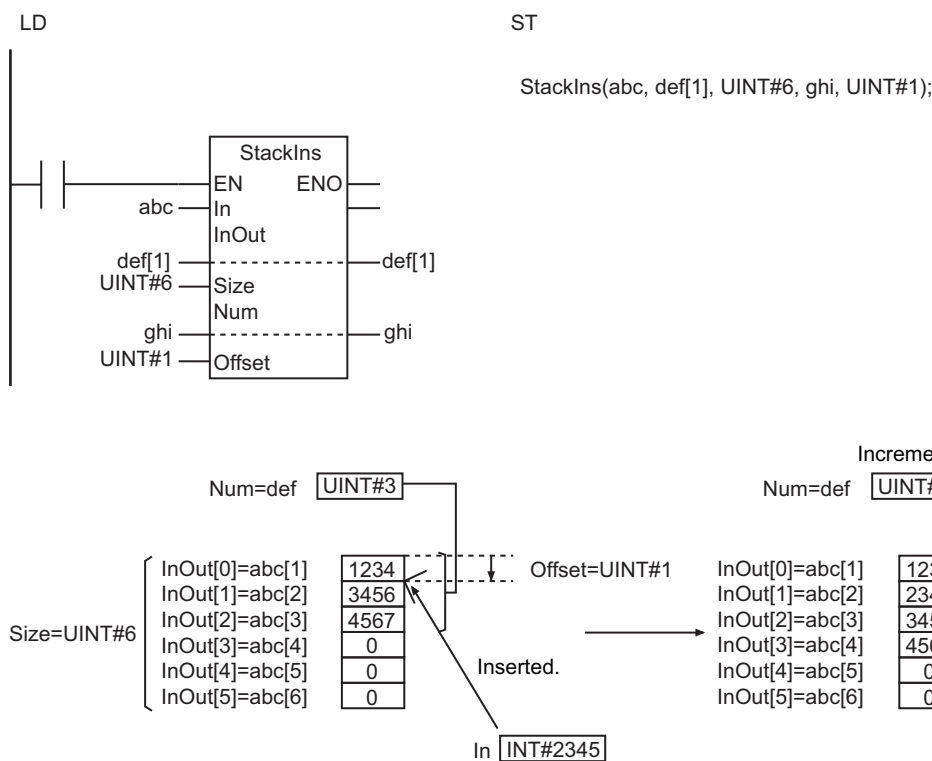
The StackIns instruction assumes that there are *Num* elements stored in stack array InOut[]. *In* (insert value) is inserted at the position of InOut[*Offset*], which is specified by *Offset*.

Each of all the higher elements, i.e., InOut[*Offset*] to InOut[*Num*-1], is moved to the next higher element in the stack array.

And then, *Num* is incremented.

For *Size* (the number of stack elements), specify the number of InOut[] elements to be used for the stack.

The following shows an example where *Size* is UINT#6, *Num* is UINT#3 and *Offset* is 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. A building error will occur if any enumerator is passed directly.
- Return value *Out* is not used when the instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and InOut[] will not change.
  - a) The value of *Size* is not 0, and *Size* is not greater than *Num*, which is not greater than or equal to *Offset*.
  - b) The value of *Size* exceeds the array area of InOut[].
  - c) *In* and InOut[] are STRING data and the number of bytes in *In* exceeds the size of InOut[].

## Sample Programming

Refer to *Sample Programming* on page 2-526 for the StackPush instruction.

# StackDel

The StackDel instruction deletes a value from a specified position in a stack.

| Instruction | Name              | FB/<br>FUN | Graphic expression | ST expression                       |
|-------------|-------------------|------------|--------------------|-------------------------------------|
| StackDel    | Delete from Stack | FUN        |                    | StackDel(InOut, Size, Num, Offset); |

## Variables

|                 | 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             | ---  | ---     |

|                 | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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

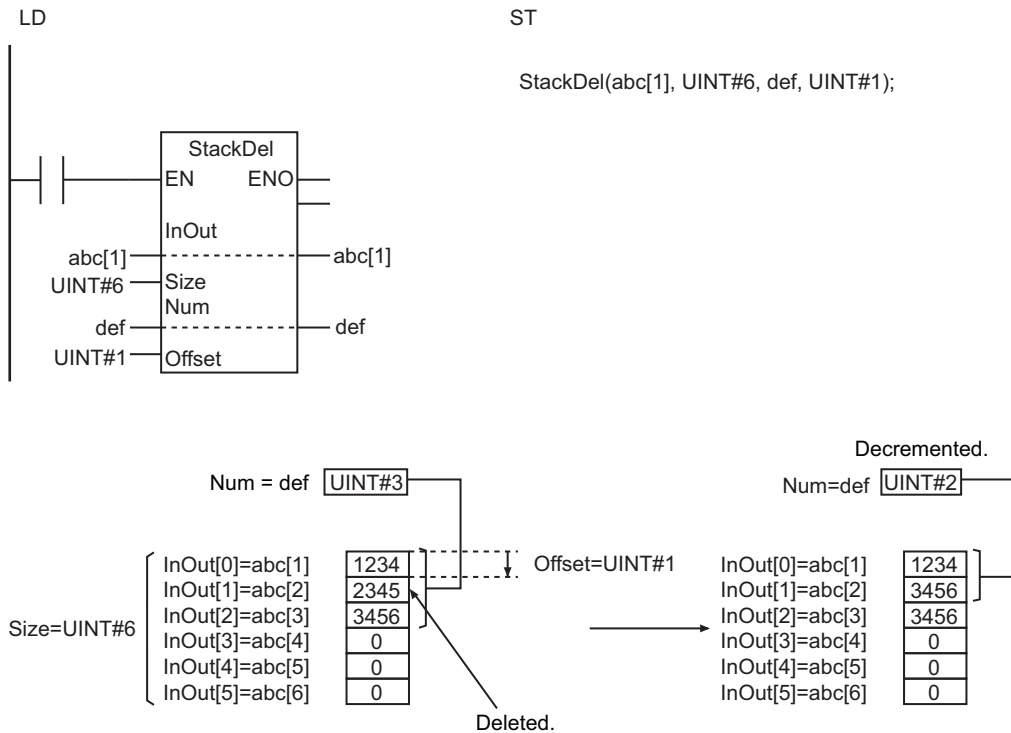
The StackDel instruction assumes that there are *Num* elements stored in stack array InOut[]. It deletes the value at the position of InOut[*Offset*], which is specified by *Offset*.

Each of all the higher elements, i.e., InOut[*Offset*+1] to InOut[*Num*-1], is moved to the next lower element in the stack array.

And then, *Num* is decremented.

For *Size* (the number of stack elements), specify the number of *InOut*[] elements to be used for the stack.

The following shows an example where *Size* is *UINT#6*, *Num* is *UINT#3* and *Offset* is *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 will occur in the following cases. *ENO* will be FALSE, and *InOut*[] will not change.
  - a) The values of *Num* and *Size* are not 0, and *Size* is not greater than or equal to *Num*, which is not greater than *Offset*.
  - b) 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|---------------|------------|--|---|
| RecSearch   | Record Search | FUN        | <pre> graph LR     subgraph RecSearch         EN[EN]         In[In]         Size[Size]         Member[Member]         Key[Key]         Mode[Mode]         InOutPos[InOutPos]         ENO[ENO]         Num[Num]         Out[Out]     end     EN --- ENO     In --- Out     Size --- Out     Member --- Out     Key --- Out     Mode --- Out     InOutPos --- Out     </pre> | Out:=RecSearch(In, Size, Member, Key, Mode, InOutPos, Num); |

## Variables

|                    | Meaning                              | I/O    | Description  | Valid range           | Unit | Default                      |
|--------------------|--------------------------------------|--------|--|-----------------------|------|------------------------------|
| In[] (array)       | Array to search                      | Input  | Array of structures to search  | ---                   | ---  | *1                           |
| 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   |                       |      | *1                           |
| Key                | Search key                           |        | Search value   |                       |      |                              |
| Mode               | Search method                        |        |  | Search method         |      | _LINEAR, _BIN_ASC, _BIN_DESC |
| 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<br>FALSE: There are no elements that match conditions | Depends on data type. | ---  | ---                          |
| Num                | Number of matches                    |        | Number of matches  |                       |      |                              |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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) |             |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Size         |             |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |

|                       | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
|                       | Specify the same data type as the search member of In[]  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |
| Key                   | Must be the same data type as <i>Member</i> .  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |
| Mode                  | Refer to <i>Function</i> on page 2-542 for the enumerators of the enumerated type <code>_eSEARCH_MODE</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |
| InOutPos[]<br>(array) |  |             |      |       |       | OK       |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |
| Out                   | OK   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |
| Num                   |  |             |      |       |       | OK       |      |       |       |      |     |      |      |                      |       |  |          |          |          |          |

\*1. 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[], i.e., from In[0] to In[Size-1], for *Member* (member to search) that matches the search key *Key*.

The member to search of an element in In[] is passed to *Member* as an argument.

If any matching element is 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.

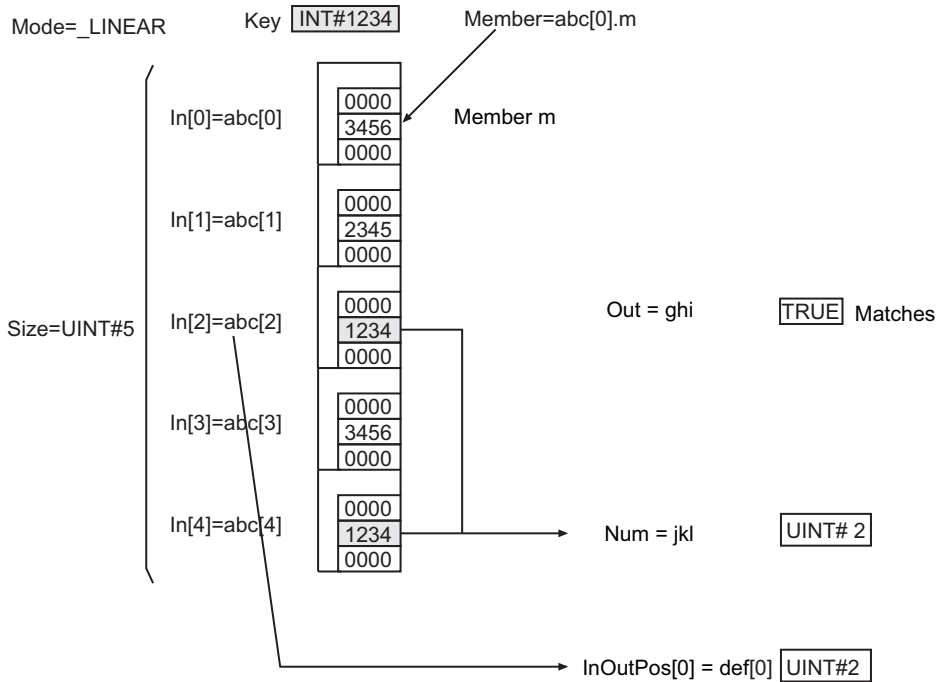
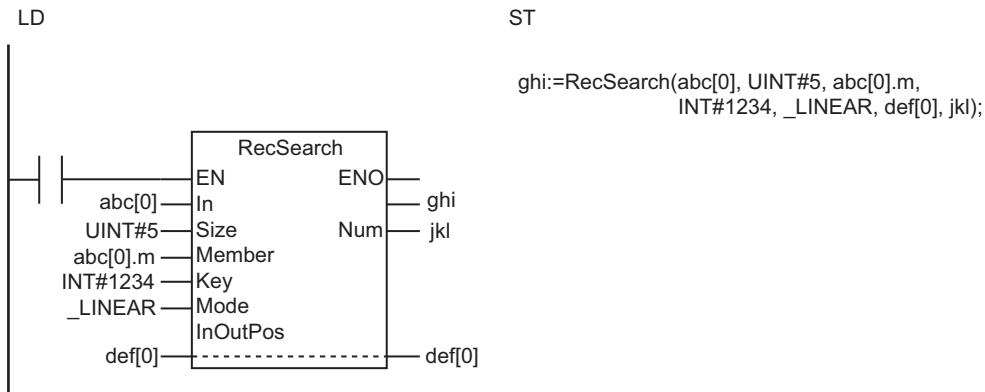
Always attach the element number to input parameter that is passed to In[], e.g., array[3].

The data type of search method *Mode* is enumerated type `_eSEARCH_MODE`. The meanings of the enumerators are as follows:

| Enumerator             | Meaning                  |
|------------------------|--------------------------|
| <code>_LINEAR</code>   | Linear search            |
| <code>_BIN_ASC</code>  | Ascending binary search  |
| <code>_BIN_DESC</code> | Descending binary search |

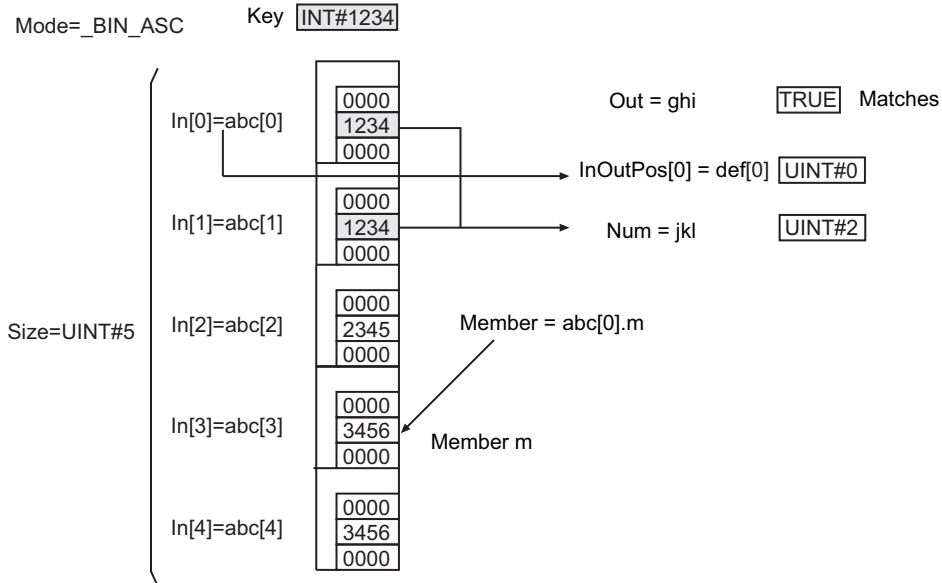
For a linear search, the search is performed in order from the first element of In[].

The following shows an example where *Size* is `UINT#5`, *Key* is `INT#1234` and *Mode* is `_LINEAR`.



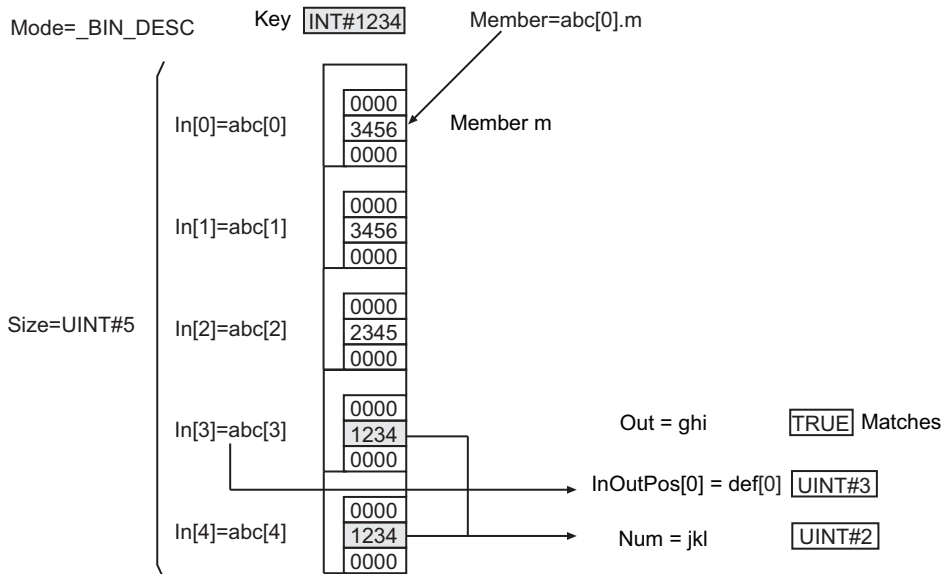
For an ascending binary search, the array elements in the input parameter that is passed to In[] must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

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 accuracy of *Member* and *Key* to the same. You can use the following instructions for the adjustment: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

## 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, expected results may not be obtained due to error, depending on the value.
- If *Key* is a real number, do not specify a non-numeric value for *Key*.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. `InOutPos[]` will not change.
- A correct result will not be obtained if the value of *Mode* is `_BIN_ASC` or `_BIN_DESC` and the elements of `In[]` are not in ascending or descending order. Sort the elements in ascending or descending order before executing this instruction.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out*, `InOutPos[]`, and *Num* will not change.
  - a) The value of *Mode* is outside the valid range.
  - b) The value of *Size* exceeds the array area of `In[]`.
  - c) *Member* is not a member of `In[]`.
  - d) The array size of `InOutPos[]` is smaller than the number of dimensions of `In[]`.
  - e) *Member* is STRING data and it does not end with 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/<br>FUN | Graphic expression | ST expression  |
|---------------------|------------------------|------------|--------------------|--|
| RecRange-<br>Search | Range Record<br>Search | FUN        |                    | Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num); |

## Variables

|                    | Meaning                              | I/O    | Description  | Valid range                  | Unit | Default                              |          |
|--------------------|--------------------------------------|--------|--|------------------------------|------|--------------------------------------|----------|
| In[] (array)       | Array to search                      | Input  | Array of structures to search  | ---                          | ---  | *1                                   |          |
| 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   |                              |      | *1                                   |          |
| MN                 | Search condition lower limit         |        | Search condition lower limit   |                              |      | _EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH | _EQ_BOTH |
| MX                 | Search condition upper limit         |        | Search condition upper limit   | _LINEAR, _BIN_ASC, _BIN_DESC |      |                                      |          |
| Condition          | Search condition                     |        | Search condition   |                              |      | Depends on data type.                | ---      |
| Mode               | Search method                        |        | Search method  | ---                          |      | ---                                  | ---      |
| 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<br>FALSE: There are no elements that match conditions | Depends on data type.        | ---  | ---                                  |          |
| Num                | Number of matches                    |        | Number of matches  | ---                          | ---  | ---                                  |          |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                       | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Member</i> .   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| MX                    | Must be the same data type as <i>Member</i> .   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Condition             | Refer to <i>Function</i> on page 2-547 for the enumerators of the enumerated type <code>_eSEARCH_CONDITION</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Mode                  | Refer to <i>Function</i> on page 2-547 for the enumerators of the enumerated type <code>_eSEARCH_MODE</code> .      |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| InOutPos[]<br>(array) |   |             |      |       |       |          | OK   |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out                   | OK  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Num                   |   |             |      |       |       |          | OK   |       |       |      |     |      |      |                      |       |  |      |     |    |        |

\*1. 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[], i.e., from In[0] to In[Size-1], for *Member* (member to search) that matches the search condition.

*Condition* specifies the search condition, and *Mode* specifies the search method. Details are provided below.

The member to search of an element in In[] is passed to *Member* as an argument.

If any element that matches the search condition is 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 *Condition* (search condition) is enumerated type `_eSEARCH_CONDITION`. The meanings of the enumerators are as follows:

| Enumerator            | Meaning                  |
|-----------------------|--------------------------|
| <code>_EQ_BOTH</code> | $MN \leq Member \leq MX$ |
| <code>_EQ_MIN</code>  | $MN \leq Member < MX$    |
| <code>_EQ_MAX</code>  | $MN < Member \leq MX$    |
| <code>_NE_BOTH</code> | $MN < Member < MX$       |

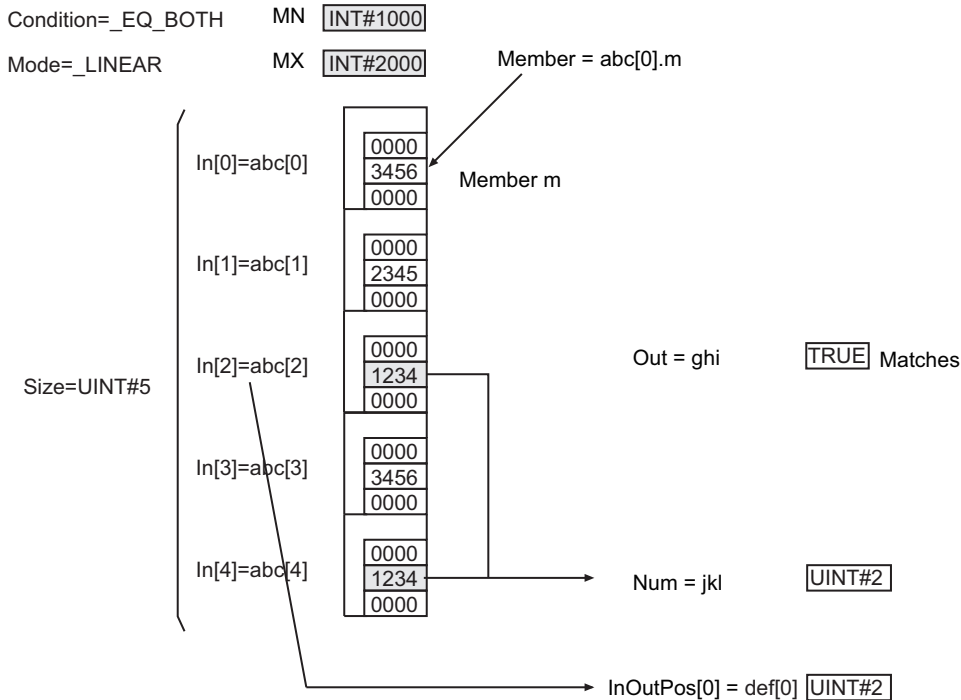
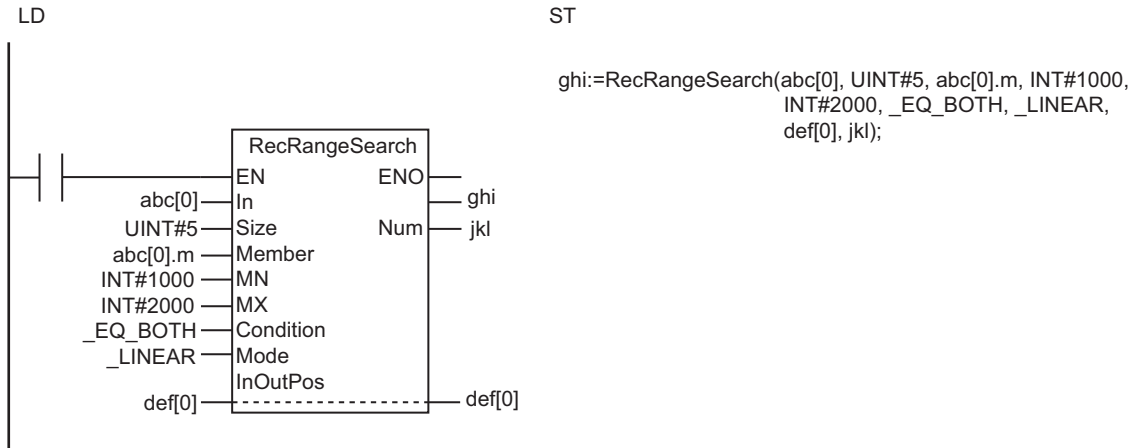
The data type of *Mode* (search method) is enumerated type `_eSEARCH_MODE`. The meanings of the enumerators are as follows:

| Enumerator           | Meaning       |
|----------------------|---------------|
| <code>_LINEAR</code> | Linear search |

| Enumerator | Meaning                  |
|------------|--------------------------|
| _BIN_ASC   | Ascending binary search  |
| _BIN_DESC  | Descending binary search |

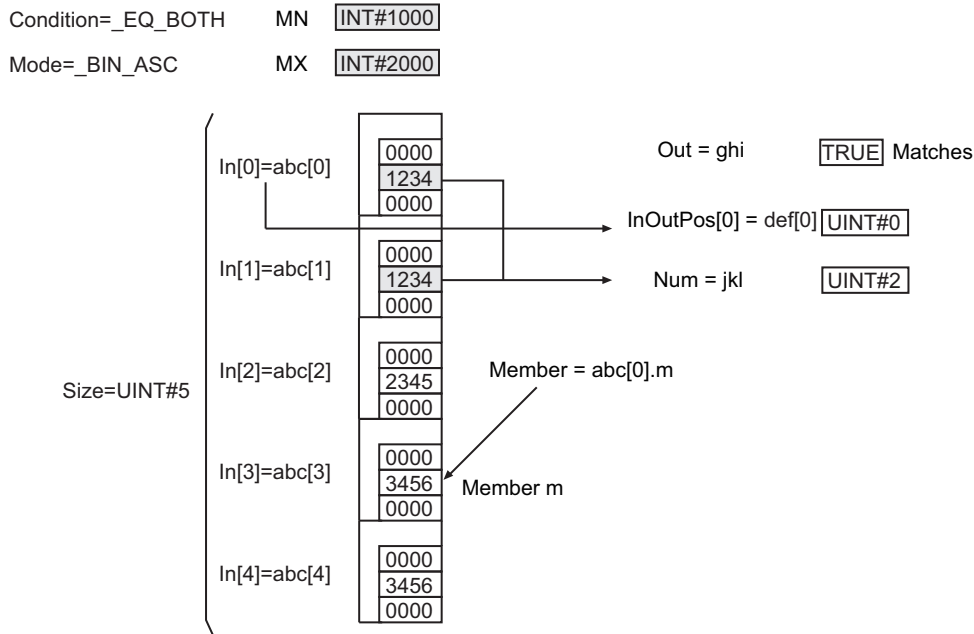
For a linear search, the search is performed in order from the first element of In[].

The following shows an example where *Size* is UINT#5, *MN* is INT#1000, *MX* is INT#2000, *Condition* is \_EQ\_BOTH, and *Mode* is \_LINEAR.



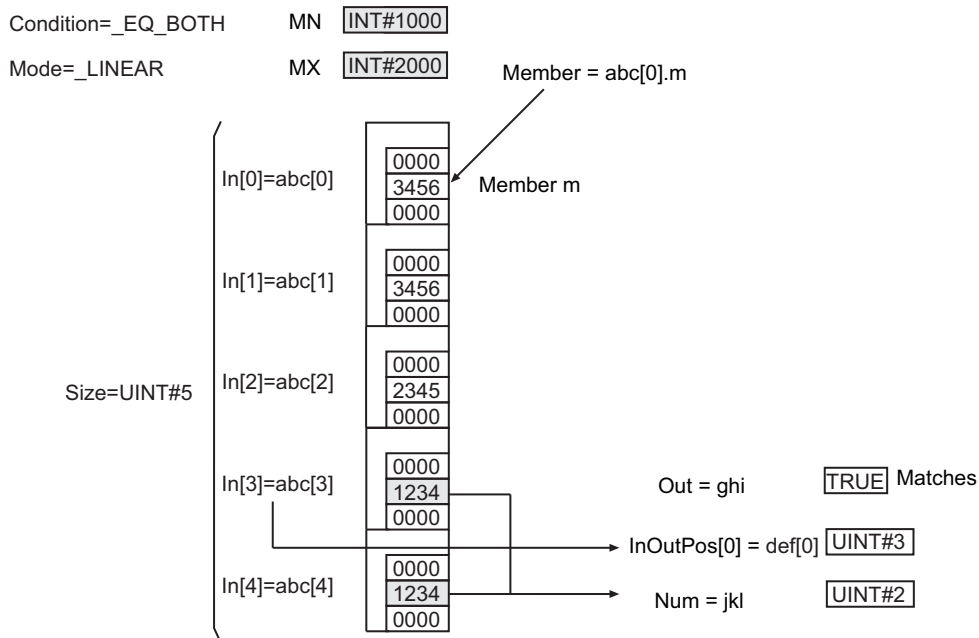
For an ascending binary search, the array elements in the input parameter that is passed to In[] must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

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 accuracy of *Member*, *MN*, and *MX* to the same. You can use the following instructions for the adjustment: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

### Precautions for Correct Use

- Use the same data type for *Member*, *MN*, and *MX* as that of the `In[]` structure member to search. 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, the desired results may not be achieved due to error, depending on the value.
- If *MN* or *MX* is a real number, do not specify nonnumeric data.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. `InOutPos[]` does not change.
- A correct result will not be obtained if the value of *Mode* is `_BIN_ASC` or `_BIN_DESC` and the elements of `In[]` are not in ascending or descending order. Sort the elements in ascending or descending order before executing this instruction.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out*, `InOutPos[]`, and *Num* will not change.
  - a) *MN* is greater than *MX*.
  - b) The value of *Condition* is outside the valid range.
  - c) The value of *Mode* is outside the valid range.
  - d) The value of *Size* exceeds the array area of `In[]`.
  - e) *Member* is not a member of `In[]`.
  - f) 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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------------|------------|--------------------|---|
| RecSort     | Record Sort | FB         |                    | RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error); |

## Variables

|                 | 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 In[] structure to sort |                       |      | *1          |
| Order           | Sort order                 |        | Sort order                       |                       |      | _ASC, _DESC |
| InOut[] (array) | Sort array                 | In-out | Array of structures to sort      | ---                   | ---  | ---         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                 | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 | OK<br>*1 |
|                 | Specify the same data type as the sort member of InOut[]  |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |          |          |          |          |
| Order           | Refer to <i>Function</i> on page 2-551 for the enumerators of the enumerated type <code>_eSORT_ORDER</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |          |          |          |          |
| InOut[] (array) | Specify an array of structures.   |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |          |          |          |          |

\*1. 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, the RecSort instruction sorts *Size* elements of InOut[] (a structure array), i.e., from InOut[0] to InOut[Size-1], based on the value of *Member* (member to sort) of the structure. *Order* specifies the sort order.

The member to sort of an element in In[] is passed to *Member* as an argument.

Always attach the element number to the in-out parameter that is passed to InOut[], e.g., array[3].

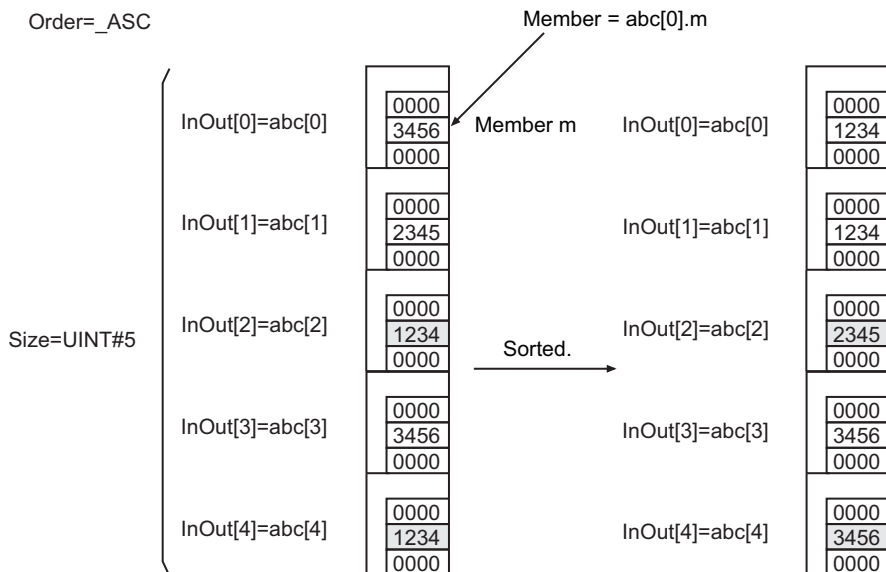
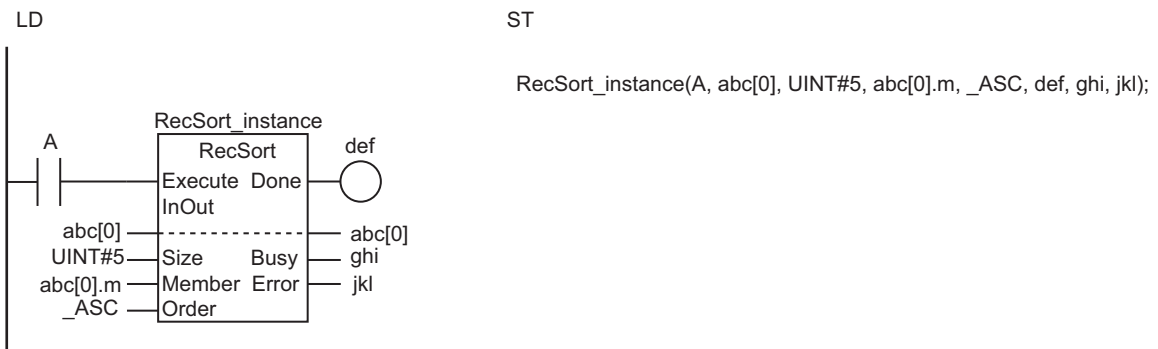
The data type of *Order* (sort order) is enumerated type `_eSORT_ORDER`. The meanings of the enumerators are as follows:

| Enumerator         | Meaning    |
|--------------------|------------|
| <code>_ASC</code>  | Ascending  |
| <code>_DESC</code> | 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 instructions, <i>LTascii</i> , <i>LEascii</i> , <i>GTascii</i> , and <i>GEascii</i> on page 2-115. Refer to the specified pages for details. |

The following shows an example where *Size* is `UINT#5` and *Order* is `_ASC`.





## 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 successfully completed, you can restore the data even if it is corrupted.  
Refer to *Sample Programming* on page 2-553.
- When you sort TIME, DT, or TOD data, adjust the precision of *Member* values to the same level. You can use the instructions, *TruncTime* on page 2-698, *TruncDt* on page 2-702, or *TruncTod* on page 2-706, to adjust the precision of values.

## 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 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 the execution.
- 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, the desired results may not be achieved due to error, depending on the value.
- When an element in the array is passed to InOut[], all the subsequent elements will be 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. *Error* will change to TRUE.
  - a) The value of *Order* is outside the valid range.
  - b) The value of *Size* exceeds the array area of InOut[].
  - c) *Member* is not a member of InOut[].
  - d) *Member* is STRING data and it does not end with a NULL character.

## Sample Programming

In this sample, the RecSort instruction sorts an array *Abc[]* of *MyStr* structures in ascending order. Sorting is performed based on the value of the *Abc[].m* member.

In order to prevent data loss due to a power interruption 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

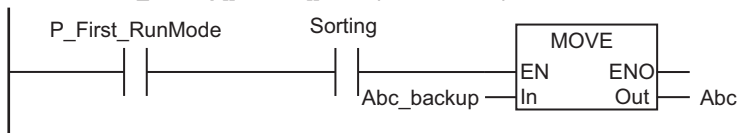
| Variable   | Data type            | Initial value               | Re-<br>tain                         | Comment         |
|------------|----------------------|-----------------------------|-------------------------------------|-----------------|
| Abc        | ARRAY[0..4] OF MyStr | [5((!:=FALSE,m:=0,n:=0.0))] | <input checked="" type="checkbox"/> | Sort array      |
| Abc_backup | ARRAY[0..4] OF MyStr | [5((!:=FALSE,m:=0,n:=0.0))] | <input checked="" type="checkbox"/> | Backup of Abc[] |

LD

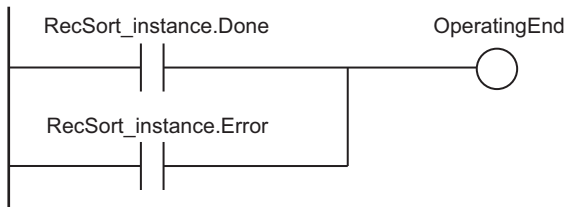
| Internal Variables | Variable         | Data type | Initial value | Retain                              | Comment               |
|--------------------|------------------|-----------|---------------|-------------------------------------|-----------------------|
|                    | Sorting          | BOOL      | FALSE         | <input checked="" type="checkbox"/> | Processing (retained) |
|                    | OperatingEnd     | BOOL      | FALSE         | <input type="checkbox"/>            | Processing completed  |
|                    | Trigger          | BOOL      | FALSE         | <input type="checkbox"/>            | Execution condition   |
|                    | Operating        | BOOL      | FALSE         | <input type="checkbox"/>            | Processing            |
|                    | RS_instance      | RS        |               | <input type="checkbox"/>            |                       |
|                    | RecSort_instance | RecSort   |               | <input type="checkbox"/>            |                       |

| External Variables | Variable   | Data type            | Comment         |
|--------------------|------------|----------------------|-----------------|
|                    | Abc        | ARRAY[0..4] OF MyStr | Sort array      |
|                    | Abc_backup | ARRAY[0..4] OF MyStr | Backup of Abc[] |

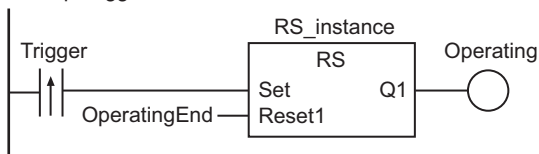
Restore **Abc\_backup[]** to **Abc[]** after power interruption.



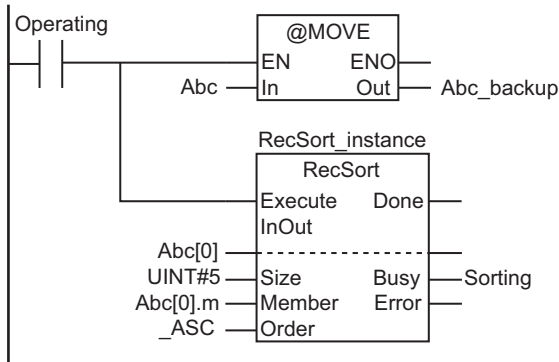
Determine if execution of the RecSort instruction is completed.



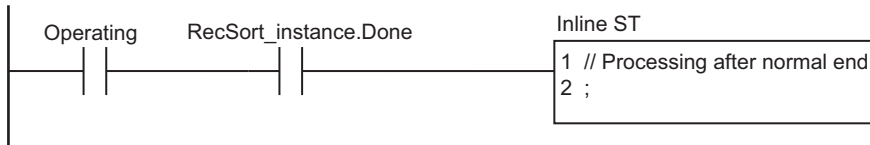
Accept trigger.



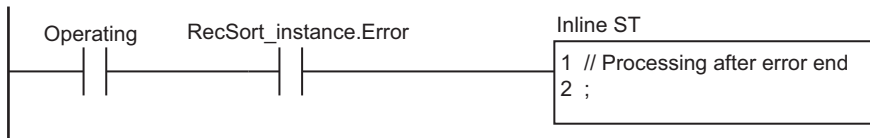
Make backup and execute RecSort instruction.



Processing after normal end.



Processing after error end



## ST

| Internal Variables | Variable         | Data type | Initial value | Retain                              | Comment   |
|--------------------|------------------|-----------|---------------|-------------------------------------|---|
|                    | Sorting          | BOOL      | FALSE         | <input checked="" type="checkbox"/> | Processing (retained)                                 |
|                    | Trigger          | BOOL      | FALSE         | <input type="checkbox"/>            | Execution condition                                   |
|                    | LastTrigger      | BOOL      | FALSE         | <input type="checkbox"/>            | Value of <i>Trigger</i> from the previous task period |
|                    | OperatingStart   | BOOL      | FALSE         | <input type="checkbox"/>            | Processing started                                    |
|                    | Operating        | BOOL      | FALSE         | <input type="checkbox"/>            | Processing  |
|                    | RS_instance      | RS        |               | <input type="checkbox"/>            |   |
|                    | RecSort_instance | RecSort   |               | <input type="checkbox"/>            |   |

| 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/<br>FUN | Graphic expression   | ST expression                    |
|-------------|-----------------------|------------|--|----------------------------------|
| RecNum      | Get Number of Records | FUN        | <pre> graph LR     subgraph RecNum_Box [(@)RecNum]         EN[EN]         In[In]         Member[Member]         EndDat[EndDat]     end     RecNum_Box --&gt; Out[Out] </pre> | Out:=RecNum(In, Member, EndDat); |

## Variables

|              | Meaning           | I/O    | Description                         | Valid range           | Unit | Default |
|--------------|-------------------|--------|-------------------------------------|-----------------------|------|---------|
| In[] (array) | Array to process  | Input  | Array of structures to process      | ---                   | ---  | *1      |
| Member       | Member to process |        | Member of In[] structure to process | Depends on data type. |      |         |
| EndDat       | End data          |        | End data                            | Depends on data type. |      |         |
| Out          | Number of records | Output | Number of records                   | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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     |
|              | Enumerations can also be specified.*2                         |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| EndDat       | Must be the same data type as the members to process in In[]. |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Out          | 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.

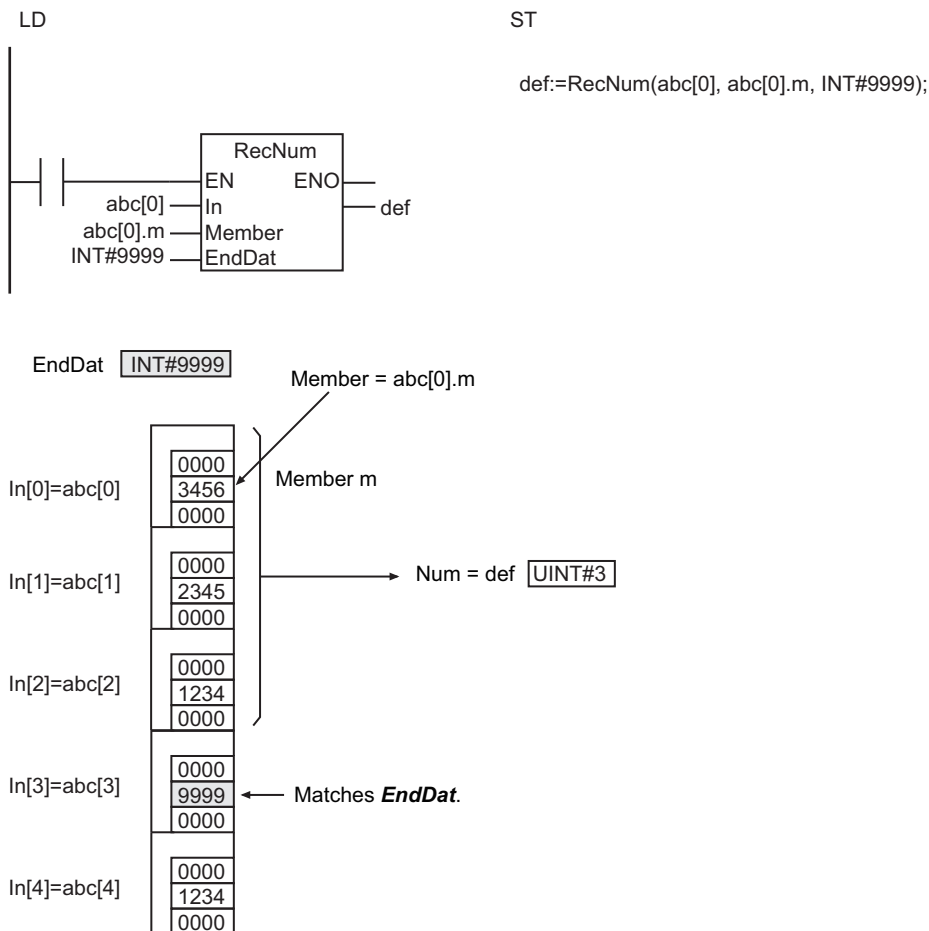
## Function

The RecNum instruction accesses a structure array In[] to search for an element whose value of *Member* (member to process) matches *EndDat* (end data). After that, the number of elements (records) before the element whose value matches *EndDat* (end data) is assigned to *Out*.

The member to process of an element in In[] is passed to *Member* as an argument.

Always attach the element number to input parameter that is passed to `In[]`, e.g., `array[3]`.

The following shows an example where *EndDat* is `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 accuracy of *Member* and *EndDat* to the same. You can use the following instructions for the adjustment: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

### 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 no member in `In[]` matches *EndDat*, the total number of elements in `In[]` is assigned to *Out*.
- If *Member* is a real number, the desired results may not be achieved due to error, depending on the value.
- If *EndDat* is a real number, do not specify nonnumeric data for *EndDat*.
- When an element in the array is passed to `In[]`, all elements below the passed element are processed.

- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) *Member* is not a member of *In[]*.
  - b) *Member* is STRING data and it does not end with a NULL character.

# RecMax and RecMin

RecMax : Searches an array of structures for the maximum value of a specified member.

RecMin : Searches an array of structures for the minimum value of a specified member.

| Instruction | Name                  | FB/<br>FUN | Graphic expression | ST expression                                  |
|-------------|-----------------------|------------|--------------------|--|
| RecMax      | Maximum Record Search | FUN        |                    | Out:=RecMax(In, Size, Member, In-OutPos, Num); |
| RecMin      | Minimum Record Search | FUN        |                    | Out:=RecMin(In, Size, Member, In-OutPos, Num); |

## Variables

|                    | Meaning                      | I/O    | Description                        | Valid range           | Unit | Default |
|--------------------|------------------------------|--------|------------------------------------|-----------------------|------|---------|
| In[] (array)       | Array to search              | Input  | Array of structures to search      | ---                   | ---  | *1      |
| 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 |                       |      | *1      |
| 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                       |                       |      |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                    | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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[] |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| InOutPos[] (array) |   |             |      |       |       |          | OK   |       |       |      |     |      |      |                      |       |  |      |     |    |        |



|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num-<br>bers |          | Times, durations,<br>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    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK   | OK                   | OK<br>*1 | OK<br>*1                                     | OK<br>*1 | OK<br>*1 | OK<br>*1 |  |
| Num |             |             |      |       |       |       | OK       |       |       |      |     |      |      |      |                      |          |  |          |          |          |  |

\*1. 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 for the minimum or maximum value of *Member* (member to search) in *Size* elements, or from In[0] to In[Size-1], of the In[] structure array.

One of the members in an element of In[] is passed to *Member* as an argument.

The element number of the element with the minimum or maximum value is assigned to InOutPos[0], and the number of elements with the value is assigned to *Num*. If more than one element is found to have the value, the lowest element number of those with the value in the In[] array is assigned to In-OutPos[0].

Always attach the element number to input parameter that is passed to In[], e.g., array[3].

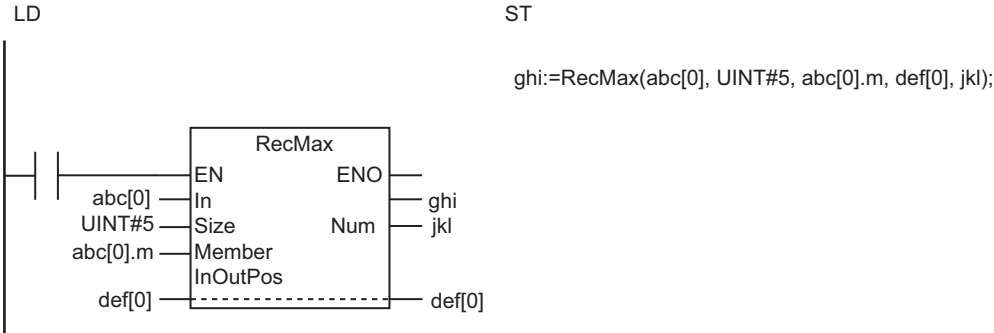
The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

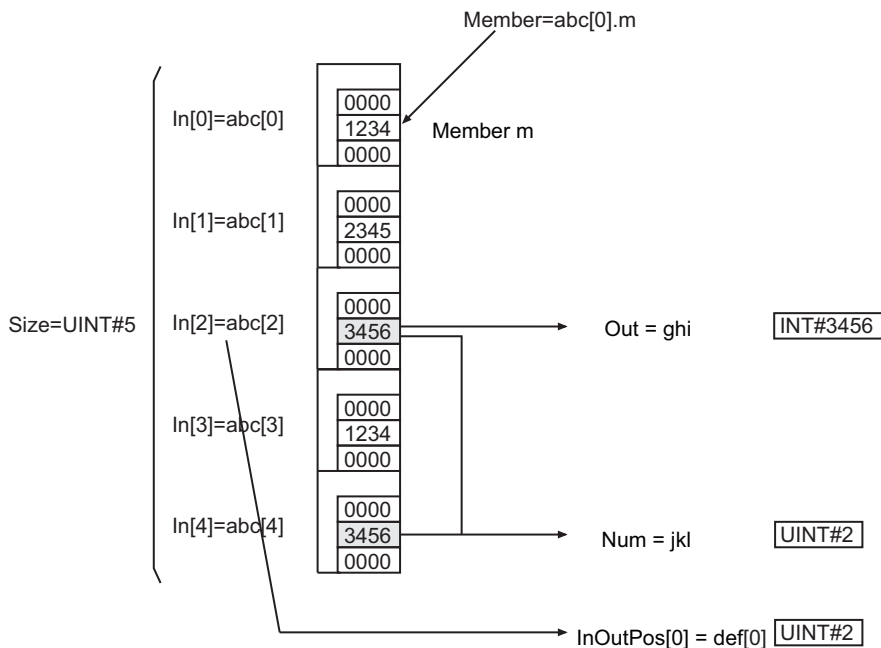
| Data type        | Relationship   |
|------------------|--|
| TIME             | The numerically larger value is considered to be larger.   |
| DATE, TOD, or DT | Later dates or times of day are considered to be larger.   |
| STRING           | The specifications are the same as for the instructions, <i>LTascii</i> , <i>LEascii</i> , <i>GTascii</i> , and <i>GEascii</i> on page 2-115. 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 *Out* (search result).

The following shows an example where *Size* is UINT#5 for the RecMax instruction.





## RecMin

The RecMin instruction searches for the minimum value. The minimum value of the member to search is assigned to search result *Out*.

### 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 accuracy of the *Member* values to the same. You can use the following instructions for the adjustment: *TruncTime* on page 2-698, *TruncDt* on page 2-702, and *TruncTod* on page 2-706.

### Precautions for Correct Use

- If you use different data types for *Member* and *Out*, they should be among the following data types, and make sure that the valid range of *Out* accommodates the valid range of *Member*.
  - a) USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *Member* is a real number, the desired results may not be achieved due to error, depending on the value.
- When an element in the array is passed to In[], all elements below the passed element are processed.

- 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 NULL characters. The values in *InOutPos*[] do not change.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos*[], and *Num* will not change.
  - a) The value of *Size* exceeds the array area of *In*[].
  - b) *Member* is not a member of *In*[].
  - c) The array size of *InOutPos*[] is smaller than the number of dimensions of *In*[].
  - d) *Member* is STRING data and it does not end with a NULL character.



# FCS Instructions

| Instruction    | Name                            | Page       |
|----------------|---------------------------------|------------|
| StringSum      | Checksum Calculation            | page 2-566 |
| StringLRC      | Calculate Text String LRC       | page 2-568 |
| StringCRCCCITT | Calculate Text String CRC-CCITT | page 2-570 |
| StringCRC16    | Calculate Text String CRC-16    | page 2-572 |
| AryLRC_**      | Calculate Array LRC Group       | page 2-574 |
| AryCRCCCITT    | Calculate Array CRC-CCITT       | page 2-576 |
| AryCRC16       | Calculate Array CRC-16          | page 2-578 |

# StringSum

The StringSum instruction calculates the checksum for a text string.

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|----------------------|------------|--------------------|---------------------------|
| StringSum   | Checksum Calculation | FUN        |                    | Out:=StringSum(In, Size); |

## Variables

|      | 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 | ---     |

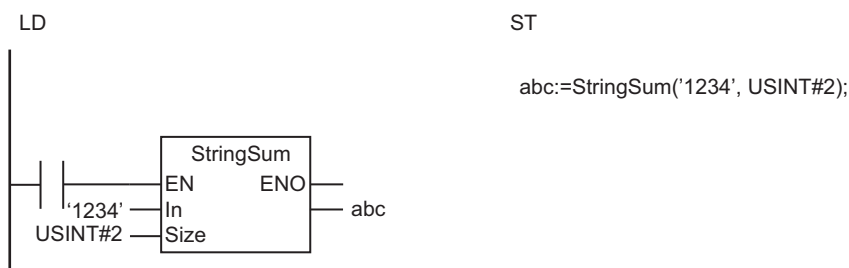
|      | Boo<br>lean | Bit strings |      |       |       |       | Integer |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------|-------------|-------------|------|-------|-------|-------|---------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|      | 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     |

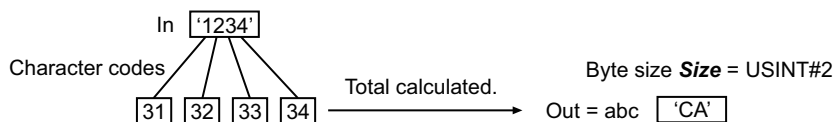
## Function

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.

In the following example, *In* is '1234' and *Size* is USINT#2.





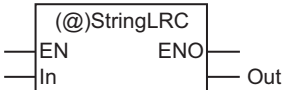
If *Size* is USINT#1 in the above example, *Out* would be 'A'.

## 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression  | ST expression       |
|-------------|---------------------------|------------|---|---------------------|
| StringLRC   | Calculate Text String LRC | FUN        |  | Out:=StringLRC(In); |

## Variables

|     | 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 a final NULL character) | ---  | ---     |

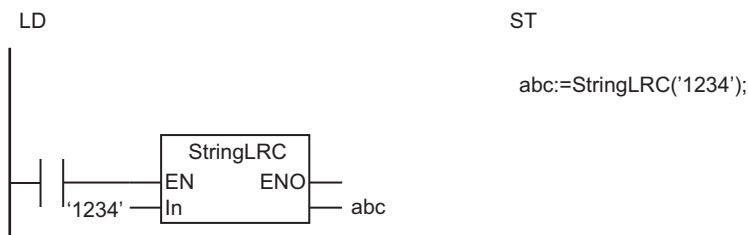
|     | Boo<br>lean | Bit strings |      |       |       | Integer |      |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|---------|------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|
|     | 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

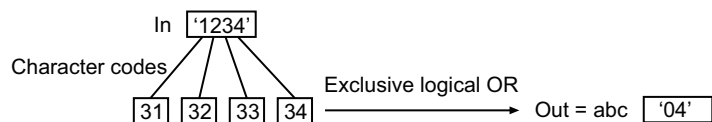
The StringLRC instruction calculates the LRC value (horizontal parity) of text string to process *In*. The LRC value is the exclusive logical OR of the character codes for the text string in *In*.

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'.







## Precautions for Correct Use

An error will occur in the following case. *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/<br>FUN | Graphic expression | ST expression                                  |
|---------------------|--|------------|--------------------|--|
| String-<br>CRCCCITT | Calculate Text<br>String CRC-<br>CCITT | FUN        |                    | Out:=StringCRCCCITT(In, Initial,<br>OutOrder); |

## Variables

|          | 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 <i>In</i> |  |      | _LOW_HIGH, _HIGH_LOW |
| Out      | CRC-CCITT value        | Output | CRC-CCITT value                     | 5 bytes (four single-byte alphanumeric characters plus the final NULL character) | ---  | ---                  |

|          | Boo<br>lean   | Bit strings |      |       |       | Integer |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-570 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |         |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| 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.

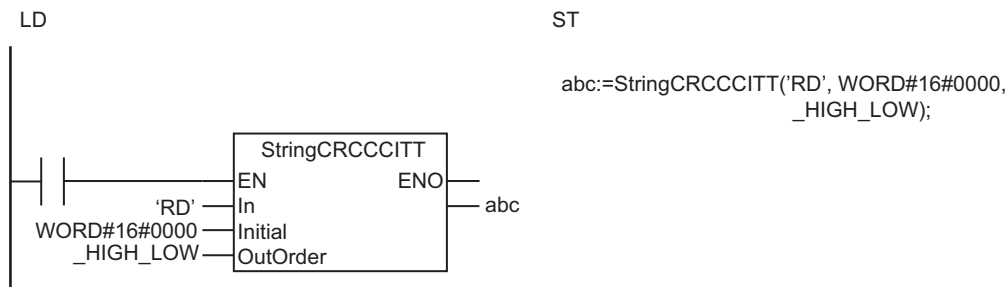
Set *Initial* to the initial value for CRC-CCITT value calculation. *OutOrder* specifies the byte order.

The data type of *OutOrder* is enumerated type `_eBYTE_ORDER`. The meanings of the enumerators are as follows:

| Enumerators            | Meaning                           |
|------------------------|-----------------------------------|
| <code>_LOW_HIGH</code> | Lower byte first, upper byte last |

| Enumerators | Meaning                           |
|-------------|-----------------------------------|
| _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.



## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside 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/<br>FUN | Graphic expression | ST expression                            |
|-------------|------------------------------|------------|--------------------|--|
| StringCRC16 | Calculate Text String CRC-16 | FUN        |                    | Out:=StringCRC16(In, Initial, OutOrder); |

## Variables

|          | 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 <i>In</i> |  |      | _LOW_HIGH, _HIGH_LOW |
| Out      | CRC-16 value           | Output | CRC-16 value                        | 5 bytes (four single-byte alphanumeric characters plus the final NULL character) | ---  | ---                  |

|          | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-572 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| 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.

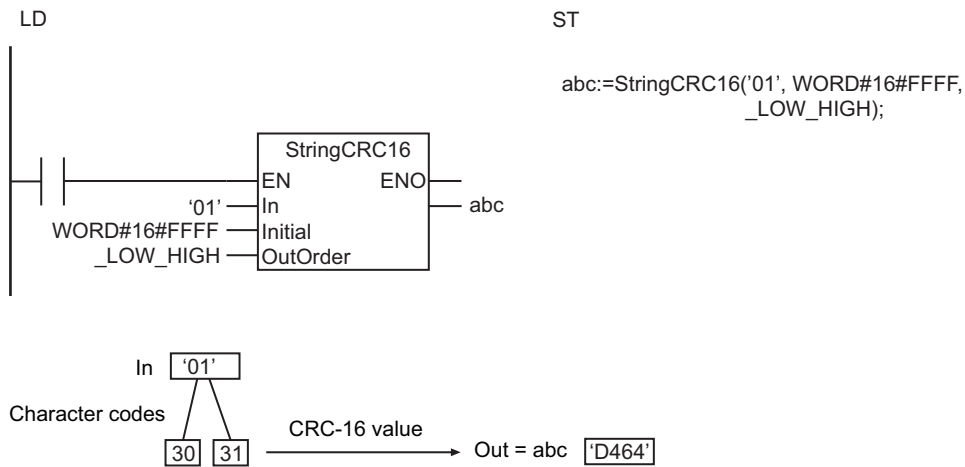
Set *Initial* to the initial value for CRC-16 value calculation. *OutOrder* specifies the byte order.

The data type of *OutOrder* is enumerated type `_eBYTE_ORDER`. The meanings of the enumerators are as follows:

| Enumerators            | Meaning                           |
|------------------------|-----------------------------------|
| <code>_LOW_HIGH</code> | Lower byte first, upper byte last |

| Enumerators            | Meaning                           |
|------------------------|-----------------------------------|
| <code>_HIGH_LOW</code> | 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`.



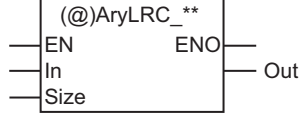
## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside the valid range.
- The number of bytes in *In* is 0 (i.e., the NULL character only).

# AryLRC\_\*\*

The AryLRC\_\*\* instructions calculates the LRC value of an array.

| Instruction | Name                      | FB/<br>FUN | Graphic expression  | ST expression   |
|-------------|---------------------------|------------|---|---|
| AryLRC_**   | Calculate Array LRC Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=AryLRC_**(In, Size);<br>**** must be a bit string data type. |

## jituVariables

|              | Meaning                       | I/O    | Description             | Valid range           | Unit | Default |
|--------------|-------------------------------|--------|-------------------------|-----------------------|------|---------|
| In[] (array) | Array to process              | Input  | Array to process        | Depends on data type. | ---  | *1      |
| Size         | Number of elements to process |        | Number of In[] elements |                       |      | 1       |
| Out          | LRC value                     | Output | LRC value               | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean                    | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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[] |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

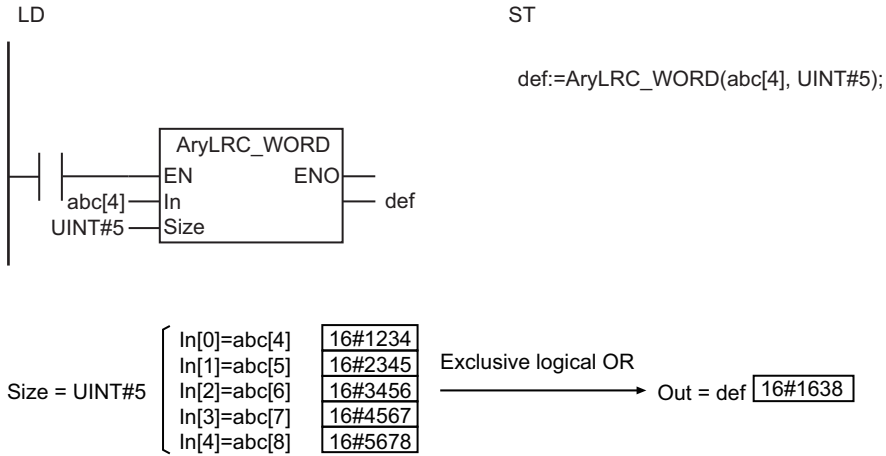
## Function

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.



### 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) 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/<br>FUN | Graphic expression | ST expression                                  |
|-------------|---------------------------|------------|--------------------|--|
| AryCRCCCITT | Calculate Array CRC-CCITT | FUN        |                    | Out:=AryCRCCCITT(In, Size, Initial, OutOrder); |

## Variables

|              | Meaning                       | I/O    | Description                         | Valid range           | Unit | Default              |
|--------------|-------------------------------|--------|-------------------------------------|-----------------------|------|----------------------|
| In[] (array) | Array to process              | Input  | Array to process                    | Depends on data type. | ---  | *1                   |
| 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 <i>In</i> |                       |      | _LOW_HIGH, _HIGH_LOW |
| Out          | CRC-CCITT value               | Output | CRC-CCITT value                     | Depends on data type. | ---  | ---                  |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-576 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out          |   | OK          |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The AryCRCCCITT instruction calculates the CRC-CCITT value of *Size* elements of an 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:

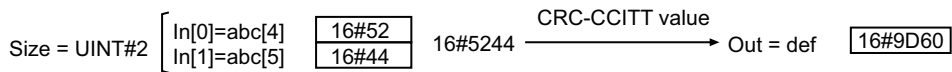
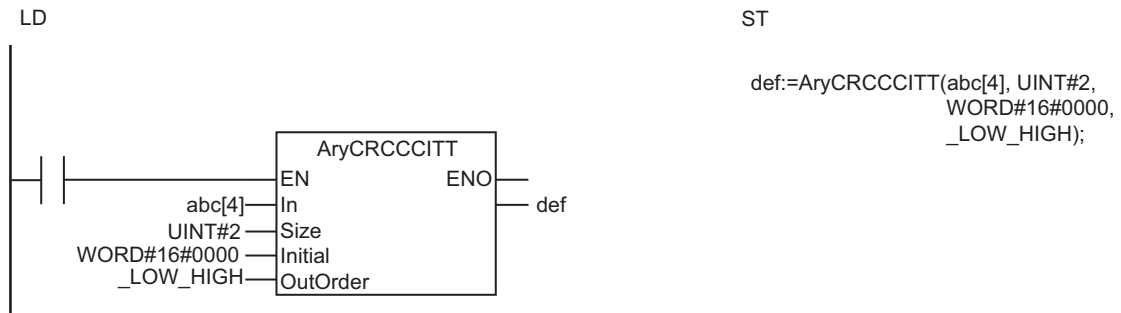
| Enumerators | Meaning                           |
|-------------|-----------------------------------|
| _LOW_HIGH   | Lower byte first, upper byte last |



| Enumerators            | Meaning                           |
|------------------------|-----------------------------------|
| <code>_HIGH_LOW</code> | Upper byte first, lower byte last |

Always attach an element number to the 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`.



## Precautions for Correct Use

- If the value of *Size* is 0, the value of *Out* is `WORD#16#0`.
- An error will occur in the following cases. *ENO* will be `FALSE`, and *Out* will not change.
  - a) The value of *OutOrder* is outside the valid range.
  - b) 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/<br>FUN | Graphic expression | ST expression                               |
|-------------|------------------------|------------|--------------------|---|
| AryCRC16    | Calculate Array CRC-16 | FUN        |                    | Out:=AryCRC16(In, Size, Initial, OutOrder); |

## Variables

|              | Meaning                       | I/O    | Description                         | Valid range           | Unit | Default                 |
|--------------|-------------------------------|--------|-------------------------------------|-----------------------|------|-------------------------|
| In[] (array) | Array to process              | Input  | Array to process                    | Depends on data type. | ---  | *1                      |
| Size         | Number of elements to process |        | Number of In[] elements             |                       |      | 1                       |
| Initial      | Initial value                 |        | Initial value of CRC-16 value       |                       |      | 16#FFF<br>F             |
| OutOrder     | Byte order                    |        | Order to process bytes in <i>In</i> |                       |      | _LOW_HIGH,<br>_HIGH_LOW |
| Out          | CRC-16 value                  | Output | CRC-16 value                        | Depends on data type. | ---  | ---                     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|              | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-578 for the enumerators of the enumerated type <code>_eBYTE_ORDER</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out          |   | OK          |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The AryCRC16 instruction calculates the CRC-16 value of *Size* array elements of an 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.

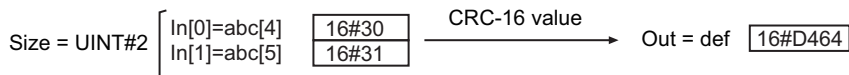
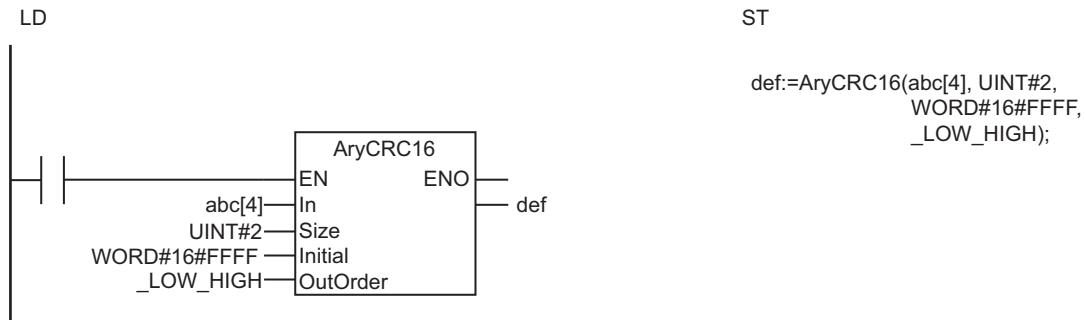
The data type of *OutOrder* is enumerated type `_eBYTE_ORDER`. The meanings of the enumerators are as follows:

| Enumerator | Meaning                           |
|------------|-----------------------------------|
| _LOW_HIGH  | Lower byte first, upper byte last |

| Enumerator | Meaning                           |
|------------|-----------------------------------|
| _HIGH_LOW  | Upper byte first, lower byte last |

Always attach the element number 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.



## Precautions for Correct Use

- If the value of *Size* is 0, the value of *Out* is WORD#16#0.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *OutOrder* is outside the valid range.
  - b) The value of *Size* exceeds the array area of In[].

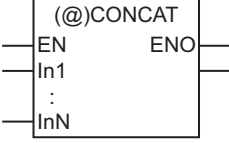


# Text String Instructions

| Instruction                 | Name                                      | Page       |
|-----------------------------|---|------------|
| CONCAT                      | Concatenate String                        | page 2-582 |
| LEFT and RIGHT              | Get String Left/Get String Right          | page 2-584 |
| MID                         | Get String Any                            | page 2-587 |
| FIND                        | Find String                               | page 2-589 |
| LEN                         | String Length                             | page 2-591 |
| REPLACE                     | Replace String                            | page 2-593 |
| DELETE                      | Delete String                             | page 2-595 |
| INSERT                      | Insert String                             | page 2-597 |
| GetByteLen                  | Get Byte Length                           | page 2-599 |
| ClearString                 | Clear String                              | page 2-600 |
| ToUpperCase and ToLowerCase | Convert to Uppercase/Convert to Lowercase | page 2-602 |
| TrimL and TrimR             | Trim String Left/Trim String Right        | page 2-604 |
| AddDelimiter                | Put Text Strings with Delimiters          | page 2-606 |
| SubDelimiter                | Get Text Strings Minus Delimiters         | page 2-618 |

# CONCAT

The CONCAT instruction joins two to five text strings.

| Instruction | Name               | FB/<br>FUN | Graphic expression  | ST expression               |
|-------------|--------------------|------------|---|-----------------------------|
| CONCAT      | Concatenate String | FUN        |  | Out:=CONCAT(In1, ..., InN); |

## Variables

|            | 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. | ---  | " *1    |
| Out        | Result of joining | Output | Text string that resulted from joining  | Depends on data type. | ---  | ---     |

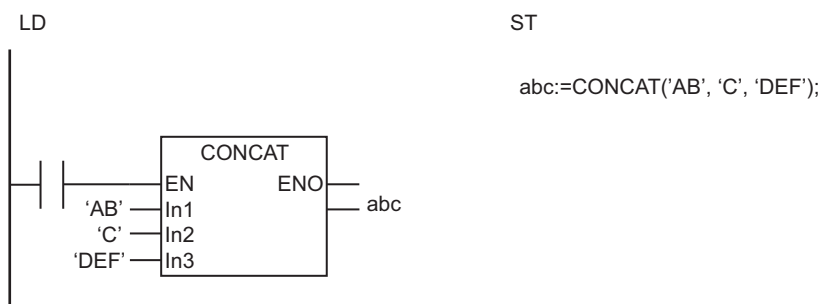
- \*1. If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

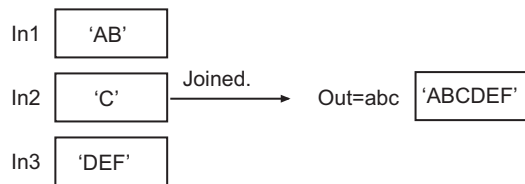
|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |    |
|------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|----|
|            | 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'.





## Precautions for Correct Use

An error will occur in the following case. *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 substring with a specified number of characters from a text string.

LEFT : Extracts the characters from the start (left) of the text string.

RIGHT : Extracts the characters from the end (right) of the text string.

| Instruction | Name             | FB/<br>FUN | Graphic expression | ST expression      |
|-------------|------------------|------------|--------------------|--------------------|
| LEFT        | Get String Left  | FUN        |                    | Out:=LEFT(In, L);  |
| RIGHT       | Get String Right | FUN        |                    | Out:=RIGHT(In, L); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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

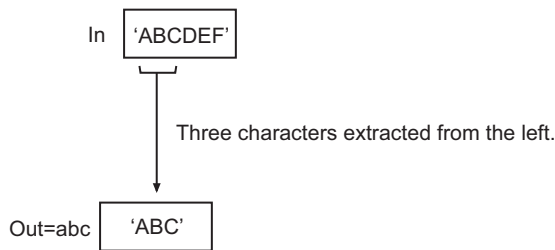
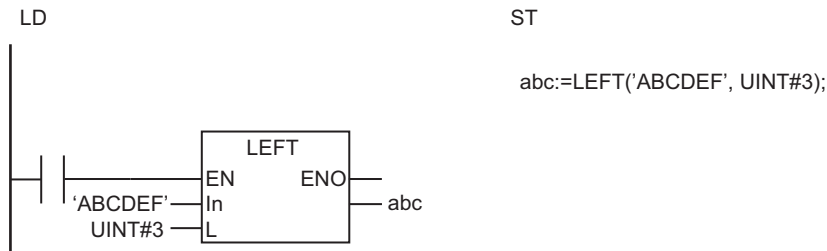
These instructions extract a text string with the number of characters specified by number of characters *L* from the source string *In*. A NULL character is placed at the end of the extraction result *Out*.

### LEFT

Extracts characters from the left (beginning) of *In*.



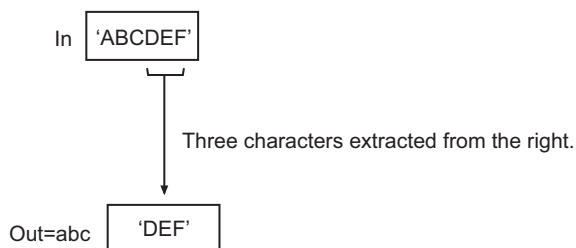
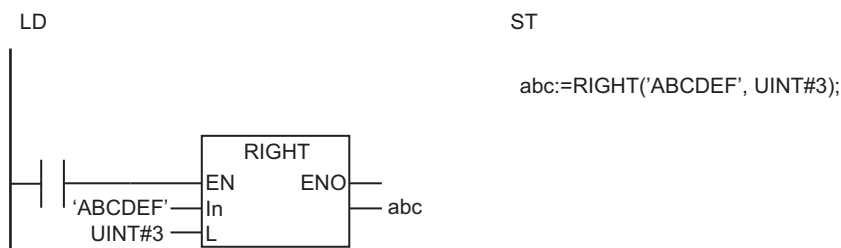
The following example is for when *In* is 'ABCDEF' and *L* is UINT#3. The value of variable *abc* will be 'ABC'.



## 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'.



## 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* results in a character code error.

# MID

The MID instruction extracts a substring with a specified number of characters from a specified position of a text string.

| Instruction | Name           | FB/<br>FUN | Graphic expression | ST expression       |
|-------------|----------------|------------|--------------------|---------------------|
| MID         | Get String Any | FUN        |                    | Out:=MID(In, L, P); |

## Variables

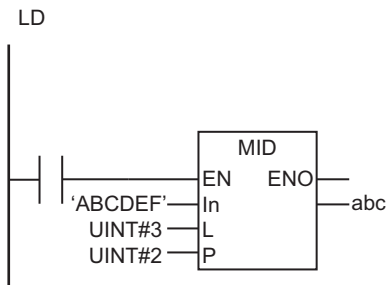
|     | Meaning              | I/O    | Description                                  | Valid range           | Unit | Default |
|-----|----------------------|--------|--|-----------------------|------|---------|
| In  | Source string        | Input  | Text string from which to extract characters | Depends on data type. | ---  | "       |
| L   | Number of characters |        | Number of characters to extract              | 0 to 1985             |      | 1       |
| P   | First character      |        | First character to extract                   | 1 to 1985             |      |         |
| Out | Extraction result    | Output | Extracted text string                        | Depends on data type. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |        | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|-------------------|--------|--|------|-----|----|--------|
|     | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL              | LRREAL | TIME   | DATE | TOD | DT | STRING |
| In  |             |             |      |       |       |       |          |       |       |      |     |      |      |                   |        |  |      |     |    | OK     |
| L   |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                   |        |  |      |     |    |        |
| P   |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                   |        |  |      |     |    |        |
| Out |             |             |      |       |       |       |          |       |       |      |     |      |      |                   |        |  |      |     |    | OK     |

## Function

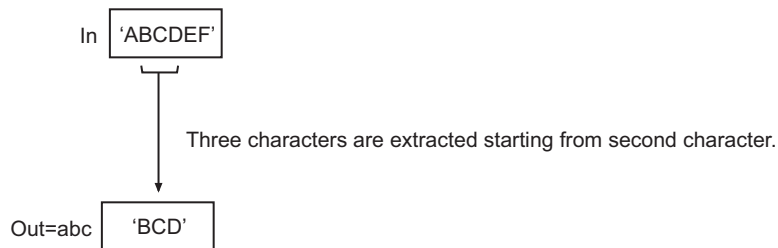
The MID instruction extracts the number of characters specified by number of characters *L* from the source string *In*. The first character to extract is specified by first character *P*. A NULL character is placed at the end of the extraction result *Out*.

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'.



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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* results in a character code error.
  - b) *In* does not have *L* characters after the position specified by *P*.
  - c) The value of *P* is 0.

# FIND

The FIND instruction searches for the position of a specified substring in a text string.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression        |
|-------------|-------------|------------|--------------------|----------------------|
| FIND        | Find String | FUN        |                    | Out:=FIND(In1, In2); |

## Variables

|     | 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             | ---  | ---     |

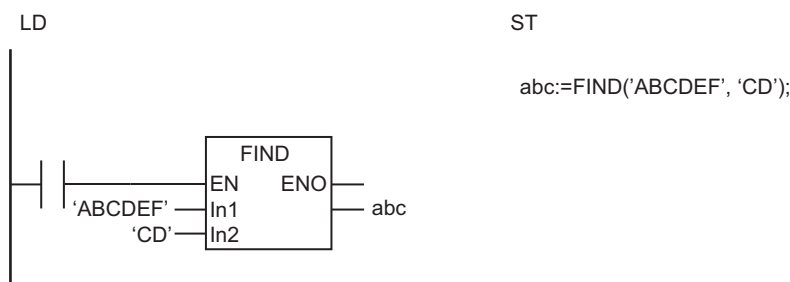
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|-------------------|-------|--|------|-----|----|--------|
|     | 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       |       |       |      |     |      |      |                   |       |  |      |     |    |        |

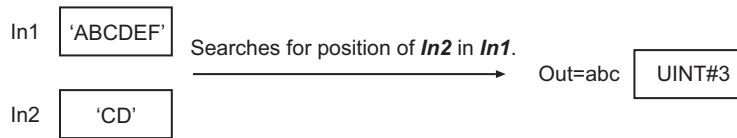
## Function

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.





### Precautions for Correct Use

- Make sure that the number of characters in *In2* is less than the number of characters in *In1*. Otherwise, the value of *Out* will be 0.
- If *In1* contains more than one *In2*, the position of *In2* which is first found in the search from the beginning of *In1* is assigned to *Out*.
- If both *In1* and *In2* contain only NULL characters, the value of *Out* is 1.
- Multi-byte characters are counted as one character each.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *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/<br>FUN | Graphic expression | ST expression |
|-------------|---------------|------------|--------------------|---------------|
| LEN         | String Length | FUN        |                    | Out:=LEN(In); |

## Variables

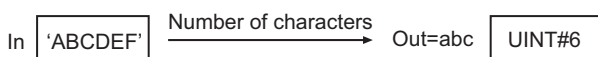
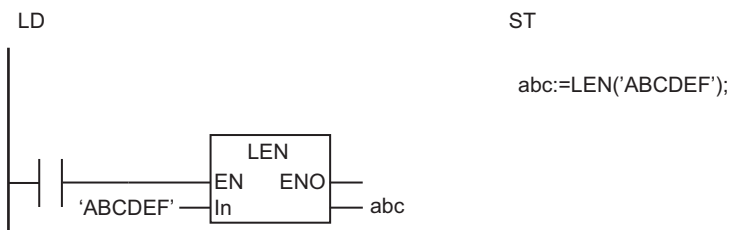
|     | Meaning       | I/O    | Description                | Valid range           | Unit | Default |
|-----|---------------|--------|----------------------------|-----------------------|------|---------|
| In  | Length string | Input  | Text string to find length | Depends on data type. | ---  | "       |
| Out | Find result   | Output | Length detection result    | 0 to 1985             | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.



## Precautions for Correct Use

- Multi-byte characters are counted as one character each.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- a) *In* results in a character code error.



# REPLACE

The REPLACE instruction replaces part of a text string with another text string.

| Instruction | Name           | FB/<br>FUN | Graphic expression | ST expression                 |
|-------------|----------------|------------|--------------------|-------------------------------|
| REPLACE     | Replace String | FUN        |                    | Out:=REPLACE(In1, In2, L, P); |

## Variables

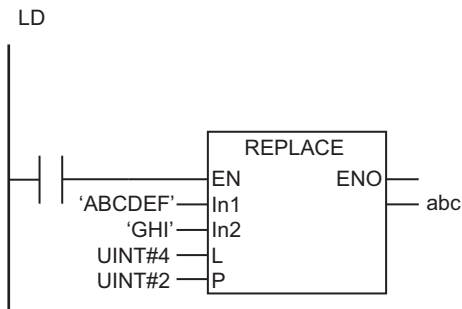
|     | Meaning                    | I/O    | Description                    | Valid range           | Unit | Default |
|-----|----------------------------|--------|--------------------------------|-----------------------|------|---------|
| In1 | String for replacement     | Input  | Text string for 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

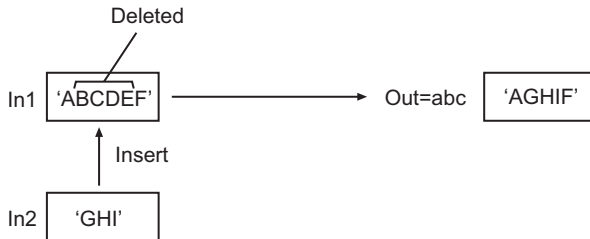
The REPLACE instruction replaces part of string for replacement *In1* with string to insert *In2*. First the number of characters specified by *L* from the position specified by *P* are deleted from *In1*. *In2* is then inserted for the deleted characters. A NULL character is placed at the end of replacement 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) *In1* results in a character code error.
  - b) *In1* does not have *L* characters after the position specified by *P*.
  - c) The value of *P* is 0.
  - d) 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/<br>FUN | Graphic expression | ST expression          |
|-------------|---------------|------------|--------------------|------------------------|
| DELETE      | Delete String | FUN        |                    | Out:=DELETE(In, L, P); |

## Variables

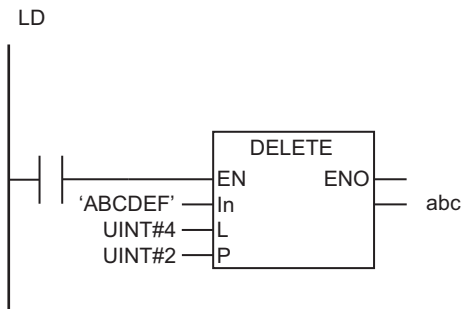
|     | Meaning                 | I/O    | Description                    | Valid range           | Unit | Default |
|-----|-------------------------|--------|--------------------------------|-----------------------|------|---------|
| In  | String for deletion     | Input  | Text string for deletion       | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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

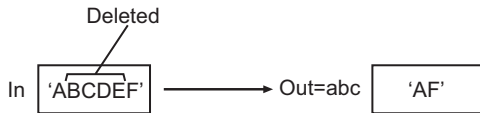
The DELETE instruction deletes the number of characters specified by *L* from the position specified by *P* from *In*. A NULL character is placed at the end of deletion result *Out*.

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'.



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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) *In* results in a character code error.
  - b) *In* does not have *L* characters after the position specified by *P*.
  - c) The value of *P* is 0.

# INSERT

The INSERT instruction inserts a text string into another text string.

| Instruction | Name          | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|---------------|------------|--------------------|---------------------------|
| INSERT      | Insert String | FUN        |                    | Out:=INSERT(In1, In2, P); |

## Variables

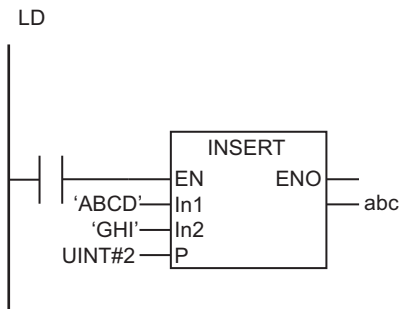
|     | Meaning                  | I/O    | Description                             | Valid range           | Unit | Default |
|-----|--------------------------|--------|---|-----------------------|------|---------|
| In1 | Original string          | Input  | Text string into which to insert string | Depends on data type. | ---  | "       |
| In2 | Insert string            |        | Text string to insert                   |                       |      |         |
| P   | Insertion start position |        | Insertion start position                |                       |      |         |
| Out | Insertion result         | Output | Text string after insertion             | Depends on data type. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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

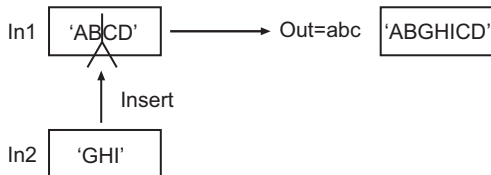
The INSERT instruction inserts insertion string *In2* into original string *In1* at insertion start position *P*. A NULL character is placed at the end of insertion result *Out*.

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'.



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 will occur in the following cases.  $ENO$  will be FALSE, and  $Out$  will not change.
  - a)  $In1$  results in a character code error.
  - b) The value of  $P$  is greater than the number of characters in  $In1$ .
  - c) 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/<br>FUN | Graphic expression | ST expression        |
|-------------|-----------------|------------|--------------------|----------------------|
| GetByteLen  | Get Byte Length | FUN        |                    | Out:=GetByteLen(In); |

## Variables

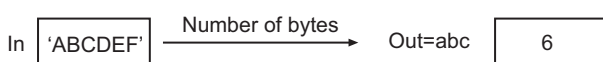
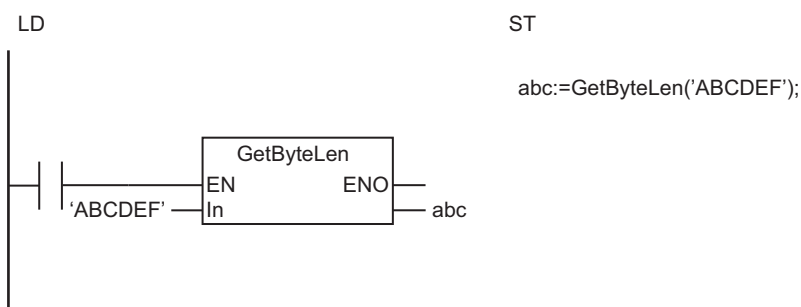
|     | Meaning         | I/O    | Description                          | Valid range           | Unit  | Default |
|-----|-----------------|--------|--------------------------------------|-----------------------|-------|---------|
| In  | Count string    | Input  | Text string to count number of bytes | 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.



## 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/<br>FUN | Graphic expression | ST expression       |
|-------------|--------------|------------|--------------------|---------------------|
| ClearString | Clear String | FUN        |                    | ClearString(InOut); |

## Variables

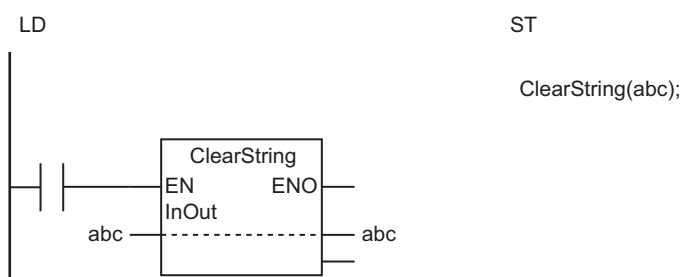
|       | Meaning      | I/O    | Description          | Valid range           | Unit | Default |
|-------|--------------|--------|----------------------|-----------------------|------|---------|
| InOut | Clear string | In-out | Text string to clear | Depends on data type. | ---  | ---     |
| Out   | Return value | Output | Always TRUE          | TRUE only             | ---  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 a STRING variable, *abc* will be all NULL characters.



The ClearString instruction stores NULL characters in the entire range of *InOut*.

The following shows an example where *abc* is a 5-character STRING variable.



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/<br>FUN | Graphic expression | ST expression     |
|-------------|---------------------------|------------|--------------------|-------------------|
| ToUCase     | Convert to Up-<br>percase | FUN        |                    | Out:=ToUCase(In); |
| ToLCase     | Convert to Low-<br>ercase | FUN        |                    | Out:=ToLCase(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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     |

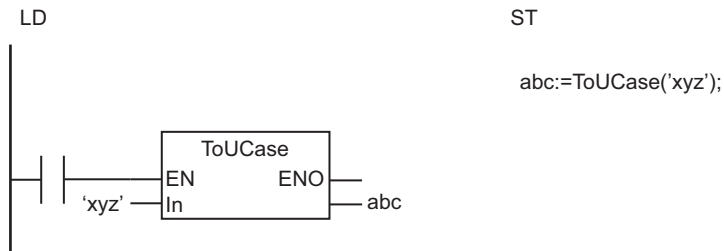
## Function

### ToUCase

The ToUCase instruction converts all single-byte letters in data to convert *In* to uppercase.

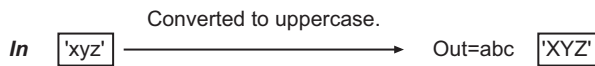
This instruction outputs a NULL character at the end of the text string. Only single-byte characters are changed.

The following example for the ToUCase instruction is for when *In* is 'xyz'. The value of variable *abc* will be 'XYZ'.



The ToUCase instruction converts all single-byte letters in *In* to uppercase.

The ToUCase instruction converts all single-byte letters in *In* to uppercase.



## ToLCase

The ToLCase instruction converts all single-byte letters in *In* to lowercase.

This instruction outputs a NULL character at the end of the text string. Only single-byte characters are changed.

## Precautions for Correct Use

- Two-byte letters are not converted.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) *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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------------------|------------|--------------------|-----------------|
| TrimL       | Trim String Left  | FUN        |                    | Out:=TrimL(In); |
| TrimR       | Trim String Right | FUN        |                    | Out:=TrimR(In); |

## Variables

|     | 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. | ---  | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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     |

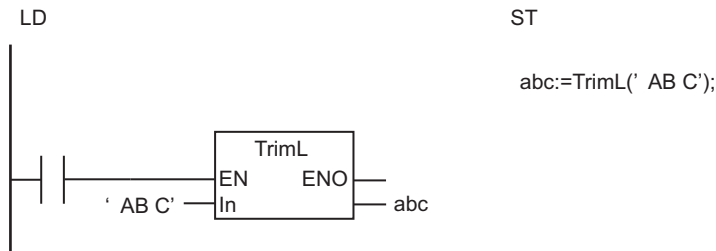
## Function

### 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.

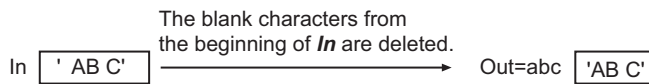
This instruction outputs a NULL character at the end of the text string. Both ASCII spaces (16#20) and two-byte Japanese spaces (16#E38080) are treated as blank characters.

The following example for the TrimL instruction is for when *In* is ' AB C'. The value of variable *abc* will be 'AB C'.



The TrimL instruction deletes blank characters from the beginning of *In*.

The TrimL instruction deletes blank characters from the beginning of *In*.



## 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.

This instruction outputs 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.

## Precautions for Correct Use

An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- *In* results in a character code error.

# AddDelimiter

The AddDelimiter instruction converts the values of all the members in a structure into a text string with delimiters.

| Instruction  | Name                             | FB/<br>FUN | Graphic expression | ST expression                     |
|--------------|----------------------------------|------------|--------------------|-----------------------------------|
| AddDelimiter | Put Text Strings with Delimiters | FUN        |                    | Out:=AddDelimiter(In, Delimiter); |



## 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.

## Variables

|           | Meaning         | I/O    | Description                          | Valid range  | Unit | Default |
|-----------|-----------------|--------|--------------------------------------|--|------|---------|
| In        | Input structure | Input  | Structure to convert to text strings | Depends on data type of members.   | ---  | *1      |
| 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) | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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        |             |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Delimiter |             |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out       |             |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    | OK     |

## Function

The AddDelimiter instruction converts each member of input structure *In* into a text string in order from the beginning, and then concatenates the strings with delimiter *Delimiter*. 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               |
|-------------------------|-----------------------|
| <code>_COMMA</code>     | ';' (comma)           |
| <code>_TAB</code>       | '\$T' (tab)           |
| <code>_SEMICOLON</code> | ';' (semicolon)       |
| <code>_SPACE</code>     | ' ' (blank character) |

The values of the members of *In* are converted according to their data types.

### ● 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

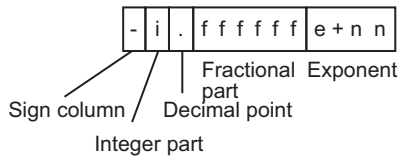
The value of the integer is converted to a text string. Upper digits that are 0 are not output to the text string. If the value of the member is negative, a minus sign (-) is added to the front of the text string.

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.<br>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.<br>'e' indicates the exponent "e".<br>'nn' is 2 or 3 digits.<br>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.

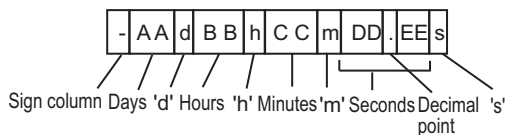
| Value of member | Text string     |
|-----------------|-----------------|
| + ∞             | 'inf'           |
| -∞              | '-inf'          |
| Nonnumeric data | 'nan' or '-nan' |

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 | '1.234567890000000e-03'  |
| LREAL#1.0e308       | '1.000000000000000e+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.<br>If the value of the member is positive, a plus sign (+) is not added. |
| Days        | The number of days is always given.<br>The range of the value is 0 to 106751.<br>Upper digits are not padded with 0.                        |
| Hours       | The number of hours is always given in two digits.<br>The range of the value is 00 to 23.   |
| Minutes     | The number of minutes is always given in two digits.<br>The range of the value is 00 to 59.   |



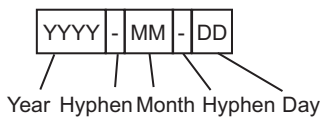
| Item                                      | Description  |
|---|--|
| Seconds                                   | The number of seconds is always given.<br>The value of 'DD' is always given in two digits between 00 and 59.<br>The value of 'EE' is always given in two digits between 000000000 and 999999999. |
| '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.

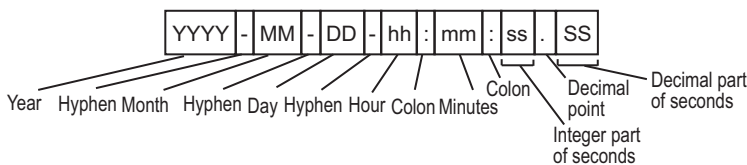


The month and day are converted to two digits each and output to the text string. An example is shown below.

| 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.



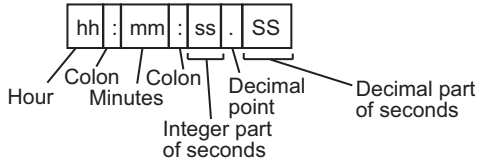
The month (MM), day (DD), hour (hh), minutes (mm), and integer part of the seconds (ss) are converted to two digits each and output to the text string. The fractional part of the seconds (ss) is converted to nine digits and output to the text string.

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**

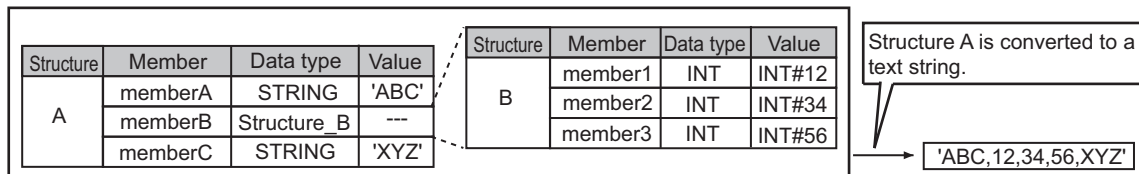
The text string is output without any changes. The NULL character at the end of the text string is not included.

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'.

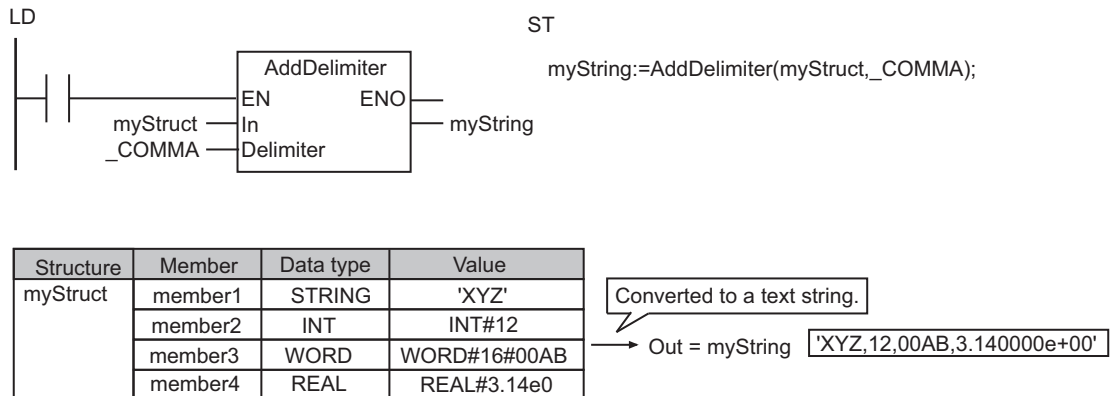
● **Array Data**

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 example shows how the *myStruct* structure is converted to the *myString* text string. The ',' (comma) is used as the delimiter.



## Additional Information

- You can combine this instruction with the instruction, *FilePuts* on page 2-1475, to easily write values to specified CSV files in an SD Memory Card. Refer to *Sample Programming* on page 2-611 for an application example.
- You can use the instruction, *SubDelimiter* on page 2-618, 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 include a delimiter in the value of a member of *In*. If a delimiter is included in the value of a member of *In*, the *SubDelimiter* instruction will not correctly convert the text string to the members of the structure.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The size of the resulting text string 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.

## 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:

- 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      |

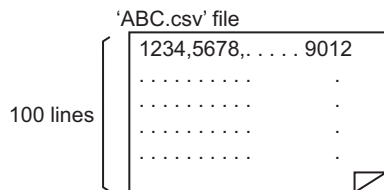
Array *myArray[0..99]* of structure type myStruct

|             | member0   | member1   | ⋮ | member9   |
|-------------|-----------|-----------|---|-----------|
| myArray[0]  | SINT#1234 | SINT#5678 | ⋮ | SINT#9012 |
| myArray[1]  | SINT#8487 | SINT#9256 | ⋮ | SINT#1211 |
| ⋮           | ⋮         | ⋮         | ⋮ | ⋮         |
| myArray[99] | SINT#0596 | SINT#6511 | ⋮ | SINT#2212 |

Converted to STRING data one line at a time.  
 STRING variable Temp 1234,5678, ⋮ ,9012

CR+LF added to the end.  
 STRING variable StrDat 1234,5678, ⋮ ,9012 CR+LF

Results written to the file.



## 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    |

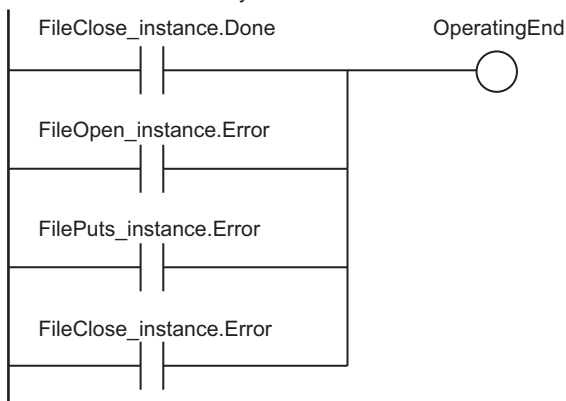
| Name    | Data type | Comment |
|---------|-----------|---------|
| member7 | SINT      | Member  |
| member8 | SINT      | Member  |
| member9 | SINT      | Member  |

## LD

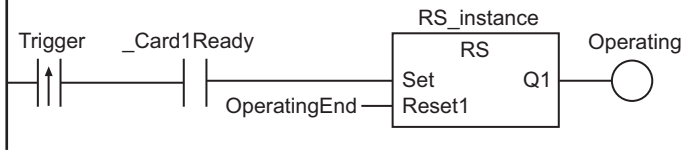
| Internal variables | Variable           | Data type                   | Initial value   | Comment              |
|--------------------|--------------------|-----------------------------|---|----------------------|
|                    | OperatingEnd       | BOOL                        | False   | Processing 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]<br>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 |

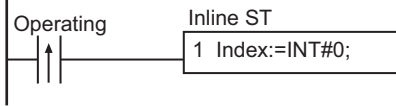
Determine if SD Memory Card instruction execution is completed.



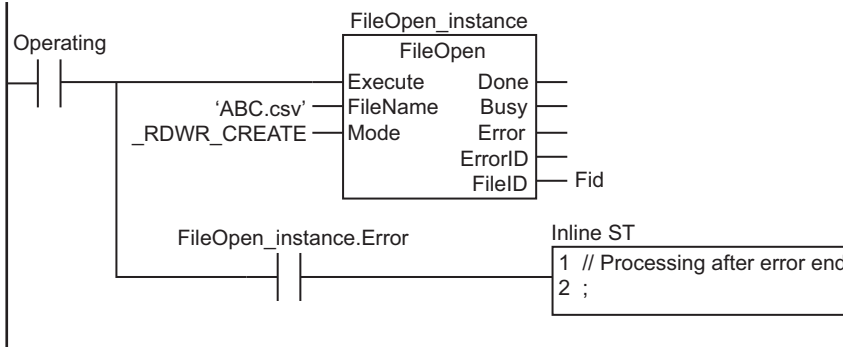
Accept trigger.



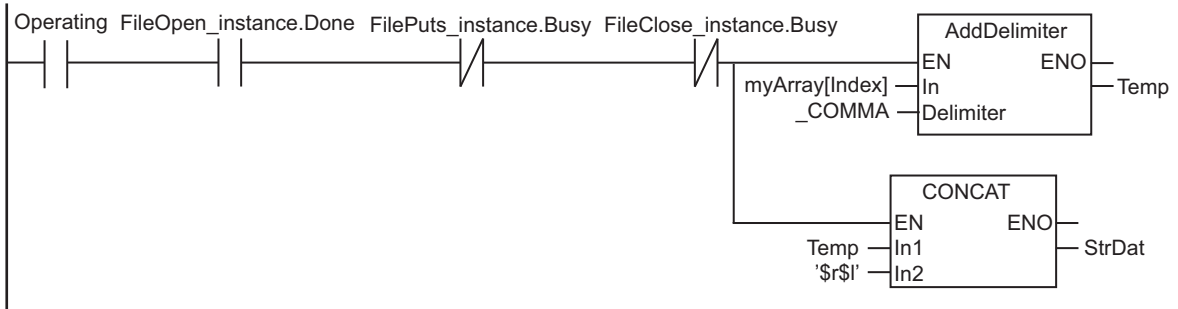
Initialize row index.



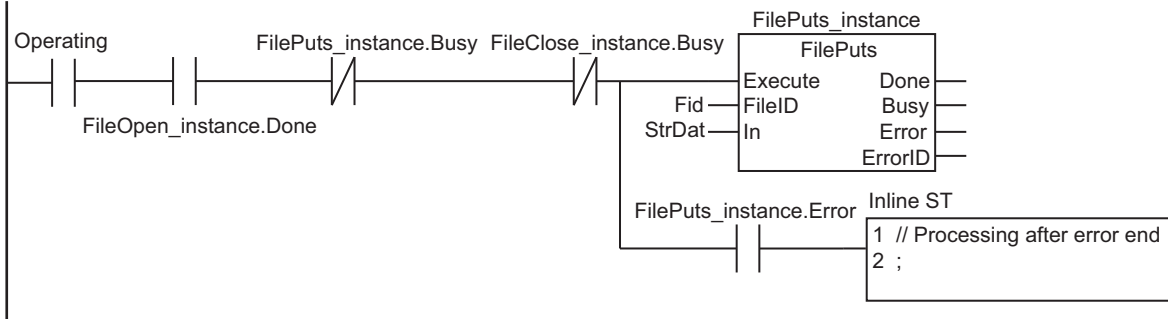
Execute FileOpen instruction.

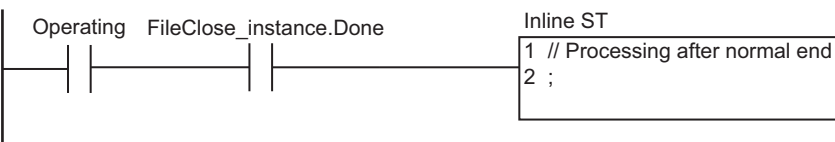
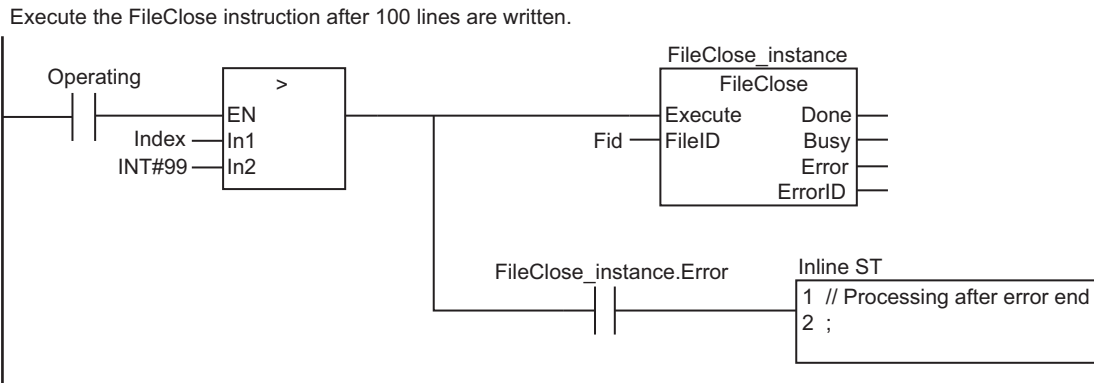
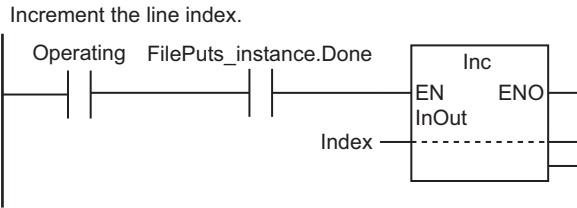


Create a text string for one line.



Write a text string for one line to the file.





**ST**

| Internal variables | Variable          | Data type                | Initial value  | Comment   |
|--------------------|-------------------|--------------------------|--|---|
|                    | Trigger           | BOOL                     | False  | Execution condition                               |
|                    | LastTrigger       | BOOL                     | False  | Value of <i>Trigger</i> 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                 |  |   |

| Internal variables | Variable           | Data type | Initial value | Comment |
|--------------------|--------------------|-----------|---------------|---------|
|                    | 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$1');

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 out delimited part of a text string and stores as the value of the members of a structure.

| Instruction  | Name                                 | FB/<br>FUN | Graphic expression | ST expression                                |
|--------------|--------------------------------------|------------|--------------------|--|
| SubDelimiter | Get Text Strings<br>Minus Delimiters | FUN        |                    | Out:=SubDelimiter(In, OutStruct, Delimiter); |



## 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.

## Variables

|           | 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  | ---  | ---     |

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-619 for the enumerators for the enumerated data type _eDELIMITER. |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| OutStruct | Structure  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Out       | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The SubDelimiter instruction converts text strings separated with *Delimiter* in *In* (input text string) into values for the members of *OutStruct* (storage structure) and assign each converted value to the corresponding member.

The data type of *Delimiter* is enumerated type `_eDELIMITER`. The meanings of the enumerators are as follows:

| Enumerator              | Meaning               |
|-------------------------|-----------------------|
| <code>_COMMA</code>     | ';' (comma)           |
| <code>_TAB</code>       | '\$T' (tab)           |
| <code>_SEMICOLON</code> | ',' (semicolon)       |
| <code>_SPACE</code>     | ' ' (blank character) |

If the number of delimited text strings in *In* exceeds the number of members of *OutStruct*, the remaining string data is ignored.

If the number of delimited text strings 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 the members of the 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 the elements of the 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 the data types of the members of *OutStruct*, as described below.

### ● 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 those for *STRING\_TO\_\*\** (*Text String-to-Bit String Conversion Group*) on page 2-319.

Conversion is not possible if the data does not express a hexadecimal number.

### ● Integer Data

The conversion rules are the same as those for *STRING\_TO\_\*\** (*Text String-to-Integer Conversion Group*) on page 2-317.

Conversion is not possible if the data does not express an integer number.

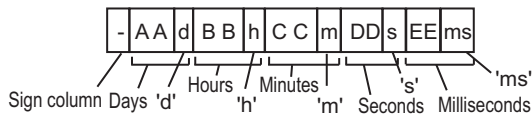
### ● Real Number Data

The conversion rules are the same as those for *STRING\_TO\_\*\** (*Text String-to-Real Number Conversion Group*) on page 2-321.

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.
- Note 2.** If any characters in the values of AA, BB, CC, DD, or EE are separated with a single ' \_ ' (underbar), the underbar is ignored.
- Note 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.
- Note 4.** If the days, hours, minutes, seconds, or milliseconds is included in the data, conversion is possible even if the other items are omitted.
- Note 5.** Even if there is a '0' before the value of the days, hours, minutes, seconds, or milliseconds, the data can still be converted.

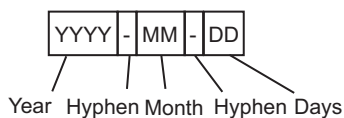
Conversion is not possible in the following cases.

- 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.

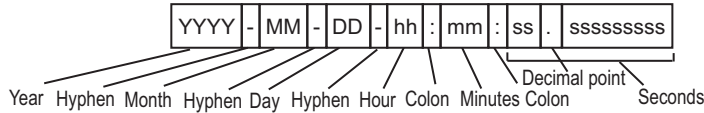
Conversion is not possible in the following cases.

- 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.
- Note 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.
- Note 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.

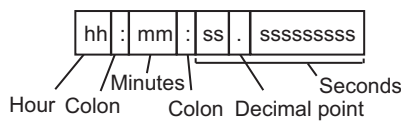
Conversion is not possible in the following cases.

- 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.
- Note 2.** If any characters in the values of hour, minutes, or seconds are separated with a single '\_' (underbar), the underbar is ignored.
- Note 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

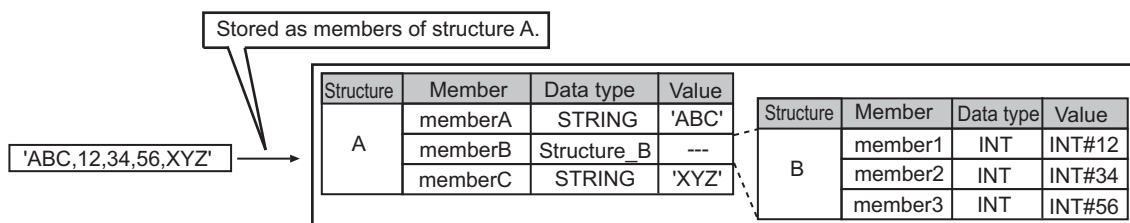
The value of the member will be the data with a NULL character added to the end. However, conversion is not possible if the text string exceeds the size of the member.

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*.

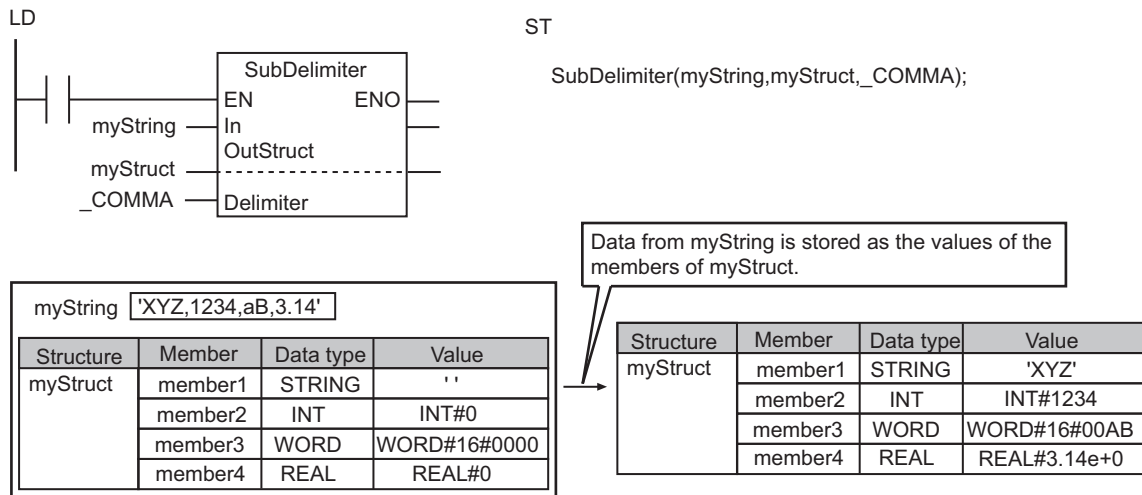
### ● Array Data

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 example shows how comma-separated data in *myString* are converted and assigned to the members of the *myStruct* structure.



## Additional Information

- You can combine this instruction with the instruction, *FileGets* on page 2-1467, to easily read values from specified CSV files in an SD Memory Card. Refer to *Sample Programming* on page 2-623 for an application example.
- Use this instruction to return a text string that was converted with the instruction, *AddDelimiter* on page 2-606, 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 delimiters for any other purpose in *In*. If you use a 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 string data before the 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 defined as an enumerator.
- An error will occur in the following cases. *ENO* will change to FALSE, and the values in *OutStruct* will be undefined.
  - Conversion to the data type of a member of *OutStruct* is not possible.
  - The conversion result exceeds the valid range of the corresponding member of *OutStruct*.
  - 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.

## 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).

'ABC.csv' file

```
OK CR
A,B,C CR
ABC,DEF CR
...
EOF
```

↓ Lines are read one at a time and stored in `myArray[]` members.

```
myArray[0].member0 'OK' myArray[1].member0 'A' myArray[2].member0 'ABC'
myArray[0].member0 Undefined myArray[1].member1 'B' myArray[2].member1 'DEF'
myArray[0].member0 Undefined myArray[1].member2 'C' myArray[2].member2 Undefined ...
myArray[0].member0 Undefined myArray[1].member3 Undefined myArray[2].member3 Undefined
myArray[0].member0 Undefined myArray[1].member4 Undefined myArray[2].member4 Undefined
```

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   |
|-----------------------|---------------------|-----------|
| <code>myStruct</code> | <code>STRUCT</code> | Structure |
| <code>member0</code>  | <code>STRING</code> | Member    |
| <code>member1</code>  | <code>STRING</code> | Member    |
| <code>member2</code>  | <code>STRING</code> | Member    |
| <code>member3</code>  | <code>STRING</code> | Member    |
| <code>member4</code>  | <code>STRING</code> | Member    |

## LD

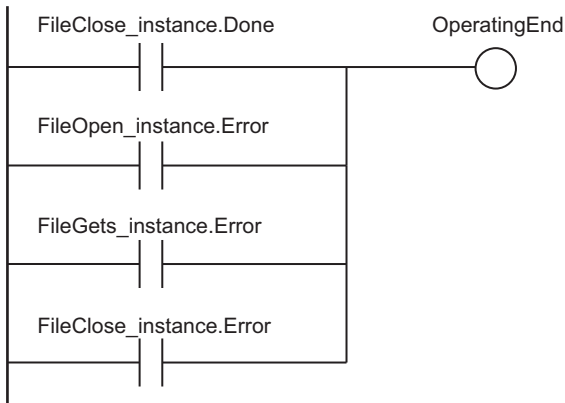
| Internal variables | Variable                  | Data type         | Initial value | Comment              |
|--------------------|---------------------------|-------------------|---------------|----------------------|
|                    | <code>OperatingEnd</code> | <code>BOOL</code> | False         | Processing completed |
|                    | <code>Trigger</code>      | <code>BOOL</code> | False         | Execution condition  |
|                    | <code>Operating</code>    | <code>BOOL</code> | False         | Processing           |



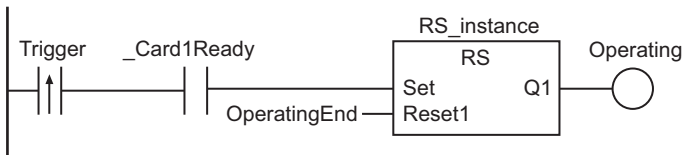
| Internal variables | Variable           | Data type                 | Initial value  | Comment                 |
|--------------------|--------------------|---------------------------|--|-------------------------|
|                    | Index              | INT                       | 0  | myArray[] 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 |

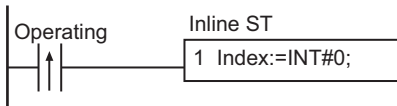
Determine if instruction execution is completed.



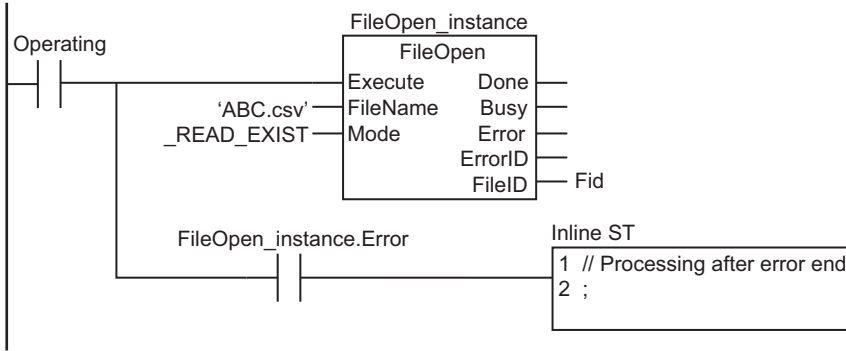
Accept trigger.



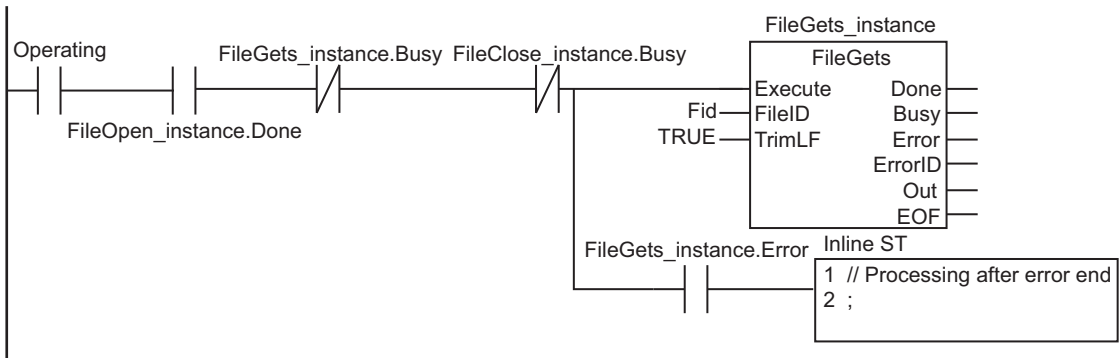
Initialize *InDat[]* element index.



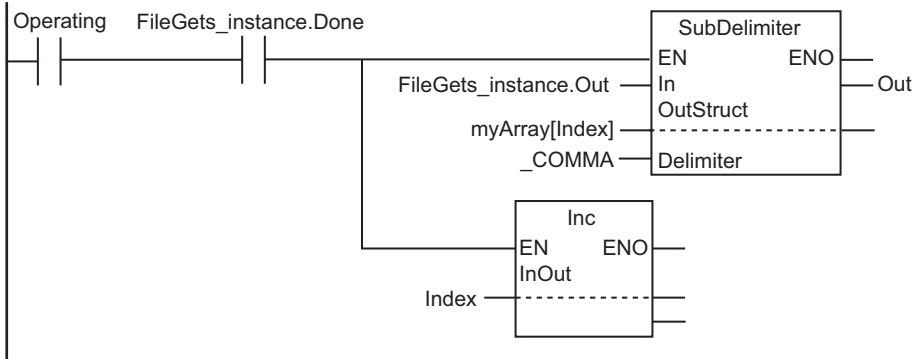
Execute FileOpen instruction.



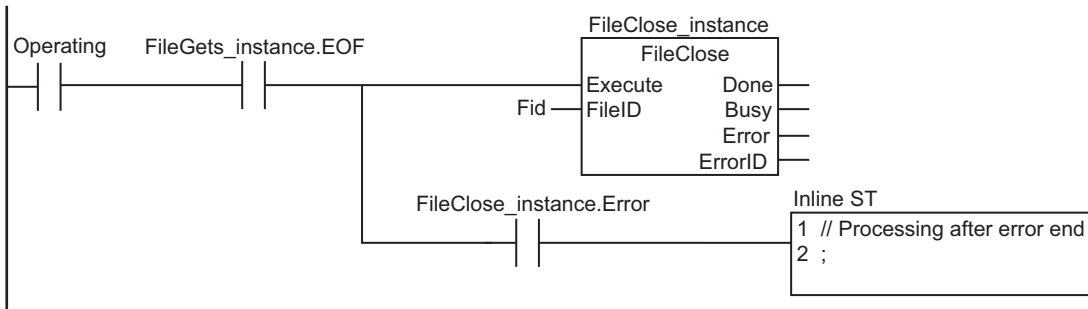
Execute FileGets instruction.



Execute SubDelimiter instruction.



Execute FileClose instruction when EOF is detected.





## ST

| Internal variables | Variable           | Data type                  | Initial value  | Comment   |
|--------------------|--------------------|----------------------------|--|---|
|                    | Trigger            | BOOL                       | False  | Execution condition                               |
|                    | LastTrigger        | BOOL                       | False  | Value of <i>Trigger</i> from previous task period |
|                    | OperatingStart     | BOOL                       | False  | Processing started                                |
|                    | Operating          | BOOL                       | False  | Processing  |
|                    | myArray            | AR-RAY[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

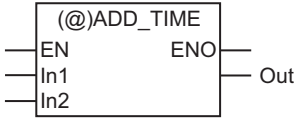
| Instruction     | Name                                   | Page       |
|-----------------|--|------------|
| ADD_TIME        | Add Time                               | page 2-633 |
| ADD_TOD_TIME    | Add Time to Time of Day                | page 2-635 |
| ADD_DT_TIME     | Add Time to Date and Time              | page 2-637 |
| SUB_TIME        | Subtract Time                          | page 2-639 |
| SUB_TOD_TIME    | Subtract Time from Time of Day         | page 2-641 |
| SUB_TOD_TOD     | Subtract Time of Day                   | page 2-643 |
| SUB_DATE_DATE   | Subtract Date                          | page 2-645 |
| SUB_DT_DT       | Subtract Date and Time                 | page 2-646 |
| SUB_DT_TIME     | Subtract Time from Date and Time       | page 2-648 |
| MULTIME         | Multiply Time                          | page 2-650 |
| DIVTIME         | Divide Time                            | page 2-652 |
| CONCAT_DATE_TOD | Concatenate Date and Time of Day       | page 2-654 |
| DT_TO_TOD       | Extract Time of Day from Date and Time | page 2-656 |
| DT_TO_DATE      | Extract Date from Date and Time        | page 2-658 |
| SetTime         | Set Time                               | page 2-660 |
| GetTime         | Get Time of Day                        | page 2-662 |
| DtToSec         | Convert Date and Time to Seconds       | page 2-664 |
| DateToSec       | Convert Date to Seconds                | page 2-666 |
| TodToSec        | Convert Time of Day to Seconds         | page 2-668 |
| SecToDt         | Convert Seconds to Date and Time       | page 2-670 |
| SecToDate       | Convert Seconds to Date                | page 2-672 |

| Instruction    | Name                           | Page       |
|----------------|--------------------------------|------------|
| SecToTod       | Convert Seconds to Time of Day | page 2-674 |
| TimeToNanoSec  | Convert Time to Nanoseconds    | page 2-676 |
| TimeToSec      | Convert Time to Seconds        | page 2-677 |
| NanoSecToTime  | Convert Nanoseconds to Time    | page 2-679 |
| SecToTime      | Convert Seconds to Time        | page 2-680 |
| ChkLeapYear    | Check for Leap Year            | page 2-682 |
| GetDaysOfMonth | Get Days in Month              | page 2-683 |
| DaysToMonth    | Convert Days to Month          | page 2-686 |
| GetDayOfWeek   | Get Day of Week                | page 2-688 |
| GetWeekOfYear  | Get Week Number                | page 2-690 |
| DtToDateStruct | Break Down Date and Time       | page 2-692 |
| DateStructToDt | Join Time                      | page 2-695 |
| TruncTime      | Truncate Time                  | page 2-698 |
| TruncDt        | Truncate Date and Time         | page 2-702 |
| TruncTod       | Truncate Time of Day           | page 2-706 |



# ADD\_TIME

The ADD\_TIME instruction adds two times.

| Instruction | Name     | FB/<br>FUN | Graphic expression  | ST expression            |
|-------------|----------|------------|---|--------------------------|
| ADD_TIME    | Add Time | FUN        |  | Out:=ADD_TIME(In1, In2); |

## Variables

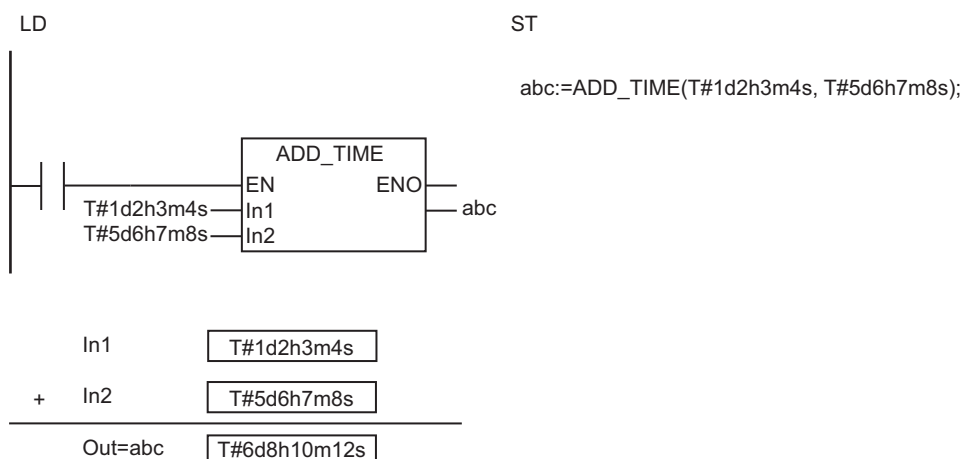
|     | Meaning    | I/O    | Description | Valid range           | Unit | Default |
|-----|------------|--------|-------------|-----------------------|------|---------|
| In1 | Add time 1 | Input  | Add time 1  | Depends on data type. | ns   | T#0s    |
| In2 | Add time 2 |        | Add time 2  |                       |      |         |
| Out | Total time | Output | Total time  | Depends on data type. | ns   | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



### Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*, and the addition will be processed as shown in the examples below.

- $T\#106751d\_23h\_47m\_16s\_854.775807ms + T\#0.000001ms$   
→  $T\#-106751d\_23h\_47m\_16s\_854.775808ms$
- $T\#-106751d\_23h\_47m\_16s\_854.775808ms + T\#-0.000001ms$   
→  $T\#106751d\_23h\_47m\_16s\_854.775807ms$

# ADD\_TOD\_TIME

The ADD\_TOD\_TIME instruction adds a time to a time of day.

| Instruction  | Name                    | FB/<br>FUN | Graphic expression | ST expression                |
|--------------|-------------------------|------------|--------------------|------------------------------|
| ADD_TOD_TIME | Add Time to Time of Day | FUN        |                    | Out:=ADD_TOD_TIME(In1, In2); |

## Variables

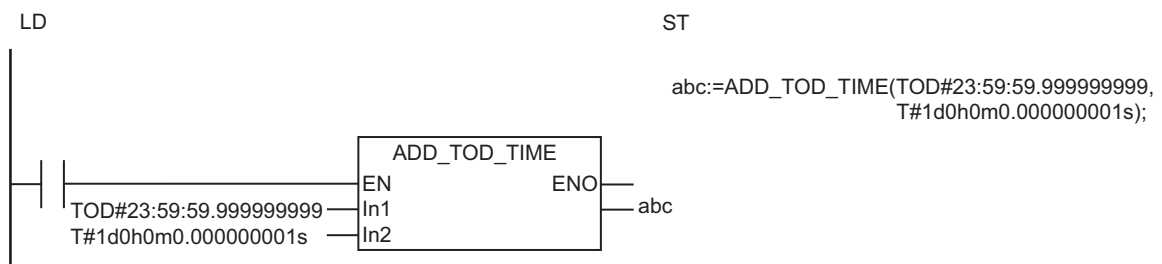
|     | Meaning               | I/O    | Description           | Valid range           | Unit                   | Default   |
|-----|-----------------------|--------|-----------------------|-----------------------|------------------------|-----------|
| In1 | Add time of day       | Input  | Add time 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 | ---       |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



|         |                        |
|---------|------------------------|
| In1     | TOD#23:59:59.999999999 |
| + In2   | T#1d0h0m0.000000001s   |
| <hr/>   |                        |
| Out=abc | TOD#0:0:0.000000000    |

### Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*, and the addition will be processed as shown in the examples below.

- 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/<br>FUN | Graphic expression | ST expression               |
|-------------|---------------------------|------------|--------------------|-----------------------------|
| ADD_DT_TIME | Add Time to Date and Time | FUN        |                    | Out:=ADD_DT_TIME(In1, In2); |

## Variables

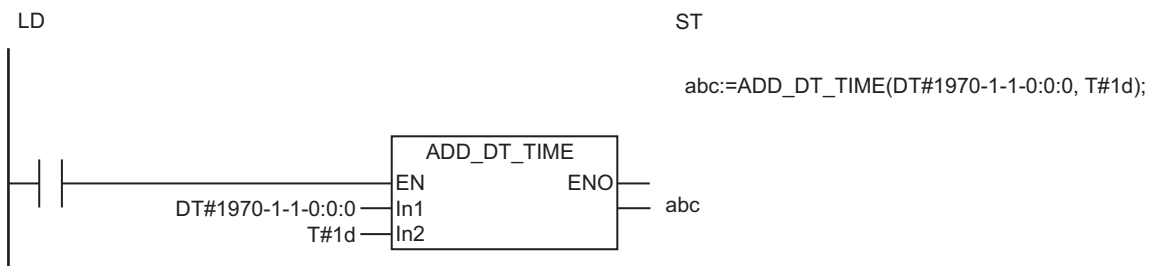
|     | Meaning                       | I/O    | Description                   | Valid range           | Unit                                     | Default           |
|-----|-------------------------------|--------|-------------------------------|-----------------------|--|-------------------|
| In1 | 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.



|         |                   |
|---------|-------------------|
| In1     | DT#1970-1-1-0:0:0 |
| + In2   | T#1d              |
|         |                   |
| Out=abc | 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

An error will not occur even if the addition result exceeds the valid range of *Out*, and the addition will be processed as shown in the examples below.

- 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 a time from another time.

| Instruction | Name          | FB/<br>FUN | Graphic expression | ST expression            |
|-------------|---------------|------------|--------------------|--------------------------|
| SUB_TIME    | Subtract Time | FUN        |                    | Out:=SUB_TIME(In1, In2); |

## Variables

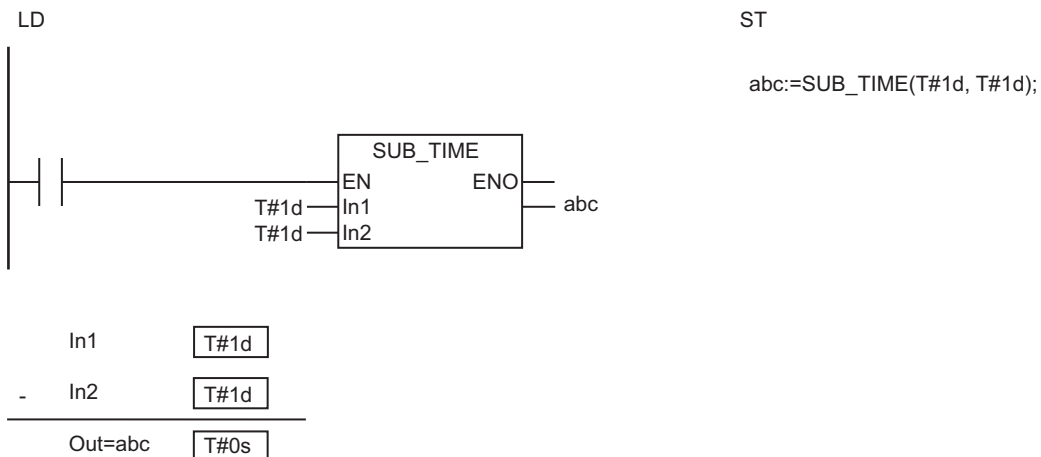
|     | Meaning          | I/O    | Description      | Valid range           | Unit | Default |
|-----|------------------|--------|------------------|-----------------------|------|---------|
| In1 | Original time    | Input  | Original time    | Depends on data type. | ns   | T#0s    |
| In2 | Time to subtract |        | Time to subtract |                       |      |         |
| Out | Resulting time   | Output | Resulting time   | Depends on data type. | ns   | ---     |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.



### Precautions for Correct Use

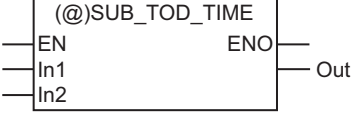
An error will not occur even if the subtraction result exceeds the valid range of *Out*, and the subtraction will be processed as shown in the examples below.

- $T\#106751d\_23h\_47m\_16s\_854.775807ms - T\#0.000001ms$   
→  $T\#-106751d\_23h\_47m\_16s\_854.775808ms$
- $T\#-106751d\_23h\_47m\_16s\_854.775808ms - T\#0.000001ms$   
→  $T\#106751d\_23h\_47m\_16s\_854.775807ms$



# SUB\_TOD\_TIME

The SUB\_TOD\_TIME instruction subtracts a time from a time of day.

| Instruction  | Name                           | FB/<br>FUN | Graphic expression   | ST expression                |
|--------------|--------------------------------|------------|--|------------------------------|
| SUB_TOD_TIME | Subtract Time from Time of Day | FUN        |  | Out:=SUB_TOD_TIME(In1, In2); |

## Variables

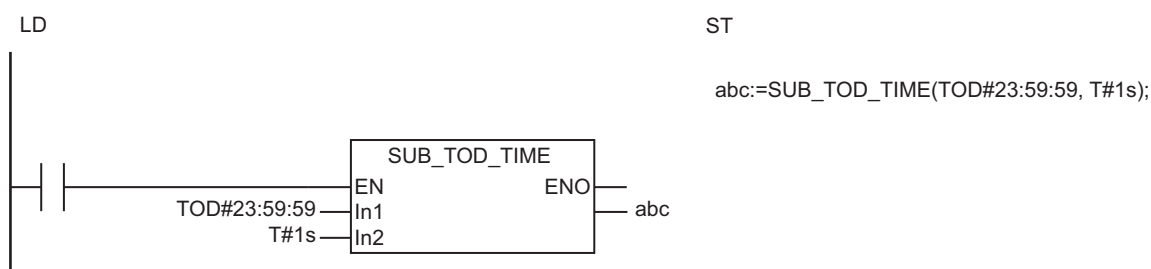
|     | Meaning               | I/O    | Description           | Valid range           | Unit                   | Default   |
|-----|-----------------------|--------|-----------------------|-----------------------|------------------------|-----------|
| In1 | Time of day           | Input  | Time of 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 | ---       |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



|         |              |
|---------|--------------|
| In1     | TOD#23:59:59 |
| - In2   | T#1s         |
| <hr/>   |              |
| 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*, and the subtraction will be processed as shown in the examples below.

- 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/<br>FUN | Graphic expression | ST expression               |
|-------------|----------------------|------------|--------------------|-----------------------------|
| SUB_TOD_TOD | Subtract Time of Day | FUN        |                    | Out:=SUB_TOD_TOD(In1, In2); |

## Variables

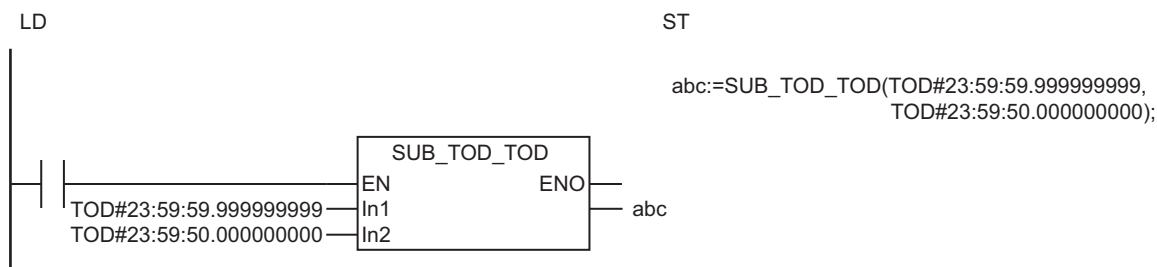
|     | Meaning        | I/O    | Description    | Valid range           | Unit                   | Default   |
|-----|----------------|--------|----------------|-----------------------|------------------------|-----------|
| In1 | Time of day 1  | Input  | Time of day 1  | Depends on data type. | Hour, 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 a time of day *In2* from another time of day *In1*. The result of subtraction in *Out* is a time.

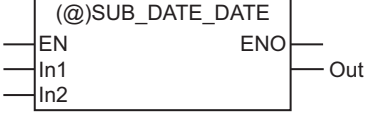
The following example is for when *In1* is TOD#23:59:59.999999999 and *In2* is TOD#23:59:50.000000000.



|         |                        |
|---------|------------------------|
| In1     | TOD#23:59:59.999999999 |
| - In2   | TOD#23:59:50.000000000 |
| <hr/>   |                        |
| Out=abc | T#0d0h0m9.999999999s   |

# SUB\_DATE\_DATE

The SUB\_DATE\_DATE instruction subtracts a date from another date.

| Instruction   | Name          | FB/<br>FUN | Graphic expression   | ST expression                 |
|---------------|---------------|------------|--|-------------------------------|
| SUB_DATE_DATE | Subtract Date | FUN        |  | Out:=SUB_DATE_DATE(In1, In2); |

2

SUB\_DATE\_DATE

## Variables

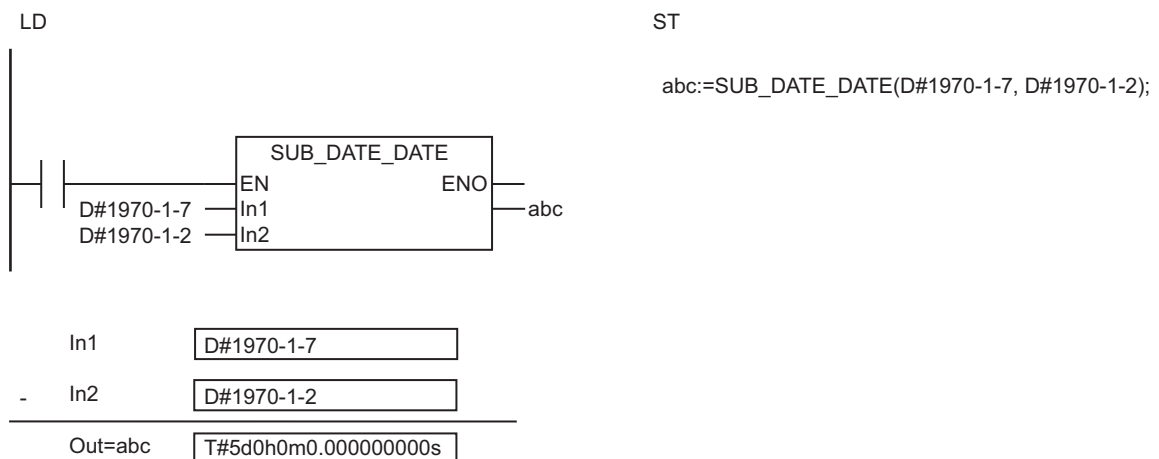
|     | Meaning        | I/O    | Description    | Valid range           | Unit             | Default     |
|-----|----------------|--------|----------------|-----------------------|------------------|-------------|
| In1 | Date 1         | Input  | Date 1         | Depends on data type. | Year, month, day | DT#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.



# SUB\_DT\_DT

The SUB\_DT\_DT instruction subtracts a date and time from another date and time.

| Instruction | Name                   | FB/<br>FUN | Graphic expression | ST expression             |
|-------------|------------------------|------------|--------------------|---------------------------|
| SUB_DT_DT   | Subtract Date and Time | FUN        |                    | Out:=SUB_DT_DT(In1, In2); |

## Variables

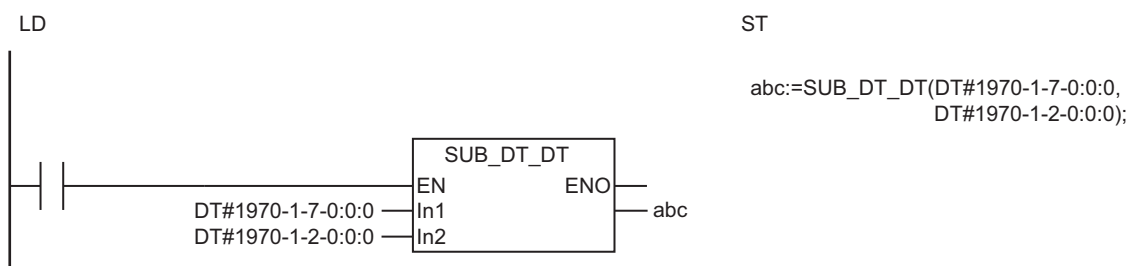
|     | Meaning         | I/O    | Description     | Valid range           | Unit                                     | Default           |
|-----|-----------------|--------|-----------------|-----------------------|--|-------------------|
| In1 | Date and time 1 | Input  | Date and time 1 | Depends 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.



|         |  |
|---------|--|
| In1     | <input type="text" value="DT#1970-1-7-0:0:0"/> |
| - In2   | <input type="text" value="DT#1970-1-2-0:0:0"/> |
| <hr/>   |  |
| Out=abc | <input type="text" value="T#5d"/>              |

### 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/<br>FUN | Graphic expression | ST expression               |
|-------------|----------------------------------|------------|--------------------|-----------------------------|
| SUB_DT_TIME | Subtract Time from Date and Time | FUN        |                    | Out:=SUB_DT_TIME(In1, In2); |

## Variables

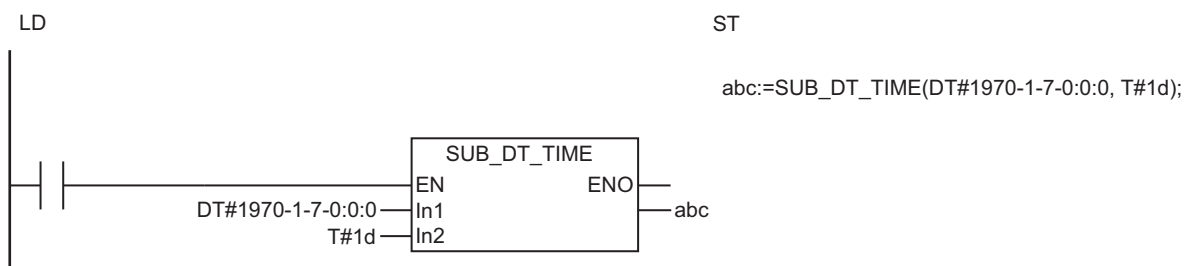
|     | Meaning                 | I/O    | Description             | Valid range           | Unit                                     | Default           |
|-----|-------------------------|--------|-------------------------|-----------------------|--|-------------------|
| In1 | Date and time           | 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.





|         |  |
|---------|--|
| In1     | <input type="text" value="DT#1970-1-7-0:0:0"/> |
| - In2   | <input type="text" value="T#1d"/>              |
| <hr/>   |  |
| Out=abc | <input type="text" value="DT#1970-1-6-0:0:0"/> |

## 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

An error will not occur even if the subtraction result exceeds the valid range of *Out*, and the subtraction will be processed as shown in the examples below.

- 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/<br>FUN | Graphic expression | ST expression           |
|-------------|---------------|------------|--------------------|-------------------------|
| MULTIME     | Multiply Time | FUN        |                    | Out:=MULTIME(In1, In2); |

## Variables

|     | Meaning        | I/O    | Description    | Valid range           | Unit | Default |
|-----|----------------|--------|----------------|-----------------------|------|---------|
| In1 | Original time  | Input  | Original time  | Depends on data type. | ns   | T#0s    |
| In2 | Multiplier     |        | Multiplier     |                       | ---  | *1      |
| Out | Resulting time | Output | Resulting time | Depends on data type. | ns   | ---     |

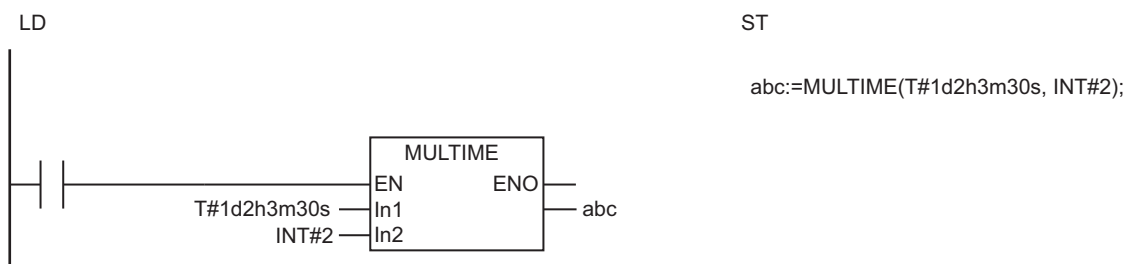
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num<br>bers |      | Times, durations,<br>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.



|         |             |
|---------|-------------|
| In1     | T#1d2h3m30s |
| x In2   | INT#2       |
| Out=abc | T#2d4h7m    |

## Precautions for Correct Use

- If *In2* is a real number, the multiplication result is rounded to the nearest nanosecond.
- If the value of *In2* is 0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

| Value of <i>In2</i> | Value of <i>Out</i> |                               |
|---------------------|---------------------|-------------------------------|
|                     | NX1P2               | Other than the left.          |
| 0                   | T#0s                | T#0s                          |
| +∞                  | T#-0d0h0m0s1e-6ms   | T#-106751d23h47m16.854775808s |
| -∞                  | T#-0d0h0m0s1e-6ms   | T#-106751d23h47m16.854775808s |
| Nonnumeric data     | T#0s                | T#-106751d23h47m16.854775808s |

- An error will not occur even if the multiplication result exceeds the valid range of *Out*, and the multiplication will be processed as shown in the examples below.
  - a) T#53375d\_23h\_53m\_38s\_427.387904ms \* USINT#2  
→ T#-106751d\_23h\_47m\_16s\_854.775808ms
  - b) T#-53375d\_23h\_53m\_38s\_427.387905ms \* USINT#2  
→ T#106751d\_23h\_47m\_16s\_854.775806ms

## Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | 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<br>2.50 → 2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2             |

# DIVTIME

The DIVTIME instruction divides a time by a specified number.

| Instruction | Name        | FB/<br>FUN | Graphic expression | ST expression           |
|-------------|-------------|------------|--------------------|-------------------------|
| DIVTIME     | Divide Time | FUN        |                    | Out:=DIVTIME(In1, In2); |

## Variables

|     | 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 |        | Number to divide by |                       | ---  | *1      |
| Out | Resulting time      | Output | Resulting time      | Depends on data type. | ns   | ---     |

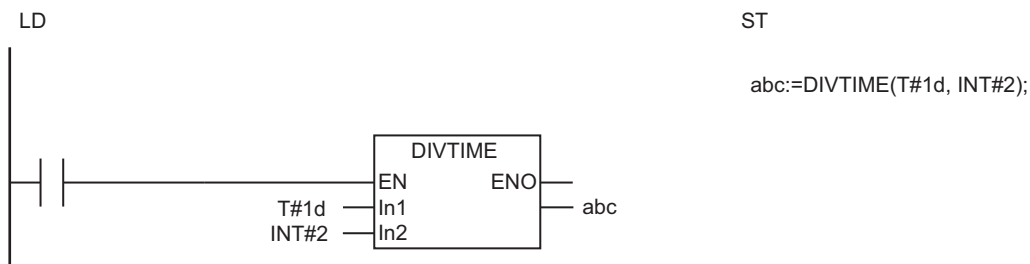
\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num-<br>bers |      | Times, durations,<br>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.



|         |       |
|---------|-------|
| 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 <i>In2</i> | Value of <i>Out</i>  |                               |
|---------------------|----------------------|-------------------------------|
|                     | NX1P2                | Other than the left.          |
| 0                   | T#0d_0h_0m_0s_1e-006 | T#-106751d23h47m16.854775808s |
| $+\infty$           | T#0s                 | T#0s                          |
| $-\infty$           | T#0s                 | T#0s                          |
| Nonnumeric data     | T#0s                 | T#-106751d23h47m16.854775808s |

- 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.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - In2* is an integer with a value of 0.

## Rounding Off

The following table shows how values are rounded.

| Value of fractional part | Description   | 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<br>2.50 → 2 |
| Greater than 0.5         | The fractional part is rounded up.  | 1.51 → 2             |



|         |   |
|---------|---|
| In1     | <input type="text" value="D#1970-1-7"/>                     |
| + In2   | <input type="text" value="TOD#23:59:59.999999999"/>         |
| <hr/>   |   |
| Out=abc | <input type="text" value="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. |


## Precautions for Correct Use

An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of combined date and time exceeds the valid value 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 when they exceed the valid range of *Out*).

# DT\_TO\_TOD

The DT\_TO\_TOD instruction extracts the time of day from a date and time.

| Instruction | Name                                   | FB/<br>FUN | Graphic expression  | ST expression       |
|-------------|--|------------|---|---------------------|
| DT_TO_TOD   | Extract Time of Day from Date and Time | FUN        |  | Out:=DT_TO_TOD(In); |

## Variables

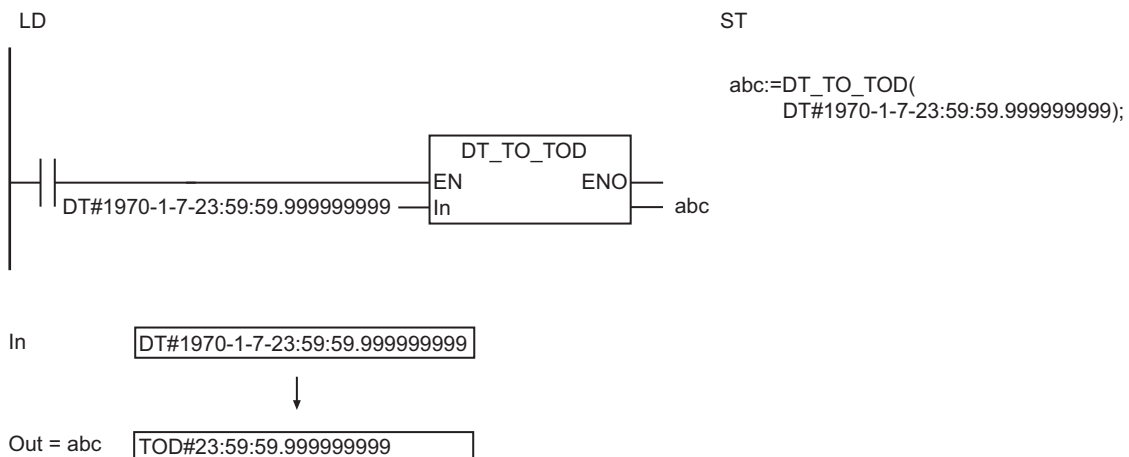
|     | Meaning       | I/O    | Description   | Valid range           | Unit                                     | Default           |
|-----|---------------|--------|---------------|-----------------------|--|-------------------|
| In  | Date and time | Input  | Date and time | Depends on data type. | 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                   | ---               |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.





**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/<br>FUN | Graphic expression | ST expression        |
|-------------|---------------------------------|------------|--------------------|----------------------|
| DT_TO_DATE  | Extract Date from Date and Time | FUN        |                    | Out:=DT_TO_DATE(In); |

## Variables

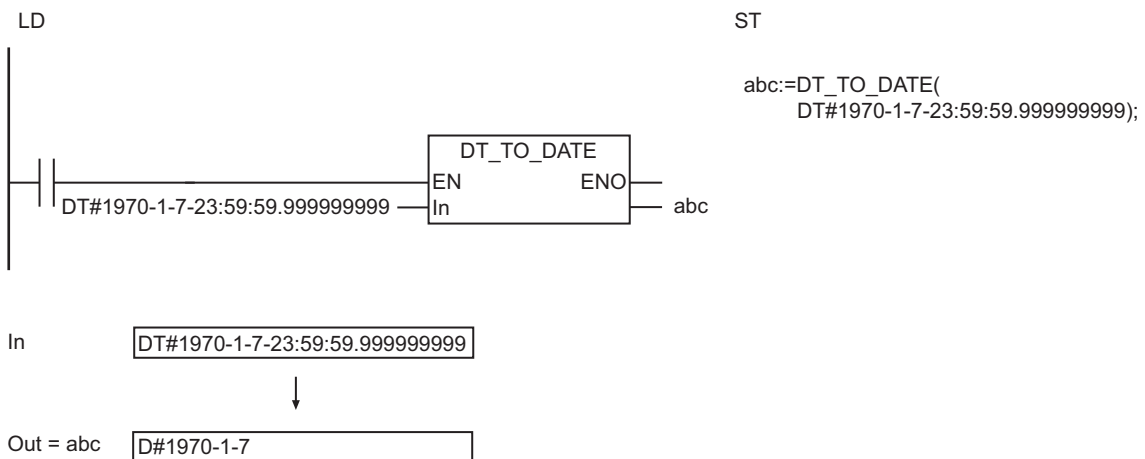
|     | Meaning       | I/O    | Description   | Valid range           | Unit                                     | Default           |
|-----|---------------|--------|---------------|-----------------------|--|-------------------|
| In  | Date and time | Input  | Date and time | Depends on data type. | Year, month, day, hour, minutes, seconds | DT#1970-1-1-0:0:0 |
| Out | Date          | Output | Date          | Depends on data type. | Year, month, day                         | ---               |

|     | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



**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. |

# SetTime

The SetTime instruction sets the system time.

| Instruction | Name     | FB/<br>FUN | Graphic expression | ST expression |
|-------------|----------|------------|--------------------|---------------|
| SetTime     | Set Time | FUN        |                    | SetTime(In);  |

## Variables

|     | Meaning      | I/O    | Description                     | Valid range | Unit                                     | Default           |
|-----|--------------|--------|---------------------------------|-------------|--|-------------------|
| In  | Time data    | Input  | Current time to set system time | *1          | Year, month, day, hour, minutes, seconds | DT#1970-1-1-0:0:0 |
| Out | Return value | Output | Always TRUE                     | TRUE only   | ---                                      | ---               |

\*1. 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).

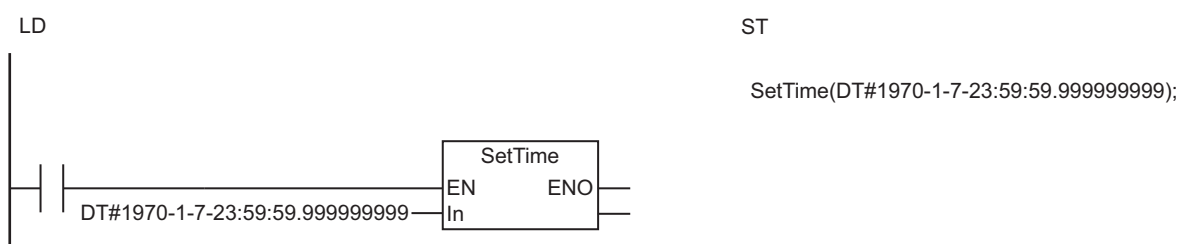
The valid range for an NJ-series CPU Unit is DT#1970-01-01-00:00:00.000000000 to DT#2106-02-06-23:59:59.999999999 (0:00:000000000 on January 1, 1970 to 23:59:59.999999999 on February 6, 2106).

|     | Boolean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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      |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

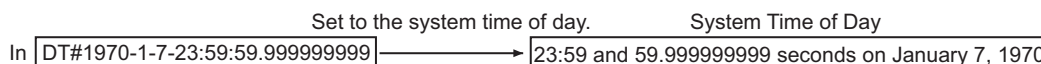
## Function

The SetTime instruction sets the system time to date and time *In*.

The following programming example is for when *In* is DT#1970-1-7:23:59:59.999999999.



The SetTime instruction sets the value of *In* to the system time of day.



## 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

The following methods can also be used to set the system time.

- Sysmac Studio
- NTP function

## Precautions for Correct Use

- For *In*, specify the time for the set time zone (do not specify Greenwich mean time (GMT)).
- You cannot set *In* with a time that is lower than 1970-1-1-0:0:0.000000000 GMT.
- A time lag will occur when updating the internal time. If the time is read immediately after executing this instruction, the old time may be read.
- Return value *Out* is not used when this instruction is used in ST.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *In* is outside the valid range.
  - b) The value of *In* is below 1970-1-1-0:0:0.000000000 GMT.

# 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

|     | Meaning      | I/O    | Description  | Valid range | Unit                                     | Default |
|-----|--------------|--------|--------------|-------------|--|---------|
| Out | Current time | Output | Current time | *1          | Year, month, day, hour, minutes, seconds | ---     |

- \*1. 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).

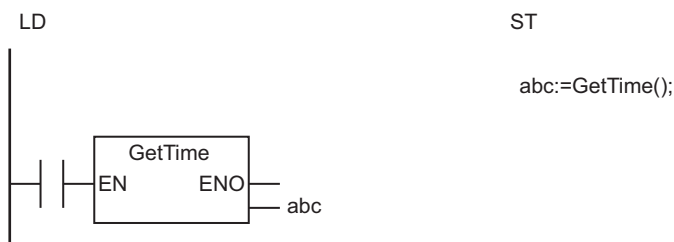
|     | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-bers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|----------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|---------------|-------|---|------|-----|----|--------|
|     | 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.

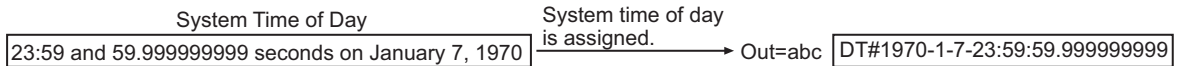
The current time of day is the time for the set time zone (not Greenwich mean time (GMT)).

The following figure shows a programming example. The current time is assigned to variable *abc*.



The GetTime instruction assigns the current time to **abc**.

For 23:59 and 59.999999999 seconds on January 7, 1970



## 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 instruction, *DtToSec* on page 2-664, 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 instruction, *DtToDateStruct* on page 2-692, to convert the current time of day to a date (year, month, day, minutes, and seconds).
- Use the instruction, *GetDayOfWeek* on page 2-688, 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/<br>FUN | Graphic expression | ST expression     |
|-------------|----------------------------------|------------|--------------------|-------------------|
| DtToSec     | Convert Date and Time to Seconds | FUN        |                    | Out:=DtToSec(In); |

## Variables

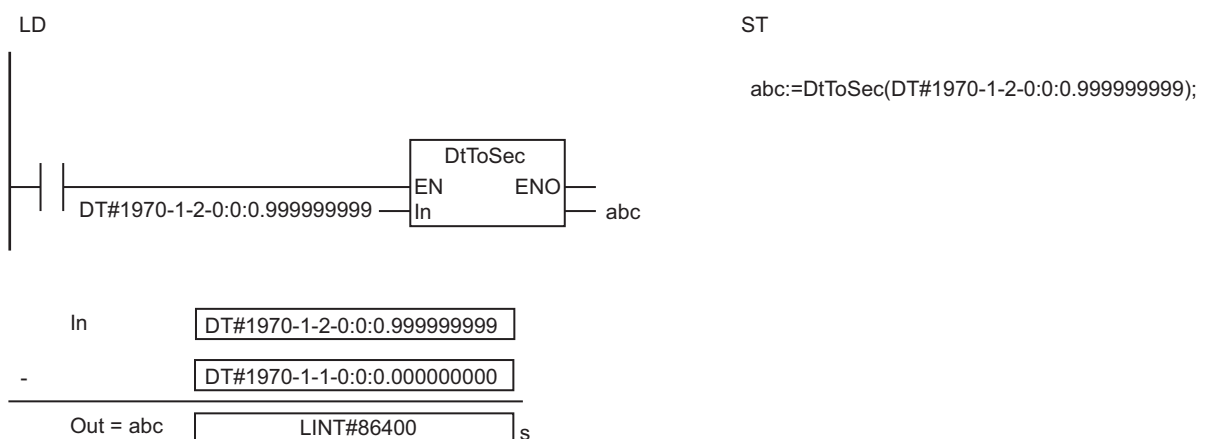
|     | Meaning       | I/O    | Description  | Valid range           | Unit                                     | Default           |
|-----|---------------|--------|--|-----------------------|--|-------------------|
| In  | Date and time | Input  | Date and time                                      | 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                                  | ---               |

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.





## Related System-defined Variables

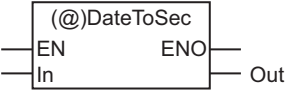
| Name         | Meaning            | Data type | Description  |
|--------------|--------------------|-----------|--|
| _CurrentTime | System Time of Day | DT        | The time of day from the system clock. The number of seconds from 00:00:00 on January 1, 1970. |

## Additional Information

Use the instruction, *SecToDt* on page 2-670, 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

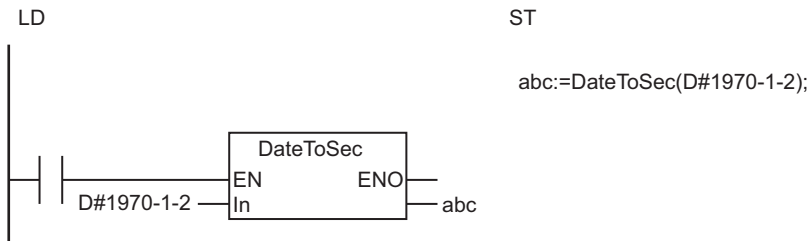
|     | Meaning | I/O    | Description  | Valid range           | Unit             | Default     |
|-----|---------|--------|--|-----------------------|------------------|-------------|
| In  | Date    | Input  | Date   | Depends on data type. | Year, month, day | DT#1970-1-1 |
| Out | Seconds | Output | Number of seconds from 00:00:00 on January 1, 1970 | 0 to 18446659200      | Seconds          | ---         |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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.



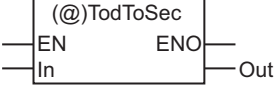
|           |                             |
|-----------|-----------------------------|
| In        | D#1970-1-2                  |
| -         | DT#1970-1-1-0:0:0.000000000 |
| Out = abc | LINT#86400 s                |

## Additional Information

Use the instruction, *SecToDate* on page 2-672, 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/<br>FUN | Graphic expression  | ST expression      |
|-------------|--------------------------------|------------|---|--------------------|
| TodToSec    | Convert Time of Day to Seconds | FUN        |  | Out:=TodToSec(In); |

## Variables

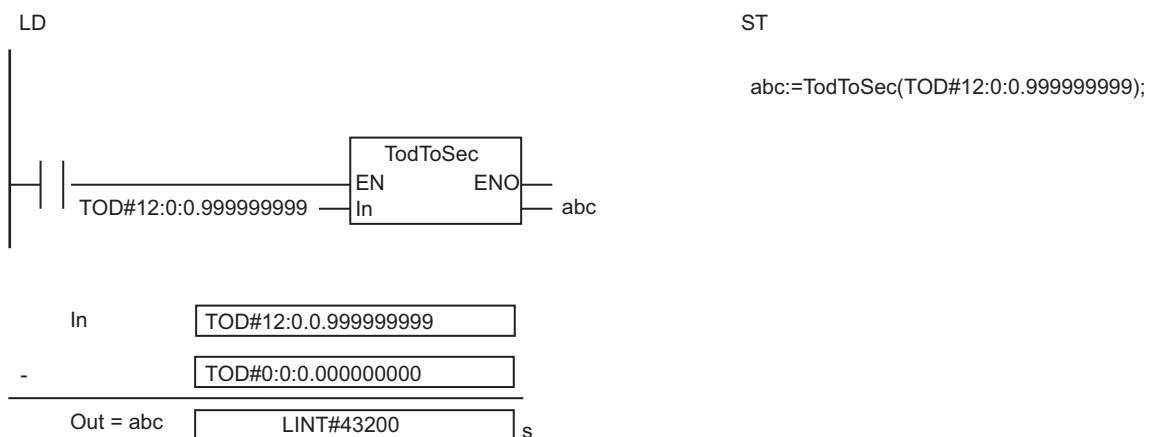
|     | Meaning     | I/O    | Description                     | Valid range           | Unit                   | Default   |
|-----|-------------|--------|---------------------------------|-----------------------|------------------------|-----------|
| In  | Time of day | Input  | Time of day                     | Depends on data type. | Hour, minutes, seconds | TOD#0:0:0 |
| Out | Seconds     | Output | Number of seconds from 00:00:00 | 0 to 86399            | Seconds                | ---       |

|     | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.

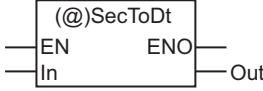


## Additional Information

Use the instruction, *SecToTod* on page 2-674, to convert the number of seconds from 00:00:00 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

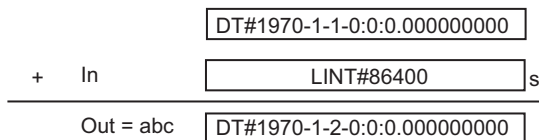
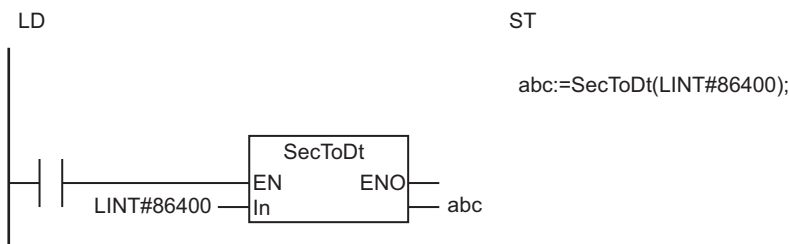
|     | Meaning       | I/O    | Description  | Valid range           | Unit                                     | Default |
|-----|---------------|--------|--|-----------------------|--|---------|
| In  | Seconds       | Input  | Number of seconds from 00:00:00 on January 1, 1970 | 0 to 18446744073      | Seconds                                  | 0       |
| Out | Date and time | Output | Date and time                                      | Depends on data type. | Year, month, day, hour, minutes, seconds | ---     |

|     | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



## 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 instruction, *DtToSec* on page 2-664, 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside the valid range.

# SecToDate

The SecToDate instruction converts the number of seconds from 00:00:00 on January 1, 1970 to a date.

| Instruction | Name                    | FB/<br>FUN | Graphic expression  | ST expression       |
|-------------|-------------------------|------------|---|---------------------|
| SecToDate   | Convert Seconds to Date | FUN        |  | Out:=SecToDate(In); |

## Variables

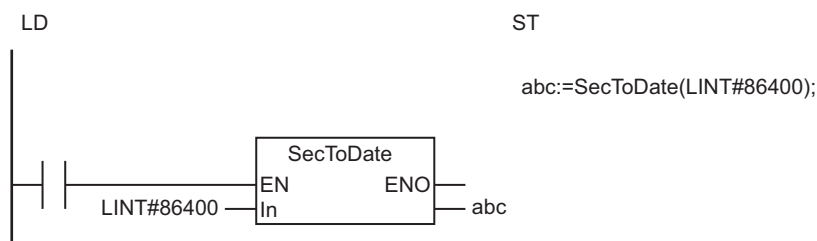
|     | Meaning | I/O    | Description  | Valid range           | Unit             | Default |
|-----|---------|--------|--|-----------------------|------------------|---------|
| In  | Seconds | Input  | Number of seconds from 00:00:00 on January 1, 1970 | 0 to 18446744073      | Seconds          | 0       |
| Out | Date    | 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 00:00:0000:00:00 on January 1, 1970 in *In* to a date. The value is truncated below date.

The following example is for when *In* is LINT#86400.



|   |           |            |
|---|-----------|------------|
|   |           | D#1970-1-1 |
| + | In        | LINT#86400 |
|   |           | s          |
|   | Out = abc | D#1970-1-2 |



## Additional Information

Use the instruction, *DateToSec* on page 2-666, to convert a date to the number of seconds from 00:00:00 on January 1, 1970.

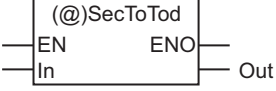
## Precautions for Correct Use

An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside the valid range.

# SecToTod

The SecToTod instruction converts the number of seconds from 00:00:00 to a time of day.

| Instruction | Name                                   | FB/<br>FUN | Graphic expression  | ST expression      |
|-------------|--|------------|---|--------------------|
| SecToTod    | Convert Sec-<br>onds to Time of<br>Day | FUN        |  | Out:=SecToTod(In); |

## Variables

|     | Meaning     | I/O    | Description                     | Valid range             | Unit                   | Default |
|-----|-------------|--------|---------------------------------|-------------------------|------------------------|---------|
| In  | Seconds     | Input  | Number of seconds from 00:00:00 | Depends on data type.*1 | Seconds                | 0       |
| Out | Time of day | Output | Time of day                     | Depends on data type.   | Hour, minutes, seconds | ---     |

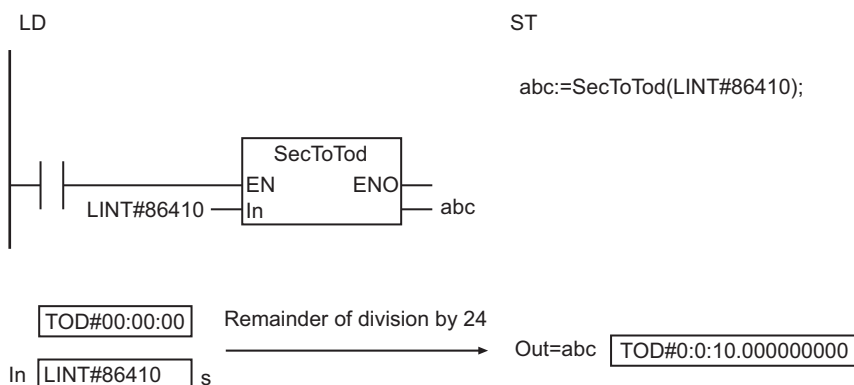
\*1. Negative numbers are excluded.

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.



## Additional Information

Use the instruction, *TodToSec* on page 2-668, to convert a time of day to the number of seconds from 00:00:00.

## Precautions for Correct Use

An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside the valid range.

# TimeToNanoSec

The TimeToNanoSec instruction converts a time to nanoseconds.

| Instruction   | Name                        | FB/<br>FUN | Graphic expression | ST expression           |
|---------------|-----------------------------|------------|--------------------|-------------------------|
| TimeToNanoSec | Convert Time to Nanoseconds | FUN        |                    | Out:=TimeToNanoSec(In); |

## Variables

|     | Meaning     | I/O    | Description | Valid range           | Unit | Default |
|-----|-------------|--------|-------------|-----------------------|------|---------|
| In  | Time        | Input  | Time        | Depends on data type. | ns   | T#0s    |
| Out | Nanoseconds | Output | Nanoseconds | *1                    | ns   | ---     |

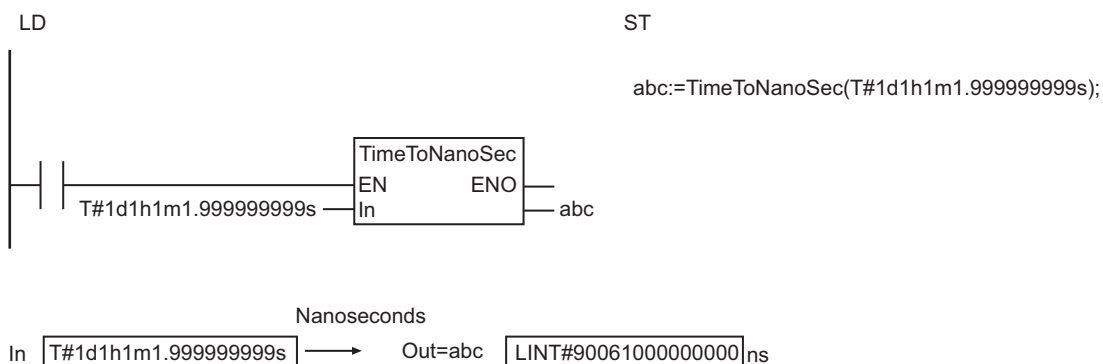
\*1. -9223372036854775808 to 9223372036854775807

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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.



## Additional Information

Use the instruction, *NanoSecToTime* on page 2-679, to convert nanoseconds to a time.

# TimeToSec

The TimeToSec instruction converts a time to seconds.

| Instruction | Name                    | FB/<br>FUN | Graphic expression | ST expression       |
|-------------|-------------------------|------------|--------------------|---------------------|
| TimeToSec   | Convert Time to Seconds | FUN        |                    | Out:=TimeToSec(In); |

## Variables

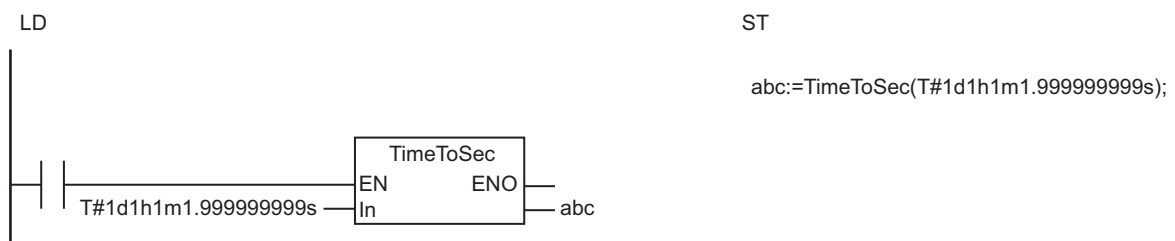
|     | Meaning | I/O    | Description | Valid range               | Unit    | Default |
|-----|---------|--------|-------------|---------------------------|---------|---------|
| In  | Time    | Input  | Time        | Depends on data type.     | ns      | T#0s    |
| Out | Seconds | Output | Seconds     | -9223372036 to 9223372036 | Seconds | ---     |

|     | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|-----|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|     | Boo<br>lean | 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.



In T#1d1h1m1.999999999s  $\xrightarrow{\text{Seconds}}$  Out=abc LINT#90061s

## Additional Information

Use the instruction, *SecToTime* on page 2-680, 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/<br>FUN | Graphic expression | ST expression           |
|---------------|------------------------------|------------|--------------------|-------------------------|
| NanoSecToTime | Convert Nano-seconds to Time | FUN        |                    | Out:=NanoSecToTime(In); |

## Variables

|     | Meaning     | I/O    | Description | Valid range           | Unit | Default |
|-----|-------------|--------|-------------|-----------------------|------|---------|
| In  | Nanoseconds | Input  | Nanoseconds | *1                    | ns   | 0       |
| Out | Time        | Output | Time        | Depends on data type. | ns   | ---     |

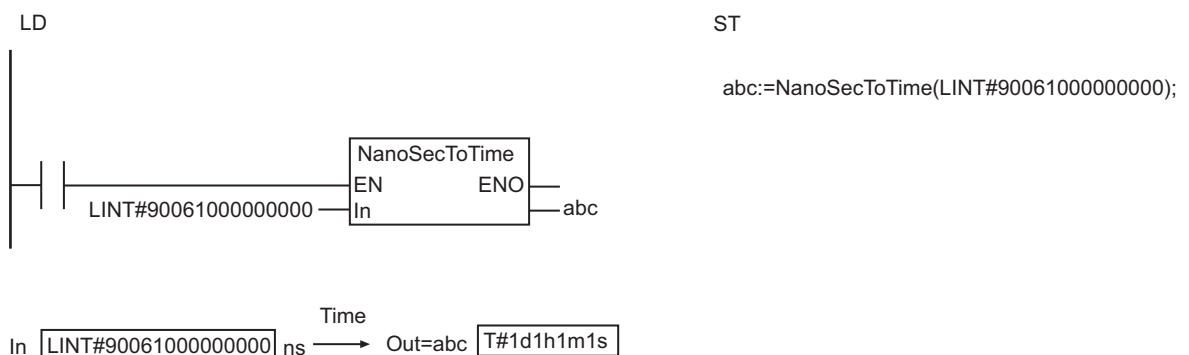
\*1. -9223372036854775808 to 9223372036854775807

|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.

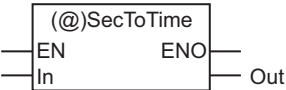


## Additional Information

Use the instruction, *TimeToNanoSec* on page 2-676, to convert a time to nanoseconds.

# SecToTime

The SecToTime instruction converts seconds to a time.

| Instruction | Name                    | FB/<br>FUN | Graphic expression  | ST expression       |
|-------------|-------------------------|------------|---|---------------------|
| SecToTime   | Convert Seconds to Time | FUN        |  | Out:=SecToTime(In); |

## Variables

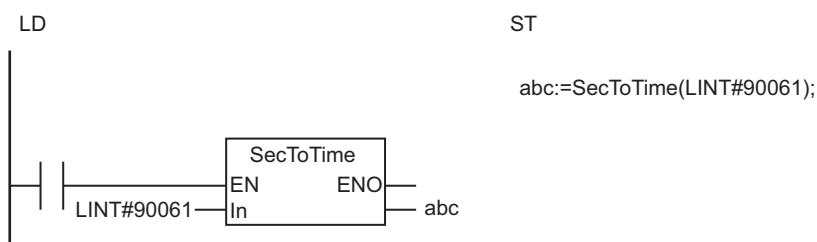
|     | Meaning | I/O    | Description | Valid range               | Unit    | Default |
|-----|---------|--------|-------------|---------------------------|---------|---------|
| In  | Seconds | Input  | Seconds     | -9223372036 to 9223372036 | Seconds | 0       |
| Out | Time    | Output | Time        | Depends on data type.     | ns      | ---     |

|     | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



In LINT#90061 s  $\xrightarrow{\text{Time}}$  Out=abc T#1d1h1m1s

## Additional Information

Use the instruction, *TimeToSec* on page 2-677, to convert a time to seconds.



## Precautions for Correct Use

- *In* is in seconds. *Out* is in nanoseconds.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *In* is outside the valid range.

# ChkLeapYear

The ChkLeapYear instruction checks if a specified year is a leap year.

| Instruction | Name                | FB/<br>FUN | Graphic expression | ST expression         |
|-------------|---------------------|------------|--------------------|-----------------------|
| ChkLeapYear | Check for Leap Year | FUN        |                    | Out:=ChkLeapYear(In); |

## Variables

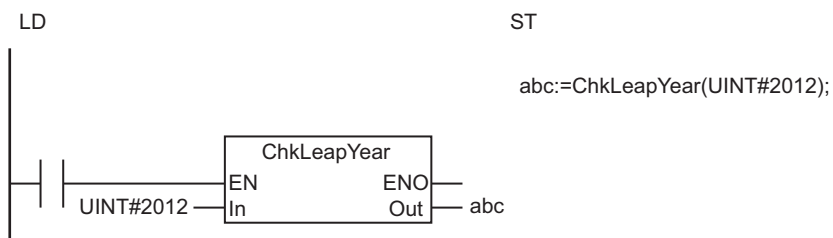
|     | Meaning | I/O    | Description                             | Valid range           | Unit | Default |
|-----|---------|--------|---|-----------------------|------|---------|
| In  | Year    | Input  | Year                                    | 1970 to 2554          | Year | 1970    |
| Out | Result  | Output | TRUE: Leap year<br>FALSE: Not leap year | Depends on data type. | ---  | ---     |

|     | Boolean |      |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.



## 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 a specified month.

| Instruction         | Name                 | FB/<br>FUN | Graphic expression | ST expression                        |
|---------------------|----------------------|------------|--------------------|--------------------------------------|
| GetDaysOf-<br>Month | Get Days in<br>Month | FUN        |                    | Out:=GetDaysOfMonth(Year,<br>Month); |

## Variables

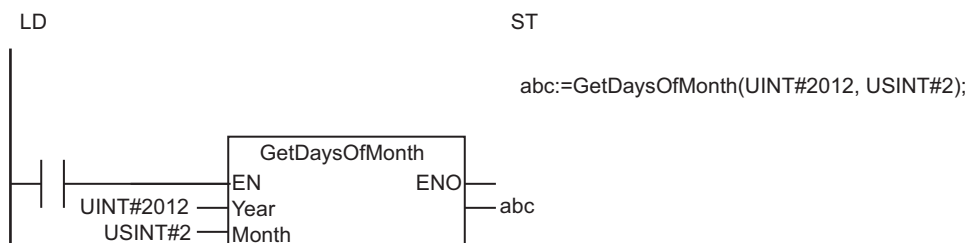
|       | Meaning | I/O    | Description | Valid range  | Unit  | Default |
|-------|---------|--------|-------------|--------------|-------|---------|
| Year  | Year    | Input  | Year        | 1970 to 2554 | Year  | 1970    |
| Month | Month   |        | Month       | 1 to 12      | Month | 1       |
| Out   | Days    | Output | Days        | 28 to 31     | Days  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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`.



## 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Month* is outside the valid range.

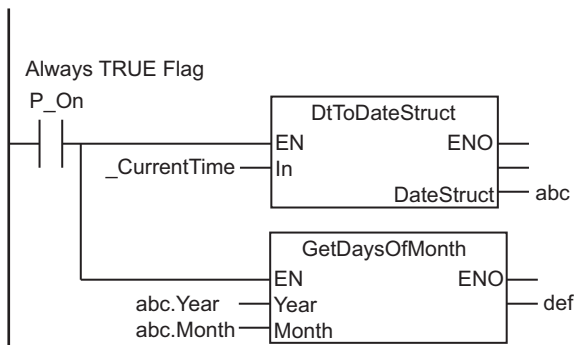
## Sample Programming

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 | <input checked="" type="checkbox"/> | 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 | <input checked="" type="checkbox"/> | 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/<br>FUN | Graphic expression | ST expression                 |
|-------------|-----------------------|------------|--------------------|-------------------------------|
| DaysToMonth | Convert Days to Month | FUN        |                    | Out:=DaysToMonth(Year, Days); |

## Variables

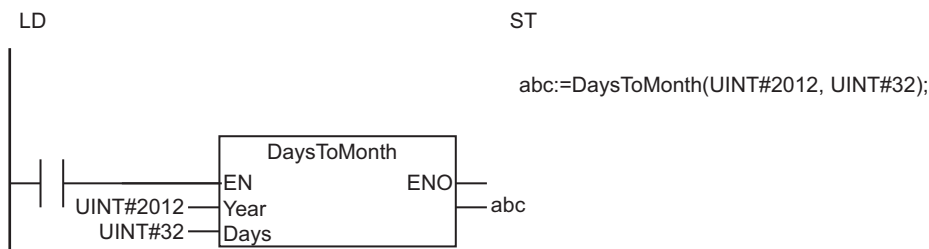
|      | Meaning | I/O    | Description                   | Valid range   | Unit  | Default |
|------|---------|--------|-------------------------------|---|-------|---------|
| Year | Year    | Input  | Year                          | 1970 to 2554  | Year  | 1970    |
| Days | Days    |        | Number of days from January 1 | 1 to 365<br>1 to 366 when <i>Year</i> is a leap year. | Days  | 1       |
| Out  | Month   | Output | Month                         | 1 to 12   | Month | ---     |

|      | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.



## 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 will occur in the following case. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Days* is outside the valid range.

# GetDayOfWeek

The GetDayOfWeek instruction gets the day of the week for a specified date (year, month, and day).

| Instruction       | Name               | FB/<br>FUN | Graphic expression | ST expression          |
|-------------------|--------------------|------------|--------------------|------------------------|
| GetDayOf-<br>Week | Get Day of<br>Week | FUN        |                    | Out:=GetDayOfWeek(In); |

## Variables

|     | Meaning          | I/O    | Description      | Valid range                              | Unit             | Default |
|-----|------------------|--------|------------------|--|------------------|---------|
| In  | Year, month, day | Input  | Year, month, day | Depends on data type.                    | Year, month, day | *1      |
| Out | Day of the week  | Output | Day of the week  | _MON, _TUE, _WED, _THU, _FRI, _SAT, _SUN | Day of the week  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-688 for the enumerators for the enumerated type <code>_eDAYOFWEEK</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

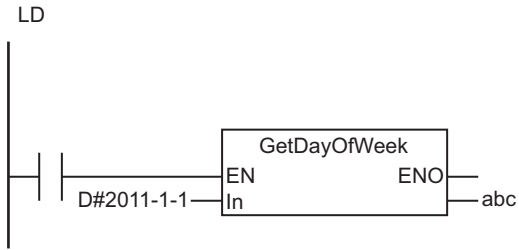
The GetDayOfWeek instruction gets the day of the week for the year, month, and day of month specified in *In*.

The data type of *Out* is enumerated type `_eDAYOFWEEK`. The meanings of the enumerators are as follows:

| Enumerator        | Meaning   |
|-------------------|-----------|
| <code>_MON</code> | Monday    |
| <code>_TUE</code> | Tuesday   |
| <code>_WED</code> | Wednesday |
| <code>_THU</code> | Thursday  |
| <code>_FRI</code> | Friday    |
| <code>_SAT</code> | Saturday  |
| <code>_SUN</code> | Sunday    |

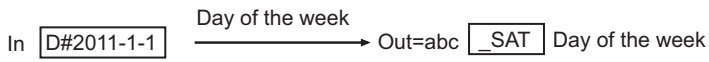
The following example is for when *In* is `D#2011-1-1`.





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. |



In  <sup>Week number</sup> → Out=abc Week

## 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/<br>FUN | Graphic expression | ST expression                        |
|----------------|--------------------------|------------|--------------------|--------------------------------------|
| DtToDateStruct | Break Down Date and Time | FUN        |                    | Out:=DtToDateStruct(In, DateStruct); |

## Variables

|            | 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#197<br>0-1-1-0:<br>0:0 |
| Out        | Return value  | Output | Always TRUE  | TRUE only             | ---                                      | ---                       |
| DateStruct | Date and time |        | Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds | ---                   |  |                           |

|            | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-692 for details on the structure <code>_sDT</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

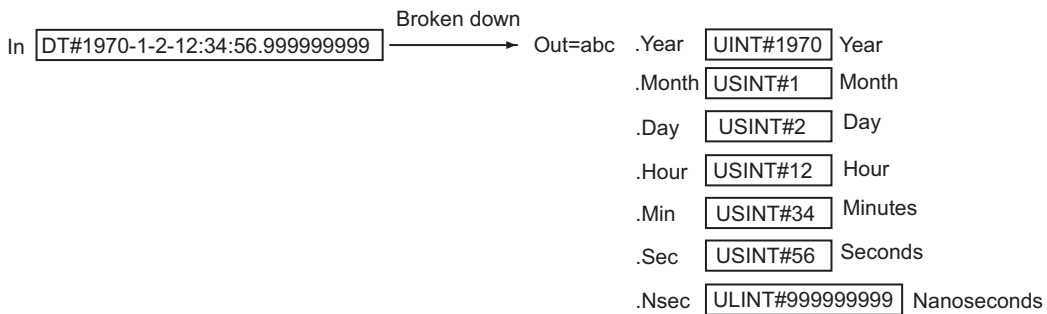
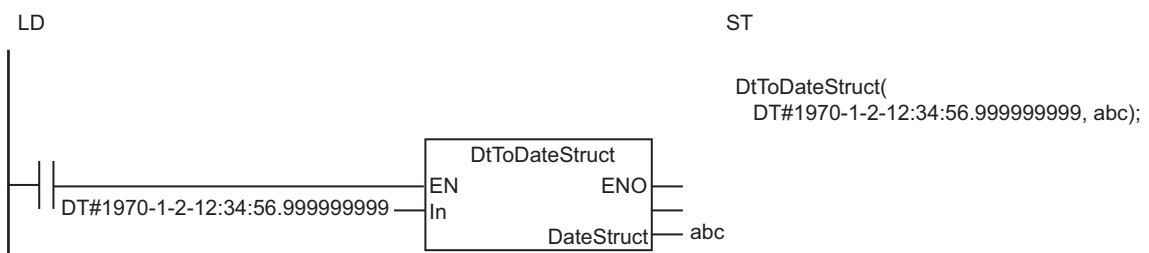
## Function

This instruction converts the date and time in *In* to the year, month, day, hour, minutes, seconds, and nanoseconds.

The data type of the output variable, *DateStruct*, is the structure `_sDT`. The meanings of the members are as follows:

| Name       | Meaning       | Content  | Data type | Valid range    | Unit        | Default |
|------------|---------------|--|-----------|----------------|-------------|---------|
| DateStruct | 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        | ---     |
| 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.

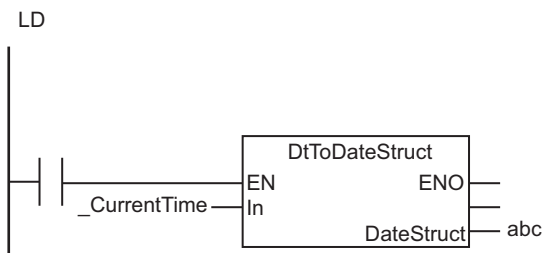


### Related System-defined Variables

| 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 instruction, *DateStructToDt* on page 2-695, 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.



ST

```
DtToDateStruct(_CurrentTime, abc);
```

## Precautions for Correct Use

Return value *Out* is not used when this 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/<br>FUN | Graphic expression | ST expression            |
|----------------|-----------|------------|--------------------|--------------------------|
| DateStructToDt | Join Time | FUN        |                    | Out:=DateStructToDt(In); |

## Variables

|     | 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 | ---     |

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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  |             |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |    |
| 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:





## Precautions for Correct Use

An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of a member of *In* is outside the valid range.
- The processing result exceeds the valid range of *Out*.

# TruncTime

The TruncTime instruction truncates a TIME variable to a specified time unit.

| Instruction | Name          | FB/<br>FUN | Graphic expression | ST expression                 |
|-------------|---------------|------------|--------------------|-------------------------------|
| TruncTime   | Truncate Time | FUN        |                    | Out:=TruncTime(In, Accuracy); |



## 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.

## Variables

|          | 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 | ---  | _NANOSEC |
| Out      | Time after truncation          | Output | Time after truncation                            | Depends on data type.                | ns   | ---      |

|          | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-698 for the enumerators of enumeration type _eSUBSEC. |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| Out      |  |             |      |       |       |          |      |       |       |      |     |      |                     |      |  | OK   |      |     |    |        |

## Function

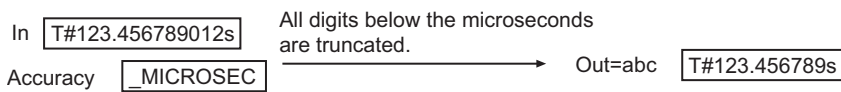
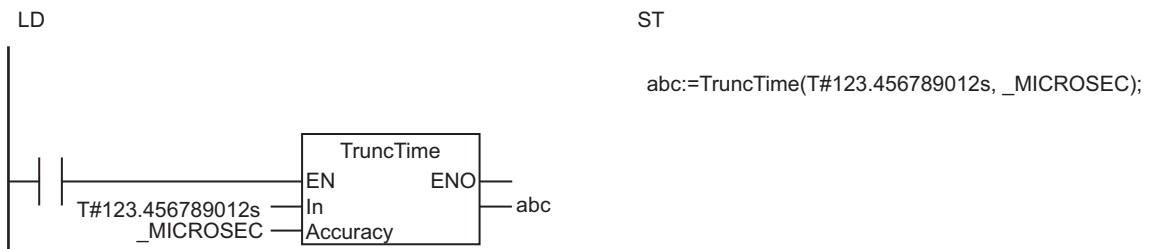
The TruncTime instruction truncates a time value *In* to the time unit of *Accuracy*. The resulting time value after truncation is stored in *Out*.

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 |

| Enumerator | Meaning |
|------------|---------|
| _SEC       | Seconds |

The following example is for when *In* is TIME#123.456789012s and *Accuracy* is \_MICROSEC.



## Additional Information

Before you compare two TIME variables with *EQ* (=) on page 2-102 or other instructions, use this instruction to convert the two variables to the same accuracy.

## 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

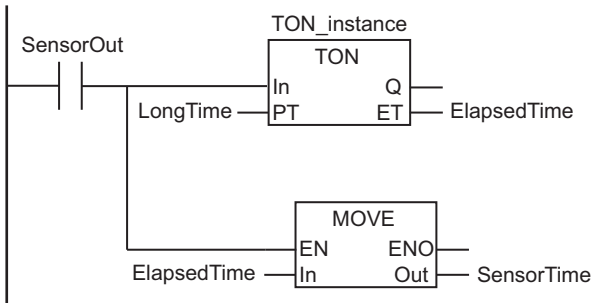
| Variable   | Data type | Initial value | Comment                    |
|------------|-----------|---------------|----------------------------|
| RecentMode | Mode      | SET           | The current operation mode |

**LD**

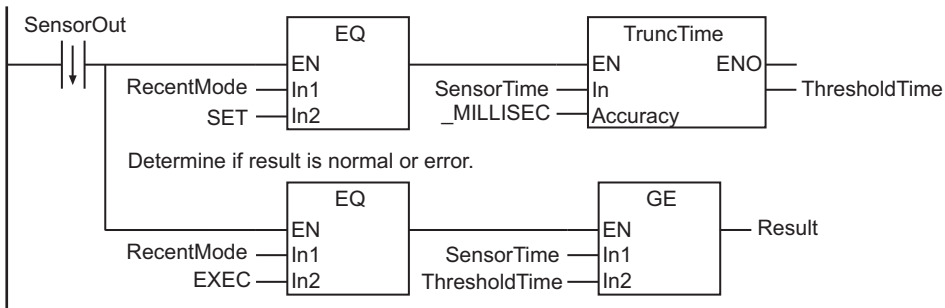
| Internal Variables | Variable      | Data type | Initial value | Comment  |
|--------------------|---------------|-----------|---------------|--|
|                    | SensorOut     | BOOL      | FALSE         | Sensor output  |
|                    | ElapsedTime   | TIME      | T#0s          | Elapsed 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 |

Measure the sensor output ON time.



Set the threshold.



## 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 to a specified time unit.

| Instruction | Name                   | FB/<br>FUN | Graphic expression | ST expression               |
|-------------|------------------------|------------|--------------------|-----------------------------|
| TruncDt     | Truncate Date and Time | FUN        |                    | Out:=TruncDt(In, Accuracy); |



## 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.

## Variables

|          | 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#197<br>0-1-1-0:<br>0:0 |
| Accuracy | Smallest unit after truncation |        | The smallest time unit to leave after truncation | _NANOSEC,<br>_MICROSEC,<br>_MILLISEC,<br>_SEC | ---                                      | _NANOSEC                  |
| Out      | Date and time after truncation | Output | Date and time after truncation                   | Depends on data type.                         | Year, month, day, hour, minutes, seconds | ---                       |

|          | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-702 for the enumerators of enumeration type <code>_eSUBSEC</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |  |
| Out      |  |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    | OK     |  |

## Function

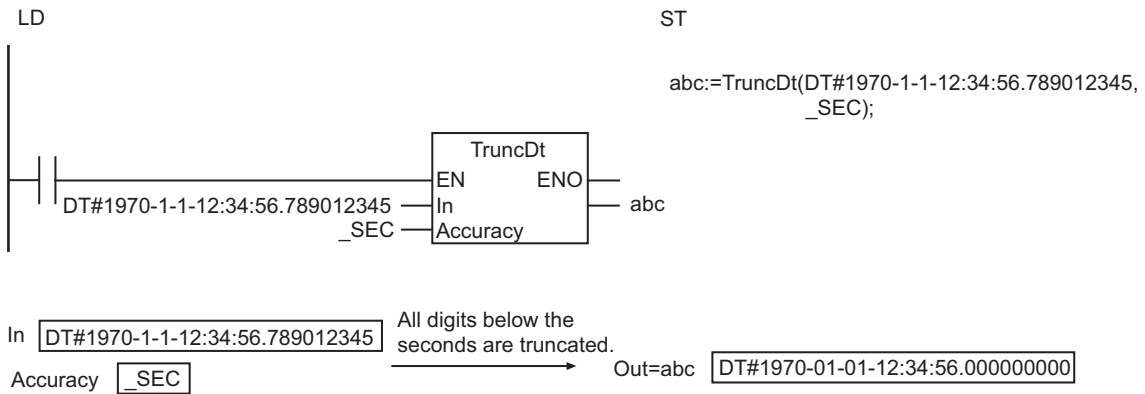
The TruncDt instruction truncates a date and time value *In* to the time unit of *Accuracy*. The resulting date and time value after truncation is stored in *Out*.

The data type of *Accuracy* is enumerated type `_eSUBSEC`. The meanings of the enumerators are as follows:

| Enumerator            | Meaning     |
|-----------------------|-------------|
| <code>_NANOSEC</code> | Nanoseconds |

| Enumerator | Meaning      |
|------------|--------------|
| _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.



## Additional Information

Before you compare two DT variables with *EQ* (=) on page 2-102 or other instructions, use this instruction to convert the two variables to the same accuracy.

## 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

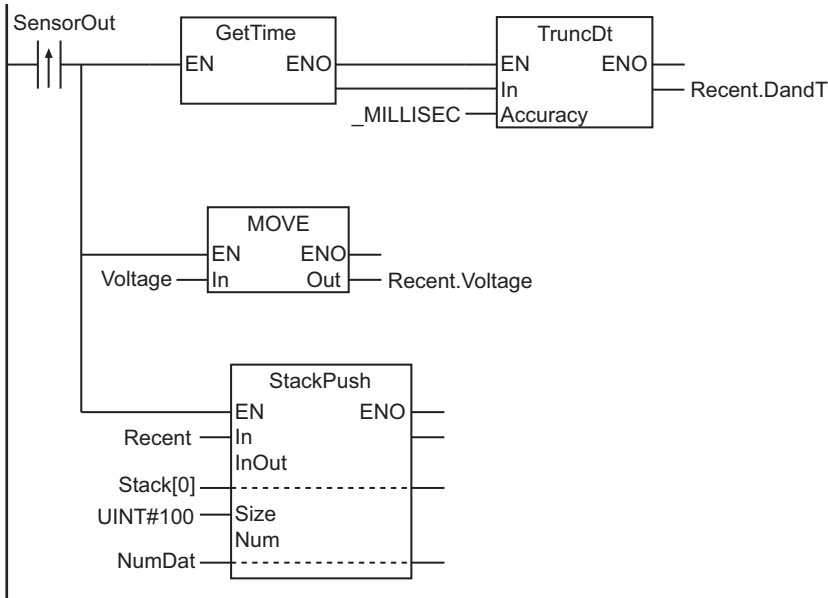
| Variable | Data type              | Initial value                                   | Comment       |
|----------|------------------------|---|---------------|
| Recent   | Record                 | (DandT:=DT#1970-1-1-0:0:0, Voltage:=0.0)        | Present value |
| Stack    | ARRAY[0..99] OF Record | [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         |

Record date and time and voltage



## 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 to a specified time unit.

| Instruction | Name                 | FB/<br>FUN | Graphic expression | ST expression                |
|-------------|----------------------|------------|--------------------|------------------------------|
| TruncTod    | Truncate Time of Day | FUN        |                    | Out:=TruncTod(In, Accuracy); |



## 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.

## Variables

|          | 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 | ---        |

|          | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-706 for the enumerators of enumeration type _eSUBSEC. |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out      |  |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      | OK  |    |        |

## Function

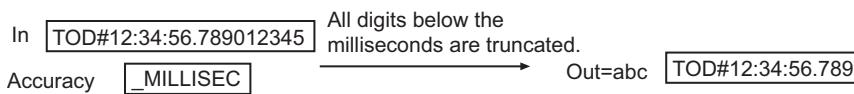
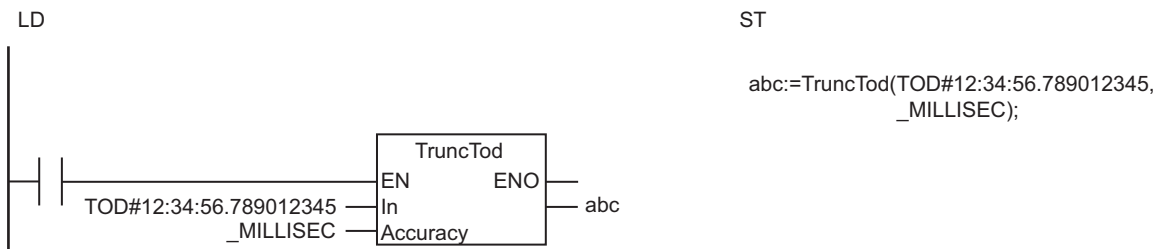
The TruncTod instruction truncates a time of day value *In* to the time unit of *Accuracy*. The resulting time of day value after truncation is stored in *Out*.

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 |

| Enumerator | Meaning |
|------------|---------|
| _SEC       | Seconds |

The following example is for when *In* is TOD#12:34:56.789012345 and *Accuracy* is \_MILLISEC.



## Additional Information

Before you compare two TOD variables with *EQ* (=) on page 2-102 or other instructions, use this instruction to convert the two variables to the same accuracy.

## 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

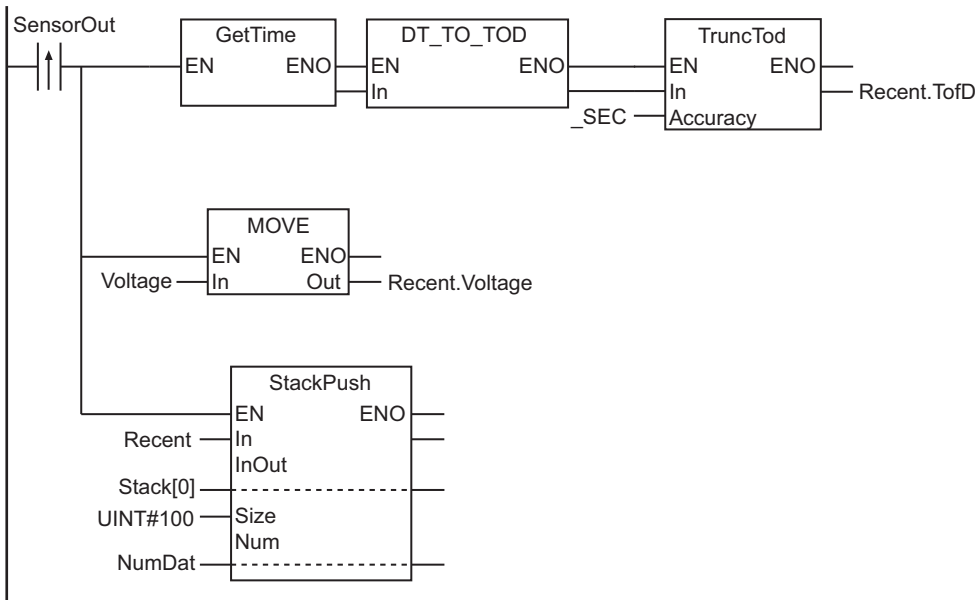
| Variable | Data type              | Initial value                          | Comment       |
|----------|------------------------|--|---------------|
| Recent   | Record                 | (TofD:=TOD#0:0:0, Voltage:=0.0)        | Present value |
| Stack    | ARRAY[0..99] OF Record | [100((TofD:=TOD#0:0:0, 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         |

Record time of day and voltage



## 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 |

| External Variables | Variable | Data type              | Comment |
|--------------------|----------|------------------------|---------|
|                    | 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

| Instruction            | Name  | Page       |
|------------------------|---|------------|
| PIDAT                  | PID Control with Autotuning                             | page 2-712 |
| PIDAT_HeatCool         | Heating/Cooling PID with Autotuning                     | page 2-742 |
| TimeProportionalOut    | Time-proportional Output                                | page 2-779 |
| LimitAlarm_**          | Upper/Lower Limit Alarm Group                           | page 2-799 |
| LimitAlarmDv_**        | Upper/Lower Deviation Alarm Group                       | page 2-804 |
| LimitAlarmDvStbySeq_** | Upper/Lower Deviation Alarm with Standby Sequence Group | page 2-809 |
| ScaleTrans             | Scale Transformation                                    | page 2-827 |
| AC_StepProgram         | Step Program  | page 2-830 |

# PIDAT

The PIDAT instruction performs PID control with autotuning (2-PID control with set point filter).

| Instruction | Name                        | FB/<br>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

|               | Meaning                        | I/O   | Description  | Valid range           | Unit | Default |
|---------------|--------------------------------|-------|--|-----------------------|------|---------|
| Run           | Execution condition            | Input | TRUE: Execute<br>FALSE: Stop                         | Depends on data type. | ---  | FALSE   |
| ManCtl        | Manual/auto control            |       | TRUE: Manual operation<br>FALSE: Automatic operation |                       |      |         |
| StartAT       | Autotuning execution condition |       | TRUE: Execute<br>FALSE: Cancel                       |                       |      |         |
| PV            | Process value                  |       | Process value  | *1                    |      | 0       |
| SP            | Set point                      |       | Set point  |                       |      |         |
| OprSetParams  | Operation setting parameters   |       | Parameters set during operation                      | ---                   |      | ---     |
| InitSetParams | Initial setting parameters     |       | Initial setting parameters                           |                       |      |         |



|                  | Meaning                      | I/O    | Description  | Valid range                               | Unit | Default |
|------------------|------------------------------|--------|--|---|------|---------|
| ProportionalBand | Proportional band            | In-out | Proportional band  | 0.01 to 1000.00                           | % FS | ---     |
| Integration-Time | Integration time             |        | Integration time<br>The higher the value is, the weaker the integral action is.<br>No integral action is performed for 0.      | T#0.0000 s to T#10000.0000 s <sup>2</sup> | s    |         |
| Derivative-Time  | Derivative time              |        | Derivative time<br>The higher the value is, the stronger the derivative action is.<br>No derivative action is performed for 0. | T#0.0000 s to T#10000.0000 s <sup>2</sup> |      |         |
| ManMV            | Manual manipulated variable  |        | Manual manipulated variable  | -320 to 320                               | %    |         |
| ATDone           | Autotuning normal completion | Output | TRUE: Normal completion<br>FALSE: <sup>*3</sup>  | Depends on data type.                     | ---  | ---     |
| ATBusy           | Autotuning busy              |        | TRUE: Autotuning<br>FALSE: Not autotuning  |   |      |         |
| MV               | Manipulated variable         |        | Manipulated variable   |   |      |         |

- \*1. Value of input range lower limit *InitSetParams.RngLowLmt* to Value of input range upper limit *InitSetParams.RngUpLmt*
- \*2. The value is truncated to four decimal places.
- \*3. FALSE indicates an error end, that PID control is in progress without autotuning, or that PID control is not in progress.

|                  | Boo lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Structure Specifications</i> on page 2-714 for details on the structure <i>_sOPR_SET_PARAMS</i> .  |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| InitSetParams    | Refer to <i>Structure Specifications</i> on page 2-714 for details on the structure <i>_sINIT_SET_PARAMS</i> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| ProportionalBand |  |             |      |       |       |       |          |       |       |      |     |      |      | OK           |       |   |      |     |    |        |
| Integration-Time |  |             |      |       |       |       |          |       |       |      |     |      |      |              |       | OK  |      |     |    |        |
| Derivative-Time  |  |             |      |       |       |       |          |       |       |      |     |      |      |              |       | 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 *Run* (execution condition) changes to TRUE. While the value of *Run* is TRUE, the following process cycle is repeated: 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 *StartAT* (autotuning execution condition) changes to TRUE, autotuning of the PID constants is executed.

### 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* <sup>1</sup> | %    | 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<br>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 $\alpha$     | Coefficient $\alpha$ of the set point filter.<br>If this value is 0, the set point filter is disabled.   | 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     |

\*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.                   | _sl-NIT_SET_PARAMETERS | ---                           | ---  | ---     |
| SampTime      | Sampling Period            | The period for PID processing.                | TIME                   | T#0.0001 s to #100.0000 s     | s    | T#0.1 s |
| RngLowLmt     | Lower Limit of Input Range | The lower limit of <i>PV</i> and <i>SP</i> .  | REAL                   | -32000 to 32000 <sup>*1</sup> | ---  | 0       |
| RngUpLmt      | Upper Limit of Input Range | The upper limit of <i>PV</i> and <i>SP</i> .  | REAL                   |                               |      | 100     |
| DirOpr        | Action Direction           | TRUE: Forward action<br>FALSE: Reverse action | 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 instruction 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 <i>ManCtl</i> | Operation mode | Value of <i>MV</i>                                       |
|------------------------|----------------|--|
| TRUE                   | Manual         | Value of <i>ManMV</i><br>(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. Refer to *Autotuning* on page 2-727 for information on autotuning.

### ● 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 <i>MV</i>                       |
|--|--|
| Less than <i>MVLowLmt</i>                              | <i>MVLowLmt</i>                          |
| Between <i>MVLowLmt</i> and <i>MVUpLmt</i> , inclusive | Manipulated variable from PID processing |
| Greater than <i>MVUpLmt</i>                            | <i>MVUpLmt</i>                           |

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* and *MVUpLmt* even if the control status of this instruction is not autotuning during automatic operation.

However, if you change *MVLowLmt* and *MVUpLmt* to an expansion direction during operation, the value of *MV* which is the same as the one in the last sampling period is output changed smoothly at this time (bumpless).

Repeated changing of *MVLowLmt* and *MVUpLmt* will affect the control performance, and sufficient control performance may not be obtained.

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.

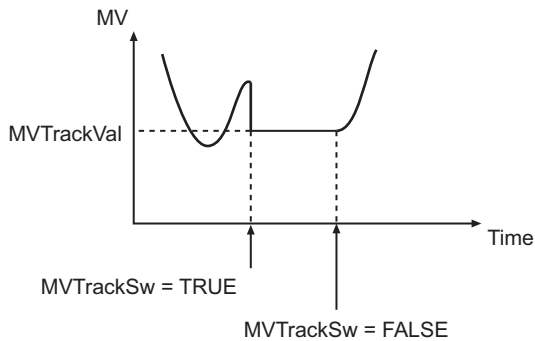
When integral operation is performed, the manual reset value is ignored. Therefore, the setting of *ManResetVal* is enabled when the value of *IntegrationTime* is 0.

- **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* changes to 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 $\alpha$ )

This parameter determines the coefficient of the set point filter.

Refer to *2-PID Control with Set Point Filter* on page 2-725 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 quick 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 smaller. However, if the process value is not stable and proper autotuning is difficult, increase the value.

Refer to *Autotuning* on page 2-727 for details.

### ● SampTime (Sampling Period)

This is the minimum value of the period for PID processing.

Refer to *Execution Timing of PID Control* on page 2-728 for details.

PID processing is not executed if the elapsed time since the last execution is shorter than *SampTime*.

### ● 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 a parameter connected to *PV* or *SP* exceeds either of these limits. *RngLowLmt* must always be less than *RngUpLmt*.

### ● DirOpr (Action Direction)

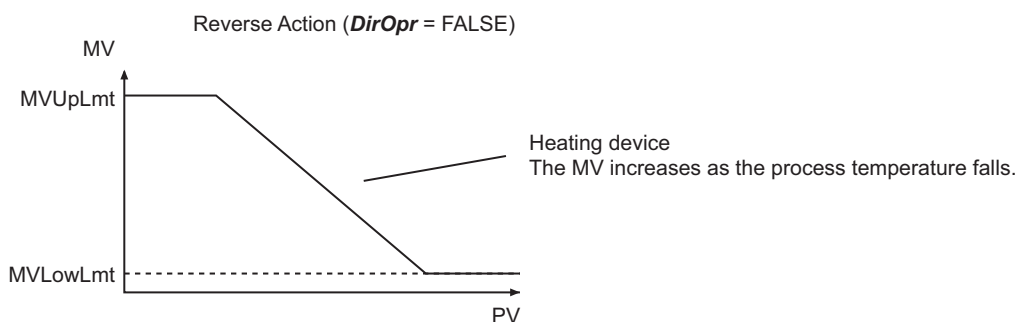
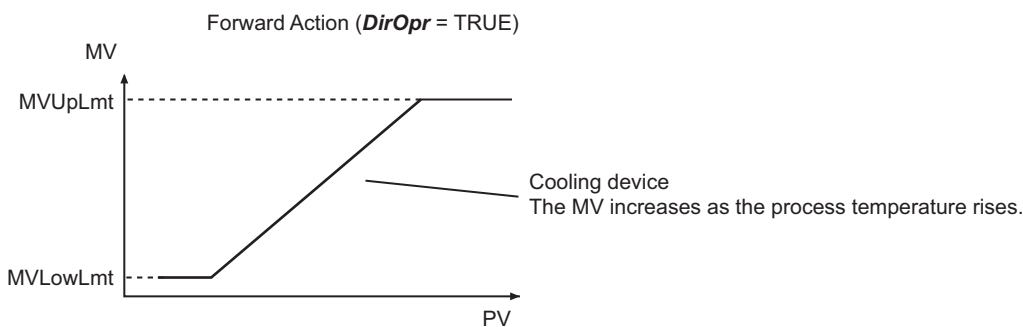
This variable specifies if *MV* is increased or decreased for changes in the value of *PV*.

These are called a forward action and a reverse action.

| Value of <i>DirOpr</i> | Meaning        | Value of <i>MV</i>                      |
|------------------------|----------------|---|
| TRUE                   | Forward action | Increases with the value of <i>PV</i> . |
| FALSE                  | Reverse action | Decreases with the value of <i>PV</i> . |

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 *Proportional Action (P)* on page 2-721 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 *Integral Action (I)* on page 2-723 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 *Derivative Action (D)* on page 2-723 for details. The larger the value of *DerivativeTime* is, the stronger the derivative action is.

### ● ManMV (Manual Manipulated Variable)

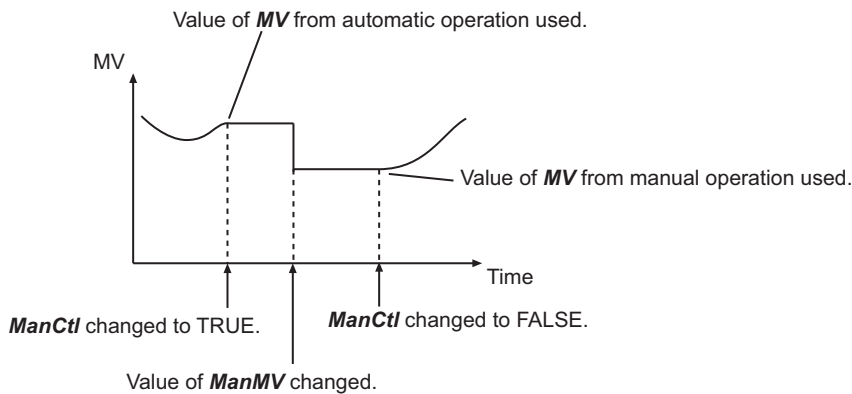
*MV* is set to this value during manual operation (while *ManCtl* is TRUE).

However, after the operation is switched from automatic to manual mode, the value of *MV* for the automatic operation is continuously applied.

*MV* is set to the value of *ManMV* only when the value of *ManMV* is changed after the operation is switched to manual mode.

When the operation is switched from manual to automatic mode, the value of *MV* for the manual operation is continuously applied.

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.

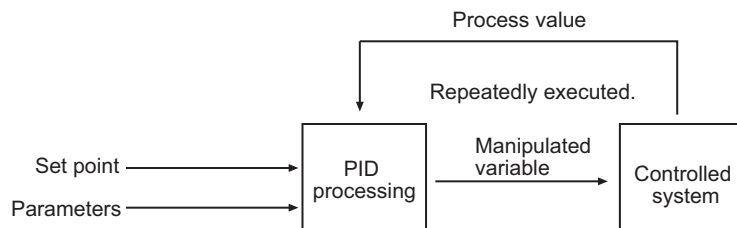


## Introduction to PID Control

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.



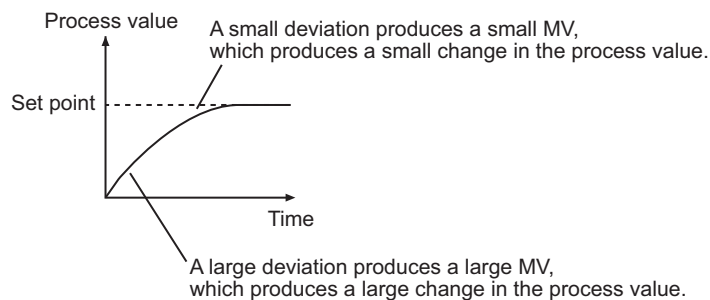
## Proportional (P), Integral (I), and Derivative (D) Actions

PID control is performed by combining the proportional action, integral action, and derivative action.

### ● 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.

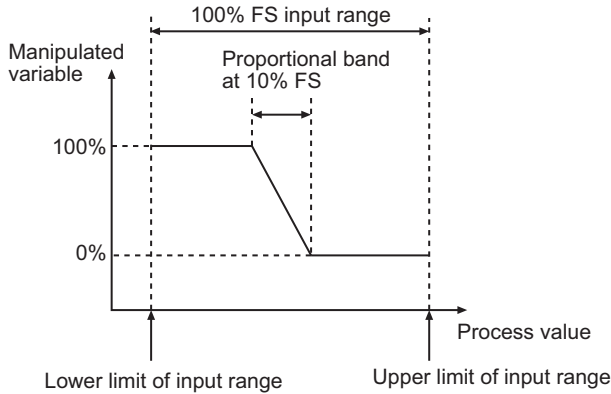
The process value of the controlled system changes as shown below.



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.



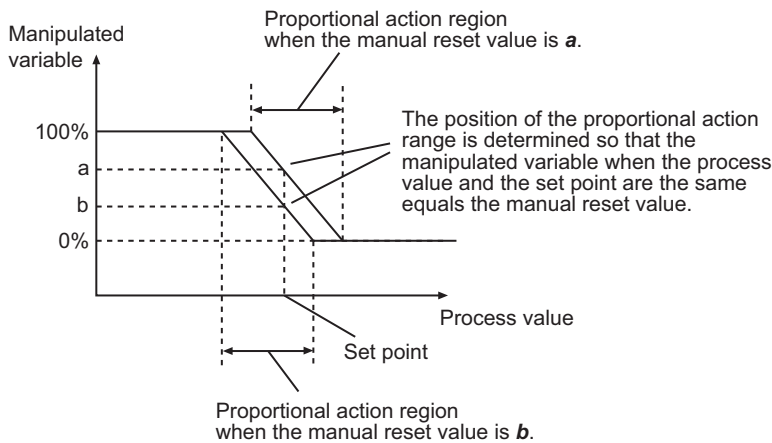
Another parameter for the proportional action is the manual reset value.

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.

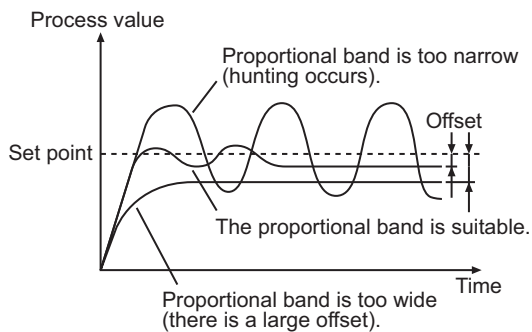
The position of the proportional action range is determined so that the manipulated variable when the process value and the set point are the same equals the manual reset value.



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.

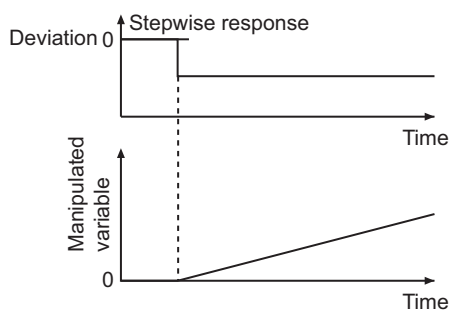
To simplify the operation, an integral action is used in combination with the proportional action.

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.

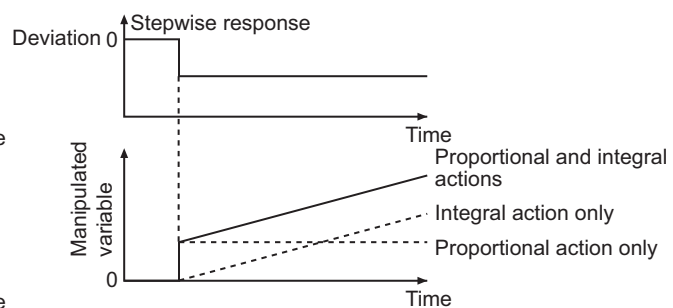
When normal distribution operation is performed, the manual reset value is ignored.

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



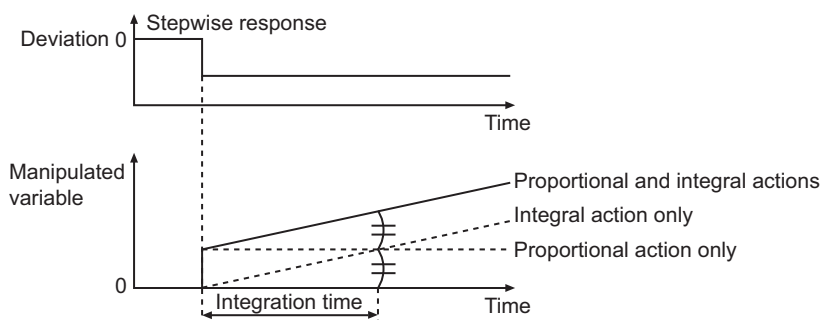
Manipulated Variable for Integral and Proportional Actions Together



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)

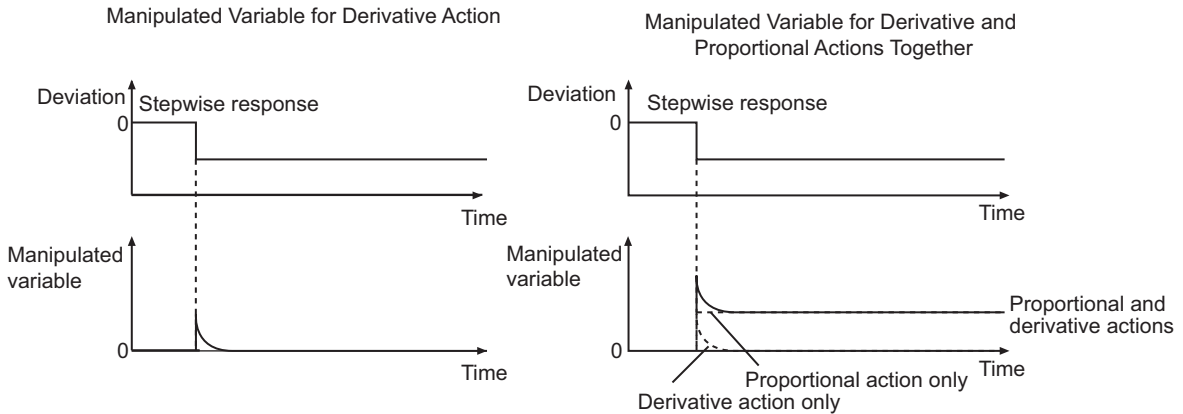
If the proportional and integral actions are used together, the offset will reach 0 and the process value will reach the set point.

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.

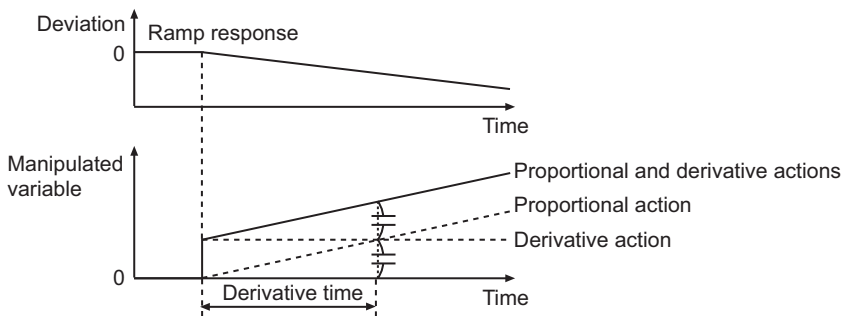
The derivative action differentiates the deviation on the time axis and then increases the absolute value of the manipulated variable in proportion to the result. In other words, the larger the change in the process value is, the larger the absolute value of the manipulated variable for the derivative 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.

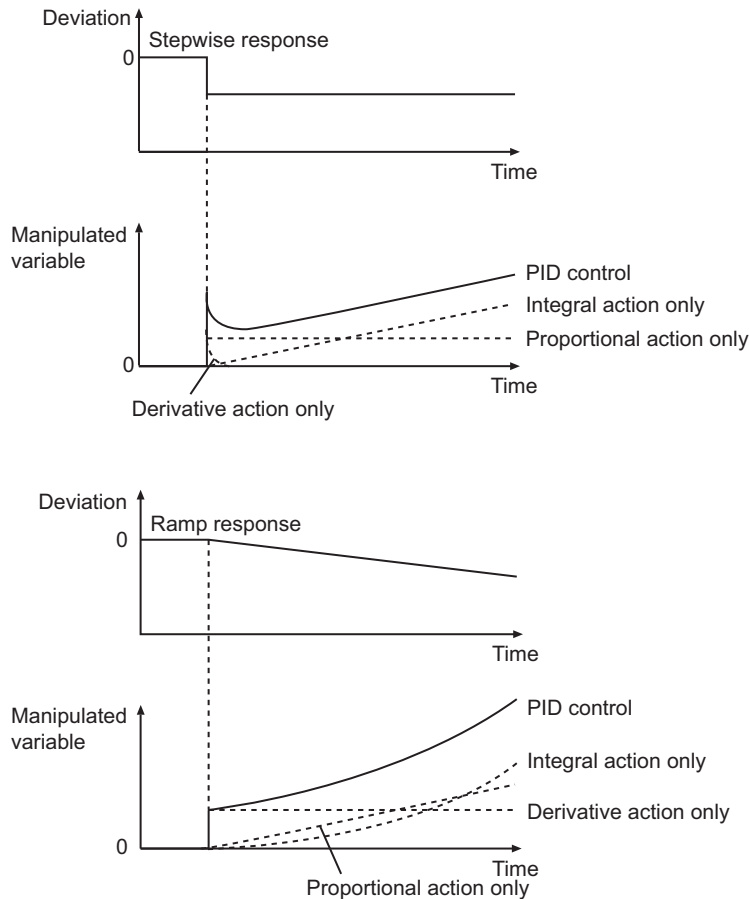
The longer the derivative time is, the stronger the derivative action is. A long derivative time provides a rapid response to disturbances, but it can also cause hunting.



### ● PID Control

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

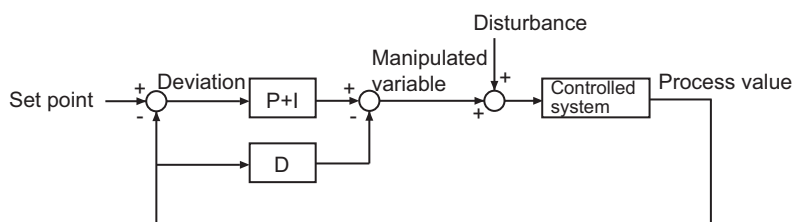
There are three main parameters that you must adjust to perform PID control: the proportional band, integration time, and derivative time. These are called the PID constants.

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 are input at different points as shown in the block diagram. 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.

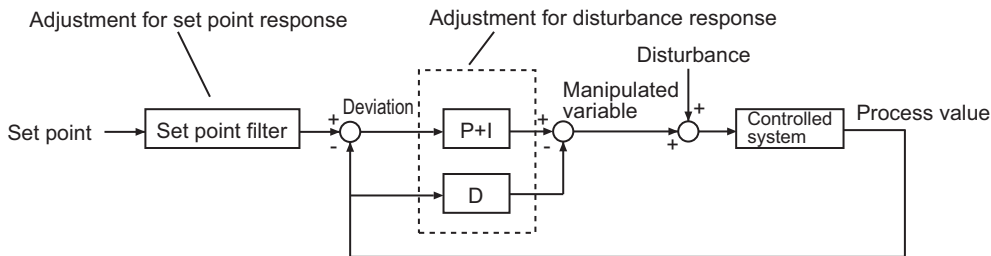
The 2 in "2-PID" indicates that there are separate parameters to adjust the set point response and the disturbance response.

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.

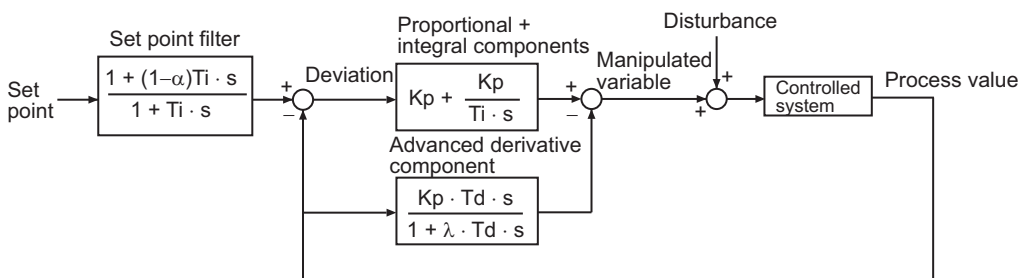
You can adjust the values of the PID constants and the set value of the set point filter independently to increase both the set point response and the disturbance response.



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  $\alpha$ .

The optimum value of  $\alpha$  is 0.65. It normally does not need to be changed. The lower the value of  $\alpha$  is, the smaller the influence of the set point filter is.



$K_p$ : Proportional constant  
 $T_i$ : Integration time  
 $T_d$ : Derivative time  
 $s$ : Laplace operator  
 $\alpha$ : 2-PID parameter  
 $\lambda$ : Incomplete derivative coefficient

## Starting PID Control

You must use suitable PID constants to execute this instruction. There are two ways to start PID control, depending on whether the optimal values of the PID constants are known or not known.

You can change the values of the PID constants during operation. You can also perform autotuning during operation. To start autotuning during operation, change the value of *StartAT* to TRUE.

### ● 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

Assign the optimum values of the PID constants to *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.

## Control Status and Manipulated Variable

Manipulated variable *MV* is determined according to the control status as shown in the following table.

| Control status                              | Value of variable                          |  |                                |  |                                       | <i>MV</i><br>(manipulated variable)           |   |
|---|--|--|--------------------------------|--|---------------------------------------|---|---|
|   | <i>ManCtl</i><br>(manual/<br>auto control) | <i>Run</i><br>(execution<br>condition) | <i>Error</i><br>(error<br>end) | <i>MVTrack</i><br>Sw<br>(MV<br>tracking<br>switch) | <i>ATBusy</i><br>(autotuning<br>busy) |   |   |
| Error end                                   | FALSE                                      | TRUE                                   | TRUE                           | ---  | FALSE                                 | <i>ErrorMV</i><br>(error MV)                  |   |
| during automatic operation (MV tracking)    |  |  | ---                            | TRUE   |                                       | FALSE   | <i>MVTrackVal</i><br>(MV tracking value)                                  |
| during automatic operation (Autotuning)     |  |  | FALSE                          | FALSE  | FALSE                                 | TRUE  | Value repeatedly changes between upper limit of MV and lower limit of MV. |
| during automatic operation (Not autotuning) |  |  | ---                            | ---  |                                       | FALSE   | Value calculated with current PID constants.                              |
| Instruction execution stopped               |  | FALSE                                  | ---                            | ---  | FALSE                                 | <i>StopMV</i><br>(Stop MV)                    |   |
| Manual operation                            |  | TRUE                                   | ---                            | ---  | ---                                   | <i>ManMV</i><br>(manual manipulated variable) |   |

## Autotuning

The 2-PID parameter  $\alpha$  is not adjusted very often, so the main parameters that are adjusted for this instruction are the PID constants.

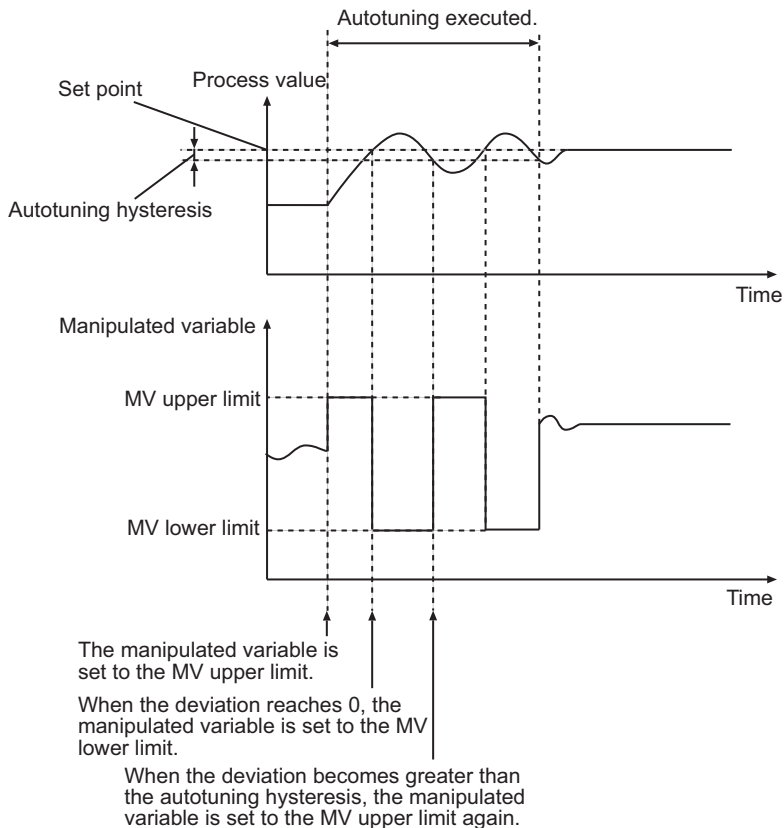
The PIDAT instruction supports autotuning of the PID constants.

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 *ATBusy* changes to FALSE during autotuning (i.e., while *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.

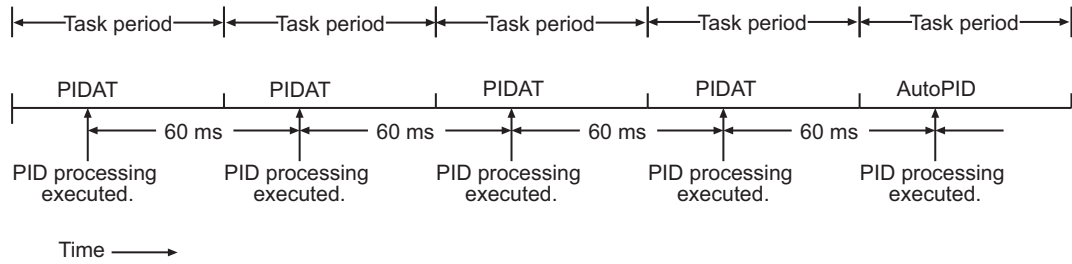
However, PID processing is not executed if the elapsed time since the last execution is shorter than *SampTime*.

If the elapsed time since the last execution exceeds *SampTime*, the excess time (elapsed time - *SampTime*) is carried forward to the next period. See below for details.



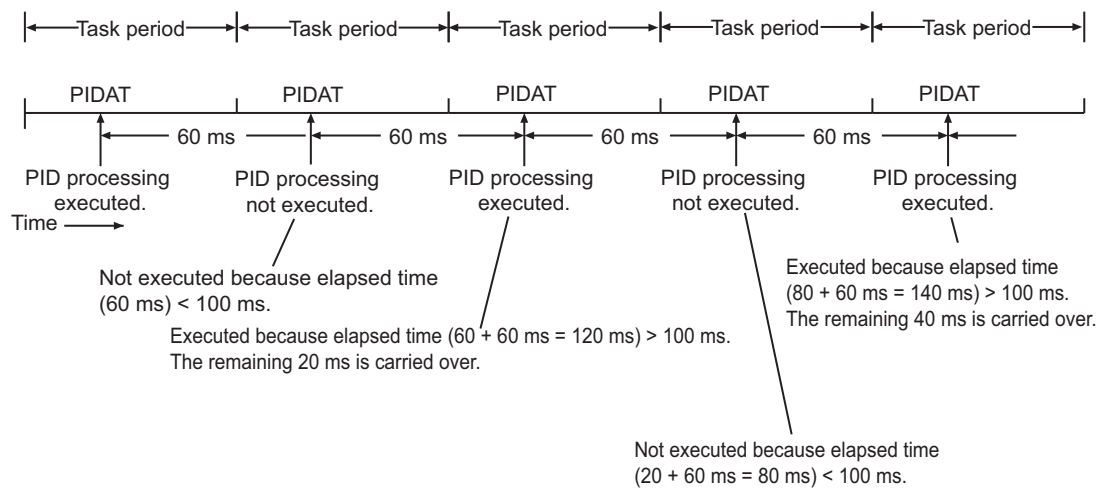
Task period = 60 ms and **SampTime** < 60 ms

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

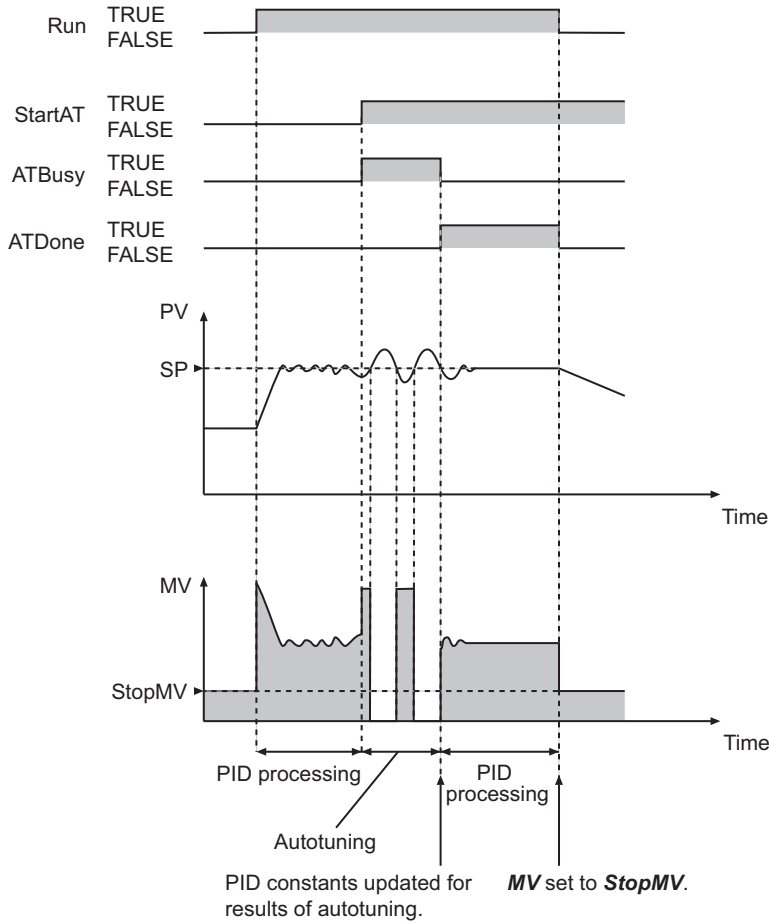
The task period is less than **SampTime**, so DIP processing is not executed every period.



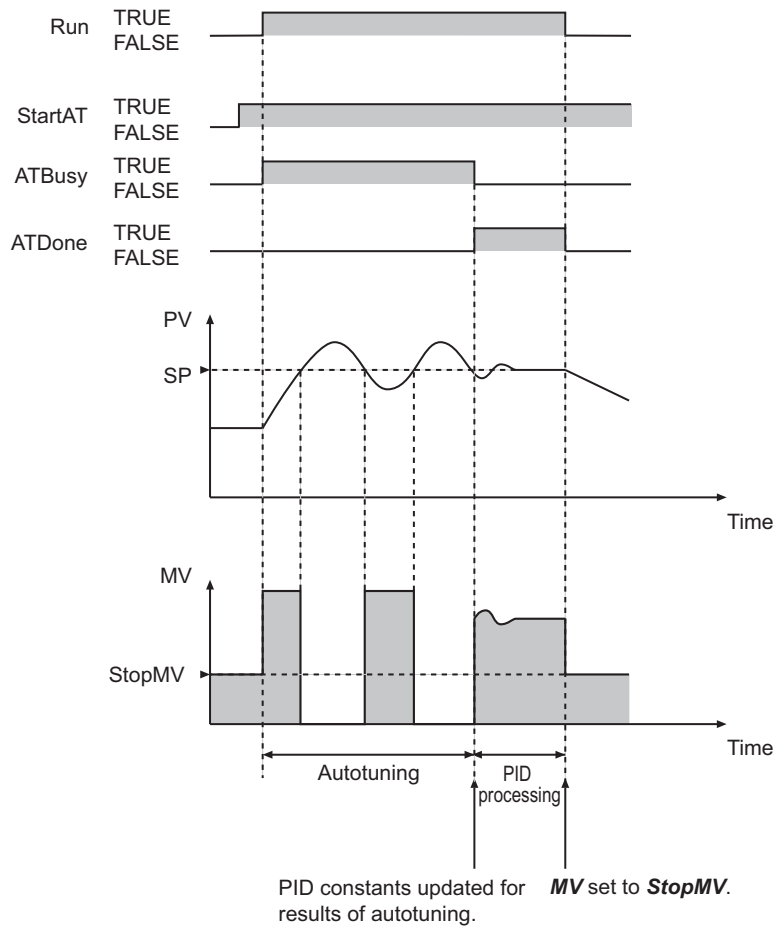
## Timing Charts

Timing charts for the instruction variables are provided below for different situations.

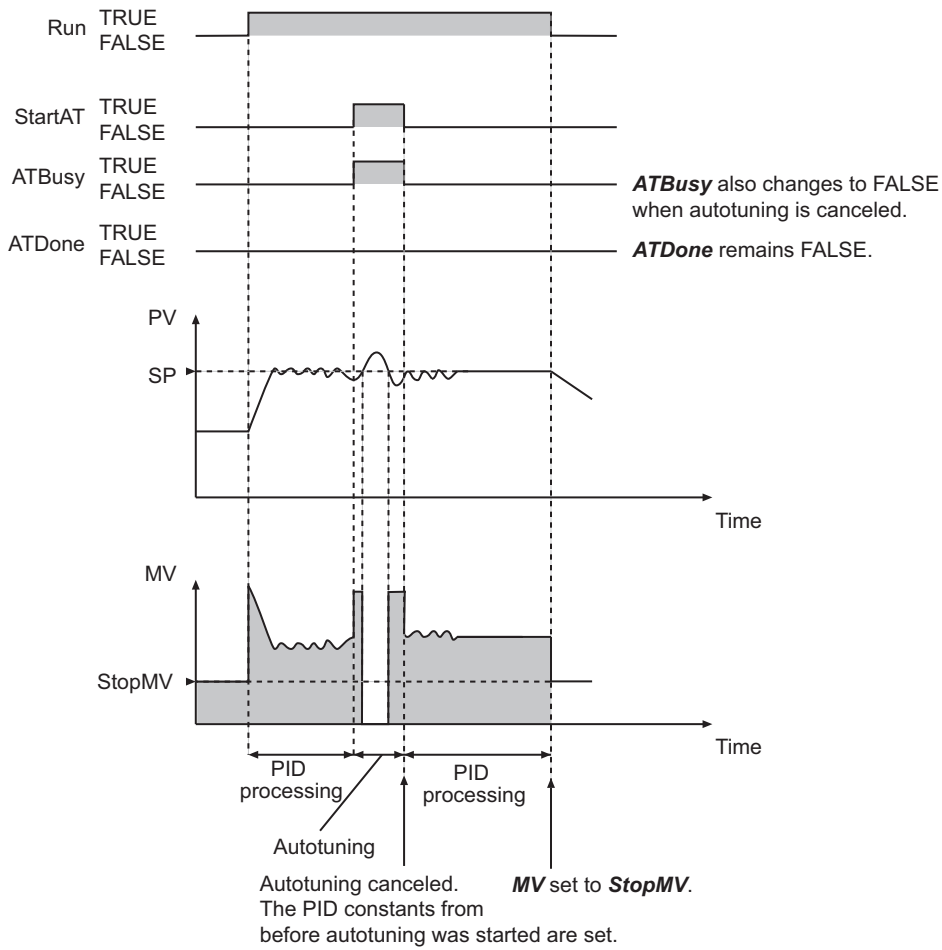
● Autotuning Executed during Automatic Operation



● Autotuning Executed at the Start of PIDAT Execution



● **Autotuning Canceled**

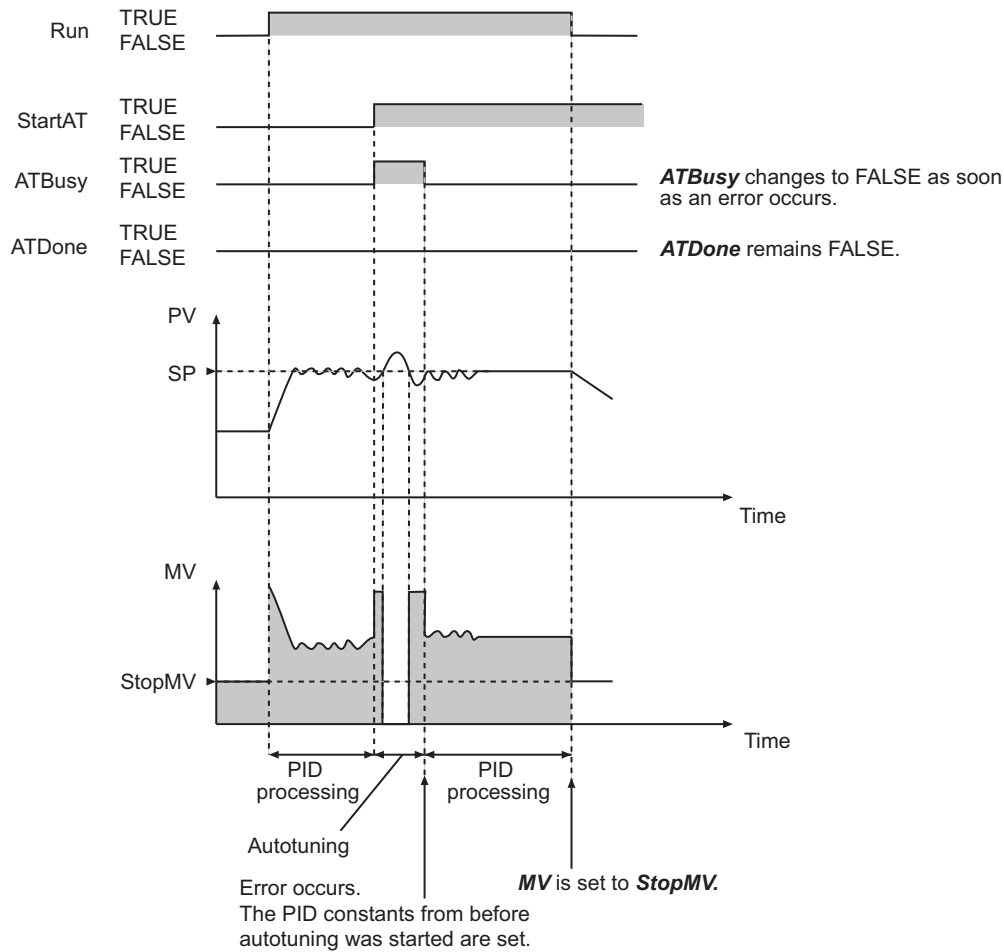


● **An Autotuning Error Occurs during Autotuning**

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.

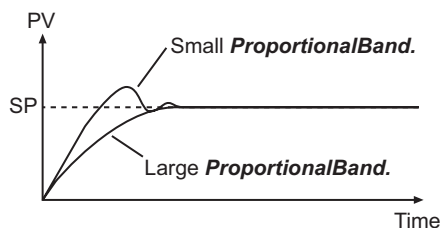
If autotuning is canceled, PID control is started again with the previous PID constants.



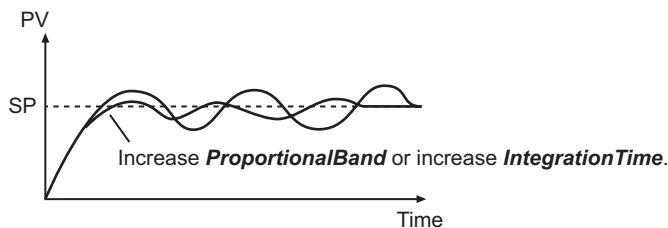
## Additional Information

### Adjusting PID Constants

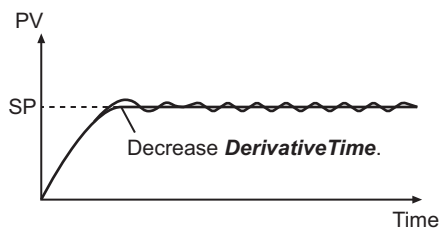
- When you need to eliminate hunting even if it takes time to stabilize the control system, increase the value of *ProportionalBand*. If a certain amount of hunting is not a problem, but it is necessary for the controlled system to stabilize quickly, decrease the value of *ProportionalBand*.



- If hunting continues too long, increase either *ProportionalBand* or *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) <sup>*1</sup> |
|-------------------------|---|
| <i>ProportionalBand</i> | 10%FS   |
| <i>IntegrationTime</i>  | 233 s   |
| <i>DerivativeTime</i>   | 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 <i>PV</i> and <i>SP</i>  | Values of <i>RngLowLmt</i> and <i>RngUpLmt</i>                              |
|---------------|--|---|
| % FS          | $PV = (\text{Process value in physical units} - \text{MIN}) / (\text{MAX} - \text{MIN}) \times 100^{*1}$<br>$SP = (\text{Set point in physical units} - \text{MIN}) / (\text{MAX} - \text{MIN}) \times 100^{*1}$ | <i>RngLowLmt</i> = 0<br><i>RngUpLmt</i> = 100                               |
| Physical unit | <i>PV</i> = Process value in physical units<br><i>SV</i> = Set point in physical units   | <i>RngLowLmt</i> = MIN <sup>*1</sup><br><i>RngUpLmt</i> = MAX <sup>*1</sup> |

\*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 <sup>*1</sup> | Automatic operation when autotuning is not being executed <sup>*2</sup> | Automatic operation when autotuning is being executed <sup>*3</sup> |
| Run       | Possible                                    | Possible  | Possible  |
| ManCtl    | Possible                                    | Possible  | Possible  |

| Variables        | Control status                              |   |   |
|------------------|---|---|---|
|                  | Instruction execution stopped* <sup>1</sup> | Automatic operation when autotuning is not being executed* <sup>2</sup> | Automatic operation when autotuning is being executed* <sup>3</sup> |
| 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  |
| 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 <i>ErrorID</i> |
|--|-------------------------|
| The value of an input variable is outside of the valid range.  | 16#0400                 |
| <i>RngLowLmt</i> is greater than or equal to <i>RngUpLmt</i> . | 16#0401                 |
| <i>MVLowLmt</i> is greater than or equal to <i>MVUpLmt</i> .   |                         |

- 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 for conversion to a time-proportional value, refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.

## 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.

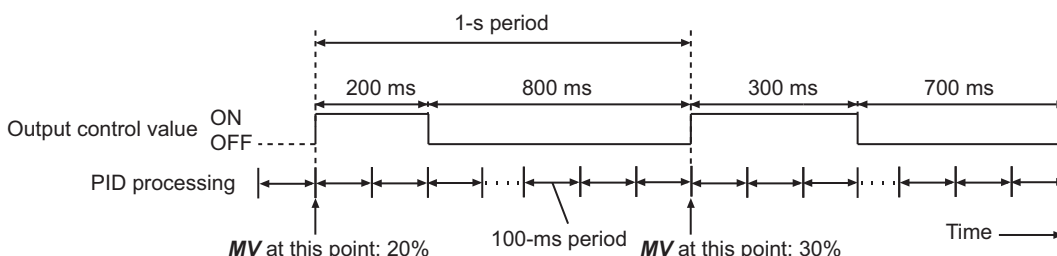
| Unit       | I/O port   | Description                          | Variable |
|------------|------------|--------------------------------------|----------|
| CJ1W-PH41U | Ch1_AllnPv | Process value for input 1 (INT data) | AI1      |
| CJ1W-OD212 | Ch1_Out00  | Bit 00 of output word 1              | DO1      |

## Processing

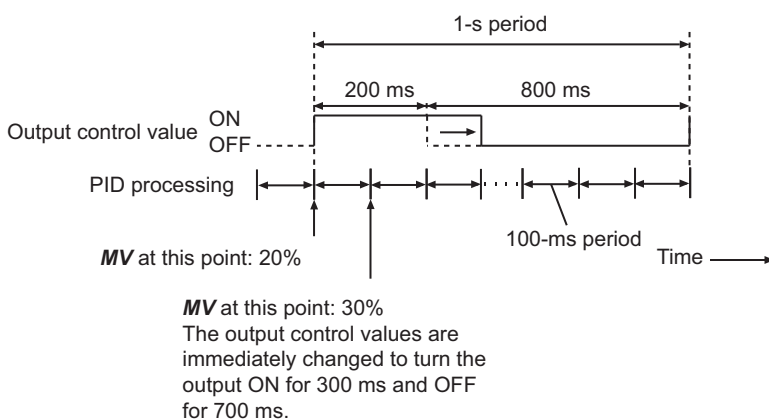
- *MV* (manipulated variable) of the PIDAT instruction is obtained to control the output to the temperature controller. The output to the temperature controller is turned ON or OFF.
- The sampling period (*InitSetParams.SampTime*) of the PIDAT instruction is set to 100 ms. The task period must be sufficiently 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 the following 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 are determined when the value of *MV* is 20% (ON for 200 ms, and OFF for 800 ms). If the new value of *MV* is 30% after 100 ms elapses, the output control values are immediately changed to turn the output ON for 300 ms and then OFF for 700 ms.



- If autotuning is performed and the value of *MV* changes to 100%, the output is immediately turned ON regardless of the control period.

## Definitions of Global Variables

### ● Global Variables

| Variable | Data type | AT specification *1                     | Comment                              |
|----------|-----------|---|--------------------------------------|
| AI1      | INT       | IOBus://rack#0/slot#0/Ch1_AllnPV        | Process value for input 1 (INT data) |
| DO1      | BOOL      | IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00 | Bit 00 of output word 1              |

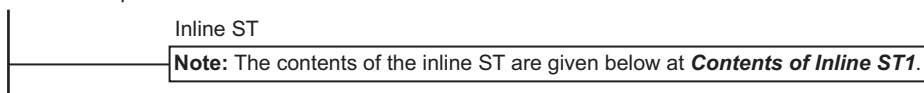
\*1. This table shows the variables for the CJ1W-PH41U Input Unit mounted to Slot #0 of Rack #0, and the CJ1W-OD212 Output Unit mounted to Slot #1 of the same rack.

**Note** The global variables for the port of each Unit are automatically generated based on the I/O mapping settings.

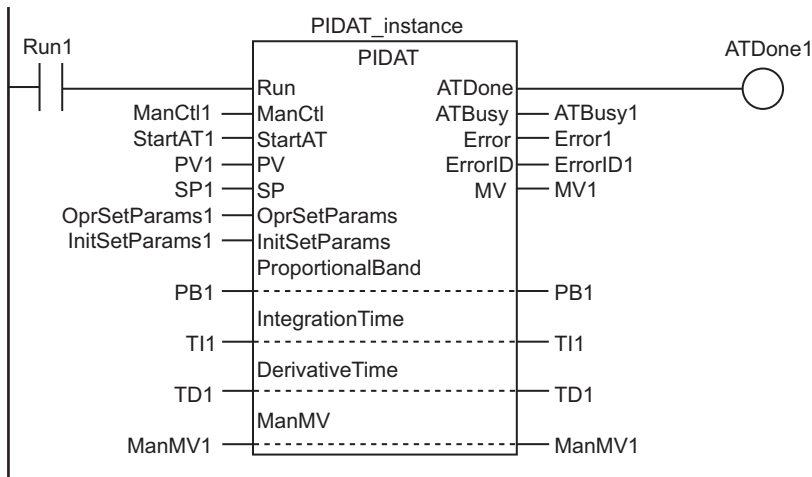
## LD

| Variable       | Data type         | Initial value  | Retain                              | Comment                        |
|----------------|-------------------|--|-------------------------------------|--------------------------------|
| Run1           | BOOL              | FALSE  | <input type="checkbox"/>            | Execution condition            |
| ManCtl1        | BOOL              | FALSE  | <input type="checkbox"/>            | Manual/auto control            |
| StartAT1       | BOOL              | FALSE  | <input type="checkbox"/>            | Autotuning execution condition |
| PV1            | REAL              | 0.0  | <input type="checkbox"/>            | Process value                  |
| SP1            | REAL              | 90   | <input type="checkbox"/>            | 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) | <input type="checkbox"/>            | Operation setting parameters   |
| InitSetParams1 | _sINIT_SET_PARAMS | (SampTime:=T#100 ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)  | <input type="checkbox"/>            | Initial setting parameters     |
| PB1            | REAL              | 10   | <input checked="" type="checkbox"/> | Proportional band              |
| TI1            | TIME              | T#0 s  | <input checked="" type="checkbox"/> | Integration time               |
| TD1            | TIME              | T#0 s  | <input checked="" type="checkbox"/> | Derivative time                |
| ManMV1         | REAL              | 0.0  | <input type="checkbox"/>            | Manual manipulated variable    |
| ATDone1        | BOOL              | FALSE  | <input type="checkbox"/>            | Autotuning normal completion   |
| ATBusy1        | BOOL              | FALSE  | <input type="checkbox"/>            | Executing autotuning           |
| Error1         | BOOL              | FALSE  | <input type="checkbox"/>            | Error                          |
| ErrorID1       | WORD              | 16#0   | <input type="checkbox"/>            | Error ID                       |
| MV1            | REAL              | 0.0  | <input type="checkbox"/>            | Manipulated variable           |
| PulseOnTime    | TIME              | T#0 s  | <input type="checkbox"/>            | Control output ON time         |
| PulseCycTime   | TIME              | T#1 s  | <input type="checkbox"/>            | Control period                 |
| ResetPulse     | BOOL              | FALSE  | <input type="checkbox"/>            | Timer reset                    |
| PIDAT_instance | PIDAT             |  | <input type="checkbox"/>            |                                |
| TOF_instance   | TOF               |  | <input type="checkbox"/>            |                                |
| TON_instance   | TON               |  | <input type="checkbox"/>            |                                |

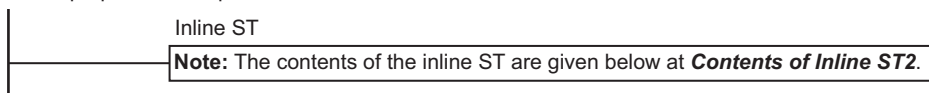
Obtain the process value.



Execute PIDAT instruction.



Time-proportional output



### ● 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

```
// Calculate ON time output control value.
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);
// Switch between ON and OFF with TOF instruction.
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);
// Measure timer reset time with TON instruction.
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse);
// Reset timer.
IF (ResetPulse=BOOL#TRUE) THEN
  TOF_instance(In:=BOOL#TRUE);
  TON_instance(In:=BOOL#FALSE);
END_IF;
// If MV1 = 100% for autotuning.
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN
  DO1:=BOOL#TRUE; // Turn ON the output immediately.
END_IF;
```

## ST

| Variable       | Data type         | Initial value  | Retain                              | Comment                        |
|----------------|-------------------|--|-------------------------------------|--------------------------------|
| Run1           | BOOL              | FALSE  | <input type="checkbox"/>            | Execution condition            |
| ManCtl1        | BOOL              | FALSE  | <input type="checkbox"/>            | Manual/auto control            |
| StartAT1       | BOOL              | FALSE  | <input type="checkbox"/>            | Autotuning execution condition |
| PV1            | REAL              | 0.0  | <input type="checkbox"/>            | Process value                  |
| SP1            | REAL              | 90   | <input type="checkbox"/>            | 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) | <input type="checkbox"/>            | Operation setting parameters   |
| InitSetParams1 | _sINIT_SET_PARAMS | (SampTime:=T#100 ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)  | <input type="checkbox"/>            | Initial setting parameters     |
| PB1            | REAL              | 10   | <input checked="" type="checkbox"/> | Proportional band              |
| TI1            | TIME              | T#0 s  | <input checked="" type="checkbox"/> | Integration time               |
| TD1            | TIME              | T#0 s  | <input checked="" type="checkbox"/> | Derivative time                |
| ManMV1         | REAL              | 0.0  | <input type="checkbox"/>            | Manual manipulated variable    |
| ATDone1        | BOOL              | FALSE  | <input type="checkbox"/>            | Autotuning normal completion   |
| ATBusy1        | BOOL              | FALSE  | <input type="checkbox"/>            | Executing autotuning           |
| Error1         | BOOL              | FALSE  | <input type="checkbox"/>            | Error                          |
| ErrorID1       | WORD              | 16#0   | <input type="checkbox"/>            | Error ID                       |
| MV1            | REAL              | 0.0  | <input type="checkbox"/>            | Manipulated variable           |
| PulseOnTime    | TIME              | T#0 s  | <input type="checkbox"/>            | Control output ON time         |
| PulseCycTime   | TIME              | T#1 s  | <input type="checkbox"/>            | Control period                 |
| ResetPulse     | BOOL              | FALSE  | <input type="checkbox"/>            | Timer reset                    |
| PIDAT_instance | PIDAT             |  | <input type="checkbox"/>            |                                |
| TOF_instance   | TOF               |  | <input type="checkbox"/>            |                                |
| TON_instance   | TON               |  | <input type="checkbox"/>            |                                |

```
// Convert PV AI1 to real number.
```

```

// CJ1W-PH41U output is ten times the process value, so divide by 10.0.
PV1:=INT_TO_REAL(AI1)/REAL#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
// Calculate ON time output control value.
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);
// Switch between ON and OFF with TOF instruction.
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);
// Reset timer.
IF (ResetPulse=BOOL#TRUE) THEN
  TOF_instance(In:=BOOL#TRUE);
  TON_instance(In:=BOOL#FALSE);
END_IF;
// If MV1 = 100% for autotuning.
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN
  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/<br>FUN | Graphic expression | ST expression   |
|----------------|--------------------------------------|------------|--------------------|---|
| PIDAT_HeatCool | Heating/Cooling PID with Auto-tuning | FB         |                    | <pre>PIDAT_HeatCool_instance(   Run,   ManCtl, StartAT, PV, SP, Dead-   Band, OprSetParams, InitSetPar-   ams, ProportionalBand_Heat, Inte-   grationTime_Heat, Derivative-   Time_Heat, Proportional-   Band_Cool, IntegrationTime_Cool,   DerivativeTime_Cool, ManMV,   CtlPrd_Cool, ATDone, ATBusy, Er-   ror, ErrorID, MV, MV_Heat,   MV_Cool);</pre> |



## 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.

| Variables     |                                |       |   |                       |      |         |
|---------------|--------------------------------|-------|---|-----------------------|------|---------|
|               | Meaning                        | I/O   | Description   | Valid range           | Unit | Default |
| Run           | Execution condition            | Input | TRUE: Execute<br>FALSE: Stop  | Depends on data type. | ---  | FALSE   |
| ManCtl        | Manual/auto control            |       | TRUE: Manual operation<br>FALSE: Automatic operation                    |                       |      |         |
| StartAT       | Autotuning execution condition |       | TRUE: Execute<br>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 <i>MV_Cool</i> | T#0.1 s to T#100 s    |      | T#20 s  |

|                        | Meaning                                  | I/O    | Description  | Valid range                                | Unit | Default |
|------------------------|--|--------|--|--|------|---------|
| Proportional Band_Heat | Proportional band for heating control    | In-out | Proportional band for heating control  | 0.01 to 1000.00                            | % FS | ---     |
| Integration-Time_Heat  | Integration time for heating control     |        | Integration time for heating control<br>The higher the value is, the weaker the integral action is.<br>No integral action is performed for 0.      | T#0.0000 s to T#10000.0000 s <sup>*2</sup> | s    |         |
| Derivative Time_Heat   | Derivative time for heating control      |        | Derivative time for heating control<br>The higher the value is, the stronger the derivative action is.<br>No derivative action is performed for 0. | T#0.0000 s to T#10000.0000 s <sup>*2</sup> |      |         |
| Proportional Band_Cool | Proportional band for cooling control    |        | Proportional band for cooling control  | 0.01 to 1000.00                            | % FS |         |
| Integration-Time_Cool  | Integration time for cooling control     | Output | Integration time for cooling control<br>The higher the value is, the weaker the integral action is.<br>No integral action is performed for 0.      | T#0.0000 s to T#10000.0000 s <sup>*2</sup> | s    | ---     |
| Derivative Time_Cool   | Derivative time for cooling control      |        | Derivative time for cooling control<br>The higher the value is, the stronger the derivative action is.<br>No derivative action is performed for 0. | T#0.0000 s to T#10000.0000 s <sup>*2</sup> |      |         |
| ManMV                  | Manual manipulated variable              |        | Manual manipulated variable  | -320 to 320                                | %    |         |
| ATDone                 | Autotuning normal completion             | Output | TRUE: Normal completion<br>FALSE: <sup>*3</sup>  | Depends on data type.                      | ---  | ---     |
| ATBusy                 | Autotuning busy                          |        | TRUE: Autotuning<br>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. The value is truncated to four decimal places.

\*3. FALSE indicates an error end, that PID control is in progress without autotuning, or that PID control is not in progress.



|                           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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                   |       |  |      |     |    |        |
| OprSetPar-<br>ams         | Refer to <i>Structure Specifications</i> on page 2-747 for details on the structure <code>_sOPR_SET_PARAMS</code> .  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| InitSetPar-<br>ams        | Refer to <i>Structure Specifications</i> on page 2-747 for details on the structure <code>_sINIT_SET_PARAMS</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| CtlPrd_Cool               |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       | OK   |      |     |    |        |
| Proportional<br>Band_Heat |  |             |      |       |       |       |          |       |       |      |     |      |      | OK                   |       |  |      |     |    |        |
| Integration-<br>Time_Heat |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       | OK   |      |     |    |        |
| Derivative<br>Time_Heat   |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       | OK   |      |     |    |        |
| Proportional<br>Band_Cool |  |             |      |       |       |       |          |       |       |      |     |      |      | OK                   |       |  |      |     |    |        |
| Integration-<br>Time_Cool |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       | OK   |      |     |    |        |
| Derivative<br>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 cycle is repeated: 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.

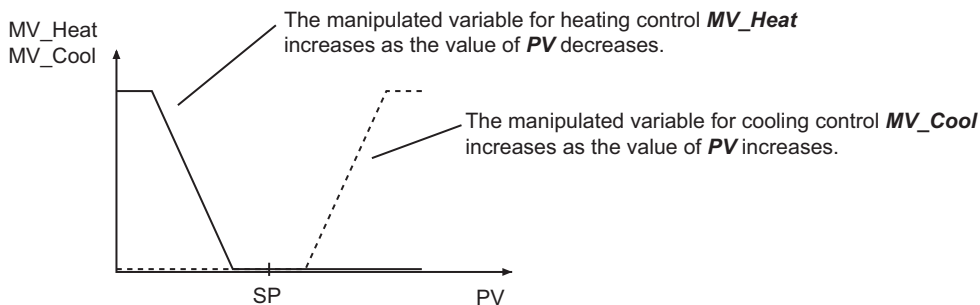
Autotuning is supported to automatically find the optimum PID constants for heating control and for cooling control.

When the value of *StartAT* (autotuning execution condition) changes to TRUE, autotuning of the PID constants for heating control and cooling control is executed.

## Difference between the PIDAT\_HeatCool and PIDAT Instructions

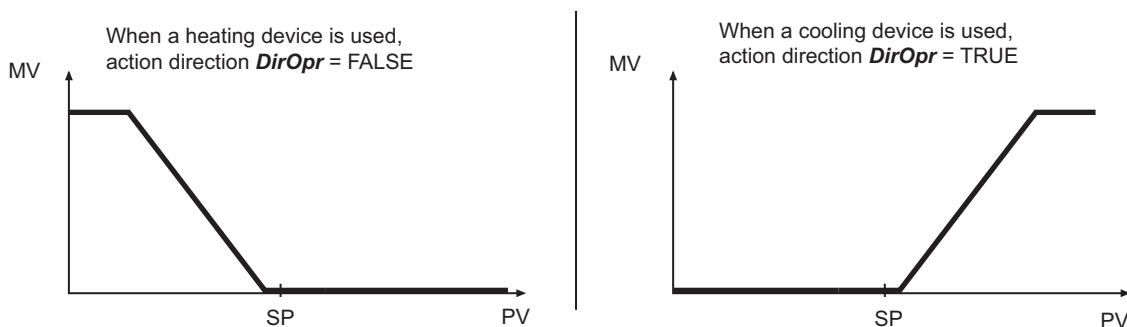
### ● PIDAT\_HeatCool Instruction

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*, which determines whether the manipulated variable is output to a heating device or a cooling device. The PIDAT\_HeatCool instruction does not use *DirOpr*.



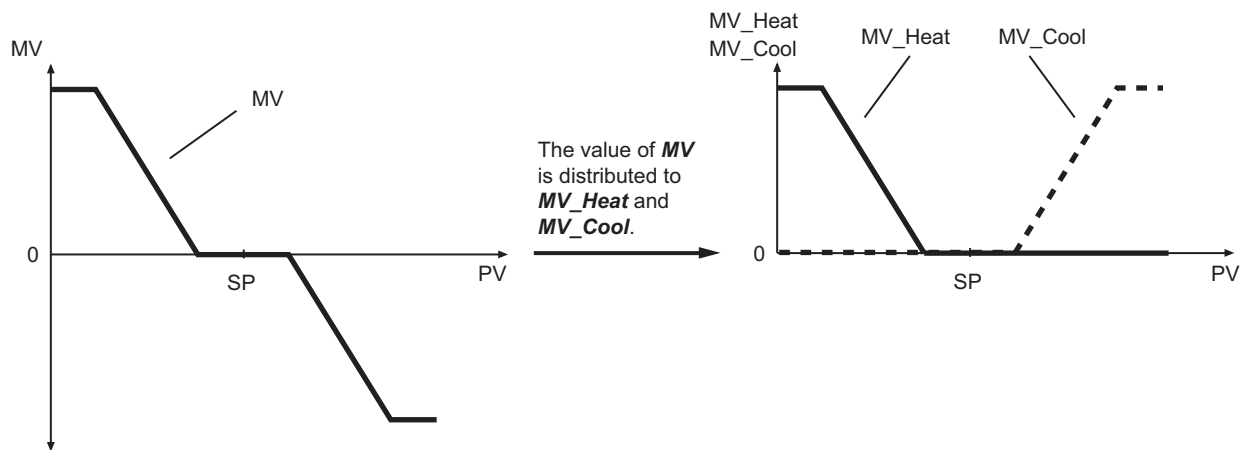
## Manipulated Variable *MV* Compared with Manipulated Variable for Heating Control *MV\_Heat* and Manipulated Variable for Cooling Control *MV\_Cool*

*MV* is a manipulated variable for temperature control where either a heating device or cooling device is used, as previously described for the PID instruction.

The PIDAT\_HeatCool instruction also calculates *MV* in the same way as the PIDAT instruction. The *MV* is distributed to the manipulated variables for the heating device and the cooling device, as *MV\_Heat* and *MV\_Cool*, respectively.

The following is a conceptual diagram to show 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 it is negative.



The above figure just indicates the concept. Actual values of  $MV\_Heat$  and  $MV\_Cool$  are not exactly the same as the absolute value of  $MV$ .

The values of  $MV\_Heat$  and  $MV\_Cool$  are calculated based on the value of  $MV$ , using special formulas.

## 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 <i>MV_Heat</i> and <i>MV_Cool</i>   | REAL             | -320 to 320* <sup>1</sup> | %    | -100        |
| MVUpLmt      | MV Upper Limit               | Upper limit of <i>MV_Heat</i> and <i>MV_Cool</i>   | REAL             |                           |      | 100         |
| ManResetVal  | Manual Reset Value           | Not used.  | REAL             |                           |      | -320 to 320 |
| MVTrackSw    | MV Tracking Switch           | MV Tracking Switch<br>TRUE: ON<br>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 $\alpha$     | Coefficient $\alpha$ of the set point filter.<br>If this value is 0, the set point filter is disabled.   | 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        |

\*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.                  | _sl-NIT_SET_PARAMS | ---                           | ---  | ---      |
| SampTime      | Sampling Period            | The period for PID processing.               | TIME               | T#0.0001 s to #100.0000 s     | s    | T#0.05 s |
| RngLowLmt     | Lower Limit of Input Range | The lower limit of <i>PV</i> and <i>SP</i> . | REAL               | -32000 to 32000* <sup>1</sup> | ---  | 0        |
| RngUpLmt      | Upper Limit of Input Range | The upper limit of <i>PV</i> and <i>SP</i> . | 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 instruction 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 <i>ManCtl</i> | Operation mode | Value of <i>MV</i>   |
|------------------------|----------------|--|
| TRUE                   | Manual         | Value of <i>ManMV</i><br>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 the autotuning. Refer to *Autotuning* on page 2-758 for details on autotuning.

### ● PV (Process Value)

This is the process value of the controlled system.

● **SP (Set Point)**

This is the set point for the controlled system.

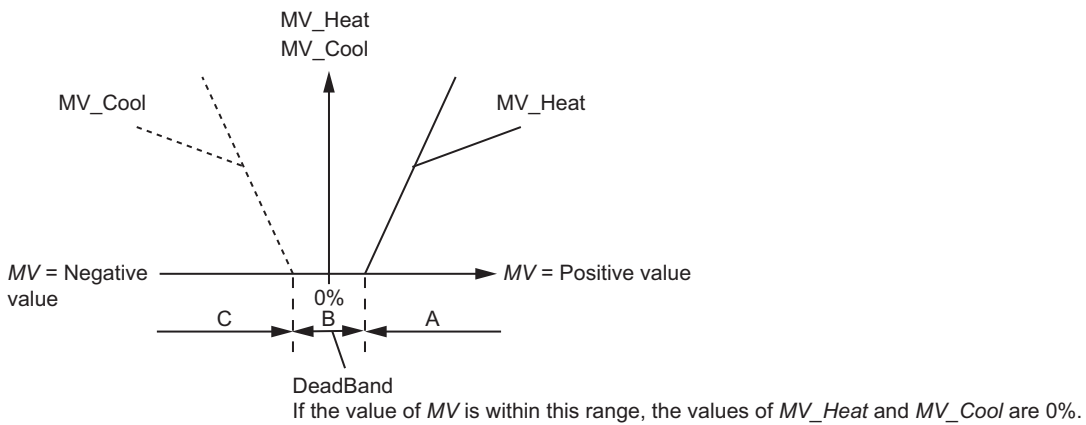
● **DeadBand (Deadband)**

*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 <i>MV</i>                 | Value of <i>MV_Heat</i>                                  | Value of <i>MV_Cool</i>                                  |
|------------------------------------|--|--|
| Larger than the deadband (Area A)  | Positive. Increases as the value of <i>MV</i> increases. | 0  |
| Within the deadband (Area B)       | 0  | 0  |
| Smaller than the deadband (Area C) | 0  | Positive. Increases as the value of <i>MV</i> decreases. |

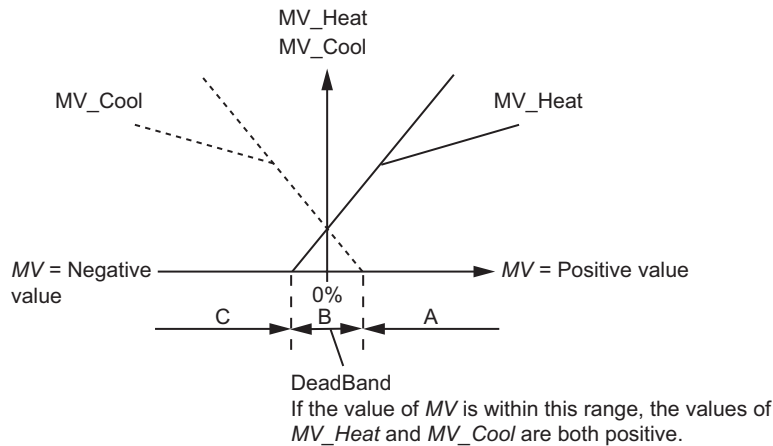


You can also set a negative value for *DeadBand*.

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 <i>MV</i>                 | Value of <i>MV_Heat</i>                                  | Value of <i>MV_Cool</i>                                  |
|------------------------------------|--|--|
| Larger than the deadband (Area A)  | Positive. Increases as the value of <i>MV</i> increases. | 0  |
| Within the deadband (Area B)       | Positive. Increases as the value of <i>MV</i> increases. | Positive. Increases as the value of <i>MV</i> decreases. |
| Smaller than the deadband (Area C) | 0  | Positive. Increases as the value of <i>MV</i> 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*.

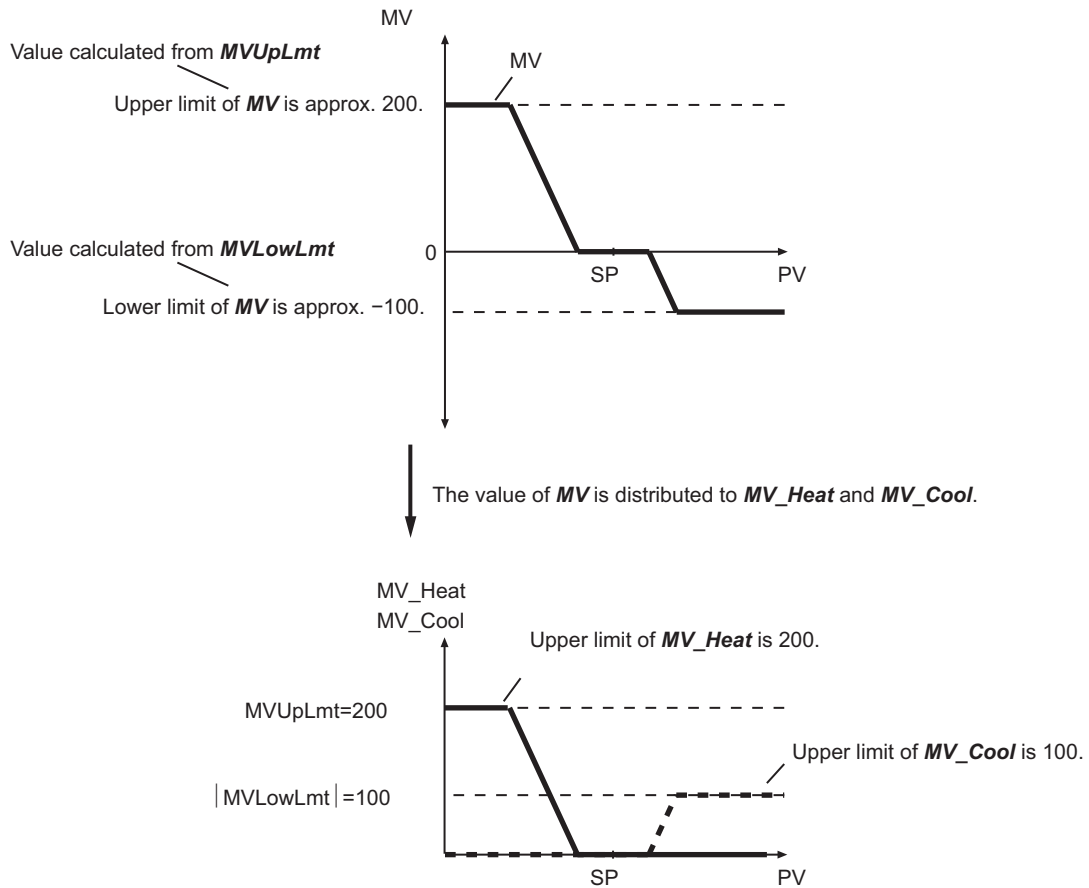
The following procedure is used to find the values of *MV\_Heat* and *MV\_Cool*.

**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.

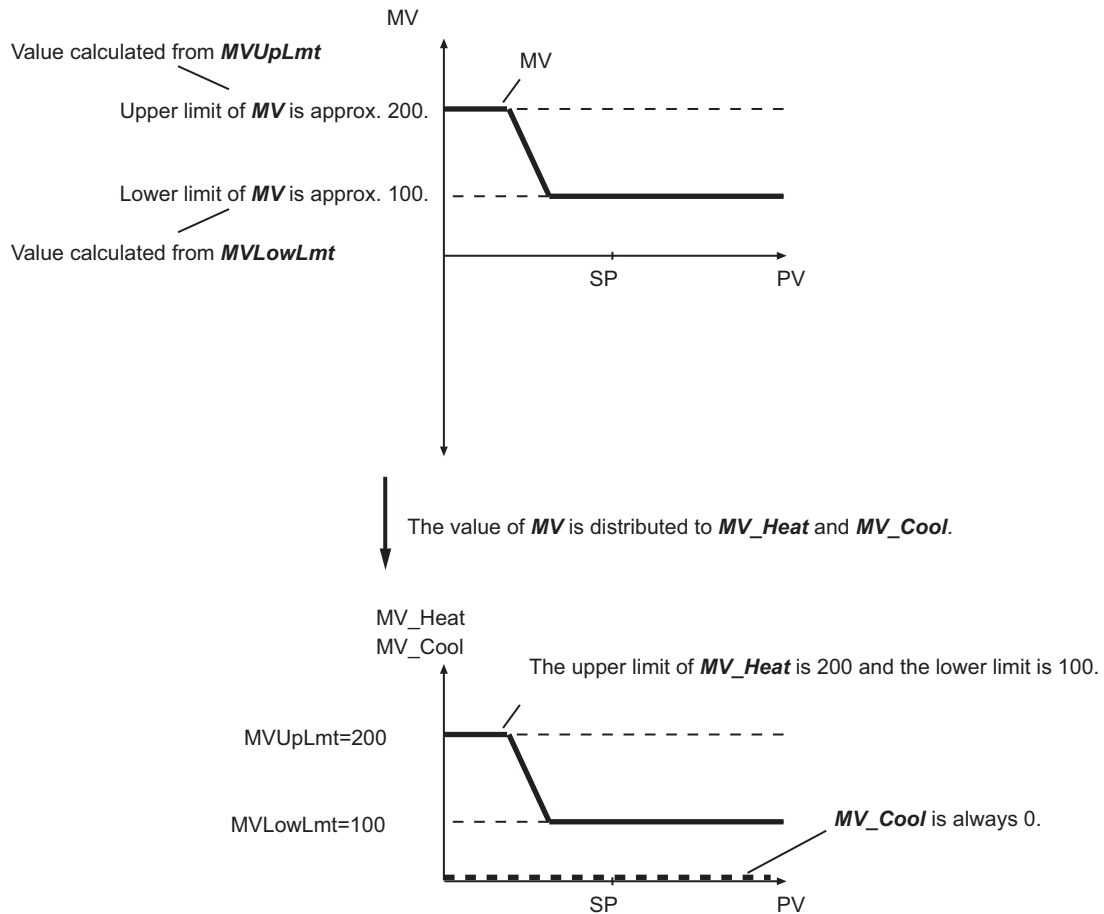
The calculated upper limit of *MV\_Heat* is 200, and the calculated lower limit is 0. The calculated upper limit of *MV\_Cool* is 100, and the calculated lower limit is 0. In other words, the upper limit of *MV\_Heat* is the same as the value of *MVUpLmt*, but the upper limit of *MV\_Cool* is the absolute 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*, respectively.





As shown above, the upper and lower limits of *MV\_Heat* and *MV\_Cool* change depending on whether *MVLowLmt* and *MVUpLmt* are positive values or negative values. Refer to the table below.

| Value of <i>MVLowLmt</i> | Value of <i>MVUpLmt</i> | <i>MV_Heat</i>  |                | <i>MV_Cool</i>                   |                                   |
|--------------------------|-------------------------|-----------------|----------------|----------------------------------|-----------------------------------|
|                          |                         | Lower limit     | Upper limit    | Lower limit                      | Upper limit                       |
| Positive                 | Positive                | <i>MVLowLmt</i> | <i>MVUpLmt</i> | 0                                | 0                                 |
| Negative                 | Positive                | 0               | <i>MVUpLmt</i> | 0                                | Absolute value of <i>MVLowLmt</i> |
| Negative                 | Negative                | 0               | 0              | Absolute value of <i>MVUpLmt</i> | Absolute value of <i>MVLowLmt</i> |

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 autotuning 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 the 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, and sufficient control performance may not be obtained.

Confirm the effects on the control performance before you repeatedly change *MVLowLmt* or *MVUpLmt* during operation.

### ● ManResetVal (Manual Reset Value)

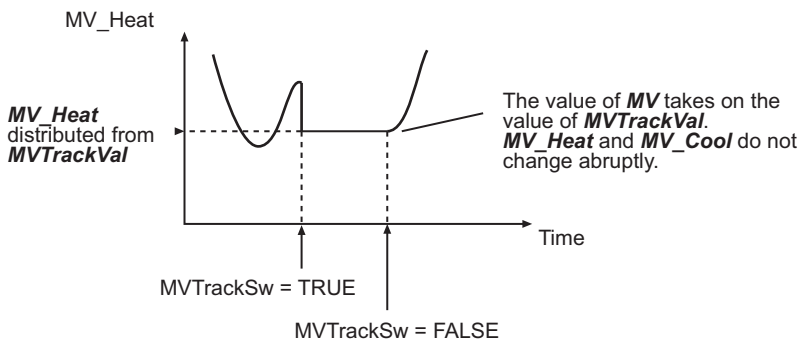
This instruction does not use this variable. Any value that is set is ignored.

### ● MVTrackSw (MV Tracking Switch)

MV tracking is a function that sets the *MV* to an external input value, *MVTrackVal* (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 $\alpha$ )

This parameter determines the coefficient of the set point filter.

Refer to *2-PID Control with Set Point Filter* on page 2-725 in the section on the PIDAT instruction 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 smaller. However, if the process value is not stable and proper autotuning is difficult, increase the value.

Refer to *Autotuning* on page 2-727 in the section on the PIDAT instruction for details.

### ● SampTime (Sampling Period)

This is the minimum value of the period for heating/cooling PID processing.

Refer to *Execution Timing of Heating/Cooling PID Control* on page 2-759 for details.

Heating/cooling PID processing is not executed if the elapsed time since the last execution is shorter than *SampTime*.

### ● 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 a parameter connected to *PV* or *SP* exceeds the corresponding 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 instruction, *TimeProportionalOut* on page 2-779. 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#20 s.

### ● ProportionalBand\_Heat and ProportionalBand\_Cool (Proportional Bands)

This is one of the three PID constants. Refer to *Proportional Action (P)* on page 2-721 in the section on the PIDAT instruction 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 *Integral Action (I)* on page 2-723 in the section on the PIDAT instruction 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 *Derivative Action (D)* on page 2-723 in the section on the PIDAT instruction for details.

The larger the value of *DerivativeTime\_Heat* or *DerivativeTime\_Cool* is, the stronger the derivative action is.

### ● ManMV (Manual Manipulated Variable)

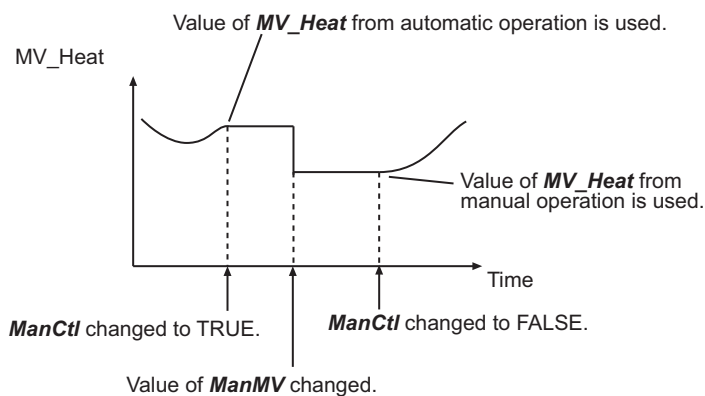
*MV* is set to this value during manual operation (while *ManCtl* is TRUE).

*MV* is set to the value of *ManMV* only when the value of *ManMV* is changed after the operation is switched from automatic to manual mode.

When the operation is switched from automatic to manual mode, the value of *MV* is set to the value of *MV\_Heat* if the value of *MV\_Heat* for the automatic operation is positive, or set to the value of *MV\_Cool* if it is not.

Also, after the operation is switched from manual to automatic mode, the value of *MV* is set to the value of *MV\_Heat* if the value of *MV\_Heat* is positive, or set to the value of *MV\_Cool* if it is not.

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 instruction, *PIDAT* on page 2-712 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 result, 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 *Proportional (P), Integral (I), and Derivative (D) Actions* on page 2-721 for the *PIDAT* instruction for details on the proportional action (P), integral action (I), and derivative action (D).

## 2-PID Control with Set Point Filter

Refer to *2-PID Control with Set Point Filter* on page 2-725 for the *PIDAT* instruction for details on the 2-PID Control with Set Point Filter.

## Starting PID Control

You must use suitable PID constants to execute this instruction. There are two ways to start PID control, depending on whether the optimal values of the PID constants are known or not known.

You can change the values of the PID constants during operation. You can also perform autotuning during operation. To start autotuning during operation, change the value of *StartAT* to TRUE.

- **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 appropriate variables, and then assign the values to input parameters.

## Control Status and Manipulated Variable

Manipulated variable *MV* is determined according to the control status as shown in the following table.

| Control status                                   | Value of variable                          |  |                                |   |   | <i>MV</i><br>(manipulated variable)              |  |
|--|--|--|--------------------------------|---|---|--|--|
|  | <i>ManCtl</i><br>(manual/<br>auto control) | <i>Run</i><br>(execution<br>condition) | <i>Error</i><br>(error<br>end) | <i>MVTrack</i><br><i>Sw</i><br>( <i>MV</i><br>tracking<br>switch) | <i>ATBusy</i><br>(autotuning<br>busy)                       |  |  |
| Error end  | FALSE                                      | TRUE                                   | TRUE                           | ---   | FALSE   | <i>ErrorMV</i><br>(error <i>MV</i> )             |  |
| during automatic operation ( <i>MV</i> tracking) |  |  | ---                            | TRUE  |   | <i>MVTrackVal</i><br>( <i>MV</i> tracking value) |  |
| during automatic operation (Autotuning)          |  |  | FALSE                          | FALSE   | FALSE   | TRUE   | Value repeatedly changes between upper limit of <i>MV</i> and lower limit of <i>MV</i> . |
| during automatic operation (Not autotuning)      |  |  | ---                            | ---   |   | FALSE  | Value calculated with current PID constants.   |
| Instruction execution stopped                    |  |  | FALSE                          | ---   | ---   | FALSE  | <i>StopMV</i> <sup>*1</sup><br>(Stop <i>MV</i> )   |
| Manual operation                                 | TRUE                                       | ---                                    | ---                            | ---   | <i>ManMV</i> <sup>*2</sup><br>(manual manipulated variable) |  |  |

\*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  $\alpha$  is not adjusted very often, so the main parameters that are adjusted for this instruction are the PID constants.

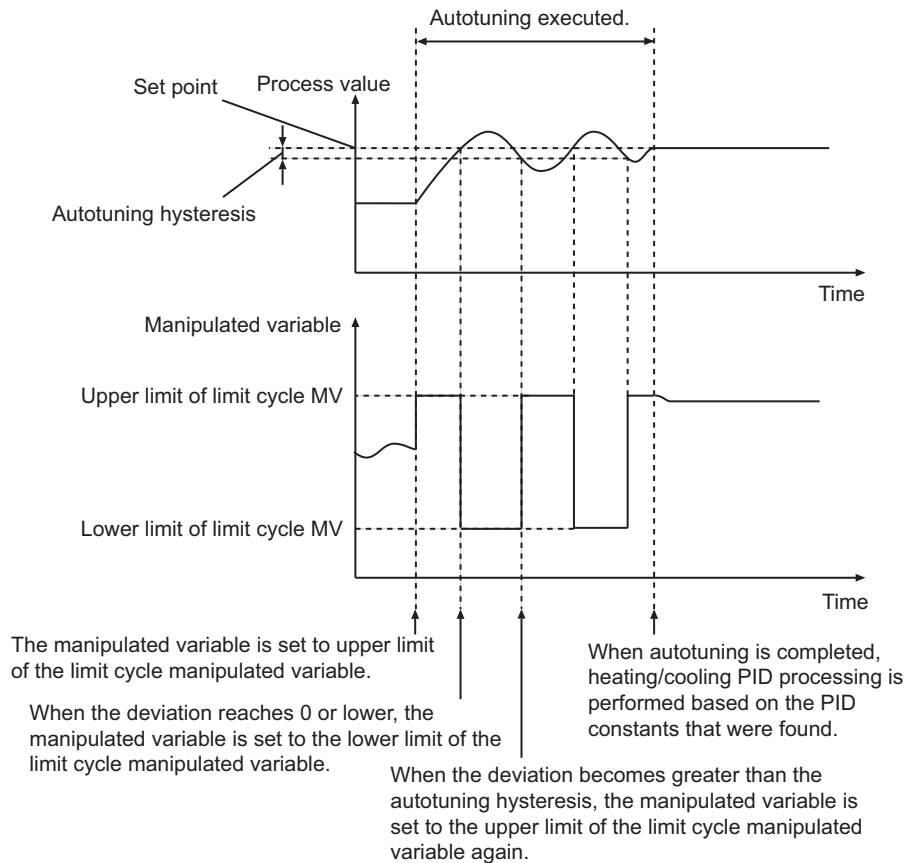
The PIDAT instruction supports autotuning of the PID constants.

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 the 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.

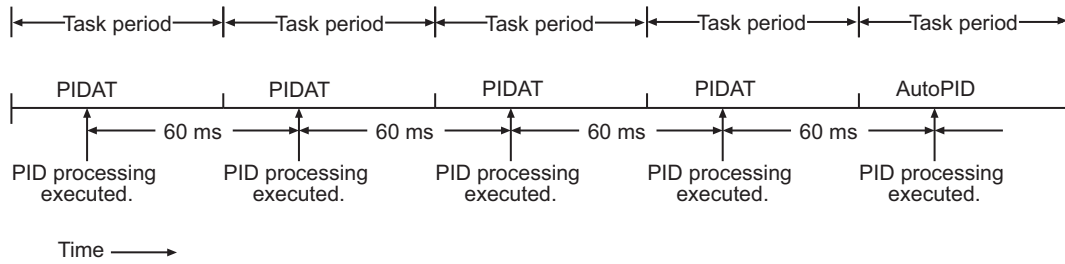
However, heating/cooling PID processing is not executed if the elapsed time since the last execution is shorter than *SampTime*.

If the elapsed time since the last execution exceeds *SampTime*, the excess time (elapsed time - *SampTime*) is carried forward to the next period. See below for details.

Even if this instruction is not executed as a result of the PrgStop or MC instruction, the elapsed time since the last execution of heating/cooling PID processing is set to 0 at the timing shown by *PID processing executed* in the following figures.

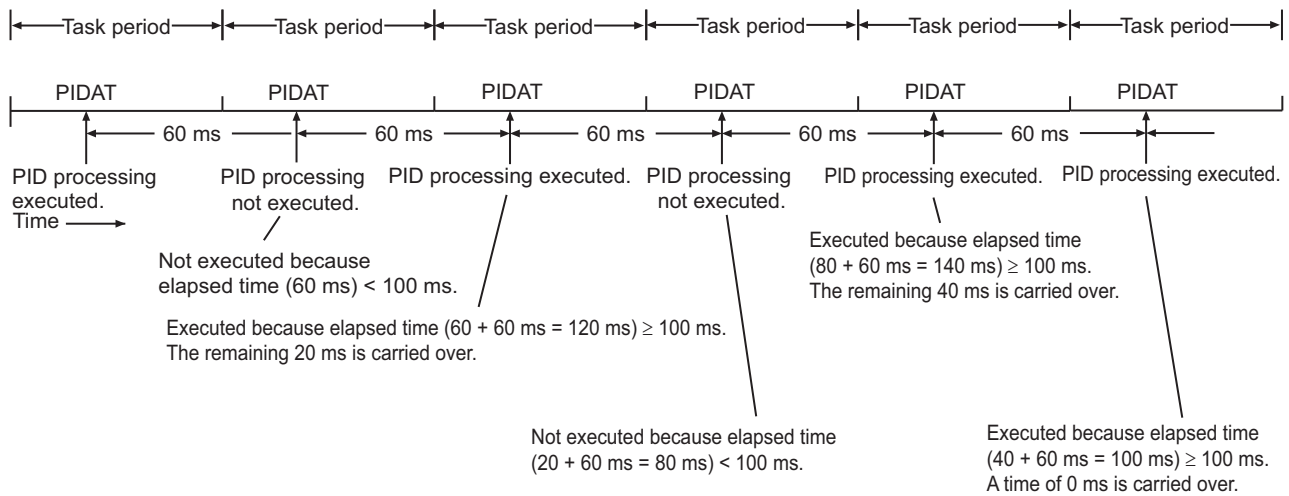
Task period = 60 ms and **SampTime** < 60 ms

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 PID processing is not executed every period.



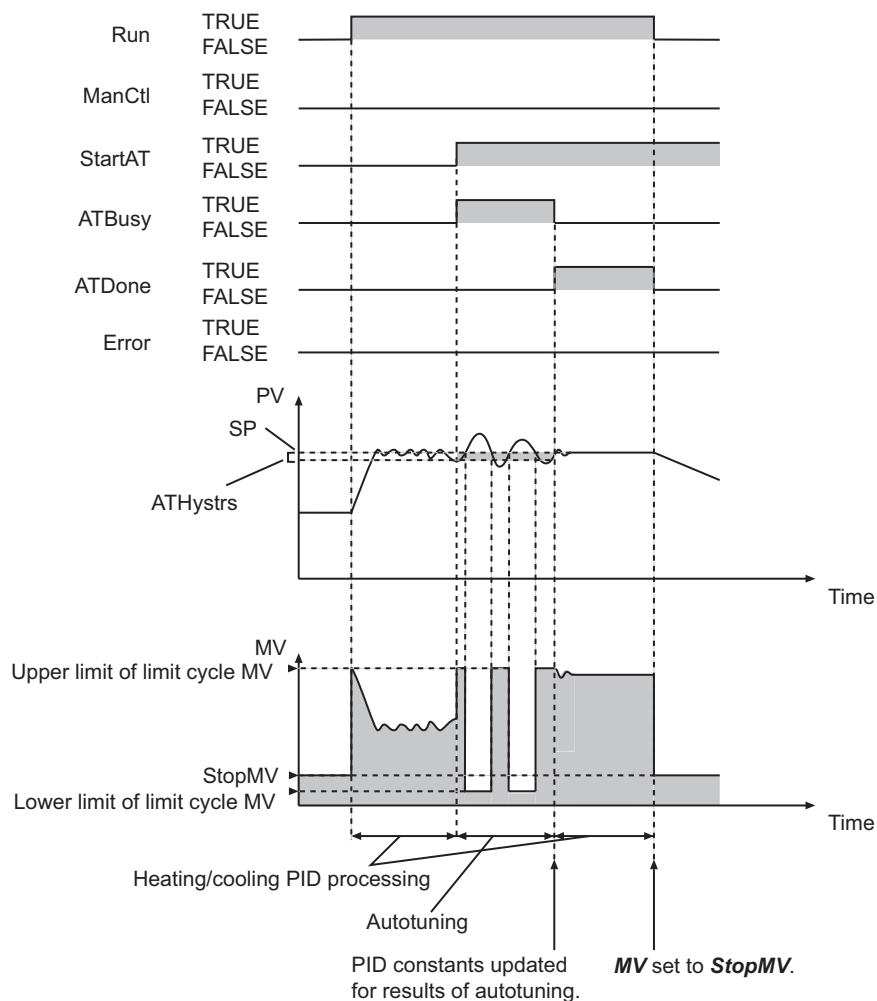
## Timing Charts

Timing charts for the instruction variables are provided below for different situations.

### ● Autotuning Executed during Automatic Operation

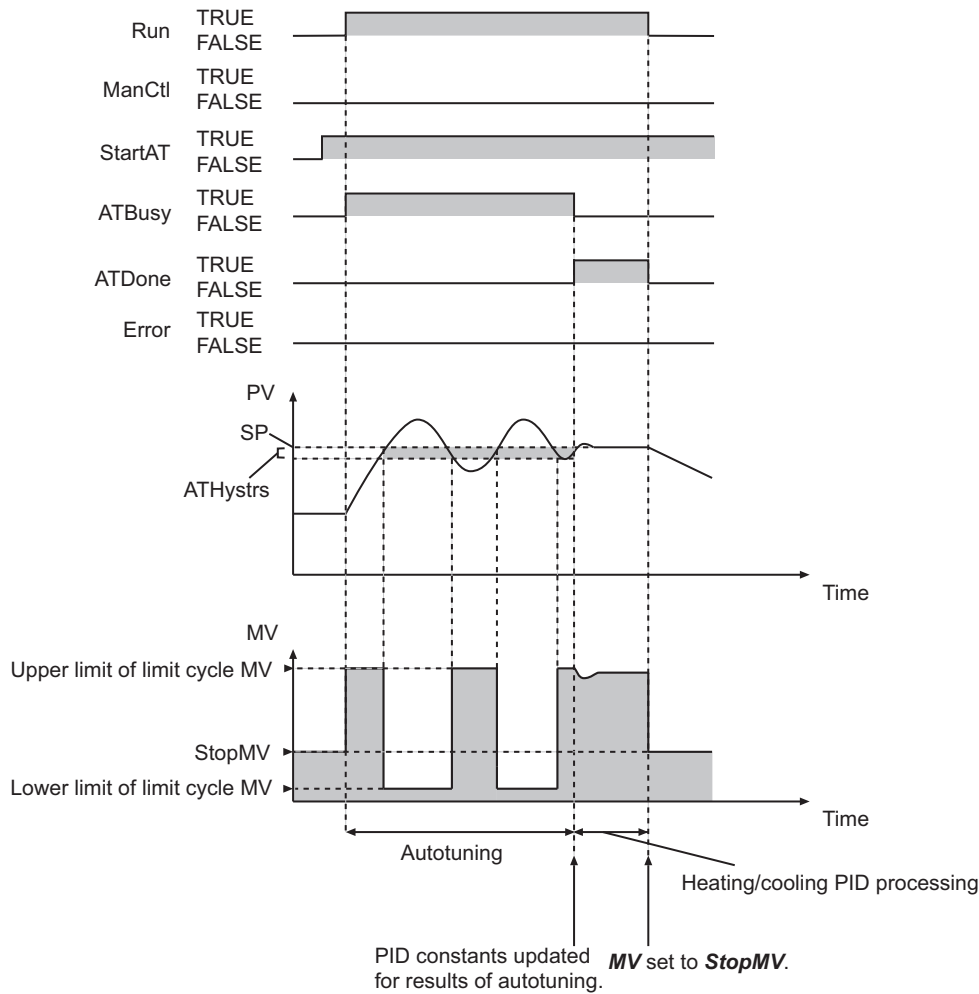
- In the following figure, the value of *ManCtl* is FALSE, so the value of *MV* will be *StopMV* as long as the value of *Run* is FALSE.
- When the value of *Run* changes to TRUE, *MV* is output based on the PID constants.
- Autotuning is executed when the value of *StartAT* changes to TRUE. The value 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.





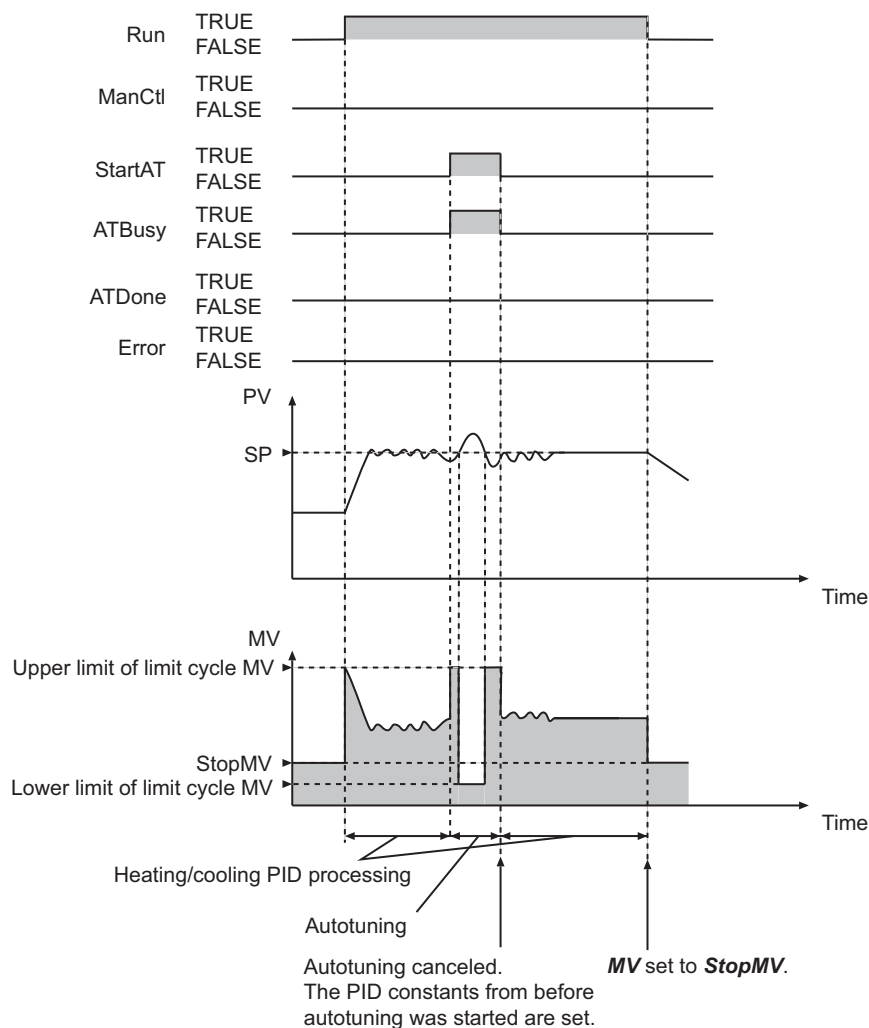
### ● Autotuning Executed at the Start of PIDAT Execution

- 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 FALSE, 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 through the 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 the autotuning. The value of *ATBusy* changes to FALSE.
- After the autotuning is canceled, *MV* is output based on the PID constants which were used just before the start of the autotuning.
- 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 the autotuning was aborted.



### ● An Autotuning Error Occurs during Autotuning

An autotuning error occurs and autotuning is stopped in the following cases.

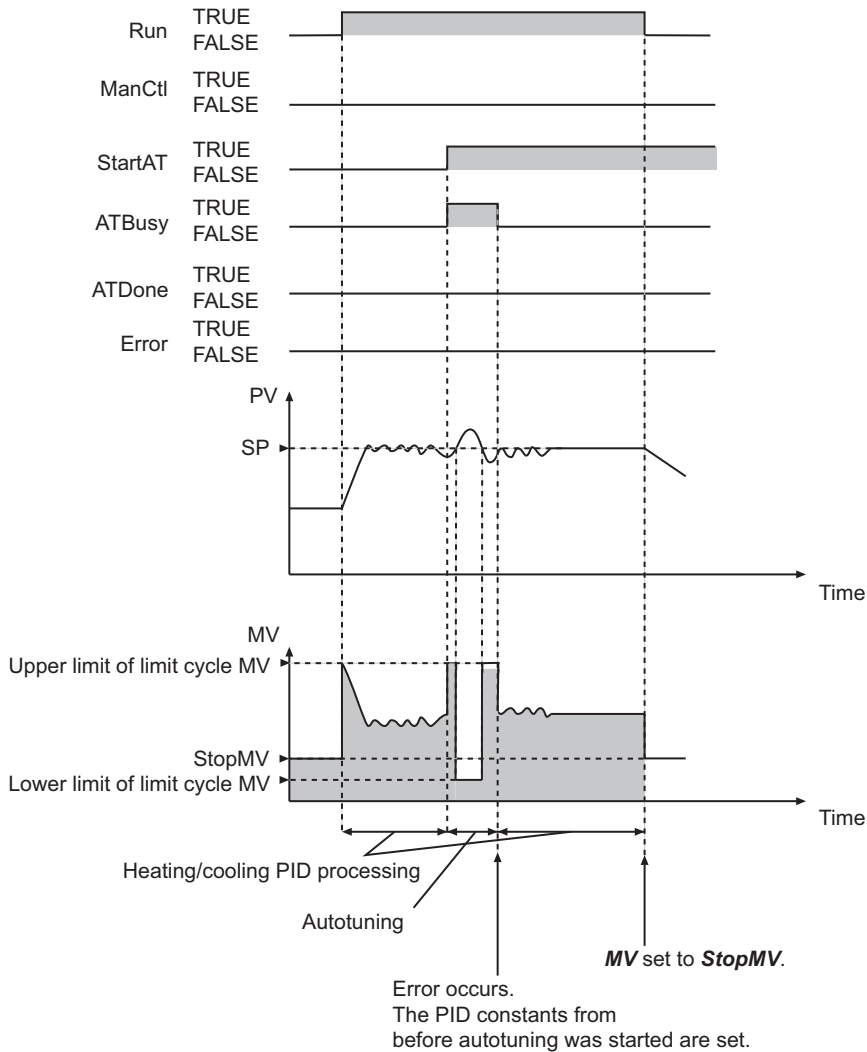
- If the manipulated variable equals the upper limit of the limit cycle manipulated variable and the time for the deviation to reach 0 exceeds 19,999 s.
- If the manipulated variable equals the lower limit of the limit cycle manipulated variable and the time for the deviation to reach  $ATH_{ystrs}$  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 the 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 the autotuning is canceled, *MV* is output based on the PID constants which were used just before the start of the autotuning.

- 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 the autotuning was aborted.



## Additional Information

### Adjusting PID Constants

Refer to *Adjusting PID Constants* on page 2-733 for the PIDAT instruction for details on 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                               |

| Variables                                   | Initial values (reference values)*1 |
|---|-------------------------------------|
| 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 <i>PV</i> and <i>SP</i>  | Values of <i>RngLowLmt</i> and <i>RngUpLmt</i>                              |
|---------------|--|---|
| % FS          | $PV = (\text{Process value in physical units} - \text{MIN}) / (\text{MAX} - \text{MIN}) \times 100^{*1}$<br>$SP = (\text{Set point in physical units} - \text{MIN}) / (\text{MAX} - \text{MIN}) \times 100^{*1}$ | <i>RngLowLmt</i> = 0<br><i>RngUpLmt</i> = 100                               |
| Physical unit | <i>PV</i> = Process value in physical units<br><i>SV</i> = Set point in physical units   | <i>RngLowLmt</i> = $\text{MIN}^{*1}$<br><i>RngUpLmt</i> = $\text{MAX}^{*1}$ |

\*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  |

| Variables             | Control status                              |   |   |
|-----------------------|---|---|---|
|                       | Instruction execution stopped <sup>*1</sup> | Automatic operation when autotuning is not being executed <sup>*2</sup> | Automatic operation when autotuning is being executed <sup>*3</sup> |
| Derivative Time_Heat  | Possible                                    | Possible  | Not possible <sup>*7</sup>  |
| ProportionalBand_Cool | Possible                                    | Possible  | Not possible <sup>*7</sup>  |
| IntegrationTime_Cool  | Possible                                    | Possible  | Not possible <sup>*7</sup>  |
| Derivative Time_Cool  | Possible                                    | Possible  | Not possible <sup>*7</sup>  |
| 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 the autotuning.
- The value of *Error* does not change to TRUE even if an error occurs during autotuning. Autotuning is not recorded in the event log.
- An error occurs in the following case.  
*Error* will change to TRUE, and the 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 the valid range, the value of *MV* is 0.

| Error  | Value of <i>ErrorID</i> |
|--|-------------------------|
| The value of an input variable is outside of the valid range.  | 16#0400                 |
| <i>RngLowLmt</i> is greater than or equal to <i>RngUpLmt</i> . | 16#0401                 |
| <i>MVLowLmt</i> is greater than or equal to <i>MVUpLmt</i> .   |                         |

- 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 constant values obtained through autotuning will revert to the previous values calculated before the backup operation. Use it with caution.
  - a) A Retain attribute is specified for the in-out parameters.

- b) 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.

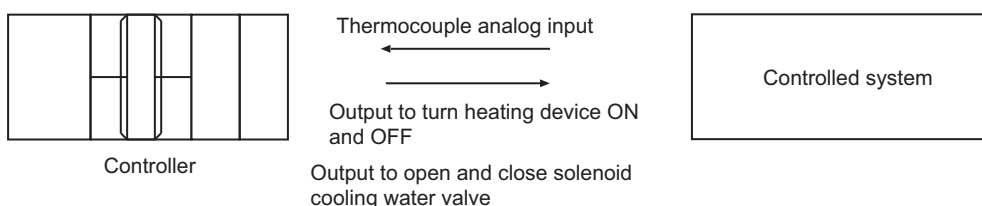
## Sample Programming

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.

### ● CJ1W-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 instruction, *TimeProportionalOut* on page 2-779 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.

## Definitions of Global Variables

### ● Global Variables

| Variable           | Data type | Initial value | AT                                      | Re-<br>tain              | Network<br>Publish  | Comment                               |
|--------------------|-----------|---------------|---|--------------------------|---------------------|---------------------------------------|
| J01_Ch1_AllnP<br>V | INT       | 0             | IOBus://rack#0/<br>slot#0/<br>Ch1_AllnP | <input type="checkbox"/> | Not pub-<br>lished. | Thermocouple input from<br>CJ1W-AD04U |



| Variable                      | Data type                  | Initial value  | AT  | Re-<br>tain                         | Network<br>Publish  | Comment  |
|-------------------------------|----------------------------|--|---|-------------------------------------|---------------------|--|
| J02_Ch1_Out00                 | BOOL                       | FALSE  | IOBus://rack#0/<br>slot#1/Ch1_Out/<br>Ch1_Out00 | <input type="checkbox"/>            | Not pub-<br>lished. | Heating output to CJ1W-<br>OC201   |
| J02_Ch1_Out04                 | BOOL                       | FALSE  | IOBus://rack#0/<br>slot#1/Ch1_Out/<br>Ch1_Out04 | <input type="checkbox"/>            | Not pub-<br>lished. | Cooling output to CJ1W-<br>OC201   |
| PTIn_Run                      | BOOL                       | FALSE  |   | <input checked="" type="checkbox"/> | Input               | Sample programming execu-<br>tion flag input from touch<br>panel           |
| PTIn_ManCtl                   | BOOL                       | FALSE  |   | <input checked="" type="checkbox"/> | Input               | Manual/auto control flag in-<br>put from touch panel                       |
| PTIn_SP                       | REAL                       |  |   | <input checked="" type="checkbox"/> | Input               | Set point input from touch<br>panel  |
| PTIn_StartAT                  | BOOL                       | FALSE  |   | <input checked="" type="checkbox"/> | Input               | Autotuning execution flag in-<br>put from touch panel                      |
| PTIn_Dead-<br>Band            | REAL                       | 0  |   | <input checked="" type="checkbox"/> | Input               | Deadband input from touch<br>panel   |
| PTIn_InitParam                | _sl-<br>NIT_SET_PA<br>RAMS | (SampTime :=<br>T#100 ms,<br>RngLowLmt := 0.0,<br>RngUpLmt :=<br>100.0, DirOpr :=<br>False)  |   | <input checked="" type="checkbox"/> | Input               | Initial setting parameter input<br>from touch panel                        |
| PTIn_InitSe-<br>tOpr_SampTime | LINT                       | 100  |   | <input checked="" type="checkbox"/> | Input               | Sampling period input from<br>touch panel (unit: ms)                       |
| PTIn_OprParam                 | _sOPR_SET<br>PARAMS        | (MVLowLmt :=<br>-100, MVUpLmt :=<br>100, ManResetV-<br>al := 0.0,<br>MVTrackSw :=<br>False, MVTrackV-<br>al := 0.0,<br>StopMV := 0.0, Er-<br>rorMV := 0.0, Al-<br>pha := 0.65, AT-<br>CalcGain := 1.0,<br>ATHystrs := 0.2) |   | <input checked="" type="checkbox"/> | Input               | Operation setting parameter<br>input from touch panel                      |
| PTOut_PV                      | REAL                       | 0  |   | <input type="checkbox"/>            | Output              | Process value output to<br>touch panel                                     |
| PT_PB_Heat                    | REAL                       | 1  |   | <input checked="" type="checkbox"/> | Input               | Proportional band for heat-<br>ing control I/O from touch<br>panel         |
| PT_TI_Heat                    | LINT                       | 1000   |   | <input checked="" type="checkbox"/> | Input               | Integration time for heating<br>control I/O from touch panel<br>(unit: ms) |
| PT_TD_Heat                    | LINT                       | 1000   |   | <input checked="" type="checkbox"/> | Input               | Derivative time for heating<br>control I/O from touch panel<br>(unit: ms)  |
| PT_PB_Cool                    | REAL                       | 1  |   | <input checked="" type="checkbox"/> | Input               | Proportional band for cooling<br>control I/O from touch panel              |

| Variable     | Data type | Initial value | AT | Re-<br>tain                         | Network<br>Publish | Comment  |
|--------------|-----------|---------------|----|-------------------------------------|--------------------|--|
| PT_TI_Cool   | LINT      | 1000          |    | <input checked="" type="checkbox"/> | Input              | Integration time for cooling control I/O from touch panel (unit: ms) |
| PT_TD_Cool   | LINT      | 1000          |    | <input checked="" type="checkbox"/> | Input              | Derivative time for cooling control I/O from touch panel (unit: ms)  |
| PT_ManMV     | REAL      | 0             |    | <input checked="" type="checkbox"/> | Input              | Manual manipulated variable I/O from touch panel                     |
| PTOut_ATDone | BOOL      | FALSE         |    | <input type="checkbox"/>            | Output             | Autotuning normal completion flag output to touch panel              |
| PTOut_ATBusy | BOOL      | FALSE         |    | <input type="checkbox"/>            | Output             | Autotuning executing flag output to touch panel                      |
| PTOut_Error  | BOOL      | FALSE         |    | <input type="checkbox"/>            | Output             | Error flag output to touch panel                                     |
| PTOut_MV     | REAL      | 0             |    | <input type="checkbox"/>            | Output             | Manipulated variable output to touch panel                           |
| PTOut_MVHeat | REAL      | 0             |    | <input type="checkbox"/>            | Output             | Manipulated variable for heating control output to touch panel       |
| PTOut_MVCool | REAL      | 0             |    | <input type="checkbox"/>            | Output             | Manipulated variable for cooling control output to touch panel       |

## LD

| Internal Variables | Variable            | Data type           | Initial value | Comment   |
|--------------------|---------------------|---------------------|---------------|---|
|                    | PB_Heat             | REAL                | 0             | Proportional band for heating control                           |
|                    | PB_Cool             | REAL                | 0             | Proportional band for cooling control                           |
|                    | 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#0 s         | Integration time for heating control                            |
|                    | TI_Cool             | TIME                | T#0 s         | Integration time for cooling control                            |
|                    | TD_Heat             | TIME                | T#0 s         | Derivative time for heating control                             |
|                    | TD_Cool             | TIME                | T#0 s         | Derivative time for cooling control                             |
|                    | ManMV               | REAL                | 0             | Manual manipulated variable                                     |
|                    | CtlPrd_Cool         | TIME                | T#20 s        | Cooling control period  |
|                    | CtlPrd_Heat         | TIME                | T#2 s         | Heating control period  |
|                    | TPOHeat_inst        | TimeProportionalOut |               | Instance of TimeProportionalOut instruction for heating control |

| Internal Variables | Variable               | Data type           | Initial value | Comment   |
|--------------------|------------------------|---------------------|---------------|---|
|                    | TPOCool_inst           | TimeProportionalOut |               | Instance of TimeProportionalOut instruction for cooling control                               |
|                    | ATHeatPhase            | BOOL                | FALSE         | Autotuning heating control flag   |
|                    | ATCoolPhase            | BOOL                | FALSE         | Autotuning cooling control flag   |
|                    | MVHeatTime             | TIME                | T#0 s         | Autotuning heating control time   |
|                    | MVCoolTime             | TIME                | T#0 s         | Autotuning cooling control time   |
|                    | AT_Heat_inst           | TP                  |               | Instance of TP instruction for heating control manipulated variable output during autotuning  |
|                    | AT_Cool_inst           | TP                  |               | Instance of TP instruction for cooling control manipulated variable output during autotuning  |
|                    | EachCtlPrd_ATHeat_inst | TON                 |               | Instance of TON instruction for heating control manipulated variable output during autotuning |
|                    | EachCtlPrd_ATCool_inst | TON                 |               | Instance of TON instruction for cooling control manipulated variable output during autotuning |
|                    | PV                     | REAL                | 0             | Process value   |

| External Variables | Variable                 | Data type         | Comment  |
|--------------------|--------------------------|-------------------|--|
|                    | J01_Ch1_AllInPV          | 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)  |

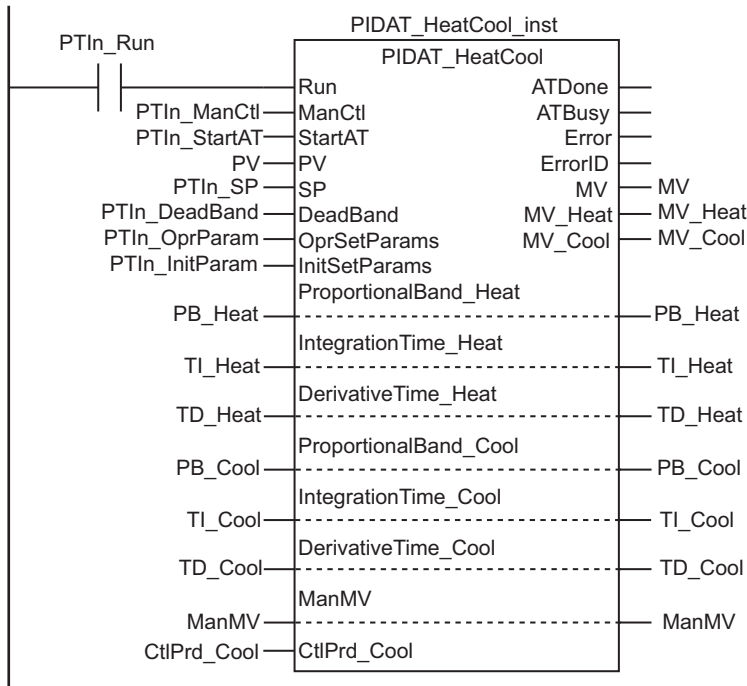
| External Variables | Variable     | Data type | Comment  |
|--------------------|--------------|-----------|--|
|                    | PT_PB_Cool   | REAL      | Proportional band for cooling control I/O from touch panel           |
|                    | PT_TI_Cool   | LINT      | Integration time 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.

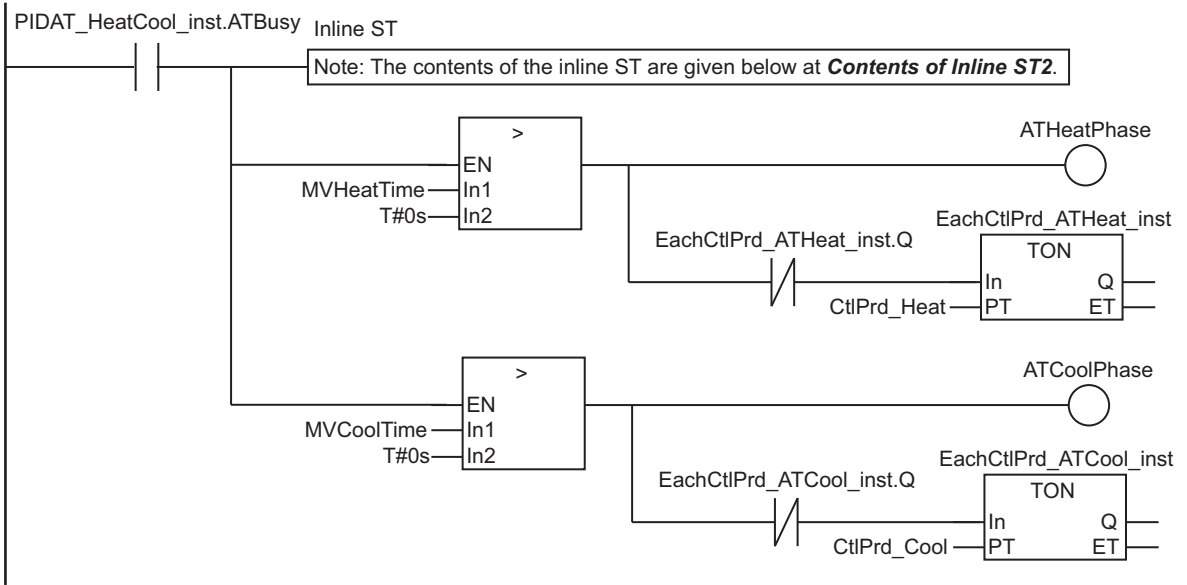
Inline ST

Note: The contents of the inline ST are given below at **Contents of Inline ST1**.

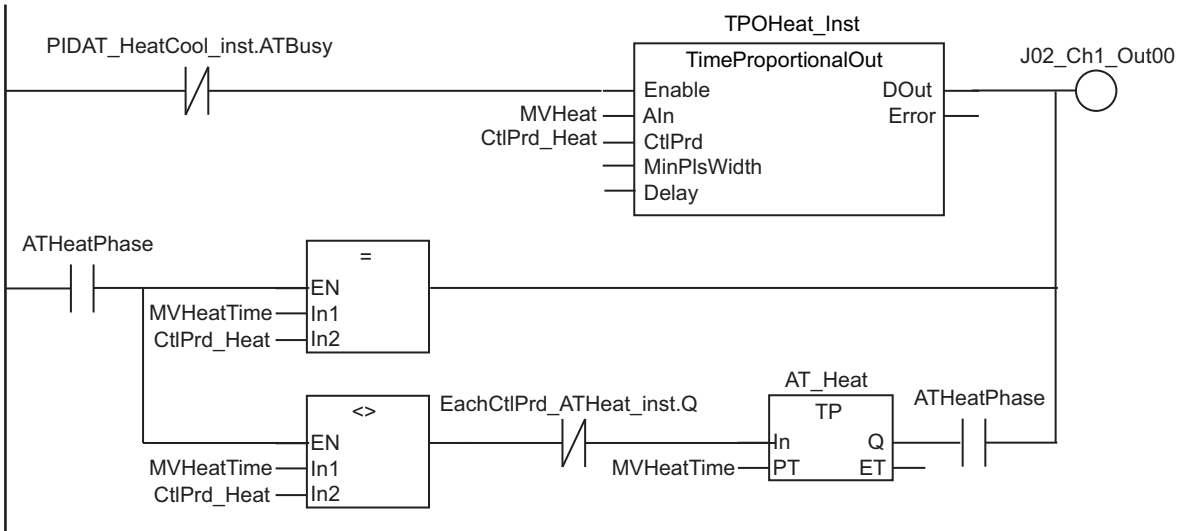
Execute PIDAT\_HeatCool instruction.



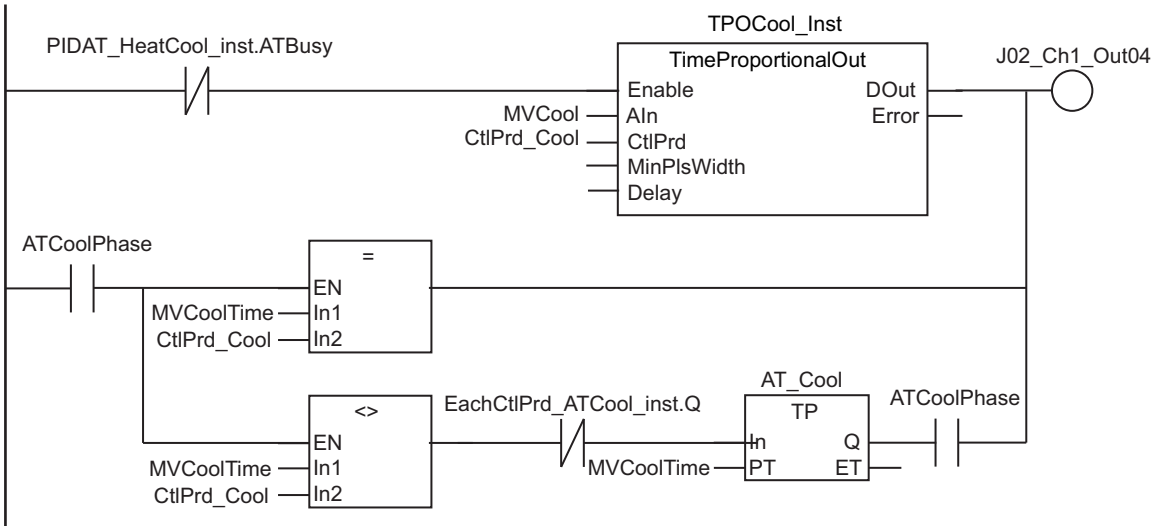
Prepare to convert to time-proportional outputs during execution of autotuning.



Heating output to CJ1W-OC201



Cooling output to CJ1W-OC201



Create output values to touch panel.

Inline ST

Note: The contents of the inline ST are given below at **Contents of Inline ST3**.

### ● 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 Variables | Variable               | Data type           | Initial value | Comment   |
|--------------------|------------------------|---------------------|---------------|---|
|                    | PB_Heat                | REAL                | 0             | Proportional band for heating control   |
|                    | PB_Cool                | REAL                | 0             | Proportional band for cooling control   |
|                    | 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#0 s         | Integration time for heating control  |
|                    | TI_Cool                | TIME                | T#0 s         | Integration time for cooling control  |
|                    | TD_Heat                | TIME                | T#0 s         | Derivative time for heating control   |
|                    | TD_Cool                | TIME                | T#0 s         | Derivative time for cooling control   |
|                    | ManMV                  | REAL                | 0             | Manual manipulated variable   |
|                    | CtlPrd_Cool            | TIME                | T#20 s        | Cooling control period  |
|                    | CtlPrd_Heat            | TIME                | T#2 s         | Heating control period  |
|                    | TPOHeat_inst           | TimeProportionalOut |               | Instance of TimeProportionalOut instruction for heating control                               |
|                    | TPOCool_inst           | TimeProportionalOut |               | Instance of TimeProportionalOut instruction for cooling control                               |
|                    | ATHeatPhase            | BOOL                | FALSE         | Autotuning heating control flag   |
|                    | ATCoolPhase            | BOOL                | FALSE         | Autotuning cooling control flag   |
|                    | MVHeatTime             | TIME                | T#0 s         | Autotuning heating control time   |
|                    | MVCoolTime             | TIME                | T#0 s         | Autotuning cooling control time   |
|                    | AT_Heat_inst           | TP                  |               | Instance of TP instruction for heating control manipulated variable output during autotuning  |
|                    | AT_Cool_inst           | TP                  |               | Instance of TP instruction for cooling control manipulated variable output during autotuning  |
|                    | EachCtlPrd_ATHeat_inst | TON                 |               | Instance of TON instruction for heating control manipulated variable output during autotuning |
|                    | EachCtlPrd_ATCool_inst | TON                 |               | Instance of TON instruction for cooling control manipulated variable output during autotuning |
|                    | PV                     | REAL                | 0             | Process value   |

| External Variables | Variable       | Data type | Comment                            |
|--------------------|----------------|-----------|------------------------------------|
|                    | J01_Ch1_AllnPv | INT       | Thermocouple input from CJ1W-AD04U |
|                    | J02_Ch1_Out00  | BOOL      | Heating output to CJ1W-OC201       |
|                    | J02_Ch1_Out04  | BOOL      | Cooling output to CJ1W-OC201       |

| External Variables | Variable                 | Data type         | Comment  |
|--------------------|--------------------------|-------------------|--|
|                    | 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_i
nst.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_i
nst.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/<br>FUN | Graphic expression | ST expression  |
|---------------------|--------------------------|------------|--------------------|--|
| TimeProportionalOut | Time-proportional Output | FB         |                    | TimeProportionalOut_instance( Enable, Aln, CtlPrd, MinPlsWidth, Delay, DOut, Error); |



## 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.

## Variables

|             | Meaning                  | I/O    | Description  | Valid range           | Unit | Default |
|-------------|--------------------------|--------|--|-----------------------|------|---------|
| Enable      | Enable                   | Input  | TRUE: Execute<br>FALSE: Reset time-proportional output                           | Depends on data type. | ---  | FALSE   |
| Aln         | Manipulated variable     |        | Manipulated variable   | 0 to 100              | %    | 0       |
| CtlPrd      | Control period           |        | Control period of time-proportional output                                       | T#0.1 s to T#100 s    | s    | T#2 s   |
| 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.<br>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      |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |  |
| Aln         |         |             |      |       |       |          |      |       |       |      |     |      |      | 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.

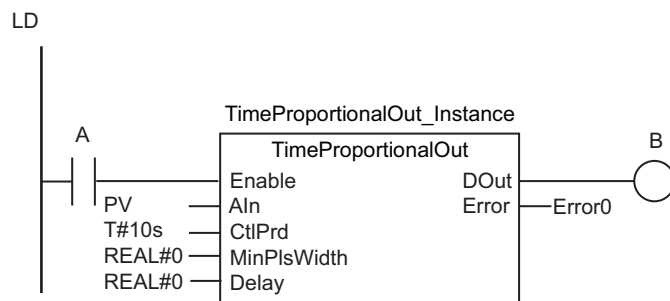
A time-proportional output converts a manipulated variable to a time ratio between ON and OFF.

While *Enable* is TRUE, the value of manipulated variable *Aln* is converted to time-proportional output *DOut* for control period *CtlPrd*.

If *Enable* changes to FALSE, the time-proportional output is reset.

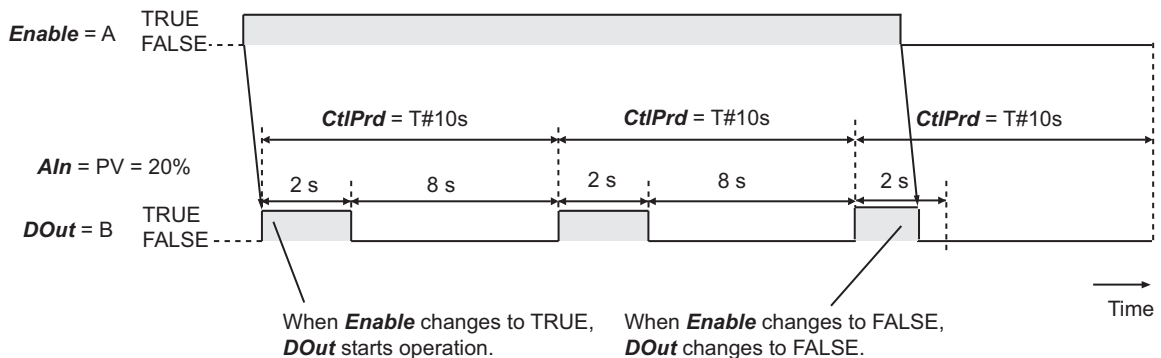
*DOut* and *Error* change to FALSE. The values of *CtlPrd*, *MinPlsWidth*, and *Delay* are updated when *Enable* changes from FALSE to TRUE.

The following example is for when the value of *CtlPrd* is 10 s and the value of *Aln* is 20%. While *Enable* is TRUE, *DOut* is TRUE for two seconds and then FALSE for eight seconds. This is repeated in a 10-second cycle.



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 *Aln* to *DOut* is referred to as the resolution of *DOut*.

If the resolution of the value of *Aln* is higher than the resolution of *DOut*, *Aln* is rounded according to the resolution of *DOut* when it is converted to *DOut*.

The resolution of *DOut* is given by the following formula.

$$\text{Resolution of } DOut (\%) = \text{Task period} / CtlPrd \times 100$$

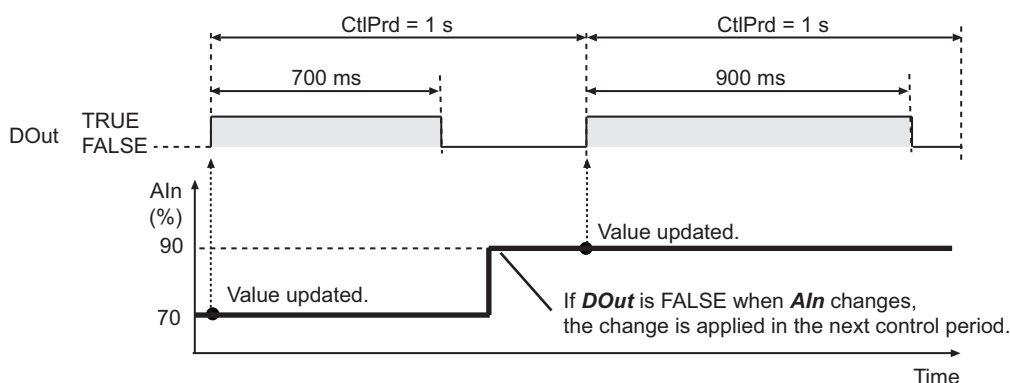
For example, if the task period is 1 ms and the value of *CtlPrd* is 1 s, the resolution of *DOut* is 0.1%. In this case, the value of *AIn* is rounded to one decimal place.

## Update Timing of the Value of Manipulated Variable *AIn*

When the 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.

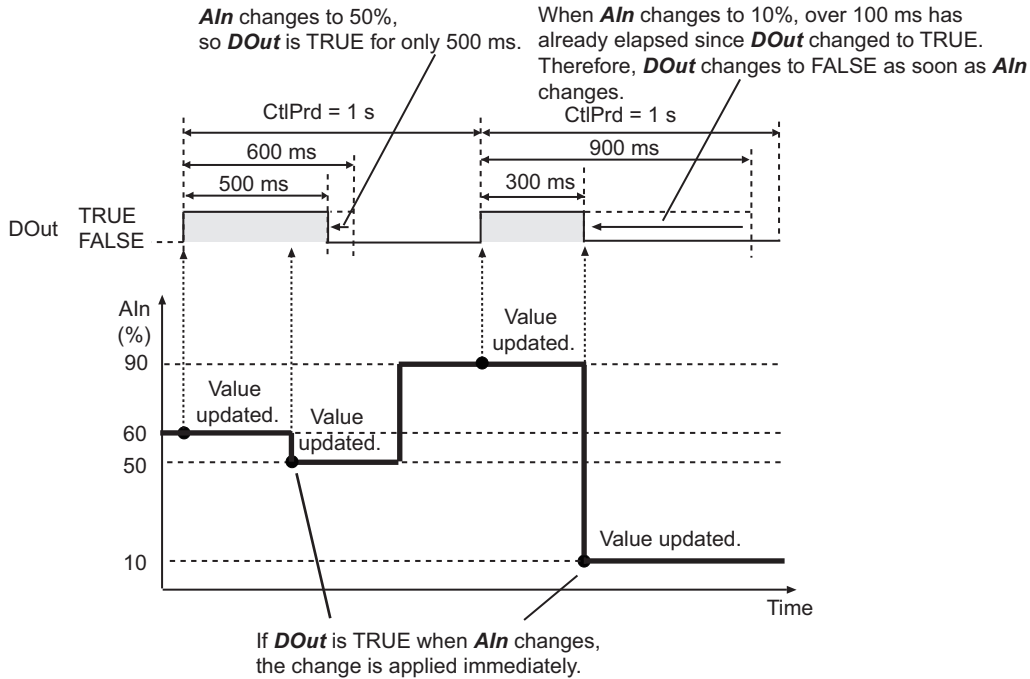


### ● *DOut* = TRUE

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.

- Assume that the value of *AIn* is 60% at the start of the control period. If the value of *AIn* changes to 50% while *DOut* is TRUE, *DOut* stays TRUE only for 500 ms.
- Assume that the value of *AIn* is 90% at the start of the control period, and that the value of *AIn* changes to 10% 300 ms after *DOut* changes to TRUE. In this case, 100 ms, which is equivalent to 10% of the control period, 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*.

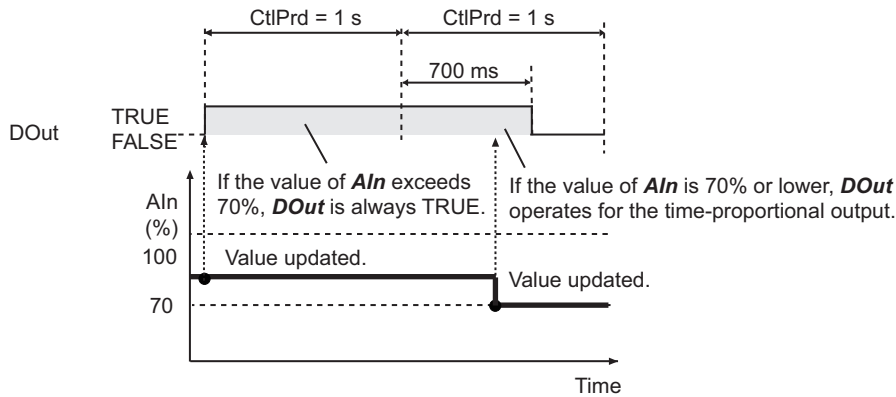
For example, if the number of times a fan is turned ON and OFF is reduced in cooling control, power consumption is reduced.

The following table shows the operation of *DOut* for the relationship between the values of *MinPlsWidth* and *AIn*.

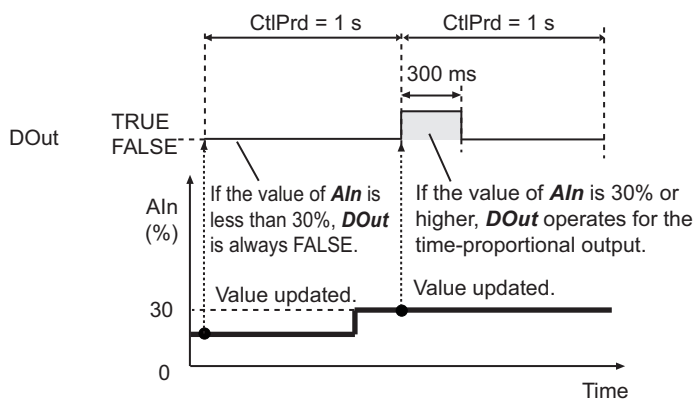
| Relationship between the values of <i>MinPlsWidth</i> and <i>AIn</i> | Operation of <i>DOut</i> |
|--|--------------------------|
| $AIn < MinPlsWidth$  | Always FALSE             |
| $MinPlsWidth \leq AIn \leq 100 - MinPlsWidth$                        | Time-proportional output |
| $AIn > 100 - MinPlsWidth$  | Always TRUE              |

For example, the following figure shows the operation of *DOut* when *MinPlsWidth* is 30%.

If the value of *AIn* is higher than 70%, the output is always TRUE. When the value is 70% or lower, the time-proportional operation is performed for the output.



If the value of *AIn* is lower than 30%, the output is always FALSE. When the value is 30% or higher, the time-proportional operation is performed for the output.



## Operation of Time-proportional Output *DOut* for Delay

The delay prevents *DOut* from changing to TRUE until the set time has elapsed since the start of the control period.

If more than one *TimeProportionalOut* instruction is executed, you can specify *Delay* to change *DOut* to TRUE at different timings for each execution.

This reduces the chance that *DOut* will turn ON simultaneously for more than one instruction.

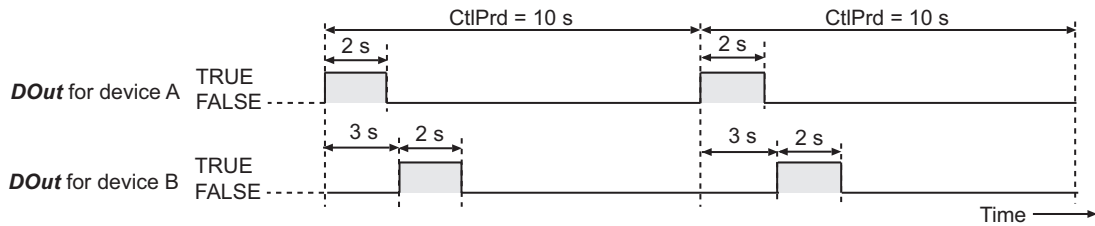
For example, if you operate more than one heating device, you can specify *Delay* with a different value for each device so that the output to each heating device will be turned ON at different timings, and thus the power to be consumed at a time can be reduced.

*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 <i>Delay</i> | Value of <i>AIn</i> | Value of <i>CtlPrd</i> |
|----------|-----------------------|---------------------|------------------------|
| Device A | 0%                    | 20%                 | 10 s                   |
| Device B | 30%                   |                     |                        |

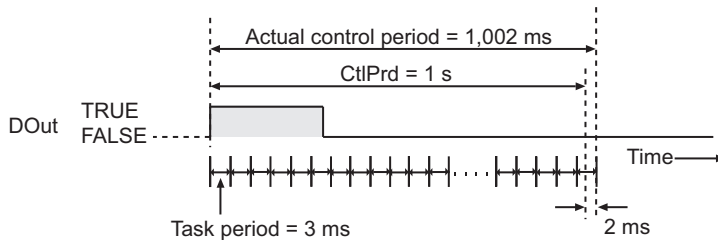
*DOut* for device A changes to TRUE at the start of the control period. *DOut* for device B changes to TRUE three seconds after the start of the control period.



## Precautions for Correct Use

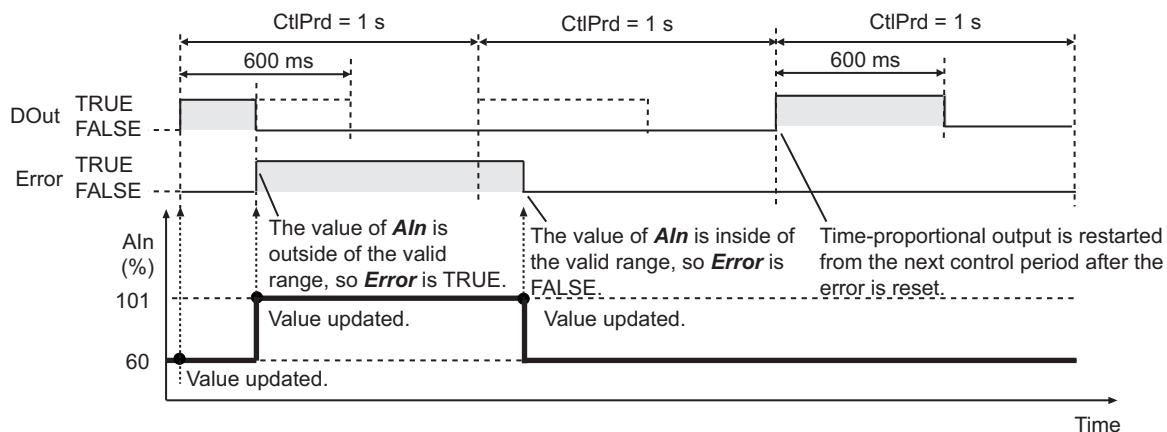
- Set the value of control period  $CtlPrd$  to a multiple of the task period of the task to which the program is assigned.

If the task period is not set to a multiple of  $CtlPrd$ , the actual control period will be from when the control period 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).

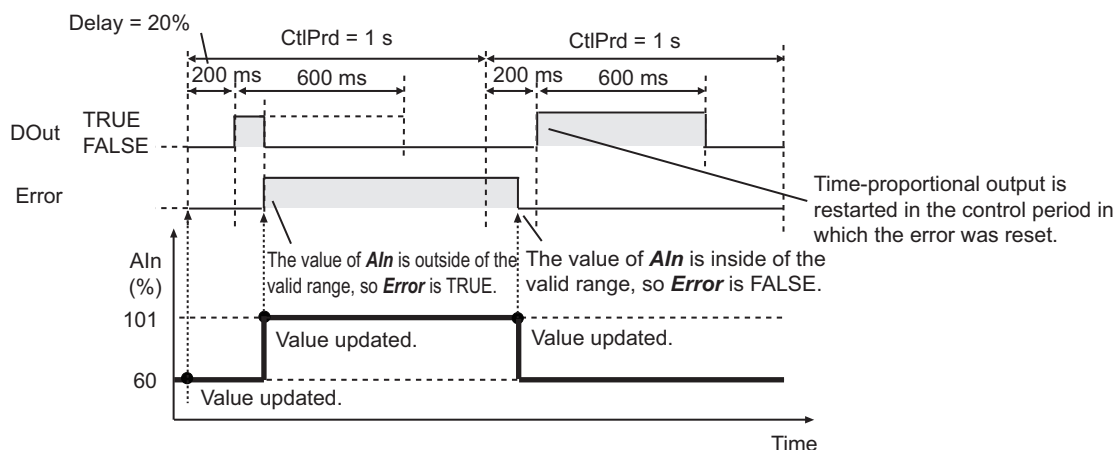


- Set the task period and control period  $CtlPrd$  so that the resolution of  $DOut$  is 0.1% or less. If the resolution of  $DOut$  exceeds 0.1%, the error between the ratio when  $DOut$  is TRUE and the value of  $AIN$  will be excessive, and control performance will decrease. For example, if  $CtlPrd$  is 10 s, set the task period to 10 ms or shorter.
- 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 until when operation starts for  $DOut$  is not constant.
- An error occurs if the value of  $AIN$ ,  $CtlPrd$ ,  $MinPisWidth$ , or  $Delay$  is outside 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.





b) 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 is reset.



## 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   |

| Item                                      | Specification |
|---|---------------|
| 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.

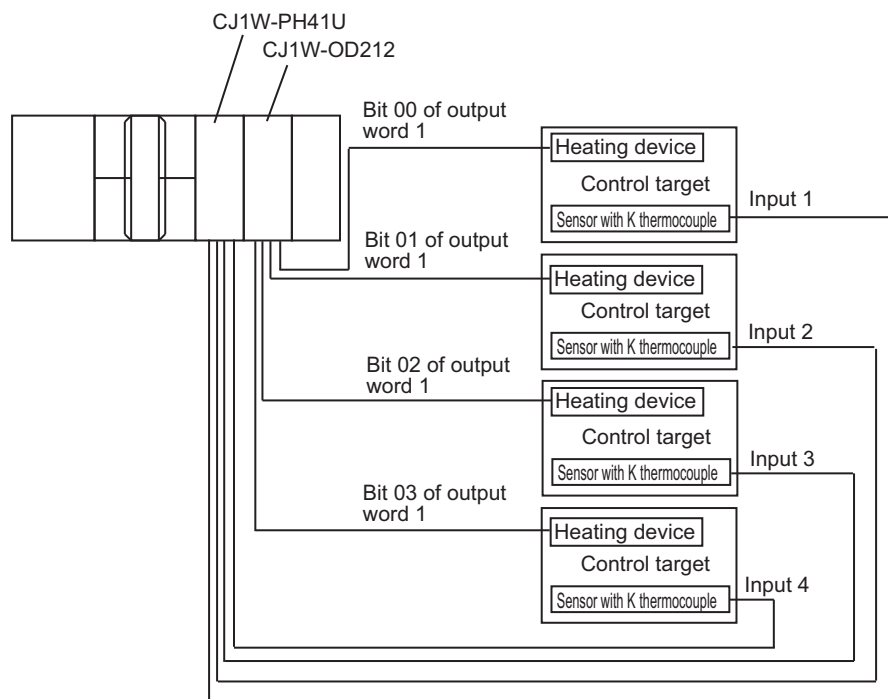
| Unit       | I/O port   | Description                          | Variable |
|------------|------------|--------------------------------------|----------|
| CJ1W-PH41U | Ch1_AllnPV | Process value for input 1 (INT data) | AI1      |
|            | Ch2_AllnPV | Process value for input 2 (INT data) | AI2      |
|            | Ch3_AllnPV | Process value for input 3 (INT data) | AI3      |
|            | Ch4_AllnPV | 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

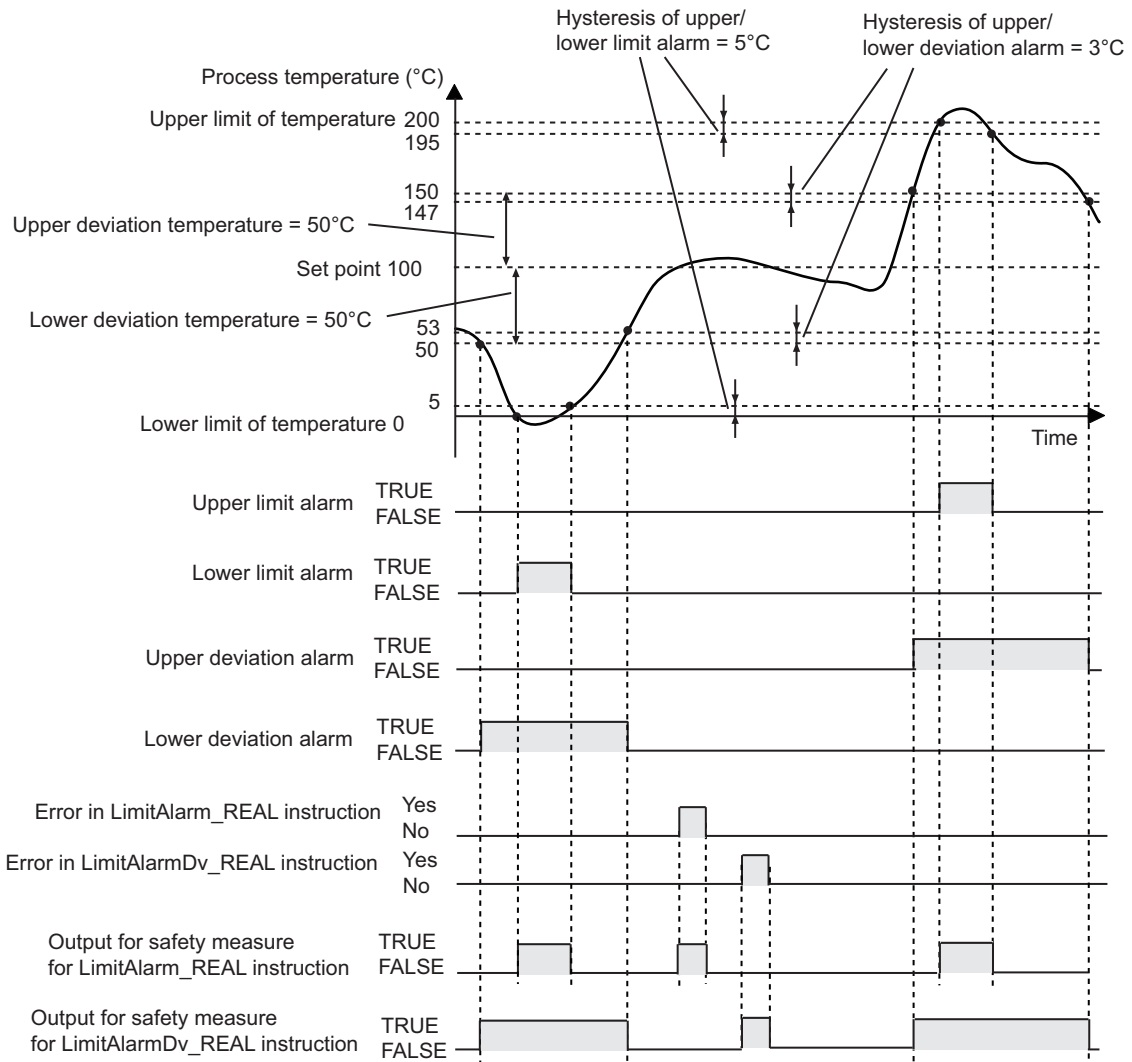


## Processing

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



Definitions of Global Variables

● Global Variables

| Variable | Data type | AT specification*1                      | Comment                              |
|----------|-----------|---|--------------------------------------|
| AI1      | INT       | IOBus://rack#0/slot#0/Ch1_AllnPv        | Process value for input 1 (INT data) |
| AI2      | INT       | IOBus://rack#0/slot#0/Ch2_AllnPv        | Process value for input 2 (INT data) |
| AI3      | INT       | IOBus://rack#0/slot#0/Ch3_AllnPv        | Process value for input 3 (INT data) |
| AI4      | INT       | IOBus://rack#0/slot#0/Ch4_AllnPv        | 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. This table shows the variables for the CJ1W-PH41U Input Unit mounted to Slot #0 of Rack #0, and the CJ1W-OD212 Output Unit mounted to Slot #1 of the same rack.

**Note** The global variables for the port of each Unit are automatically generated based on the I/O mapping settings.

## LD

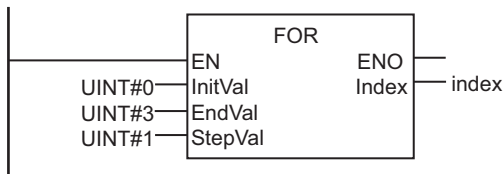
| Internal Variables | Name                    | Data type           | Default    | Retain                   | Comment   |
|--------------------|-------------------------|---------------------|------------|--------------------------|---|
|                    | index                   | UINT                | 0          | <input type="checkbox"/> | Loop index  |
|                    | LimitAlarm_ON           | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Limit Alarm instruction                  |
|                    | LimitAlarmDv_ON         | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Deviation Alarm instruction              |
|                    | TimeProportional-Out_ON | BOOL                | True       | <input type="checkbox"/> | Execution of TimeproportionalOutput instruction                   |
|                    | AI                      | INT                 | 0          | <input type="checkbox"/> | Present value   |
|                    | PV                      | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Process value   |
|                    | SP                      | ARRAY[0..3] OF REAL | [4(100)]   | <input type="checkbox"/> | Set point   |
|                    | DOut_TPO                | BOOL                | False      | <input type="checkbox"/> | Time-proportional output  |
|                    | HighVal                 | ARRAY[0..3] OF REAL | [4(200)]   | <input type="checkbox"/> | Upper limit set value of upper/lower limit alarm                  |
|                    | LowVal                  | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Lower limit set value of upper/lower limit alarm                  |
|                    | Hystrs_LimitAlarm       | ARRAY[0..3] OF REAL | [4(5)]     | <input type="checkbox"/> | Hysteresis of upper/lower limit alarm                             |
|                    | Q_LimitAlarm            | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower limit alarm output                                    |
|                    | HighAlm                 | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper limit alarm   |
|                    | LowAlm                  | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower limit alarm   |
|                    | Error_LimitAlarm        | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Error in LimitAlarm_REAL instruction                              |
|                    | Alm_LimitAlarm          | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Output for safety measure for Upper/Lower Limit Alarm instruction |
|                    | DvHighVal               | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Upper deviation set value of upper/lower deviation alarm          |
|                    | DvLowVal                | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Lower deviation set value of upper/lower deviation alarm          |
|                    | Q_LimitAlarmDv          | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower deviation alarm output                                |
|                    | HighAlmDv               | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper deviation alarm   |

| Internal Variables | Name                 | Data type           | Default   | Retain                              | Comment   |
|--------------------|----------------------|---------------------|---|-------------------------------------|---|
|                    | LowAlmDv             | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Lower deviation alarm   |
|                    | Error_LimitAlarmDv   | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Error in LimitAlarmDv_REAL instruction                                |
|                    | Hysters_LimitAlarmDv | ARRAY[0..3] OF REAL | [4(3)]  | <input type="checkbox"/>            | Hysteresis of upper/lower deviation alarm                             |
|                    | Alm_LimitAlarmDv     | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Output for safety measure for Upper/Lower Deviation Alarm instruction |
|                    | Run                  | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Execution condition   |
|                    | ManCtl               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Manual/auto control   |
|                    | StartAT              | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | 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, ATHysters:=0.2) | <input type="checkbox"/>            | Operation setting parameters  |
|                    | InitSetParams        | _sINIT_SET_PARAMS   | (SampTime:=T#100 ms, RngLowLmt:=-10.0, RngUpLmt:=1000.0, DirOpr:=False)   | <input type="checkbox"/>            | Initial setting parameters  |
|                    | PB                   | ARRAY[0..3] OF REAL | [4(10)]   | <input checked="" type="checkbox"/> | Proportional band   |
|                    | TI                   | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Integration time  |
|                    | TD                   | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Derivative time   |
|                    | ManMV                | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Manual manipulated variable   |
|                    | ATDone               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning normal completion  |
|                    | ATBusy               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning busy   |
|                    | Error_PIDAT          | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Error in PIDAT instruction  |
|                    | ErrorID              | ARRAY[0..3] OF WORD | [4(16#0)]   | <input type="checkbox"/>            | Error ID for PIDAT instruction  |
|                    | MV                   | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Manipulated variable  |
|                    | CtlPrd               | ARRAY[0..3] OF TIME | [4(T#1 s)]  | <input type="checkbox"/>            | Control period  |
|                    | MinPlsWidth          | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Minimum pulse width   |
|                    | Delay                | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | ON-delay time   |

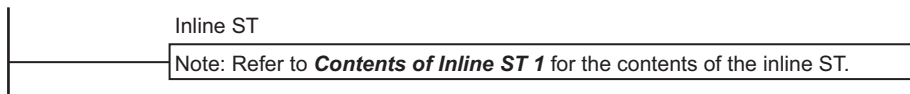
| Internal Variables | Name                         | Data type                          | Default    | Retain                   | Comment                                  |
|--------------------|------------------------------|------------------------------------|------------|--------------------------|--|
|                    | Error_TimeProportionalOut    | ARRAY[0..3] OF BOOL                | [4(False)] | <input type="checkbox"/> | Error in TimeProportionalOut instruction |
|                    | LimitAlarm_REAL_instance     | ARRAY[0..3] OF LimitAlarm_REAL     |            | <input type="checkbox"/> |  |
|                    | LimitAlarmDv_REAL_instance   | ARRAY[0..3] OF LimitAlarmDv_REAL   |            | <input type="checkbox"/> |  |
|                    | PIDAT_instance               | ARRAY[0..3] OF PIDAT               |            | <input type="checkbox"/> |  |
|                    | TimeProportionalOut_instance | ARRAY[0..3] OF TimeProportionalOut |            | <input type="checkbox"/> |  |

| External Variables | Name | Data type | Comment                    |
|--------------------|------|-----------|----------------------------|
|                    | AI1  | INT       | Input No.1 (Process value) |
|                    | AI2  | INT       | Input No.2 (Process value) |
|                    | AI3  | INT       | Input No.3 (Process value) |
|                    | AI4  | INT       | Input No.4 (Process value) |
|                    | DO1  | BOOL      | output word 1 (Bit 00)     |
|                    | DO2  | BOOL      | output word 1 (Bit 01)     |
|                    | DO3  | BOOL      | output word 1 (Bit 02)     |
|                    | DO4  | BOOL      | output word 1 (Bit 03)     |

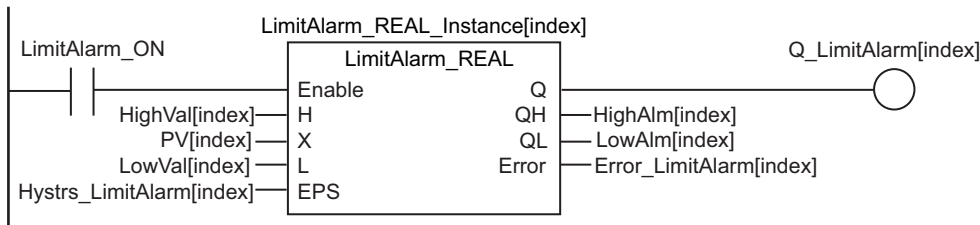
Control temperature for four points.



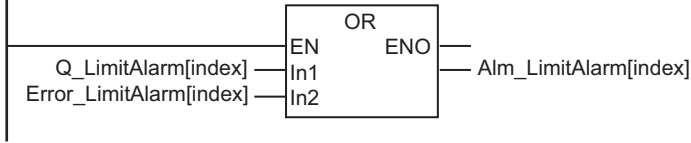
Obtain the process value.



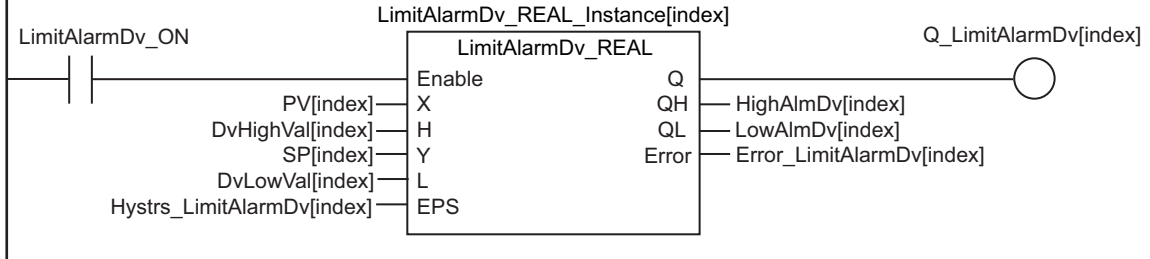
Upper/lower limit alarm



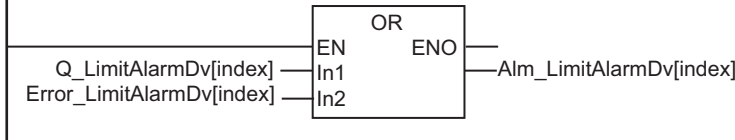
Perform an output as a safety measure if an error occurs in the LimitAlarm\_REAL instruction or if an upper/lower limit alarm occurs.



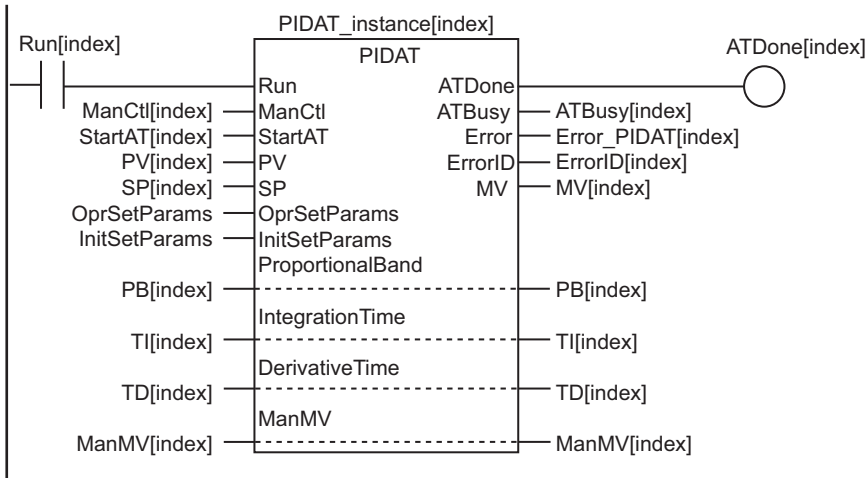
Upper/lower deviation alarm



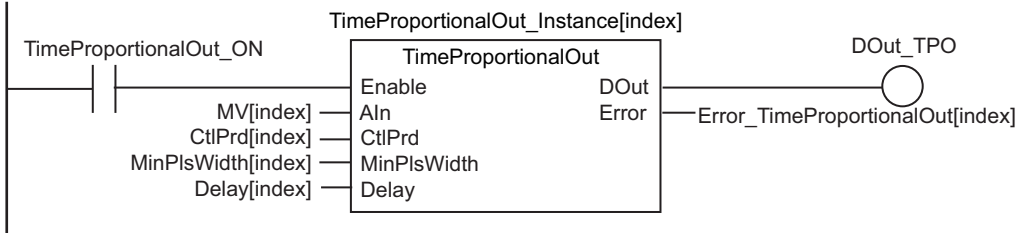
Perform an output as a safety measure if an error occurs in the LimitAlarmDv\_REAL instruction or if an upper/lower limit alarm occurs.



Execute PIDAT instruction.

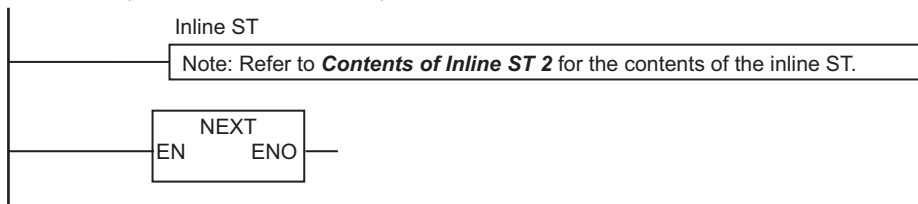


Time-proportional output





Perform outputs for bits 00 to 03 of output word 1.



### ● 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 proces
s 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:=DOOut_TPO;
INT#1:
  DO2:=DOOut_TPO;
INT#2:
  DO3:=DOOut_TPO;
ELSE
  DO4:=DOOut_TPO;
END_CASE;
```

## ST

| Internal Variables | Name          | Data type | Default | Re-tain                  | Comment  |
|--------------------|---------------|-----------|---------|--------------------------|--|
|                    | index         | UINT      | 0       | <input type="checkbox"/> | Loop index                                       |
|                    | LimitAlarm_ON | BOOL      | True    | <input type="checkbox"/> | Execution of Upper/Lower Limit Alarm instruction |

| Internal Variables | Name                    | Data type           | Default    | Retain                   | Comment   |
|--------------------|-------------------------|---------------------|------------|--------------------------|---|
|                    | LimitAlarmDv_ON         | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Deviation Alarm instruction              |
|                    | TimeProportional-Out_ON | BOOL                | True       | <input type="checkbox"/> | Execution of Timeproportional Output instruction                  |
|                    | AI                      | INT                 | 0          | <input type="checkbox"/> | Present value   |
|                    | PV                      | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Process value   |
|                    | SP                      | ARRAY[0..3] OF REAL | [4(100)]   | <input type="checkbox"/> | Set point   |
|                    | DOut_TPO                | BOOL                | False      | <input type="checkbox"/> | Time-proportional output  |
|                    | HighVal                 | ARRAY[0..3] OF REAL | [4(200)]   | <input type="checkbox"/> | Upper limit set value of upper/lower limit alarm                  |
|                    | LowVal                  | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Lower limit set value of upper/lower limit alarm                  |
|                    | Hystrs_LimitAlarm       | ARRAY[0..3] OF REAL | [4(5)]     | <input type="checkbox"/> | Hysteresis of upper/lower limit alarm                             |
|                    | Q_LimitAlarm            | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower limit alarm output                                    |
|                    | HighAlm                 | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper limit alarm   |
|                    | LowAlm                  | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower limit alarm   |
|                    | Error_LimitAlarm        | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Error in LimitAlarm_REAL instruction                              |
|                    | Alm_LimitAlarm          | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Output for safety measure for Upper/Lower Limit Alarm instruction |
|                    | DvHighVal               | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Upper deviation set value of upper/lower deviation alarm          |
|                    | DvLowVal                | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Lower deviation set value of upper/lower deviation alarm          |
|                    | Q_LimitAlarmDv          | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower deviation alarm output                                |
|                    | HighAlmDv               | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper deviation alarm   |
|                    | LowAlmDv                | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower deviation alarm   |
|                    | Error_LimitAlarmDv      | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Error in LimitAlarmDv_REAL instruction                            |
|                    | Hystrs_LimitAlarmDv     | ARRAY[0..3] OF REAL | [4(3)]     | <input type="checkbox"/> | Hysteresis of upper/lower deviation alarm                         |

| Internal Variables | Name                       | Data type                        | Default  | Retain                              | Comment   |
|--------------------|----------------------------|----------------------------------|--|-------------------------------------|---|
|                    | Alm_LimitAlarmDv           | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Output for safety measure for Upper/Lower Deviation Alarm instruction |
|                    | Run                        | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Execution condition   |
|                    | ManCtl                     | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Manual/auto control   |
|                    | StartAT                    | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | 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) | <input type="checkbox"/>            | Operation setting parameters  |
|                    | InitSetParams              | _sINIT_SET_PARAMS                | (SampTime:=T#100 ms, RngLowLmt:=-10.0,RngUpLmt:=1000.0, DirOpr:=False)   | <input type="checkbox"/>            | Initial setting parameters  |
|                    | PB                         | ARRAY[0..3] OF REAL              | [4(10)]  | <input checked="" type="checkbox"/> | Proportional band   |
|                    | TI                         | ARRAY[0..3] OF TIME              | [4(T#0 s)]   | <input checked="" type="checkbox"/> | Integration time  |
|                    | TD                         | ARRAY[0..3] OF TIME              | [4(T#0 s)]   | <input checked="" type="checkbox"/> | Derivative time   |
|                    | ManMV                      | ARRAY[0..3] OF REAL              | [4(0.0)]   | <input type="checkbox"/>            | Manual manipulated variable   |
|                    | ATDone                     | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Autotuning normal completion  |
|                    | ATBusy                     | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Autotuning busy   |
|                    | Error_PIDAT                | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Error in PIDAT instruction  |
|                    | ErrorID                    | ARRAY[0..3] OF WORD              | [4(16#0)]  | <input type="checkbox"/>            | Error ID for PIDAT instruction  |
|                    | MV                         | ARRAY[0..3] OF REAL              | [4(0.0)]   | <input type="checkbox"/>            | Manipulated variable  |
|                    | CtlPrd                     | ARRAY[0..3] OF TIME              | [4(T#1 s)]   | <input type="checkbox"/>            | Control period  |
|                    | MinPlsWidth                | ARRAY[0..3] OF REAL              | [4(0.0)]   | <input type="checkbox"/>            | Minimum pulse width   |
|                    | Delay                      | ARRAY[0..3] OF REAL              | [4(0.0)]   | <input type="checkbox"/>            | ON-delay time   |
|                    | Error_TimeProportionalOut  | ARRAY[0..3] OF BOOL              | [4(False)]   | <input type="checkbox"/>            | Error in TimeProportionalOut instruction                              |
|                    | LimitAlarm_REAL_instance   | ARRAY[0..3] OF LimitAlarm_REAL   |  | <input type="checkbox"/>            |   |
|                    | LimitAlarmDv_REAL_instance | ARRAY[0..3] OF LimitAlarmDv_REAL |  | <input type="checkbox"/>            |   |
|                    | PIDAT_instance             | ARRAY[0..3] OF PIDAT             |  | <input type="checkbox"/>            |   |

| Internal Variables | Name                          | Data type                           | Default | Retain                   | Comment |
|--------------------|-------------------------------|-------------------------------------|---------|--------------------------|---------|
|                    | TimeProportional-Out_instance | ARRAY[0..3] OF Time-ProportionalOut |         | <input type="checkbox"/> |         |

| External Variables | Name | Data type | Comment                    |
|--------------------|------|-----------|----------------------------|
|                    | AI1  | INT       | Input No.1 (Process value) |
|                    | AI2  | INT       | Input No.2 (Process value) |
|                    | AI3  | INT       | Input No.3 (Process value) |
|                    | AI4  | INT       | Input No.4 (Process value) |
|                    | DO1  | BOOL      | output word 1 (Bit 00)     |
|                    | DO2  | BOOL      | output word 1 (Bit 01)     |
|                    | DO3  | BOOL      | output word 1 (Bit 02)     |
|                    | DO4  | BOOL      | output word 1 (Bit 03)     |

```
// 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 :=Hysters_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_R
EAL 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 :=Hysters_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_R
EAL 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],

```

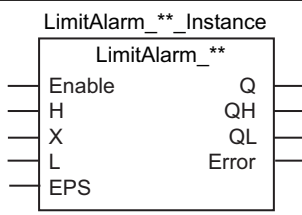
```
    Delay :=Delay[index],
    DOut =>DOut_TPO,
    Error =>Error_TimeProportionalOut[index]);

// 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/<br>FUN | Graphic expression   | ST expression   |
|---------------|-------------------------------------|------------|--|---|
| LimitAlarm_** | Upper/Lower<br>Limit Alarm<br>Group | FB         |  <p>**** must be REAL or LREAL.</p> | LimitAlarm_**_instance(Enable, H,<br>X, L, EPS, Q, QH, QL, Error);<br><br>**** must be REAL or LREAL. |



## 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.

## Variables

|        | Meaning               | I/O    | Description   | Valid range             | Unit | Default |
|--------|-----------------------|--------|---|-------------------------|------|---------|
| Enable | Enable                | Input  | TRUE: Execute<br>FALSE: Reset alarm   | Depends on data type.   | ---  | FALSE   |
| H      | Upper limit set value |        | Upper limit set value for the input value   |                         |      |         |
| 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.*1 |      |         |
| Q      | Alarm output          | Output | TRUE: There is either an upper limit alarm or a lower limit alarm.<br>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.<br>FALSE: There is no upper limit alarm.   |                         |      |         |
| QL     | Lower limit alarm     |        | TRUE: There is a lower limit alarm.<br>FALSE: There is no lower limit alarm.  |                         |      |         |

\*1. Negative numbers are excluded.

|        | Boo<br>lean                          | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>H</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| L      | Must be same data type as <i>H</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| EPS    | Must be same data type as <i>H</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| 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.

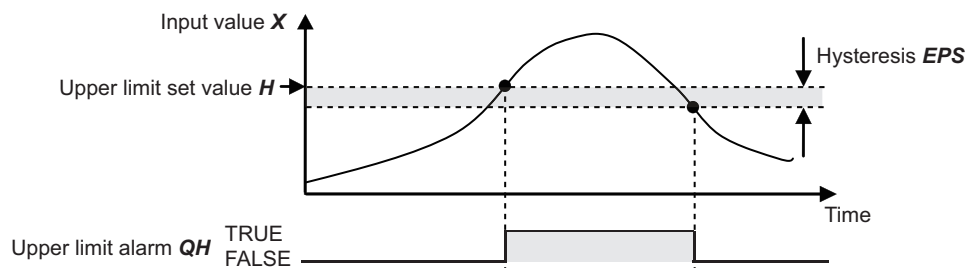
## Operation of Upper Limit Alarm *QH*

The value of *QH* (Upper limit alarm) changes as shown below.

You can set the hysteresis to prevent hunting in the limit alarm.

- If Input value  $X > \text{Upper limit set value } H$ , the value is TRUE.
- If Input value  $X < \text{Upper limit set value } H - \text{Hysteresis } EPS$ , the value is FALSE.



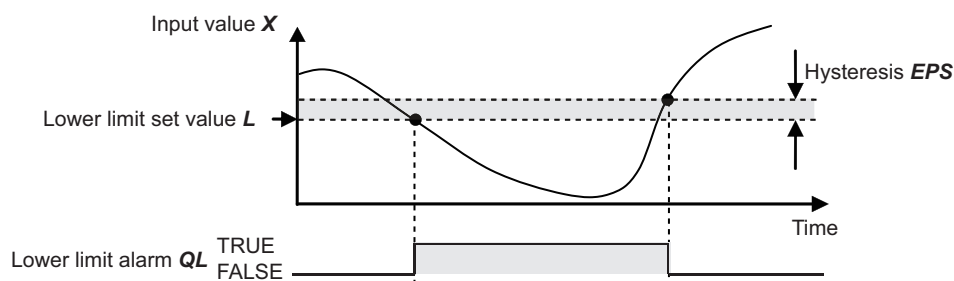


## Operation of Lower Limit Alarm QL

The value of QL (Lower limit alarm) changes as shown below.

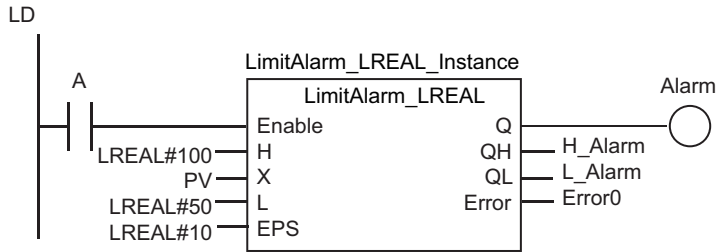
You can set the hysteresis to prevent hunting in the limit alarm.

- If Input value  $X < \text{Lower limit set value } L$ , the value is TRUE.
- If Input value  $X > \text{Lower limit set value } L + \text{Hysteresis } EPS$ , the value is FALSE.



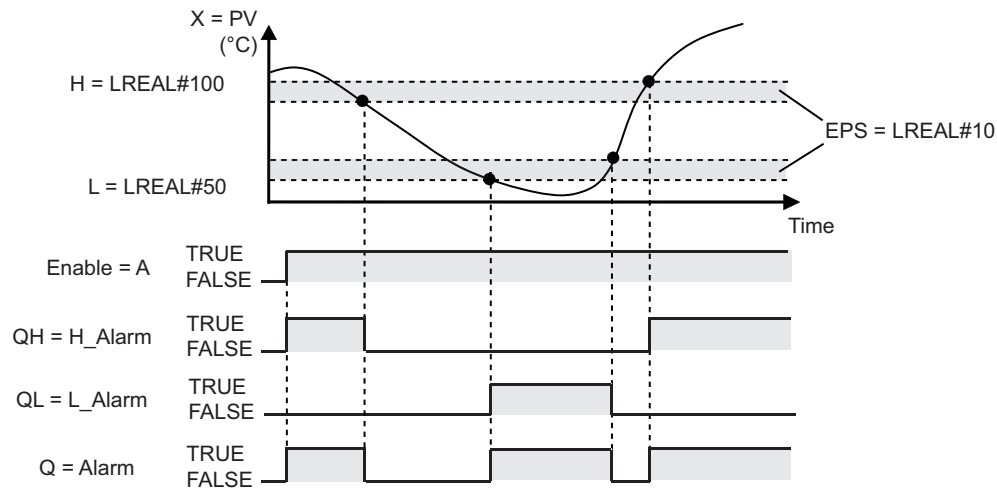
## Notation Example

The following notation example sets upper limit set value  $H$  to  $100^{\circ}\text{C}$ , lower limit set value  $L$  to  $50^{\circ}\text{C}$ , and hysteresis  $EPS$  to  $10^{\circ}\text{C}$ .



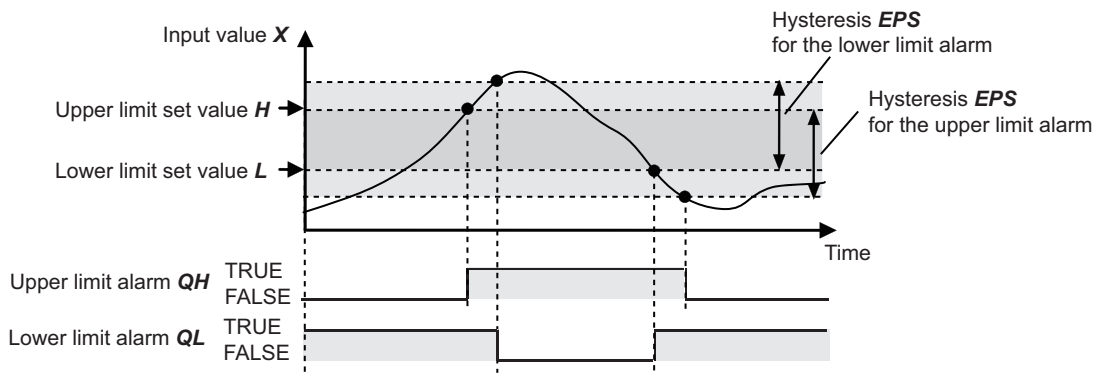
ST

```
LimitAlarm_REAL_instance(A,LREAL#100,PV,LREAL#50,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

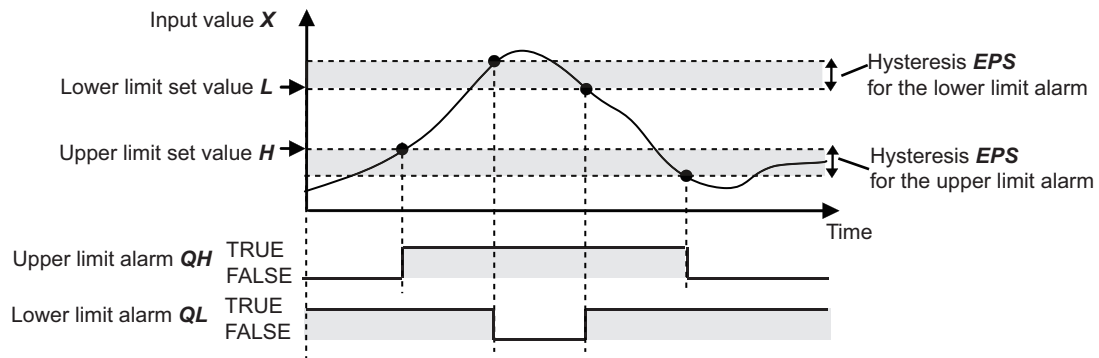


### Additional Information

- Use the LimitAlarm\_REAL instruction to reduce the instruction execution time.
- You can set as follows:  $H - L < EPS$ . If you do so, both *QH* and *QL* can be TRUE at the same time.



- You can set as follows:  $H < L$ . 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 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 can be maintained even when an error causes  $Q$ ,  $QH$ , and  $QL$  to change to FALSE. For an application example, refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.

## Sample Programming

Refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.

# 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/<br>FUN | Graphic expression   | ST expression  |
|-----------------|-----------------------------------|------------|--|--|
| LimitAlarmDv_** | Upper/Lower Deviation Alarm Group | FB         | <p>LimitAlarmDv_** Instance</p> <p>LimitAlarmDv_**</p> <p>Enable Q</p> <p>X QH</p> <p>H QL</p> <p>Y Error</p> <p>L</p> <p>EPS</p> <p>**** must be REAL or LREAL.</p> | LimitAlarmDv_**instance(Enable, X, H, Y, L, EPS, Q, QH, QL, Error);<br><br>**** must be REAL or LREAL. |



## 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.

## Variables

|        | Meaning                   | I/O   | Description   | Valid range             | Unit | Default |
|--------|---------------------------|-------|---|-------------------------|------|---------|
| Enable | Enable                    | Input | TRUE: Execute<br>FALSE: Reset alarm   | Depends on data type.   | ---  | FALSE   |
| X      | Input value               |       | Value to monitor  |                         |      |         |
| 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.*1 |      | 0       |

|    | Meaning                | I/O    | Description   | Valid range           | Unit | Default |
|----|------------------------|--------|---|-----------------------|------|---------|
| Q  | Deviation alarm output | Output | TRUE: There is either an upper deviation alarm or a lower deviation alarm.<br>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.<br>FALSE: There is no upper deviation alarm.   |                       |      |         |
| QL | Lower deviation alarm  |        | TRUE: There is a lower deviation alarm.<br>FALSE: There is no lower deviation alarm.  |                       |      |         |

\*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 |
| 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.

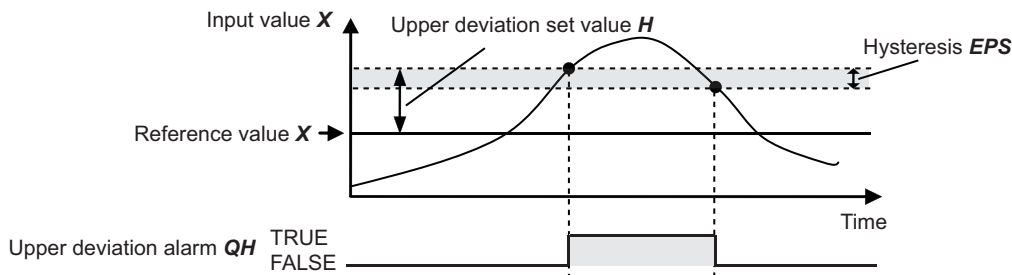
The name of the instruction is determined by the data types of *X*, *H*, *Y*, *L*, and *EPS*.

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.

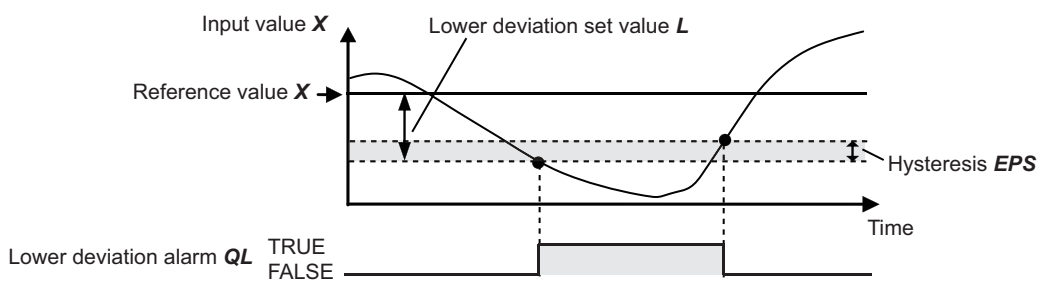
- If Input value *X* - Reference value *Y* > Upper deviation set value *H*, then the value is TRUE.
- If Input value *X* - Reference value *Y* < Upper deviation set value *H* - Hysteresis *EPS*, then the value is FALSE.



## Operation of Lower Deviation Alarm *QL*

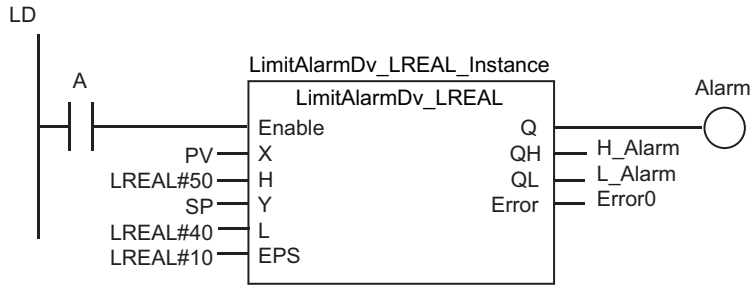
Lower deviation alarm *QL* is the alarm for a downward deviation in respect to reference value *Y*. The value of *QL* changes as shown below. You can set the hysteresis to prevent hunting in the deviation alarm.

- If  $-(\text{Input value } X - \text{Reference value } Y) > \text{Lower deviation set value } L$ , then the value is TRUE.
- If  $-(\text{Input value } X - \text{Reference value } Y) < \text{Lower deviation set value } L - \text{Hysteresis } EPS$ , then the value is FALSE.

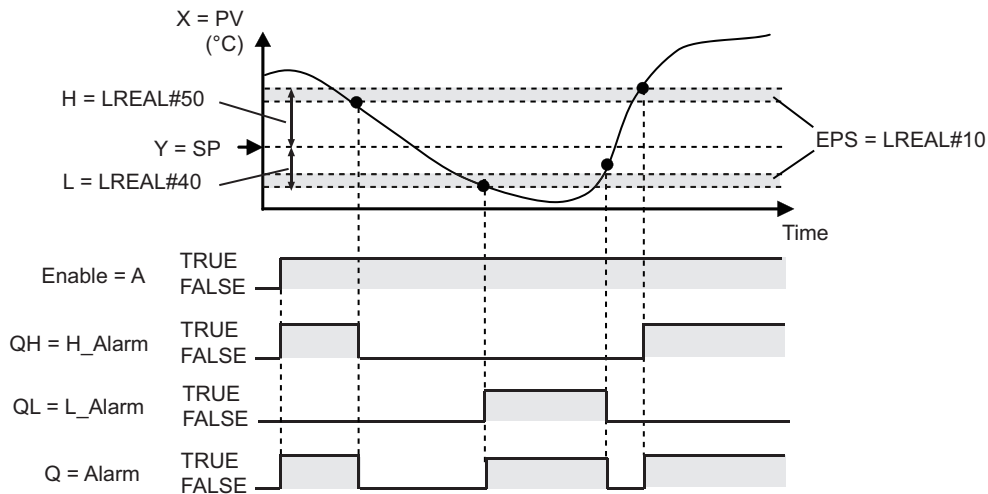


## Notation Example

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.

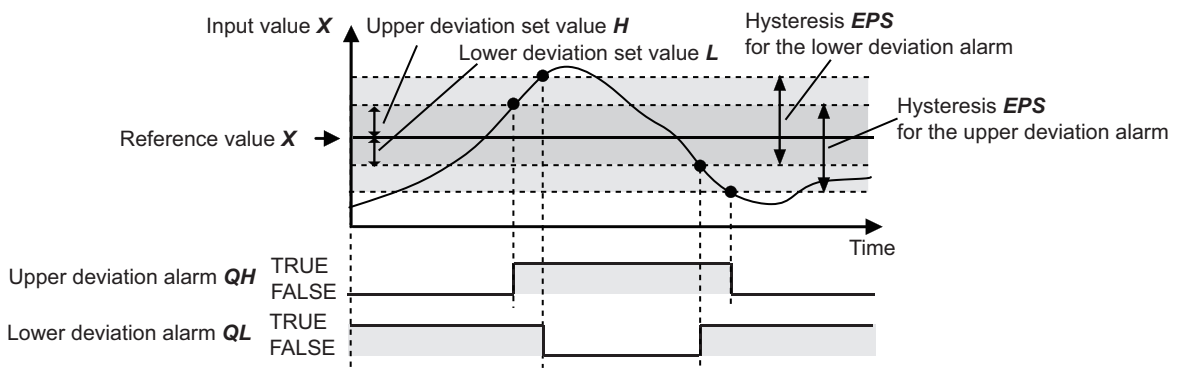


```
ST
LimitAlarmDv_REAL_instance(A,PV,LREAL#50,SP,LREAL#40,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

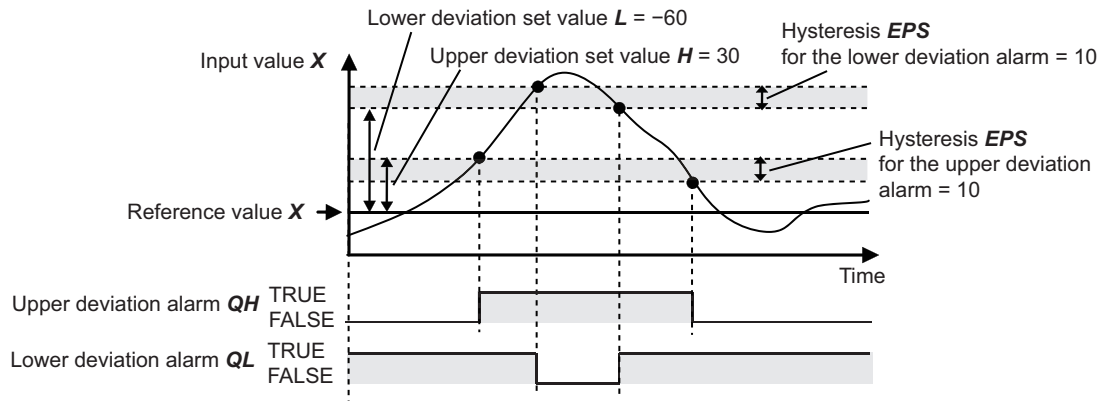


### Additional Information

- 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  $H + L$  to less than 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 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. For an application example, refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.

## Sample Programming

Refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.



# LimitAlarmDvStbySeq\_\*\*

The LimitAlarmDvStbySeq\_\*\* instruction outputs upper and lower deviation alarms with a standby sequence.

| Instruction            | Name  | FB/<br>FUN | Graphic expression   | ST expression   |
|------------------------|---|------------|--|---|
| LimitAlarmDvStbySeq_** | Upper/Lower Deviation Alarm with Standby Sequence Group | FB         | <p>LimitAlarmDvStbySeq_**_Instance</p> <p>LimitAlarmDvStbySeq_**</p> <p>Enable Q</p> <p>X QH</p> <p>H QL</p> <p>Y StbySeqFlag</p> <p>L Error</p> <p>EPS</p> <p>**** must be REAL or LREAL.</p> | LimitAlarmDvStbySeq_**_instance(Enable, X, H, Y, L, EPS, Q, QH, QL, StbySeqFlag, Error);<br><br>**** must be REAL or LREAL. |



## 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.

## Variables

|        | Meaning                   | I/O   | Description   | Valid range             | Unit | Default |
|--------|---------------------------|-------|---|-------------------------|------|---------|
| Enable | Enable                    | Input | TRUE: Execute<br>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  |                         |      |         |
| 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.*1 | 0    |         |

|              | Meaning                       | I/O    | Description   | Valid range           | Unit | Default |
|--------------|-------------------------------|--------|---|-----------------------|------|---------|
| Q            | Deviation alarm output        | Output | TRUE: There is either an upper deviation alarm or a lower deviation alarm.<br>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.<br>FALSE: There is no upper deviation alarm.   |                       |      |         |
| QL           | Lower deviation alarm         |        | TRUE: There is a lower deviation alarm.<br>FALSE: There is no lower deviation alarm.  |                       |      |         |
| StbySeq-Flag | Standby Sequence Enabled Flag |        | TRUE: Enabled<br>FALSE: Disabled  |                       |      |         |

\*1. Negative numbers are excluded.

|              | Boo lean | Bit strings                  |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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.

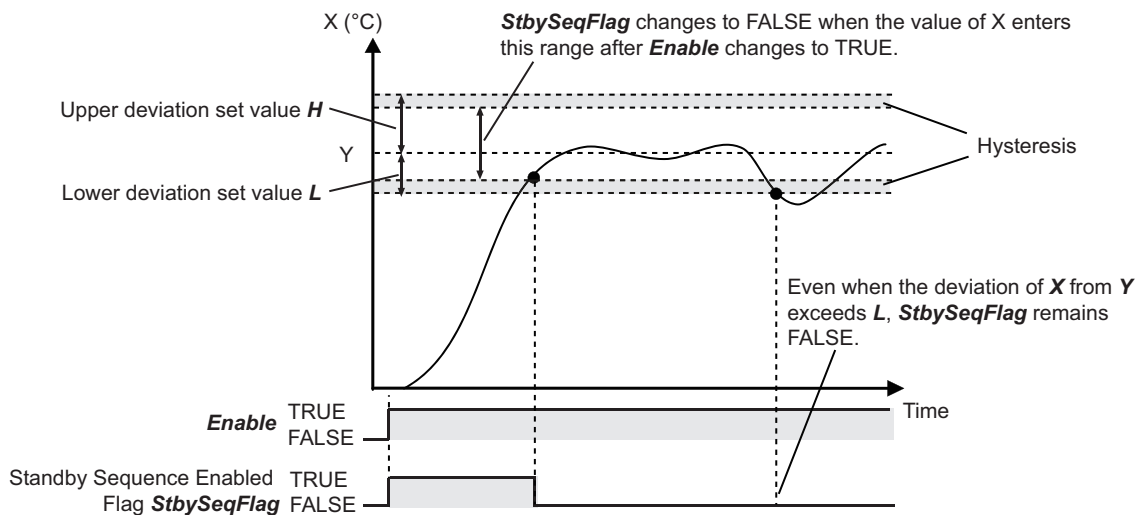
The values of  $X$ ,  $H$ ,  $Y$ ,  $L$ , and  $EPS$  are continuously updated while  $Enable$  is TRUE.

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 LimitAlarmDvStbySeq\_LREAL, the data types of  $X$ ,  $H$ ,  $Y$ ,  $L$ , and  $EPS$  are all LREAL.

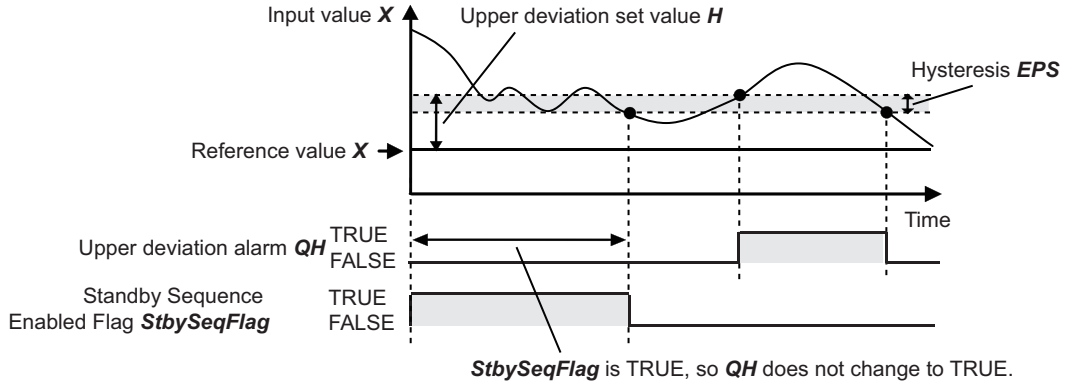
## Operation of Upper Deviation Alarm $QH$

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 the value is TRUE.
- If Input value  $X$  - Reference value  $Y$  < Upper deviation set value  $H$  - Hysteresis  $EPS$ , then the value is FALSE.

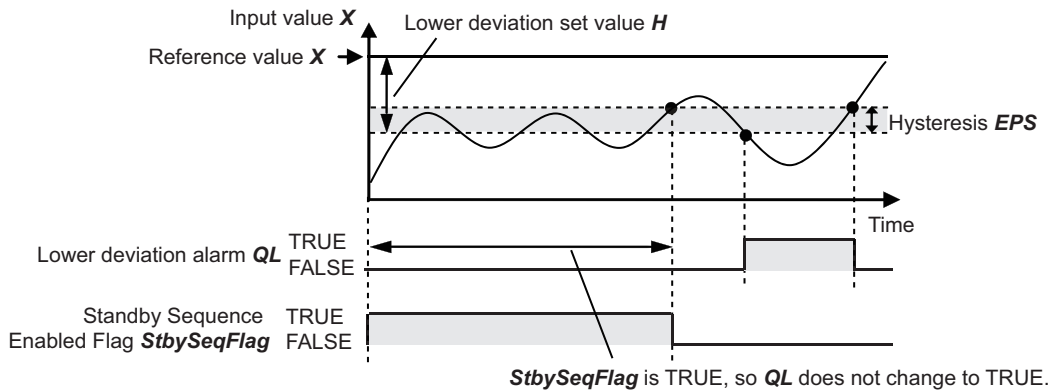


## Operation of Lower Deviation Alarm QL

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.

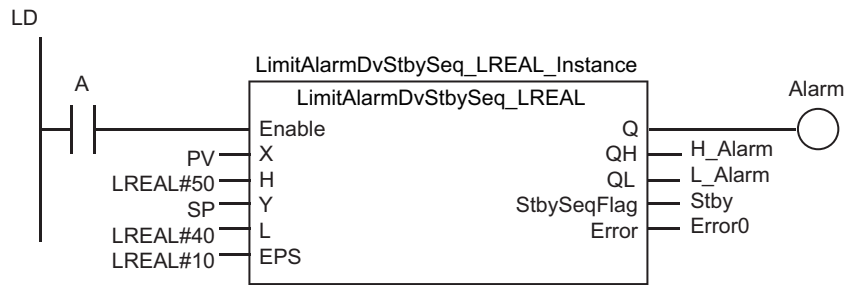
You can set the hysteresis to prevent hunting in the deviation alarm.

- If  $-(\text{Input value } X - \text{Reference value } Y) > \text{Lower deviation set value } L$ , then the value is TRUE.
- If  $-(\text{Input value } X - \text{Reference value } Y) < \text{Lower deviation set value } L - \text{Hysteresis } EPS$ , then the value is FALSE.

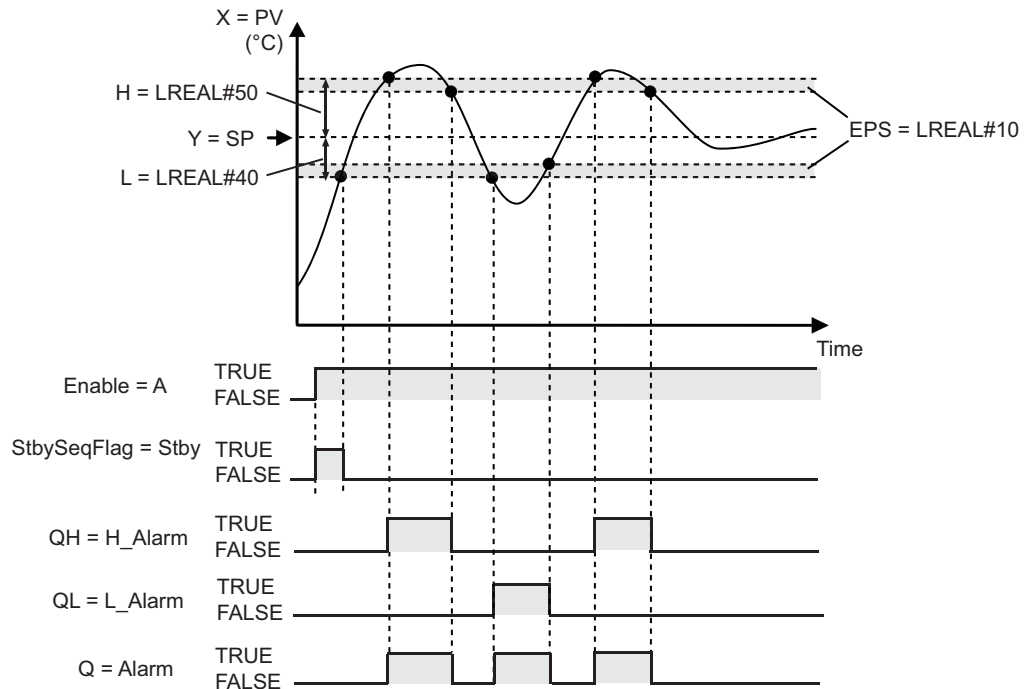


## Notation Example

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.



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  $EPS$  larger than  $H + L$ . If you do so, both  $QH$  and  $QL$  can be TRUE at the same time. Refer to the instruction, `LimitAlarmDv_**` on page 2-804.
- You can set as follows:  $H + L < 0$ . If you do so, either  $QH$  or  $QL$  will always be TRUE while `StbySeqFlag` is FALSE. Refer to the instruction, `LimitAlarmDv_**` on page 2-804.

## Precautions for Correct Use

- An error occurs if the value of  $EPS$  is outside 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* on page 2-814 for an application example.

## 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_AllnPV | Process value for input 1 (INT data) | AI1      |
|            | Ch2_AllnPV | Process value for input 2 (INT data) | AI2      |
|            | Ch3_AllnPV | Process value for input 3 (INT data) | AI3      |
|            | Ch4_AllnPV | 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 |
|-------|--------|
| A11   | DO1    |
| A12   | DO2    |
| A13   | DO3    |
| A14   | DO4    |

The task period of the task to which the program is assigned is 1 ms.

### ● Configuration Diagram

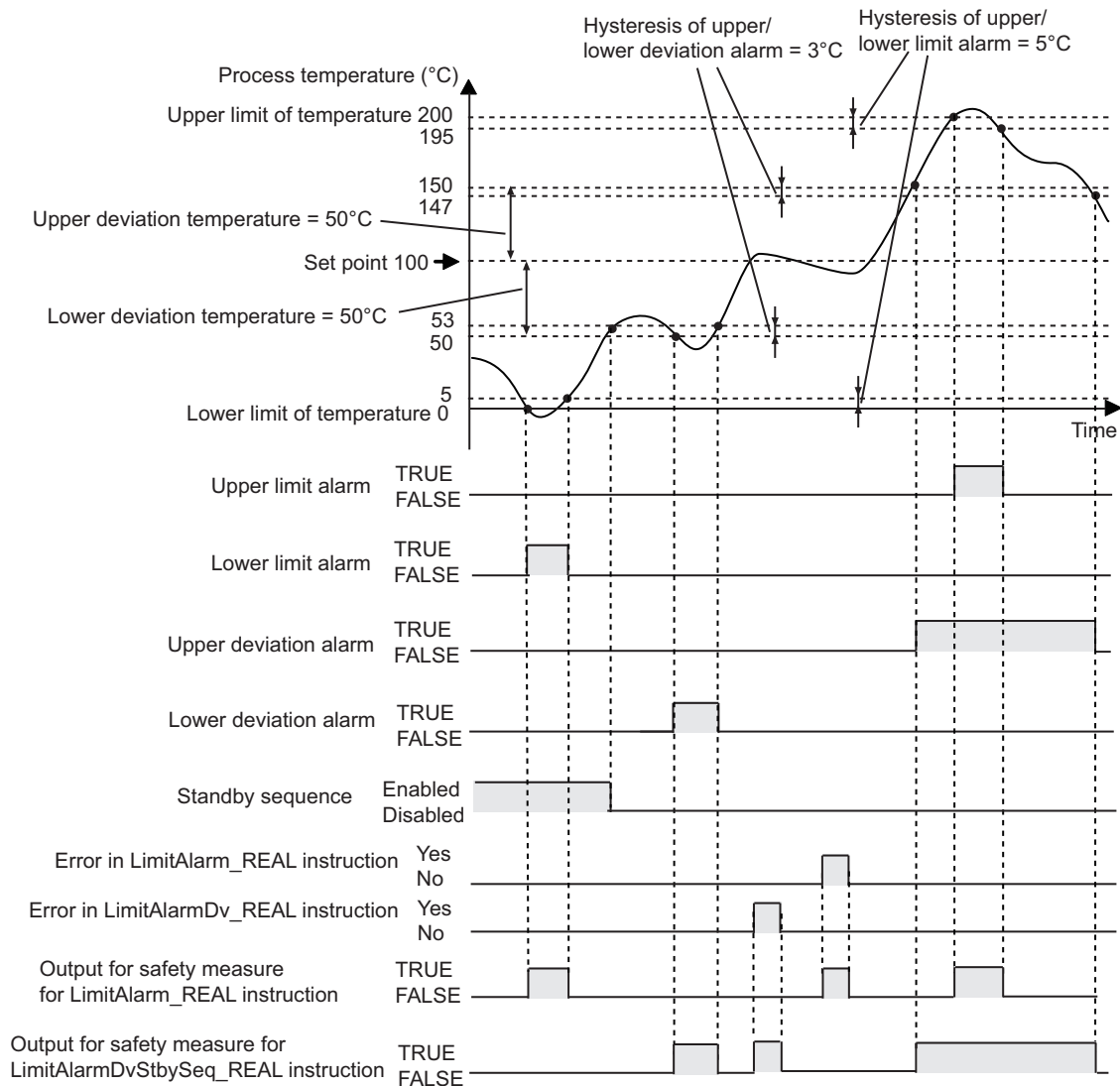
Refer to *Sample Programming* on page 2-785 for the TimeProportionalOut instruction.

## 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



## Definitions of Global Variables

### ● Global Variables

| Variable | Data type | AT specification*1                      | Comment                              |
|----------|-----------|---|--------------------------------------|
| AI1      | INT       | IOBus://rack#0/slot#0/Ch1_AllnPV        | Process value for input 1 (INT data) |
| AI2      | INT       | IOBus://rack#0/slot#0/Ch2_AllnPV        | Process value for input 2 (INT data) |
| AI3      | INT       | IOBus://rack#0/slot#0/Ch3_AllnPV        | Process value for input 3 (INT data) |
| AI4      | INT       | IOBus://rack#0/slot#0/Ch4_AllnPV        | 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. This table shows the variables for the CJ1W-PH41U Input Unit mounted to Slot #0 of Rack #0, and the CJ1W-OD212 Output Unit mounted to Slot #1 of the same rack.



**Note** The global variables for the port of each Unit are automatically generated based on the I/O mapping settings.

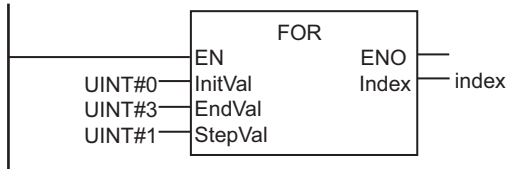
## LD

| Name                   | Data type           | Default    | Retain                   | Comment  |
|------------------------|---------------------|------------|--------------------------|--|
| index                  | UINT                | 0          | <input type="checkbox"/> | Loop index   |
| LimitAlarm_ON          | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Limit Alarm instruction                           |
| LimitAlarmDvStbySeq_ON | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Deviation Alarm with Standby Sequence instruction |
| TimeProportionalOut_ON | BOOL                | True       | <input type="checkbox"/> | Execution of Time-ProportionalOut instruction                              |
| AI                     | INT                 | 0          | <input type="checkbox"/> | Present value  |
| PV                     | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Process value  |
| SP                     | ARRAY[0..3] OF REAL | [4(100)]   | <input type="checkbox"/> | Set point  |
| DOut_TPO               | BOOL                | False      | <input type="checkbox"/> | Time-proportional output   |
| HighVal                | ARRAY[0..3] OF REAL | [4(200)]   | <input type="checkbox"/> | Upper limit set value of upper/lower limit alarm                           |
| LowVal                 | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Lower limit set value of upper/lower limit alarm                           |
| Hysters_LimitAlarm     | ARRAY[0..3] OF REAL | [4(5)]     | <input type="checkbox"/> | Hysteresis of upper/lower limit alarm                                      |
| Q_LimitAlarm           | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower limit alarm output   |
| HighAlm                | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper limit alarm  |
| LowAlm                 | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower limit alarm  |
| Error_LimitAlarm       | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Error in LimitAlarm_REAL instruction                                       |
| Alm_LimitAlarm         | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Output for safety measure for Upper/Lower Limit Alarm instruction          |
| DvHighVal              | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Upper deviation set value of upper/lower deviation alarm                   |

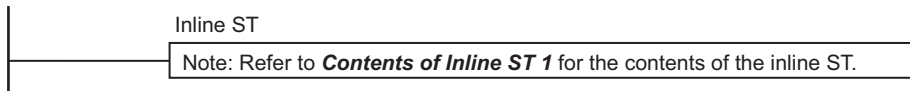
| Name                      | Data type           | Default   | Retain                              | Comment   |
|---------------------------|---------------------|---|-------------------------------------|---|
| DvLowVal                  | ARRAY[0..3] OF REAL | [4(50)]   | <input type="checkbox"/>            | Lower deviation set value of upper/lower deviation alarm              |
| Q_LimitAlarmDv            | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Upper/lower deviation alarm output                                    |
| HighAlmDv                 | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Upper deviation alarm   |
| LowAlmDv                  | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Lower deviation alarm   |
| StbySeqFlag               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Standby Sequence Enabled Flag   |
| Error_LimitAlarmDvStbySeq | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Error in LimitAlarmDvStbySeq_REAL instruction                         |
| Hysters_LimitAlarmDv      | ARRAY[0..3] OF REAL | [4(3)]  | <input type="checkbox"/>            | Hysteresis of upper/lower deviation alarm                             |
| Alm_LimitAlarmDv          | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Output for safety measure for Upper/Lower Deviation Alarm instruction |
| Run                       | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Execution condition   |
| ManCtl                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Manual/auto control   |
| StartAT                   | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | 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, ATHysters:=0.2) | <input type="checkbox"/>            | Operation setting parameters  |
| InitSetParams             | _sINIT_SET_PARAMS   | (SampTime:=T#100 ms, RngLowLmt:=-10.0, RngUpLmt:=1000.0, DirOpr:=False)   | <input type="checkbox"/>            | Initial setting parameters  |
| PB                        | ARRAY[0..3] OF REAL | [4(10)]   | <input checked="" type="checkbox"/> | Proportional band   |
| TI                        | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Integration time  |
| TD                        | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Derivative time   |
| ManMV                     | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Manual manipulated variable   |
| ATDone                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning normal completion  |
| ATBusy                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning busy   |

| Name                              | Data type                | Default    | Retention                | Comment                                  |
|-----------------------------------|--------------------------|------------|--------------------------|--|
| Error_PIDAT                       | ARRAY[0..3] OF BOOL      | [4(False)] | <input type="checkbox"/> | Error in PIDAT instruction               |
| ErrorID                           | ARRAY[0..3] OF WORD      | [4(16#0)]  | <input type="checkbox"/> | Error ID for PIDAT instruction           |
| MV                                | ARRAY[0..3] OF REAL      | [4(0.0)]   | <input type="checkbox"/> | Manipulated variable                     |
| CtlPrd                            | ARRAY[0..3] OF TIME      | [4(T#1 s)] | <input type="checkbox"/> | Control period                           |
| MinPlsWidth                       | ARRAY[0..3] OF REAL      | [4(0.0)]   | <input type="checkbox"/> | Minimum pulse width                      |
| Delay                             | ARRAY[0..3] OF REAL      | [4(0.0)]   | <input type="checkbox"/> | ON-delay time                            |
| Error_TimeProportionalOut         | ARRAY[0..3] OF BOOL      | [4(False)] | <input type="checkbox"/> | Error in TimeProportionalOut instruction |
| LimitAlarm_REAL_instance          | LimitAlarm_REAL          |            | <input type="checkbox"/> |  |
| LimitAlarmDvStbySeq_REAL_instance | LimitAlarmDvStbySeq_REAL |            | <input type="checkbox"/> |  |
| PIDAT_instance                    | PIDAT                    |            | <input type="checkbox"/> |  |
| TimeProportionalOut_instance      | TimeProportionalOut      |            | <input type="checkbox"/> |  |

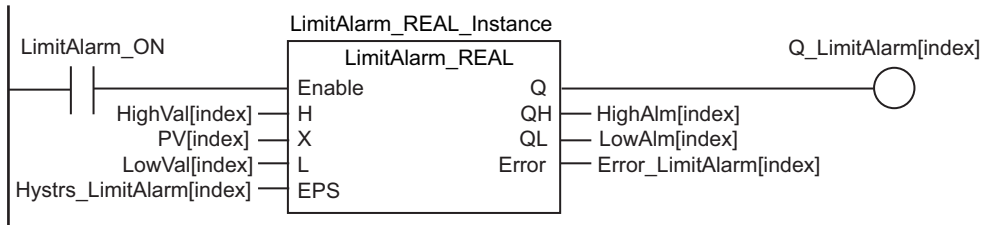
Control temperature for four points.



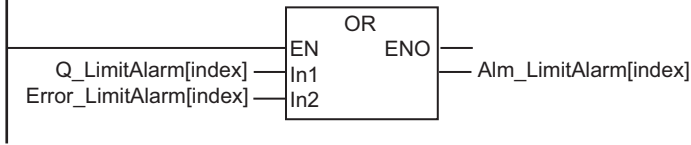
Obtain the process value.



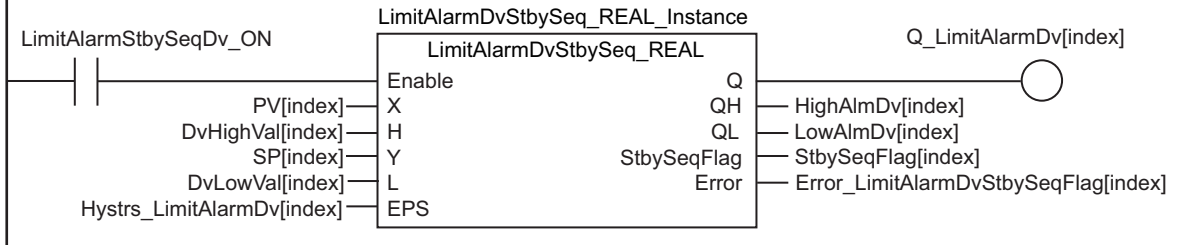
Upper/lower limit alarm



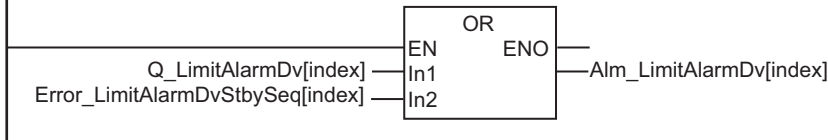
Perform an output as a safety measure if an error occurs in the LimitAlarm\_REAL instruction or if an upper/lower limit alarm occurs.



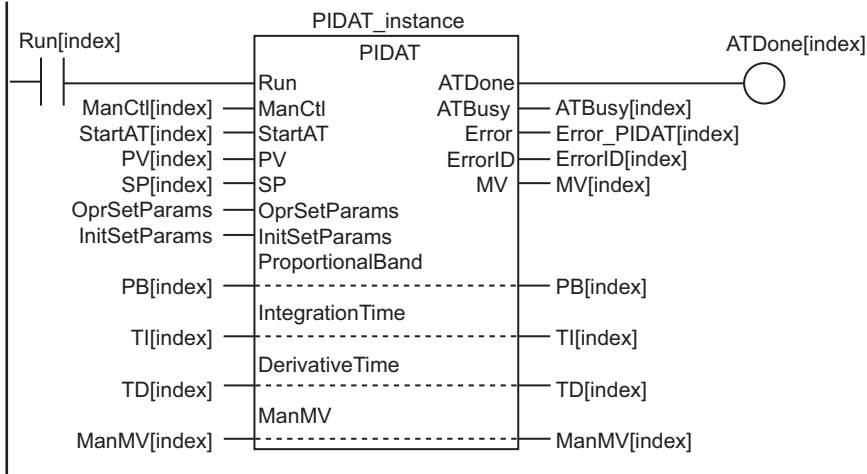
Upper/lower deviation alarm with standby sequence



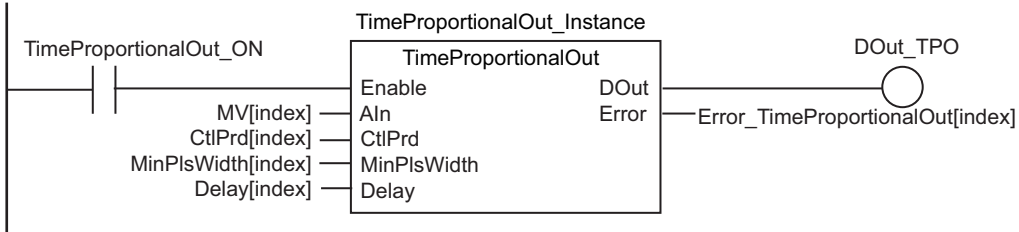
Perform an output as a safety measure if an error occurs in the LimitAlarmDvStbySeq\_REAL instruction or if an upper/lower limit alarm occurs.



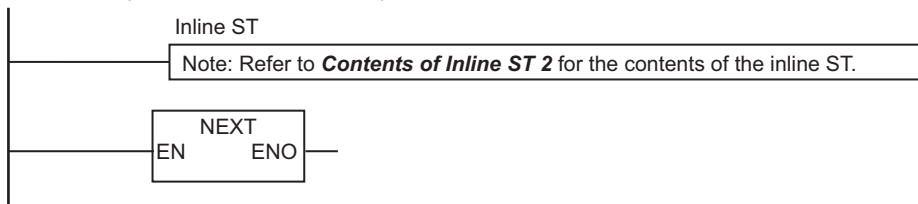
Execute PIDAT instruction.



Time-proportional output



Perform outputs for bits 00 to 03 of output word 1.



### ● 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 proces
s 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       | <input type="checkbox"/> | Loop index                                       |
| LimitAlarm_ON | BOOL      | True    | <input type="checkbox"/> | Execution of Upper/Lower Limit Alarm instruction |

| Name                   | Data type           | Default    | Retain                   | Comment  |
|------------------------|---------------------|------------|--------------------------|--|
| LimitAlarmDvStbySeq_ON | BOOL                | True       | <input type="checkbox"/> | Execution of Upper/Lower Deviation Alarm with Standby Sequence instruction |
| TimeProportionalOut_ON | BOOL                | True       | <input type="checkbox"/> | Execution of Time-proportional Output instruction                          |
| AI                     | INT                 | 0          | <input type="checkbox"/> | Present value  |
| PV                     | ARRAY[0..3] OF REAL | 0.0        | <input type="checkbox"/> | Process value  |
| SP                     | ARRAY[0..3] OF REAL | [4(100)]   | <input type="checkbox"/> | Set point  |
| DOut_TPO               | BOOL                | False      | <input type="checkbox"/> | Time-proportional output   |
| HighVal                | ARRAY[0..3] OF REAL | [4(200)]   | <input type="checkbox"/> | Upper limit set value of upper/lower limit alarm                           |
| LowVal                 | ARRAY[0..3] OF REAL | [4(0.0)]   | <input type="checkbox"/> | Lower limit set value of upper/lower limit alarm                           |
| Hysters_LimitAlarm     | ARRAY[0..3] OF REAL | [4(5)]     | <input type="checkbox"/> | Hysteresis of upper/lower limit alarm                                      |
| Q_LimitAlarm           | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower limit alarm output   |
| HighAlm                | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper limit alarm  |
| LowAlm                 | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower limit alarm  |
| Error_LimitAlarm       | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Error in LimitAlarm_REAL instruction                                       |
| Alm_LimitAlarm         | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Output for safety measure for Upper/Lower Limit Alarm instruction          |
| DvHighVal              | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Upper deviation set value of upper/lower deviation alarm                   |
| DvLowVal               | ARRAY[0..3] OF REAL | [4(50)]    | <input type="checkbox"/> | Lower deviation set value of upper/lower deviation alarm                   |
| Q_LimitAlarmDv         | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper/lower deviation alarm output   |
| HighAlmDv              | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Upper deviation alarm  |
| LowAlmDv               | ARRAY[0..3] OF BOOL | [4(False)] | <input type="checkbox"/> | Lower deviation alarm  |

| Name                      | Data type           | Default   | Retain                              | Comment   |
|---------------------------|---------------------|---|-------------------------------------|---|
| StbySeqFlag               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Standby Sequence Enabled Flag   |
| Error_LimitAlarmDvStbySeq | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Error in LimitAlarmDvStbySeq_REAL instruction                         |
| Hysters_LimitAlarmDv      | ARRAY[0..3] OF REAL | [4(3)]  | <input type="checkbox"/>            | Hysteresis of upper/lower deviation alarm                             |
| Alm_LimitAlarmDv          | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Output for safety measure for Upper/Lower Deviation Alarm instruction |
| Run                       | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Execution condition   |
| ManCtl                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Manual/auto control   |
| StartAT                   | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | 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, ATHysters:=0.2) | <input type="checkbox"/>            | Operation setting parameters  |
| InitSetParams             | _sINIT_SET_PARAMS   | (SampTime:=T#100 ms, RngLowLmt:=-10.0, RngUpLmt:=1000.0, DirOpr:=False)   | <input type="checkbox"/>            | Initial setting parameters  |
| PB                        | ARRAY[0..3] OF REAL | [4(10)]   | <input checked="" type="checkbox"/> | Proportional band   |
| TI                        | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Integration time  |
| TD                        | ARRAY[0..3] OF TIME | [4(T#0 s)]  | <input checked="" type="checkbox"/> | Derivative time   |
| ManMV                     | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Manual manipulated variable   |
| ATDone                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning normal completion  |
| ATBusy                    | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Autotuning busy   |
| Error_PIDAT               | ARRAY[0..3] OF BOOL | [4(False)]  | <input type="checkbox"/>            | Error in PIDAT instruction  |
| ErrorID                   | ARRAY[0..3] OF WORD | [4(16#0)]   | <input type="checkbox"/>            | Error ID for PIDAT instruction  |
| MV                        | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Manipulated variable  |
| CtlPrd                    | ARRAY[0..3] OF TIME | [4(T#1 s)]  | <input type="checkbox"/>            | Control period  |
| MinPlsWidth               | ARRAY[0..3] OF REAL | [4(0.0)]  | <input type="checkbox"/>            | Minimum pulse width   |

| Name                              | Data type                | Default    | Retain                   | Comment                                  |
|-----------------------------------|--------------------------|------------|--------------------------|--|
| Delay                             | ARRAY[0..3] OF REAL      | [4(0.0)]   | <input type="checkbox"/> | ON-delay time                            |
| Error_TimeProportionalOut         | ARRAY[0..3] OF BOOL      | [4(False)] | <input type="checkbox"/> | Error in TimeProportionalOut instruction |
| LimitAlarm_REAL_instance          | LimitAlarm_REAL          |            | <input type="checkbox"/> |  |
| LimitAlarmDvStbySeq_REAL_instance | LimitAlarmDvStbySeq_REAL |            | <input type="checkbox"/> |  |
| PIDAT_instance                    | PIDAT                    |            | <input type="checkbox"/> |  |
| TimeProportionalOut_instance      | TimeProportionalOut      |            | <input type="checkbox"/> |  |

```
// 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
```



```

L 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/<br>FUN | Graphic expression | ST expression                                    |
|-------------|----------------------|------------|--------------------|--|
| ScaleTrans  | Scale Transformation | FUN        |                    | Out :=ScaleTrans(SclIn, X0, Y0, X1, Y1, SclOfs); |



## 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.

## Variables

|        | Meaning                  | I/O    | Description                      | Valid range           | Unit | Default |
|--------|--------------------------|--------|----------------------------------|-----------------------|------|---------|
| SclIn  | Input value              | Input  | Value to scale                   | Depends on data type. | ---  | *1      |
| X0     | Input range lower limit  |        | Lower limit of input range       |                       |      |         |
| 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 | ---                   | ---  |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|        | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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

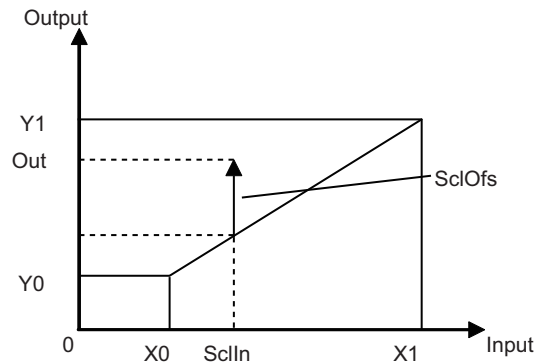
The Scale Trans instruction scales the value of input value *ScIn* from an input range to an output range.

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 *ScOfs* is added to the value that was scaled to the output range, and the result is output as output value *Out*. *ScOfs* is used, for example, to correct error in temperature control.

The following conversion is used.

$$\text{Out} = \frac{Y1-Y0}{X1-X0} (\text{ScIn} - X0) + Y0 + \text{ScOfs}$$

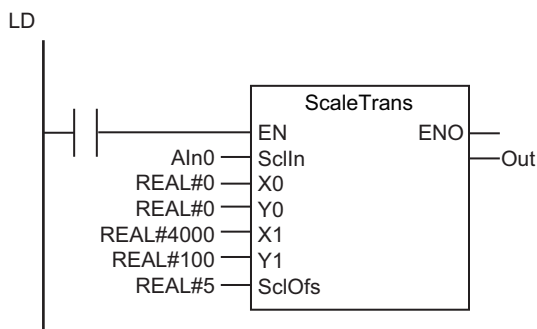


## Notation Example

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.

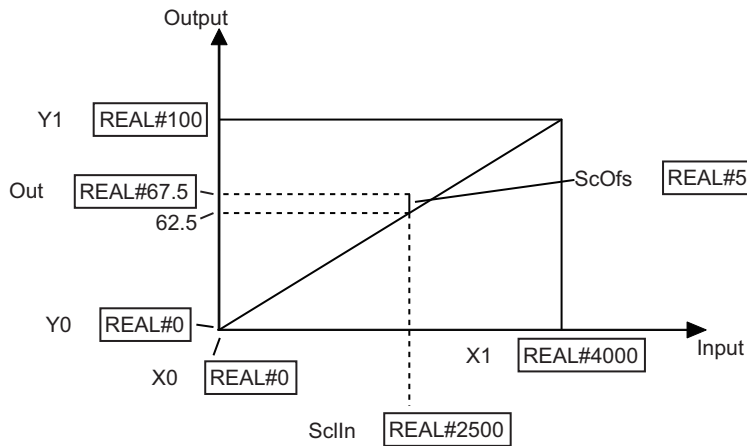
The following values are used: *ScIn* = REAL#2500, *X0* = REAL#0, *X1* = REAL#4000, *Y0* = REAL#0, *Y1* = REAL#100, and *ScOfs* = REAL#5.

The value of *Out* will be REAL#67.5.



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 *ScIn* to the range of *PV* or *SP* of the PIDAT instruction, pass the following parameters to *Y0* and *Y1*.

| Variable | Parameter   |
|----------|---|
| Y0       | <i>InitSetParams.RngLowLmt</i> (input range lower limit of the PIDAT instruction) |
| Y1       | <i>InitSetParams.RngUpLmt</i> (input range upper limit of the PIDAT instruction)  |

- Settings are also possible with  $X1 < X0$  and  $Y1 < Y0$ .

# 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/<br>FUN | Graphic expression | ST expression  |
|----------------|--------------|------------|--------------------|--|
| AC_StepProgram | Step Program | FB         |                    | <pre>AC_StepProgram_instance(   Enable, Hold, Advance, PV, IntegrationTime,   Alpha, Option, ProgramPattern, Done, Busy,   Error, ErrorID, Wait, StepNo, PresentSP,   PredictSP, TimelInfo);</pre> |



## 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.

## Variables

|                         | Meaning                  | I/O    | Description   | Valid range           | Unit | Default |  |     |       |
|-------------------------|--------------------------|--------|---|-----------------------|------|---------|--|-----|-------|
| Enable                  | Enable                   | Input  | TRUE: Execute<br>FALSE: Stop  | Depends on data type. | ---  | FALSE   |  |     |       |
| Hold                    | Hold                     |        | TRUE: Hold<br>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) <sup>*1</sup>  |                       |      |         | 0  |     |       |
| Integration-Time        | Integration time         |        | Integration time <sup>*2</sup>  |                       |      |         | T#0.0000 s to T#10000.0000 s <sup>*3</sup> | s   | T#0 s |
| Alpha                   | 2-PID Parameter $\alpha$ |        | 2-PID parameter $\alpha$ <sup>*4</sup>  |                       |      |         | 0.00 to 1.00                               | --- | 0     |
| Option                  | Option                   |        | Option <sup>*5</sup>  |                       |      | ---     | ---  | --- |       |
| Program-Pattern[] array | Program pattern          | In-out | Program pattern   | ---                   | ---  | ---     |  |     |       |

|           | Meaning             | I/O    | Description  | Valid range           | Unit | Default |
|-----------|---------------------|--------|--|-----------------------|------|---------|
| Wait      | Waiting             | Output | TRUE: Waiting<br>FALSE: Not waiting                          | Depends on data type. | ---  | ---     |
| StepNo    | Present step number |        | The number of the current step                               | 0 to 255*6            |      |         |
| 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 2-835 for details.
- \*2. It is the same as *IntegrationTime* in the PIDAT instruction. Refer to *IntegrationTime (Integration Time)* on page 2-836 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  $\alpha$ )* on page 2-836 for details.
- \*5. Refer to *Structure Specifications* on page 2-833 for details.
- \*6. The valid range is 0 to 99 for NX701, NX1P2, and NJ-series CPU Units with unit version 1.20 or earlier, and NX102 CPU Units with unit version 1.31 or earlier.

|                              | Boolean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Structure Specifications</i> on page 2-833 for details on the structure <code>_sAC_STEP_OPTION</code> .                 |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| ProgramPattern[] array*1*2*3 | Refer to <i>Structure Specifications</i> on page 2-833 for details on the structure <code>_sAC_STEP_DATA</code> . Specify an array. |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Wait                         | OK  |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| StepNo                       |   |             |      |       |       | OK    |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| PresentSP                    |   |             |      |       |       |       |          |       |       |      |     |      |      | OK           |       |   |      |     |    |        |
| PredictSP                    |   |             |      |       |       |       |          |       |       |      |     |      |      | OK           |       |   |      |     |    |        |
| TimeInfo                     | Refer to <i>Structure Specifications</i> on page 2-833 for details on the structure <code>_sAC_STEP_TIME</code> .                   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

- \*1. The maximum number of elements in an array depends on the unit version of the CPU Unit or Controller, as below.
  - The maximum number of elements in an array is 256 for NX102 CPU Units with unit version 1.32 or later, and NX701, NX1P2 and NJ-series CPU Units with unit version 1.21 or later.
  - It is 100 for unit versions other than those above.
- \*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

The AC\_StepProgram instruction calculates *PresentSP* (present set point) and *PredictSP* (predicted set point) every task period in order to perform manipulated variable control for a temperature controller in association with the PIDAT instruction.

The present set point is the set point in the present task period.

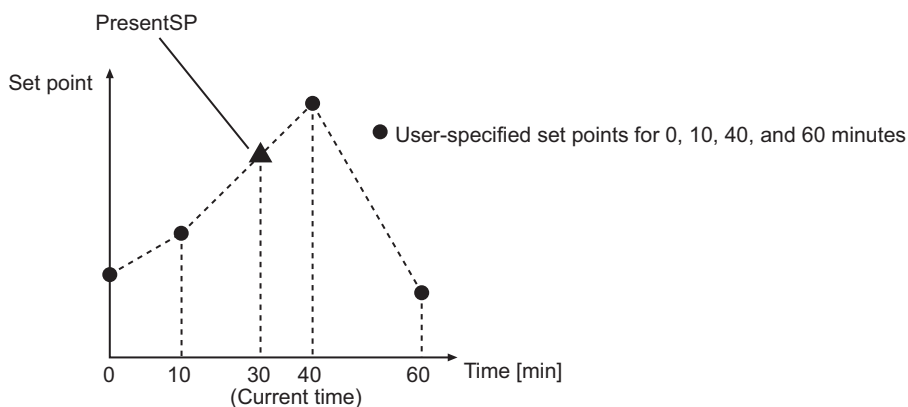
The predicted set point is arrived at by applying delay compensation for 2-PID control to the present set point.

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*.

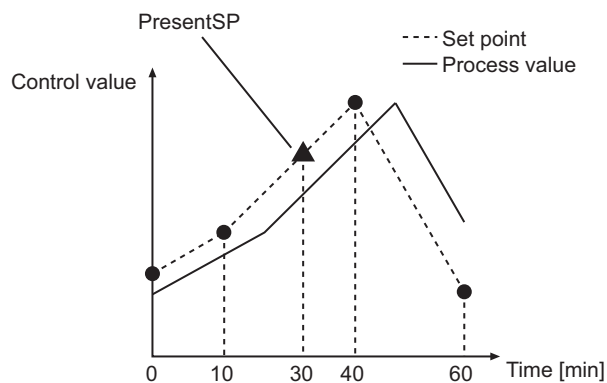


### ***PredictSP* (Predicted Set Point)**

Predicted set point *PredictSP* is a set point obtained by applying 2-PID control delay compensation to present set point *PresentSP*.

If *PresentSP* is passed to *SP* for the PIDAT instruction without the delay compensation, *PV* for the PIDAT instruction will not match the set point. This is illustrated in the following figure.

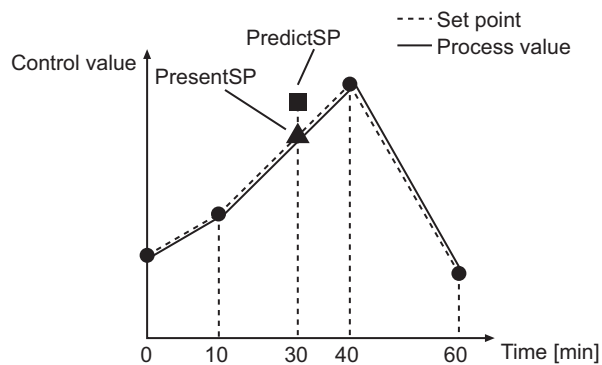




This delay can be corrected using *PredictSP*.

The *AC\_StepProgram* instruction calculates *PredictSP* based on integration time *IntegrationTime* and 2-PID parameter  $\alpha$  *Alpha*.

By passing *PredictSP* to *SP* for 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<br>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 255 *1           | ---  | 0                          |
| EndStepNo   | End step number   | The step number from which to end processing*2               | USINT                |                       |      |                            |
| Reserved    | Reserved.         | Reserved.  | ARRAY[0..31] OF BYTE | Depends on data type. |      | All 32 elements contain 0. |

\*1. The valid range is 0 to 99 for NX701, NX1P2, and NJ-series CPU Units with unit version 1.20 or earlier, and NX102 CPU Units with unit version 1.31 or earlier.

\*2. When 0 is set, the largest element number in ProgramPattern[] is regarded 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 |
|----------------|-----------------------|---|----------------|-----------------------|------|---------|
| ProgramPattern | 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#0 s   |
| WaitWidth      | Wait width            | The wait width of the step*2                      | REAL           |                       | ---  | 0       |
| WaitTimeLimit  | Wait time upper limit | The upper limit of the wait width of the step*1*3 | TIME           |                       | s    | T#0 s   |

\*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#0 s.

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<br>_TIME | ---                | ---  | ---     |
| Program-Time     | Program time        | The total of <i>TimeWidth</i> from step 0 to <i>EndStepNo</i> .                    | TIME               | Non-negative value | s    | T#0 s   |
| 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.

Refer to *Holding* on page 2-841 for details on holding.

### ● Advance (Advance)

If the value changes to TRUE during instruction execution, processing moves to the next step.

Refer to *Advancing* on page 2-843 for details on advancing.

### ● PV (Process Value)

This variable gives the process value of the controlled system. It is the same as *PV* for the PIDAT instruction.

**● IntegrationTime (Integration Time)**

This variable is the same as *IntegrationTime* for the PIDAT instruction.

Input the value or variable of *IntegrationTime* for the PIDAT instruction or the PIDAT\_HeatCool instruction.

**● Alpha (2-PID Parameter  $\alpha$ )**

This variable is the same as *OprSetParams.Alpha* for the PIDAT instruction.

Input the value or variable of *OprSetParams.Alpha* for 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.

Refer to *Start at PV* on page 2-841 for details on starting at the process value.

**● 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.

When 0 is set for *EndStepNo*, the largest element number in *ProgramPattern[]* is regarded as the end step number.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

**● ReachSP (Target Set Point)**

This variable gives the set point that should be reached at the end of the step in the program pattern.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

**● TimeWidth (Time Width)**

This variable gives the time width for the step in the program pattern.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

**● WaitWidth (Wait Width)**

This variable gives the threshold for performing waiting in the step in the program pattern.

Refer to *Waiting* on page 2-839 for details on waiting.

**● 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.

Refer to *Waiting* on page 2-839 for details on waiting.

**● Wait (Waiting)**

This variable is a flag that indicates if waiting is in progress.

If *Wait* is TRUE, waiting is in progress.

Refer to *Waiting* on page 2-839 for details on waiting.

- **StepNo (Present Step Number)**

This variable gives the number of the current step.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

- **PresentSP (Present Set Point)**

This variable gives the calculated present set point.

Refer to *PresentSP (Present Set Point)* on page 2-832 for details.

- **PredictSP (Predicted Set Point)**

This variable gives the calculated predicted set point.

Refer to *PredictSP (Predicted Set Point)* on page 2-832 for details.

- **ProgramTime (Program Time)**

This variable gives the total of *TimeWidth* from step 0 to *EndStepNo* in the program pattern.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

- **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.

Refer to *Waiting* on page 2-839 for details on waiting, and refer to *Holding* on page 2-841 for details on holding.

- **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.

Refer to *Waiting* on page 2-839 for details on waiting, and refer to *Holding* on page 2-841 for details on holding.

- **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.

Refer to *Waiting* on page 2-839 for details on waiting, and refer to *Holding* on page 2-841 for details on holding.

- **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.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

Refer to *Waiting* on page 2-839 for details on waiting, and refer to *Holding* on page 2-841 for details on holding.

- **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.

Refer to *Program Pattern* on page 2-838 for details on program patterns and steps.

Refer to *Waiting* on page 2-839 for details on waiting, and refer to *Holding* on page 2-841 for details on holding.

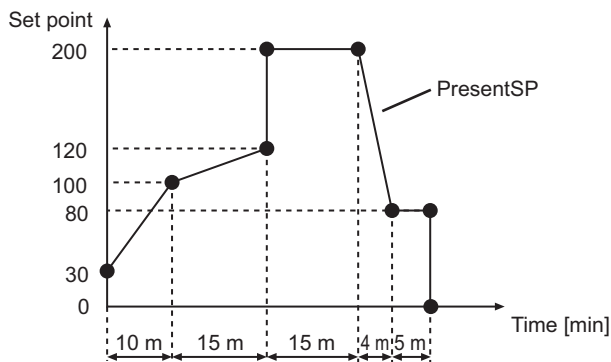
## Program Pattern

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 the elapsed time since instruction execution and the set points 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 <i>ReachSP</i>   | 30                              | 100    | 120    | 200   | 200    | 80    | 80    | 0     |
| Value of <i>TimeWidth</i> | T#0 s                           | T#10 m | T#15 m | T#0 s | T#15 m | T#4 m | T#5 m | T#0 s |



Linear interpolation is performed for the set point for each step 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 <i>TimeWidth</i> | Step number | Time width of the step                                       |
|---------------------------|-------------|--|
| T#0 s                     | 0           | Treated as T#0 s.  |
|                           | Not 0       | Treated as one task period.                                  |
| Positive                  | ---         | The value of <i>TimeWidth</i> 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#0 s                  | The value of <i>ReachSP</i> for step 0 is the initial value for <i>PresentSP</i> . Actual processing starts from step 1. |
|             | Not T#0 s              | 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 for 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. If the values are changed, the operation will be as follows:

| Variable           | New step number   | Operation   |
|--------------------|---|---|
| <i>StartStepNo</i> | ---   | Processing will start from the beginning of the step specified by the new <i>StartStepNo</i> .          |
| <i>EndStepNo</i>   | 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 <i>EndStepNo</i> 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.<br>The value of <i>Done</i> changes to TRUE. |

## Waiting

Due to delays in the controlled system, the value of *PV* may not reach the value of *ReachSP* within the time width specified in *TimeWidth* for the current step.

Waiting can be applied to continue the current step beyond the time width specified in *TimeWidth*.

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 the point of time and the process 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 the process 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 continues 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*.

During execution of waiting, the value of *Wait* is TRUE.

If the waiting is completed, the value of *Wait* changes to FALSE.

### ● Timing during Waiting

The operations of the time-related variables during waiting are described in the following table.

| Name             | Operation  |
|------------------|--|
| ElapsedTime      | 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 <i>TimeWidth</i> for the current step and then retains that value.                                      |
| StepLeftTime     | Goes to 0 and then retains that value.   |

### ● PresentSP and PredictSP during Waiting

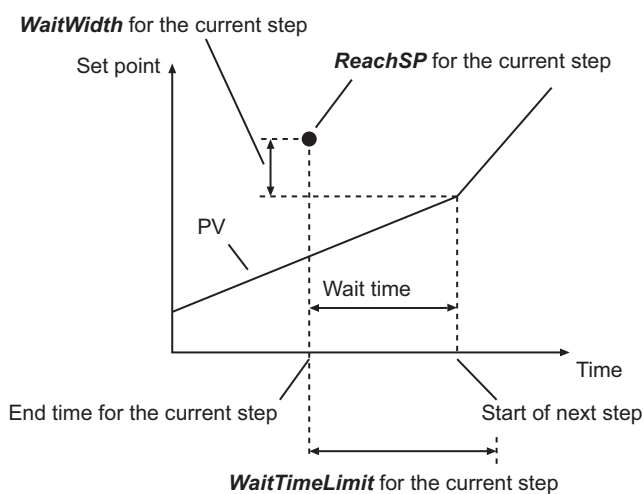
During waiting, both *PresentSP* and *PredictSP* retain the value of *ReachSP*.

### ● Example of Waiting

The following shows a graph of *PV* where 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* exceeds *WaitWidth* after the end time for the current step, so waiting occurs.

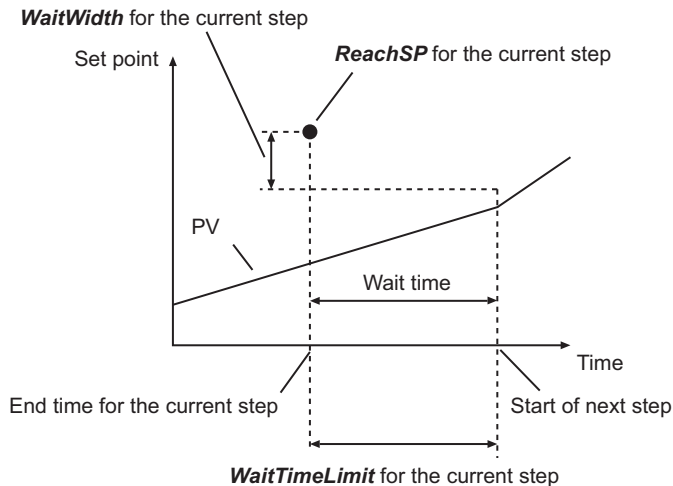
When the difference between *ReachSP* and *PV* becomes less than or equal to *WaitWidth*, the process moves to the next step.



The following shows a graph of *PV* where the difference between *ReachSP* and *PV* does not become equal to or less than *WaitWidth* within the time set for *WaitTimeLimit*.

The process moves to the next step after the time set for *WaitTimeLimit* 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  |
|------------------|--|
| ElapsedTime      | Stops timing and retains the value from when holding started.  |
| ProgressTime     | Starts timing again from the retained value when holding ends. |
| 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 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 until just before the value of *PresentSP* starts to increase.

**3** Processing is started from the point that was found in the above search.

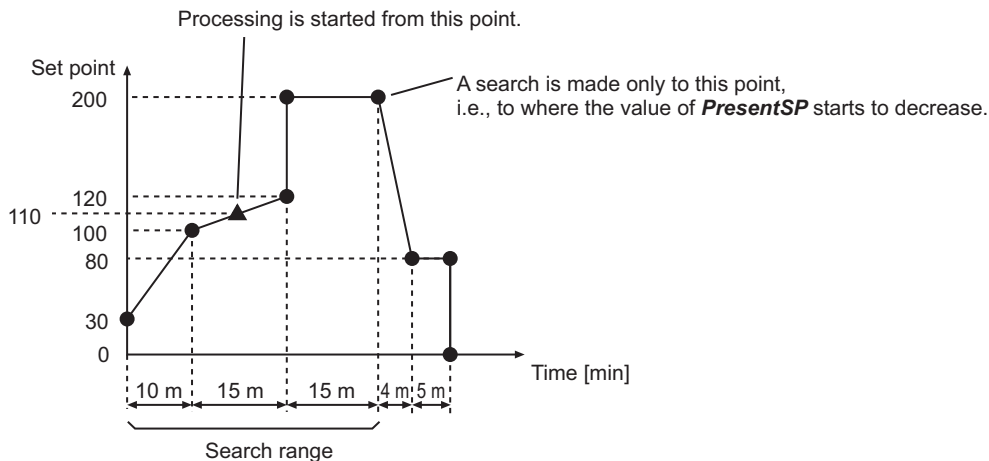
If there is no time in the search range where *PV* and *PresentSP* have the same value, processing is started from step 0.

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 <i>ReachSP</i>   | 30                              | 100    | 120    | 200   | 200    | 80    | 80    | 0     |
| Value of <i>TimeWidth</i> | T#0 s                           | T#10 m | T#15 m | T#0 s | T#15 m | T#4 m | T#5 m | T#0 s |

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   |
|--------------|---|
| ElapsedTime  | 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 <i>EndStepNo</i> .      |

| Name             | Operation  |
|------------------|--|
| 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, the process 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 <i>TimeWidth</i> from step 0 until the current step. |
| LeftTime         | Gives the time from the next step to the end of <i>EndStepNo</i> .      |
| StepProgressTime | Contains 0 because processing moves to the start of the next step.      |
| StepLeftTime     | Gives the value of <i>TimeWidth</i> 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.

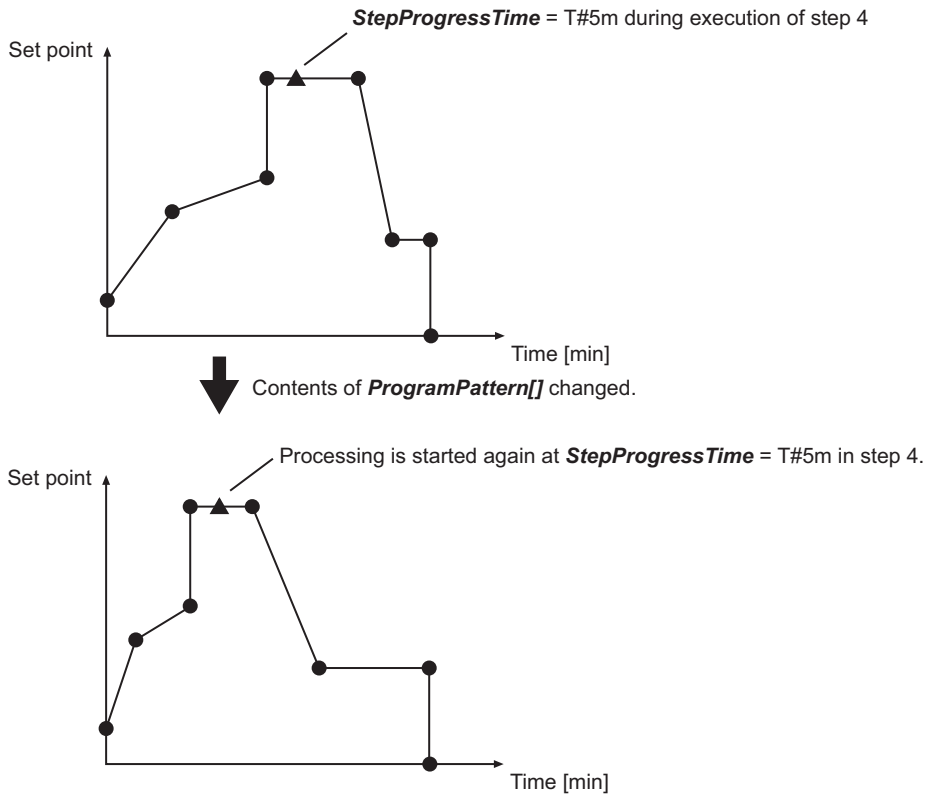
If you change the contents of *ProgramPattern[]*, the *PresentSP* is calculated again.

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#5 m. After you change the program pattern, processing will start again at a value of T#5 m 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 <i>TimeWidth</i> from step 0 to <i>EndStepNo</i> after the change.  |
| ElapseTime       | Continues timing.  |
| ProgressTime     | Gives the total of <i>StepProgressTime</i> and the total of <i>TimeWidth</i> from step 0 to one step before the current step after the change. |
| LeftTime         | Gives the time from the present to the end of <i>EndStepNo</i> 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*.

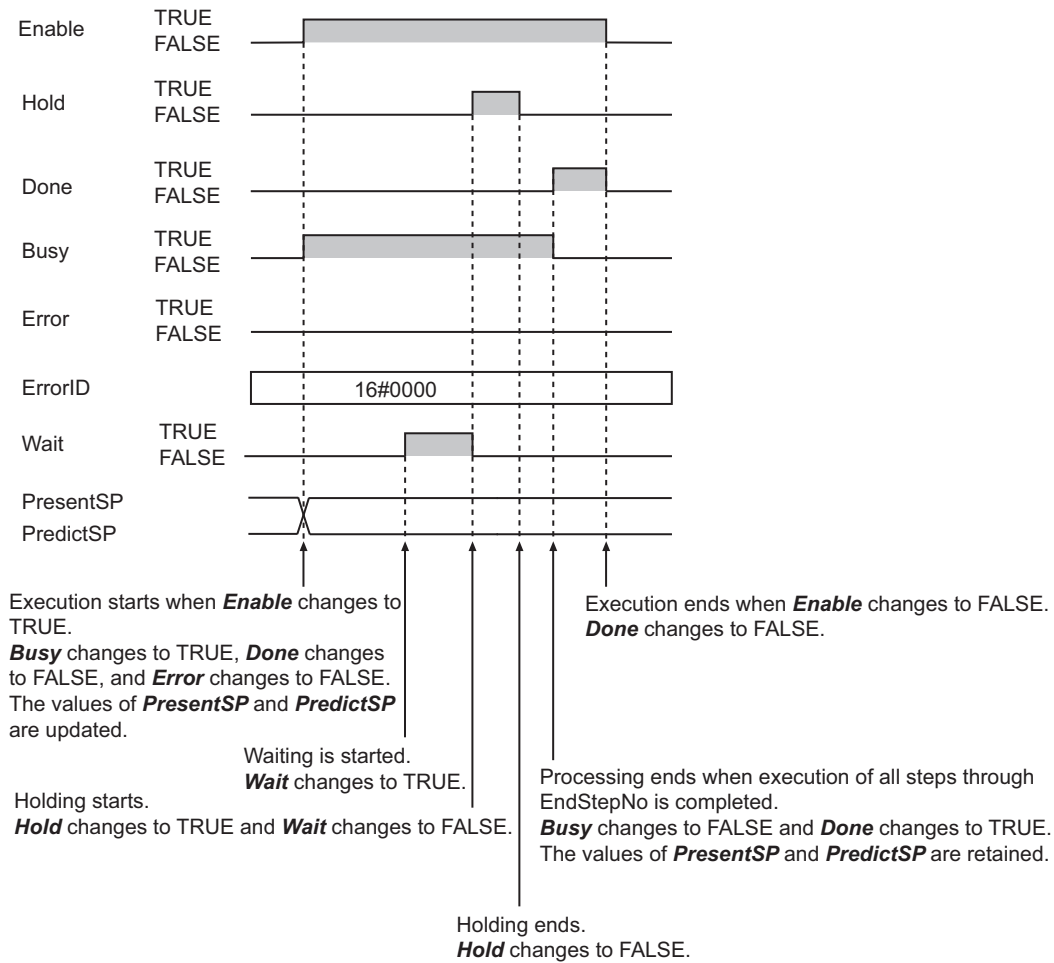
However, if the value of *StepProgressTime* is larger than the value of *WaitTimeLimit* after the change, waiting is ended immediately and processing moves to the next step.

● **Changing the Program Pattern during Holding**

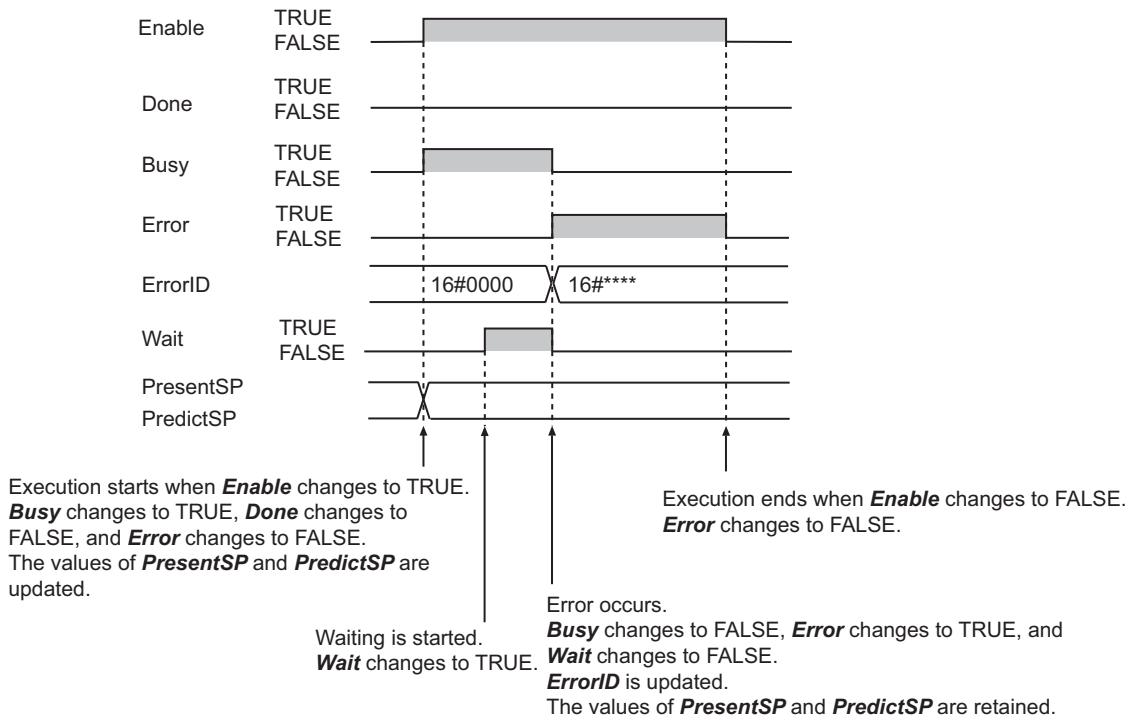
If you change the program pattern during holding, holding continues for the recalculated *PresentSP*.

## 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 will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.

| Error   | Value of <i>ErrorID</i> |
|---|-------------------------|
| The value of <i>IntegrationTime</i> , <i>Alpha</i> , <i>StartStepNo</i> , or <i>EndStepNo</i> is outside the valid range. | 16#0400                 |
| The final element number in the <i>ProgramPattern[]</i> array exceeds 255* <sup>1</sup> .                                 | 16#0416                 |

\*1. The final element number is 99 for NX701, NX1P2, and NJ-series CPU Units with unit version 1.20 or earlier, and NX102 CPU Units with unit version 1.31 or earlier.

## 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

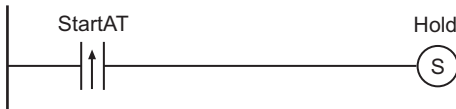
| Internal Variables | Variable       | Data type                     | Initial value   | Comment                                 |
|--------------------|----------------|-------------------------------|---|---|
|                    | ACSP_Enable    | BOOL                          | FALSE   | Enable for AC_StepProgram               |
|                    | Hold           | BOOL                          | FALSE   | Hold                                    |
|                    | Advance        | BOOL                          | FALSE   | Advance                                 |
|                    | Option         | _sAC_STEP_OPTION              | (StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Re-served:=[32(16#0)])   | Option                                  |
|                    | ProgramPattern | ARRAY[0..7] OF _sAC_STEP_DATA | <p>[(ReachSP:=30.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=100.0, Time-Width:=T#10 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=120.0, Time-Width:=T#15 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=150.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=150.0, Time-Width:=T#15 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=80.0, Time-Width:=T#4 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=80.0, Time-Width:=T#5 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),</p> <p>(ReachSP:=10.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m)]</p> | Program pattern                         |
|                    | ACSP_Busy      | BOOL                          | FALSE   | Execution of AC_StepProgram in progress |
|                    | ACSP_Error     | BOOL                          | FALSE   | AC_StepProgram error                    |

| Internal Variables | Variable      | Data type                | Initial value  | Comment                                  |
|--------------------|---------------|--------------------------|--|--|
|                    | 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                      |
|                    | TimeInfo      | _sAC_STEP_TIME           | (ProgramTime:=T#0 s, ElapseTime:=T#0 s, ProgressTime:=T#0 s, LeftTime:=T#0 s, StepProgressTime:=T#0 s, StepLeftTime:=T#0 s)                                  | 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#250 ms, 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  | TimeProportional-Out error               |
|                    | PIDbank       | ARRAY[0..7] OF sPID_BANK | [8((PB:=10, TI:=T#233 s, TD:=T#60 s))]   | Storage array for optimum PID parameters |
|                    | ACSP          | AC_StepProgram           |  |  |
|                    | PID           | PIDAT                    |  |  |

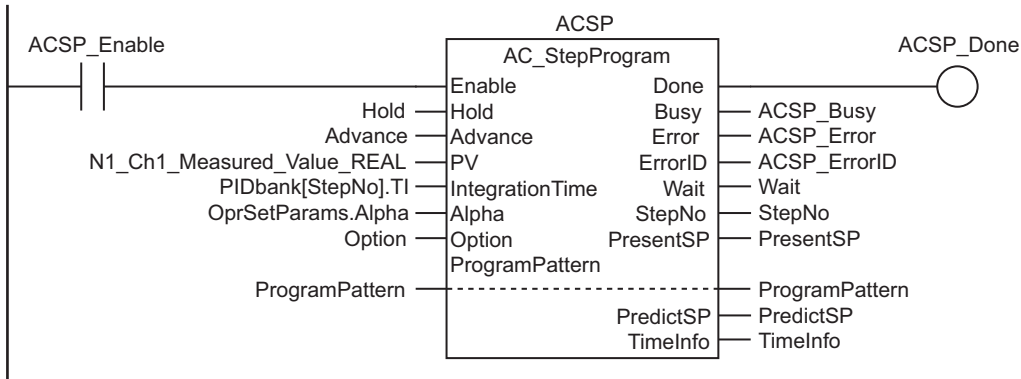
| Internal Variables | Variable | Data type            | Initial value | Comment |
|--------------------|----------|----------------------|---------------|---------|
|                    | TPO      | TimeProportional-Out |               |         |

| External Variables | Variable                   | Data type | Constant | Comment                       |
|--------------------|----------------------------|-----------|----------|-------------------------------|
|                    | N1_Ch1_Measured_Value_REAL | REAL      | ☐        | Channel measured value (REAL) |
|                    | N2_Output_Bit_00           | BOOL      | ☐        | Output bit                    |

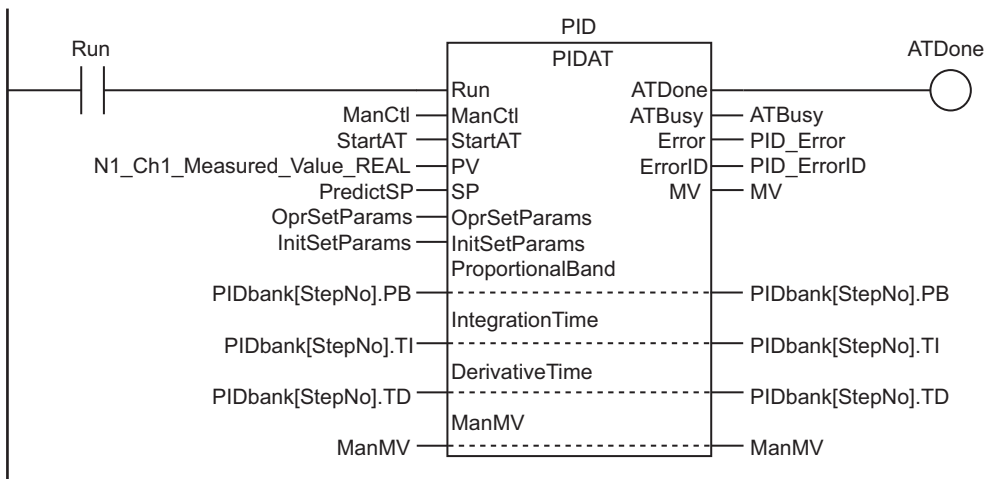
Perform holding for AC\_StepProgram instruction during autotuning.



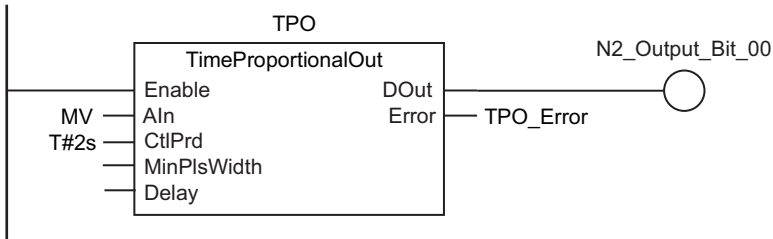
Execute AC\_StepProgram instruction.



Execute PIDAT instruction.



Execute TimeProportionalOut instruction.



## 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                    |

| Internal Variables | Variable       | Data type                        | Initial value  | Comment                                 |
|--------------------|----------------|----------------------------------|--|---|
|                    | ProgramPattern | ARRAY[0..7] OF<br>_sAC_STEP_DATA | [(ReachSP:=30.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=100.0, Time-Width:=T#10 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=120.0, Time-Width:=T#15 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=150.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=150.0, Time-Width:=T#15 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=80.0, Time-Width:=T#4 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=80.0, Time-Width:=T#5 m, Wait-Width:=3.0, WaitTimeLimit:=T#1 m),<br>(ReachSP:=10.0, Time-Width:=T#0 s, Wait-Width:=3.0, WaitTimeLimit:=T#1 m)] | 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#0 s, ElapseTime:=T#0 s, ProgressTime:=T#0 s, LeftTime:=T#0 s, StepProgressTime:=T#0 s, StepLeftTime:=T#0 s)                                  | 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#250 ms, 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  | TimeProportional-Out error                              |
|                    | PIDbank       | ARRAY[0..7] OF sPID_BANK | [8((PB:=10, TI:=T#233 s, TD:=T#60 s))]   | 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           |  |   |

| Internal Variables | Variable | Data type            | Initial value | Comment |
|--------------------|----------|----------------------|---------------|---------|
|                    | PID      | PIDAT                |               |         |
|                    | TPO      | TimeProportional-Out |               |         |

| 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

| Instruction    | Name                            | Page       |
|----------------|---------------------------------|------------|
| TraceSamp      | Data Trace Sampling             | page 2-859 |
| TraceTrig      | Data Trace Trigger              | page 2-862 |
| GetTraceStatus | Read Data Trace Status          | page 2-865 |
| SetAlarm       | Create User-defined Error       | page 2-868 |
| ResetAlarm     | Reset User-defined Error        | page 2-873 |
| GetAlarm       | Get User-defined Error Status   | page 2-875 |
| ResetPLCError  | Reset PLC Controller Error      | page 2-877 |
| GetPLCError    | Get PLC Controller Error Status | page 2-880 |
| ResetCJBError  | Reset CJ Bus Controller Error   | page 2-882 |
| GetCJBError    | Get I/O Bus Error Status        | page 2-884 |
| GetEIPErr      | Get EtherNet/IP Error Status    | page 2-886 |
| ResetMCErr     | Reset Motion Control Error      | page 2-888 |
| GetMCErr       | Get Motion Control Error Status | page 2-894 |
| ResetECErr     | Reset EtherCAT Error            | page 2-896 |
| GetECErr       | Get EtherCAT Error Status       | page 2-898 |
| ResetNXBErr    | Reset NX Bus Error              | page 2-901 |
| GetNXBErr      | Get NX Bus Error Status         | page 2-903 |
| GetNXUnitErr   | Get NX Unit Error Status        | page 2-905 |
| SetInfo        | Create User-defined Information | page 2-912 |
| ResetUnit      | Restart Unit                    | page 2-914 |
| GetNTPStatus   | Read NTP Status                 | page 2-919 |

| <b>Instruction</b>       | <b>Name</b>                      | <b>Page</b> |
|--------------------------|----------------------------------|-------------|
| RestartNXUnit            | Restart NX Unit                  | page 2-921  |
| NX_ChangeWriteMode       | Change to NX Unit Write Mode     | page 2-927  |
| NX_SaveParam             | Save NX Unit Parameters          | page 2-933  |
| PLC_ReadTotalPowerOnTime | Read PLC Total Power ON Time     | page 2-939  |
| NX_ReadTotalPowerOnTime  | Read NX Unit Total Power ON Time | page 2-942  |

# TraceSamp

The TraceSamp instruction performs sampling for a data trace.

| Instruction | Name                | FB/<br>FUN | Graphic expression | ST expression              |
|-------------|---------------------|------------|--------------------|----------------------------|
| TraceSamp   | Data Trace Sampling | FUN        |                    | TraceSamp(TraceNo, Point); |

## Variables

|         | Meaning               | I/O    | Description           | Valid range           | Unit | Default |
|---------|-----------------------|--------|-----------------------|-----------------------|------|---------|
| TraceNo | Trace number          | Input  | Trace number          | 0 to 3 <sup>*1</sup>  | ---  | 0       |
| Point   | Sampling point number |        | Sampling point number | Depends on data type. |      |         |
| Out     | Return value          | Output | Always TRUE           | TRUE only             | ---  | ---     |

\*1. The range is 0 to 1 for NX102, NX1P2, NJ301 and NJ101 CPU Units.

|         | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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          |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |

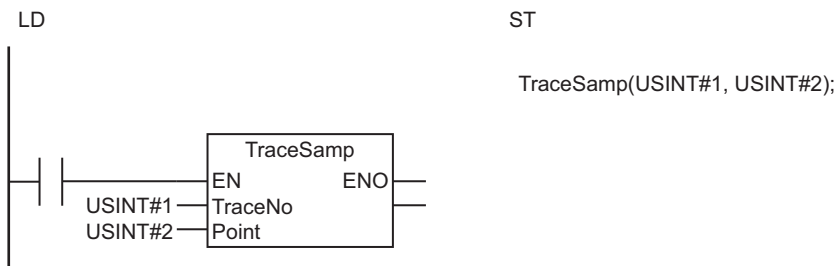
## Function

The TraceSamp instruction performs sampling for a data trace.

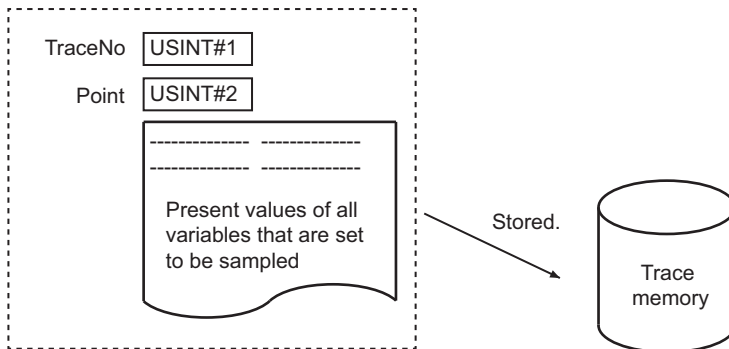
The sampling settings are specified from the Sysmac Studio. The present values for all variables that are set to be sampled are read and stored with trace number *TraceNo* and sampling point number *Point* in trace memory.

This instruction is executed only during execution of data tracing and only when the sampling timing is set to **Use sampling instruction** 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.



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                     |
|--|-------------------|----------------------------|---------------------------------|
| <code>_PLC_TraceSta[0..3]</code> <sup>*1</sup> | Trace Information | <code>_sTRACE_STA[]</code> | Trace information <sup>*2</sup> |

\*1. NX102, NX1P2, NJ301 or NJ101 CPU Unit: The variable name is `_PLC_TraceSta[0..1]`.

\*2. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

### Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* 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.
  - a) Data tracing is stopped.
  - b) The sampling timing is not set to **Use sampling instruction** in the trace settings.
  - c) 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.

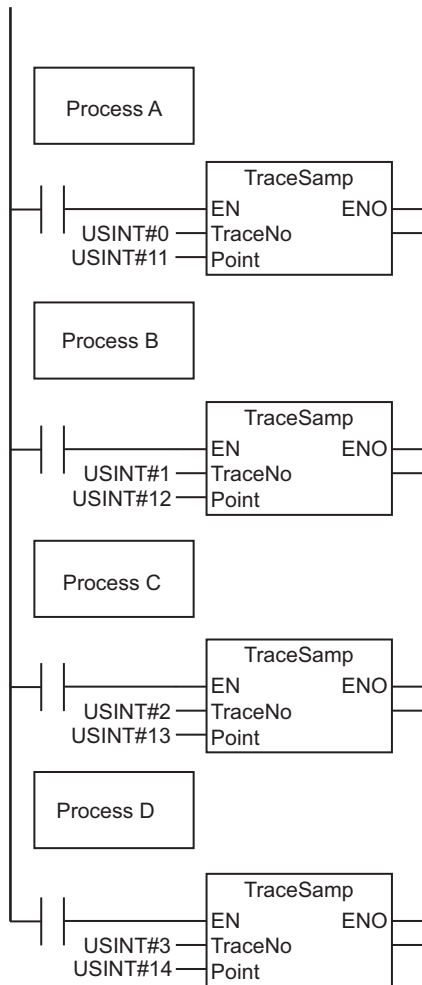
- a) The value of *TraceNo* is outside of the valid range.

## 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/<br>FUN | Graphic expression | ST expression       |
|-------------|--------------------|------------|--------------------|---------------------|
| TraceTrig   | Data Trace Trigger | FUN        |                    | TraceTrig(TraceNo); |

## Variables

|         | Meaning      | I/O    | Description  | Valid range         | Unit | Default |
|---------|--------------|--------|--------------|---------------------|------|---------|
| TraceNo | Trace number | Input  | Trace number | 0 to 3 <sup>1</sup> | ---  | 0       |
| Out     | Return value | Output | Always TRUE  | TRUE only           | ---  | ---     |

\*1. The range is 0 to 1 for NX102, NX1P2, NJ301 and NJ101 CPU Units.

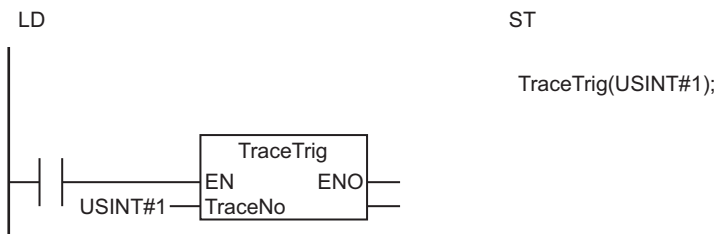
|         | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|---------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|-------------------|-------|--|------|-----|----|--------|
|         | 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

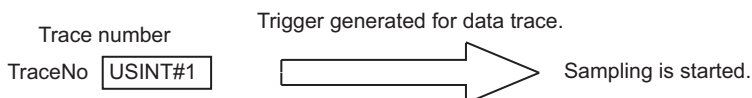
The TraceTrig instruction generates a trigger for data tracing.

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.



Here, a data trace trigger is generated for trace number *TraceNo*.



## Related System-defined Variables

| Name   | Meaning           | Data type                  | Description                     |
|--|-------------------|----------------------------|---------------------------------|
| <code>_PLC_TraceSta[0..3]</code> <sup>*1</sup> | Trace Information | <code>_sTRACE_STA[]</code> | Trace information <sup>*2</sup> |

\*1. NX102, NX1P2, NJ301 or NJ101 CPU Unit: The variable name is `_PLC_TraceSta[0..1]`.

\*2. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* 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 this instruction is used in ST.
- In the following cases, nothing is done and the instruction ends normally.
  - a) Data tracing is stopped.
  - b) The trigger condition has already been met.
  - c) The value of *TraceNo* is not the trace number set from the Sysmac Studio.
  - d) **A continuous trace** is specified as the trace type for the trace number that is specified with *TraceNo*.
- An error will occur in the following case. *ENO* will be FALSE.
  - a) The value of *TraceNo* is outside the valid range.

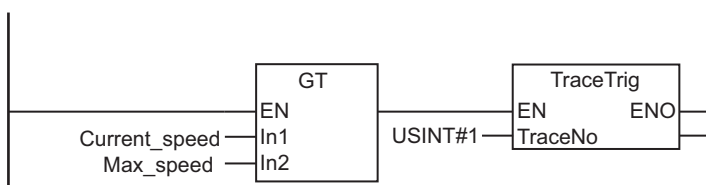
## Sample Programming

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       |
|----------------------|-----------|---------------|---------------|
| <i>Current_speed</i> | INT       | 0             | Current speed |
| <i>Max_speed</i>     | 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/<br>FUN | Graphic expression | ST expression  |
|---------------------|---------------------------|------------|--------------------|--|
| GetTraceSta-<br>tus | Read Data<br>Trace Status | FUN        |                    | GetTraceStatus(TraceNo, IsStart, IsComplete, ParamErr, IsTrigger); |

## Variables

|            | Meaning              | I/O   | Description   | Valid range           | Unit | Default |
|------------|----------------------|---|---|-----------------------|------|---------|
| TraceNo    | Trace number         | Input   | Trace number  | 0 to 3 <sup>*1</sup>  | ---  | 0       |
| Out        | Return value         | Output  | Always TRUE   | TRUE only             | ---  | ---     |
| IsStart    | Executing flag       |   | TRUE: Data trace in progress.<br>FALSE: Data trace not in progress.               | Depends on data type. |      |         |
| IsComplete | Completed flag       |   | TRUE: Data trace was completed.<br>FALSE: Data trace in progress or not executed. |                       |      |         |
| ParamErr   | Parameter error flag |   | TRUE: Data trace setting error.<br>FALSE: No data trace setting error.            |                       |      |         |
| IsTrigger  | Trigger flag         | TRUE: Data trace trigger condition met.<br>FALSE: Data trace trigger condition not met. |   |                       |      |         |

\*1. The range is 0 to 1 for NX102, NX1P2, NJ301 and NJ101 CPU Units.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| IsComplete | OK          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| ParamErr   | OK          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| IsTrigger  | OK          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

## Function

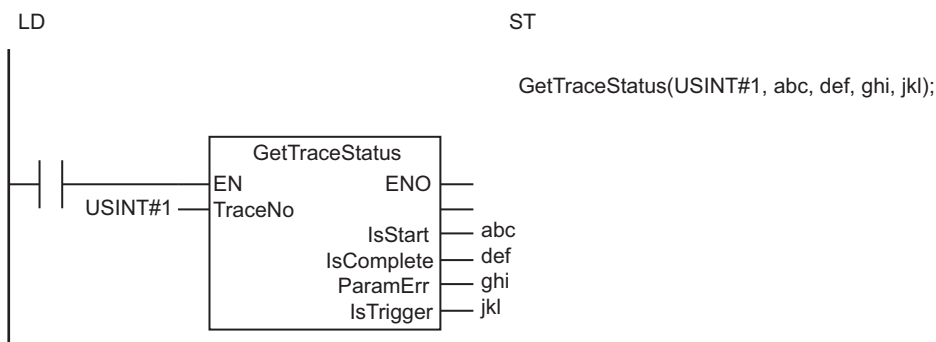
The GetTraceStatus instruction reads the execution status of the data trace that is specified with trace number *TraceNo*.

The status that is read is output to execution flag *IsStart*, completed flag *IsComplete*, parameter error flag *ParamErr*, and trigger flag *IsTrigger*.

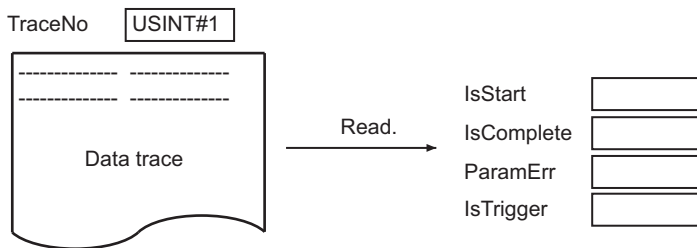
The value of *ParamErr* changes to TRUE when one of the following errors is found in the trace settings.

- 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.



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         |
|-------------------------------------|-------------------|----------------------------|---------------------|
| <code>_PLC_TraceSta[0..3]</code> *1 | Trace Information | <code>_sTRACE_STA[]</code> | Trace information*2 |

\*1. NX102, NX1P2, NJ301 or NJ101 CPU Unit: The variable name is `_PLC_TraceSta[0..1]`.

\*2. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details on data tracing.

## Precautions for Correct Use

- Return value *Out* is not used when this instruction is used in ST.

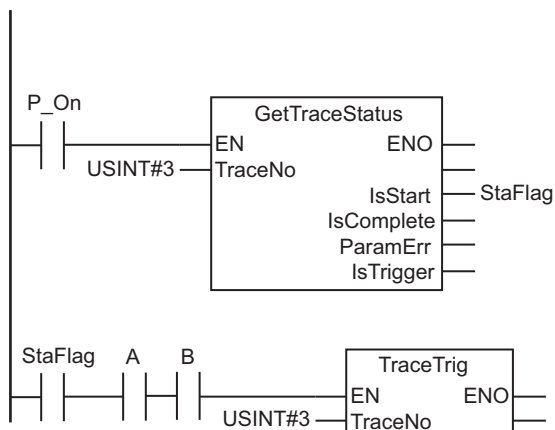
- This instruction reads the contents of the `_PLC_TraceSta[]` system-defined variable. You cannot access this variable directly. Always use this instruction to read the contents of the variable.
- An error will occur in the following case. *ENO* will be FALSE.
  - a) The value of *TraceNo* is outside 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/<br>FUN | Graphic expression | ST expression                 |
|-------------|---------------------------|------------|--------------------|-------------------------------|
| SetAlarm    | Create User-defined Error | FUN        |                    | SetAlarm(Code, Info1, Info2); |

## Variables

|       | 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. |      | *1      |
| Info2 | Attached information 2 |        |   |                       |      |         |
| 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.

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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*.

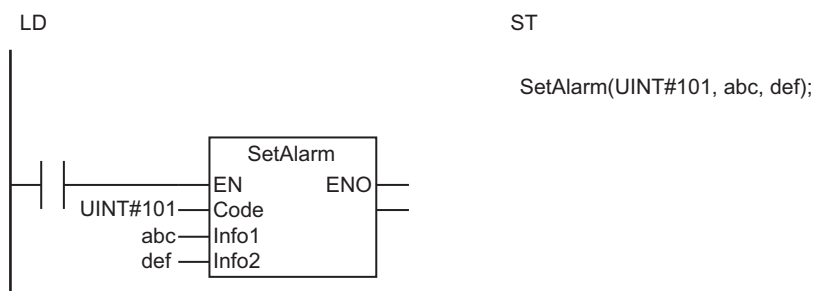
Event codes are defined in the event setting table on the Sysmac Studio.

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.

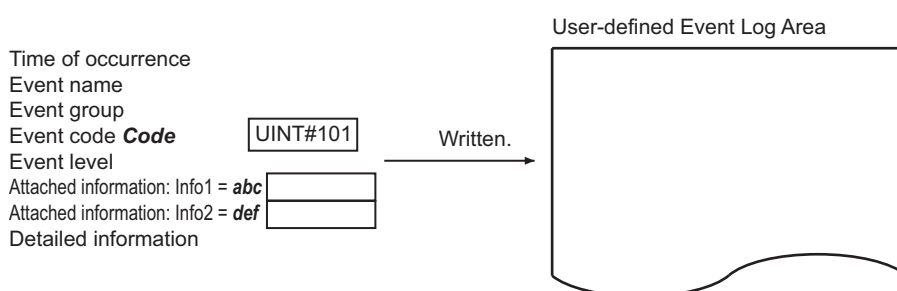
The event level that corresponds to the event code is recorded. The event levels are given below. The smaller the event code is, the higher the event level is.

| 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.



A user-defined error with event code **Code** is generated. Also, 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.



## 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)* for details.

## Additional Information

You can specify 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 are passed to *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 set with an event code which is not registered in 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 will occur in the following cases. *ENO* will be FALSE.
  - a) The value of *Code* is outside the valid range.
  - b) An attempt was made to generate more than the maximum number of user-defined errors.

## Sample Programming

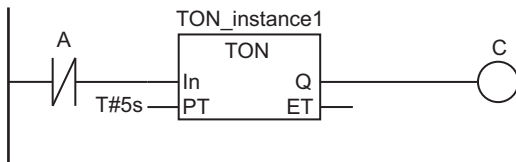
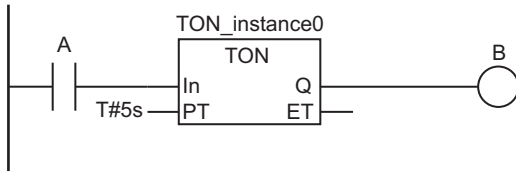
In this sample, the value of variable *A* changes between TRUE and FALSE every five seconds. The value of *A* is monitored. If it does not change for more than five seconds, a user-defined error with event code 102 is generated. UINT#123 and UINT#456 are given as the attached information. When variable *F* changes to TRUE, the user-defined error is cleared.

### LD

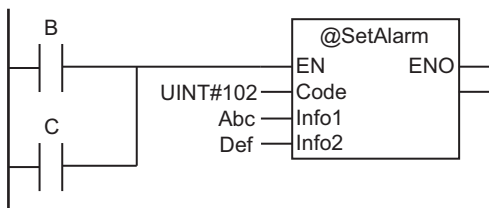
| Internal Variables | Variable      | Data type | Initial value |
|--------------------|---------------|-----------|---------------|
|                    | A             | BOOL      | FALSE         |
|                    | B             | BOOL      | FALSE         |
|                    | C             | 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      | <input checked="" type="checkbox"/> | Error Status of User-defined Errors |

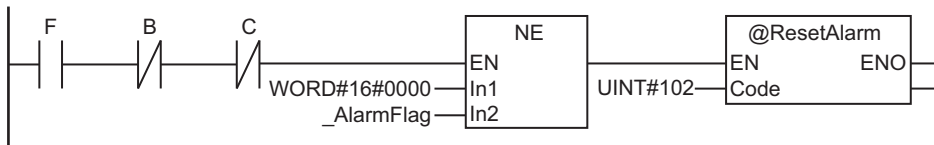
Check the value of variable **A**.



Create user-defined error.



Reset user-defined error.



## 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;

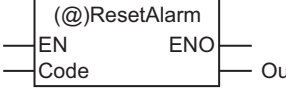
// Create user-defined error.
IF (B=TRUE) OR (C=TRUE) THEN
    SetAlarm(
        Code:=UINT#102,
        Info1 :=Abc,
        info2 :=Def);
END_IF;

// 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/<br>FUN | Graphic expression  | ST expression     |
|-------------|--------------------------|------------|---|-------------------|
| ResetAlarm  | Reset User-defined Error | FUN        |  | ResetAlarm(Code); |

## Variables

|      | Meaning      | I/O    | Description  | Valid range | Unit | Default |
|------|--------------|--------|--|-------------|------|---------|
| Code | Event code   | Input  | Event code of user-defined error to reset<br>16#0: Reset all application errors. | 0 to 40000  | ---  | 0       |
| Out  | Return value | Output | Always TRUE  | TRUE only   | ---  | ---     |

|      | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |  |

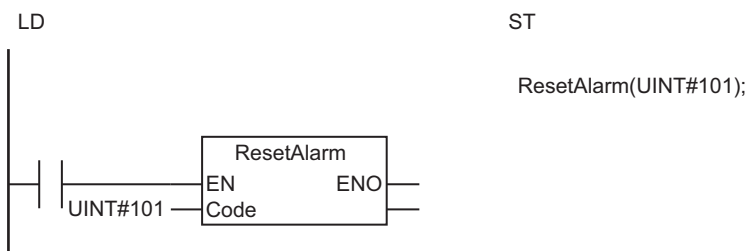
## Function

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".

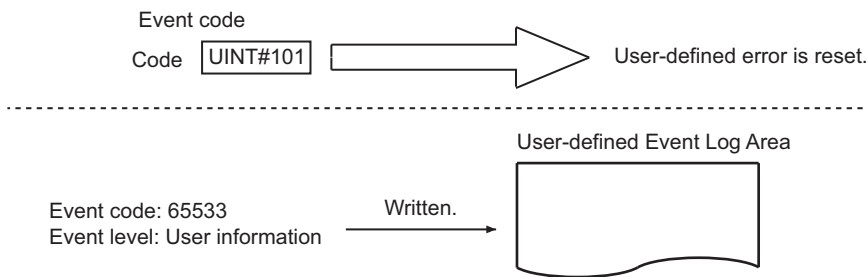
If the value of *Code* is 0, all current user-defined errors are reset.

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.



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)* 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 will occur in the following case. *ENO* will be FALSE.
  - a) The value of *Code* is outside the valid range.

### Sample Programming

Refer to *Sample Programming* on page 2-870 for the SetAlarm instruction.

# 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/<br>FUN | Graphic expression | ST expression               |
|-------------|-------------------------------|------------|--------------------|-----------------------------|
| GetAlarm    | Get User-defined Error Status | FUN        |                    | Out:=GetAlarm(Level, Code); |

## Variables

|       | Meaning                  | I/O    | Description   | Valid range           | Unit | Default |
|-------|--------------------------|--------|---|-----------------------|------|---------|
| Out   | Error flag               | Output | TRUE: User-defined error exists.<br>FALSE: No user-defined error  | Depends on data type. |      |         |
| Level | Highest event level      |        | Highest event level of all current user-defined errors<br>0: No user-defined error<br>1 to 8: Event level         | 0 to 8                | ---  | ---     |
| Code  | Highest level event code |        | Highest level event code of all current user-defined errors<br>0: No user-defined error<br>1 to 40000: Event code | 0 to 40000            |      |         |

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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   |       |       |      |     |      |                      |      |  |      |      |     |    |        |

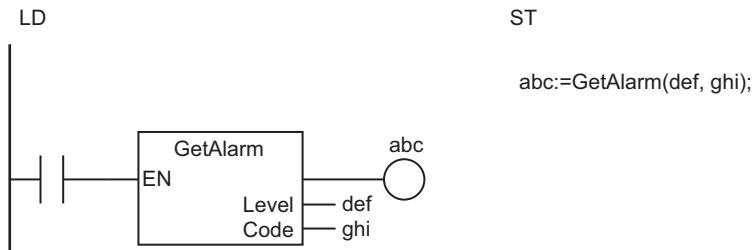
## Function

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*.

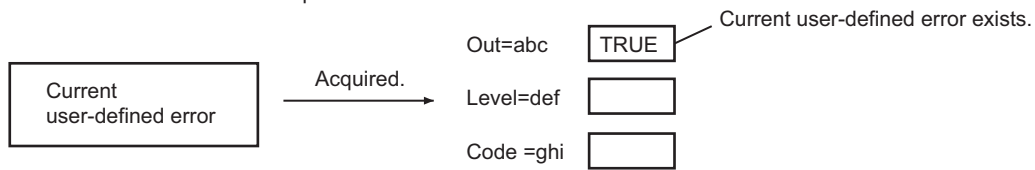
If there are currently no user-defined errors, the value of error flag *Out* is FALSE.

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.



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 |

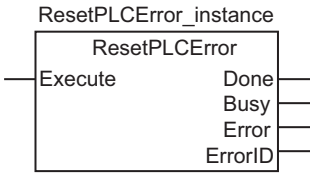
\*1. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

### Precautions for Correct Use

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

# ResetPLCError

The ResetPLCError instruction resets errors in the PLC Function Module.

| Instruction   | Name                       | FB/<br>FUN | Graphic expression  | ST expression                                       |
|---------------|----------------------------|------------|---|---|
| ResetPLCError | Reset PLC Controller Error | FB         |  | ResetPLCError(Execute, Done, Busy, Error, ErrorID); |

## Variables

Only common variables are used.

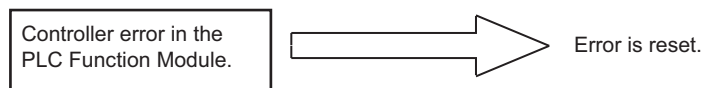
## Function

The ResetPLCError instruction resets errors in the PLC Function Module.

The following figure shows a programming example.



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)* 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.

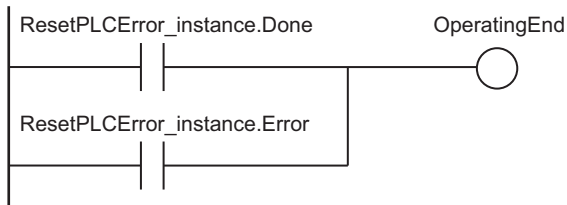
## Sample Programming

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).

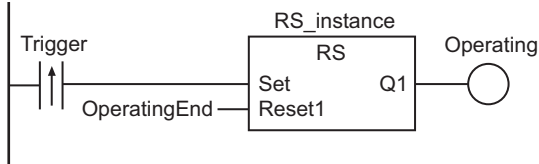
### LD

| 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 |               |                      |

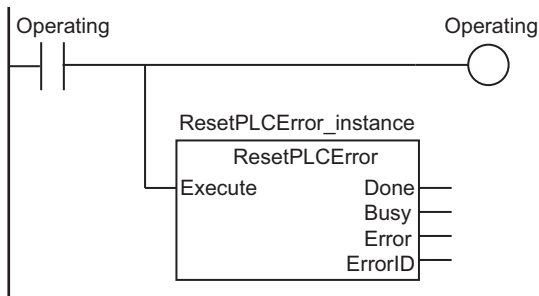
Determine if execution of the ResetPLCError has ended.



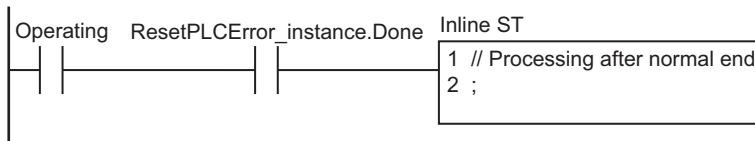
Accept trigger.

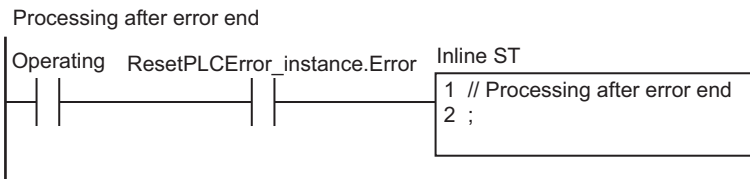


Execute ResetPLCError instruction.



Processing after normal end



**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/<br>FUN | Graphic expression | ST expression                  |
|-------------|---------------------------------|------------|--------------------|--------------------------------|
| GetPLCError | Get PLC Controller Error Status | FUN        |                    | Out:=GetPLCError(Level, Code); |

## Variables

|       | Meaning                  | I/O    | Description   | Valid range                                   | Unit | Default |
|-------|--------------------------|--------|---|---|------|---------|
| Out   | Error flag               | Output | TRUE: Controller error exists.<br>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<br>0: No Controller error<br>2: Partial fault level<br>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<br>16#0000_0000: No Controller error<br>16#0007_0000 to 16#FFFF_FFFF: Event code | 16#00000000,<br>16#00070000 to<br>16#FFFFFFFF |      |         |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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

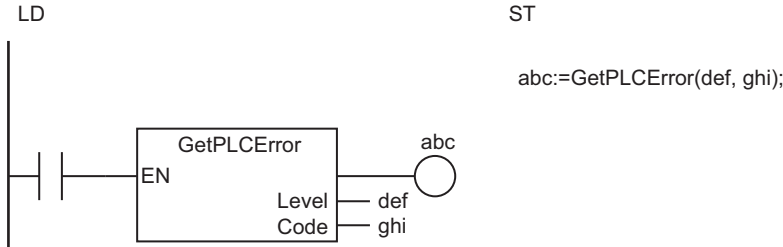
The GetPLCError instruction gets the highest level status and the highest level event code of the current Controller errors in the PLC Function Module and outputs them to *Level* and *Code*.



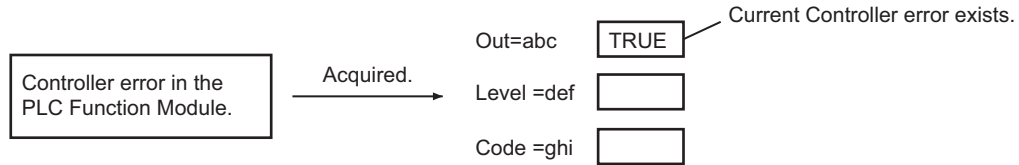
If there are currently no Controller errors, the value of error flag *Out* is FALSE.

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.



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)* for details.

# ResetCJBError

The ResetCJBError instruction resets Controller errors in the I/O bus.

| Instruction   | Name                          | FB/<br>FUN | Graphic expression | ST expression  |
|---------------|-------------------------------|------------|--------------------|--|
| ResetCJBError | Reset CJ Bus Controller Error | FB         |                    | ResetCJBError_instance(Execute, UnitNo, Done, Busy, Error, ErrorID); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units.

## Variables

|        | Meaning     | I/O   | Description                           | Valid range   | Unit | Default   |
|--------|-------------|-------|---------------------------------------|---|------|-----------|
| UnitNo | Unit number | Input | Unit number for which to reset errors | _CBU_No00 to _CBU_No15, _SIO_No00 to _SIO_No95, _UNIT_ALL | ---  | _UNIT_ALL |

|        | Boo lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-882 for the enumerators of the enumerated type <code>_eUnitNo</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

## Function

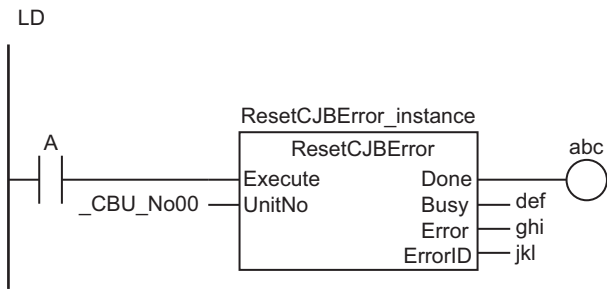
The ResetCJBError instruction resets a Controller error in the I/O bus.

If the Unit specified with *UnitNo* is a CJ-series Special Unit, the Unit is restarted as well.

The data type of *UnitNo* is enumerated type `_eUnitNo`. The meanings of the enumerators are as follows:

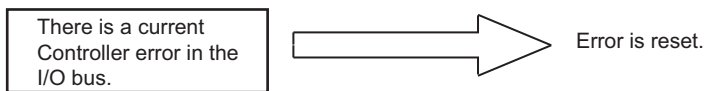
| Enumerators            | Meaning                                   |
|------------------------|---|
| _CBU_No00 to _CBU_No15 | Unit number of CPU Bus Unit, 00 to 15     |
| _SIO_No00 to _SIO_No95 | Unit number of Special I/O Unit, 00 to 95 |
| _UNIT_ALL              | All Units                                 |

The following example is for when *UnitNo* is `_CBU_No00`. The Controller error on the I/O bus is reset and the CPU Bus Unit with unit number 0 is restarted.

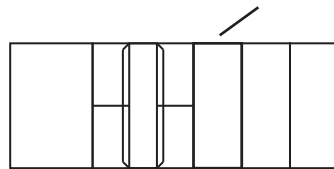


```
ST
ResetCJBError_instance(A, _CBU_No00,
    abc, def, ghi, jkl);
```

The ResetCJBError instruction resets a Controller error in the I/O bus. Also, the CPU Bus Unit with unit number *UnitNo* is restarted.



UnitNo = \_CBU\_No00: CPU Bus Unit with unit number 0 is restarted.



### Related System-defined Variables

| Name        | Meaning              | Data type | Description                                  |
|-------------|----------------------|-----------|--|
| _CJB_ErrSta | I/O Bus Error Status | WORD      | Contains the error status of the I/O bus. *1 |

\*1. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

### Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetCJBError instruction to confirm that the errors were reset.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *UnitNo* is outside the valid range.
  - b) The Unit specified with *UnitNo* does not exist.

# GetCJBError

The GetCJBError instruction gets the highest level status and highest level event code of the current Controller errors in the I/O bus of the NJ-series CPU Unit.

| Instruction | Name                     | FB/<br>FUN | Graphic expression | ST expression                  |
|-------------|--------------------------|------------|--------------------|--------------------------------|
| GetCJBError | Get I/O Bus Error Status | FUN        |                    | Out:=GetCJBError(Level, Code); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units.

## Variables

|       | Meaning                  | I/O    | Description   | Valid range                            | Unit | Default |
|-------|--------------------------|--------|---|--|------|---------|
| Out   | Error flag               | Output | TRUE: Controller error exists.<br>FALSE: No Controller error  | Depends on data type.                  | ---  | ---     |
| Level | Highest level status     |        | Highest level status of all current Controller errors in the I/O bus<br>0: No Controller error<br>2: Partial fault level<br>3: Minor fault level          | 0, 2, or 3                             |      |         |
| Code  | Highest level event code |        | Highest level event code of all current Controller errors in the I/O bus<br>16#0000_0000: No Controller error<br>16#0007_0000 to 16#FFFF_FFFF: Event code | 16#00000000, 16#00070000 to 16#FFFFFFF |      |         |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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    |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

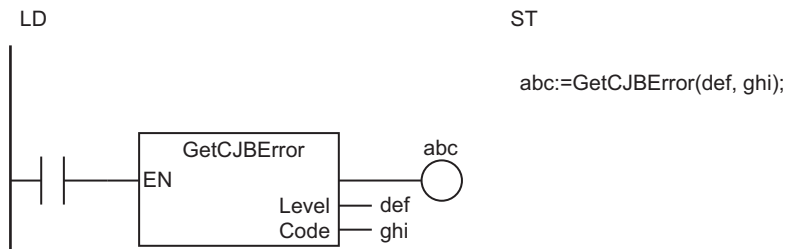
## Function

The GetCJBError instruction gets the highest level status and the highest level event code of the current Controller errors in the I/O bus and outputs them to *Level* and *Code*.

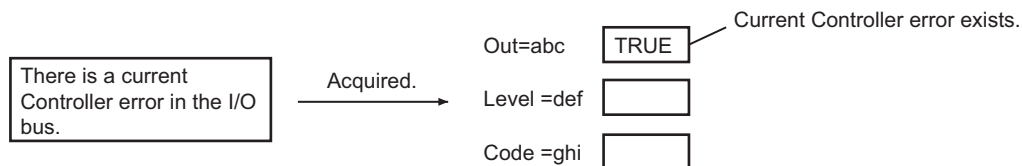
If there are currently no Controller errors, the value of error flag *Out* is FALSE.

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.



The GetCJBError instruction gets the highest level status and the highest level event code of the current Controller errors in the I/O bus and outputs them to *Level* and *Code*.



## Related System-defined Variables

| Name        | Meaning              | Data type | Description                                  |
|-------------|----------------------|-----------|--|
| _CJB_ErrSta | I/O Bus Error Status | WORD      | Contains the error status of the I/O bus. *1 |

\*1. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* 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/<br>FUN | Graphic expression | ST expression                  |
|-------------|------------------------------|------------|--------------------|--------------------------------|
| GetEIPError | Get EtherNet/IP Error Status | FUN        |                    | Out:=GetEIPError(Level, Code); |

## Variables

|       | Meaning                  | I/O    | Description   | Valid range                             | Unit | Default |
|-------|--------------------------|--------|---|---|------|---------|
| Out   | Error flag               | Output | TRUE: Controller error exists.<br>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<br>0: No Controller error<br>2: Partial fault level<br>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<br>16#0000_0000: No Controller error<br>16#0007_0000 to 16#FFFF_FFFF: Event code | 16#00000000, 16#00070000 to 16#FFFFFFFF |      |         |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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    |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

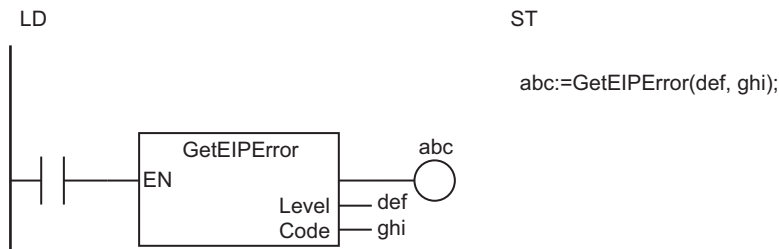
## Function

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*.

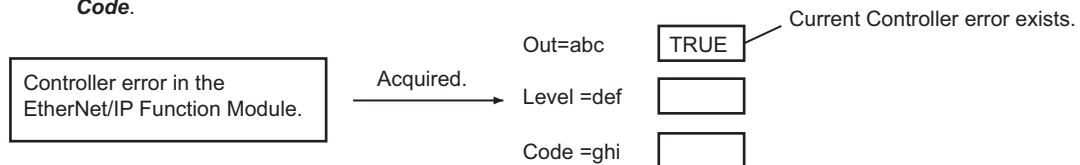
If there are currently no Controller errors, the value of error flag *Out* is FALSE.

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.



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)* for details.

# ResetMCErr

The ResetMCErr instruction resets Controller errors in the Motion Control Function Module.

| Instruction | Name                       | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|----------------------------|------------|--------------------|---|
| ResetMCErr  | Reset Motion Control Error | FB         |                    | ResetMCErr_instance(Execute, Done, Busy, Failure Error, ErrorID); |

## Variables

|         | Meaning     | I/O    | Description  | Valid range           | Unit | Default |
|---------|-------------|--------|--|-----------------------|------|---------|
| Failure | Failure end | Output | TRUE: The errors were not reset.<br>FALSE: The errors were reset normally. | Depends on data type. | ---  | ---     |

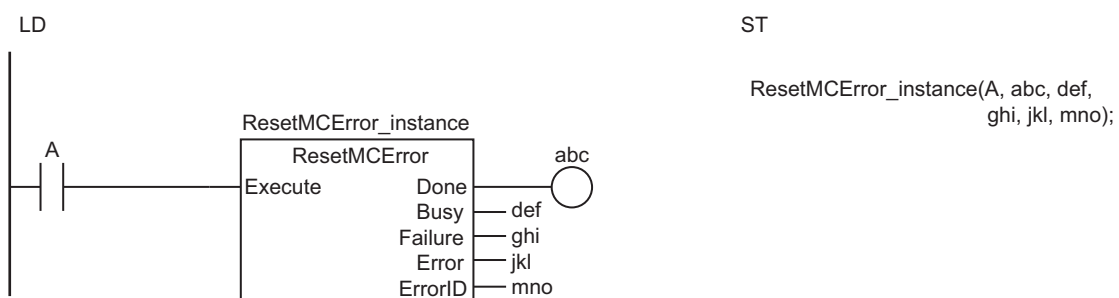
|         | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|---------|----------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|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

## Function

The ResetMCErr instruction resets a Controller error in the Motion Control Function Module. If the errors are not reset, the value of *Failure* changes to TRUE.

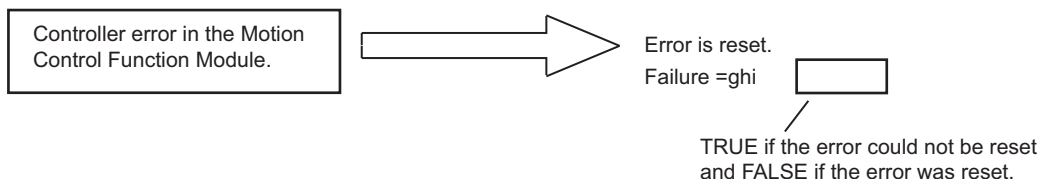
No matter what task the program that executes the ResetMCErr is placed in, this instruction resets errors for all axes and all axes groups.

The following figure shows a programming example.





The ResetMCErr 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)* for details.

## Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetMCErr 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 ResetECErr instruction is not executed at the same time. If the ResetMCErr and ResetECErr 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     | 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.

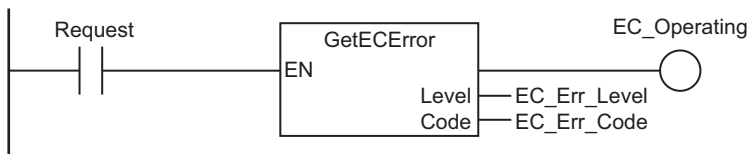
The processing procedure is as follows:

- 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 GetMCErr instruction is executed to detect any Controller errors in the Motion Control Function Module.
- 4** If errors are detected, they are reset with the ResetMCErr instruction.

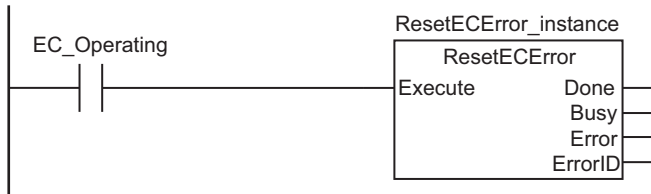
### LD

| Variable              | Data type    | Initial value | Comment   |
|-----------------------|--------------|---------------|---|
| Request               | BOOL         | FALSE         | Error detection reset request                               |
| EC_Err_Level          | UINT         | 0             | EtherCAT Master Function Module<br>Highest event level      |
| EC_Err_Code           | DWORD        | DWORD#16#0    | EtherCAT Master Function Module<br>Highest level event code |
| EC_Operating          | BOOL         | FALSE         | Resetting error in EtherCAT Master Function Module          |
| MC_Err_Level          | UINT         | 0             | Motion Control Function Module<br>Highest event level       |
| MC_Err_Code           | DWORD        | DWORD#16#0    | Motion Control Function Module<br>Highest level event code  |
| MC_Operating          | BOOL         | FALSE         | Resetting error in Motion Control Function Module           |
| Normal_End            | BOOL         | FALSE         | Normal end  |
| ResetECError_instance | ResetECError |               |   |
| ResetMCErr_instance   | ResetMCErr   |               |   |

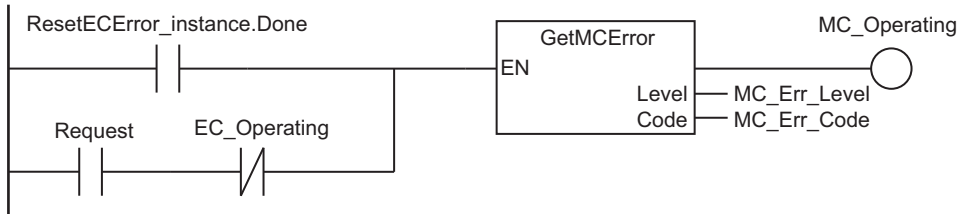
Execute GetECError instruction.



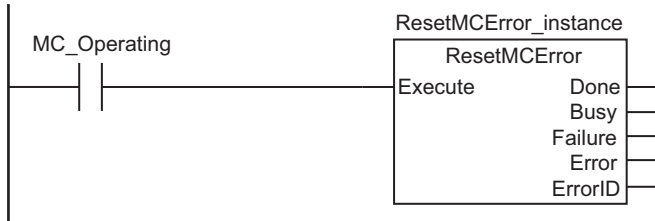
Execute ResetECError instruction if error occurs in EtherCAT Master Function Module.



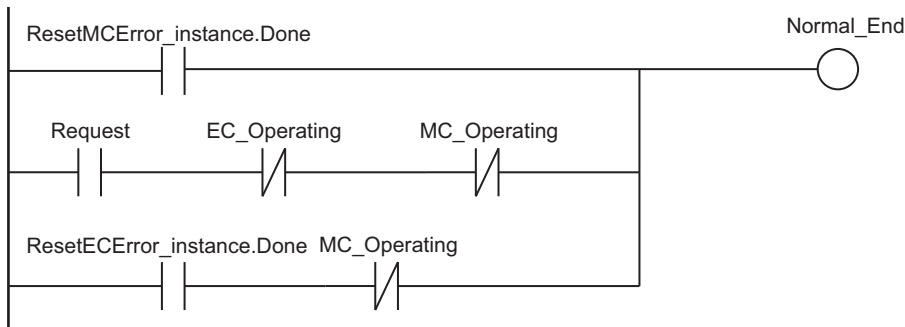
Execute GetMCErr instruction after resetting error in EtherCAT Master Function Module or if there is no error.



Execute ResetMCErr instruction if error occurs in Motion Control Function Module.



Processing after normal end



## 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 |               |   |
| ResetMCErr_instance   | ResetMCErr   |               |   |

```
// Determine error resetting requests.
```

```
IF (Request=TRUE) THEN
```

```

// Detect Controller errors in EtherCAT Master Function Module.
  EC_Error:=GetECError(EC_Err_Level, EC_Err_Code);

// Detect Controller errors in Motion Control Function Module.
  MC_Error:=GetMCError(MC_Err_Level, MC_Err_Code);

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
        ResetMCError_instance(Execute:=TRUE); // Recover operation for all slaves.
        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_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/<br>FUN | Graphic expression | ST expression                 |
|-------------|---------------------------------|------------|--------------------|-------------------------------|
| GetMCError  | Get Motion Control Error Status | FUN        |                    | Out:=GetMCError(Level, Code); |

## Variables

|       | Meaning                  | I/O    | Description  | Valid range                            | Unit | Default |
|-------|--------------------------|--------|--|--|------|---------|
| Out   | Error flag               | Output | TRUE: Controller error exists.<br>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<br>0: No Controller error<br>2: Partial fault level<br>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<br>16#0000_0000: No Controller error<br>16#0007_0000 to 16#FFFF_FFFF: Event code | 16#00000000, 16#00070000 to 16#FFFFFFF |      |         |

|       | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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    |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

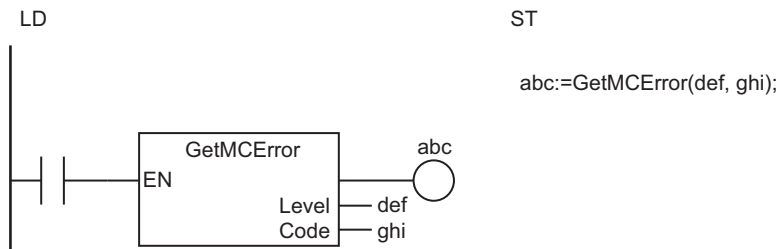
## Function

The GetMCErr 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*.

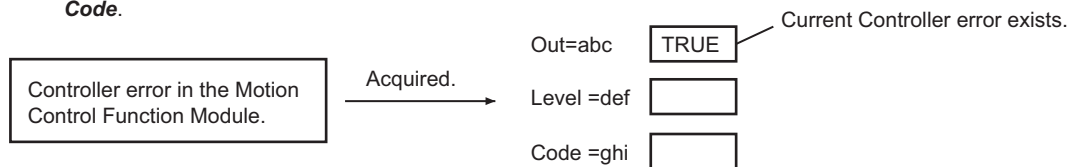
If there are currently no Controller errors, the value of error flag *Out* is FALSE.

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.



The GetMCErr 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 | 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)* for details.

## Sample Programming

Refer to *Sample Programming* on page 2-889 for the ResetMCErr instruction.

# ResetECError

The ResetECError instruction resets Controller errors in the EtherCAT Master Function Module.

| Instruction  | Name                      | FB/<br>FUN | Graphic expression | ST expression  |
|--------------|---------------------------|------------|--------------------|--|
| ResetECError | Reset Ether-<br>CAT Error | FB         |                    | ResetECError_instance(Execute,<br>Done, Busy, Error, ErrorID); |

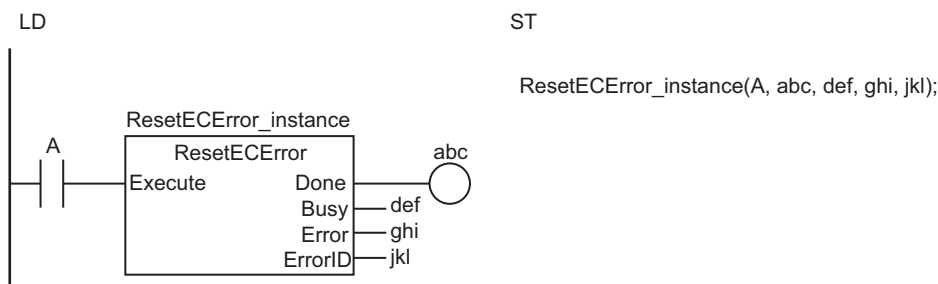
## Variables

Only common variables are used.

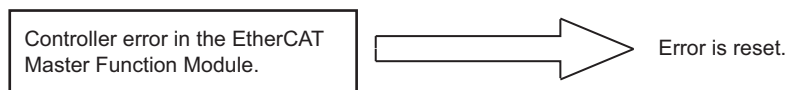
## Function

The ResetECError instruction resets Controller errors in the EtherCAT Master Function Module.

The following figure shows a programming example.



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)* 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 ResetMCErr, 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\_DisconnectSlave, EC\_ConnectSlave, EC\_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX\_ChangeWriteMode.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) This instruction is executed again while processing to clear a Controller error from the EtherCAT Master Function Module is in progress.
  - b) The EC\_DisconnectSlave, EC\_ConnectSlave, EC\_ChangeEnableSetting, ResetECError, RestartNXUnit, or NX\_ChangeWriteMode instruction is already in execution.

## Sample Programming

Refer to *Sample Programming* on page 2-889 for the ResetMCErr instruction.

# GetECError

The GetECError instruction detects errors in the EtherCAT Master Function Module.

| Instruction | Name                      | FB/<br>FUN | Graphic expression | ST expression                 |
|-------------|---------------------------|------------|--------------------|-------------------------------|
| GetECError  | Get EtherCAT Error Status | FUN        |                    | Out:=GetECError(Level, Code); |



## Version Information

A CPU Unit with unit version 1.02 or later is required to detect slave errors with this instruction.

## Variables

|       | Meaning                  | I/O    | Description   | Valid range                            | Unit | Default |
|-------|--------------------------|--------|---|--|------|---------|
| Out   | Error flag               | Output | TRUE: Error exists* <sup>1</sup><br>FALSE: No error   | Depends on data type.                  | ---  | ---     |
| Level | Highest level status     |        | Status of the current error with the highest level* <sup>1</sup><br>0: No error<br>2: Partial fault level<br>3: Minor fault level | 0, 2, or 3                             |      |         |
| Code  | Highest level event code |        | Event code of the current error with the highest level* <sup>1</sup>  | 16#00000000, 16#00070000 to 16#FFFFFFF |      |         |

\*1. The errors that are detected depend on the unit version of the CPU Unit and the version of the Sysmac Studio. Refer to *Detected Errors and Output Variable Values* on page 2-899 for details.

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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                |

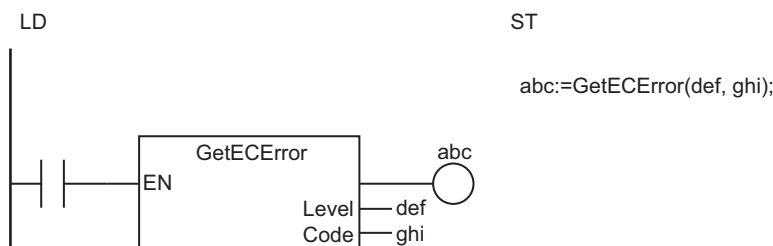
The following table shows the relationship between the unit version of the CPU Unit, the status of the EtherCAT Master Function Module and values of the output variables.

| Unit version of CPU Unit | Status of EtherCAT Master Function Module | Value of Out | Value of Level   | Value of Code   |
|--------------------------|---|--------------|--|---|
| 1.04 or later            | No error                                  | FALSE        | 0  | 16#0000_0000  |
|                          | Communications port error or master error | TRUE         | Status of the current error with the highest level<br>2: Partial fault level<br>3: Minor fault level | Event code of the current error with the highest level*1<br>16#0007_0000 to<br>16#FFFF_FFFF |
|                          | Slave error                               |              |  | 16#0000_0000  |
| 1.02 or 1.03             | No error                                  | FALSE        | 0  | 16#0000_0000  |
|                          | Communications port error or master error | TRUE         | Status of the current error with the highest level<br>2: Partial fault level<br>3: Minor fault level | Event code of the current error with the highest level*1<br>16#0007_0000 to<br>16#FFFF_FFFF |
|                          | Slave error                               |              |  | 16#0000_0000  |
| 1.00 or 1.01             | No error                                  | FALSE        | 0  | 16#0000_0000  |
|                          | Communications port error or master error | TRUE         | Status of the current error with the highest level<br>2: Partial fault level<br>3: Minor fault level | Event code of the current error with the highest level*1<br>16#0007_0000 to<br>16#FFFF_FFFF |
|                          | Slave error                               |              |  | 16#0000_0000  |

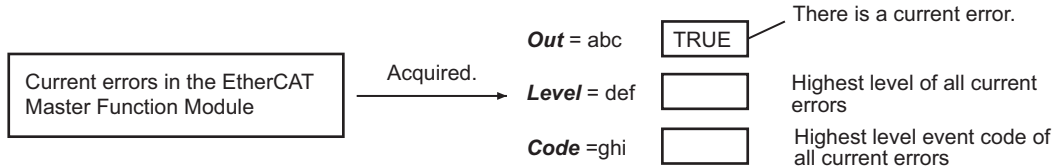
\*1. If there is more than one error at the highest event level, the value is the event code for the error that occurred first.

## Notation Example

The following figure shows a programming example.



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. *1                               |
| _EC_PortErr *2 | Communications Port Error | WORD      | Contains a summary of the EtherCAT master communications port errors. *1                                  |
| _EC_MstrErr *2 | Master Error              | WORD      | Contains a summary of the EtherCAT master errors and the slave errors detected by the EtherCAT master. *1 |
| _EC_SlavErr    | Slave Error               | WORD      | Contains a summary of the overall EtherCAT slave error status. *1   |

\*1. Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual (Cat. No. W505)* for details.

\*2. The GetECError instruction gets the errors that are shown by \_EC\_PortErr (Communications Port Error) and \_EC\_MstrErr (Master Error).

### Sample Programming

Refer to *Sample Programming* on page 2-889 for the ResetMCErr instruction.

# ResetNXBError

The ResetNXBError instruction resets Controller errors in the NX Bus Function Module.

| Instruction   | Name               | FB/<br>FUN | Graphic expression | ST expression  |
|---------------|--------------------|------------|--------------------|--|
| ResetNXBERror | Reset NX Bus Error | FB         |                    | ResetNXBError_instance(Execute, Done, Busy, Error, ErrorID); |



## Precautions for Correct Use

You can use this instruction for the NX102 CPU Unit and NX1P2 CPU Unit.

## Variables

Only common variables are used.

## Function

The ResetNXBError instruction resets the current Controller errors in the NX Bus Function Module.

After an error is reset, the *Busy* output variable changes to FALSE and the *Done* output variable changes to TRUE.

For Safety Control Units, however, errors are not reset.

If this instruction is executed again in a different instance from the instance for which error reset processing is in progress, the latter instruction will result in an error.

The *Error* output variable of the latter command changes to TRUE and the error code (Multi-execution of Instructions: 041A) is output to the *ErrorID* output variable.

## Related System-defined Variables

| Name        | Meaning                             |
|-------------|-------------------------------------|
| _NXB_ErrSta | NX Bus Function Module Error Status |

## Additional Information

- If you execute this instruction on the Simulator, *Done* will change to TRUE, *Busy* and *Error* to FALSE, and *ErrorID* to 0 when *Execute* changes from FALSE to TRUE. And errors will not be cleared.
- If the cause of the error is not solved and the error status remains after resetting an error, the error reset may appear to be impossible.
- The event log is not cleared.

### Precautions for Correct Use

You cannot use this instruction in an event task. A compiling error will occur.

# GetNXBError

The GetNXBError instruction gets the highest level status of the current Controller errors in the NX Bus Function Module of the NX-series CPU Unit.

| Instruction | Name                    | FB/<br>FUN | Graphic expression | ST expression                       |
|-------------|-------------------------|------------|--------------------|-------------------------------------|
| GetNXBError | Get NX Bus Error Status | FUN        |                    | Out:=GetNXBError(UnitProxy, Level); |



## Precautions for Correct Use

You can use this instruction for the NX102 CPU Unit and NX1P2 CPU Unit.

## Variables

|           | Meaning              | I/O    | Description   | Valid range           | Unit | Default |
|-----------|----------------------|--------|---|-----------------------|------|---------|
| Out       | Error flag           | Output | TRUE: Error exists.<br>FALSE: No error  | Depends on data type. | ---  | ---     |
| UnitProxy | Lower level NX Unit  |        | NX Unit where the status of the current error is <i>Level</i>   | ---                   |      |         |
| Level     | Highest level status |        | Status of the current error with the highest level<br>0: No error<br>2: Partial fault level<br>3: Minor fault level | 0, 2, or 3            |      |         |

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| UnitProxy | Refer to <i>Function</i> on page 2-903 for details on the structure <code>_sNXUNIT_ID</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Level     |  |             |      |       |       | OK    |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

## Function

The GetNXBError instruction gets the highest level status (partial fault or minor fault) of the current Controller errors in the NX Bus Function Module.

This instruction applies to the NX102 CPU Unit, NX1P2 CPU Unit and NX Unit connected to the NX bus on the CPU Unit.

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.

*UnitProxy* returns the *UnitProxy* of the NX Unit where the status of the current error is *Level*.

If errors of the same level occur in more than one Unit, it returns *UnitProxy* of the NX Unit that is closest to the master and has the lowest unit number.

The data type of *UnitProxy* is structure `_sNXUNIT_ID`. The specifications are as follows:

| Name       | Meaning                  | Description   | Data type                |
|------------|--------------------------|---|--------------------------|
| UnitProxy  | Lower level NX Unit      | NX Unit where the status of the current error is <i>Level</i> | <code>_sNXUNIT_ID</code> |
| 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 the NX Unit                                    | UDINT                    |
| Path       | Path                     | Path information to the NX Unit                               | BYTE[64]                 |
| PathLength | Valid <i>Path</i> length | Valid <i>Path</i> length                                      | USINT                    |

Pass a `_sNXUNIT_ID` structure variable that is created in the variable table to *UnitProxy*.

## Related System-defined Variables

| Name                                    | Meaning                                    |
|---|--|
| <code>_NXB_ErrSta</code>                | NX Bus Function Module Error Status        |
| <code>_NXB_MstrErrSta</code>            | NX Bus Function Module Master Error Status |
| <code>_NXB_UnitErrSta[1] to [63]</code> | NX Bus Function Module Unit Error Status   |

## Additional Information

If you execute this instruction on the Simulator, this instruction always returns No error, and *Out* changes to FALSE, *Level* changes to 0, and the value of *UnitProxy* is undefined.



# GetNXUnitError

The GetNXUnitError instruction gets the highest level status and highest level event code of the current Controller errors in the NX Bus Function Module of the NX-series CPU Unit or NX Units.

| Instruction    | Name                     | FB/<br>FUN | Graphic expression | ST expression   |
|----------------|--------------------------|------------|--------------------|---|
| GetNXUnitError | Get NX Unit Error Status | FB         |                    | GetNXUnitError_instance(Execute, UnitProxy, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, Level, Code); |



## Precautions for Correct Use

You can use this instruction for the NX102 CPU Unit and NX1P2 CPU Unit.

## Variables

|           | Meaning                  | I/O    | Description   | Valid range                                  | Unit | Default         |
|-----------|--------------------------|--------|---|--|------|-----------------|
| UnitProxy | Specified Unit           | Input  | Specifies the target NX Unit.   | ---  | ---  | *1              |
| TimeOut   | Timeout time             |        | Timeout time<br>If 0 is set, the timeout time is 2.0 s.   | 0 to 60000                                   | ms   | 2000<br>(2.0 s) |
| Level     | Highest level status     | Output | Status of the current error with the highest level<br>0: No error<br>2: Partial fault level<br>3: Minor fault level | 0, 2, or 3                                   | ---  | ---             |
| Code      | Highest level event code |        | Highest level event code of all the current errors  | 16#00000000,<br>16#00070000 to<br>16#FFFFFFF |      |                 |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
| TimeOut   |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
| Level     |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
| Code      |             |             |      | OK    |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |

## Function

The GetNXUnitError instruction outputs the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the NX Bus Function Module and NX Units, as well as the lower level NX Units from which to get the data.

This instruction applies to the NX102 CPU Unit, NX1P2 CPU Unit and NX Unit connected to the NX bus on the CPU Unit.

The Unit from which to get data is specified with *UnitProxy*.

The instruction is completed when the value of *Done* changes to TRUE.

The data type of *UnitProxy* is structure `_sNXUNIT_ID`. The specifications 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 the specified Unit               | UDINT       |
| Path       | Path                     | Path information to the specified Unit          | BYTE[64]    |
| PathLength | Valid <i>Path</i> length | Valid <i>Path</i> length                        | USINT       |

Pass a device variable that is assigned to the specified Unit to *UnitProxy*.

*TimeOut* specifies the timeout time. If a response does not return within the timeout time, it judges that communications failed.

*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. If errors of the same level occur in more than one Unit, it gives the oldest event code. If there is no error, it gives 16#00000000.

## Specified NX Unit and Variable Values

The values output to in-out and output variables vary depending on the specified NX Unit.

The relationship between the specified NX Unit and the value of each variable is shown in the following table.

| Specified Unit        | NX102 CPU Unit<br>NX1P2 CPU Unit  | NX Unit on CPU Unit             |
|-----------------------|---|---------------------------------|
| Value of <i>Level</i> | Highest level status of CPU Unit  | Highest level status of NX Unit |
| Value of <i>Code</i>  | If there are more than one error: Highest level event code<br>If there are more than one error in the same <i>level</i> : Oldest event code<br>If there is no error: 16#0000_0000 |                                 |

## Combined Use with the GetNXBError Instruction

In the user program, normally use the GetNXBError (Get NX Bus Error Status) instruction to monitor if there are errors on the NX bus.

When the *Level* output variable of the GetNXBError instruction is other than 0, the value that indicates the NX Unit with the highest event level is stored in *UnitProxy* of the instruction.

To get *Level* and *Code* of the NX Unit where the error occurred, execute the GetNXUnitError instruction.

### Related System-defined Variables

| Name                | Meaning              |
|---------------------|----------------------|
| _NXB_UnitErrFlagTbl | NX Unit Error Status |

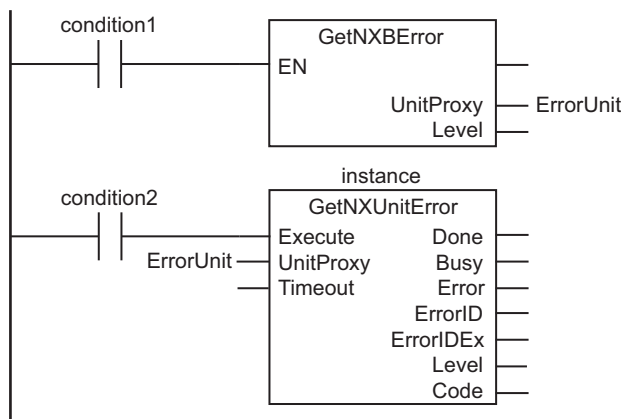
### Additional Information

#### Parameters to Be Passed to *UnitProxy*

The following explains parameters to be passed to the *UnitProxy* input variable.

- **Using Only the User Program to Pass Parameters to *UnitProxy***

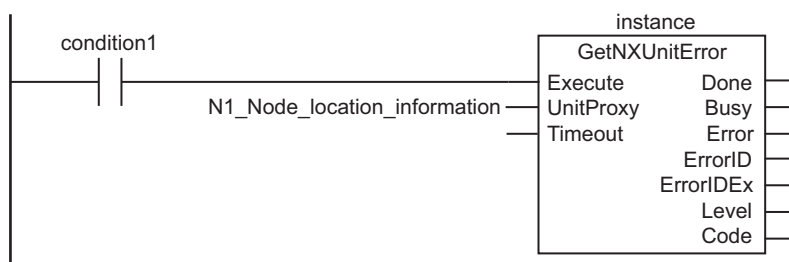
To *UnitProxy* of the GetNXUnitError instruction, pass the value of *ErrorUnit*; it contains the value of *UnitProxy* you get with the GetNXBError instruction sent to the NX Bus Function Module.



- **Using the Device Variable to Pass Parameters to *UnitProxy***

Create a device variable that specifies the Unit on the NX bus and pass it to *UnitProxy* of this instruction.

| External variables | Variable                     | Data type   | Constant                            | Comment |
|--------------------|------------------------------|-------------|-------------------------------------|---------|
|                    | N1_Node_location_information | _sNXUNIT_ID | <input checked="" type="checkbox"/> |         |



## Run in the Simulator

When this instruction is executed and changes the value of *Execute* from FALSE to TRUE, the values for the related variables will change as shown below.

| Output variable | Meaning |
|-----------------|---------|
| Done            | TRUE    |
| Busy            | FALSE   |
| Error           | FALSE   |
| ErrorID         | 0       |
| ErrorIDEx       | 0       |
| Level           | 0       |
| Code            | 0       |

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when 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 the processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- You cannot execute this instruction more than once at the same time. Only one instance can be executed at a time.
- *Error* will change to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i> | Description   |
|-------------------------|---------------------------|---|
| 16#0400                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The value of <i>UnitProxy</i> is outside the valid range.</li> <li>• The value of <i>TimeOut</i> is outside the valid range.</li> </ul>    |
| 16#041A                 | 16#00000000               | While this instruction was being executed, it was executed again.   |
| 16#2C00                 | 16#00000401               | The specified Unit does not support the instruction.  |
|                         | 16#00001001               | 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#00001002               |   |
|                         | 16#00170000               |   |
|                         | 16#00200000               |   |
|                         | 16#00210000               |   |
|                         | 16#00001010               | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .   |
|                         | 16#00001101               | The Unit is not correct.<br>Check the Unit.   |
|                         | 16#0000110B               | The size of the read data is too large.<br>Make sure that the read data specification is correct.   |
|                         | 16#00001110               | There is no object that corresponds to the value of <i>Obj.Index</i> .  |
|                         | 16#00001111               | There is no object that corresponds to the value of <i>Obj.Subindex</i> .   |
|                         | 16#00002101               | The specified NX object cannot be written.  |
|                         | 16#00002110               | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.  |
|                         | 16#00002210               | The specified Unit is not in a mode that allows writing data.   |

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Description  |
|-------------------------|--|--|
|                         | 16#00002213  | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
|                         | 16#00002230  | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
|                         | 16#00002231  | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction.  |
|                         | 16#0000250F  | Hardware access failed.<br>Execute the instruction again.  |
|                         | 16#00002601<br>16#00002602<br>16#00100000  | The specified Unit does not support this instruction.<br>Check the version of the Unit.  |
|                         | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.   |
|                         | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.  |
|                         | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.  |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.  |
|                         | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.  |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.   |

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Description   |
|-------------------------|--|---|
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000                               | An error occurred in the communications network.<br>Reduce the amount of communications traffic.  |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000 | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON. |
| 16#2C02                 | 16#00000000  | A timeout occurred during communications.   |

## Sample Programming

This sample passes the Unit, level, and code values to variables for display on a touch panel if an error occurs on the NX bus.

The system configuration is as shown below.

Assume that three NX Units are connected to an NX1P2 CPU Unit.

## Device Variables

| NX Units    | Device Variable              |
|-------------|------------------------------|
| 1st NX Unit | N1_Node_location_information |
| 2nd NX Unit | N2_Node_location_information |
| 3rd NX Unit | N3_Node_location_information |

## Definitions of Global Variables

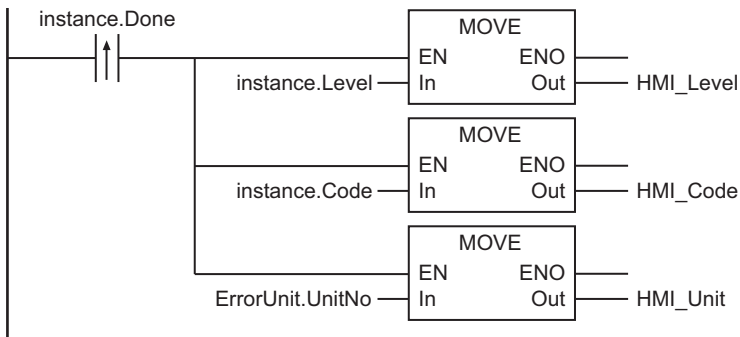
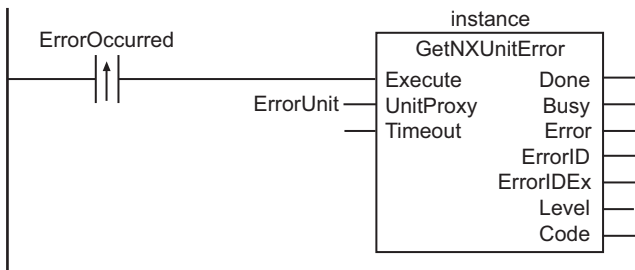
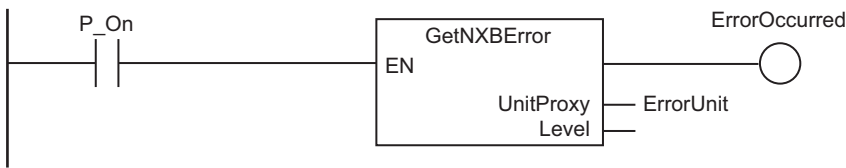
### ● Global Variables

| Variable                     | Data type   | Constant                            | Comment |
|------------------------------|-------------|-------------------------------------|---------|
| N1_Node_location_information | _sNXUNIT_ID | <input checked="" type="checkbox"/> |         |
| N2_Node_location_information | _sNXUNIT_ID | <input checked="" type="checkbox"/> |         |
| N3_Node_location_information | _sNXUNIT_ID | <input checked="" type="checkbox"/> |         |
| HMI_Level *1                 | UINT        | <input type="checkbox"/>            |         |
| HMI_Code *1                  | DWORD       | <input type="checkbox"/>            |         |
| HMI_Unit *1                  | UDINT       | <input type="checkbox"/>            |         |

\*1. The variables that begin with *HMI\_* are variables for display on a touch panel.

**LD**

| Internal variables | Variable      | Data type      | Constant | Comment |
|--------------------|---------------|----------------|----------|---------|
|                    | instance      | GetNXUnitError | ☐        |         |
|                    | ErrorOccurred | BOOL           | ☐        |         |
|                    | ErrorUnit     | _sNXUNIT_ID    | ☐        |         |



# SetInfo

The SetInfo instruction creates user-defined information.

| Instruction | Name                            | FB/<br>FUN | Graphic expression | ST expression                |
|-------------|---------------------------------|------------|--------------------|------------------------------|
| SetInfo     | Create User-defined Information | FUN        |                    | SetInfo(Code, Info1, Info2); |

## Variables

|       | 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. |      | *1      |
| Info2 | Attached information 2 |        |   |                       |      |         |
| 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.

|       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |

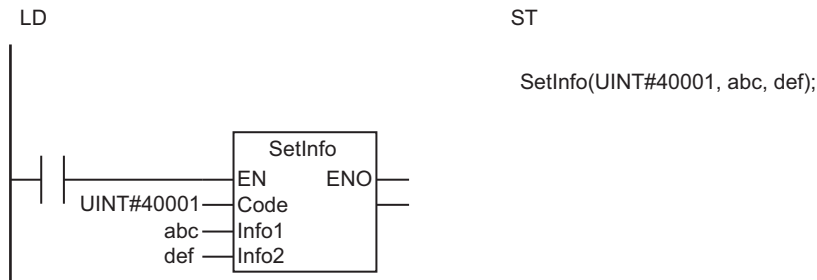
## Function

The SetInfo instruction generates the user-defined information specified by event code *Code*.

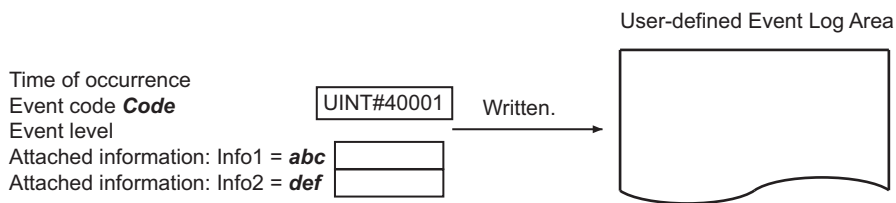
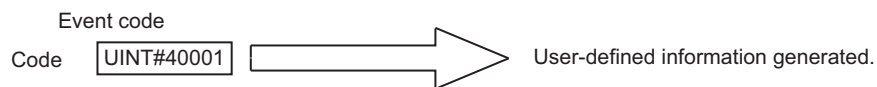
The time of occurrence, event code *Code*, event level, attached information *Info1*, and attached information *Info2* are stored in the user event log area that corresponds to the level of the event code.

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.





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 will occur in the following case. *ENO* will be FALSE.
  - a) The value of *Code* is outside the valid range.

# ResetUnit

The ResetUnit instruction restarts a CPU Bus Unit or Special I/O Unit.

| Instruction | Name         | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------|------------|--------------------|--|
| ResetUnit   | Restart Unit | FB         |                    | ResetUnit_instance(Execute, UnitNo, Done, Busy, Error, ErrorID); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units.

## Variables

|        | Meaning     | I/O   | Description                    | Valid range                                    | Unit | Default   |
|--------|-------------|-------|--------------------------------|--|------|-----------|
| UnitNo | Unit number | Input | Unit number of Unit to restart | _CBU_No00 to _CBU_No15, _SIO_No00 to _SIO_No95 | ---  | _CBU_No00 |

|        | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-914 for the enumerators of enumeration type _eUnitNo. |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |

## Function

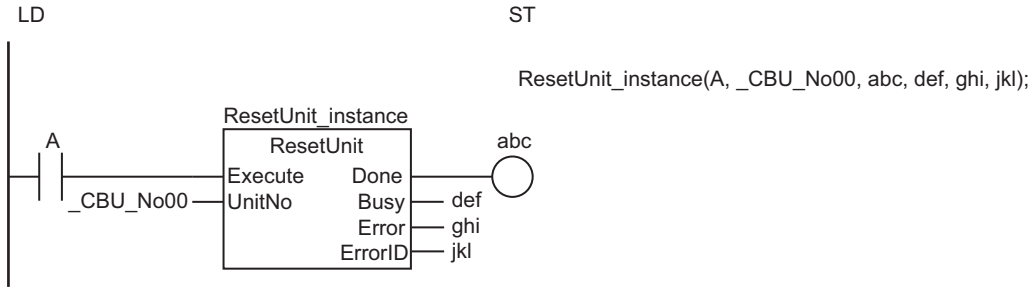
The ResetUnit instruction restarts a CPU Bus Unit or Special I/O Unit.

Specify *UnitNo* for the Unit to be restarted.

The data type of *UnitNo* is enumerated type \_eUnitNo. The meanings of the enumerators are as follows:

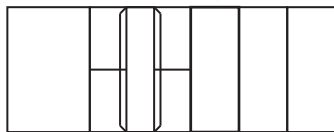
| Enumerators            | Meaning                                   |
|------------------------|---|
| _CBU_No00 to _CBU_No15 | Unit number of CPU Bus Unit, 00 to 15     |
| _SIO_No00 to _SIO_No95 | Unit number of Special I/O Unit, 00 to 95 |

The following example is for when *UnitNo* is \_CBU\_No00. CPU Bus Unit with unit number 0 is restarted.



The ResetUnit instruction restarts the CPU Bus Unit or Special I/O Unit with **UnitNo**.

**UnitNo** = \_CBU\_No00: CPU Bus Unit with unit number 0 is restarted.



### Precautions for Correct Use

- This instruction will not end in an error even if restart processing is in progress for the Unit specified by **UnitNo**. The value of **Busy** remains TRUE, and the value of **Done** changes to TRUE when restart processing is finished. Restart requests are not queued.
- The Unit is restarted if the value of **Execute** is TRUE when operation starts.
- An error will occur in the following cases. **Error** will change to TRUE.
  - a) The value of **UnitNo** is outside the valid range.
  - b) The Unit specified with **UnitNo** does not exist.
  - c) Restart processing failed.

### Sample Programming

When the value of **Trigger** changes to TRUE, the baud rate of serial port 1 on the Serial Communications Unit with a unit number of 0 is set to 38,400 bps and the Unit is restarted.

## Definitions of Global Variables

### ● Global Variables

| Name               | Data type | Initial value | AT specification *1                  | Retain                              | Comment   |
|--------------------|-----------|---------------|--------------------------------------|-------------------------------------|-----------|
| SCU_P1_BaudrateCfg | USINT     | 0             | IOBus://rack#0/slot#0/P1_BaudrateCfg | <input checked="" type="checkbox"/> | Baud rate |

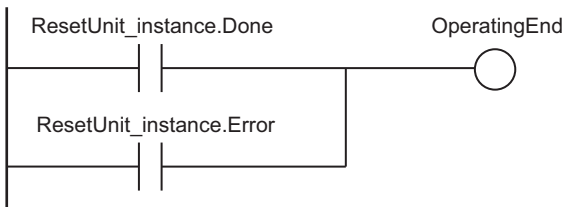
\*1. AT when the Serial Communications Unit is mounted to slot number 0 in rack number 0

# LD

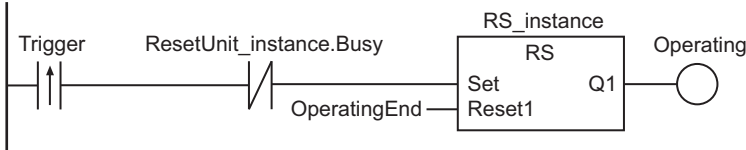
| Internal Variables | Variable           | Data type | Initial value | Comment              |
|--------------------|--------------------|-----------|---------------|----------------------|
|                    | OperatingEnd       | BOOL      | FALSE         | Processing completed |
|                    | Trigger            | BOOL      | FALSE         | Execution condition  |
|                    | Operating          | BOOL      | FALSE         | Processing           |
|                    | RS_instance        | RS        |               |                      |
|                    | ResetUnit_instance | ResetUnit |               |                      |

| External Variables | Variable           | Data type | Comment   |
|--------------------|--------------------|-----------|-----------|
|                    | SCU_P1_BaudrateCfg | USINT     | Baud rate |

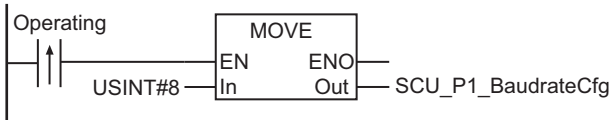
Determine if execution of the ResetUnit has ended.



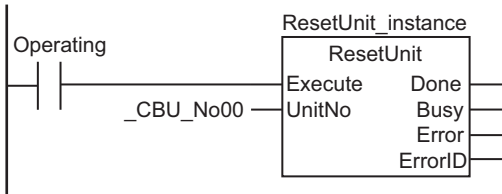
Accept trigger.



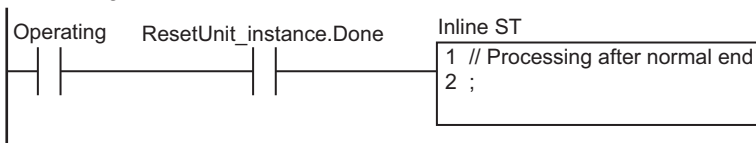
Set baud rate in device variable.

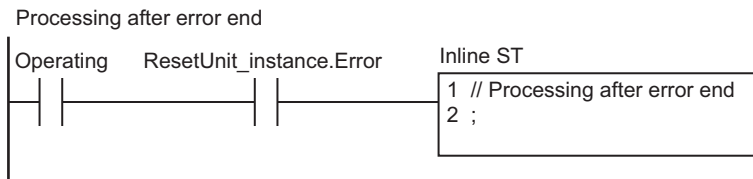


Execute ResetUnit instruction.



Processing after normal end



**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                                 |
|                    | ResetUnit_instance | ResetUnit |               |  |

| External Variables | Variable           | Data type | Comment   |
|--------------------|--------------------|-----------|-----------|
|                    | SCU_P1_BaudrateCfg | USINT     | Baud rate |

```

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (ResetUnit_instance.Busy=FALSE) ) T
HEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize ResetUnit_instance and set baud rate in device variable.
IF (OperatingStart=TRUE) THEN
    ResetUnit_instance(Execute:=FALSE);
    SCU_P1_BaudrateCfg:=USINT#8;
    OperatingStart:=FALSE;
END_IF;

// Execute ResetUnit instruction.
IF (Operating=TRUE) THEN
    ResetUnit_instance(
        Execute:=TRUE, // Execution condition
        UnitNo :=_CBU_No00); // Unit number

    IF (ResetUnit_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;
  
```

```
IF (ResetUnit_instance.Error=TRUE) THEN
  // Processing after error end
  Operating:=FALSE;
END_IF;
END_IF;
```

# GetNTPStatus

The GetNTPStatus instruction reads the NTP status.

| Instruction  | Name            | FB/<br>FUN | Graphic expression | ST expression                       |
|--------------|-----------------|------------|--------------------|-------------------------------------|
| GetNTPStatus | Read NTP Status | FUN        |                    | GetNTPStatus(ExecTime, ExecNormal); |

## Variables

|            | Meaning                        | I/O    | Description                          | Valid range           | Unit                                     | Default |
|------------|--------------------------------|--------|--------------------------------------|-----------------------|--|---------|
| Out        | Return value                   | Output | Always TRUE                          | TRUE only             | ---                                      | ---     |
| ExecTime   | NTP last normal operation time |        | NTP last normal operation time       | Depends on data type. | Year, month, day, hour, minutes, seconds |         |
| ExecNormal | NTP normal end flag            |        | TRUE: Normal end<br>FALSE: Error end |                       | ---                                      |         |

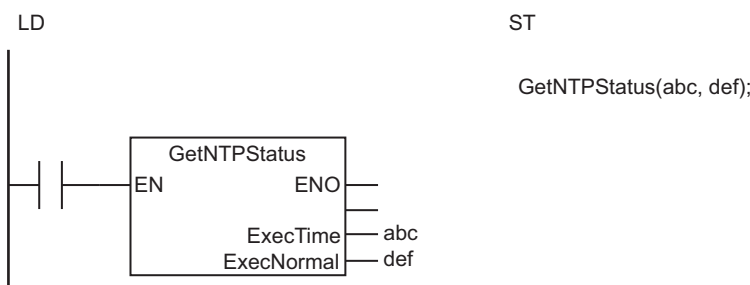
|            | Boolean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|------------|---------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|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

## Function

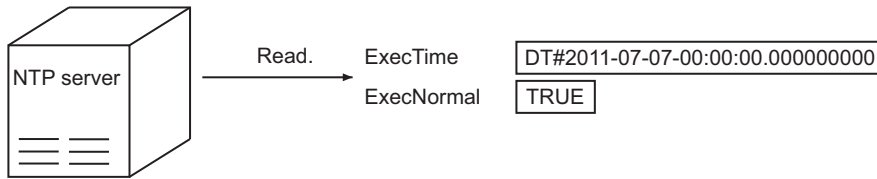
The GetNTPStatus instruction reads the NTP status.

The following information is read: *ExecTime* (NTP last normal operation time) and *ExecNormal* (NTP normal end flag).

The following figure shows a programming example.



The GetNTPStatus instruction reads the NTP status.  
The values of **ExecTime** and **ExecNormal** will be as follows when the last normal NTP operation was at 00:00.00 on July 7, 2011.



### Related System-defined Variables

| Name           | Meaning    | Data type    | Description                |
|----------------|------------|--------------|----------------------------|
| _EIP_NTPResult | NTP Status | _sNTP_RESULT | Contains the NTP status.*1 |

\*1. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

### Precautions for Correct Use

- Return value *Out* is not used when this instruction is used in ST.
- This instruction reads the contents of the `_EIP_NTPResult` system-defined variable. You cannot access this variable directly. Always use this instruction to read the contents of the variable.



# RestartNXUnit

The RestartNXUnit instruction restarts an EtherCAT Coupler Unit or NX Units.

| Instruction   | Name             | FB/<br>FUN | Graphic expression | ST expression  |
|---------------|------------------|------------|--------------------|--|
| RestartNXUnit | Restart NX Units | FB         |                    | RestartNXUnit_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx); |



## 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 the function of *restarting specified NX Units*.

- CPU Units
- Sysmac Studio
- EtherCAT Coupler Units
- NX Units

If the unit version of a product does not support the function of *restarting specified NX Units*, 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 the function of *restarting specified NX Units*.

## Variables

|           | 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 | ---         | ---  | *1      |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-921 for details on the structure <code>_sNXUNIT_ID</code> . |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |

## 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*.

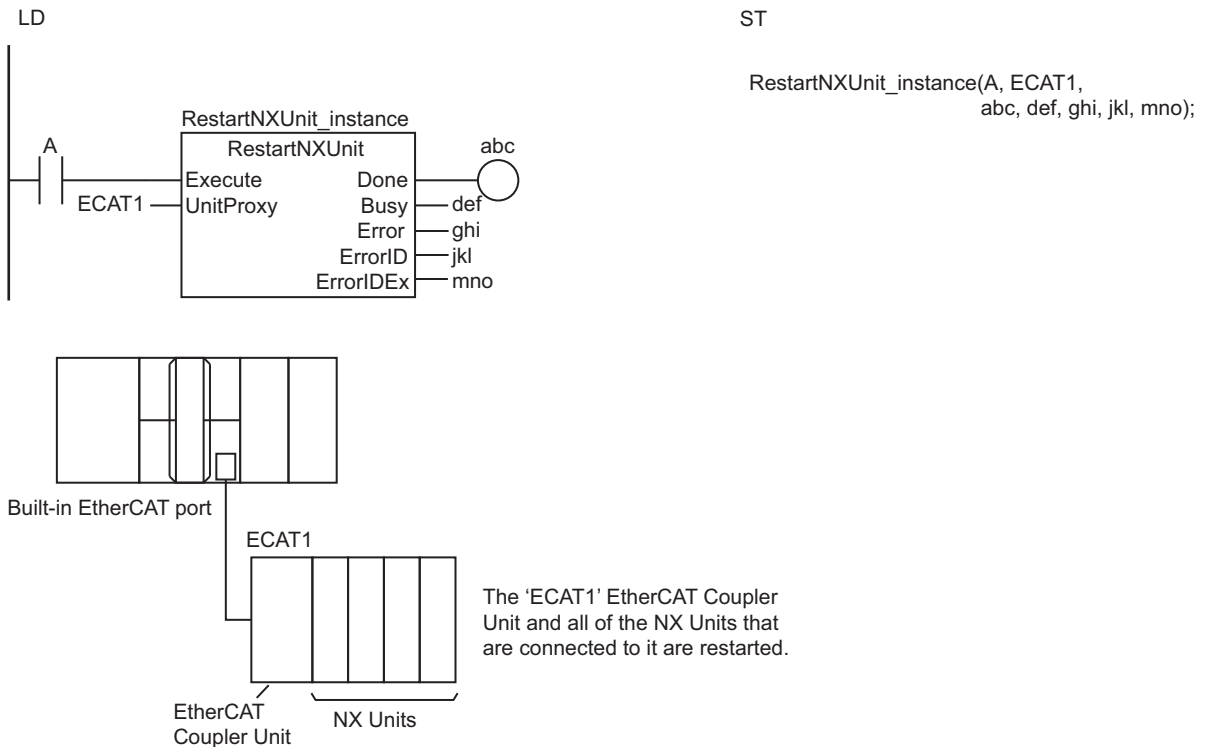
The data type of *UnitProxy* is structure `_sNXUNIT_ID`. The meanings of the members are as follows:

| Name       | Meaning                  | Content   | Data type                |
|------------|--------------------------|---|--------------------------|
| UnitProxy  | Specified Unit           | Specified Unit                                  | <code>_sNXUNIT_ID</code> |
| 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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT                    |

To *UnitProxy*, pass the 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.

## 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.



## Related System-defined Variables

| Name   | Meaning                                    | Data type | Description  |
|--|--|-----------|--|
| <code>_EC_MBXSlaVtBl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible.       |
| <code>_NXB_UnitMsgActiveTb[i]</code>                       | NX Unit Message Enabled Status             | BOOL      | This table indicates the slaves that can perform message communications.<br>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 instruction, *NX\_ChangeWriteMode* on page 2-927, to change the Unit to a mode that allows writing data.
- 2** Use the instruction, *NX\_WriteObj* on page 2-1037, to write data to the Unit.
- 3** Use the instruction, *NX\_SaveParam* on page 2-933, 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 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 the execution.
- Refer to *Using this Section* on 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 instruction, *ResetMCErr* on page 2-888, to reset the Controller error.
- For *UnitProxy*, specify the device variable that is assigned to an 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)* for details on assigning device variables.
- If an attempt is made to execute the *RestartNXUnit* instruction during execution of another *RestartNXUnit* instruction or the *NX\_ChangeWriteMode* on page 2-927 instruction, it 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 during which 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-4 Instructions Related to NX Message Communications Errors* on page A-35 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* changes to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>   | Meaning  |
|-------------------------|---|--|
| 16#0419                 | 16#00000000   | The data type of <i>UnitProxy</i> is not correct.  |
| 16#2C00                 | 16#00000401   | The specified Unit does not support the instruction.   |
|                         | 16#00001001   | 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#00001002   |  |
|                         | 16#00170000   |  |
|                         | 16#00200000   |  |
|                         | 16#00210000   |  |
|                         | 16#00001010   | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
|                         | 16#00001101   | The Unit is not correct.<br>Check the Unit.  |
|                         | 16#0000110B   | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
|                         | 16#00001110   | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
|                         | 16#00001111   | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
|                         | 16#00002101   | The specified NX object cannot be written.   |
|                         | 16#00002110   | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
|                         | 16#00002210   | The specified Unit is not in a mode that allows writing data.  |
|                         | 16#00002213   | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
|                         | 16#00002230   | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
| 16#00002231             | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction. |  |
| 16#0000250F             | Hardware access failed.<br>Execute the instruction again.   |  |
| 16#00002601             | The specified Unit does not support this instruction.<br>Check the version of the Unit.   |  |
| 16#00002602             |   |  |
| 16#00100000             |   |  |

| Value of ErrorID | Value of ErrorIDEx   | Meaning   |
|------------------|--|---|
|                  | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.                        |
|                  | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.   |
|                  | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.   |
|                  | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.  |
|                  | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.   |
|                  | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.   |
|                  | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.   |
|                  | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.   |
|                  | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.  |
|                  | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000   | An error occurred in the communications network.<br>Reduce the amount of communications traffic.  |
|                  | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000   | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON.   |
| 16#2C01          | 16#00000000  | An attempt was made to queue more than 192 RestartNXUnit and NX_ChangeWriteMode instructions.   |
| 16#2C02          | 16#00000000  | A timeout occurred during communications.   |
| 16#2C05          | ---  | An error occurred in the EtherCAT network.<br>Check the value of <i>UnitProxy</i> 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#00000000  | A slave that cannot be specified for the instruction was connected at the slave node address of the specified Unit. Check the value of <i>UnitProxy</i> and the EtherCAT network configuration. |

## Sample Programming

Refer to *Sample Programming* on page 2-1042 for the NX\_WriteObj instruction.

# NX\_ChangeWriteMode

The NX\_ChangeWriteMode instruction changes an EtherCAT Coupler Unit or NX Unit to a mode that allows writing data.

| Instruction             | Name                            | FB/<br>FUN | Graphic expression | ST expression   |
|-------------------------|---------------------------------|------------|--------------------|---|
| NX_Change-<br>WriteMode | Change to NX<br>Unit Write Mode | FB         |                    | NX_ChangeWriteMode_in-<br>stance(Execute, UnitProxy, Done,<br>Busy, Error, ErrorID, ErrorIDEx); |



## 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.

## Variables

|           | Meaning        | I/O   | Description                          | Valid range | Unit | Default |
|-----------|----------------|-------|--------------------------------------|-------------|------|---------|
| UnitProxy | Specified Unit | Input | Unit for which to<br>change the mode | ---         | ---  | *1      |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-927 for details on the structure <code>_sNXUNIT_ID</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

## 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 so that data can be written to the Unit.

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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT       |

To *UnitProxy*, pass the device variable that is assigned to the specified Unit.

## 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.

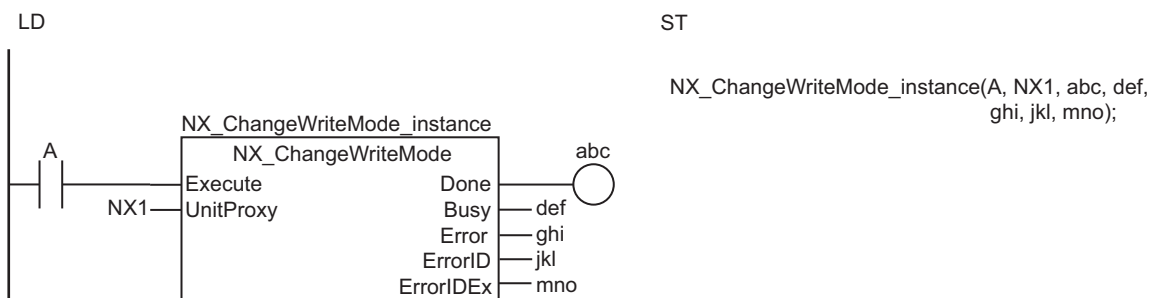
- Power OFF Retain attribute
- The values are updated when the Unit is restarted.

Use the following procedure to execute the related instructions.

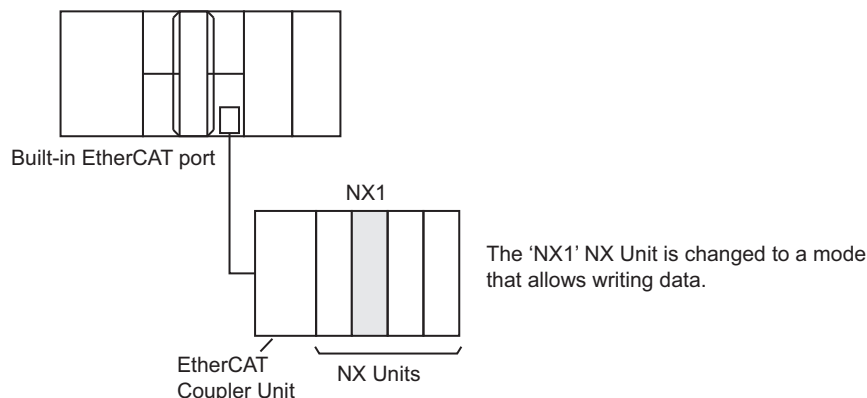
- 1** Use the *NX\_ChangeWriteMode* instruction to change the Units to a mode that allows writing data.
- 2** Use the instruction, *NX\_WriteObj* on page 2-1037, to write data to the Unit.
- 3** Use the instruction, *NX\_SaveParam* on page 2-933, to save the data that you wrote.
- 4** Use the instruction, *RestartNXUnit* on page 2-921, 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.







## Related System-defined Variables

| Name   | Meaning                                    | Data type | Description  |
|--|--|-----------|--|
| <code>_EC_MBXSlavTbl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible.       |
| <code>_NXB_UnitMsgActiveTbl[i]</code>                      | NX Unit Message Enabled Status             | BOOL      | This table indicates the slaves that can perform message communications.<br>Use this variable to confirm that communications with the relevant slave are possible. |

## Precautions for Correct Use

- Execution of this instruction is continued until 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 the execution.
- Refer to *Using this Section* on 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 instruction, *ResetMCError* on page 2-888, to reset the Controller error.
- For *UnitProxy*, specify the device variable that is assigned 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 in the I/O Map of the Sysmac Studio.  
Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details on assigning device variables.
- If an attempt is made to execute the `NX_ChangeWriteMode` instruction during execution of another `NX_ChangeWriteMode` instruction or the *RestartNXUnit* on page 2-921 instruction, it 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 during which 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-4 Instructions Related to NX Message Communications*

Errors on page A-35 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* changes to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>   | Meaning  |
|-------------------------|---|--|
| 16#0419                 | 16#00000000   | The data type of <i>UnitProxy</i> is not correct.  |
| 16#2C00                 | 16#00000401   | The specified Unit does not support the instruction.   |
|                         | 16#00001001<br>16#00001002<br>16#00170000<br>16#00200000<br>16#00210000 | An input parameter, output parameter, or in-out parameter is incorrect.<br>Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.   |
|                         | 16#00001010   | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
|                         | 16#00001101   | The correct Unit was not specified.<br>Check the Unit.   |
|                         | 16#0000110B   | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
|                         | 16#00001110   | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
|                         | 16#00001111   | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
|                         | 16#00002101   | The specified NX object cannot be written.   |
|                         | 16#00002110   | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
|                         | 16#00002210   | The specified Unit is not in a mode that allows writing data.  |
|                         | 16#00002213   | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
|                         | 16#00002230   | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
|                         | 16#00002231   | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction.  |
|                         | 16#0000250F   | Hardware access failed.<br>Execute the instruction again.  |
|                         | 16#00002601<br>16#00002602<br>16#00100000                               | The specified Unit does not support this instruction.<br>Check the version of the Unit.  |

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning  |
|-------------------------|--|--|
|                         | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.                           |
|                         | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.  |
|                         | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.  |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.  |
|                         | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.  |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.   |
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000   | An error occurred in the communications network.<br>Reduce the amount of communications traffic.   |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000   | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON.  |
| 16#2C01                 | 16#00000000  | An attempt was made to queue more than 192 NX_ChangeWriteMode and RestartNXUnit instructions.  |
| 16#2C02                 | 16#00000000  | A timeout occurred during communications.  |
| 16#2C05                 | ---  | An error occurred in the EtherCAT network.<br>Check the value of <i>UnitProxy</i> and the EtherCAT network configuration.  |
| 16#2C07                 | 16#00000000  | A slave that cannot be specified for the instruction was connected at the slave node address of the specified Unit.<br>Check the value of <i>UnitProxy</i> and the EtherCAT network configuration. |

## Sample Programming

Refer to *Sample Programming* on page 2-1042 for the NX\_WriteObj instruction.

# NX\_SaveParam

The NX\_SaveParam instruction saves the data that was written to an EtherCAT Coupler Unit or NX Unit.

| Instruction  | Name                    | FB/<br>FUN | Graphic expression | ST expression  |
|--------------|-------------------------|------------|--------------------|--|
| NX_SaveParam | Save NX Unit Parameters | FB         |                    | NX_SaveParam_instance(Execute, UnitProxy, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx); |



## 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.

## Variables

|           | Meaning        | I/O   | Description   | Valid range | Unit | Default         |
|-----------|----------------|-------|---|-------------|------|-----------------|
| UnitProxy | Specified Unit | Input | Unit for which to save data                             | ---         | ---  | *1              |
| TimeOut   | Timeout time   |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 0 to 60,000 | ms   | 2000<br>(2.0 s) |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |  |
| 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 instruction, *NX\_WriteObj* on page 2-1037, 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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT       |

To *UnitProxy*, pass the device variable that is assigned to the specified Unit.

## 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 related instructions.

The procedure for each case is 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 instruction, *NX\_ChangeWriteMode* on page 2-927, to change the Unit to a mode that allows writing data.
- 2** Use the instruction, *NX\_WriteObj* on page 2-1037, to write data to the Unit.
- 3** Use the *NX\_SaveParam* instruction to save the data that you wrote.
- 4** Use the instruction, *RestartNXUnit* on page 2-921, to restart the Unit.

### ● Execution Procedure 2

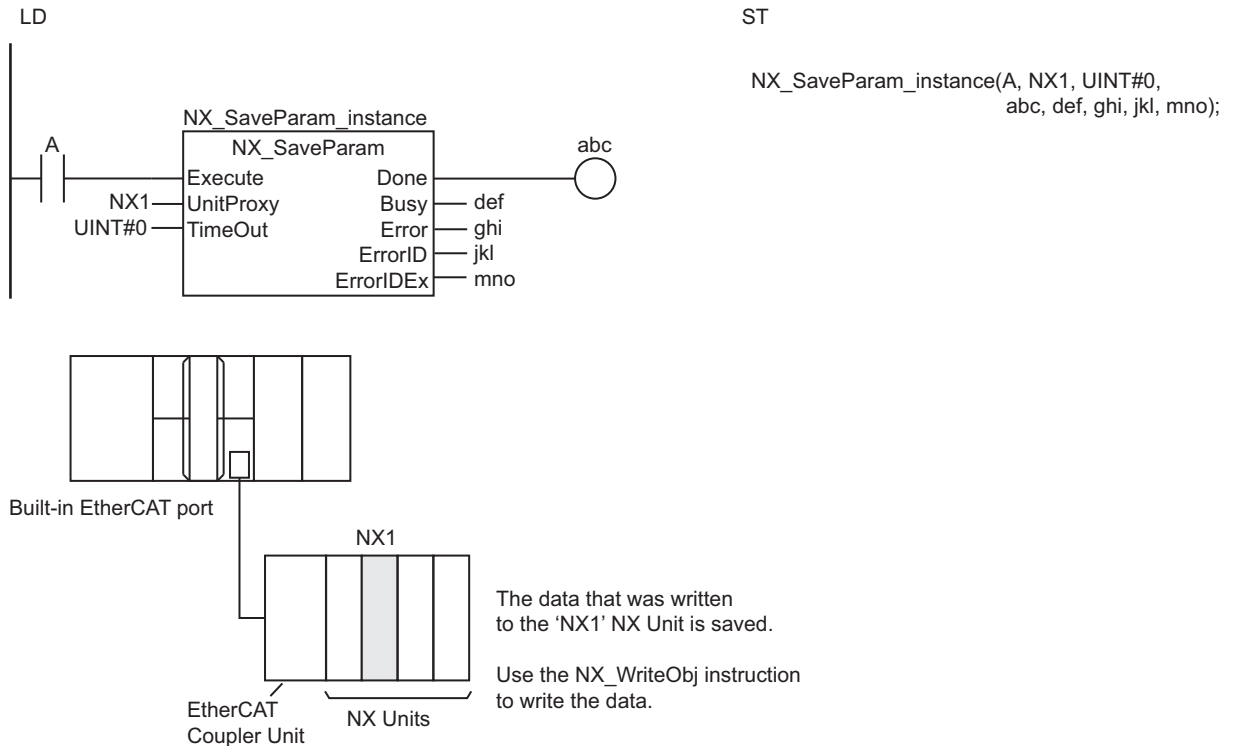
Use the following procedure to write data with the following attributes.

- Power OFF Retain attribute
- The values are updated as soon as they are written.

- 1** Use the instruction, *NX\_WriteObj* on page 2-1037, 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.



## Related System-defined Variables

| Name  | Meaning                                    | Data type | Description  |
|---|--|-----------|--|
| <code>_EC_MBxSlavTb[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |

## Precautions for Correct Use

- Execution of this instruction is continued until 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 the execution.
- Refer to *Using this Section* on 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 TRUE, and the value of *Done* changes to TRUE when the data saving 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 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 in the I/O Map of the Sysmac Studio.  
Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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 *NX\_WriteObj* on page 2-1037. 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-4 Instructions Related to NX Message Communications Errors* on page A-35 for a list of the instructions that are related to NX Message Communications Errors.
- *Error* changes to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>   | Meaning  |
|-------------------------|---|--|
| 16#0400                 | 16#00000000   | <ul style="list-style-type: none"> <li>• The value of <i>UnitProxy</i> is outside the valid range.</li> <li>• The value of <i>TimeOut</i> is outside the valid range.</li> </ul>   |
| 16#0419                 | 16#00000000   | The data type of <i>UnitProxy</i> is not correct.  |
| 16#2C00                 | 16#00000401   | The specified Unit does not support the instruction.   |
|                         | 16#00001001<br>16#00001002<br>16#00170000<br>16#00200000<br>16#00210000 | An input parameter, output parameter, or in-out parameter is incorrect.<br>Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.   |
|                         | 16#00001010   | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
|                         | 16#00001101   | The correct Unit was not specified.<br>Check the Unit.   |
|                         | 16#0000110B   | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
|                         | 16#00001110   | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
|                         | 16#00001111   | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
|                         | 16#00002101   | The specified NX object cannot be written.   |
|                         | 16#00002110   | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
|                         | 16#00002210   | The specified Unit is not in a mode that allows writing data.  |
|                         | 16#00002213   | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
|                         | 16#00002230   | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |



| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning   |
|-------------------------|--|---|
|                         | 16#00002231  | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction. |
|                         | 16#0000250F  | Hardware access failed.<br>Execute the instruction again.   |
|                         | 16#00002601<br>16#00002602<br>16#00100000  | The specified Unit does not support this instruction.<br>Check the version of the Unit.   |
|                         | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.          |
|                         | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.   |
|                         | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.   |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.   |
|                         | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.   |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.  |
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000   | An error occurred in the communications network.<br>Reduce the amount of communications traffic.  |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000   | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON.                                   |
| 16#2C01                 | 16#00000000  | The number of instructions that can be simultaneously executed was exceeded.  |

| <b>Value of <i>ErrorID</i></b> | <b>Value of <i>ErrorIDEx</i></b> | <b>Meaning</b>                            |
|--------------------------------|----------------------------------|---|
| 16#2C02                        | 16#00000000                      | A timeout occurred during communications. |

### Sample Programming

Refer to *Sample Programming* on page 2-1042 for the NX\_WriteObj instruction.

# PLC\_ReadTotalPowerOnTime

The PLC\_ReadTotalPowerOnTime instruction reads the total power ON time from a specified CPU Unit.

| Instruction              | Name                         | FB/<br>FUN | Graphic expression | ST expression                            |
|--------------------------|------------------------------|------------|--------------------|--|
| PLC_ReadTotalPowerOnTime | Read PLC Total Power ON Time | FUN        |                    | Out:=PLC_ReadTotalPowerOnTime(UnitType); |



## Precautions for Correct Use

You can use this instruction for the NX102 CPU Unit and NX1P2 CPU Unit.

## Variables

|          | Meaning             | I/O    | Description                                    | Valid range           | Unit | Default   |
|----------|---------------------|--------|--|-----------------------|------|-----------|
| UnitType | Unit type           | Input  | Specifies the target Unit.                     | _CPU_UNIT             | ---  | _CPU_UNIT |
| Out      | Total Power ON Time | Output | Outputs the total power ON time that was read. | Depends on data type. | ---  | ---       |

|          | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |  |
|----------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|--|
|          | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT         | REAL | LREAL                                     | TIME | DATE | TOD | DT | STRING |  |
| UnitType |             |             |      |       |       |       |          |       |       |      |     |      |              |      |   |      |      |     |    |        | Refer to <i>Function</i> on page 2-939 for the enumerators of the enumerated type <code>_ePLC_UNIT_TYPE</code> . |
| Out      |             |             |      |       |       |       |          |       |       |      |     |      |              |      |   | OK   |      |     |    |        |  |

## Function

The PLC\_ReadTotalPowerOnTime instruction reads the approximate total power ON time from a specified CPU Unit.

The accuracy is 1 hour per month.

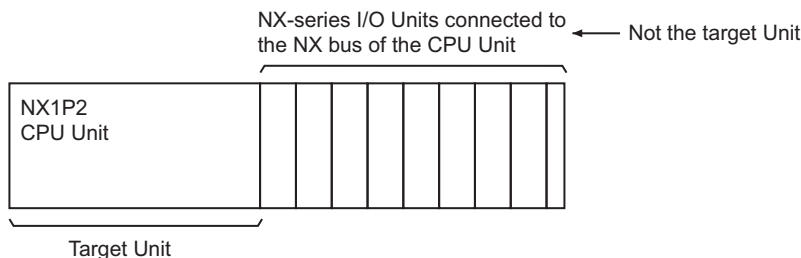
Specify *UnitType* for the Unit from which the data is read.

The data type of *UnitType* is enumerated type `_ePLC_UNIT_TYPE`. The meaning of the enumerator is as follows:

| Enumerator             | Meaning                |
|------------------------|------------------------|
| <code>_CPU_UNIT</code> | CPU Unit is specified. |

## Target Unit

### ● When Using the NX1P2 CPU Unit



The target Unit for reading is the NX1P2 CPU Unit only.

Set *UnitType* with the `_CPU_UNIT` enumerator of the enumerated type `_ePLC_UNIT_TYPE` to specify a target Unit.

NX-series I/O Units connected to the NX bus of the CPU Unit are not the target Unit for reading.



#### Additional Information

To read the total power ON time for an NX-series I/O Unit, use `NX_ReadTotalPowerOnTime` on page 2-942.

## Additional Information

If you execute this instruction in the Simulator, the value of *Out* is always T#0s.

## Precautions for Correct Use

If the specified Unit is not the target for reading, an error will occur.  
`ENO` will be FALSE, and the value of *Out* will not change.

## Sample Programming

This sample creates the two variables `Maintenance_Mode` and `Run_Mode` in the program.

In the two, the instruction reads the total power ON time of the NX1P2 CPU Unit when the button to read the total power ON time is pressed in `Maintenance_Mode`.

If the total power ON time exceeds 5 years, the Unit replacement warning lamp lights.

Once the Unit is replaced and the button for completion of Unit replacement is pressed, the Unit replacement warning lamp turns off.

The system configuration is as shown below.

| Unit   | Description  |
|--------|--|
| Unit 1 | Unit 1, which is connected to the NX bus of the NX-series CPU Unit<br>NX-series I/O Unit ( ID) |
| Unit 2 | Unit 2, which is connected to the NX bus of the NX-series CPU Unit<br>NX-series I/O Unit (IO)  |
| Unit 3 | NX1P2 CPU Unit (Target for reading the total power ON time)                                    |

## ST

| External Variables | Variable                  | Data type | Comment                            |
|--------------------|---------------------------|-----------|------------------------------------|
|                    | J01_Ch1_In00              | BOOL      | Maintenance mode button            |
|                    | J01_Ch1_In01              | BOOL      | Button to read total power ON time |
|                    | J02_Ch1_Out00             | BOOL      | Unit replacement warning lamp      |
|                    | Maintenance_Mode          | BOOL      |                                    |
|                    | Run_Mode                  | BOOL      |                                    |
|                    | PushButton_Read           | BOOL      |                                    |
|                    | Lamp_Warning_UnitLifeTime | BOOL      |                                    |
|                    | PowerOnTime               | TIME      |                                    |
|                    | R_TRIG_instance1          | R_TRIG    |                                    |
|                    | PushButton_Read_R_TRIG    | BOOL      |                                    |
|                    | RS_instance               | RS        |                                    |

```

// Get button status.
Maintenance_Mode := J01_Ch1_In00;
Run_Mode := NOT(J01_Ch1_In00);
PushButton_Read := J01_Ch1_In01;

R_TRIG_instance1(c1k:=PushButton_Read, Q=>PushButton_Read_R_TRIG);

// Read total operating time.
PowerOnTime := PLC_ReadTotalPowerOnTime(EN:=(Maintenance_Mode & PushButton_Read_R_TRIG),
UnitType:=_CPU_UNIT);

RS_instance( Set:=(PowerOnTime > T#1825d),
Reset1:=Maintenance_Mode,
Q1=>Lamp_Warning_UnitLifeTime);

// Output warning to lamp.
J02_Ch1_Out00 := Lamp_Warning_UnitLifeTime;

```

# NX\_ReadTotalPowerOnTime

The NX\_ReadTotalPowerOnTime instruction reads the total power ON time from a Communications Coupler Unit or NX Unit.

| Instruction             | Name                             | FB/<br>FUN | Graphic expression | ST expression  |
|-------------------------|----------------------------------|------------|--------------------|--|
| NX_ReadTotalPowerOnTime | Read NX Unit Total Power ON Time | FB         |                    | NX_ReadTotalPowerONTime_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx, TotalPowerOnTime); |

## Variables

|                   | Meaning             | I/O    | Description                                   | Valid range           | Unit | Default |
|-------------------|---------------------|--------|---|-----------------------|------|---------|
| UnitProxy         | Specified Unit      | Input  | Specifies the target NX Unit.                 | ---                   | ---  | *1      |
| TotalPower-OnTime | Total power ON time | Output | Stores the total power ON time that was read. | Depends on data type. | ---  | 0       |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|                   | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-942 for details on the structure <code>_sNXUNIT_ID</code> . |
| TotalPower-OnTime |          |             |      |       |       |       |          |       |       |      |     |      |      |              |       | 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 total power ON time is read, 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                  | 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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT       |

To *UnitProxy*, pass the 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.

## 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 NX102 CPU Unit and 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]<br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible.       |
| _NXB_UnitMsgActiveTbl[i]                      | NX Unit Message Enabled Status             | BOOL      | This table indicates the slaves that can perform message communications.<br>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 in the next task period.

The read value of *TotalPowerOnTime* will be 0.

## Precautions for Correct Use

- Execution of this instruction is continued until 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 the execution.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- For *UnitProxy*, specify the device variable that is assigned 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 in the I/O Map of the Sysmac Studio.

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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* changes to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>   | Meaning  |
|-------------------------|---|--|
| 16#0400                 | 16#00000000   | The value of <i>UnitProxy</i> is outside the valid range.  |
| 16#0419                 | 16#00000000   | The data type of <i>UnitProxy</i> is not correct.  |
| 16#2C00                 | 16#00000401   | The specified Unit does not support the instruction.   |
|                         | 16#00001001   | An input parameter, output parameter, or in-out parameter is incorrect.<br>Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter. |
|                         | 16#00001002   |  |
|                         | 16#00170000   |  |
|                         | 16#00200000   |  |
|                         | 16#00210000   |  |
|                         | 16#00001010   | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
|                         | 16#00001101   | The Unit is not correct.<br>Check the Unit.  |
|                         | 16#0000110B   | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
|                         | 16#00001110   | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
|                         | 16#00001111   | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
|                         | 16#00002101   | The specified NX object cannot be written.   |
|                         | 16#00002110   | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
| 16#00002210             | The specified Unit is not in a mode that allows writing data.   |  |
| 16#00002213             | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed. |  |



| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning  |
|-------------------------|--|--|
|                         | 16#00002230  | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write destination NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
|                         | 16#00002231  | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction.  |
|                         | 16#0000250F  | Hardware access failed.<br>Execute the instruction again.  |
|                         | 16#00002601<br>16#00002602<br>16#00100000  | The specified Unit does not support this instruction.<br>Check the version of the Unit.  |
|                         | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.   |
|                         | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.  |
|                         | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.  |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.  |
|                         | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.  |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.   |

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning   |
|-------------------------|--|---|
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000                               | An error occurred in the communications network.<br>Reduce the amount of communications traffic.  |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000 | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON. |
| 16#2C01                 | 16#00000000  | The number of instructions that can be simultaneously executed was exceeded.  |
| 16#2C02                 | 16#00000000  | A timeout occurred during communications.   |
| 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.

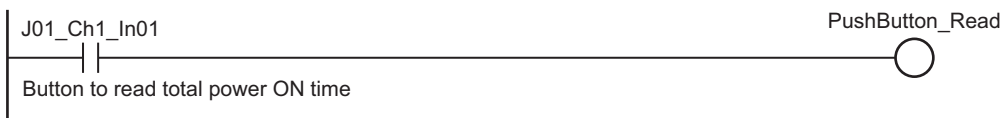
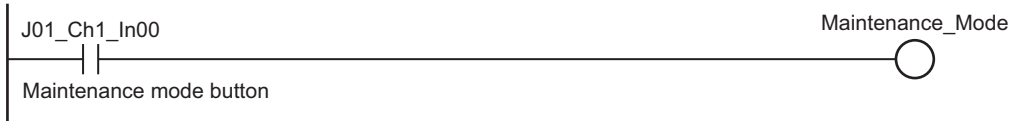
| Unit   | Description   |
|--------|---|
| Unit 1 | NX Unit (ID)  |
| Unit 2 | NX Unit (OD)  |
| Unit 3 | NX Unit (Unit from which to read the total power ON time) |

## LD

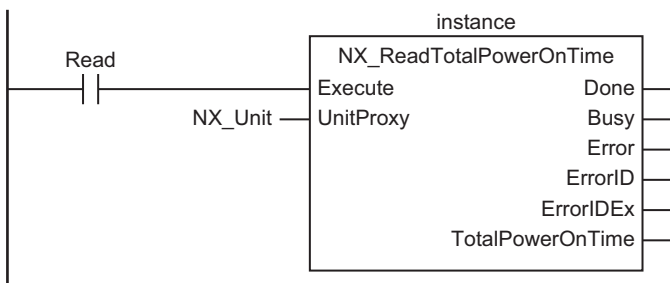
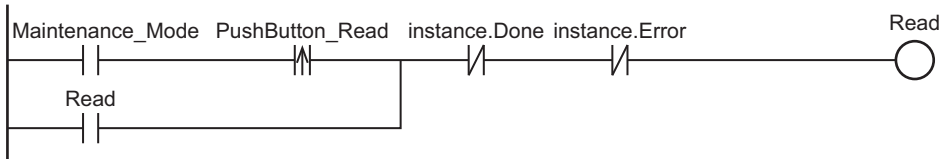
| Internal Variables | Variable                  | Data type               | Initial value | Comment                         |
|--------------------|---------------------------|-------------------------|---------------|---------------------------------|
|                    | Maintenance_Mode          | 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 instance             | BOOL                    | FALSE         |                                 |
|                    |                           | NX_ReadTotalPowerOnTime |               |                                 |

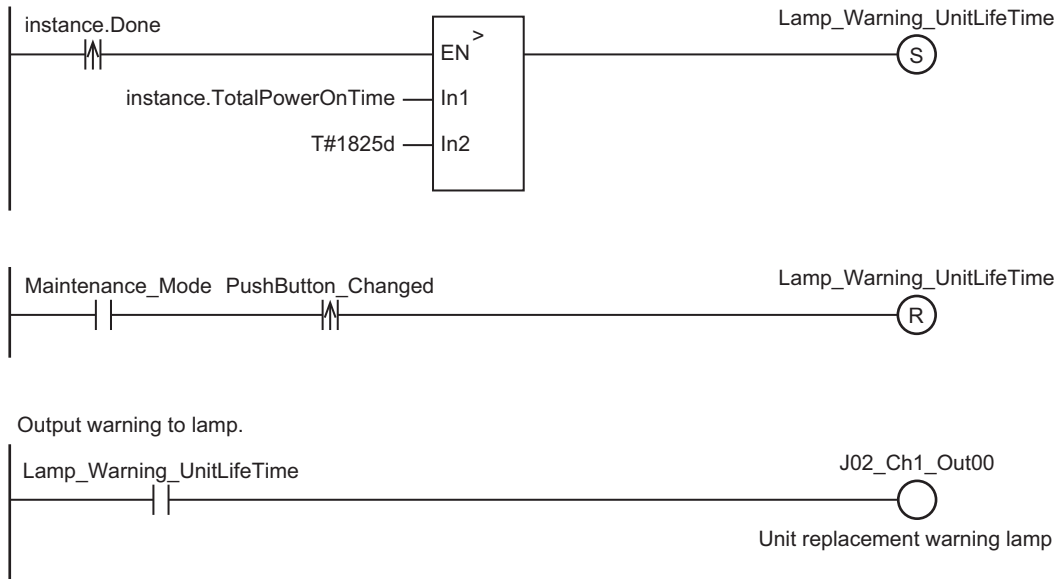
| External Variables | Variable      | Data type   | Comment                                   |
|--------------------|---------------|-------------|---|
|                    | NX_Unit       | _sNXUNIT_ID |   |
|                    | J01_Ch1_In00  | BOOL        | Maintenance mode button                   |
|                    | 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.





**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_ReadTotalPowerOnTime |               |                                 |
|                    | 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 |

| External Variables | Variable      | Data type | Comment                                   |
|--------------------|---------------|-----------|---|
|                    | 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

| Instruction | Name                | Page       |
|-------------|---------------------|------------|
| PrgStart    | Enable Program      | page 2-952 |
| PrgStop     | Disable Program     | page 2-961 |
| PrgStatus   | Read Program Status | page 2-980 |

# PrgStart

The PrgStart instruction enables the execution of the specified program.

| Instruction | Name           | FB/<br>FUN | Graphic expression | ST expression                       |
|-------------|----------------|------------|--------------------|-------------------------------------|
| PrgStart    | Enable Program | FUN        |                    | Out:=PrgStart(PrgName, isFirstRun); |



## 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.

## Variables

|            | Meaning                          | I/O    | Description   | Valid range  | Unit | Default |
|------------|----------------------------------|--------|---|--|------|---------|
| PrgName    | Program name                     |        | 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 <i>P_First_Run</i> system-defined variable in the first task period when the program is executed<br>TRUE: Change to TRUE.<br>FALSE: Change to FALSE. | Depends on data type.  | ---  | TRUE    |
| Out        | Normal end flag                  | Output | Normal end flag<br>TRUE: Normal end<br>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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 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.

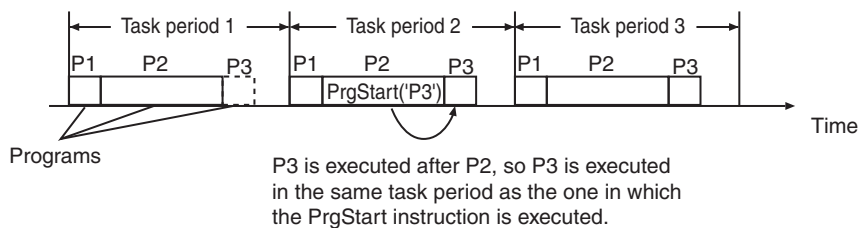
The value of *Out* is TRUE if the instruction ends normally, and FALSE if the instruction ends in an error.

## 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.

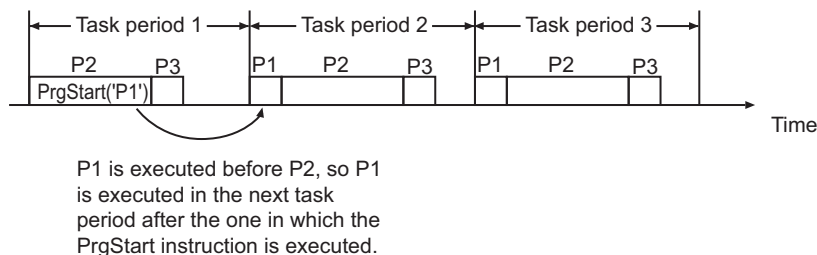
### ● Enabling a Program Executed After the PrgStart Instruction

- 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.

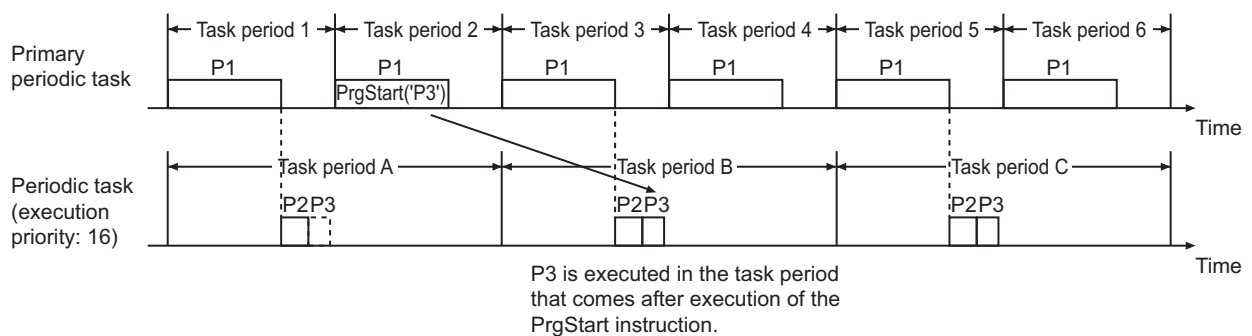


## Operation Example When a Program in a Different Task Is Specified

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.

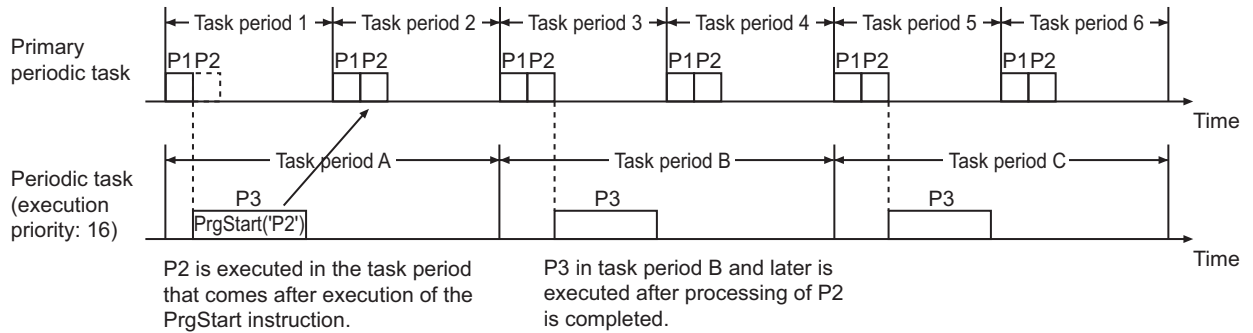
### ● Enabling a Program in a Task with a Lower Execution Priority Than the Current Task

- 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.



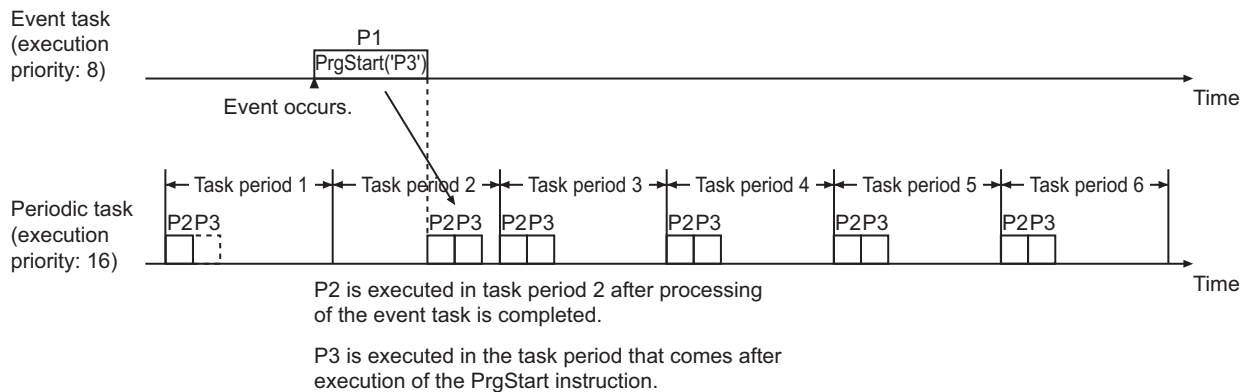
### ● Enabling a Program in a Task with a Higher Execution Priority Than the Current Task

- 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.



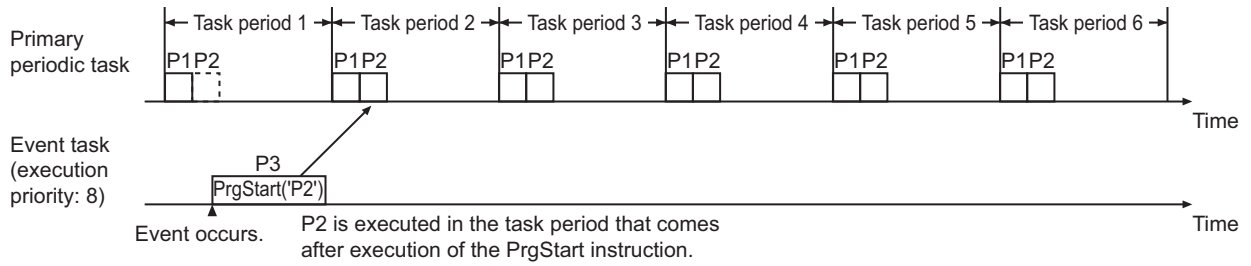
### ● Enabling a Program in a Task with a Lower Execution Priority from an Event Task

- 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.



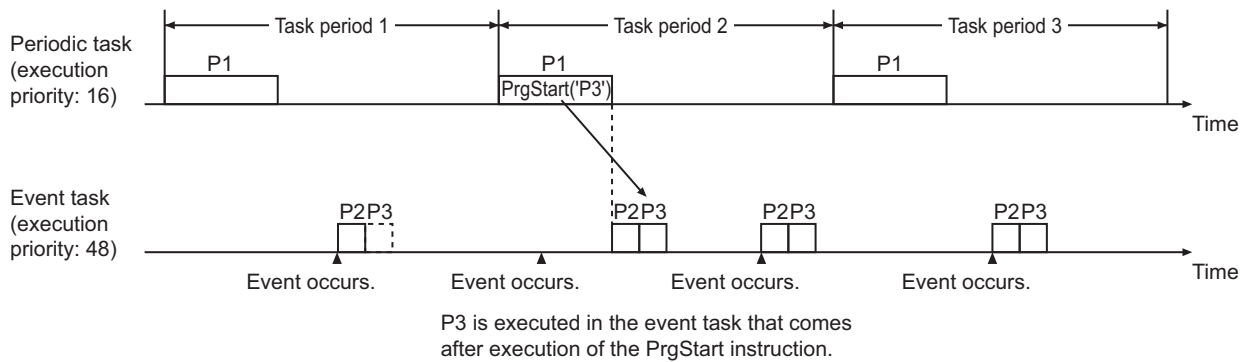
### ● Enabling a Program in a Task with a Higher Execution Priority from an Event Task

- 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.



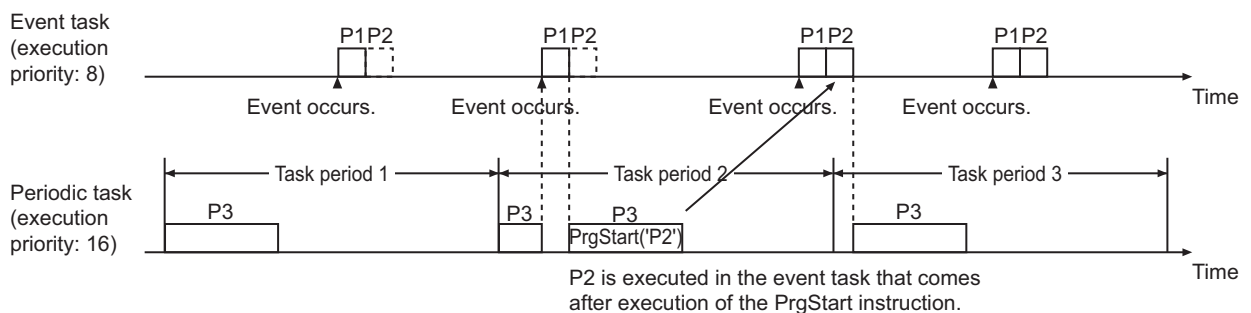
● **Enabling a Program in an Event Task with a Lower Execution Priority from a Periodic Task**

- 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.



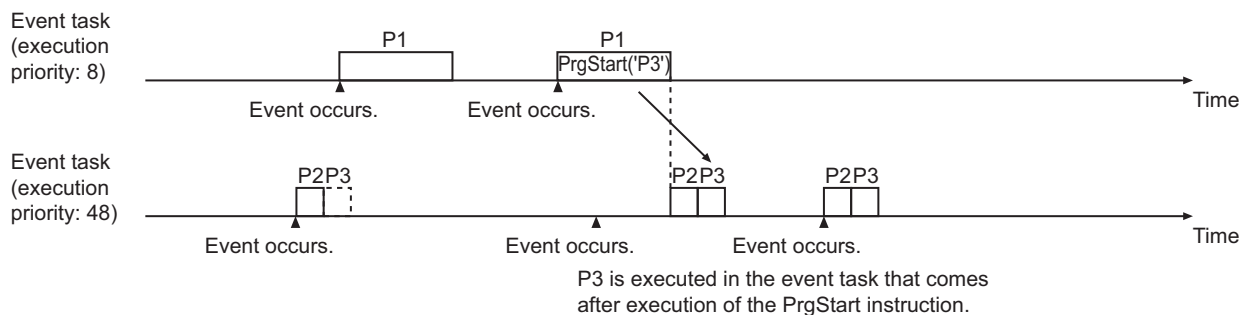
● **Enabling a Program in an Event Task with a Higher Execution Priority from a Periodic Task**

- 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.



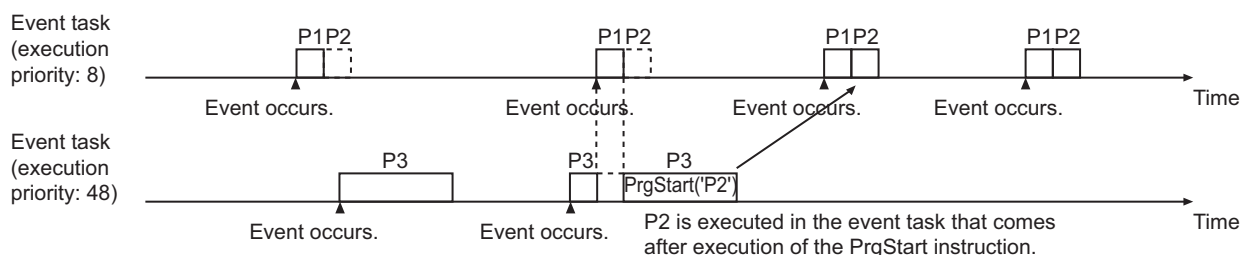
### ● Enabling a Program in an Event Task with a Lower Execution Priority from an Event Task

- 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.

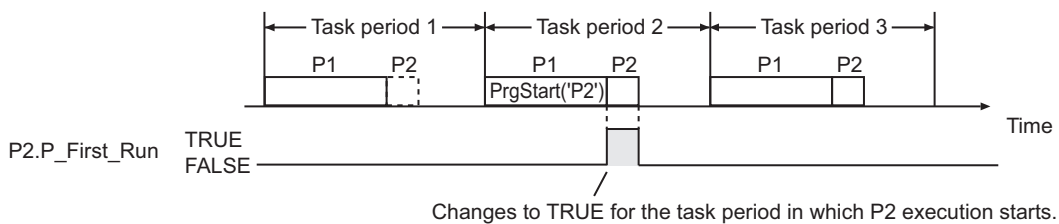
The relation between *isFirstRun* and *P\_First\_Run* is shown in the following table.

The behavior of *P\_First\_Run* depends on whether the specified program is disabled or already enabled.

| Value of <i>isFirstRun</i> | Status of the program | Value of <i>P_First_Run</i>  |
|----------------------------|-----------------------|--|
| 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.   |

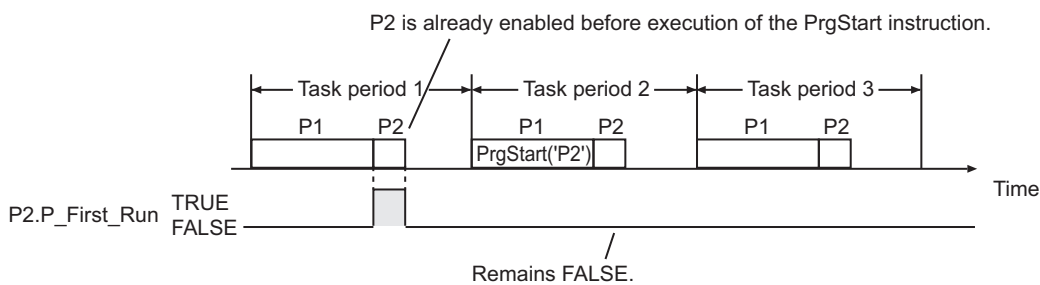
● **When the Value of *isFirstRun* Is TRUE and the Program Is Disabled**

The value of *P\_First\_Run* changes to TRUE for one task period when execution of the program starts. After that, the value of *P\_First\_Run* changes to FALSE.



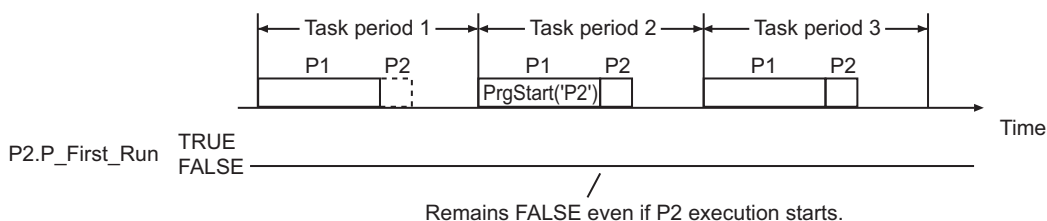
● **When the Value of *isFirstRun* Is TRUE and the Program Is Already Enabled**

The value of *P\_First\_Run* remains FALSE even if the PrgStart instruction is executed.



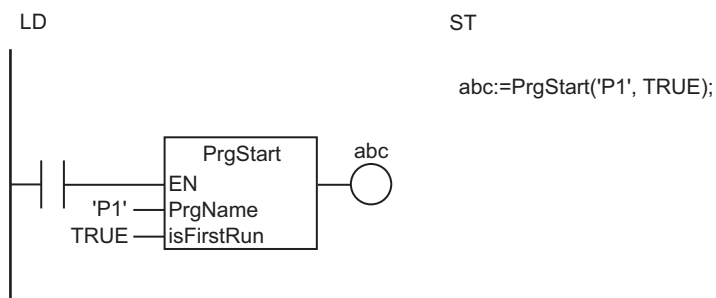
● **When the Value of *isFirstRun* Is FALSE**

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'.



## 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 <i>isFirstRun</i> is changed to FALSE and the PrgStart instruction is executed, P_First_Run remains FALSE even after 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 instruction, *PrgStop* on page 2-961, to disable a specified program from the user program.
- Use the instruction, *PrgStatus* on page 2-980, 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. It means that the PrgStart or PrgStop instruction will be disabled after the change, if executed before the change.
- 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.
- Internal variables, input variables, output variables and in-out variables of the specified program retain the same values as those for the previous execution of the program. To initialize these variables before execution of the program, change the value of *isFirstRun* to TRUE and execute the instruction, and then perform initialization processing in the specified program if the value of *P\_First\_Run* is TRUE.
- An error will occur in the following case. *Out* will be FALSE.
  - a) The program specified by *PrgName* does not exist.

### Sample Programming

Refer to the *Sample Programming* on page 2-968 for the PrgStop instruction.



# PrgStop

The PrgStop instruction disables execution of the specified program.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression          |
|-------------|-----------------|------------|--------------------|------------------------|
| PrgStop     | Disable Program | FUN        |                    | Out:=PrgStop(PrgName); |



## 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.

## Variables

|         | Meaning         | I/O    | Description   | Valid range   | Unit | Default |
|---------|-----------------|--------|---|---|------|---------|
| PrgName | Program name    | Input  | Name of specified program                               | 128 bytes max.<br>(127 single-byte alphanumeric characters plus the final NULL character) | ---  | *1      |
| Out     | Normal end flag | Output | Normal end flag<br>TRUE: Normal end<br>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.

|         | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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.

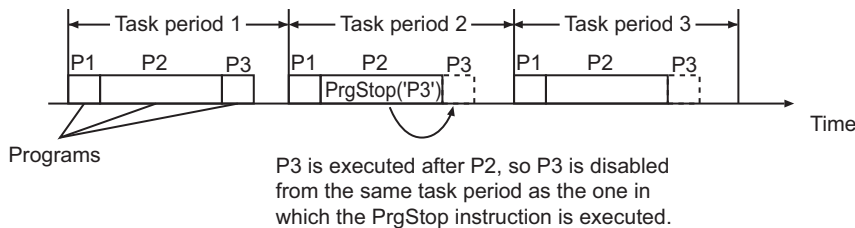
The value of *Out* is TRUE if the instruction ends normally, and FALSE if the instruction ends in an error.

## 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.

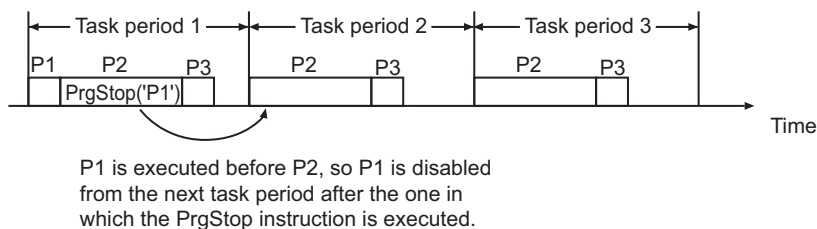
### ● Disabling a Program Executed After the PrgStop Instruction

- 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.



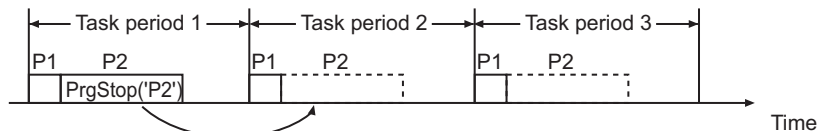
### ● Disabling a Program Executed Before the PrgStop Instruction

- 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.



The program is executed to the end in the task period in which the PrgStop instruction is executed.

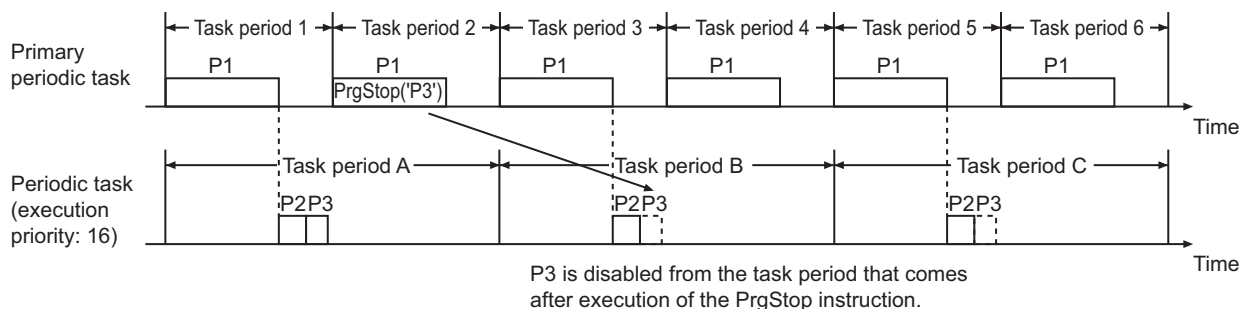
The program is disabled from the next task period after the one in which the PrgStop instruction is executed.

## 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.

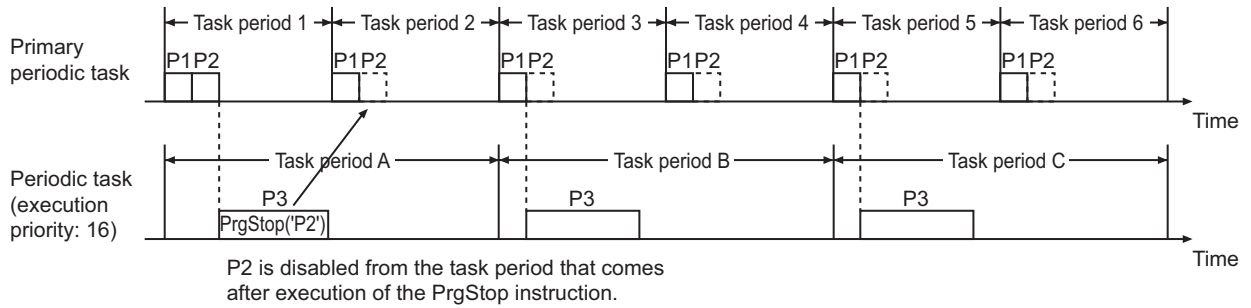
### Disabling a Program in a Task with a Lower Execution Priority Than the Current Task

- 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.



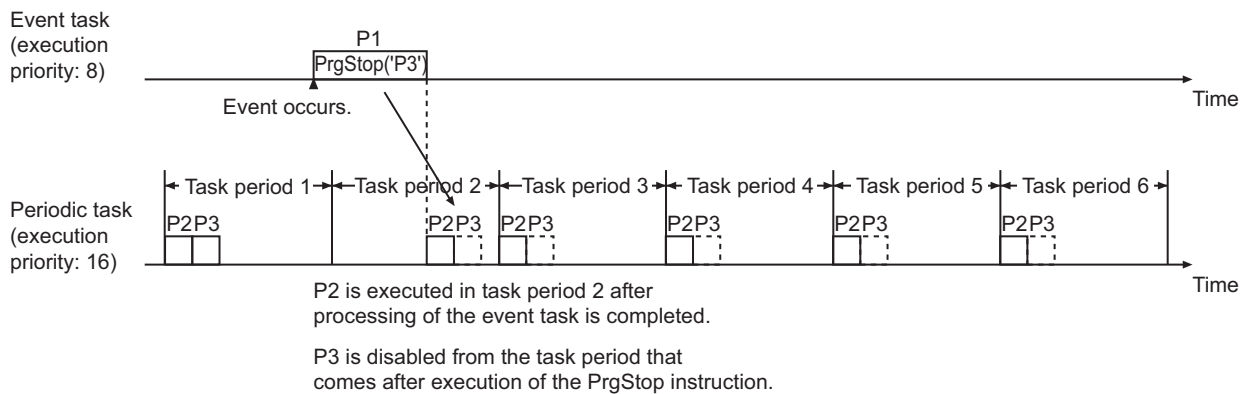
### Disabling a Program in a Task with a Higher Execution Priority Than the Current Task

- 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.



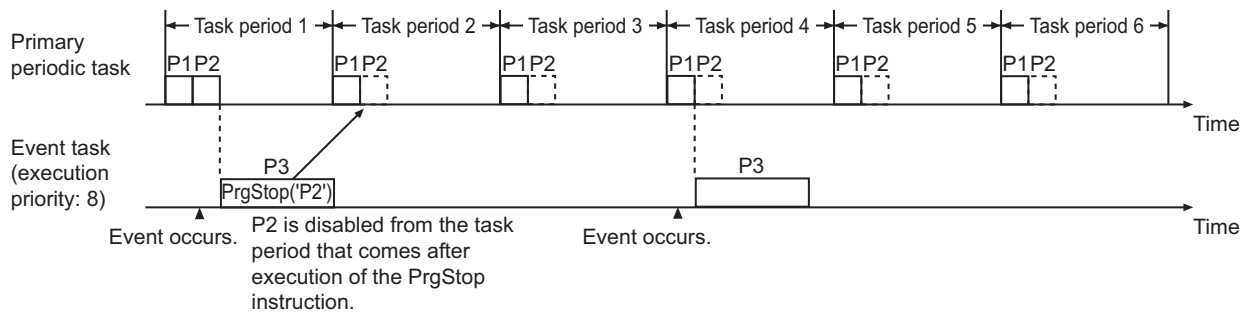
● **Disabling a Program in a Task with a Lower Execution Priority from an Event Task**

- 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.



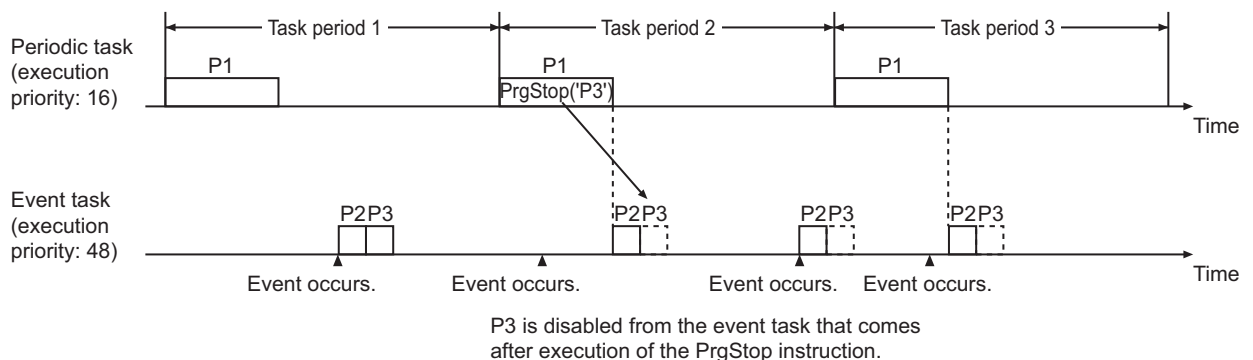
● **Disabling a Program in a Task with a Higher Execution Priority from an Event Task**

- 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.



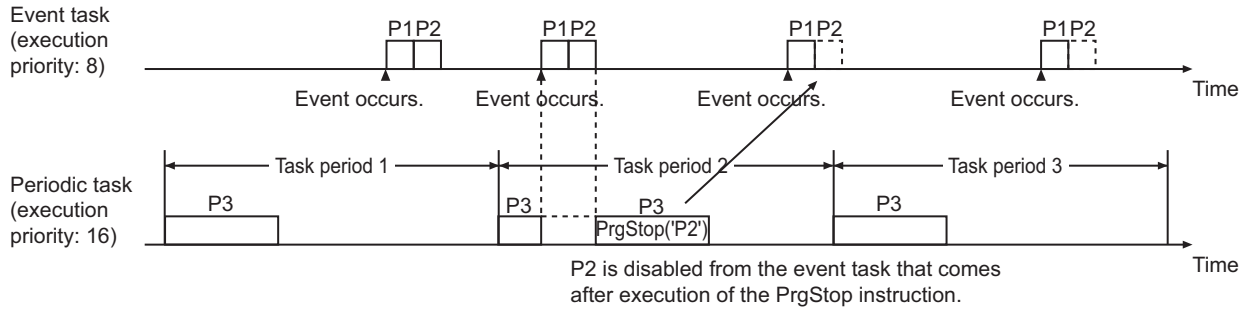
### ● Disabling a Program in an Event Task with a Lower Execution Priority from a Periodic Task

- 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.



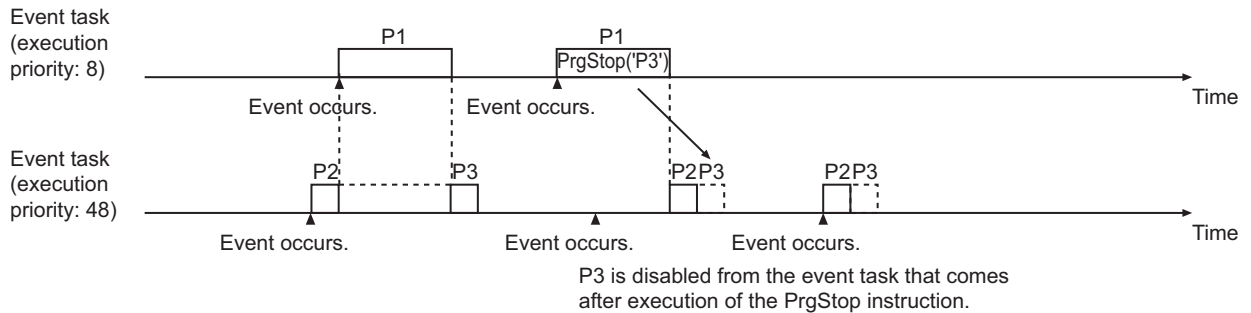
### ● Disabling a Program in an Event Task with a Higher Execution Priority from a Periodic Task

- 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.



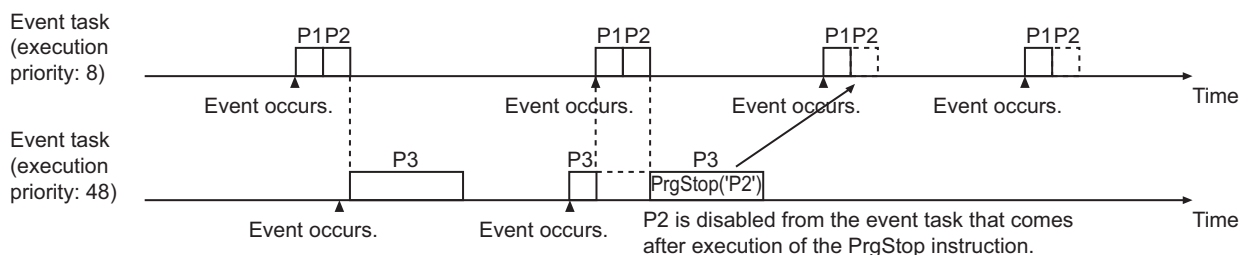
● **Disabling a Program in an Event Task with a Lower Execution Priority from an Event Task**

- 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.



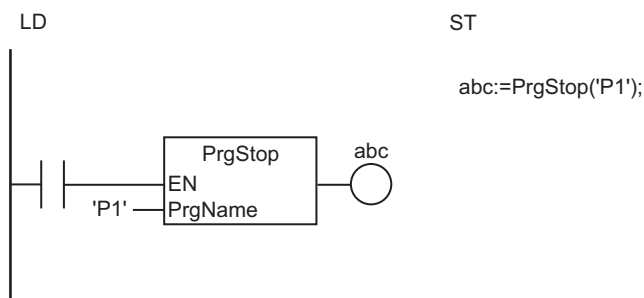
● **Disabling a Program in an Event Task with a Higher Execution Priority from an Event Task**

- 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'.



## Additional Information

- Use the instruction, *PrgStart* on page 2-952, to enable a specified program from the user program.
- Use the instruction, *PrgStatus* on page 2-980, 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 a program that has this kind of instruction, check if the value of *Busy* for the instruction is *FALSE* to make sure that execution of the instruction is not in progress.
- Execution of the *NX\_DOutTimeStamp* or *NX\_AryDOutTimeStamp* instruction may require more than one task. Before you disable a program that has these instructions, check if the value of *Enable* for the instruction is *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. It means that the *PrgStart* or *PrgStop* instruction will be disabled after the change, if executed before the change.
- 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.
  - a) The execution of a motion control instruction is not in progress.

- b) 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 in progress.
- c) There are no time stamp instructions that are 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.
  - a) The program specified by *PrgName* does not exist.

## Sample Programming

This section provides two example programs for explanation.

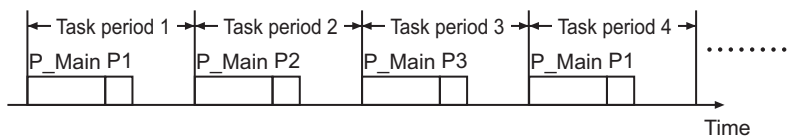
### Example of sequential execution of programs

The following shows an example where three programs are executed one by one for every task period.

In this example, P1, P2, and P3 are provided as example programs.

These programs are executed sequentially one by one in rotation for every task period.

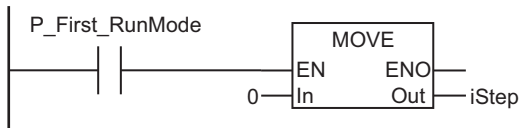
P\_Main is the program that gives instructions to enable or disable the three programs.



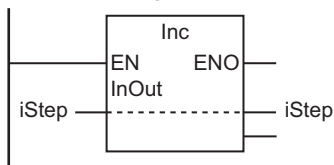
#### ● LD

| Variable | Data type | Default | Comment                      |
|----------|-----------|---------|------------------------------|
| iStep    | DINT      | 0       | Number of program to execute |

Set *iStep* variable to 0 at start of operation.

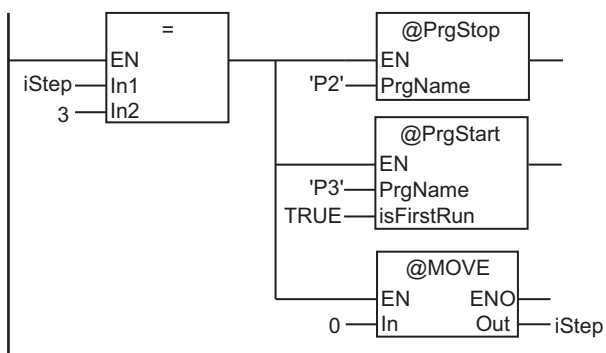
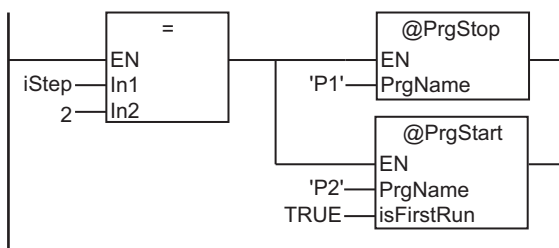
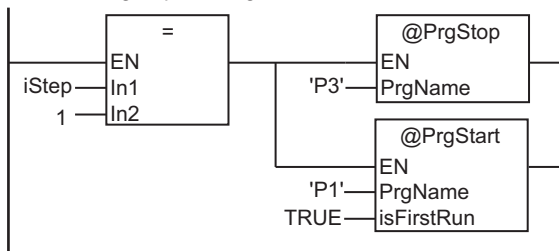


Increment *iStep* variable.





Execute PrgStop and PrgStart instructions.



## ● 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;

```

## Execution of Specified Programs at the Next Start-up

This example shows a case where some of programs are specified to be executed at the next start-up.

You need to specify which programs should be executed at the next start-up before turning OFF the power supply to the Controller.

The next time the power supply is turned ON, only the specified 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 <sup>*1</sup> , 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<br>Config1, TRUE, TRUE, TRUE, FALSE, FALSE<br>Config2, TRUE, TRUE, FALSE, TRUE, FALSE<br>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.

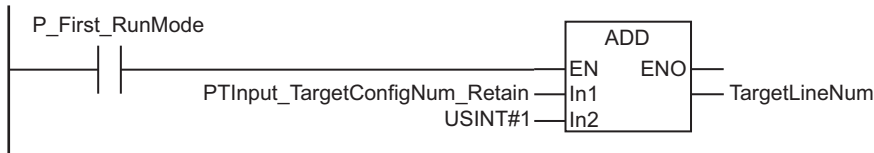
| Structure | Variable   | Data type  | Offset type | Comment                   |
|-----------|------------|------------|-------------|---------------------------|
| ▼         | myConfig   | STRUCT     | NJ          | Module configuration      |
|           | configName | STRING[32] |             | Module 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

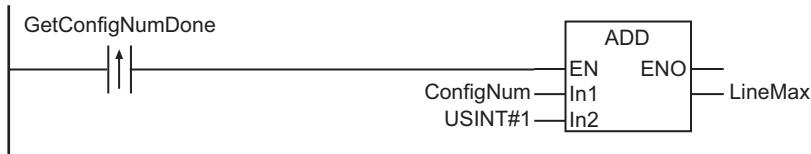
| Variable                       | Data type | Default | Retain                              | Comment   |
|--------------------------------|-----------|---------|-------------------------------------|---|
| Open                           | FileOpen  |         | <input type="checkbox"/>            | Instance of FileOpen instruction  |
| TopLineGetter                  | FileGets  |         | <input type="checkbox"/>            | Instance of FileGets instruction  |
| LineGetter                     | FileGets  |         | <input type="checkbox"/>            | Instance of FileGets instruction  |
| Close                          | FileClose |         | <input type="checkbox"/>            | Instance of FileClose instruction   |
| PTInput_TargetConfigNum_Retain | USINT     | 0       | <input checked="" type="checkbox"/> | Number of the module configuration to execute next time operation starts                              |
| CurrentLineNum                 | USINT     | 1       | <input type="checkbox"/>            | Current configuration file row  |
| TargetLineNum                  | USINT     | 0       | <input type="checkbox"/>            | Row for <i>CurrentConfig</i> in configuration file  |
| ConfigNum                      | USINT     | 1       | <input type="checkbox"/>            | Number given in row 1 of configuration file   |
| LineMax                        | USINT     | 3       | <input type="checkbox"/>            | Number of rows in configuration file obtained from <i>ConfigNum</i>                                   |
| isOverLine                     | BOOL      | FALSE   | <input type="checkbox"/>            | Error flag when value of <i>PTInput_TargetConfigNum_Retain</i> is larger than value of <i>LineMax</i> |
| Busy                           | BOOL      | FALSE   | <input type="checkbox"/>            | Processing flag   |
| SubDeliiNG                     | BOOL      | FALSE   | <input type="checkbox"/>            | Read error end flag for <i>CurrentConfig</i>  |
| Error                          | BOOL      | FALSE   | <input type="checkbox"/>            | Error flag  |
| opening                        | BOOL      | FALSE   | <input type="checkbox"/>            | Configuration file open execution flag  |

| Variable         | Data type | Default  | Retain                   | Comment  |
|------------------|-----------|--|--------------------------|--|
| myFileID         | DWORD     | 0  | <input type="checkbox"/> | File ID of configuration file                              |
| TopLineGetting   | BOOL      | FALSE  | <input type="checkbox"/> | <i>ConfigNum</i> read execution flag                       |
| GetConfigNumDone | BOOL      | FALSE  | <input type="checkbox"/> | <i>ConfigNum</i> read done flag                            |
| SelectDone       | BOOL      | FALSE  | <input type="checkbox"/> | <i>CurrentConfig</i> read done flag                        |
| reading          | BOOL      | FALSE  | <input type="checkbox"/> | Configuration file row 2 or higher read execution flag     |
| CurrentConfig    | myConfig  | (configName:="",<br>moduleA:=FALSE,<br>moduleB:=FALSE,<br>moduleC:=FALSE,<br>moduleD:=FALSE) | <input type="checkbox"/> | Module configuration to execute next time operation starts |
| Error_exceptOpen | BOOL      | FALSE  | <input type="checkbox"/> | Configuration file close execution flag when error occurs  |

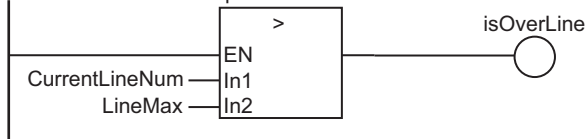
Get number of the module configuration to execute next time operation starts.



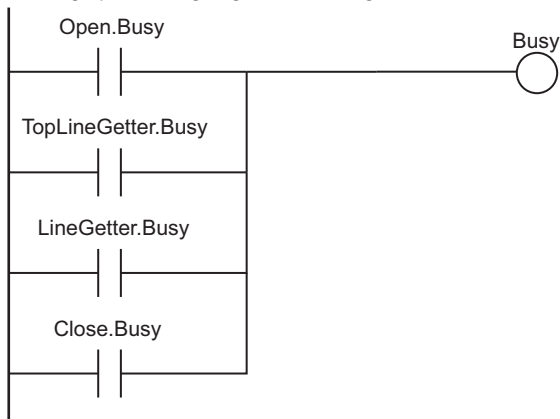
Calculate number of rows from contents of row 1 of configuration file.

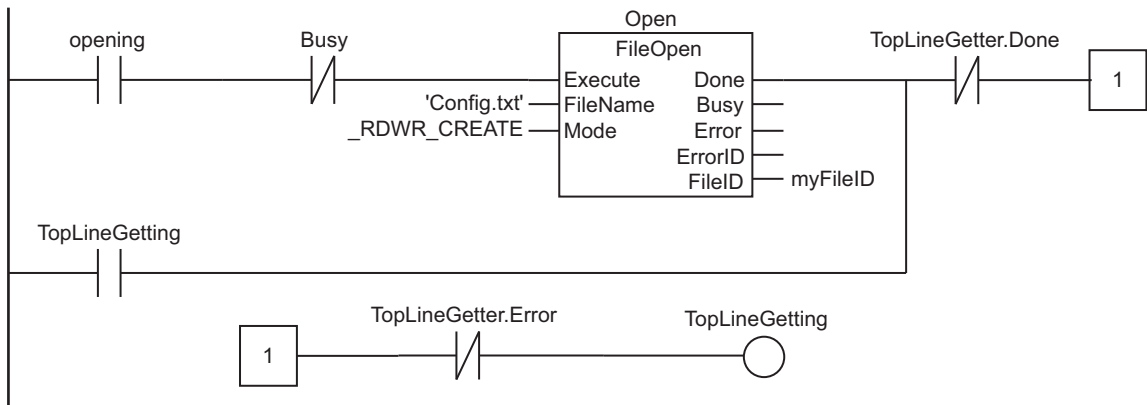
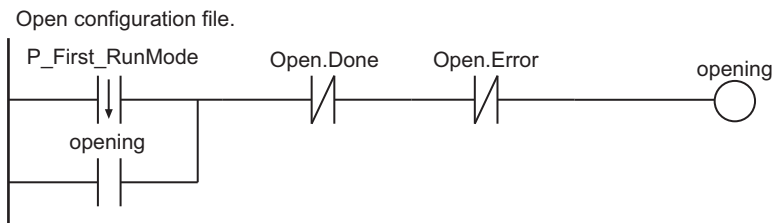
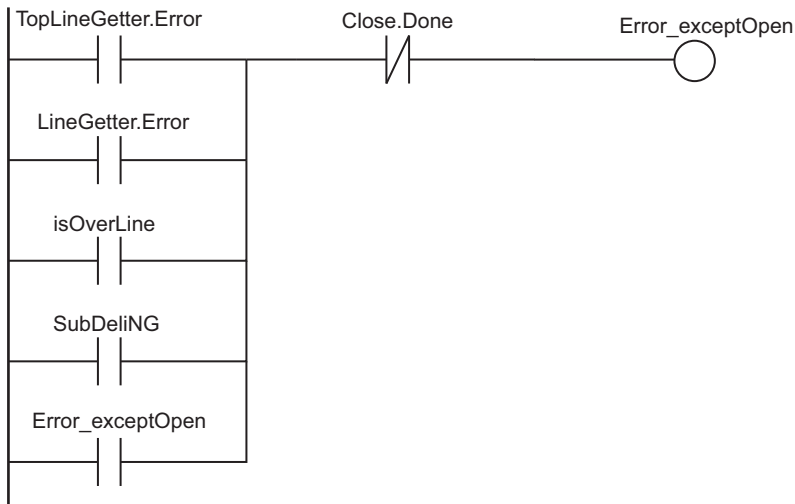
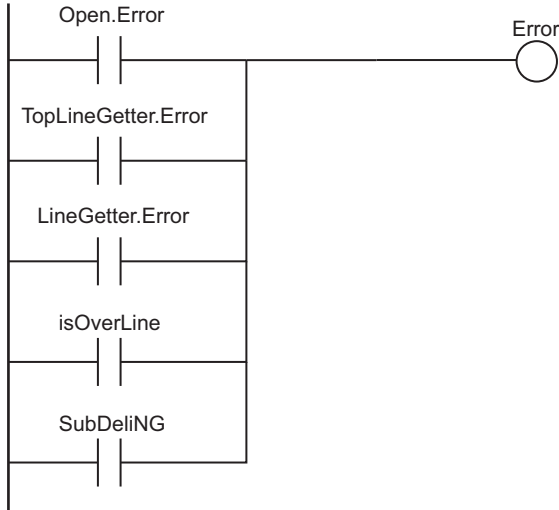


Detect error when number of rows in configuration file does not match number of the module configuration to execute next time operation starts.

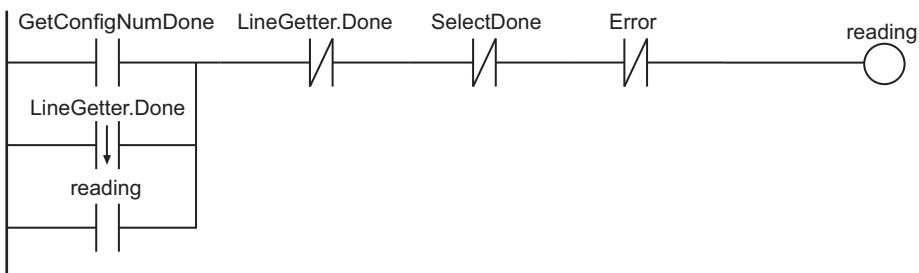
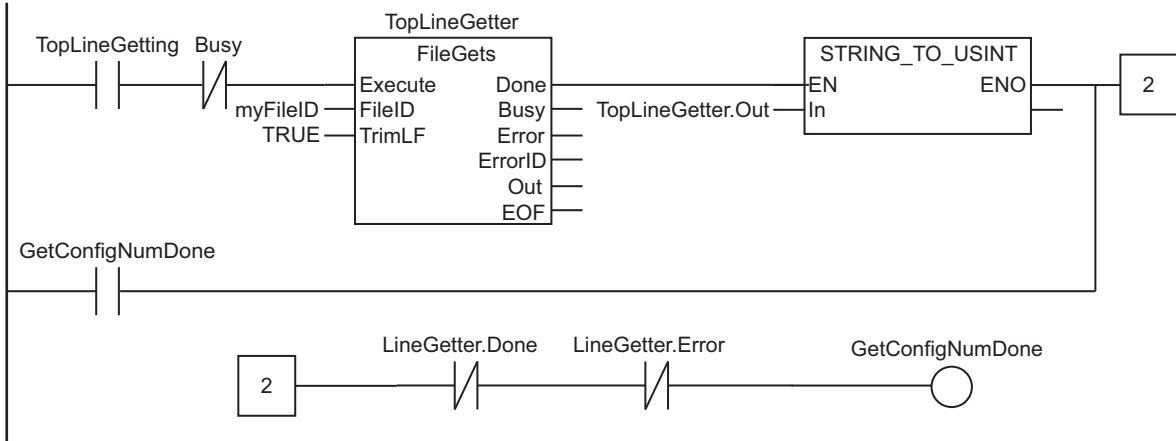


Manage processing flag and error flags.

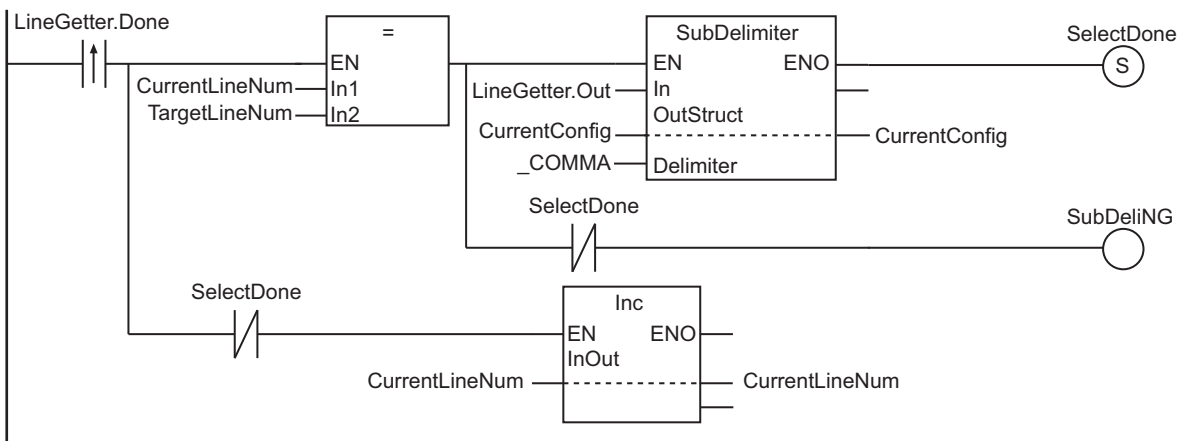
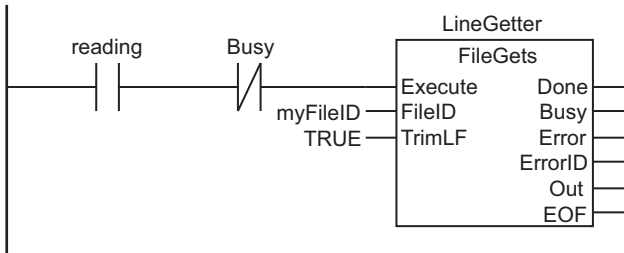




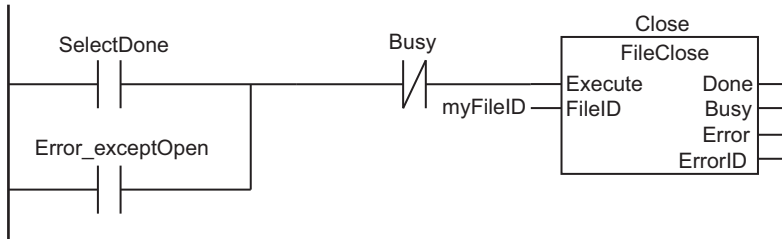
Read row 1 of configuration file.



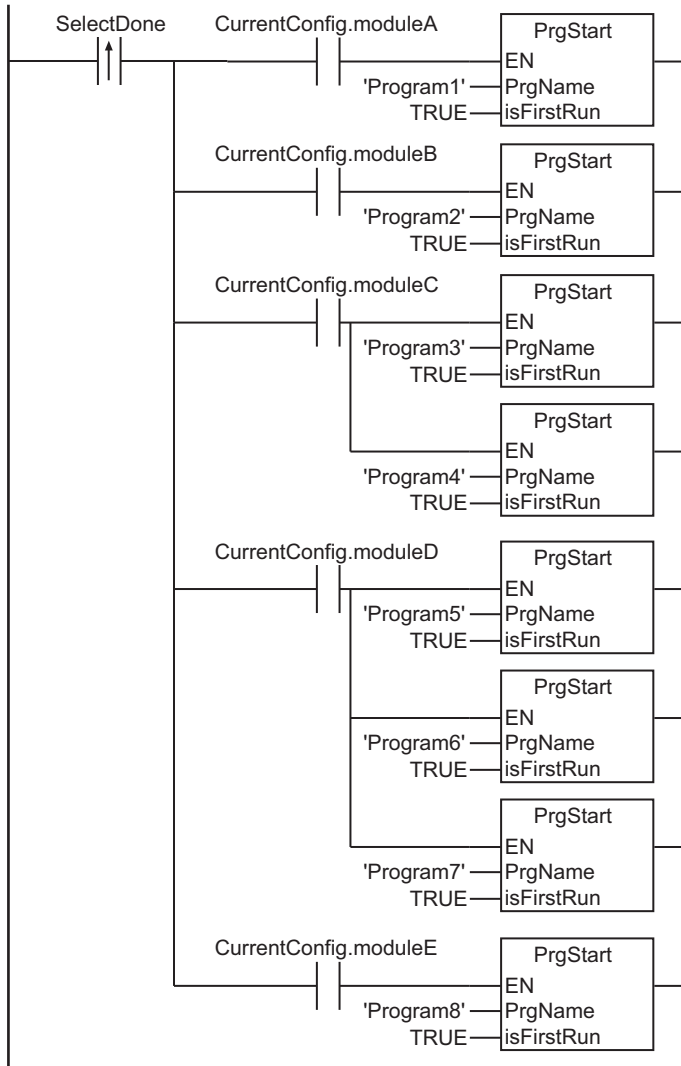
Read row 2 or higher of configuration file.



Close configuration file.



Execute PrgStart instruction.



● ST

| Variable      | Data type | Default | Retain                   | Comment                           |
|---------------|-----------|---------|--------------------------|-----------------------------------|
| Open          | FileOpen  |         | <input type="checkbox"/> | Instance of FileOpen instruction  |
| TopLineGetter | FileGets  |         | <input type="checkbox"/> | Instance of FileGets instruction  |
| LineGetter    | FileGets  |         | <input type="checkbox"/> | Instance of FileGets instruction  |
| Close         | FileClose |         | <input type="checkbox"/> | Instance of FileClose instruction |

| Variable                       | Data type | Default   | Retain                              | Comment   |
|--------------------------------|-----------|---|-------------------------------------|---|
| PTInput_TargetConfigNum_Retain | USINT     | 0   | <input checked="" type="checkbox"/> | Number of the module configuration to execute next time operation starts                              |
| CurrentLineNum                 | USINT     | 1   | <input type="checkbox"/>            | Current configuration file row  |
| TargetLineNum                  | USINT     | 0   | <input type="checkbox"/>            | Row for <i>CurrentConfig</i> in configuration file  |
| ConfigNum                      | USINT     | 1   | <input type="checkbox"/>            | Number given in row 1 of configuration file   |
| LineMax                        | USINT     | 3   | <input type="checkbox"/>            | Number of rows in configuration file obtained from <i>ConfigNum</i>                                   |
| isOverLine                     | BOOL      | FALSE   | <input type="checkbox"/>            | Error flag when value of <i>PTInput_TargetConfigNum_Retain</i> is larger than value of <i>LineMax</i> |
| Busy                           | BOOL      | FALSE   | <input type="checkbox"/>            | Processing flag   |
| SubDelinG                      | BOOL      | FALSE   | <input type="checkbox"/>            | Read error end flag for <i>CurrentConfig</i>  |
| Error                          | BOOL      | FALSE   | <input type="checkbox"/>            | Error flag  |
| opening                        | BOOL      | FALSE   | <input type="checkbox"/>            | Configuration file open execution flag  |
| myFileID                       | DWORD     | 0   | <input type="checkbox"/>            | File ID of configuration file   |
| TopLineGetting                 | BOOL      | FALSE   | <input type="checkbox"/>            | <i>ConfigNum</i> read execution flag  |
| GetConfigNumDone               | BOOL      | FALSE   | <input type="checkbox"/>            | <i>ConfigNum</i> read done flag   |
| SelectDone                     | BOOL      | FALSE   | <input type="checkbox"/>            | <i>CurrentConfig</i> read done flag   |
| reading                        | BOOL      | FALSE   | <input type="checkbox"/>            | Configuration file row 2 or higher read execution flag  |
| CurrentConfig                  | myConfig  | (configName:=",<br>moduleA:=FALSE,<br>moduleB:=FALSE,<br>moduleC:=FALSE,<br>moduleD:=FALSE) | <input type="checkbox"/>            | Module configuration to execute next time operation starts  |
| Error_exceptOpen               | BOOL      | FALSE   | <input type="checkbox"/>            | Configuration file close execution flag when error occurs   |
| R_GetConfigNumDone             | R_TRIG    |   | <input type="checkbox"/>            | Instance of R_TRIG instruction  |
| RS_1                           | RS        |   | <input type="checkbox"/>            | Instance of RS instruction  |
| RS_2                           | RS        |   | <input type="checkbox"/>            | Instance of RS instruction  |
| SecondCycle                    | F_TRIG    |   | <input type="checkbox"/>            | Instance of F_TRIG instruction  |
| RS_3                           | RS        |   | <input type="checkbox"/>            | Instance of RS instruction  |
| ConvertDone                    | BOOL      | FALSE   | <input type="checkbox"/>            | Conversion done flag for converting character in row 1 of configuration file to a number.             |
| RS_4                           | RS        |   | <input type="checkbox"/>            | Instance of RS instruction  |
| F_LineGetterDone               | F_TRIG    |   | <input type="checkbox"/>            | Instance of F_TRIG instruction  |



| Variable         | Data type | Default | Retain                   | Comment  |
|------------------|-----------|---------|--------------------------|--|
| R_LineGetterDone | R_TRIG    |         | <input type="checkbox"/> | Instance of R_TRIG instruction   |
| isTargetLine     | BOOL      | FALSE   | <input type="checkbox"/> | Flag to indicate that current row is the row of the module configuration to execute next time operation starts |
| SubDeliCondition | BOOL      | FALSE   | <input type="checkbox"/> | Expansion execution flag from module configuration to <i>CurrentConfig</i>                                     |
| RS_5             | RS        |         | <input type="checkbox"/> | Instance of RS instruction   |
| SubDeliDone      | BOOL      | FALSE   | <input type="checkbox"/> | Expansion done flag from module configuration to <i>CurrentConfig</i>  |
| R_SelectDone     | R_TRIG    |         | <input type="checkbox"/> | 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 Sub
DelinG;

RS_1(Set:= (TopLineGetter.Error OR LineGetter.Error OR isOverLine OR SubDeliNG), re
set1 := 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 => myFileID);
RS_3(Set := Open.Done, Reset1:=(TopLineGetter.Done OR TopLineGetter.Error), Q1=>Top
LineGetting);

// 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=>C
onvertDone);
RS_4(Set := ConvertDone, Reset1:=(LineGetter.Done OR LineGetter.Error), Q1=>GetConf
igNumDone);
F_LineGetterDone(Clk:=LineGetter.Done);
RS_5(Set := (GetConfigNumDone OR F_LineGetterDone.Q), Reset1:=(LineGetter.Done OR S
electDone 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 := CurrentConf
ig, 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 := myFileID
);

// Execute PrgStart instruction.
R_SelectDone(Clk:=SelectDone);
//moduleA
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleA), PrgName :='Program1', isFi
rstRun:=TRUE);
//moduleB
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleB), PrgName :='Program2', isFi
rstRun:=TRUE);
//moduleC
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program3', isFi
rstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program4', isFi
rstRun:=TRUE);
//moduleD
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program5', isFi
rstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program6', isFi
rstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program7', isFi
rstRun:=TRUE);

```

```
//moduleE  
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleE), PrgName :='Program8', isFi  
rstRun:=TRUE);
```

# PrgStatus

The PrgStatus instruction reads the status of the specified program.

| Instruction | Name                | FB/<br>FUN | Graphic expression | ST expression            |
|-------------|---------------------|------------|--------------------|--------------------------|
| PrgStatus   | Read Program Status | FUN        |                    | Out:=PrgStatus(PrgName); |



## 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.

## Variables

|         | Meaning        | I/O    | Description   | Valid range   | Unit | Default |
|---------|----------------|--------|---|---|------|---------|
| PrgName | Program name   | Input  | Name of specified program   | 128 bytes max.<br>(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<br>TRUE: Enabled.<br>FALSE: Disabled. | Depends on data type.   | ---  | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|         | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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  | <ul style="list-style-type: none"> <li>The <i>Initial Status</i> for the relevant program is set to <b>Run</b> on the Sysmac Studio.</li> <li>The PrgStart instruction was executed for the program.</li> </ul> |
| Disabled the next time the timing for execution occurs | <ul style="list-style-type: none"> <li>The <i>Initial Status</i> for the relevant program is set to <b>Stop</b> on the Sysmac Studio.</li> <li>The PrgStop instruction was executed for the program.</li> </ul> |

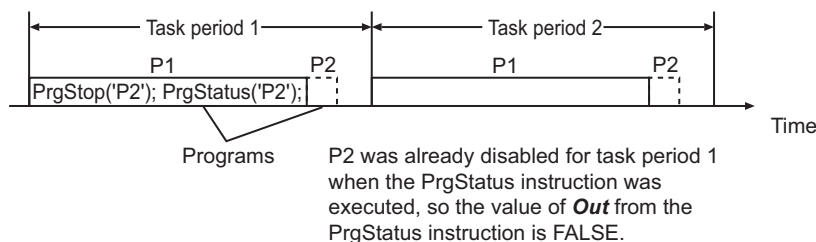
The specified program can be in the same task as this instruction, or it can be in a different task.

## Operation Example

This section provides some examples of the operation of this instruction.

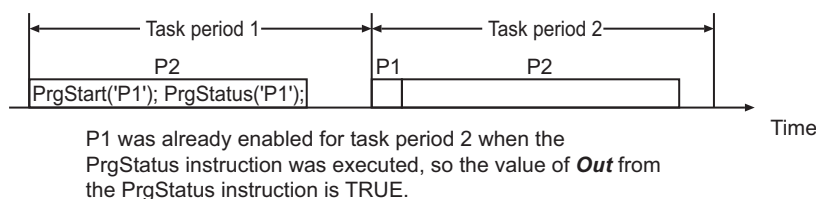
### ● Reading the Status of a Program After the PrgStatus Instruction in the Current Task

- 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* for the PrgStatus instruction is FALSE.



### ● Reading the Status of a Program Before the PrgStatus Instruction in the Current Task

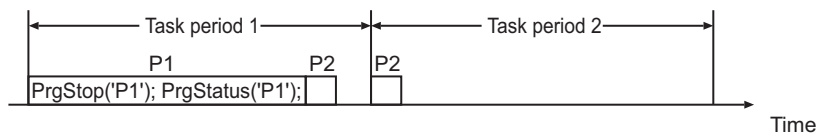
- In this example, there are two programs, P1 and P2, in the same task.
- The PrgStart instruction with P1 specified is executed in P2 of task period 1.
- The PrgStatus instruction with P1 specified is then executed in P2 of task period 1.
- P1 was enabled for task period 2, so the value of *Out* from 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.

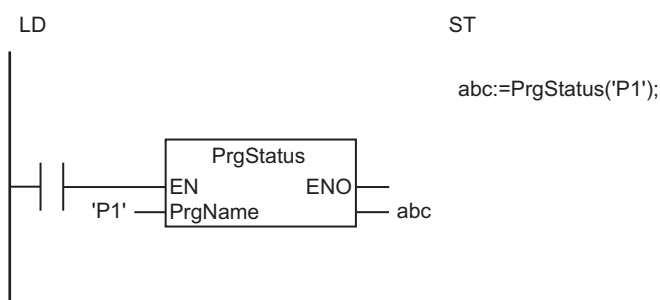
- P1 was disabled for task period 2, so the value of *Out* for the PrgStatus instruction is FALSE.



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.



### Additional Information

- Use the instruction, *PrgStart* on page 2-952, to enable a specified program from the user program.
- Use the instruction, *PrgStop* on page 2-961, 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.
  - a) The program specified by *PrgName* does not exist.

### 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. |

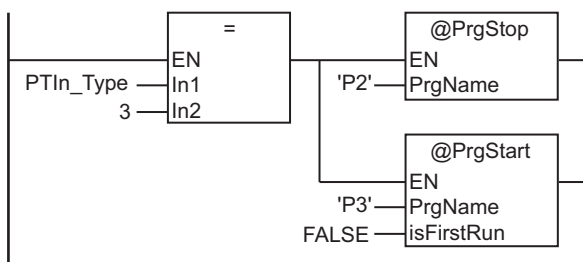
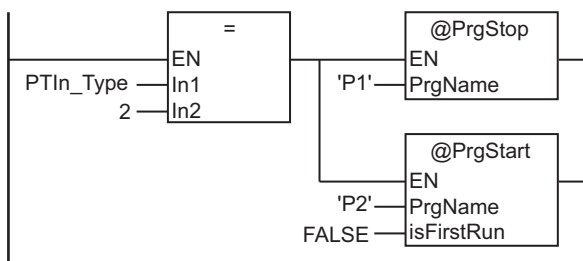
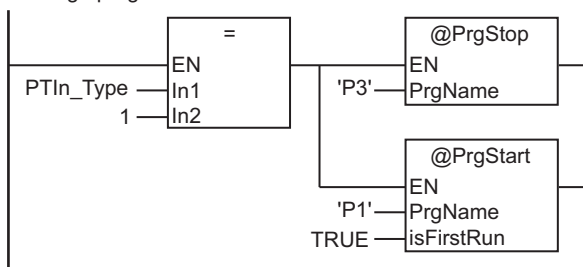
## Global Variables

| 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              |

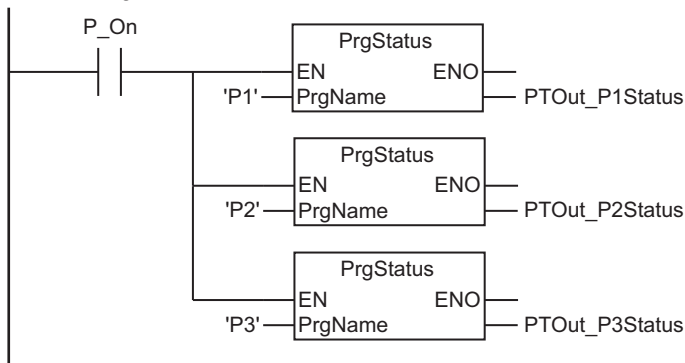
## LD

| 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

| <b>Instruction</b>     | <b>Name</b>                   | <b>Page</b> |
|------------------------|-------------------------------|-------------|
| EC_CoESDOWrite         | Write EtherCAT CoE SDO        | page 2-986  |
| EC_CoESDORead          | Read EtherCAT CoE SDO         | page 2-989  |
| EC_StartMon            | Start EtherCAT Packet Monitor | page 2-995  |
| EC_StopMon             | Stop EtherCAT Packet Monitor  | page 2-1001 |
| EC_SaveMon             | Save EtherCAT Packets         | page 2-1003 |
| EC_CopyMon             | Transfer EtherCAT Packets     | page 2-1005 |
| EC_DisconnectSlave     | Disconnect EtherCAT Slave     | page 2-1007 |
| EC_ConnectSlave        | Connect EtherCAT Slave        | page 2-1015 |
| EC_ChangeEnableSetting | Enable/Disable EtherCAT Slave | page 2-1017 |
| NX_WriteObj            | Write NX Unit Object          | page 2-1037 |
| NX_ReadObj             | Read NX Unit Object           | page 2-1053 |

# EC\_CoESDOWrite

The EC\_CoESDOWrite instruction writes a value to a CoE (CAN Application Protocol over EtherCAT) object of a specified slave on the EtherCAT network.

| Instruction    | Name                   | FB/<br>FUN | Graphic expression | ST expression   |
|----------------|------------------------|------------|--------------------|---|
| EC_CoESDOWrite | Write EtherCAT CoE SDO | FB         |                    | EC_CoESDOWrite_instance(Execute, NodeAdr, SdoObj, TimeOut, WriteDat, WriteSize, Done, Busy, Error, ErrorID, AbortCode); |

## Variables

|           | Meaning            | I/O    | Description  | Valid range            | Unit  | Default    |
|-----------|--------------------|--------|--|------------------------|-------|------------|
| NodeAdr   | Slave node address | Input  | Node address of the slave to access                            | 1 to 512* <sup>1</sup> | ---   | ---        |
| SdoObj    | SDO parameter      |        | SDO parameter  | ---                    | ---   | ---        |
| TimeOut   | Timeout time       |        | 0: 2.0 s<br>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* <sup>2</sup>                                  | 1 to 2048              | Bytes | ---        |
| AbortCode | Abort code         | Output | Response code for SDO access specified by CoE<br>0: Normal end | Depends on data type.  | ---   | ---        |

\*1. The range is 1 to 192 for the NX102 CPU Unit, NX1P2 CPU Unit, and 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.

|           | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-987 for details on the structure <code>_sSDO_ACCESS</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| 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 size 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  | <code>_sSDO_ACCESS</code> | ---                   | ---  | ---     |
| 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<br>TRUE: Access data for all subindexes<br>FALSE: Access data for the specified subindex | BOOL                      | Depends on data type. | ---  | ---     |

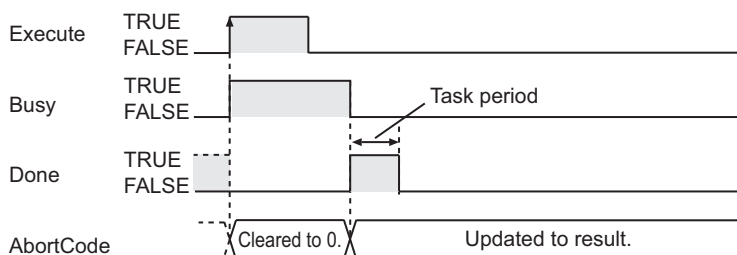
After the write is completed, the instruction waits for a response for the period of time specified with *TimeOut*.

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 value and meaning 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  |
|--|--|-----------|--|
| <code>_EC_MBXSlavTbl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>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)* for details on EtherCAT communications.
- Refer to *A-5 SDO Abort Codes* on page A-36 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 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 the execution.
- 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 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The EtherCAT master is not in a state that allows message communications.
  - b) The slave specified with *NodeAdr* does not exist.
  - c) The slave specified with *NodeAdr* is not in a state that allows communications.
  - d) The slave returns an error response.
  - e) 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\_CoESDORRead

The EC\_CoESDORRead instruction reads a value from a CoE (CAN Application Protocol over EtherCAT) object of a specified slave on the EtherCAT network.

| Instruction    | Name                  | FB/<br>FUN | Graphic expression | ST expression   |
|----------------|-----------------------|------------|--------------------|---|
| EC_CoESDORRead | Read EtherCAT CoE SDO | FB         |                    | EC_CoESDORRead_instance(Execute, NodeAdr, SdoObj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, AbortCode, ReadSize); |

## Variables

|           | Meaning            | I/O    | Description  | Valid range            | Unit  | Default   |
|-----------|--------------------|--------|--|------------------------|-------|-----------|
| NodeAdr   | Slave node address | Input  | Node address of the slave to access  | 1 to 512 <sup>*1</sup> | ---   | ---       |
| SdoObj    | SDO parameter      |        | SDO parameter  | ---                    |       |           |
| TimeOut   | Timeout time       |        | 0: 2.0 s<br>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<br>0: Normal end             | Depends on data type.  | ---   | ---       |
| ReadSize  | Read data size     |        | Size of data stored in <i>ReadDat</i> after the data is read <sup>*2</sup> |                        | Bytes |           |
| ReadDat   | Read data          | In-out | Read data buffer   | Depends on data type.  | ---   | ---       |

\*1. The range is 1 to 192 for the NX102 CPU Unit, NX1P2 CPU Unit, and 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.

|           | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-990 for details on the structure <i>_sSDO_ACCESS</i> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| TimeOut   |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| AbortCode |   |             |      | OK    |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| ReadSize  |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |

|  | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|--|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|  | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT                 | REAL | LREAL  | TIME | DATE | TOD | DT | STRING |
| 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*. The size of the stored data 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  | <code>_sSDO_ACCESS</code> | ---                   | ---  | ---     |
| 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<br>TRUE: Access data for all subindexes<br>FALSE: Access data for the specified subindex | BOOL                      |                       |      |         |

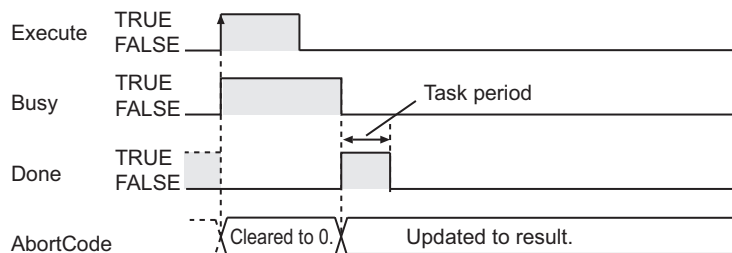
After the read is completed, the instruction waits for the response for the period of time specified with *TimeOut*.

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 value and meaning 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  |
|--|--|-----------|--|
| <code>_EC_MBXSlavTbl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>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)* for details on EtherCAT communications.
- Refer to *A-5 SDO Abort Codes* on page A-36 for the SDO abort codes.

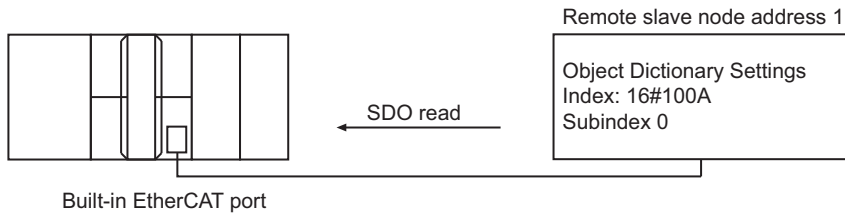
## Precautions for Correct Use

- 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 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, and an error code is assigned to *ErrorID*.
  - 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*.



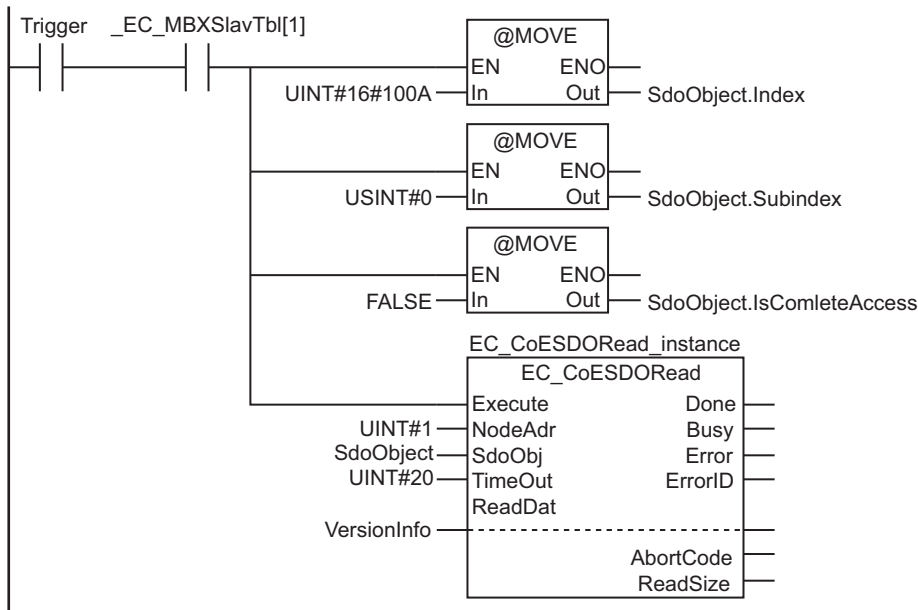
**LD**

| 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_CoESDORead_instance | EC_CoESDORead |  |                     |

| External Variables | Variable       | Data type               | Constant                            | Comment                                    |
|--------------------|----------------|-------------------------|-------------------------------------|--|
|                    | _EC_MBXSlavTbl | ARRAY[1..512] OF BOOL*1 | <input checked="" type="checkbox"/> | Message Communications Enabled Slave Table |

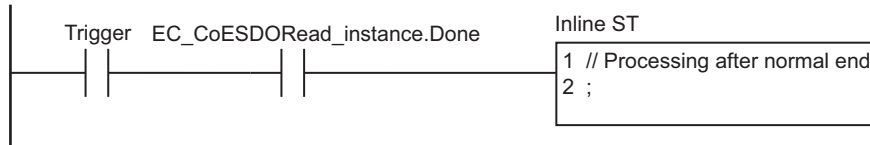
\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

Accept trigger.

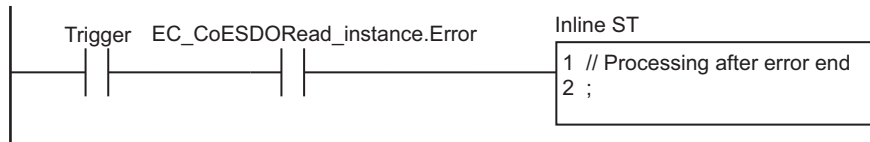




Processing after normal end



Processing after error end



## 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. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

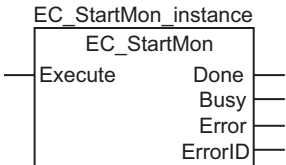
```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSdoRead=FALSE) AND (_EC_MBXSlavTbl[1]=TRUE) ) THEN
  DoSdoRead :=TRUE;
  SdoObject.Index :=UINT#16#100A;
  SdoObject.Subindex :=USINT#0;
  SdoObject.IsCompleteAccess:=FALSE;
  EC_CoESDORead_instance(
    Execute:=FALSE, // Initialize 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 packet monitoring for EtherCAT communications.

| Instruction | Name                          | FB/<br>FUN | Graphic expression  | ST expression  |
|-------------|-------------------------------|------------|---|--|
| EC_StartMon | Start EtherCAT Packet Monitor | FB         |  | EC_StartMon_instance(Execute, Done, Busy, Error, ErrorID); |



## Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- You cannot use this instruction for project unit version 1.40 or later.
- 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.

## 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.<br>TRUE: Stopped.<br>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.<br>TRUE: Saving.<br>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.
- To save packet data in an internal file in the main memory of the CPU Unit, execute the `EC_StopMon` instruction to stop packet monitoring, and 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)* 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 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 will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) Project unit version 1.40 or later is used.
  - b) A packet data save operation to an internal file in the main memory of the CPU Unit is in progress.
  - c) 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 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:

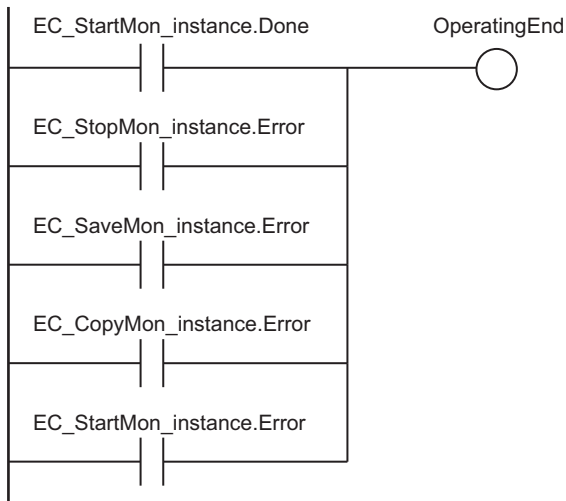
- 1** The system-defined variable `_EC_ErrSta` (EtherCAT Error) is monitored and processing is started if an error occurs.
- 2** The `EC_StopMon` instruction is used to stop execution of packet monitoring for EtherCAT communications.
- 3** The `EC_SaveMon` instruction is used to save EtherCAT communications 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

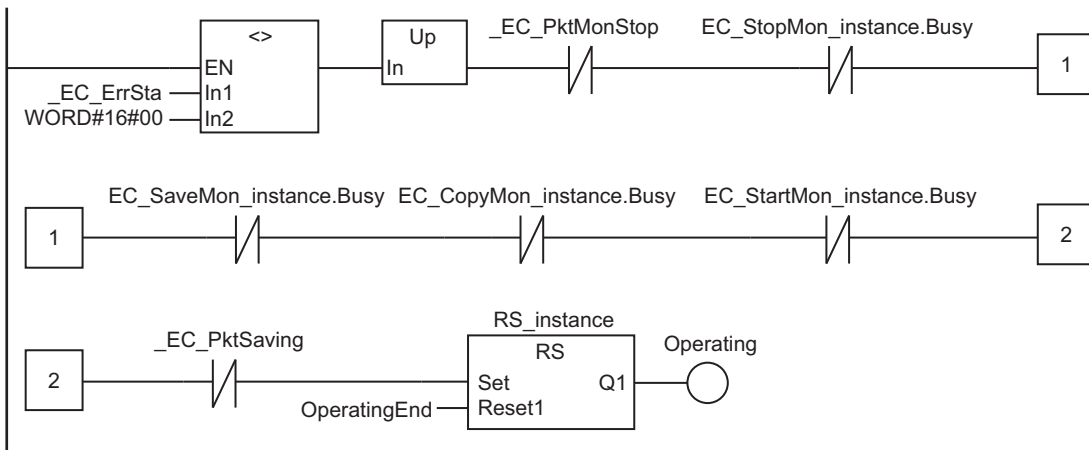
| Internal Variables | Variable             | Data type   | Initial value | Comment              |
|--------------------|----------------------|-------------|---------------|----------------------|
|                    | OperatingEnd         | BOOL        | FALSE         | Processing completed |
|                    | 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      | <input checked="" type="checkbox"/> | Built-in EtherCAT Error   |
|                    | _EC_PktMonStop | BOOL      | <input checked="" type="checkbox"/> | Packet Monitoring Stopped |
|                    | _EC_PktSaving  | BOOL      | <input checked="" type="checkbox"/> | Saving Packet Data File   |
|                    | _Card1Ready    | BOOL      | <input checked="" type="checkbox"/> | SD Memory Card Ready Flag |

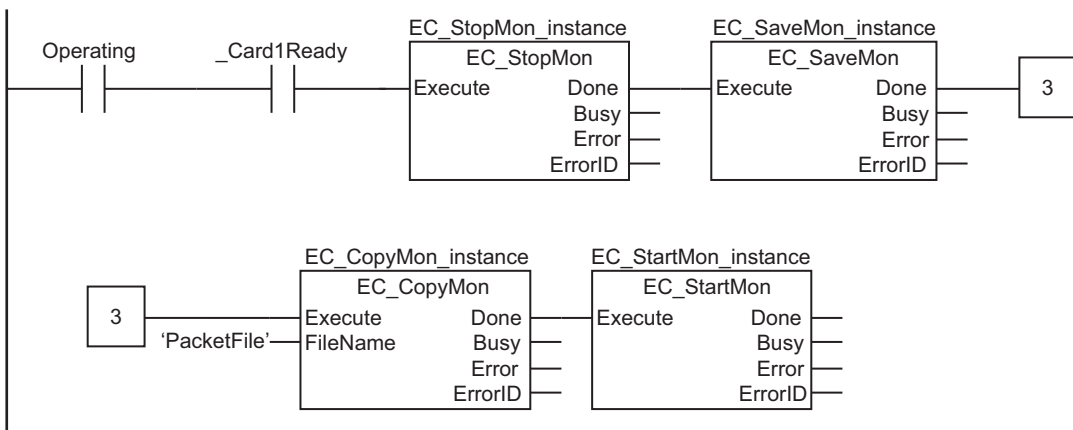
Determine if instruction execution is completed.



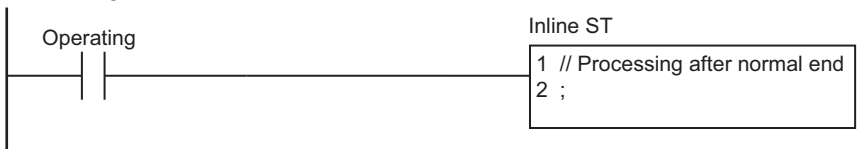
Monitor for EtherCAT errors.



Instruction execution



Processing after normal end



## 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 |               |  |

| Internal Variables | Variable             | Data type   | Initial value | Comment |
|--------------------|----------------------|-------------|---------------|---------|
|                    | EC_StartMon_instance | EC_StartMon |               |         |

| External Variables | Variable       | Data type | Constant                            | Comment                   |
|--------------------|----------------|-----------|-------------------------------------|---------------------------|
|                    | _EC_ErrSta     | WORD      | <input checked="" type="checkbox"/> | Built-in EtherCAT Error   |
|                    | _EC_PktMonStop | BOOL      | <input checked="" type="checkbox"/> | Packet Monitoring Stopped |
|                    | _EC_PktSaving  | BOOL      | <input checked="" type="checkbox"/> | Saving Packet Data File   |
|                    | _Card1Ready    | BOOL      | <input checked="" type="checkbox"/> | 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/<br>FUN | Graphic expression | ST expression   |
|-------------|------------------------------|------------|--------------------|---|
| EC_StopMon  | Stop EtherCAT Packet Monitor | FB         |                    | EC_StopMon_instance(Execute, Done, Busy, Error, ErrorID); |



## Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- You cannot use this instruction for project unit version 1.40 or later.
- 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.

## 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.<br>TRUE: Stopped.<br>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.<br>TRUE: Saving.<br>FALSE: Not saving. |

### Additional Information

- When you save collected packet data in an internal file in the main memory of the CPU Unit, you need to execute this instruction to stop the packet monitoring function, and then execute the EC\_SaveMon instruction to save the data.
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual (Cat. No. W505)* for details on EtherCAT communications.

### Precautions for Correct Use

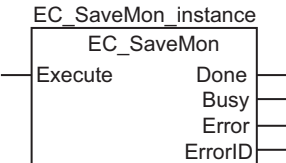
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using 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 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 will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) Project unit version 1.40 or later is used.
  - b) Packet monitoring is already stopped.
  - c) 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

Refer to *Sample Programming* on page 2-996 for the EC\_StartMon instruction.

# 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/<br>FUN | Graphic expression  | ST expression   |
|-------------|-----------------------|------------|---|---|
| EC_SaveMon  | Save EtherCAT Packets | FB         |  | EC_SaveMon_instance(Execute, Done, Busy, Error, ErrorID); |



## Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- You cannot use this instruction for project unit version 1.40 or later.
- 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.

## 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.<br>TRUE: Stopped.<br>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.<br>TRUE: Saving.<br>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)* 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 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\_WriteObj*.
- An error will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) Project unit version 1.40 or later is used.
  - b) Packet monitoring is in progress.
  - c) More than 32 of the following instructions are 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

Refer to *Sample Programming* on page 2-996 for the *EC\_StartMon* instruction.

# EC\_CopyMon

The EC\_CopyMon instruction transfers packet data in an internal file in the main memory of the CPU Unit to the SD Memory Card.

| Instruction | Name                       | FB/<br>FUN | Graphic expression | ST expression   |
|-------------|----------------------------|------------|--------------------|---|
| EC_CopyMon  | Transfer Ether-CAT Packets | FB         |                    | EC_CopyMon_instance(Execute, FileName, Done, Busy, Error, ErrorID); |



## Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- You cannot use this instruction for project unit version 1.40 or later.
- 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.

## Variables

|          | Meaning   | I/O   | Description                     | Valid range           | Unit | Default |
|----------|-----------|-------|---------------------------------|-----------------------|------|---------|
| FileName | File name | Input | File name on the SD Memory Card | Depends on data type. | ---  | ---     |

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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.<br>TRUE: Saving.<br>FALSE: Not saving. |

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual (Cat. No. W505)* 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 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 will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) Project unit version 1.40 or later is used.
  - b) A packet data file save operation is in progress.
  - c) 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

Refer to *Sample Programming* on page 2-996 for the EC\_StartMon instruction.

# EC\_DisconnectSlave

The EC\_DisconnectSlave instruction disconnects the specified slave from the EtherCAT network.

| Instruction        | Name                      | FB/<br>FUN | Graphic expression  | ST expression   |
|--------------------|---------------------------|------------|---|---|
| EC_DisconnectSlave | Disconnect EtherCAT Slave | FB         | <pre> graph LR     subgraph EC_DisconnectSlave_instance [EC_DisconnectSlave_instance]         EC_DisconnectSlave[EC_DisconnectSlave]     end     Execute[Execute] --&gt; EC_DisconnectSlave     NodeAdr[NodeAdr] --&gt; EC_DisconnectSlave     EC_DisconnectSlave --&gt; Done[Done]     EC_DisconnectSlave --&gt; Busy[Busy]     EC_DisconnectSlave --&gt; Error[Error]     EC_DisconnectSlave --&gt; ErrorID[ErrorID] </pre> | EC_DisconnectSlave_instance( Execute, NodeAdr, Done, Busy, Error, ErrorID); |

## Variables

|         | Meaning            | I/O   | Description                             | Valid range | Unit | Default |
|---------|--------------------|-------|---|-------------|------|---------|
| NodeAdr | Slave node address | Input | Node address of the slave to disconnect | 1 to 512*1  | ---  | ---     |

\*1. The range is 1 to 192 for the NX102 CPU Unit, NX1P2 CPU Unit, and 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]<br>"i" is the node address.   | Network Connected Slave Table | BOOL[]    | This variable shows if slaves are part of (i.e., exist on) the network.<br>TRUE: Part of the network.<br>FALSE: Not part of the network.   |
| _EC_DisconnSlavTbl[i]<br>"i" is the node address. | Disconnected Slave Table      | BOOL[]    | This variable shows the slaves for which there are currently disconnect commands in effect.<br>TRUE: Disconnect command is in effect.<br>FALSE: Disconnect command is not in effect. |

| Name   | Meaning              | Data type | Description  |
|--|----------------------|-----------|--|
| <code>_EC_DisableSlavTbl[i]</code><br>"i" is the node address. | Disabled Slave Table | BOOL[]    | This variable shows if slaves are disabled on the network.<br>TRUE: Disabled.<br>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)* 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 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`.
- You cannot specify slaves in a ring topology network to disconnect. However, slaves on a drop line from the ring can be specified and disconnected.
- An error occurs in the following case. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) 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.
  - b) The slave specified with *NodeAdr* is disabled.
  - c) The `EC_DisconnectSlave`, `EC_ConnectSlave`, `EC_ChangeEnableSetting`, `ResetECError`, `RestartNXUnit`, or `NX_ChangeWriteMode` instruction is already in execution.
  - d) 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.

This allows this program 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 instructions.

You cannot execute the EC\_ChangeEnableSetting instruction at the same time as the EC\_DisconnectSlave or EC\_ConnectSlave instruction.

The same global variable, *ExclusiveFlg*, is used in *Sample Programming* on page 2-1020 for the EC\_ChangeEnableSetting instruction to explain exclusive control for instructions.

## Definitions of Global Variables

### ● Global Variables

| Variable     | Data type | Initial value | Comment                    |
|--------------|-----------|---------------|----------------------------|
| ExclusiveFlg | BOOL      | FALSE         | Instruction Exclusive Flag |

## LD

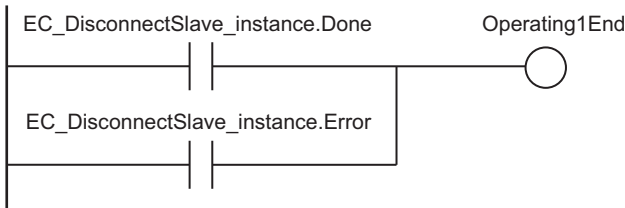
| Internal Variables | Variable                    | Data type          | Initial value | Comment                 |
|--------------------|-----------------------------|--------------------|---------------|-------------------------|
|                    | Operating1End               | BOOL               | 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    |               |                         |
|                    | HMI_ConnectErrorID*1        | WORD               | 16#0000       |                         |

\*1. The variables that begin with *HMI\_* are variables for display on a touch panel.

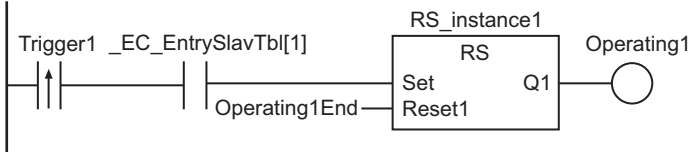
| External Variables | Variable           | Data type               | Constant | Comment                       |
|--------------------|--------------------|-------------------------|----------|-------------------------------|
|                    | _EC_EntrySlavTbl   | ARRAY[1..512] OF BOOL*1 | ☑        | Network Connected Slave Table |
|                    | _EC_DisconnSlavTbl | ARRAY[1..512] OF BOOL*1 | ☑        | Disconnected Slave Table      |
|                    | ExclusiveFlg       | BOOL                    | ☐        | Instruction Exclusive Flag    |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

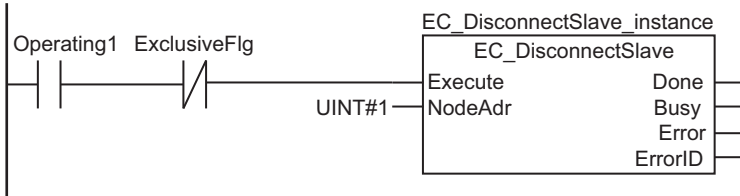
Determine if execution of the EC\_DisconnectSlave instruction is completed.



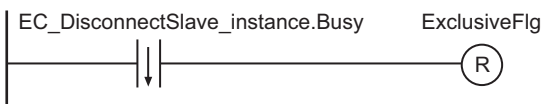
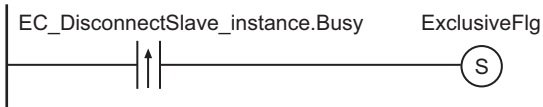
Accept trigger 1.



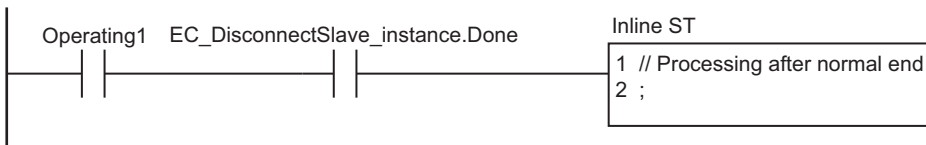
Execute EC\_DisconnectSlave instruction.



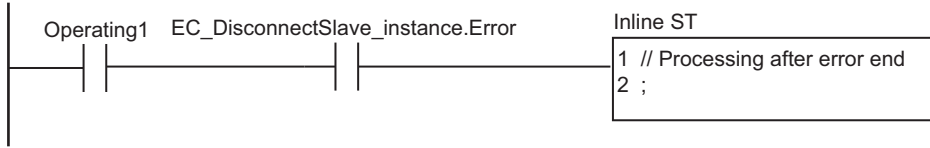
Exclusive control of instructions



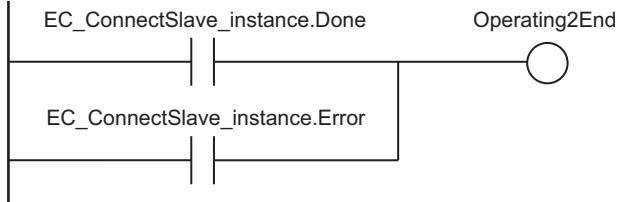
Processing after normal end



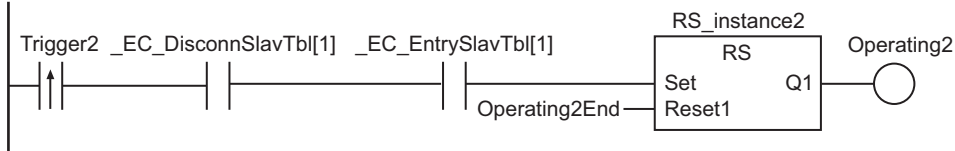
Processing after error end



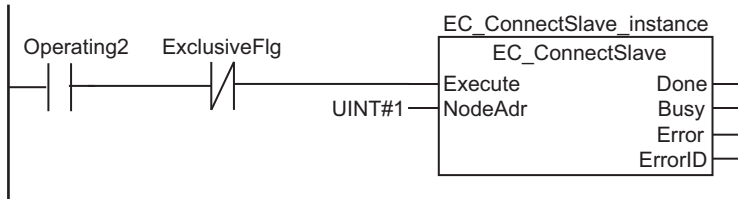
Determine if execution of the EC\_ConnectSlave instruction is completed.



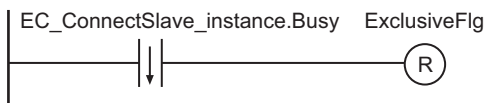
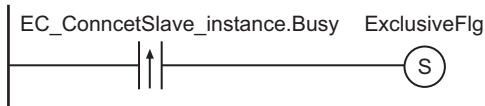
Accept trigger 2.



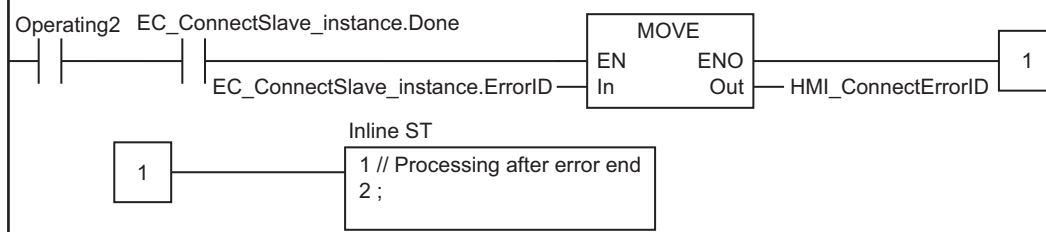
Execute EC\_ConnectSlave instruction.

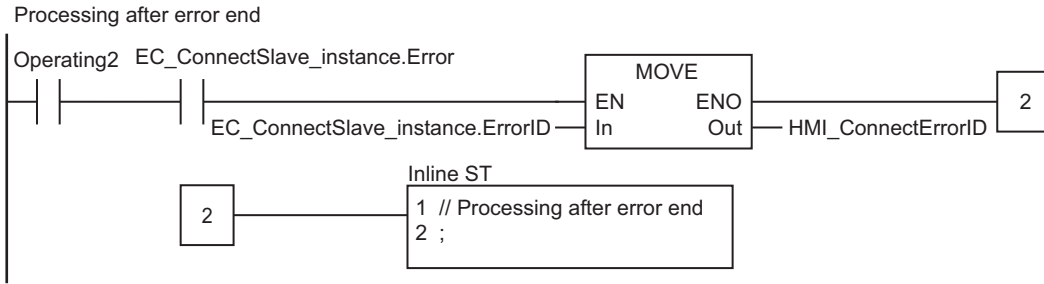


Exclusive control of instructions



Processing after normal end





**ST**

| Internal Variables | Variable                    | Data type          | Initial value | Comment  |
|--------------------|-----------------------------|--------------------|---------------|--|
|                    | Trigger1                    | BOOL               | FALSE         | Execution condition 1  |
|                    | LastTrigger1                | BOOL               | FALSE         | Value of <i>Trigger1</i> 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 <i>Trigger2</i> 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                 |               |  |
|                    | HMI_ConnectErrorID*1        | WORD               | 16#0000       |  |

\*1. The variables that begin with *HMI\_* are variables for display on a touch panel.

| External Variables | Variable         | Data type               | Constant                            | Comment                       |
|--------------------|------------------|-------------------------|-------------------------------------|-------------------------------|
|                    | _EC_EntrySlavTbl | ARRAY[1..512] OF BOOL*1 | <input checked="" type="checkbox"/> | Network Connected Slave Table |

| External Variables | Variable           | Data type                  | Constant | Comment                    |
|--------------------|--------------------|----------------------------|----------|----------------------------|
|                    | _EC_DisconnSlavTbl | ARRAY[1..512] OF<br>BOOL*1 | ☑        | Disconnected Slave Table   |
|                    | ExclusiveFlg       | BOOL                       |          | Instruction Exclusive Flag |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

```
// 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_instancel(EC_DisconnectSlave_instance.Busy, DisconnectSet);
F_TRIG_instancel(EC_DisconnectSlave_instance.Busy, DisconnectReset);
RS_instancel(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) AND
(_EC_EntrySlavTbl[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
        HMI_ConnectErrorID:=EC_ConnectSlave_instance.ErrorID;
        Operating2:=FALSE;
    END_IF;

    IF (EC_ConnectSlave_instance.Error=TRUE) THEN
        // Processing after error end
        HMI_ConnectErrorID:=EC_ConnectSlave_instance.ErrorID;
        Operating2:=FALSE;
    END_IF;
END_IF;
```

# EC\_ConnectSlave

The EC\_ConnectSlave instruction connects the specified slave to the EtherCAT network.

| Instruction     | Name                   | FB/<br>FUN | Graphic expression | ST expression   |
|-----------------|------------------------|------------|--------------------|---|
| EC_ConnectSlave | Connect EtherCAT Slave | FB         |                    | EC_ConnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID); |

## Variables

|         | 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 0 to 192 for the NX102 CPU Unit, NX1P2 CPU Unit, and 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

|         | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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  |
|--|-------------------------------|-----------|--|
| <code>_EC_EntrySlavTbl[i]</code><br>"i" is the node address.   | Network Connected Slave Table | BOOL[]    | This variable shows if slaves are part of (i.e., exist on) the network.<br>TRUE: Part of the network.<br>FALSE: Not part of the network.   |
| <code>_EC_DisconnSlavTbl[i]</code><br>"i" is the node address. | Disconnected Slave Table      | BOOL[]    | This variable shows the slaves for which there are currently disconnect commands in effect.<br>TRUE: Disconnect command is in effect.<br>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)* 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 EtherCAT ports.
- 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 cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) 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.
  - b) The `EC_DisconnectSlave`, `EC_ConnectSlave`, `EC_ChangeEnableSetting`, `ResetECError`, `RestartNXUnit`, or `NX_ChangeWriteMode` instruction is already in execution.
  - c) 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

Refer to *Sample Programming* on page 2-1008 for the `EC_DisconnectSlave` instruction.



# EC\_ChangeEnableSetting

The EC\_ChangeEnableSetting instruction enables or disables an EtherCAT slave.

| Instruction            | Name                          | FB/<br>FUN | Graphic expression | ST expression  |
|------------------------|-------------------------------|------------|--------------------|--|
| EC_ChangeEnableSetting | Enable/Disable EtherCAT Slave | FB         |                    | EC_ChangeEnableSetting_instance(Execute, NodeAdr, IsEnable, Done, Busy, Error, ErrorID); |



## 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.

## Variables

|          | 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 <sup>*1</sup> | ---  | 1       |
| IsEnable | Enable/disable designation |       | Designation of whether to enable or disable the specified EtherCAT slave<br>TRUE: Enable<br>FALSE: Disable | Depends on data type.  |      | TRUE    |

\*1. The range is 1 to 192 for the NX102 CPU Unit, NX1P2 CPU Unit, and 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. *Done* changes to TRUE when this instruction is successfully completed.

Enabling or disabling the slave is completed when the instruction is completed normally.

The instruction may not be successfully completed, depending on the status of the specified EtherCAT slave: whether the specified EtherCAT slave is enabled or disabled, connected or disconnected, and present or not present in the EtherCAT network.

The following table shows how the EtherCAT slave status changes after this instruction is executed.

| Status before instruction execution |                        |                       | Value of <i>IsEnable</i> | Status after instruction execution |                        |
|-------------------------------------|------------------------|-----------------------|--------------------------|------------------------------------|------------------------|
| Enabled/disabled                    | Connected/disconnected | Present* <sup>1</sup> |                          | Normal/error end                   | Enabled/disabled       |
| Enabled                             | Connected              | Yes                   | TRUE (Enabled)           | Normal end                         | Enabled                |
|                                     | Disconnected           | Yes                   |                          | Error end* <sup>2</sup>            | Enabled                |
|                                     |                        | No                    |                          |                                    |                        |
| Disabled                            | ---* <sup>3</sup>      | Yes                   |                          | Normal end                         | Enabled                |
|                                     |                        | No                    |                          | Error end* <sup>4</sup>            | Disabled* <sup>4</sup> |
| Enabled                             | Connected              | Yes                   |                          | FALSE (Disabled)                   | Normal end             |
|                                     | Disconnected           | Yes                   | Error end* <sup>2</sup>  |                                    | Enabled                |
|                                     |                        | No                    |                          |                                    |                        |
| Disabled                            | ---* <sup>3</sup>      | Yes                   | Normal end               |                                    | Disabled               |
|                                     |                        | No                    | Error end* <sup>4</sup>  |                                    | Disabled* <sup>4</sup> |

\*1. This indicates whether the specified EtherCAT slave is physically connected to the EtherCAT network.  
Yes: Physically connected. No: Not physically connected.

\*2. For project unit version 1.40 or later, Error code 180A is returned.  
For project unit version earlier than 1.40, Error code 1800 is returned.

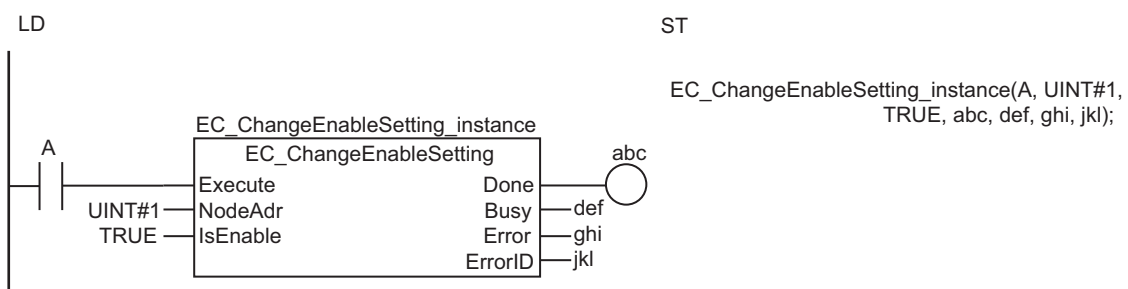
\*3. EtherCAT slaves that are disabled are not considered to be either connected or disconnected.

\*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.

### ● Application Example

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*.



### Related System-defined Variables

| Name   | Meaning                       | Data type | Description  |
|--|-------------------------------|-----------|--|
| <code>_EC_EntrySlavTbl[i]</code><br>"i" is the node address. | Network Connected Slave Table | BOOL[]    | This variable shows if slaves are part of (i.e., exist on) the network.<br>TRUE: Part of the network.<br>FALSE: Not part of the network. |

| Name   | Meaning                  | Data type | Description  |
|--|--------------------------|-----------|--|
| <code>_EC_DisconnSlavTbl[i]</code><br>"i" is the node address. | Disconnected Slave Table | BOOL[]    | This variable shows the slaves for which there are currently disconnect commands in effect.<br>TRUE: Disconnect command is in effect.<br>FALSE: Disconnect command is not in effect. |
| <code>_EC_DisableSlavTbl[i]</code><br>"i" is the node address. | Disabled Slave Table     | BOOL[]    | This variable shows if slaves are disabled on the network.<br>TRUE: Disabled.<br>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)* for details on EtherCAT communications.
- Use *EC\_ConnectSlave* on page 2-1015 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 EtherCAT ports.
- You cannot execute this instruction during execution of the following instructions: *EC\_DisconnectSlave*, *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\_WriteObj*.
- An error will occur in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID*.
  - a) 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.
  - b) The value of *NodeAdr* is outside the valid range.
  - c) The *EC\_DisconnectSlave*, *EC\_ConnectSlave*, *EC\_ChangeEnableSetting*, *ResetECError*, *RestartNXUnit*, or *NX\_ChangeWriteMode* instruction is already in execution.
  - d) 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*.
  - e) The instruction is executed for a slave in a ring topology network.  
This will result in an error, regardless of whether the network is in cable redundancy status or ring disconnection status when the instruction is executed.

## Sample Programming

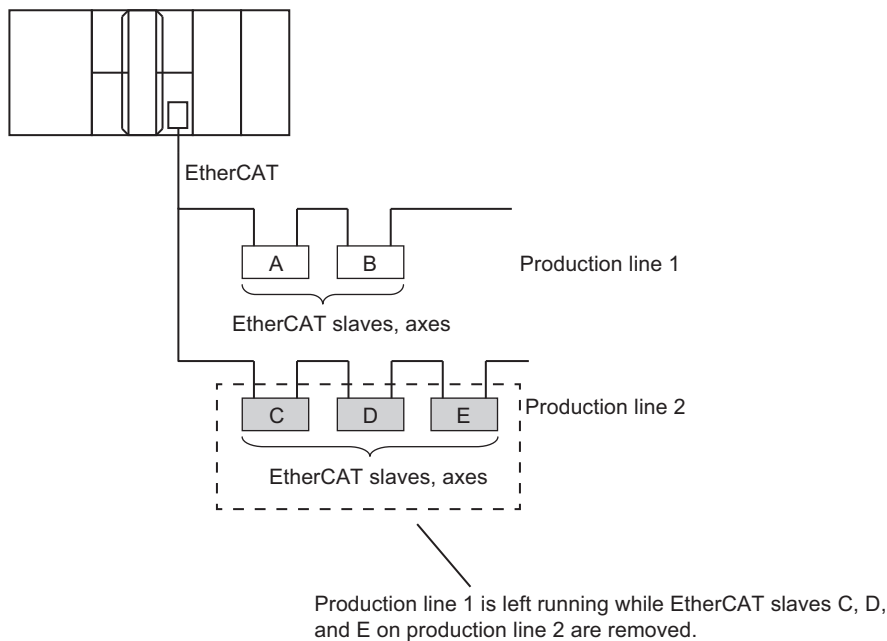
This section provides the following two examples for explanation.

- Example of disconnecting EtherCAT slaves from the EtherCAT network
- Example of connecting EtherCAT slaves to an EtherCAT network

### 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)* 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 instructions.

You cannot execute the EC\_ChangeEnableSetting instruction at the same time as the EC\_DisconnectSlave or EC\_ConnectSlave instruction.

The same global variable, *ExclusiveFlg*, is used in *Sample Programming* on page 2-1008 for the EC\_DisconnectSlave instruction to explain 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            |

## ● Global Variables

| Variable     | Data type  | Initial value | AT specification | Constant                            | Comment                            |
|--------------|------------|---------------|------------------|-------------------------------------|------------------------------------|
| MC_Axis000   | _sAXIS_REF |               | MC://_MC_AX[0]   | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave C |
| MC_Axis001   | _sAXIS_REF |               | MC://_MC_AX[1]   | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave D |
| MC_Axis002   | _sAXIS_REF |               | MC://_MC_AX[2]   | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave E |
| ExclusiveFlg | BOOL       | FALSE         |                  | <input type="checkbox"/>            | Instruction Exclusive Flag         |

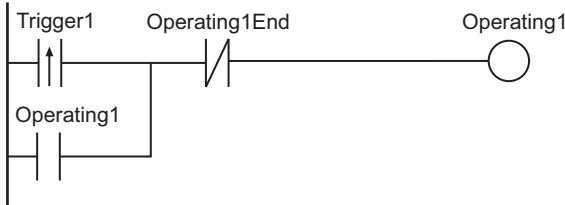
## ● LD

| Internal Variables | Variable                    | Data type              | Initial value | Comment   |
|--------------------|-----------------------------|------------------------|---------------|---|
|                    | Operating1End               | BOOL                   | FALSE         | Processing completed  |
|                    | Trigger1                    | BOOL                   | FALSE         | Execution condition   |
|                    | Operating1                  | BOOL                   | FALSE         | Processing  |
|                    | AxisUnuseDone_DevC          | BOOL                   | FALSE         | Changing axis to unused axis completed for EtherCAT slave C |
|                    | SlaveDisableDone_DevC       | BOOL                   | FALSE         | Disabling EtherCAT slave C completed                        |
|                    | 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 completed                        |
|                    | 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 completed                        |
|                    | 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_ChangeEnableSetting_DevC | EC_ChangeEnableSetting |               |   |
|                    | MC_ChangeAxisUse_DevD       | MC_ChangeAxisUse       |               |   |
|                    | EC_ChangeEnableSetting_DevD | EC_ChangeEnableSetting |               |   |
|                    | MC_ChangeAxisUse_DevE       | MC_ChangeAxisUse       |               |   |
|                    | EC_ChangeEnableSetting_DevE | EC_ChangeEnableSetting |               |   |

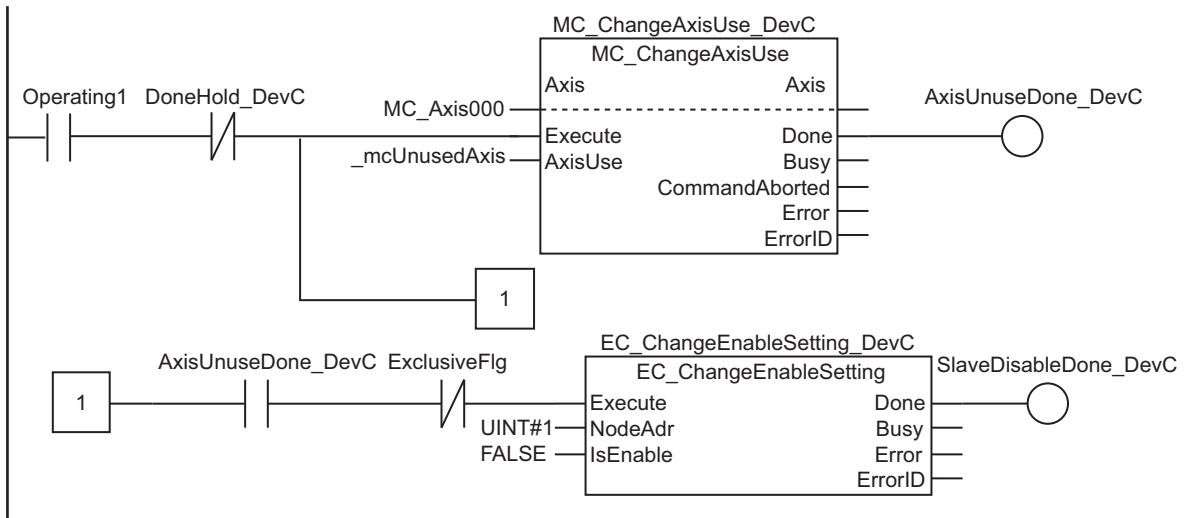
| External Variables | Variable   | Data type  | Constant | Comment                            |
|--------------------|------------|------------|----------|------------------------------------|
|                    | MC_Axis000 | _sAXIS_REF | ☑        | Axis variable for EtherCAT slave C |
|                    | MC_Axis001 | _sAXIS_REF | ☑        | Axis variable for EtherCAT slave D |
|                    | MC_Axis002 | _sAXIS_REF | ☑        | Axis variable for EtherCAT slave E |

| External Variables | Variable     | Data type | Constant | Comment                    |
|--------------------|--------------|-----------|----------|----------------------------|
|                    | ExclusiveFlg | BOOL      |          | Instruction Exclusive Flag |

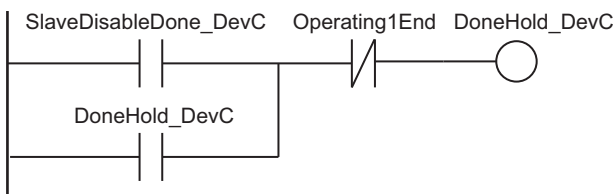
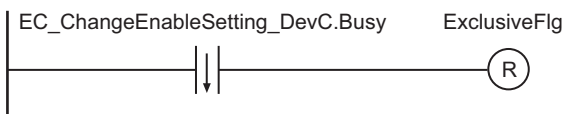
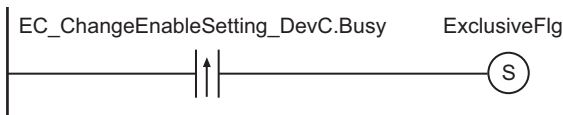
Accept execution condition trigger.



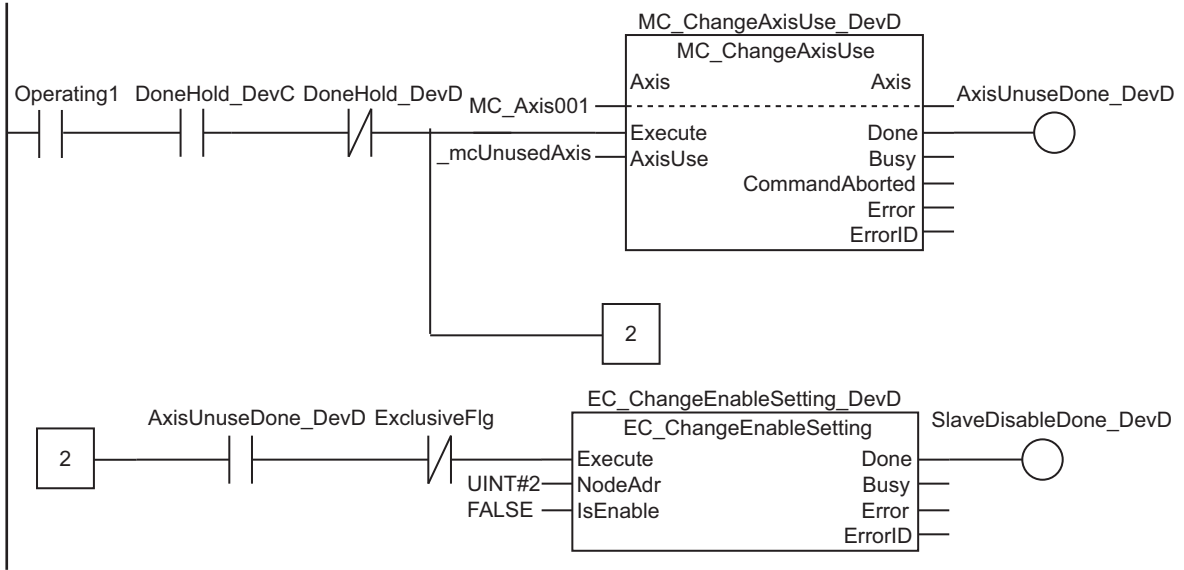
Change axis to unused axis and disable EtherCAT slave C.



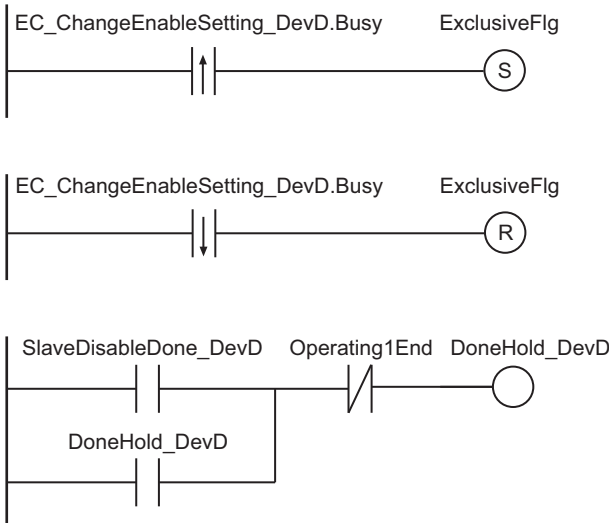
Exclusive control of instructions



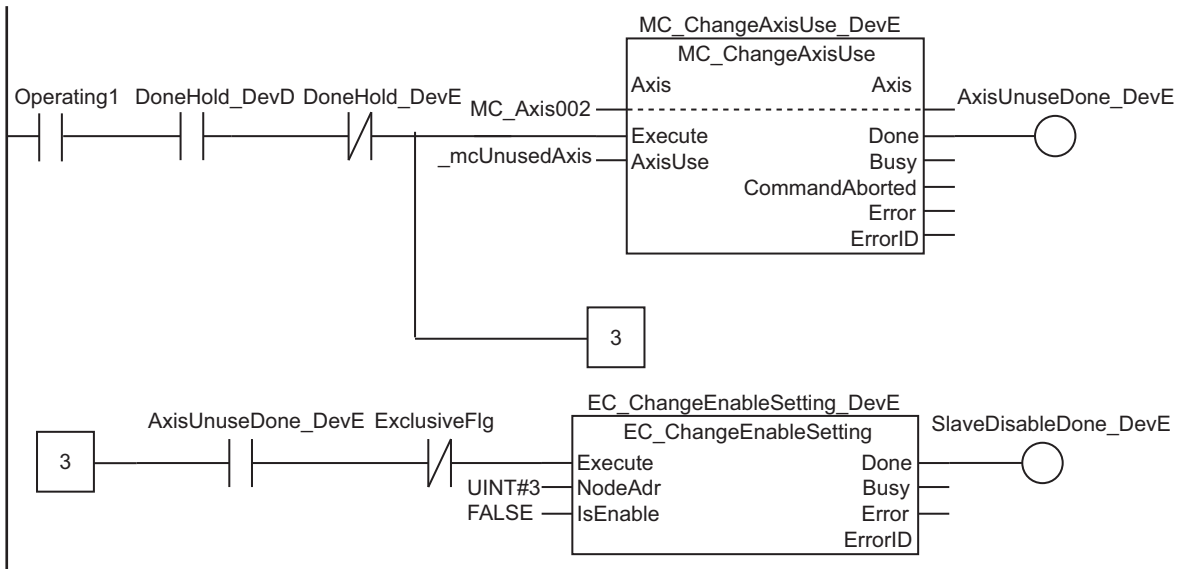
Change axis to unused axis and disable EtherCAT slave D.



Exclusive control of instructions

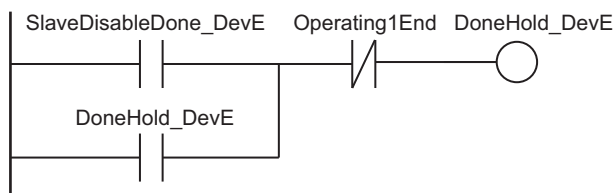
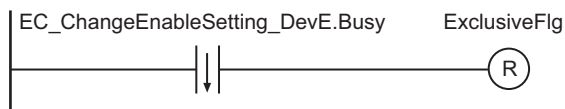


Change axis to unused axis and disable EtherCAT slave E.

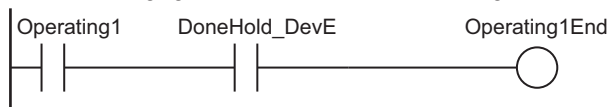




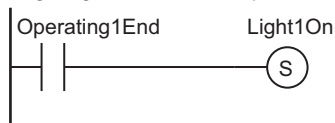
Exclusive control of instructions



Confirm changing axis to unused axis and disabling EtherCAT slave E.



Lighting removal OK lamp



## ● ST

| Internal Variables | 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 |               |   |

| Internal Variables | Variable                    | Data type              | Initial value | Comment |
|--------------------|-----------------------------|------------------------|---------------|---------|
|                    | 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 | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave C |
|                    | MC_Axis001   | _sAXIS_REF | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave D |
|                    | MC_Axis002   | _sAXIS_REF | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave E |
|                    | ExclusiveFlg | BOOL       | <input type="checkbox"/>            | 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(Exclusiv
eFlg)),
  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);

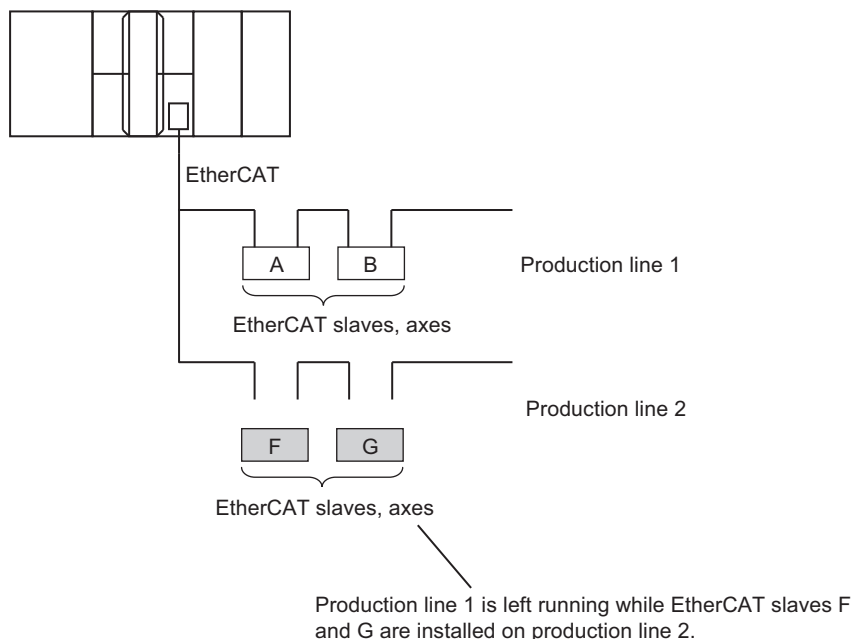
// Lighting removal OK lamp
SR_instance1(
  Set1:=Operating1End,
  Q1 =>Light1On);
```

### 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)* 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.

*ExclusiveFlg* is defined as a global variable in this sample program.

That allows this sample programming to perform exclusive control with EC\_ChangeEnableSetting instructions in the other tasks.

In this case, however, the same global variable (*ExclusiveFlg*) must also be used in the other tasks to perform exclusive control of instructions.

You cannot execute the EC\_ChangeEnableSetting instruction at the same time as the EC\_DisconnectSlave or EC\_ConnectSlave instruction.

The same global variable, *ExclusiveFlg*, is used in *Sample Programming* on page 2-1008 for the EC\_DisconnectSlave instruction to explain 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            |

### ● Global Variables

| Variable     | Data type  | Initial value | AT specification | Constant                            | Comment                            |
|--------------|------------|---------------|------------------|-------------------------------------|------------------------------------|
| MC_Axis003   | _sAXIS_REF |               | MC://_MC_AX[3]   | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave F |
| MC_Axis004   | _sAXIS_REF |               | MC://_MC_AX[4]   | <input checked="" type="checkbox"/> | Axis variable for EtherCAT slave G |
| ExclusiveFlg | BOOL       | FALSE         |                  | <input type="checkbox"/>            | Instruction Exclusive Flag         |

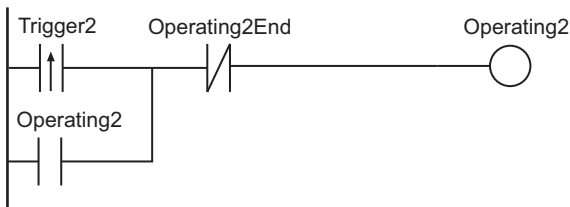
### ● LD

| 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 |

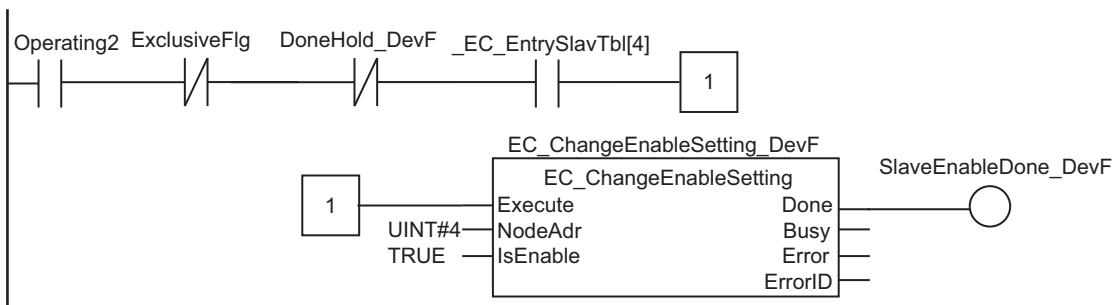
| Internal Variables | Variable                    | Data type              | Initial value | Comment   |
|--------------------|-----------------------------|------------------------|---------------|---|
|                    | SlaveEnableDone_DevG        | BOOL                   | FALSE         | Enabling 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.

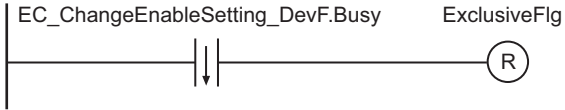


Enable EtherCAT slave F.

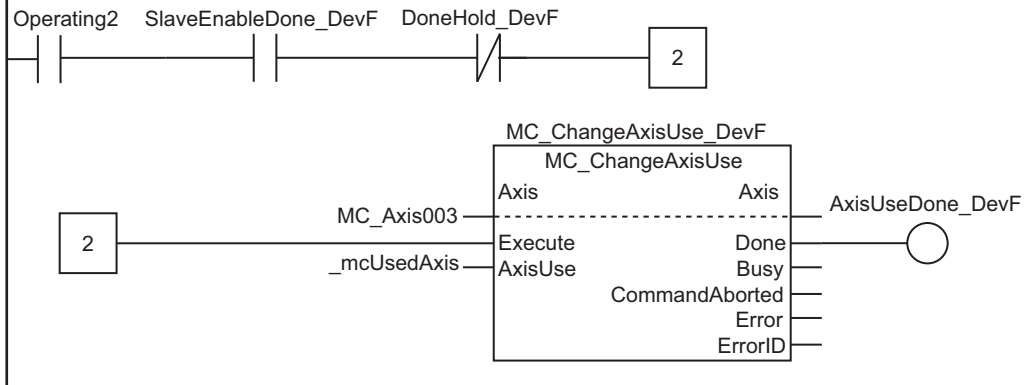


Exclusive control of instructions. Start enabling EtherCAT slave F and confirm completion.

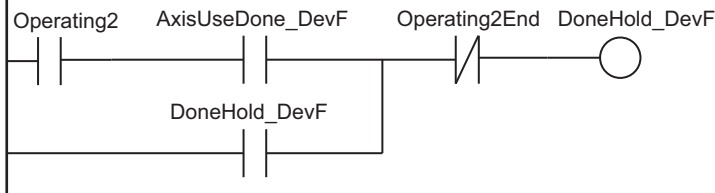




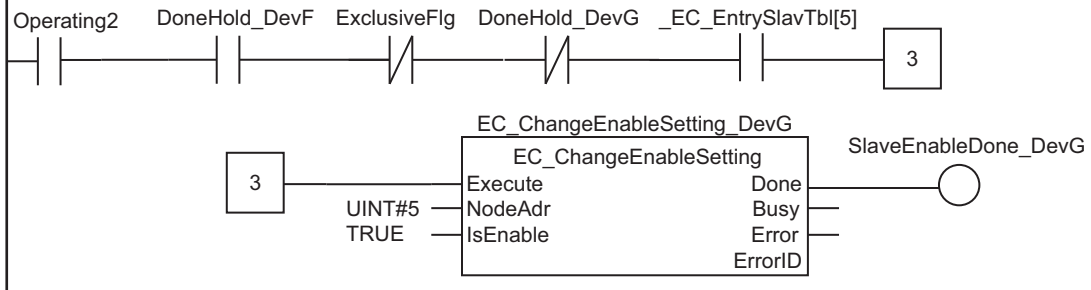
Change axis to used axis for EtherCAT slave F.



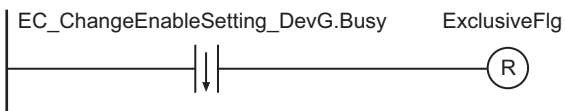
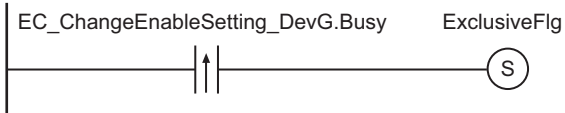
Exclusive control of instructions. Confirm that all processing for EtherCAT slave F is completed.



Enable EtherCAT slave G.

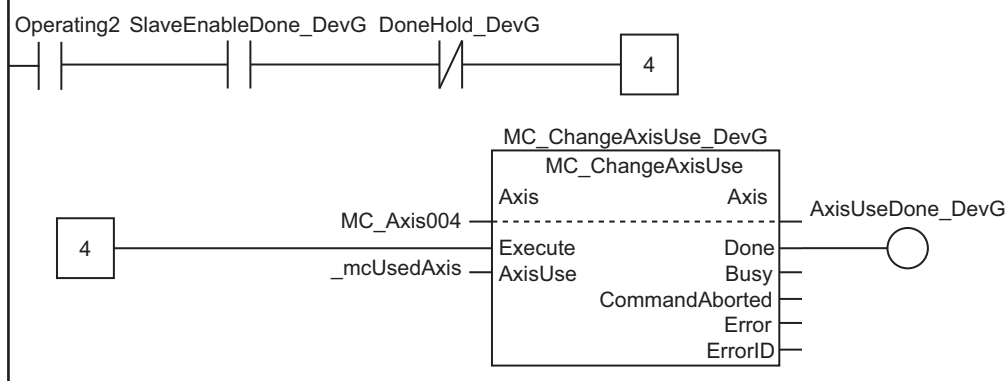


Exclusive control of instructions. Start enabling EtherCAT slave G and confirm completion.

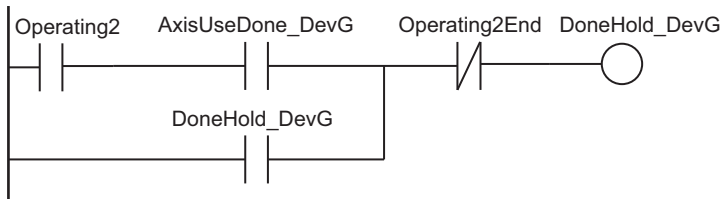




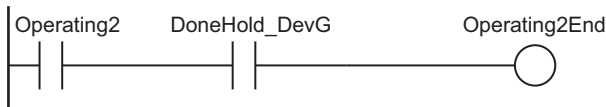
Change axis to used axis for EtherCAT slave G.



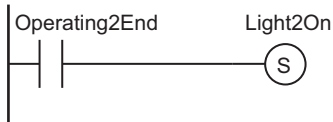
Exclusive control of instructions. Confirm that all processing for EtherCAT slave G is completed.



Confirm completion of processing for EtherCAT slave G



Lighting installation completed lamp



● ST

| Internal Variables | Variable        | Data type | Initial value | Comment   |
|--------------------|-----------------|-----------|---------------|---|
|                    | Operating2End   | BOOL      | FALSE         | Processing completed                                  |
|                    | 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 Exclusive Flag ON                         |

| Internal Variables | Variable                    | Data type              | Initial value | Comment                        |
|--------------------|-----------------------------|------------------------|---------------|--------------------------------|
|                    | ExclusiveFlgReset           | BOOL                   | FALSE         | Instruction Exclusive Flag OFF |
|                    | R_TRIG_instance2            | R_TRIG                 |               |                                |
|                    | RS_instance2                | RS                     |               |                                |
|                    | SR_instance2                | SR                     |               |                                |
|                    | MC_ChangeAxisUse_DevF       | MC_ChangeAxisUse       |               |                                |
|                    | 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) & _EC_EntrySlavTbl[4]),
  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) &
        _EC_EntrySlavTbl[5]),
    NodeAdr :=UINT#5,
    IsEnable:=TRUE);

// Exclusive control of instructions. Start enabling EtherCAT slave G 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/<br>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); |



## 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.

## Variables

|           | Meaning          | I/O   | Description   | Valid range           | Unit | Default         |
|-----------|------------------|-------|---|-----------------------|------|-----------------|
| UnitProxy | Specified Unit   | Input | Unit to which to write data                             | ---                   | ---  | *1              |
| Obj       | Object parameter |       | Object parameter  |                       |      | ---             |
| TimeOut   | Timeout time     |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 0 to 60,000           | ms   | 2000<br>(2.0 s) |
| WriteDat  | Write data       |       | Data to write to NX object                              | Depends on data type. | ---  | *1              |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean                     | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 |                                 |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |    |
| Obj       |                                 |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |    |
| TimeOut   |                                 |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |    |
| WriteDat  | OK                              | 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                                  | <code>_sNXUNIT_ID</code> |
| 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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT                    |

To *UnitProxy*, pass the device variable that is assigned to the specified Unit.

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 | <code>_sNXOBJ_ACCESS</code> | ---                   | ---  | ---     |
| Index                          | Index            | Index            | UINT                        | Depends on data type. | ---  | 0       |
| Subindex                       | Subindex         | Subindex         | USINT                       |                       |      |         |
| IsCompleteAccess <sup>*1</sup> | 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.

### ● 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 instruction, *NX\_ChangeWriteMode* on page 2-927, 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 instruction, *NX\_SaveParam* on page 2-933, to save the data that you wrote.
- 4** Use the instruction, *RestartNXUnit* on page 2-921, 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 instruction, *NX\_SaveParam* on page 2-933, 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.

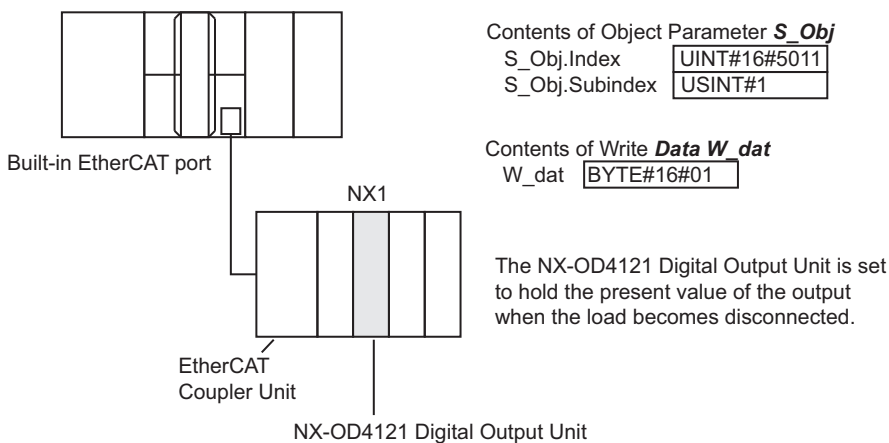
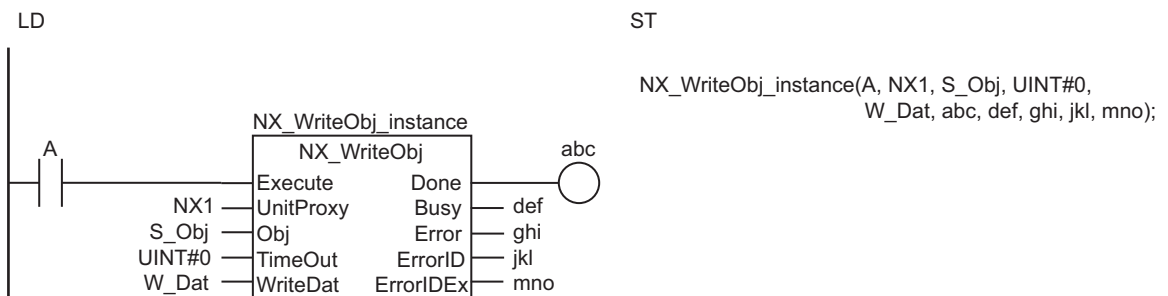
**Notation Example**

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.



## Related System-defined Variables

| Name   | Meaning                                    | Data type | Description  |
|--|--|-----------|--|
| <code>_EC_MBXSlaVtbl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible.       |
| <code>_NXB_UnitMsgActiveTbl[i]</code>                      | NX Unit Message Enabled Status             | BOOL      | This table indicates the slaves that can perform message communications.<br>Use this variable to confirm that communications with the relevant slave are possible. |

## Precautions for Correct Use

- Execution of this instruction is continued until 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 the execution.
- Refer to *Using this Section* on 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 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 in the I/O Map of the Sysmac Studio.  
Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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 instruction, *NX\_SaveParam* on page 2-933, 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 *A-4 Instructions Related to NX Message Communications Errors* on page A-35 for a list of the instructions that are related to NX Message Communications Errors.
- *Error* changes to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i> | Meaning  |
|-------------------------|---------------------------|--|
| 16#0400                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The value of <i>UnitProxy</i> is outside the valid range.</li> <li>• The value of <i>TimeOut</i> is outside the valid range.</li> </ul> |
| 16#0419                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The data type of <i>UnitProxy</i> is not correct.</li> <li>• The data type of <i>WriteDat</i> is not correct.</li> </ul>                |
| 16#041B                 | 16#00000000               | More than 2,048 bytes of data was specified for <i>WriteDat</i> .  |
| 16#2C00                 | 16#00000401               | The specified Unit does not support the instruction.   |



| Value of <i>ErrorID</i>   | Value of <i>ErrorIDEx</i> | Meaning  |
|---|---------------------------|--|
| 16#00001001<br>16#00001002<br>16#00170000<br>16#00200000<br>16#00210000 |                           | 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#00001010   |                           | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
| 16#00001101   |                           | The correct Unit was not specified.<br>Check the Unit.   |
| 16#0000110B   |                           | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
| 16#00001110   |                           | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
| 16#00001111   |                           | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
| 16#00002101   |                           | The specified NX object cannot be written.   |
| 16#00002110   |                           | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
| 16#00002210   |                           | The specified Unit is not in a mode that allows writing data.  |
| 16#00002213   |                           | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
| 16#00002230   |                           | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
| 16#00002231   |                           | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction.  |
| 16#0000250F   |                           | Hardware access failed.<br>Execute the instruction again.  |
| 16#00002601<br>16#00002602<br>16#00100000                               |                           | The specified Unit does not support this instruction.<br>Check the version of the Unit.  |
| 16#00002603   |                           | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use.   |
| 16#00002621   |                           | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.  |
| 16#00010000   |                           | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.  |
| 16#00110000   |                           | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.   |

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning   |
|-------------------------|--|---|
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.                     |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.   |
|                         | 16#00310000  | The specified Unit not supported for connection.<br>Check the version of the Unit.  |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.  |
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000   | An error occurred in the communications network.<br>Reduce the amount of communications traffic.  |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000   | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON. |
| 16#2C01                 | 16#00000000  | The number of instructions that can be simultaneously executed was exceeded.  |
| 16#2C02                 | 16#00000000  | A timeout occurred during communications.   |
| 16#2C03                 | 16#00000000  | The size of the send message is not correct.  |

## Sample Programming

This section provides the following two examples for explanation.

- Writing data with the Power Off Retain attribute to an NX Unit, which is reflected in the Unit settings at a restart of the NX Unit.
- Writing data with the Power Off Retain attribute to an NX Unit, which is immediately reflected in the Unit settings.

### Example of Writing Data That Is Updated at Restart of the 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  $\mu$ s.

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 $\mu$ s | 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.

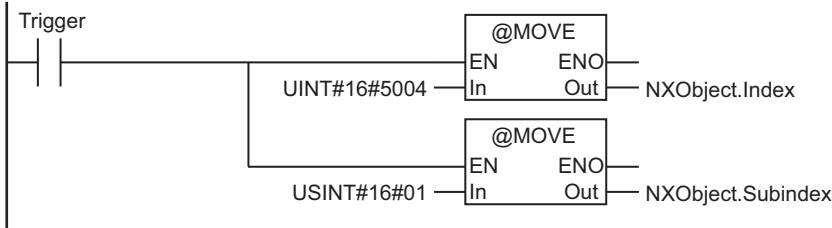
#### ● LD

| 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      |  |  |

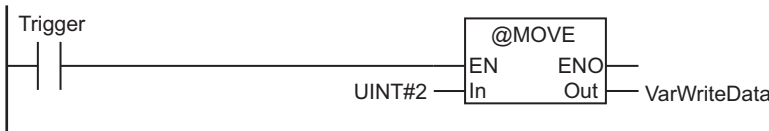
| External Variables | Variable           | Constant | Data type                  | Comment                                    |
|--------------------|--------------------|----------|----------------------------|--|
|                    | _EC_MBXSlavTb<br>I | ☑        | ARRAY[1..512] OF<br>BOOL*1 | Message Communications Enabled Slave Table |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

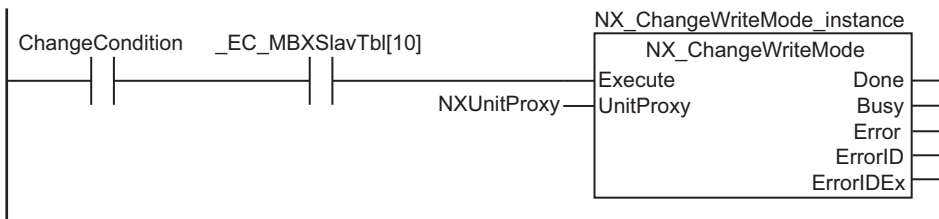
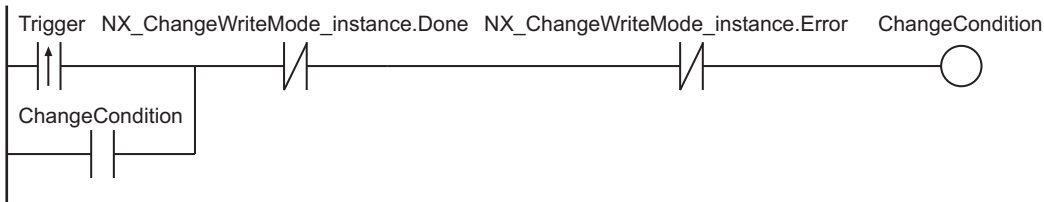
Prepare object parameter.



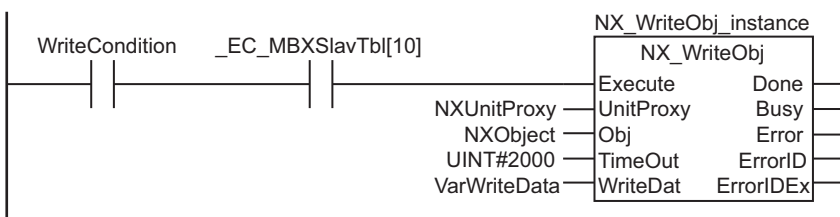
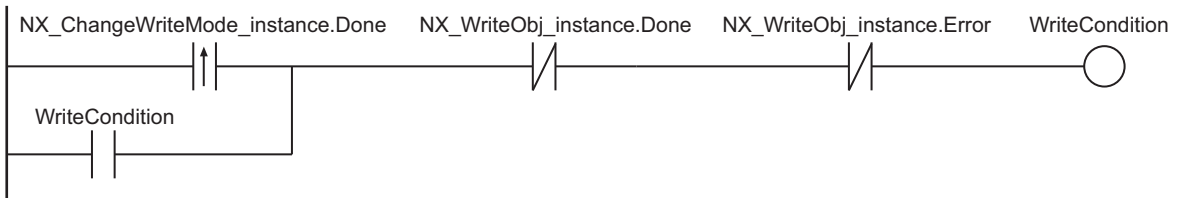
Prepare write data.



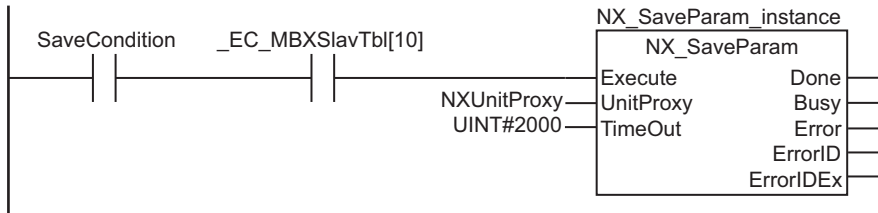
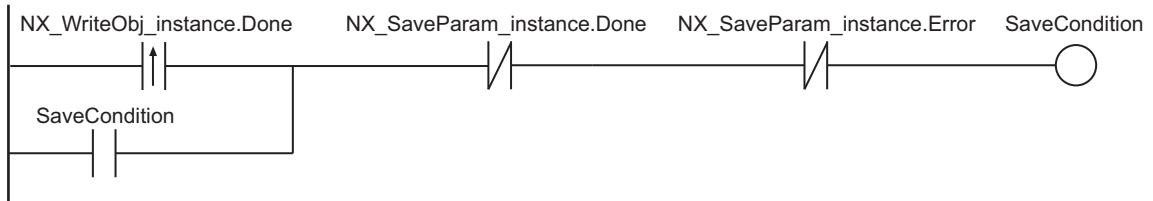
Execute NX\_ChangeWriteMode instruction.



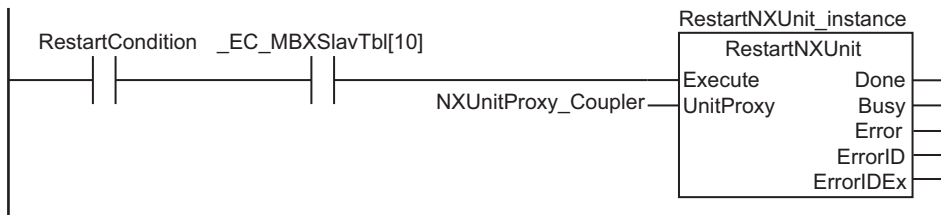
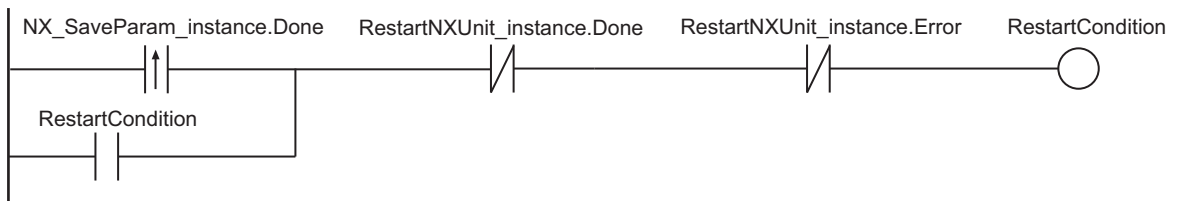
Execute NX\_WriteObj instruction.



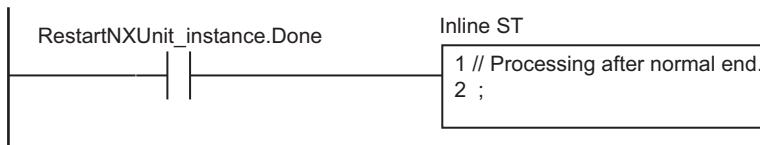
Execute NX\_SaveParam instruction.



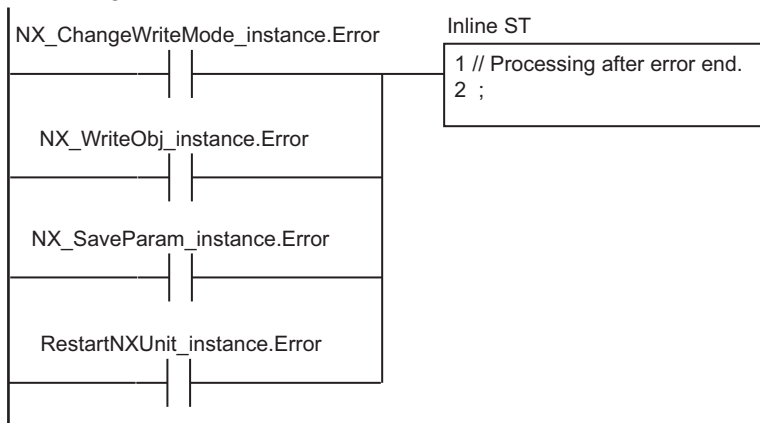
Execute RestartNXUnit instruction.



Processing after normal end.




Processing after error end.



## ● 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_ChangeWrite-Mode_instance | NX_ChangeWrite-Mode |  |  |
|                    | 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_MBXSlavTb<br> |  | ARRAY[1..512] OF<br>BOOL*1 | Message Communications Enabled Slave Table |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

```
// 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)) THE
N
    // Processing after error end.
    ErrorEnd := ErrorEnd + UINT#1;
END_IF;

```

### Example of Writing Data That Is Immediately Updated

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.



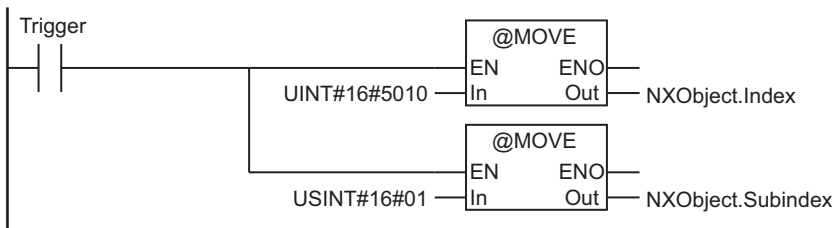
● LD

| 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   |  |  |

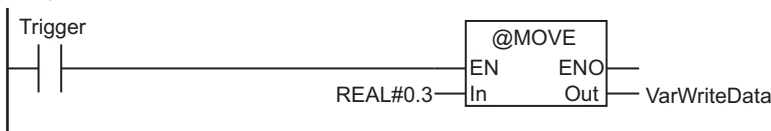
| External Variables | Variable       | Constant                            | Data type               | Comment                                    |
|--------------------|----------------|-------------------------------------|-------------------------|--|
|                    | _EC_MBXSlavTbl | <input checked="" type="checkbox"/> | ARRAY[1..512] OF BOOL*1 | Message Communications Enabled Slave Table |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

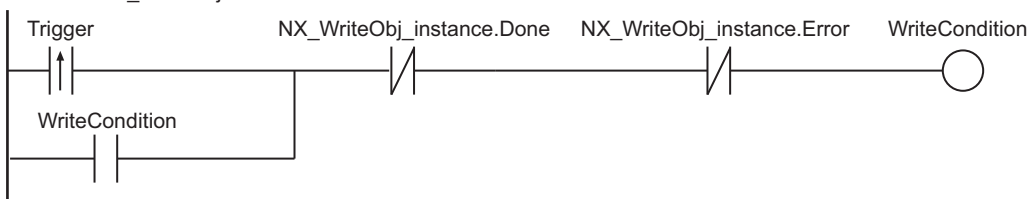
Prepare object parameter.

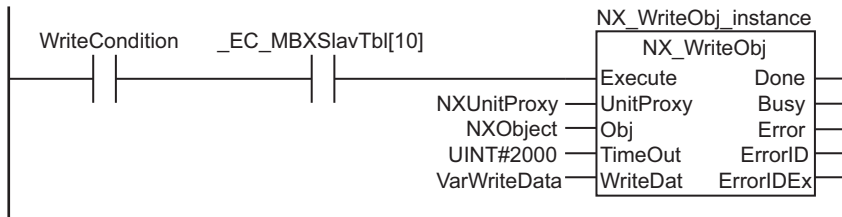


Prepare write data.

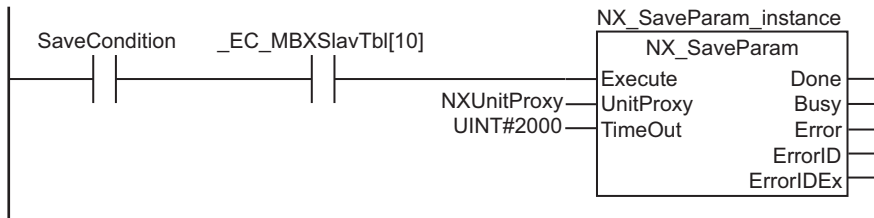
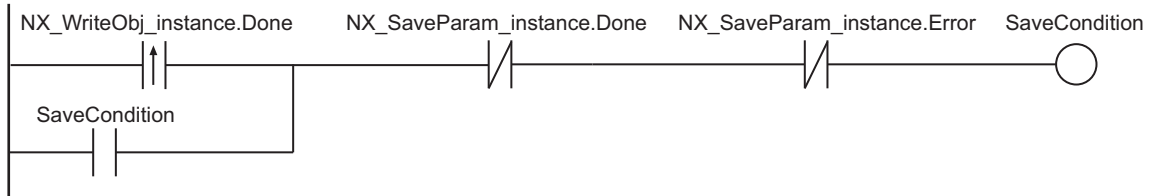


Execute NX\_WriteObj instruction.

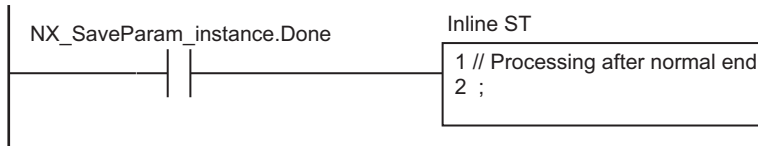




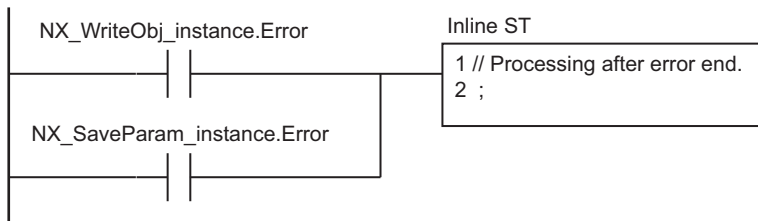
Execute NX\_SaveParam instruction.



Processing after normal end.




Processing after error end.



● 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  |

| Internal Variables | Variable              | Data type      | Initial value                                    | Comment          |
|--------------------|-----------------------|----------------|--|------------------|
|                    | 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_MBXSlatTbl |  | ARRAY[1..512] OF BOOL*1 | Message Communications Enabled Slave Table |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

```
// 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_MBXSlatTbl[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/<br>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); |



## 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.

## Variables

|           | Meaning          | I/O    | Description   | Valid range           | Unit | Default         |
|-----------|------------------|--------|---|-----------------------|------|-----------------|
| UnitProxy | Specified Unit   | Input  | Unit from which to read data                            | ---                   | ---  | *1              |
| Obj       | Object parameter |        | Object parameter  | ---                   | ---  | ---             |
| TimeOut   | Timeout time     |        | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 0 to 60,000           | ms   | 2000<br>(2.0 s) |
| ReadDat   | Read data        | In-out | Data read from NX object                                | Depends on data type. | ---  | ---             |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |    |
| Obj       |             |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |    |
| TimeOut   |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |    |
| ReadDat   | OK          | 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 this 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 <i>Path</i> length | Valid <i>Path</i> length                        | USINT       |

To `UnitProxy`, pass the device variable that is assigned to the specified Unit.

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 <sup>*1</sup> | Complete access  | Complete access  | BOOL           | FALSE only            | ---  | FALSE   |

\*1. This member is not used for this instruction. Always set the value to FALSE.

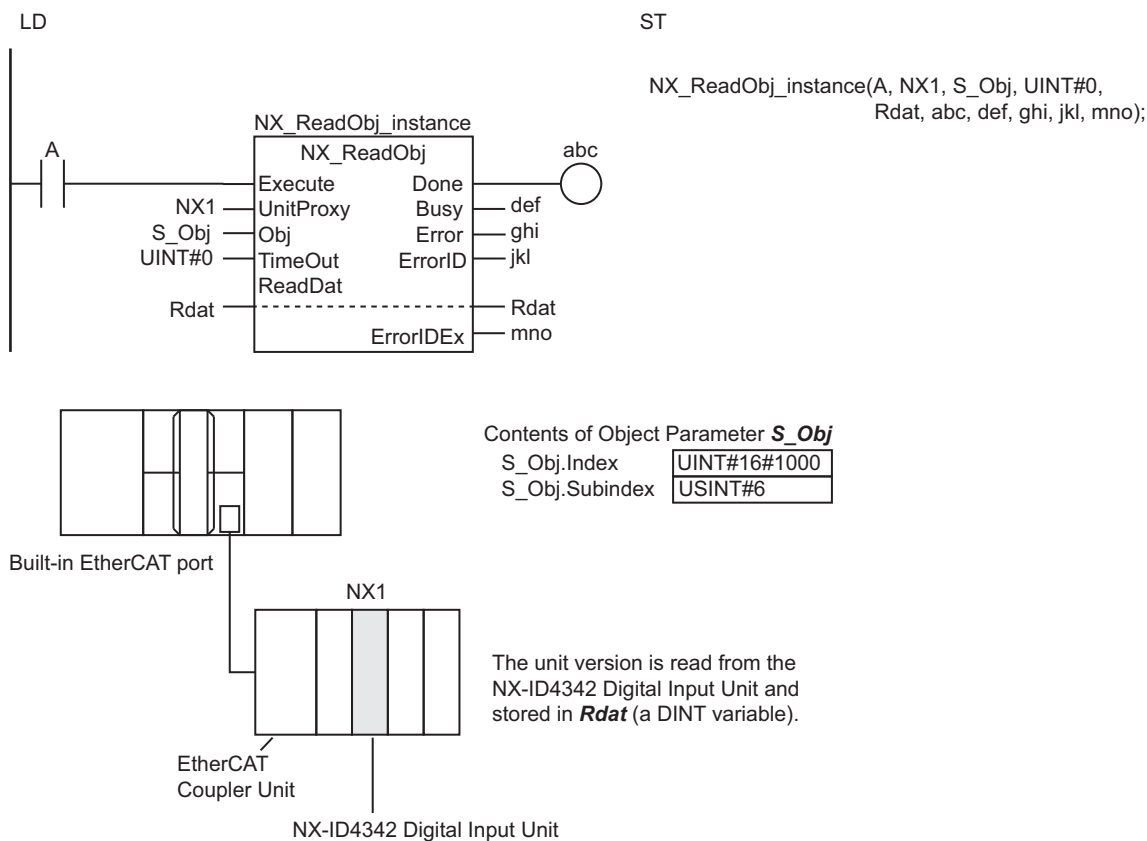
## Notation Example

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`.



## Related System-defined Variables

| Name   | Meaning                                    | Data type | Description  |
|--|--|-----------|--|
| <code>_EC_MBXSlavTbl[i]</code><br>"i" is the node address. | Message Communications Enabled Slave Table | BOOL      | This variable indicates whether communications are possible for each slave.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible.       |
| <code>_NXB_UnitMsgActiveTbl[i]</code>                      | NX Unit Message Enabled Status             | BOOL      | This table indicates the slaves that can perform message communications.<br>Use this variable to confirm that communications with the relevant slave are possible. |

## Precautions for Correct Use

- Execution of this instruction is continued until 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 the execution.
- Refer to *Using this Section* on 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 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 in the I/O Map of the Sysmac Studio.  
Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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 *A-4 Instructions Related to NX Message Communications Errors* on page A-35 for a list of the instructions that are related to NX Message Communications Errors.
- *Error* will change to TRUE if an error occurs. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>   | Meaning  |
|-------------------------|---|--|
| 16#0400                 | 16#00000000   | <ul style="list-style-type: none"> <li>• The value of <i>UnitProxy</i> is outside the valid range.</li> <li>• The value of <i>TimeOut</i> is outside the valid range.</li> </ul>   |
| 16#0410                 | 16#00000000   | <i>ReadDat</i> is STRING data and it does not end with a NULL character.   |
| 16#0419                 | 16#00000000   | <ul style="list-style-type: none"> <li>• The data type of <i>UnitProxy</i> is not correct.</li> <li>• The data type of <i>ReadDat</i> is not correct.</li> </ul>   |
| 16#041C                 | 16#00000000   | The size of <i>ReadDat</i> is not the same as the size of the NX object to read.   |
| 16#2C00                 | 16#00000401   | The specified Unit does not support the instruction.   |
|                         | 16#00001001<br>16#00001002<br>16#00170000<br>16#00200000<br>16#00210000 | 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#00001010   | The data size of the specified NX object does not agree with the data size specified in <i>WriteDat</i> .  |
|                         | 16#00001101   | The correct Unit was not specified.<br>Check the Unit.   |
|                         | 16#0000110B   | The size of the read data is too large.<br>Make sure that the read data specification is correct.  |
|                         | 16#00001110   | There is no object that corresponds to the value of <i>Obj.Index</i> .   |
|                         | 16#00001111   | There is no object that corresponds to the value of <i>Obj.Subindex</i> .  |
|                         | 16#00002101   | The specified NX object cannot be written.   |
|                         | 16#00002110   | The value of <i>WriteDat</i> exceeds the range of values for the NX object to write.   |
|                         | 16#00002210   | The specified Unit is not in a mode that allows writing data.  |
|                         | 16#00002213   | Instruction execution was not possible because the specified Unit was performing an I/O check.<br>Execute the instruction after the I/O check is completed.  |
|                         | 16#00002230   | The status of the specified Unit does not agree with the value of the read source or write destination NX object.<br>Take the following actions if the value of <i>Obj.Index</i> is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. <ul style="list-style-type: none"> <li>• Delete the read source or write designation NX object from the I/O allocation settings.</li> <li>• Reset the error for the specified Unit.</li> <li>• Place the specified Unit in a mode that does not allow writing data.</li> </ul> |
|                         | 16#00002231   | Instruction execution was not possible because the specified Unit was performing initialization.<br>Wait for the Unit to start normal operation and then execute the instruction.  |
|                         | 16#0000250F   | Hardware access failed.<br>Execute the instruction again.  |



| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i>  | Meaning  |
|-------------------------|--|--|
|                         | 16#00002601<br>16#00002602<br>16#00100000  | The specified Unit does not support this instruction.<br>Check the version of the Unit.  |
|                         | 16#00002603  | Execution of the instruction failed.<br>Execute the instruction again.<br>Make sure that at least one channel is <b>Enabled</b> in the selection of the channels to use. |
|                         | 16#00002621  | The NX Unit is not in a status in which it can acknowledge the instruction.<br>Wait for a while and then execute the instruction again.                                  |
|                         | 16#00010000  | The specified Unit does not exist.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00110000  | The specified port number does not exist.<br>Make sure that the Unit configuration is correct.   |
|                         | 16#00120000<br>16#00130000<br>16#00150000<br>16#00160000   | The value of <i>UnitProxy</i> is not correct.<br>Set the variable that indicates the specified EtherCAT Coupler Unit again.  |
|                         | 16#00140000  | The specified node address is not correct.<br>Make sure that the Unit configuration is correct.  |
|                         | 16#00300000<br>16#80010000   | The specified Unit is busy.<br>Execute the instruction again.  |
|                         | 16#00310000  | The specified Unit is not supported for connection.<br>Check the version of the Unit.  |
|                         | 16#80000000<br>16#80050000<br>16#81010000<br>16#81020000<br>16#82020000<br>16#82030000<br>16#82060000 to<br>16#8FFF0000<br>16#90010000 to<br>16#FFFE0000 | An error occurred in the communications network.<br>Execute the instruction again.   |
|                         | 16#80020000<br>16#80030000<br>16#81030000<br>16#82000000   | An error occurred in the communications network.<br>Reduce the amount of communications traffic.   |
|                         | 16#80040000<br>16#81000000<br>16#82010000<br>16#82040000<br>16#82050000<br>16#90000000   | An error occurred in the communications network.<br>Check the Unit and cable connections.<br>Make sure that the power supply to the Unit is ON.                          |
| 16#2C01                 | 16#00000000  | The number of instructions that can be simultaneously executed was exceeded.   |
| 16#2C02                 | 16#00000000  | A timeout occurred during communications.  |

## Sample Programming

In this example, the value of the *I/O Refresh Method 1* object parameter is read out 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   |

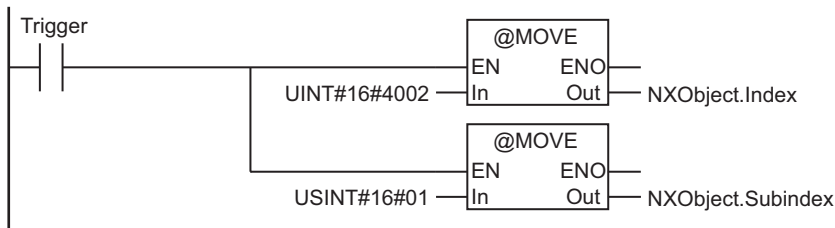
### LD

| 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     |  |                                  |

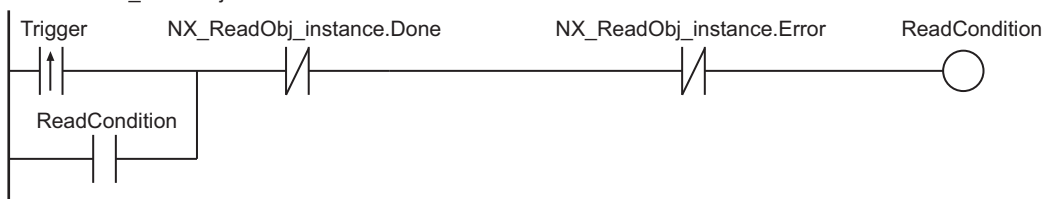
| External Variables | Variable       | Constant | Data type               | Comment                                    |
|--------------------|----------------|----------|-------------------------|--|
|                    | _EC_MBXSlavTbl | ☑        | ARRAY[1..512] OF BOOL*1 | Message Communications Enabled Slave Table |

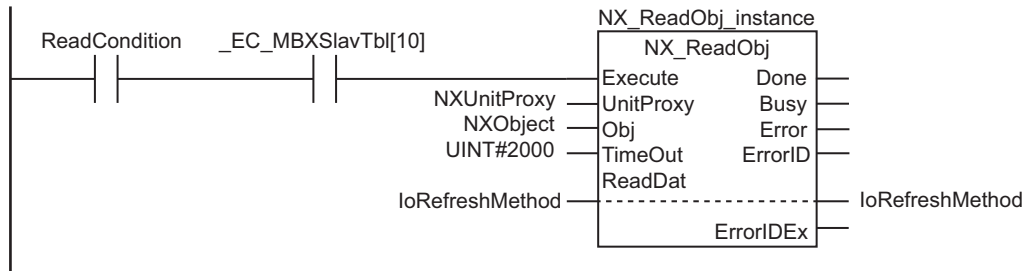
\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY [1..192] OF BOOL*.

Prepare object parameter.

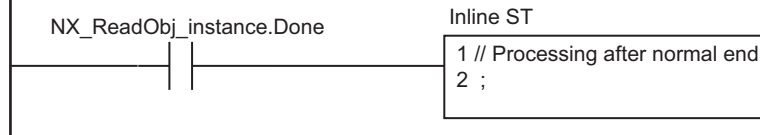


Execute NX\_ReadObj instruction.

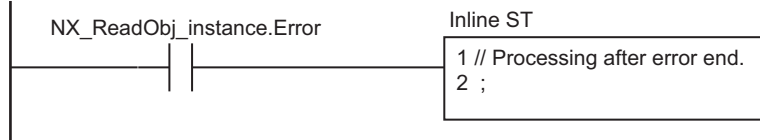




Processing after normal end.



Processing after error end.



## 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, Sub-index:=0, Is-CompleteAccess:=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<br>BOOL*1 | Message Communications Enabled<br>Slave Table |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and 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
```

```
    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

| Instruction  | Name                        | Page        |
|--------------|-----------------------------|-------------|
| IOL_ReadObj  | Read IO-Link Device Object  | page 2-1062 |
| IOL_WriteObj | Write IO-Link Device Object | page 2-1071 |

# IOL\_ReadObj

The IOL\_ReadObj instruction reads data from IO-Link device objects.

| Instruction | Name                       | FB/<br>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 <i>ErrorID</i> is 4800 hex. | 16#0000 to 16#FFFF    | ---   | ---     |
| ReadSize   | Read data size                  |        | Size of data stored in <i>ReadDat</i>  | 10#1 to 10#232        | Bytes | ---     |

|                                 | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1063 for details on the structure <i>_sDEVICE_PORT</i> .   |             |      |       |       |       |          |       |       |      |     |      |      |      |                      |      |  |     |    |        |  |  |
| DeviceObj                       | Refer to <i>Function</i> on page 2-1063 for details on the structure <i>_sIOOBJ_ACCESS</i> .  |             |      |       |       |       |          |       |       |      |     |      |      |      |                      |      |  |     |    |        |  |  |
| RetryCfg                        | Refer to <i>Function</i> on page 2-1063 for details on the structure <i>_sIOL_RETRY_CFG</i> . |             |      |       |       |       |          |       |       |      |     |      |      |      |                      |      |  |     |    |        |  |  |
| 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. |   |             |      |       |       |       |          |       |       |      |     |      |      |      |                      |      |  |     |    |        |  |  |

|           | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|           | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| 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 | De-<br>fault |
|------------|------------------------|---|----------------------------|---|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---   | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify   | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOption-Board</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control  | <code>_sNXUNIT_ID</code>   | ---   | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control   | <code>_sECAT_ID</code>     | ---   | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control   | <code>_sOPTBOARD_ID</code> | ---   | ---  | ---          |
| Reserved   | Reserved               | Reserved  | ---                        | ---   | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>2: Port 2<br>3: Port 3<br>4: Port 4<br>5: Port 5<br>6: Port 6<br>7: Port 7<br>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                      |
|-------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>    | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code> | 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 | <code>_sIOLOBJ_ACCESS</code> | ---                   | ---  | ---     |
| 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                                    | <code>_sIOL_RETRY_CFG</code> | ---                   | ---   | ---     |
| TimeOut  | Timeout time            | Timeout time<br>If 0 is set, the timeout time is 2.0 s.                        | TIME                         | 0 to 300 s            | ---   | T#2.0s  |
| RetryNum | Number of retries       | Number of retries at timeout<br>If 0 is set, the number of retries is 3 times. | UINT                         | Depends on data type. | Times | 3       |

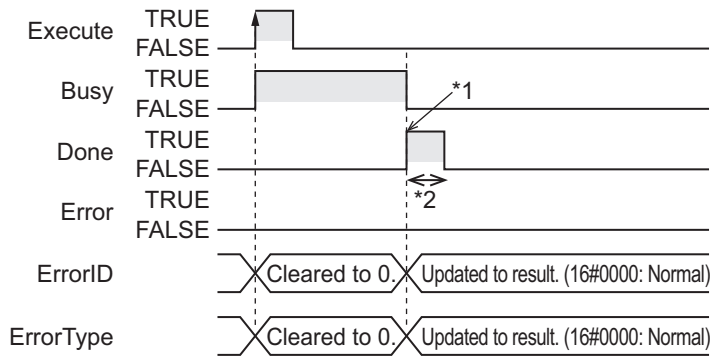
Data read from the IO-Link device is stored in the *ReadDat* in-out variable.

## Timing Charts

The following figures show the timing charts.



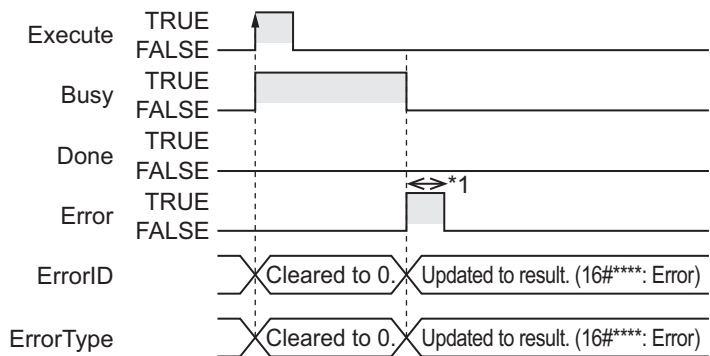
### ● Normal end



\*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.<br>Slaves are given in the table in the order of slave node addresses.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.

## Precautions for Correct Use

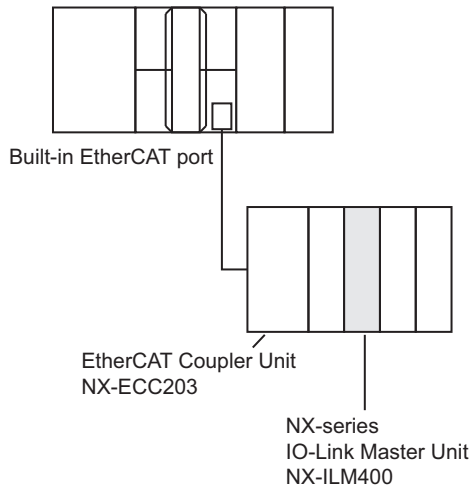
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- For *DevicePort.NxUnit* 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)* 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) A value that is out of range was set for *DevicePort.NxUnit* or *DevicePort.EcatSlave*.
  - b) 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*.
  - c) 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/>
  - d) The specified IO-Link master unit does not exist.
  - e) 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.
  - f) The specified IO-Link master unit is not in a condition to receive messages.
  - g) 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.
  - h) A timeout occurred during communications.
  - i) The specified port of the IO-Link master unit is not the IO-Link mode. The port is disabled or in the SIO mode.
  - j) The IO-Link device is not connected to the specified port on the IO-Link master unit.
  - k) The IO power is not supplied to the specified port of the IO-Link master unit.
  - l) 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).



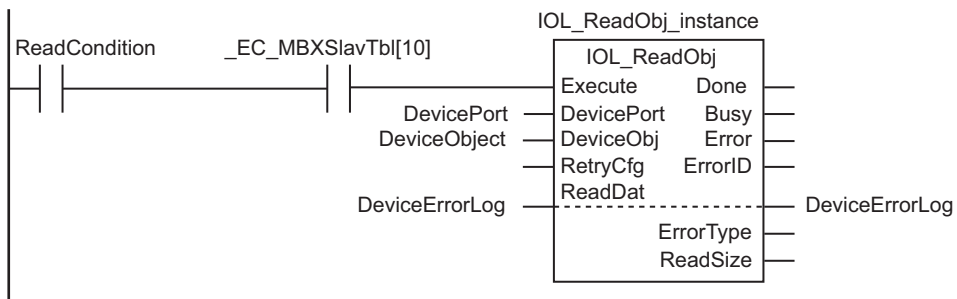
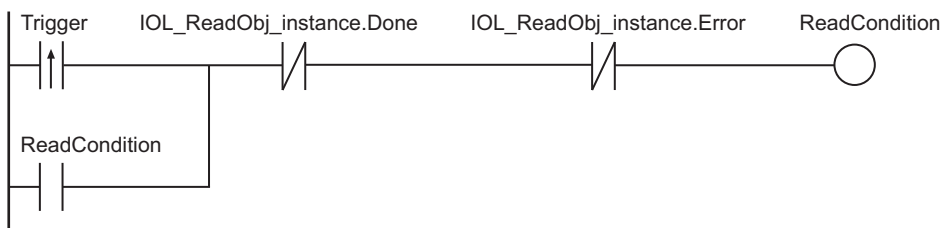
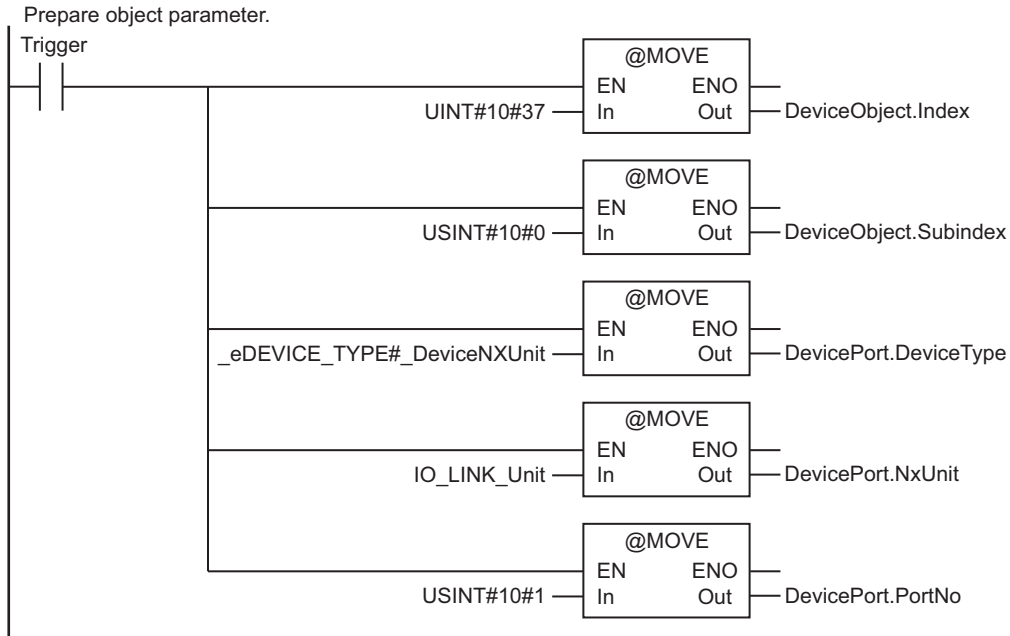
The error log (Index:37/Subindex:0) of 30 bytes is read from the photoelectric sensor (E3Z) connected to port 1 on the NX-ILM400. The read data is stored in *DeviceErrorLog*.  
The node address of the NX-ECC203 is 10.

## 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 | <input checked="" type="checkbox"/> | ARRAY[1..512] OF BOOL <sup>*1</sup>  | Message Communications Enabled Slave Table |
|                    | IO_LINK_Unit   | <input checked="" type="checkbox"/> | Set the device variable which specifies NX-ILM400 as the initial value of the structure member <i>NxUnit</i> . |  |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY[1..192] OF BOOL*.



## 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 | <input checked="" type="checkbox"/> | ARRAY[1..512] OF BOOL*1  | Message Communications Enabled Slave Table |
|                    | IO_LINK_Unit   | <input checked="" type="checkbox"/> | Set the device variable which specifies NX-ILM400 as the initial value of the structure member <i>NxUnit</i> . |  |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and 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 <sup>*1</sup>  | 10#1 to 10#232        | Bytes | ---     |
| ErrorType  | Error type                      | Output | Error code that is returned by IO-Link device is stored when <i>ErrorID</i> 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.

|            | Boo lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-1072 for details on the structure <i>_sDEVICE_PORT</i> .   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| DeviceObj  | Refer to <i>Function</i> on page 2-1072 for details on the structure <i>_sIOLOBJ_ACCESS</i> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| RetryCfg   | Refer to <i>Function</i> on page 2-1072 for details on the structure <i>_sIOL_RETRY_CFG</i> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| 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.   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

|           | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|           | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| 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 | De-<br>fault |
|------------|------------------------|---|----------------------------|---|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---   | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify   | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOption-Board</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control  | <code>_sNXUNIT_ID</code>   | ---   | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control   | <code>_sECAT_ID</code>     | ---   | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control   | <code>_sOPTBOARD_ID</code> | ---   | ---  | ---          |
| Reserved   | Reserved               | Reserved  | ---                        | ---   | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>2: Port 2<br>3: Port 3<br>4: Port 4<br>5: Port 5<br>6: Port 6<br>7: Port 7<br>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                      |
|-------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>    | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code> | 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 | <code>_sIOLOBJ_ACCESS</code> | ---                   | ---  | ---     |
| 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:

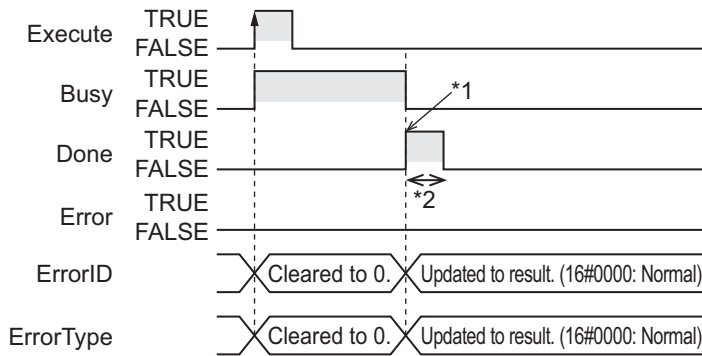
| Name     | Meaning                 | Description  | Data type                    | Valid range           | Unit  | Default |
|----------|-------------------------|--|------------------------------|-----------------------|-------|---------|
| RetryCfg | Execution retry setting | Setting for the instruction execution retry                                    | <code>_sIOL_RETRY_CFG</code> | ---                   | ---   | ---     |
| TimeOut  | Timeout time            | Timeout time<br>If 0 is set, the timeout time is 2.0 s.                        | TIME                         | 0 to 300 s            | ---   | T#2.0s  |
| RetryNum | Number of retries       | Number of retries at timeout<br>If 0 is set, the number of retries is 3 times. | 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

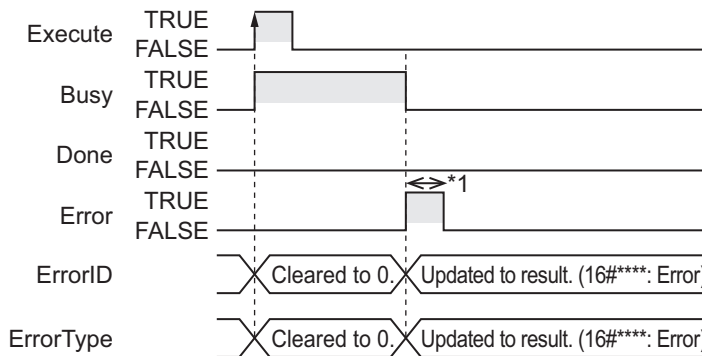
The following figures show the timing charts.

● Normal end



\*1. Writing completed.  
\*2. Task period

● Error end



\*1. Task period

Related System-defined Variables

| Name           | 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. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.

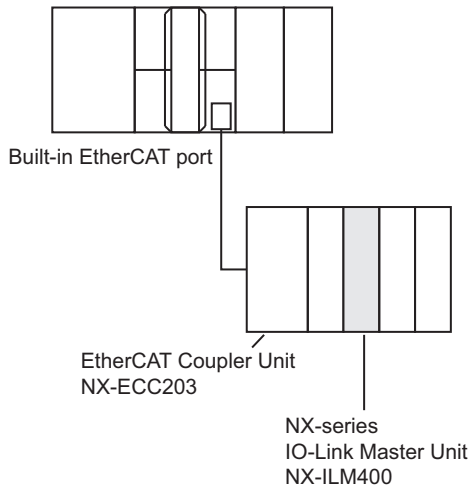
Precautions for Correct Use

- 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)* 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) A value that is out of range was set for *DevicePort.NxUnit* or *DevicePort.EcatSlave*.
  - b) The value of *TimeOut* is outside of the valid range.
  - c) The data type of *DevicePort* is invalid.
  - d) More than 232 bytes of data was specified for *WriteDat*.
  - e) 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/>
  - f) The specified IO-Link master unit does not exist.
  - g) 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.
  - h) The specified IO-Link master unit is not in a condition to receive messages.
  - i) More than 32 of the following instructions were executed at the same time: EC\_CoESDOWrite, EC\_CoESDORed, EC\_StartMon, EC\_StopMon, EC\_SaveMon, EC\_CopyMon, EC\_DisconnectSlave, EC\_ConnectSlave, EC\_ChangeEnableSetting, IOL\_ReadObj, and IOL\_WriteObj.
  - j) A timeout occurred during communications.
  - k) The specified port of the IO-Link master unit is not the IO-Link mode. The port is disabled or in the SIO mode.
  - l) The IO-Link device is not connected to the specified port on the IO-Link master unit.
  - m) The IO power is not supplied to the specified port of the IO-Link master unit.
  - n) 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 photoelectric sensor (E3Z) connected to port 1 on the NX-ILM400. The written data is stored in *SwitchPoint*.

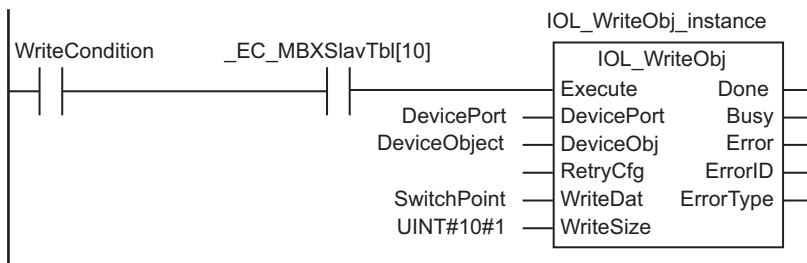
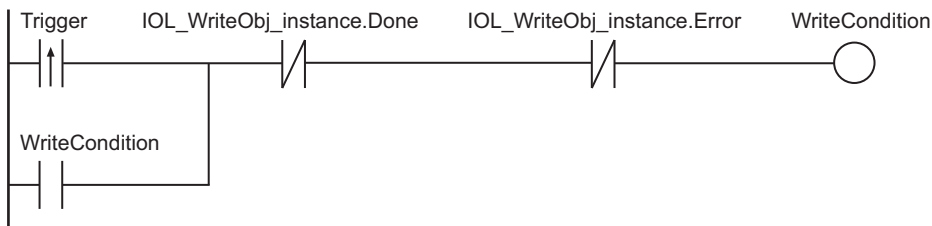
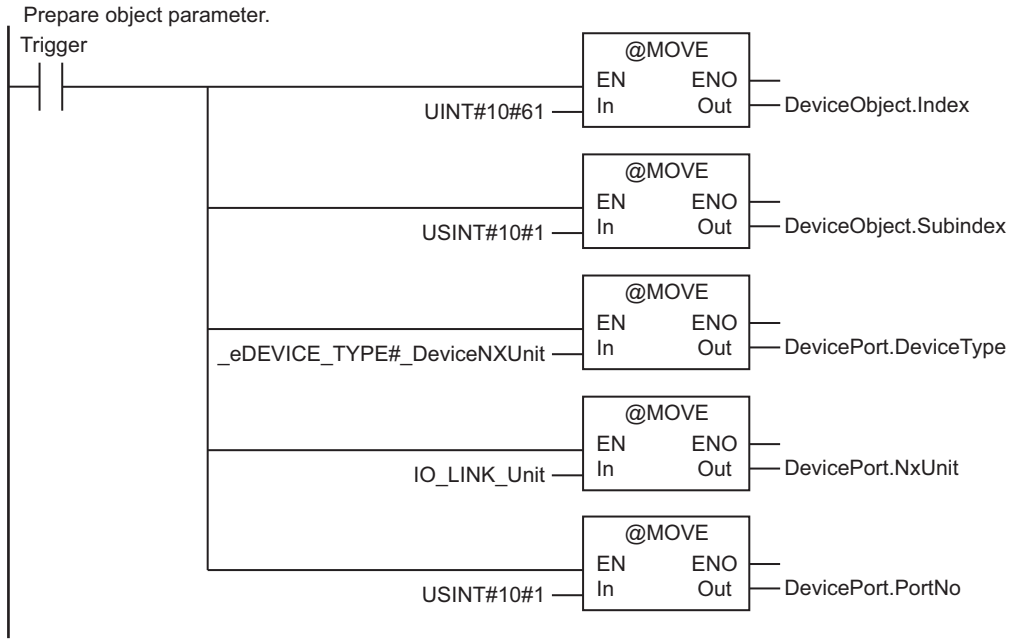
The node address of the NX-ECC203 is 10.

### 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, Subindex:=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 | <input checked="" type="checkbox"/> | ARRAY[1..512] OF BOOL <sup>*1</sup>  | Message Communications Enabled Slave Table |
|                    | IO_LINK_Unit   | <input checked="" type="checkbox"/> | Set the device variable which specifies NX-ILM400 as the initial value of the structure member <i>NxUnit</i> . |  |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and NJ-series CPU Units, the data type is *ARRAY[1..192] OF BOOL*.



## 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 | <input checked="" type="checkbox"/> | ARRAY[1..512] OF BOOL*1  | Message Communications Enabled Slave Table |
|                    | IO_LINK_Unit   | <input checked="" type="checkbox"/> | Set the device variable which specifies NX-ILM400 as the initial value of the structure member <i>NxUnit</i> . |  |

\*1. For the NX102 CPU Units, NX1P2 CPU Units, and 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

| Instruction         | Name   | Page        |
|---------------------|--|-------------|
| CIPOpen             | Open CIP Class 3 Connection (Large_Forward_Open)     | page 2-1083 |
| CIPOpenWithDataSize | Open CIP Class 3 Connection with Specified Data Size | page 2-1093 |
| CIPRead             | Read Variable Class 3 Explicit                       | page 2-1097 |
| CIPWrite            | Write Variable Class 3 Explicit                      | page 2-1103 |
| CIPSend             | Send Explicit Message Class 3                        | page 2-1109 |
| CIPClose            | Close CIP Class 3 Connection                         | page 2-1114 |
| CIPUCMMRead         | Read Variable UCMM Explicit                          | page 2-1117 |
| CIPUCMMWrite        | Write Variable UCMM Explicit                         | page 2-1123 |
| CIPUCMMSend         | Send Explicit Message UCMM                           | page 2-1130 |
| SkUDPCreate         | Create UDP Socket                                    | page 2-1141 |
| SkUDPRcv            | UDP Socket Receive                                   | page 2-1149 |
| SkUDPSend           | UDP Socket Send                                      | page 2-1152 |
| SkTCPAccept         | Accept TCP Socket                                    | page 2-1155 |
| SkTCPConnect        | Connect TCP Socket                                   | page 2-1158 |
| SkTCPRcv            | TCP Socket Receive                                   | page 2-1167 |
| SkTCPSend           | TCP Socket Send                                      | page 2-1170 |
| SkGetTCPStatus      | Read TCP Socket Status                               | page 2-1173 |
| SkClose             | Close TCP/UDP Socket                                 | page 2-1176 |
| SkClearBuf          | Clear TCP/UDP Socket Receive Buffer                  | page 2-1179 |
| SkSetOption         | Set TCP Socket Option                                | page 2-1182 |
| ModbusTCPcmd        | Send Modbus TCP General Command                      | page 2-1187 |

| <b>Instruction</b> | <b>Name</b>                   | <b>Page</b> |
|--------------------|-------------------------------|-------------|
| ModbusTCPRead      | Send Modbus TCP Read Command  | page 2-1195 |
| ModbusTCPWrite     | Send Modbus TCP Write Command | page 2-1203 |
| ChangeIPAdr        | Change IP Address             | page 2-1211 |
| ChangeFTPAccount   | Change FTP Account            | page 2-1220 |
| ChangeNTPServerAdr | Change NTP Server Address     | page 2-1224 |
| FTPGetFileList     | Get FTP Server File List      | page 2-1229 |
| FTPGetFile         | Get File from FTP Server      | page 2-1243 |
| FTPPutFile         | Put File onto FTP Server      | page 2-1252 |
| FTPRemoveFile      | Delete FTP Server File        | page 2-1263 |
| FTPRemoveDir       | Delete FTP Server Directory   | page 2-1273 |

# CIPOpen

The CIPOpen instruction 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/<br>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

|           | 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) |
| Handle    | Handle       | Output | Handle       | ---                   | ---   | ---      |

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-1083 for details on the structure <code>_sCIP_HANDLE</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

## 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.

*Handle* is output when the connection is open.

*TimeOut* specifies the connection timeout time.

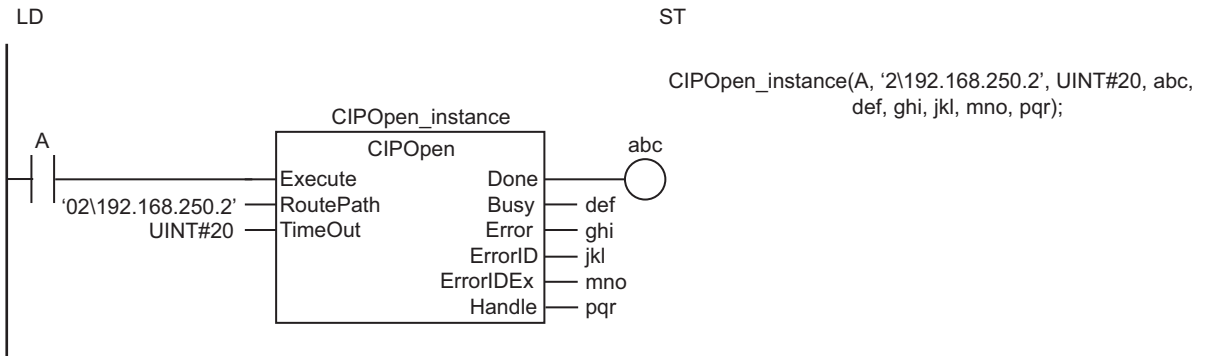
If a response does not return from the remote node within the connection timeout time after the CIP-Send, CIPWrite, or CIPRead instruction is executed, it is assumed that communications failed.

The connection timeout time is reset when the CIPRead, CIPWrite, or CIPSend instruction is executed and the remote node returns a response.

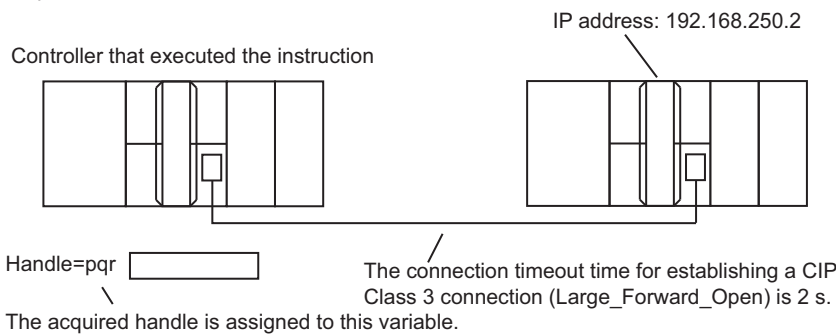
The data type of *Handle* is structure `_sCIP_HANDLE`. The specifications are as follows:

| Name   | Meaning | Description | Data type         | Valid range              | Unit | Default |
|--------|---------|-------------|-------------------|--------------------------|------|---------|
| Handle | Handle  | Handle      | _sCIP_HAN-<br>DLE | ---                      | ---  | ---     |
| Handle | Handle  | Handle      | UDINT             | Depends on<br>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*.



The Open CIP Class 3 Connection (*Large\_Forward\_Open*) 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   |
|---|---------|-----------|---|
| <i>_EIP_EtnOnlineSta</i> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible. |
| <i>_EIP1_EtnOnlineSta</i> <sup>*2</sup> |         |           |   |
| <i>_EIP2_EtnOnlineSta</i> <sup>*3</sup> |         |           |   |

- \*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. You can specify *\_EIP\_EtnOnlineSta* instead of *\_EIP1\_EtnOnlineSta*.
- \*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- Refer to the following manuals for details on CIP communications.
  - a) *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)*
  - b) *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 the instruction, *CIPOpenWithDataSize* on page 2-1093.

## 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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) The value of *TimeOut* is outside of the valid range.
  - b) The text string in *RoutePath* is not valid.
  - c) More than 32 CIP-related instructions were executed simultaneously.
  - d) An attempt was made to open a connection beyond the *CIPClass* connection resources (32 connections).
  - e) A connection opened response was not received.
  - f) The remote node to which to open a connection does not support *Large\_Forward\_Open*.
  - g) There is a setting error for the local IP address.
  - h) A duplicated IP error occurred.
  - i) All TCP connections are already in use.
  - j) 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

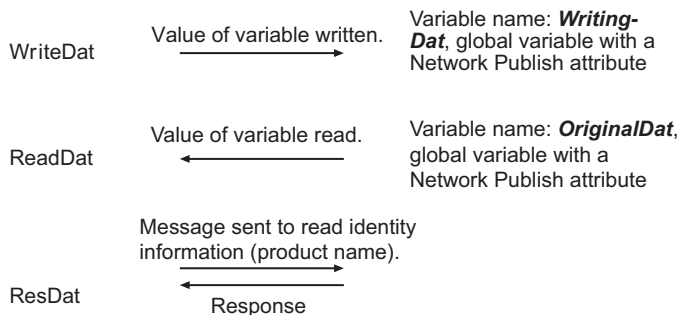
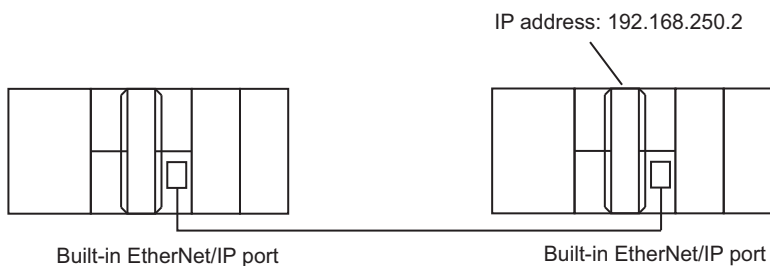
This sample uses CIP class 3 messages to write a variable, read a variable, and send a message. The Controllers are connected to an EtherNet/IP network. The IP address of the remote node is 192.168.250.2.

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. *WritingDat* 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 |

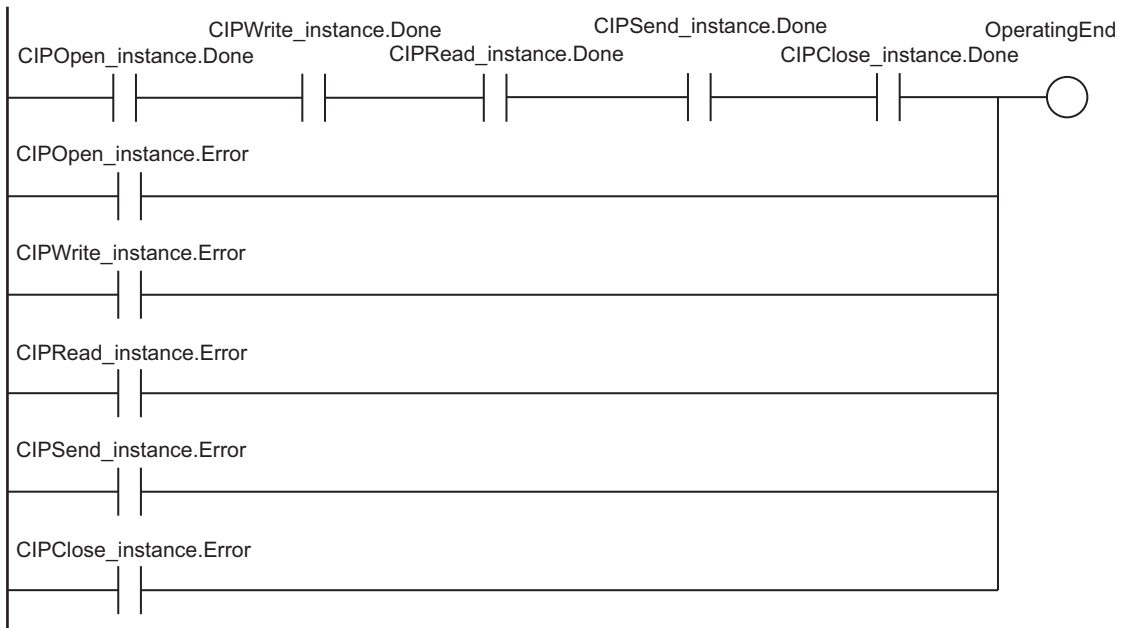
- 5** The CIPClose instruction is used to close the class 3 connection.



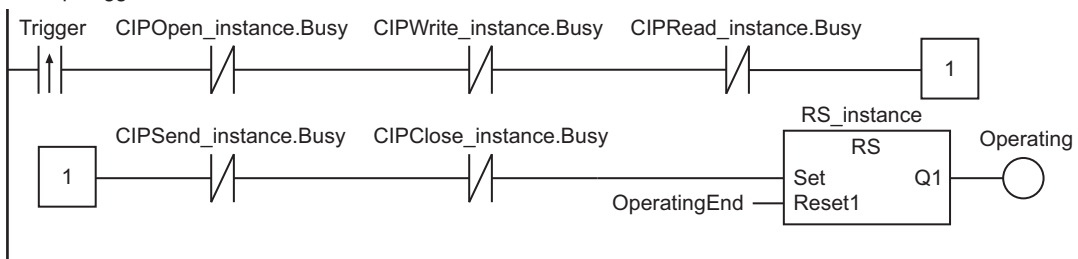
**LD**

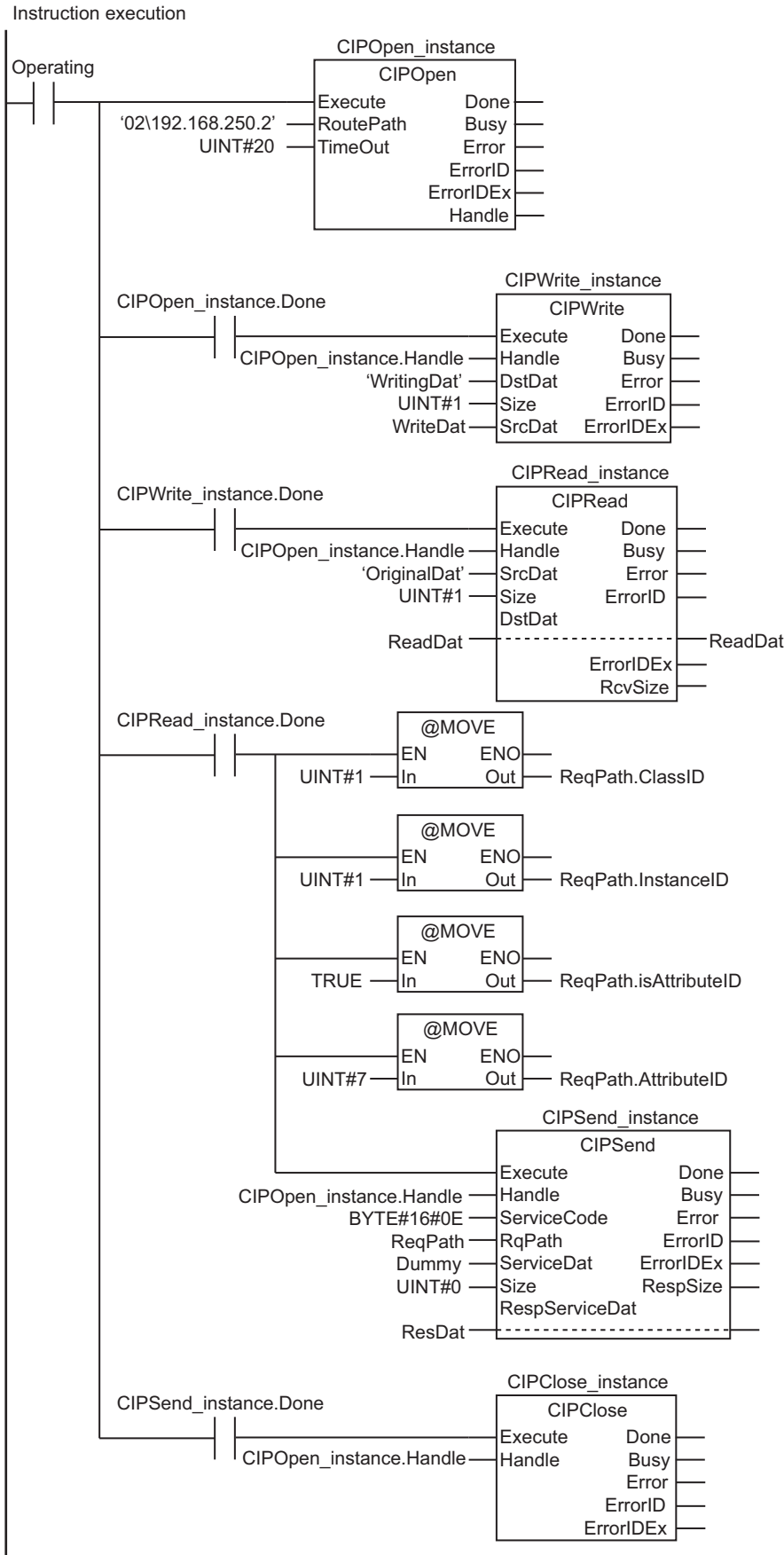
| Variable          | Data type            | Initial value   | Comment              |
|-------------------|----------------------|---|----------------------|
| OperatingEnd      | BOOL                 | FALSE   | Processing completed |
| Trigger           | BOOL                 | FALSE   | Execution 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             |   |                      |

Determine if instruction execution is completed.

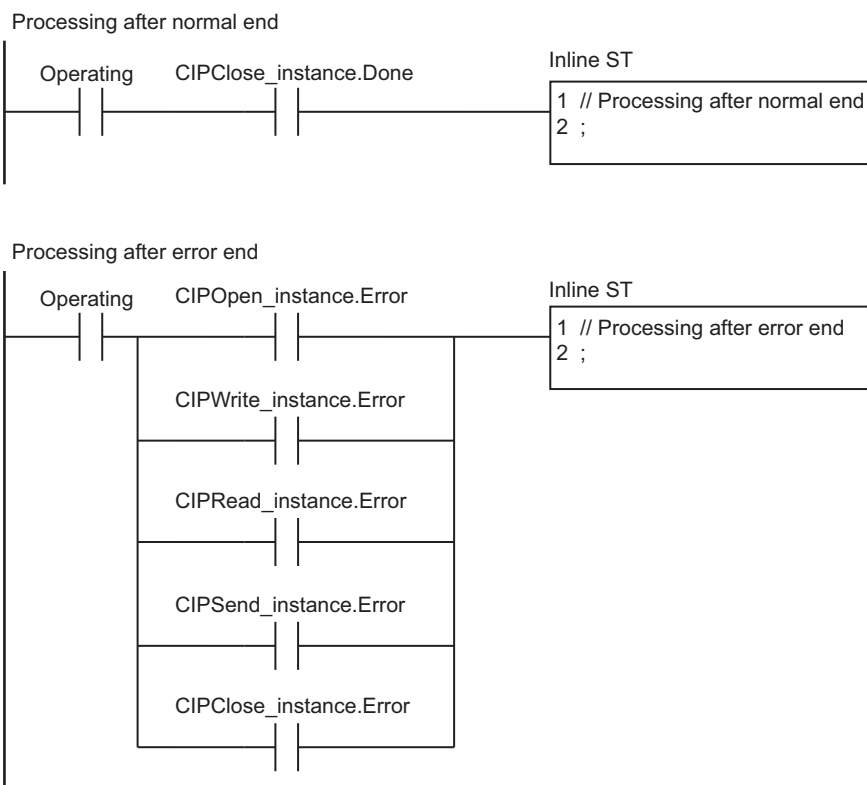


Accept trigger.









## 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
  CIPCclose_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/<br>FUN | Graphic expression | ST expression   |
|--------------------------|---|------------|--------------------|---|
| CIPOpenWith-<br>DataSize | Open CIP Class<br>3 Connection<br>with Specified<br>Data Size | FB         |                    | CIPOpen_instance(Execute, RoutePath, TimeOut, DataSize, Done, Busy, Error, ErrorID, ErrorIDEx, Handle); |



## 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.

## Variables

|           | 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<br>(2 s) |
| 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.

|           | Boo<br>lean  | Bit strings |          |           |           | Integers  |          |           |           |          |         |          |          | Real<br>num<br>bers |           | Times, durations,<br>dates, and text strings |          |         |    |            |
|-----------|--|-------------|----------|-----------|-----------|-----------|----------|-----------|-----------|----------|---------|----------|----------|---------------------|-----------|--|----------|---------|----|------------|
|           | BOO<br>L   | BY<br>TE    | WO<br>RD | DWO<br>RD | LWO<br>RD | US<br>INT | UI<br>NT | UD<br>INT | UL<br>INT | SI<br>NT | IN<br>T | DI<br>NT | LI<br>NT | RE<br>AL            | LR<br>EAL | TI<br>ME                                     | DA<br>TE | TO<br>D | DT | STR<br>ING |
| RoutePath |  |             |          |           |           |           |          |           |           |          |         |          |          |                     |           |  |          |         |    | OK         |
| TimeOut   |  |             |          |           |           |           | OK       |           |           |          |         |          |          |                     |           |  |          |         |    |            |
| DataSize  |  |             |          |           |           |           | OK       |           |           |          |         |          |          |                     |           |  |          |         |    |            |
| Handle    | Refer to <i>Function</i> on page 2-1093 for details on the structure <code>_sCIP_HANDLE</code> . |             |          |           |           |           |          |           |           |          |         |          |          |                     |           |  |          |         |    |            |

## 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 <i>DataSize</i> [bytes] | Service            |
|----------------------------------|--------------------|
| 509 or less                      | Forward_Open       |
| 510 to 8,192*1                   | Large_Forward_Open |

\*1. The range is 510 to 1,994 for NX1P2 and NJ-series CPU Units.

*Handle* is output when the connection is open.

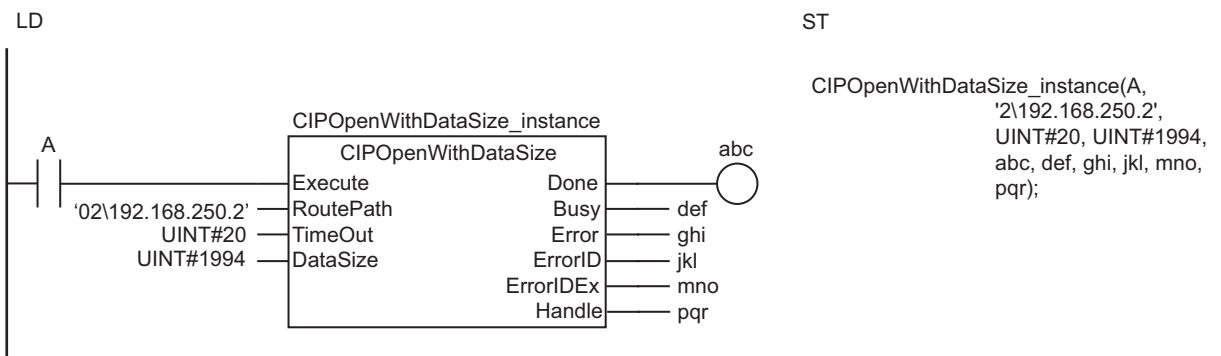
*TimeOut* specifies the connection timeout time. If a response does not return from the remote node within the connection timeout time after the CIPSend, CIPWrite, or CIPRead instruction is executed, it is assumed that communications failed.

The connection timeout time is reset when the CIPRead, CIPWrite, or CIPSend instruction is executed and the remote node returns a response.

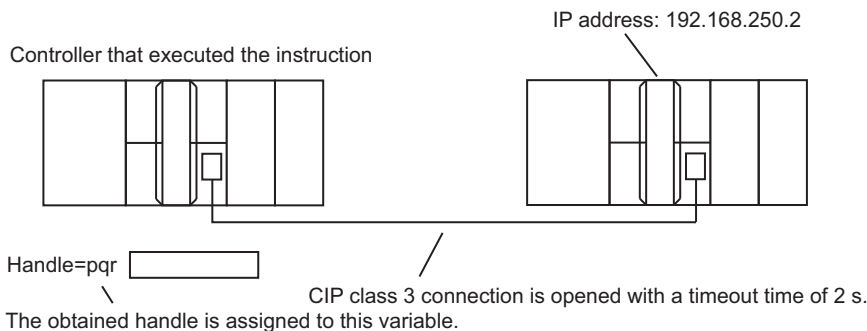
The data type of *Handle* is structure `_sCIP_HANDLE`. The specifications are as follows:

| Name   | Meaning | Description | Data type                 | Valid range           | Unit | Default |
|--------|---------|-------------|---------------------------|-----------------------|------|---------|
| Handle | Handle  | Handle      | <code>_sCIP_HANDLE</code> | ---                   | ---  | ---     |
|        | 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*.



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   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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)*
  - 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 instruction, *CIPOpen* on page 2-1083.

## 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 *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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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.

- h) A duplicated IP error occurred.
- i) All TCP connections are already in use.
- j) 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.

---



# 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/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------------------------|------------|--------------------|--|
| CIPRead     | Read Variable<br>Class 3 Explicit | FB         |                    | CIPRead_instance(Execute, Handle, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize); |

## Variables

|         | Meaning                    | I/O    | Description  | Valid range           | Unit  | Default |
|---------|----------------------------|--------|--|-----------------------|-------|---------|
| Handle  | Handle                     | Input  | Handle obtained with CIPOpen or CIPOpen-WithDataSize 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          |       | 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*1          | Bytes | ---     |

\*1. The range is 0 to 1,988 for NX1P2 and NJ-series CPU Units.

|         | Boo<br>lean  | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-1098 for details on the structure <code>_sCIP_HANDLE</code> . |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |  |
| 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.*1 |  |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |  |
| RcvSize |  |  |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |  |

\*1. 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. If *SrcDat* is not an array, always specify 1.

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              | <i>DataSize</i> in CIPOpenWithDataSize instruction -10 |
|   | STRING                 | <i>DataSize</i> in CIPOpenWithDataSize instruction -8  |
|   | Other data type        | <i>DataSize</i> in CIPOpenWithDataSize instruction -6  |

The data type of *Handle* is structure `_sCIP_HANDLE`. The specifications are as follows:

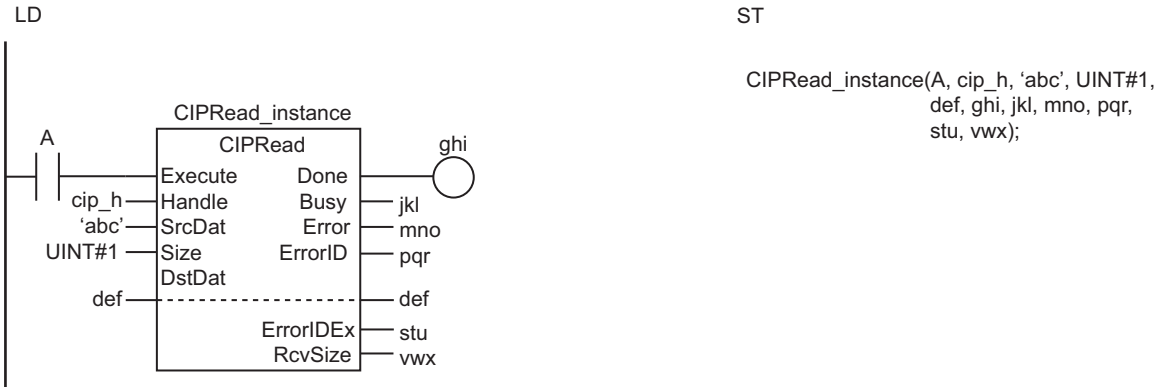
| Name   | Meaning | Description | Data type                 | Valid range           | Unit | Default |
|--------|---------|-------------|---------------------------|-----------------------|------|---------|
| Handle | Handle  | Handle      | <code>_sCIP_HANDLE</code> | ---                   | ---  | ---     |
|        | 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*.

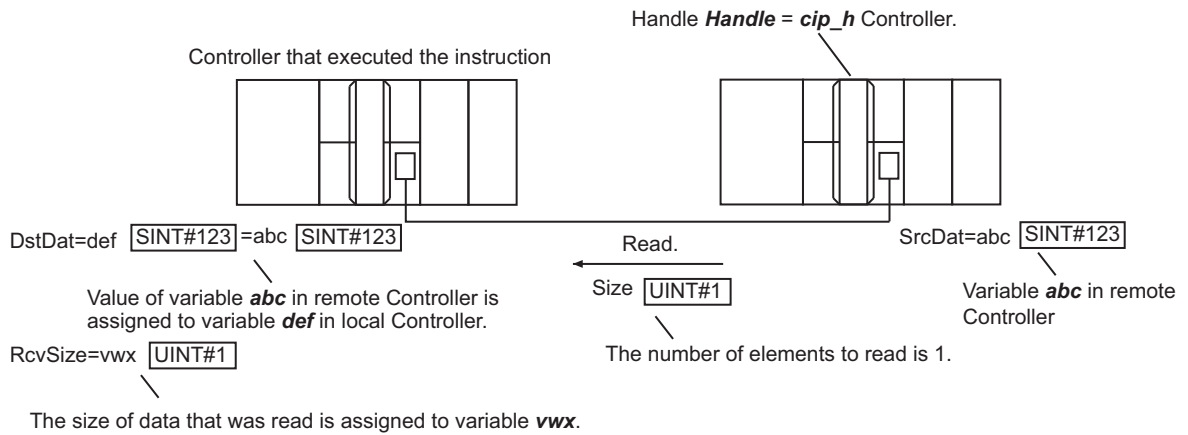
In the following example, the value of variable *abc* in the remote Controller is read and stored in the variable *def* in the local Controller. The number of elements to read *Size* is `UINT#1`.

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`.



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**.



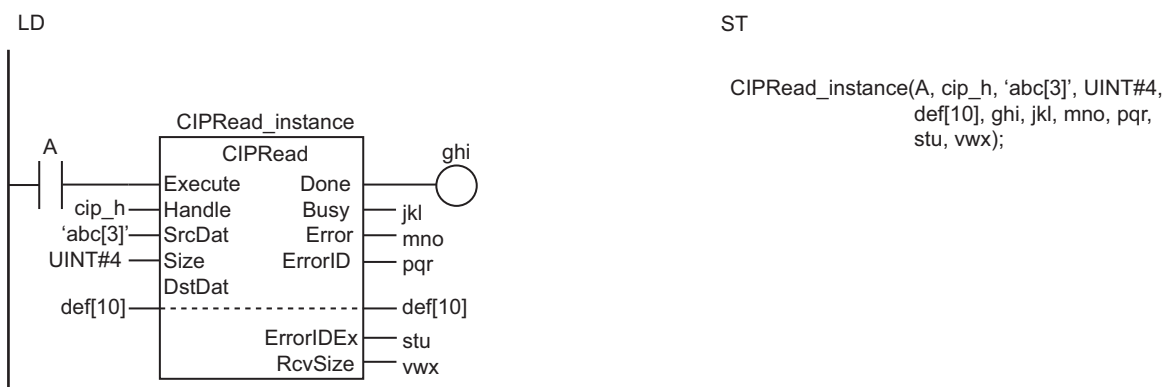
## Reading Arrays

To read array data, pass a subscripted array element to **SrcDat** as the parameter. Also pass a subscripted array element to **DstDat** as the parameter.

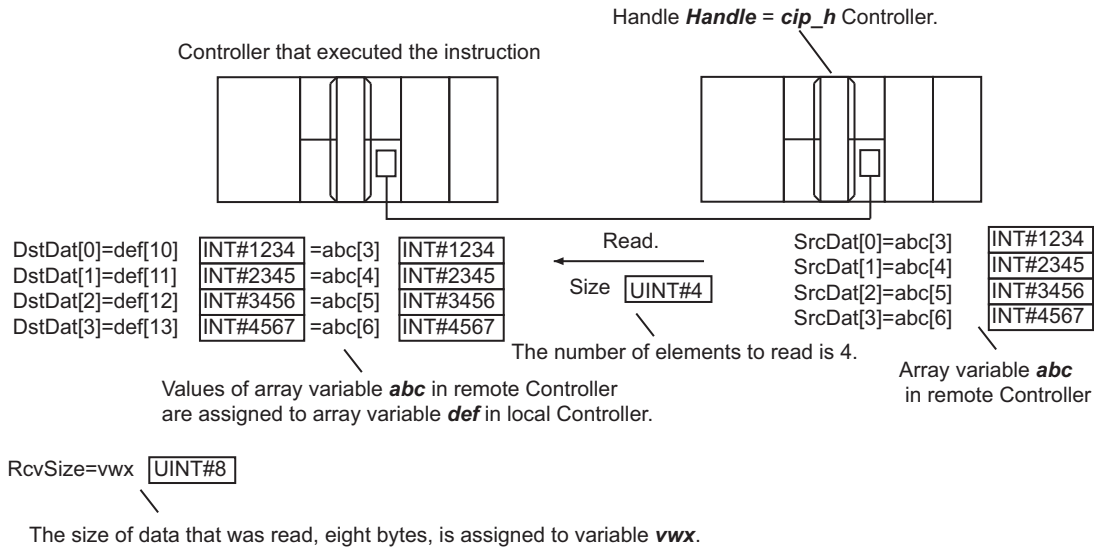
The following example reads the four array variable elements **abc[3]** to **abc[6]** from the remote Controller and stores the results in array variable elements **def[10]** to **def[13]** in the local Controller.

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**.



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   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

- \*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.
- \*3. Use this variable name for port 2 on an NX-series CPU Unit.

### 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)*
- *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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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 | <ul style="list-style-type: none"> <li>• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39)</li> <li>• A text string that consists of only a single '_' (underbar) ASCII character</li> <li>• Any text string that includes two or more consecutive '_' (underbar) ASCII characters</li> <li>• Any text string that starts with an '_' (underbar) ASCII character</li> <li>• Any text string that ends with an '_' (underbar) ASCII character</li> <li>• Any text string that starts with 'P_'</li> </ul> |

- An error occurs in the following cases. *Error* will change to TRUE.
  - a) The value of *Size* is outside of the valid range.
  - b) The text string in *SrcDat* is not valid.
  - c) The data type of the value that was read does not agree with the data type of *DstDat*.
  - d) The size of data that was read exceeds the range of *DstDat*.
  - e) A data type that is not supported was specified for *DstDat*.
  - f) An error response defined by CIP was returned.
  - g) The value of *Handle.Handle* is outside of the valid range.
  - h) More than 32 CIP-related instructions were executed simultaneously.
  - i) The connection that was established with the CIPOpen or CIPOpenWithDataSize instruction has timed out.
  - j) 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. <ul style="list-style-type: none"> <li>• Basic data type</li> <li>• Enumeration</li> <li>• Structure</li> <li>• Union</li> <li>• Array</li> </ul> |

| Value       | Error  |
|-------------|--|
| 16#05000000 | The specified source variable is one of the following and it does not exist on the other Controller. <ul style="list-style-type: none"> <li>• Enumeration enumerator</li> <li>• Structure member</li> <li>• Union member</li> <li>• Array element</li> </ul> |
| 16#08000000 | The requested service does not support.  |
| 16#0C008010 | The specified source variable is being downloaded.   |
| 16#0C008011 |  |
| 16#11000000 | The value of <i>Size</i> 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 *Sample Programming* on page 2-1085 for the CIPOpen instruction.

# 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/<br>FUN | Graphic expression  | ST expression  |
|-------------|------------------------------------|------------|---|--|
| CIPWrite    | Write Variable<br>Class 3 Explicit | FB         | <pre> graph LR     subgraph CIPWrite_instance         CIPWrite     end     CIPWrite -- Execute --- Out1[Done]     CIPWrite -- Handle --- Out2[Busy]     CIPWrite -- DstDat --- Out3[Error]     CIPWrite -- Size --- Out4[ErrorID]     CIPWrite -- SrcDat --- Out5[ErrorIDEx] </pre> | CIPWrite_instance(Execute, Handle, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx); |

## Variables

|        | Meaning                     | I/O   | Description  | Valid range           | Unit | Default |
|--------|-----------------------------|-------|--|-----------------------|------|---------|
| Handle | Handle                      | Input | Handle obtained with CIPOpen or CIPOpen-WithDataSize instruction | ---                   | ---  | ---     |
| 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 8,178*1          |      | 1       |
| SrcDat | Source data                 |       | 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.

|        | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |      | Times, durations, dates, and text strings |      |     |    |        |    |
|--------|--|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|------|---|------|-----|----|--------|----|
|        | BOOL   | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL         | REAL | TIME                                      | DATE | TOD | DT | STRING |    |
| Handle |  |             |      |       |       |       |          |       |       |      |     |      |      |              |      |   |      |     |    |        |    |
|        | Refer to <i>Function</i> on page 2-1103 for details on the structure <code>_sCIP_HANDLE</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |      |   |      |     |    |        |    |
| 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     | OK |
|        | An enumeration, array*1, structure, structure member, or union member can also be specified.     |             |      |       |       |       |          |       |       |      |     |      |      |              |      |   |      |     |    |        |    |

\*1. 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 *SrcDat* is written.

*Size* specifies the number of elements to write.

If *DstDat* is an array, specify the number of elements to write.

If *DstDat* is not an array, always specify 1.

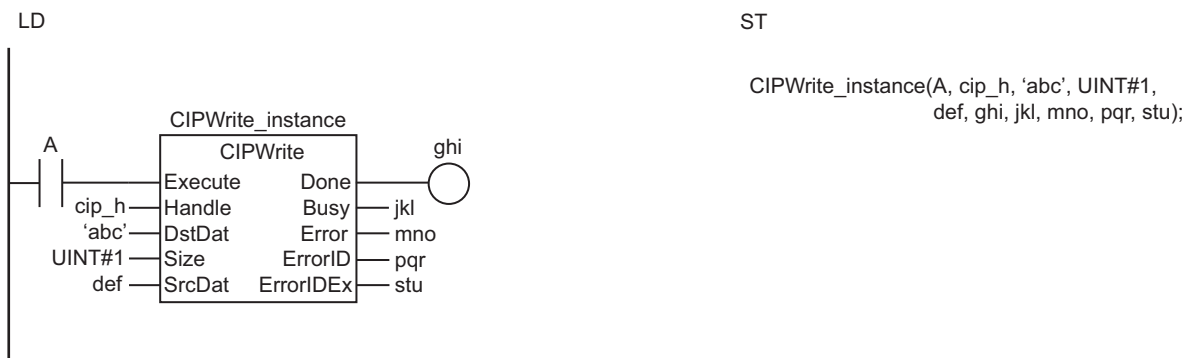
If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

The data type of *Handle* is structure `_sCIP_HANDLE`. The specifications are as follows:

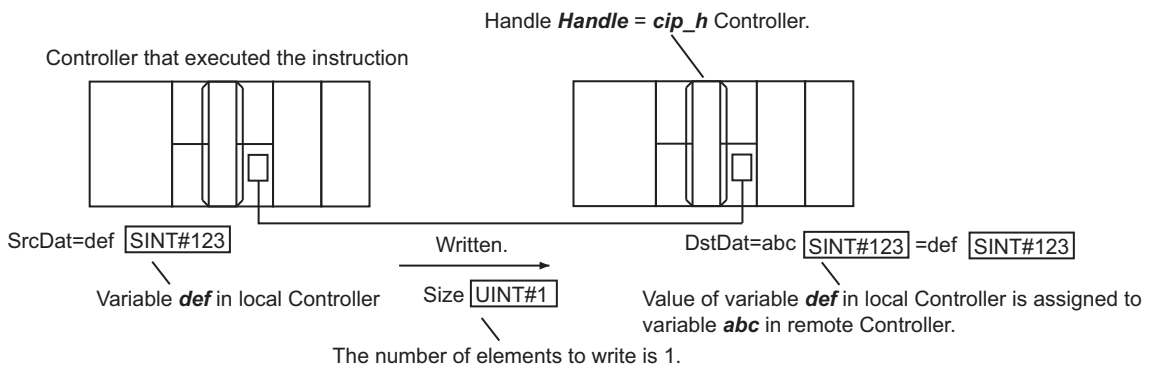
| Name   | Meaning | Description | Data type                 | Valid range           | Unit | Default |
|--------|---------|-------------|---------------------------|-----------------------|------|---------|
| Handle | Handle  | Handle      | <code>_sCIP_HANDLE</code> | ---                   | ---  | ---     |
| 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`.



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.

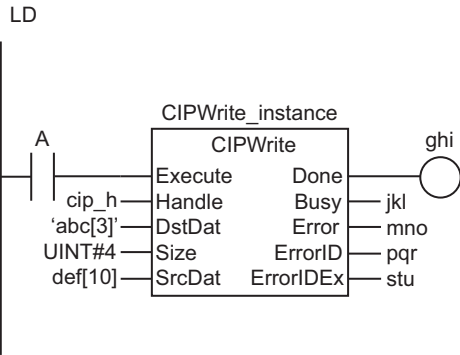


## Writing Arrays

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]`.

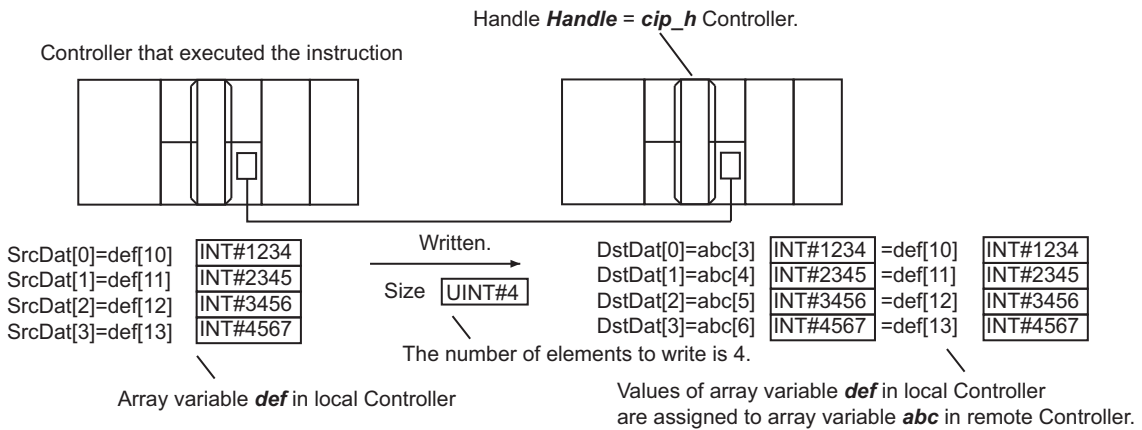




ST

```
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]** to **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  |
|-----------------------|--|
|                       | Connections established with the CIPOpen instruction <ul style="list-style-type: none"> <li>Data type of variable specified for <i>DstDat</i> is a structure: 1,984 bytes</li> <li>Data type of variable specified for <i>DstDat</i> is a STRING: 1,986 bytes</li> <li>Other data types: 1,988 bytes</li> </ul>  |
| Base size             | Connections established with the CIPOpenWithDataSize instruction <ul style="list-style-type: none"> <li>Use the following formula when the data type of variable specified for <i>DstDat</i> is a structure:<br/>                         Base size <b>bytes</b> = <i>DataSize</i> in CIPOpenWithDataSize instruction -10</li> <li>Use the following formula when the data type of variable specified for <i>DstDat</i> is a STRING:<br/>                         Base size <b>bytes</b> = <i>DataSize</i> in CIPOpenWithDataSize instruction -8</li> <li>Use the following formula for other data types.<br/>                         Base size <b>bytes</b> = <i>DataSize</i> in CIPOpenWithDataSize instruction -6</li> </ul> |

| Item in above formula                  | Meaning   |
|--|---|
| Size of variable name of <i>DstDat</i> | <ul style="list-style-type: none"> <li>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.</li> <li>If the number of bytes of ASCII characters in a level is an odd number, add 1.</li> <li>If a level in the structure is an array, add four times the number of dimensions in the array.</li> <li>Periods and commas in the structure and arrays are not included in the variable name size.</li> </ul> <p>Example 1: When the Variable Name of <i>DstDat</i> Is 'aaa.bbbbb[1,2,3].cc'</p> <ul style="list-style-type: none"> <li>The text string "aaa" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes.</li> <li>"bbbb" of "bbbb[1,2,3]" in the second level is a 5-byte text string. It is an odd number, so 1 is added to make 6 bytes.</li> <li>Also "bbbb[1,2,3]" is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes.</li> <li>The text string "cc" in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation.</li> <li>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.</li> </ul> <p>Example 2: When the Variable Name of <i>DstDat</i> Is 'val'</p> <ul style="list-style-type: none"> <li>The text string "val" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes.</li> <li>If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes.</li> </ul> <p>Example 3: When the Variable Name of <i>DstDat</i> Is 'array[8]'</p> <ul style="list-style-type: none"> <li>The text string "array" in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes.</li> <li>It is a one-dimensional array. Therefore, 1 times 4, or 4, is added.</li> <li>If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.</li> </ul> |

## 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.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta*2 |         |           |   |
| _EIP2_EtnOnlineSta*3 |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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)
- 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 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 the execution.
- 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 it. A building error will occur if an enumerator is passed directly.
- You can use this instruction through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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 | <ul style="list-style-type: none"> <li>• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39)</li> <li>• A text string that consists of only a single '_' (underbar) ASCII character</li> <li>• Any text string that includes two or more consecutive '_' (underbar) ASCII characters</li> <li>• Any text string that starts with an '_' (underbar) ASCII character</li> <li>• Any text string that ends with an '_' (underbar) ASCII character</li> <li>• Any text string that starts with 'P_'</li> </ul> |

- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *Size* is outside the valid range.
  - b) The text string in *DstDat* is not valid.
  - c) The value of *Size* exceeds the range of *SrcDat*.
  - d) A data type that is not supported was specified for *SrcDat*.
  - e) An error response defined by CIP was returned.
  - f) The value of *Handle.Handle* is outside the valid range.
  - g) More than 32 CIP-related instructions were executed simultaneously.
  - h) The connection that was established with the CIPOpen or CIPOpenWithDataSize instruction has timed out.
  - i) 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. <ul style="list-style-type: none"> <li>• Basic data type</li> <li>• Enumeration</li> <li>• Structure</li> <li>• Union</li> <li>• Array</li> </ul>                        |
| 16#05000000 | The specified source variable is one of the following and it does not exist on the other Controller. <ul style="list-style-type: none"> <li>• Enumeration enumerator</li> <li>• Structure member</li> <li>• Union member</li> <li>• Array element</li> </ul>                             |
| 16#08000000 | The requested service is not supported.  |
| 16#0C008010 | The specified source variable is being downloaded.   |
| 16#0C008011 |  |
| 16#1F00102  | <ul style="list-style-type: none"> <li>• The specified destination variable has a Constant attribute, so it cannot be written.</li> <li>• The write data does not agree with the number of write elements.</li> </ul>  |
| 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#20008022 | The data type of server variables is different from that of client variables.  |
| 16#20008028 | <ul style="list-style-type: none"> <li>• The specified destination variable is an enumeration and the write data is not the value of an enumerator.</li> <li>• The specified destination variable has a Range Specification attribute and the write data is out of the range.</li> </ul> |
| 16#26000000 | The specified destination variable contains only the NULL character.   |

### Sample Programming

Refer to *Sample Programming* on page 2-1085 for the CIPOpen instruction.

# CIPSend

The CIPSend instruction sends a class 3 CIP message to a specified device on a CIP network.

| Instruction | Name                          | FB/<br>FUN | Graphic expression   | ST expression  |
|-------------|-------------------------------|------------|--|--|
| CIPSend     | Send Explicit Message Class 3 | FB         | <pre> graph LR     subgraph CIPSend_instance [CIPSend_instance]         subgraph CIPSend             Execute --- Done             Handle --- Busy             ServiceCode --- Error             RqPath --- ErrorID             ServiceDat --- ErrorIDEx             Size --- RespSize             RespServiceDat --- RespSize         end     end             </pre> | CIPSend_instance(Execute, Handle, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize); |

## Variables

|                | Meaning                    | I/O    | Description  | Valid range           | Unit  | Default |
|----------------|----------------------------|--------|--|-----------------------|-------|---------|
| Handle         | Handle                     | Input  | Handle obtained with CIPOpen or CIPOpen-WithDataSize instruction | ---                   | ---   | ---     |
| Service-Code   | Service code               |        | Service code   | Depends on data type. |       |         |
| RqPath         | Request path               |        | Request path   | ---                   |       |         |
| ServiceDat     | Service data               |        | Service data to send   | Depends on data type. |       | *1      |
| 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 | ---     |

\*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 |
| Handle       |         | Refer to <i>Function</i> on page 2-1110 for details on the structure <code>_sCIP_HANDLE</code> .   |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Service-Code |         | OK   |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| RqPath       |         | Structure <code>_sREQUEST_PATH</code> or <code>_sREQUEST_PATH_EX</code> *1.<br>Refer to <i>Data type of RqPath</i> on page 2-1110 for details. |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| ServiceDat   |         | OK   | OK   | OK    | OK    | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK           |       |   |      |     |    |        |
| Size         |         |  |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |

|                     | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|---------------------|--|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                     | BOOL   | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| RespServi-<br>ceDat |  | 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.

If *ServiceDat* is not an array, always specify *1*.

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      | <code>_sCIP_HANDLE</code> | ---                   | ---  | ---     |
| Handle | Handle  | Handle      | UDINT                     | Depends on data type. | ---  | ---     |

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 |
|---------------------|---------|
| <code>_8BIT</code>  | 8 bits  |
| <code>_16BIT</code> | 16 bits |
| <code>_32BIT</code> | 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.

## Data type of *RqPath*

The data type of *RqPath* is structure `_sREQUEST_PATH` or `_sREQUEST_PATH_EX`. Normally, use `_sREQUEST_PATH`.

When you specify any logical format size, use `_sREQUEST_PATH_EX`.

### ● `_sREQUEST_PATH` type

| Name          | Meaning         | Description   | Data type                   | Valid range           | Unit | Default |
|---------------|-----------------|---|-----------------------------|-----------------------|------|---------|
| RqPath        | Request path    | Request path  | <code>_sREQUEST_PATH</code> | ---                   | ---  | ---     |
| ClassID       | Class ID        | Class ID  | UINT                        | Depends on data type. | ---  | 0       |
| InstanceID    | Instance ID     | Instance ID   | UINT                        |                       |      | FALSE   |
| isAttributeID | Attribute usage | TRUE:Attribute ID used.<br>FALSE:Attribute ID not used. | BOOL                        |                       |      | 0       |
| 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  | <code>_sREQUEST_PATH_EX</code>    | ---                   | ---  | ---                |
| ClassIDLogicalFormat     | Class ID logical format     | Class ID data size                                      | <code>_eCIP_LOGICAL_FORMAT</code> | Depends on data type. | ---  | <code>_8BIT</code> |
| ClassID                  | Class ID                    | Class ID  | UDINT                             |                       |      | 0                  |
| InstanceIDLogicalFormat  | Instance ID logical format  | Instance ID data size                                   | <code>_eCIP_LOGICAL_FORMAT</code> |                       |      | <code>_8BIT</code> |
| InstanceID               | Instance ID                 | Instance ID   | UDINT                             |                       |      | 0                  |
| isAttributeID            | Attribute usage             | TRUE:Attribute ID used.<br>FALSE:Attribute ID not used. | BOOL                              |                       |      | FALSE              |
| AttributeIDLogicalFormat | Attribute ID logical format | Attribute ID data size                                  | <code>_eCIP_LOGICAL_FORMAT</code> |                       |      | <code>_8BIT</code> |
| AttributeID              | Attribute ID                | Attribute ID  | UDINT                             |                       |      | 0                  |

## 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                    | Up to 8,188 bytes* <sup>1</sup> of response data from the server can be received. |

\*1. The maximum size is 1,990 bytes 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                     | <ul style="list-style-type: none"> <li>Connection established with the CIPOpen instruction: 1,992 bytes</li> <li>Connection established with the CIPOpenWithDataSize instruction: <i>DataSize</i> for the CIPOpenWithDataSize instruction - 2</li> </ul> |
| Attribute usage* <sup>1</sup> | <ul style="list-style-type: none"> <li>Attribute ID used: 14 bytes</li> <li>Attribute ID not used: 10 bytes</li> </ul>   |

\*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* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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)*
- 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 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 the execution.
- 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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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 will occur in the following cases. *Error* will change to TRUE.
  - a) A value that is outside the valid range is set for *RqPath.ClassIDLogicalFormat* or *RqPath.AttributeIDLogicalFormat*.
  - b) 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*.
  - c) The value of *Size* exceeds the write data range.
  - d) The value of *Size* exceeds the range of *ServiceDat*.
  - e) The value of *RespSize* exceeds the range of *RespServiceDat*.
  - f) A data type that is not supported was specified for *ServiceDat*.
  - g) A data type that is not supported was specified for *RespServiceDat*.
  - h) A variable which has any data type other than *\_sREQUEST\_PATH* or *\_sREQUEST\_PATH\_EX* is specified for *RqPath*.
  - i) An error response defined by CIP was returned.
  - j) The value of *Handle.Handle* is outside the valid range.
  - k) More than 32 CIP-related instructions were executed simultaneously.
  - l) The connection that was established with the *CIPOpen* or *CIPOpenWithDataSize* instruction has timed out.
  - m) 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 *Sample Programming* on page 2-1085 for the *CIPOpen* instruction.

# CIPClose

The CIPClose instruction closes the CIP class 3 connection to the specified handle.

| Instruction | Name                                 | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------------------------------|------------|--------------------|--|
| CIPClose    | Close CIP<br>Class 3 Con-<br>nection | FB         |                    | CIPClose_instance(Execute, Handle, Done, Busy, Error, ErrorID, ErrorIDEx); |

## Variables

|        | Meaning | I/O   | Description  | Valid range | Unit | Default |
|--------|---------|-------|--|-------------|------|---------|
| Handle | Handle  | Input | Handle obtained with CIPOpen or CIPOpen-WithDataSize instruction | ---         | ---  | ---     |

|        | Boo<br>lean | Bit strings  |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1114 for details on the structure <code>_sCIP_HANDLE</code> . |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

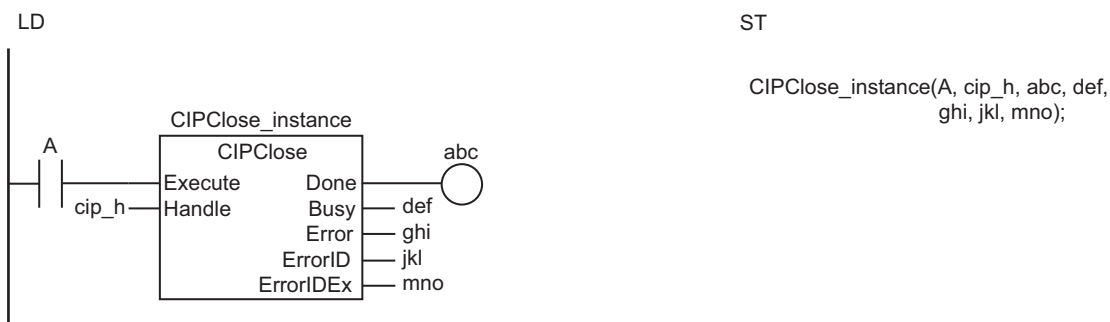
## Function

The CIPClose instruction closes the CIP class 3 connection specified with the handle *Handle*.

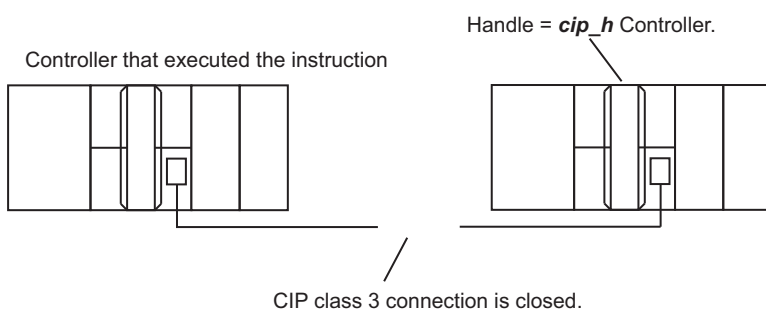
The data type of *Handle* is structure `_sCIP_HANDLE`. The specifications are as follows:

| Name   | Meaning | Description | Data type                 | Valid range           | Unit | Default |
|--------|---------|-------------|---------------------------|-----------------------|------|---------|
| Handle | Handle  | Handle      | <code>_sCIP_HANDLE</code> | ---                   | ---  | ---     |
| 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`).



The CIPClose instruction closes the CIP class 3 connection specified with **Handle**.



## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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)*
- *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 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 the execution.
- 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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.

- This instruction does not use *ErrorIDEx*.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *Handle.Handle* is outside the valid range.
  - b) More than 32 CIP-related instructions were executed simultaneously.

### Sample Programming

Refer to *Sample Programming* on page 2-1085 for the CIPOpen instruction.

# 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/<br>FUN | Graphic expression | ST expression  |
|------------------|--------------------------------|------------|--------------------|--|
| CIPUCMM-<br>Read | Read Variable<br>UCMM Explicit | FB         |                    | CIPUCMMRead_instance(Execute, RoutePath, TimeOut, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize); |

## Variables

|           | 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 | ---      |

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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       |       |       |      |     |      |      |              |       |   |      |     |    |        |
| 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.*1 |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| RcvSize   |  |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |

\*1. 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.

If *SrcDat* is not an array, always specify 1.

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.

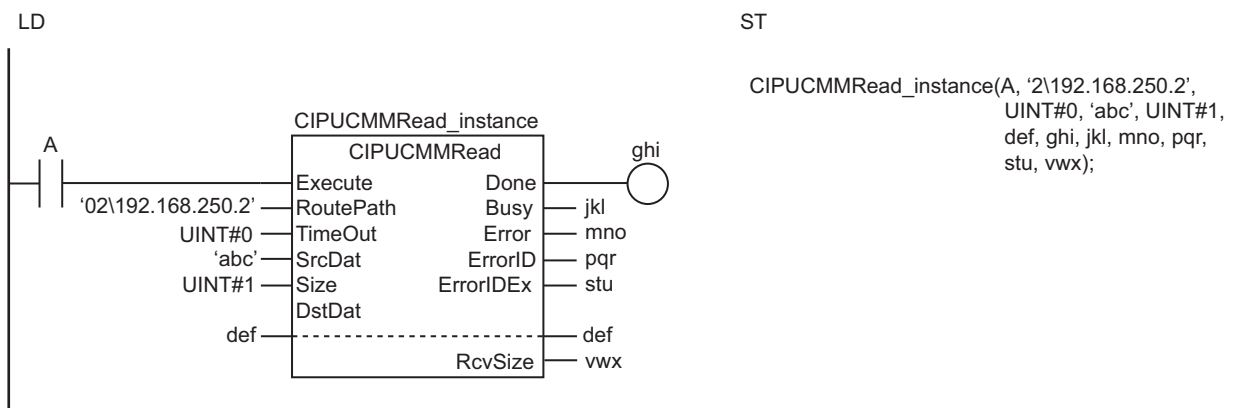
If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

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.

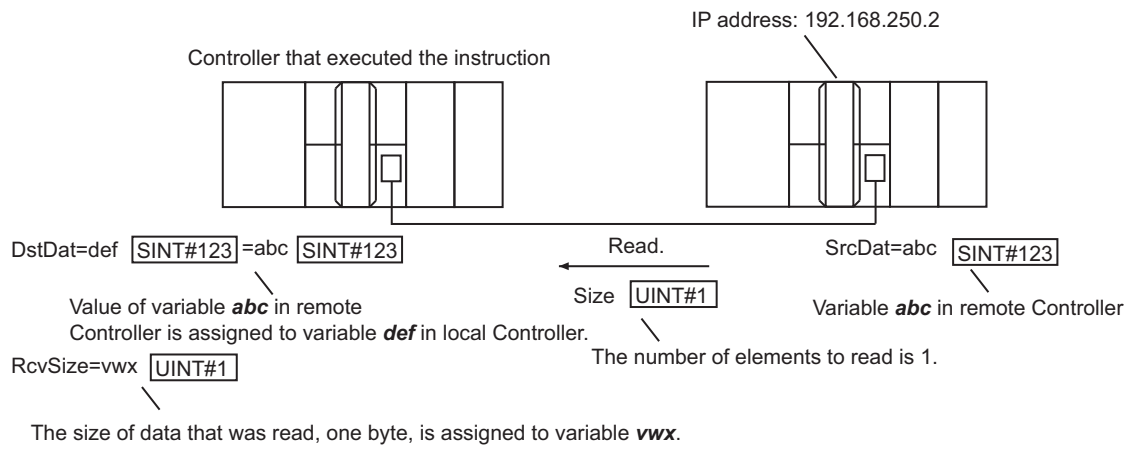
The number of elements to read *Size* is UINT#1.

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.



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**.



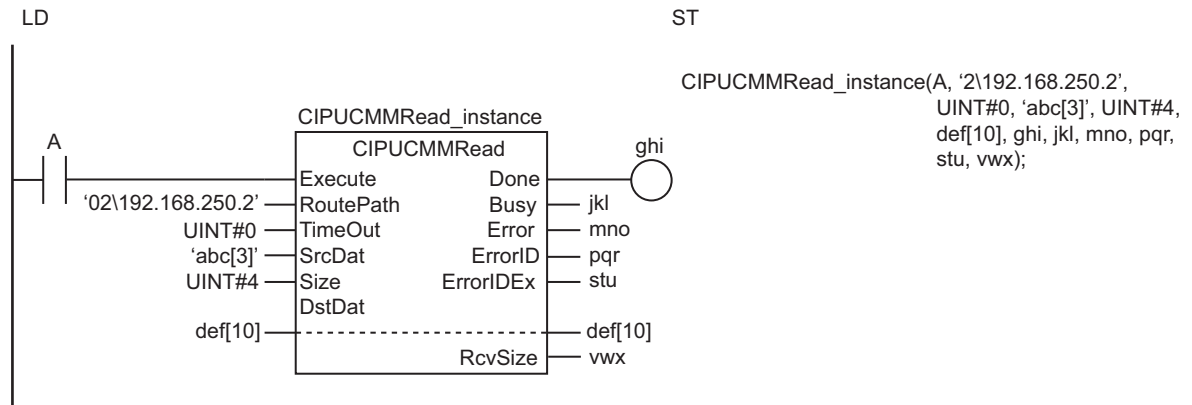
## Reading Arrays

To read array data, pass a subscripted array element to **SrcDat** as the parameter. Also pass a subscripted array element to **DstDat** as the parameter.

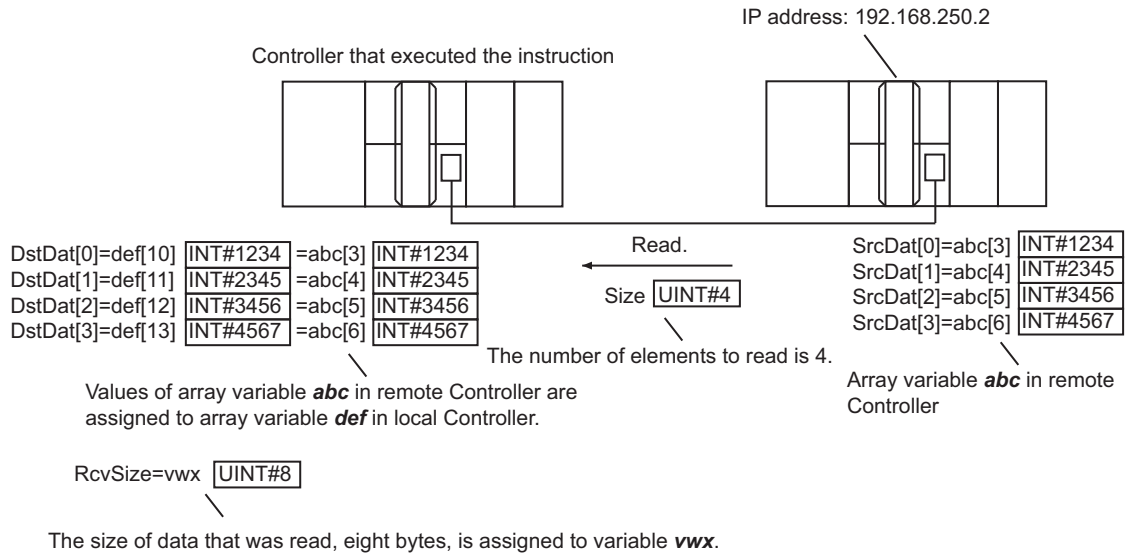
The following example reads the four array variable elements **abc[3]** to **abc[6]** from the remote Controller and stores the results in array variable elements **def[10]** to **def[13]** in the local Controller.

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**.



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   |
|------------------------------------|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> *1  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> *2 |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> *3 |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

### 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)*
- *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 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 the execution.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-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 | <ul style="list-style-type: none"> <li>• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39)</li> <li>• A text string that consists of only a single '_' (underbar) ASCII character</li> <li>• Any text string that includes two or more consecutive '_' (underbar) ASCII characters</li> <li>• Any text string that starts with an '_' (underbar) ASCII character</li> <li>• Any text string that ends with an '_' (underbar) ASCII character</li> <li>• Any text string that starts with 'P_'</li> </ul> |

- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *TimeOut* is outside the valid range.
  - b) The value of *Size* is outside the valid range.
  - c) The text string in *SrcDat* is not valid.
  - d) The data type of the value that was read does not agree with the data type of *DstDat*.
  - e) The size of data that was read exceeds the range of *DstDat*.
  - f) A data type that is not supported was specified for *DstDat*.
  - g) An error response defined by CIP was returned.
  - h) The text string in *RoutePath* is not valid.
  - i) More than 32 CIP-related instructions were executed simultaneously.
  - j) A response was not received even though the timeout time was exceeded.
  - k) There is a setting error for the local IP address.
  - l) The instruction was executed when there was a BOOTP server error.
  - m) 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 | <p>The specified source variable is one of the following data types and it does not exist on the other Controller.</p> <ul style="list-style-type: none"> <li>• Basic data type</li> <li>• Enumeration</li> <li>• Structure</li> <li>• Union</li> <li>• Array</li> </ul> |
| 16#05000000 | <p>The specified source variable is one of the following and it does not exist on the other Controller.</p> <ul style="list-style-type: none"> <li>• Enumeration enumerator</li> <li>• Structure member</li> <li>• Union member</li> <li>• Array element</li> </ul>      |

| Value       | Error   |
|-------------|---|
| 16#08000000 | The requested service does not support.   |
| 16#0C008010 | The specified source variable is being downloaded.  |
| 16#0C008011 |   |
| 16#11000000 | The value of <i>Size</i> 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 *Sample Programming* on page 2-1134 for the CIPUCMMSend instruction.

# 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/<br>FUN | Graphic expression | ST expression  |
|---------------|------------------------------|------------|--------------------|--|
| CIPUCMM-Write | Write Variable UCMM Explicit | FB         |                    | CIPUCMMWrite_instance(Execute, RoutePath, TimeOut, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx); |

## Variables

|           | 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. | ---   | *1       |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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*1, structure, structure member, or union member can also be specified. |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |  |

\*1. 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 *SrcDat* is written.

*Size* specifies the number of elements to write.

If *DstDat* is an array, specify the number of elements to write.

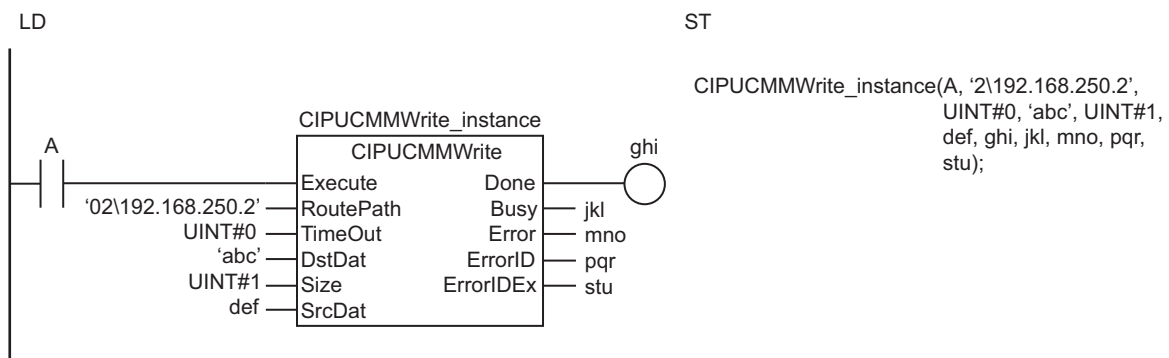
If *DstDat* is not an array, always specify 1.

If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

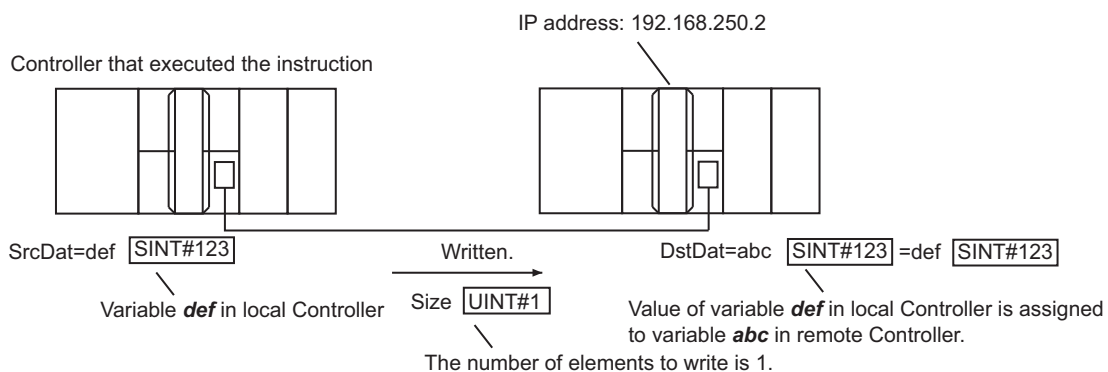
*TimeOut* specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

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.



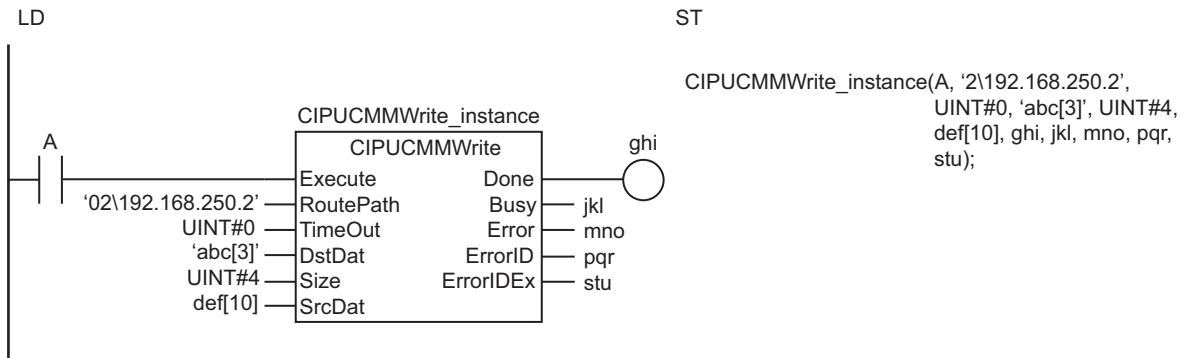
Value of variable *SrcDat* in local Controller is assigned to variable *DstDat* in remote Controller on the CIP network specified by the route path *RoutePath*. *Size* specifies the number of elements to write.



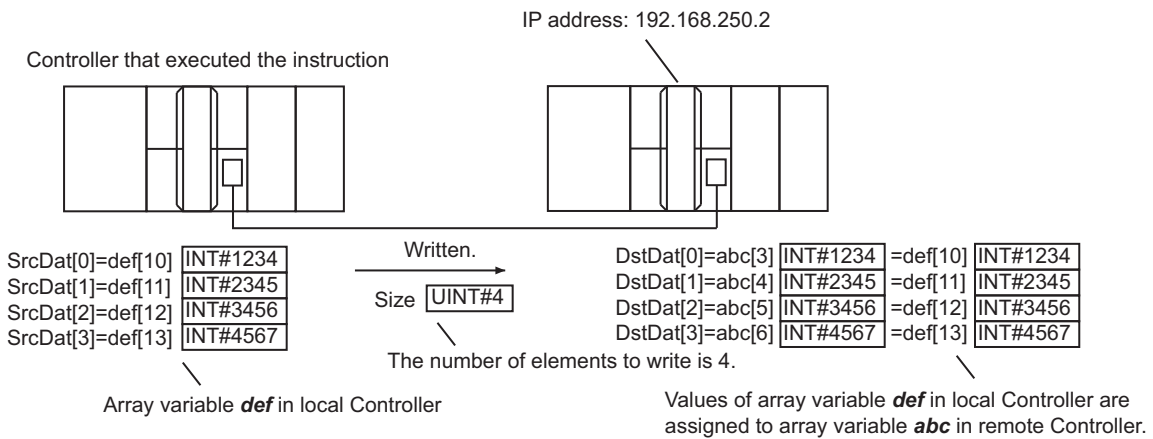
## Writing Arrays

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].



Values of array variable elements **def[10]** to **def[13]** in local Controller are assigned to array variable elements **abc[3]** to **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             | <ul style="list-style-type: none"> <li>Data type of variable specified for <i>DstDat</i> is a structure: 492 bytes</li> <li>Data type of variable specified for <i>DstDat</i> is a STRING: 494 bytes</li> <li>Other data types: 496 bytes</li> </ul> |

| Item in above formula                         | Meaning   |
|---|---|
| <p>Size of variable name of <i>DstDat</i></p> | <ul style="list-style-type: none"> <li>• 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.</li> <li>• If the number of bytes of ASCII characters in a level is an odd number, add 1.</li> <li>• If a level in the structure is an array, add four times the number of dimensions in the array.</li> <li>• Periods and commas in the structure and arrays are not included in the variable name size.</li> </ul> <p>Example 1: When the Variable Name of <i>DstDat</i> Is 'aaa.bbbbb[1,2,3].cc'</p> <ul style="list-style-type: none"> <li>• The text string "aaa" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes.</li> <li>• "bbbb" of "bbbb[1,2,3]" in the second level is a 5-byte text string. It is an odd number, so 1 is added to make 6 bytes.</li> <li>• Also "bbbb[1,2,3]" is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes.</li> <li>• The text string "cc" in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation.</li> <li>• 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.</li> </ul> <p>Example 2: When the Variable Name of <i>DstDat</i> Is 'val'</p> <ul style="list-style-type: none"> <li>• The text string "val" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes.</li> <li>• If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes.</li> </ul> <p>Example 3: When the Variable Name of <i>DstDat</i> Is 'array[8]'</p> <ul style="list-style-type: none"> <li>• The text string "array" in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes.</li> <li>• It is a one-dimensional array. Therefore, 1 times 4, or 4, is added.</li> <li>• If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.</li> </ul> |

| Item in above formula | Meaning   |
|-----------------------|---|
| Path information size | <ul style="list-style-type: none"> <li>• If there are no hops, the path information size is 0 bytes. *1</li> <li>• If there are hops, the path information size is the route path size plus 12 bytes.</li> <li>• The route path size is the bytes size of the ASCII characters in the route path.</li> <li>• However, the following precautions apply.               <ol style="list-style-type: none"> <li>a) If the address portion starts with "#", calculate the network and address portions as a total of 2 bytes.</li> <li>b) If the address portion does not start with "#", calculate the network portion as 2 bytes.</li> <li>c) 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.</li> <li>d) Do not include the level separator, "\", between levels of the route path in the route path size.</li> <li>e) Do not include the first hop in the route path size.</li> </ol> </li> </ul> <p>Example 1: When the Route Path Is '01#11\02\192.168.250.2\01#01'</p> <ul style="list-style-type: none"> <li>• The first hop in the route path size is not included, so ignore "01#11" at the start of the path.</li> <li>• The network type is "02", so use 2 bytes in the calculation.</li> <li>• 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.</li> <li>• For the following "01#01", the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.</li> <li>• If you add all of the above sizes, the size of the route path is 18 bytes.</li> <li>• If we then add 12 bytes to the route path size, the path information size is 30 bytes.</li> </ul> <p>Example 2: When the Route Path Is '02\192.168.250.2\01#00'</p> <ul style="list-style-type: none"> <li>• The first hop in the route path size is not included, so ignore "02\192.168.250.2" at the start of the path.</li> <li>• For the following "01#00", the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.</li> <li>• Therefore, the size of the route path is 2 bytes.</li> <li>• If we then add 12 bytes to the route path size, the path information size is 14 bytes.</li> </ul> <p>Example 3: When the Route Path Is '02\192.168.250.2'</p> <ul style="list-style-type: none"> <li>• If there are no hops, the path information size is 0 bytes.</li> </ul> |

- \*1. 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.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta*2 |         |           |   |
| _EIP2_EtnOnlineSta*3 |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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)*
- *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 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 the execution.
- 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 it. A building error will occur if an enumerator is passed directly.
- You can use this instruction through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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 | <ul style="list-style-type: none"> <li>• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39)</li> <li>• A text string that consists of only a single '_' (underbar) ASCII character</li> <li>• Any text string that includes two or more consecutive '_' (underbar) ASCII characters</li> <li>• Any text string that starts with an '_' (underbar) ASCII character</li> <li>• Any text string that ends with an '_' (underbar) ASCII character</li> <li>• Any text string that starts with 'P_'</li> </ul> |

- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *TimeOut* is outside the valid range.
  - b) The value of *Size* is outside the valid range.
  - c) The text string in *DstDat* is not valid.
  - d) The value of *Size* exceeds the range of *SrcDat*.
  - e) A data type that is not supported was specified for *SrcDat*.
  - f) An error response defined by CIP was returned.
  - g) The text string in *RoutePath* is not valid.



- h) More than 32 CIP-related instructions were executed simultaneously.
  - i) A response was not received even though the timeout time was exceeded.
  - j) There is a setting error for the local IP address.
  - k) 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. <ul style="list-style-type: none"> <li>• Basic data type</li> <li>• Enumeration</li> <li>• Structure</li> <li>• Union</li> <li>• Array</li> </ul> |
| 16#05000000 | The specified source variable is one of the following and it does not exist on the other Controller. <ul style="list-style-type: none"> <li>• Enumeration enumerator</li> <li>• Structure member</li> <li>• Union member</li> <li>• Array element</li> </ul>      |
| 16#08000000 | The requested service does not support.   |
| 16#0C008010 | The specified destination variable is being downloaded.   |
| 16#0C008011 |   |
| 16#1F000102 | <ul style="list-style-type: none"> <li>• The specified destination variable has a Constant attribute, so it cannot be written.</li> <li>• The write data does not agree with the number of write elements.</li> </ul>   |
| 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 | <ul style="list-style-type: none"> <li>• The specified destination variable is an enumeration and the write data is not the value of an enumerator.</li> <li>•</li> </ul>   |
| 16#26000000 | The specified destination variable name is only the NULL character.   |

## Sample Programming

Refer to *Sample Programming* on page 2-1134 for the CIPUCMMSend instruction.

# CIPUCMMSend

The CIPUCMMSend instruction sends a UCMM CIP message to a specified device on a CIP network.

| Instruction      | Name                             | FB/<br>FUN | Graphic expression | ST expression  |
|------------------|----------------------------------|------------|--------------------|--|
| CIPUCMM-<br>Send | Send Explicit<br>Message<br>UCMM | FB         |                    | CIPUCMMSend_instance(Execute, RoutePath, TimeOut, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize); |

## Variables

|                | 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) |
| Service-Code   | Service code               |        | Service code               | Depends on data type. | ---   | ---        |
| RqPath         | Request path               |        | Request path               | ---                   | ---   | ---        |
| ServiceDat     | Command data               |        | Data to send               | Depends on data type. | ---   | *1         |
| 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 | ---        |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|  | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |     |    |        |
|--|--|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|------|--|------|-----|----|--------|
|  | BOOL   | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | REAL | TIME   | DATE | TOD | DT | STRING |
| RoutePath  |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |      |  |      |     |    | OK     |
| TimeOut  |  |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |      |  |      |     |    |        |
| Service-Code   |  | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |      |  |      |     |    |        |
| RqPath   | Structure <code>_sREQUEST_PATH</code> or <code>_sREQUEST_PATH_EX</code> *1.<br>Refer to <i>Data type of RqPath</i> on page 2-1110 for details. |             |      |       |       |       |          |       |       |      |     |      |      |                      |      |  |      |     |    |        |
| ServiceDat   |  | OK          | OK   | OK    | OK    | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK                   |      |  |      |     |    |        |
| An array, structure member, or union member can also be specified. |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |      |  |      |     |    |        |

|                | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| 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.

If *ServiceDat* is not an array, always specify 1.

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 *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 |
|---------------------|---------|
| <code>_8BIT</code>  | 8 bits  |
| <code>_16BIT</code> | 16 bits |
| <code>_32BIT</code> | 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.

## Data type of *RqPath*

The data type of *RqPath* is structure `_sREQUEST_PATH` or `_sREQUEST_PATH_EX`.

Normally, use `_sREQUEST_PATH`.

When you specify any logical format size, use `_sREQUEST_PATH_EX`.

### ● **\_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           |                       |      | FALSE   |
| isAttributeID | Attribute usage | TRUE:Attribute ID used.<br>FALSE:Attribute ID not used. | BOOL           |                       |      | 0       |
| AttributeID   | Attribute ID    | Attribute ID  | UINT           |                       |      |         |

**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.<br>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       |

## 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<br>Attribute ID not used: 10 bytes  |
| Path information size | <ul style="list-style-type: none"> <li>If there are no hops, the path information size is 0 bytes. *2</li> <li>If there are hops, the path information size is the route path size plus 12 bytes.</li> <li>The route path size is the bytes size of the ASCII characters in the route path.</li> <li>However, the following precautions apply. <ul style="list-style-type: none"> <li>a) If the address portion starts with "#", calculate the network and address portions as a total of 2 bytes.</li> <li>b) If the address portion does not start with "#", calculate the network portion as 2 bytes.</li> <li>c) 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.</li> <li>d) Do not include the level separator, "\", between levels of the route path in the route path size.</li> <li>e) Do not include the first hop in the route path size.</li> </ul> </li> </ul> <p>Example 1: When the Route Path Is '01\#11\02\192.168.250.2\01\#01'</p> <ul style="list-style-type: none"> <li>The first hop in the route path size is not included, so ignore "01\#11" at the start of the path.</li> <li>The network type is "02", so use 2 bytes in the calculation.</li> <li>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.</li> <li>For the following "01\#01", the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.</li> <li>If you add all of the above sizes, the size of the route path is 18 bytes.</li> <li>If we then add 12 bytes to the route path size, the path information size is 30 bytes.</li> </ul> <p>Example 2: When the Route Path Is '02\192.168.250.2\01\#00'</p> <ul style="list-style-type: none"> <li>The first hop in the route path size is not included, so ignore "02\192.168.250.2" at the start of the path.</li> <li>For the following "01\#00", the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.</li> <li>Therefore, the size of the route path is 2 bytes.</li> <li>If we then add 12 bytes to the route path size, the path information size is 14 bytes.</li> </ul> <p>Example 3: When the Route Path Is '02\192.168.250.2'</p> <ul style="list-style-type: none"> <li>If there are no hops, the path information size is 0 bytes.</li> </ul> |

\*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.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta*2 |         |           |   |
| _EIP2_EtnOnlineSta*3 |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.
- \*3. Use this variable name for port 2 on an NX-series CPU Unit.

### 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)*
- *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 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 the execution.
- 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 through a built-in EtherNet/IP port on an NJ/NX-series CPU Unit, or through a port on 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 will occur in the following cases. *Error* will change to TRUE.
  - a) A value that is outside the valid range is set for *RqPath.ClassIDLogicalFormat* or *RqPath.AttributeIDLogicalFormat*.
  - b) 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*.
  - c) The value of *TimeOut* is outside the valid range.
  - d) The value of *Size* exceeds the write data range.
  - e) The value of *Size* exceeds the range of *ServiceDat*.
  - f) The value of *RespSize* exceeds the range of *RespServiceDat*.
  - g) A data type that is not supported was specified for *ServiceDat*.
  - h) A data type that is not supported was specified for *RespServiceDat*.
  - i) A variable which has any data type other than `_sREQUEST_PATH` or `_sREQUEST_PATH_EX` is specified for *RqPath*.
  - j) There is a setting error for the local IP address.
  - k) A duplicated IP error occurred.
  - l) The instruction was executed when there was a BOOTP server error.
  - m) An error response defined by CIP was returned.
  - n) The text string in *RoutePath* is not valid.
  - o) More than 32 CIP-related instructions were executed simultaneously.
  - p) A response was not received even though the timeout time was exceeded.

### Sample Programming

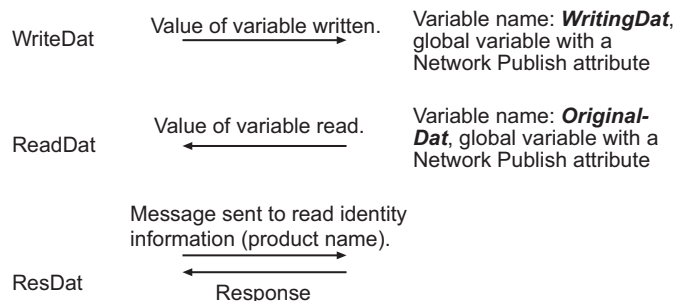
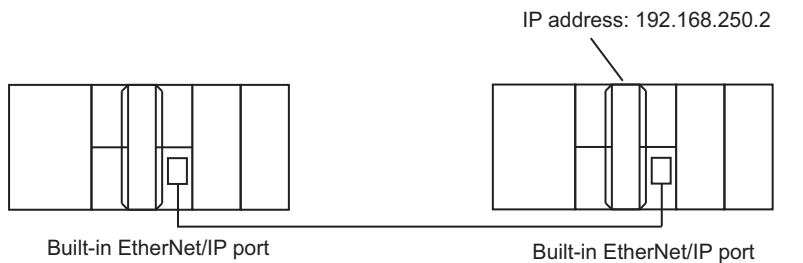
This sample uses CIP UCMM messages to write a variable, read a variable, and send a message.

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 |

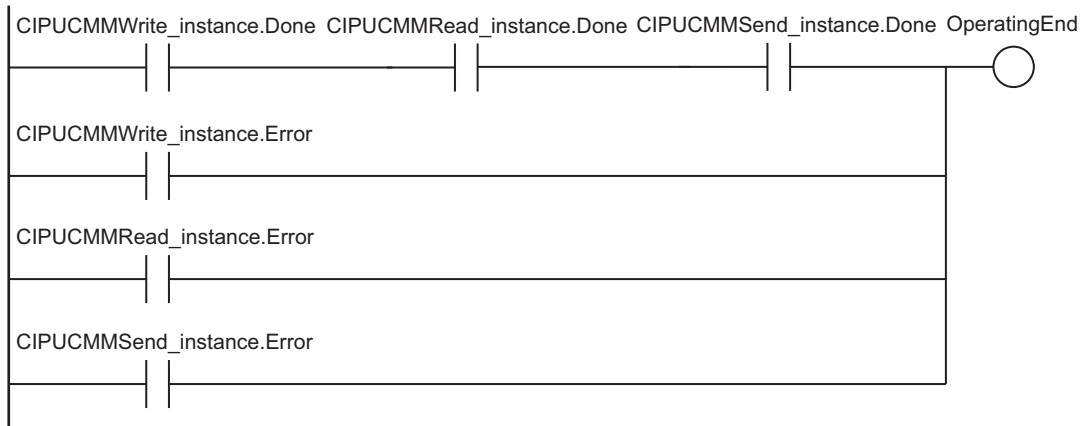


**LD**

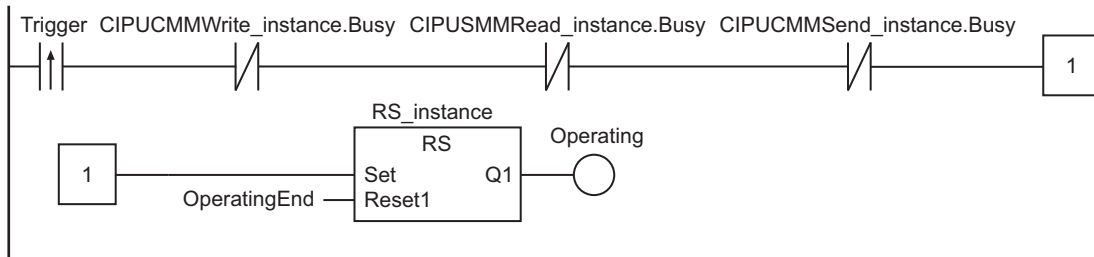
| Variable     | Data type | Initial value | Comment              |
|--------------|-----------|---------------|----------------------|
| OperatingEnd | BOOL      | FALSE         | Processing completed |

| Variable              | Data type            | Initial value   | Comment             |
|-----------------------|----------------------|---|---------------------|
| 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          |   |                     |

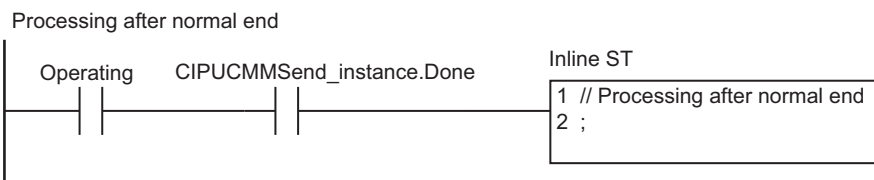
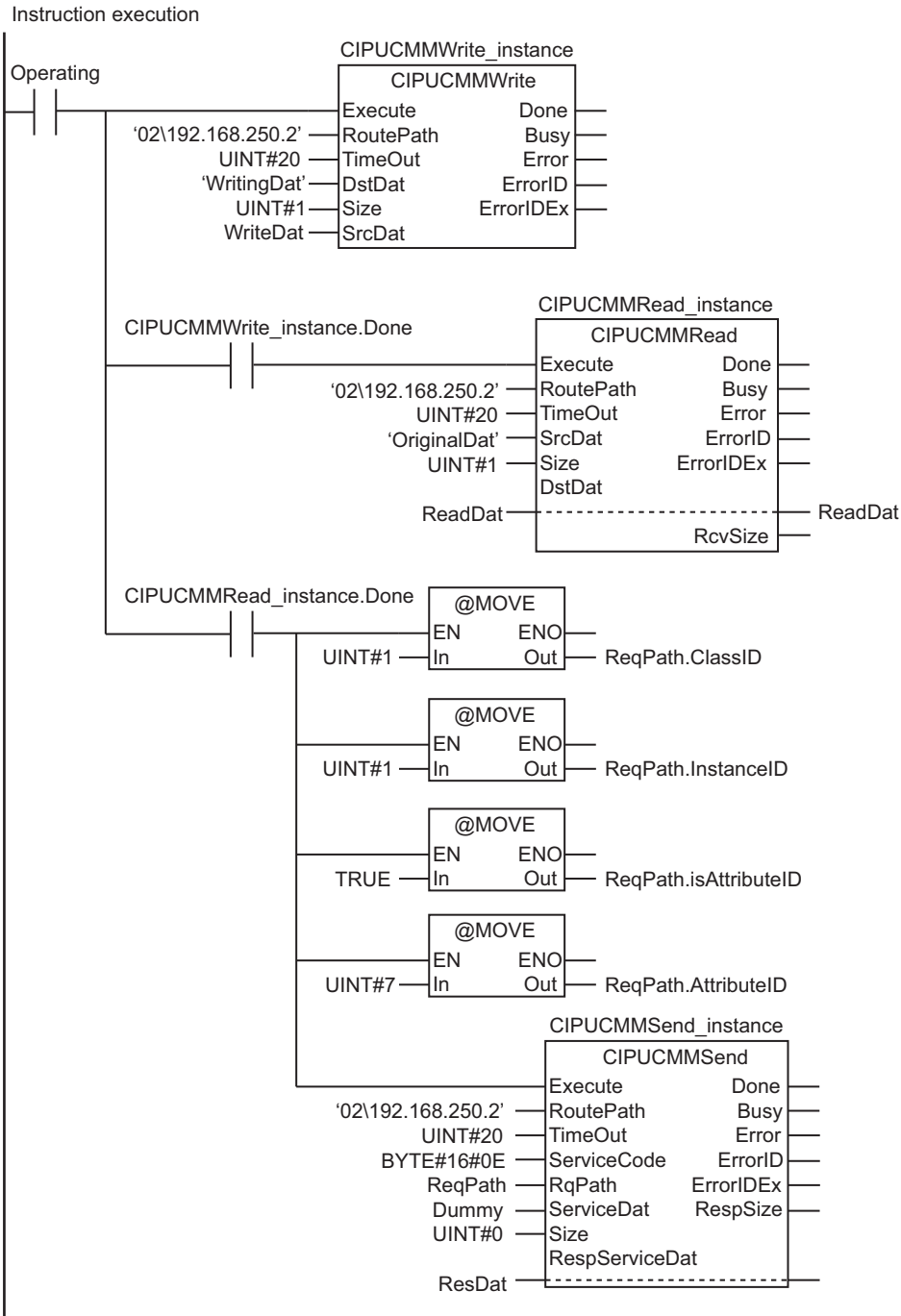
Determine if instruction execution is completed.

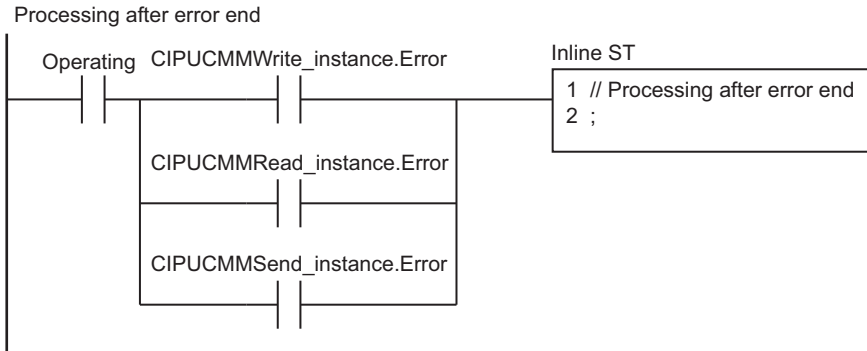


Accept trigger.









## 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 | <input checked="" type="checkbox"/> | 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(
    Execute           :=FALSE,           // Initialize instance.
    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 EtherNet/IP.

| Instruction  | Name              | FB/<br>FUN | Graphic expression | ST expression   |
|--------------|-------------------|------------|--------------------|---|
| SktUDPCreate | Create UDP Socket | FB         |                    | SktUDPCreate_instance(Execute, SrcUdpPort, Done, Busy, Error, ErrorID, Socket); |

## Variables

|            | 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                | ---         | ---  | ---     |

|            | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1141 for details on the structure <code>_sSOCKET</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

## Function

The SktUDPCreate instruction opens the port specified with the local UDP port number *SrcUdpPort*.

To do this, it executes the `Socket()` and `Bind()` socket functions.

Information on the socket that is opened is stored in *Socket*.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

The UDP port is open when 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              | Handle for data communications   | UDINT            | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo               | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. | ---  | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>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* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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.

- Except for NX102 CPU Units, 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, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.

For NX102 CPU Units, a maximum of 64 instructions can be executed.

- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of *SrcUdpPort* is outside of the valid range.
  - c) The port that is specified with *SrcUdpPort* is already open, or close processing is in progress for it.
  - d) 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.

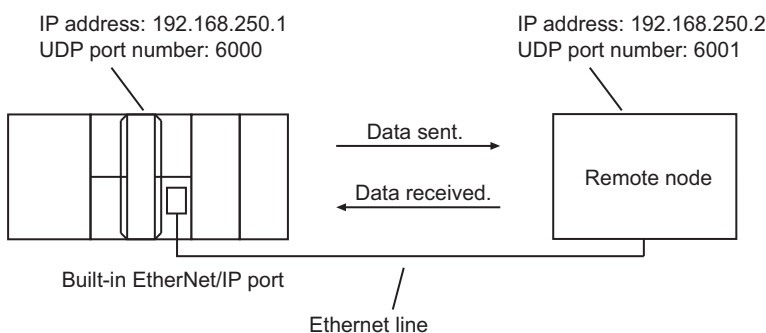
| Unit version of CPU Unit | Number of sockets |
|--------------------------|-------------------|
| 1.03 or later            | 30 max.*1         |
| 1.02 or earlier          | 16 max.           |

\*1. For NX102 CPU Units, a maximum of 60 instructions can be opened.

- 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 CPU Unit and a remote node.



## User program of NJ/NX-series CPU Unit

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      | <input checked="" type="checkbox"/> | 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,<br>SrcAdr:=(PortNo:=0,<br>IpAdr:="),<br>DstAdr:=(PortNo:=0,<br>IpAdr:=")) | Socket              |
|                    | SendSocketDat         | ARRAY[0..1999] OF BYTE | [2000(16#0)]  | Send data           |
|                    | SktUDPCreate_instance | SktUDPCreate           |   |                     |

| Internal Variables | Variable           | Data type | Initial value | Comment |
|--------------------|--------------------|-----------|---------------|---------|
|                    | SkUDPSend_instance | SkUDPSend |               |         |
|                    | SkUDPRcv_instance  | SkUDPRcv  |               |         |
|                    | SkClose_instance   | SkClose   |               |         |

| 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;

```

# SkUDPRcv

The SkUDPRcv instruction reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.

| Instruction | Name               | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------------|------------|--------------------|--|
| SkUDPRcv    | UDP Socket Receive | FB         |                    | SkUDPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize, SendNodeAdr); |

## Variables

|                  | Meaning             | I/O    | Description   | Valid range           | Unit  | Default |
|------------------|---------------------|--------|---|-----------------------|-------|---------|
| Socket           | Socket              | Input  | Socket  | ---                   | ---   | ---     |
| TimeOut          | Timeout time        |        | 0: No timeouts<br>1 to 65535: 0.1 to 6553.5 s       | 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                                 | ---                   | ---   | ---     |

|                  | Boo lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-1150 for details on the structure <code>_sSOCKET</code> .         |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| TimeOut          |  |             |      |       |       |          | OK   |       |       |      |     |      |              |      |   |      |      |     |    |        |
| Size             |  |             |      |       |       |          | OK   |       |       |      |     |      |              |      |   |      |      |     |    |        |
| RcvDat[] (array) |  | OK          |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| RcvSize          |  |             |      |       |       |          | OK   |       |       |      |     |      |              |      |   |      |      |     |    |        |
| SendNodeAdr      | Refer to <i>Function</i> on page 2-1150 for details on the structure <code>_sSOCKET_ADDRESS</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

## 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 period of time that is set with timeout time *TimeOut*.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

Storage of the data to RcvDat[] is completed when 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   | <code>_sSOCKET</code>         | ---                   | ---  | ---     |
| Handle               | Handle              | Handle for data communications   | UDINT                         | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT                          | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING                        | Depends on data type. |      | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT                          | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING                        | Depends on data type. |      | ''      |

\*1. 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                | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo      | Port number         | UDP 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   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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 the 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.
- Except for NX102 CPU Units, 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`, `SktSetOption`, `ModbusTCPCmd`, `ModbusTCPRead`, and `ModbusTCPWrite`.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) Data reception is in progress for the socket specified with *Socket*.
  - c) The socket specified with *Socket* is not open.
  - d) The handle specified by *Socket.Handle* does not exist.

## Sample Programming

Refer to *Sample Programming* on page 2-1143 for the `SktUDPCreate` instruction.

# SktUDPSend

The SktUDPSend instruction sends data from a UDP port for the built-in EtherNet/IP.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------|------------|--------------------|--|
| SktUDPSend  | UDP Socket Send | FB         |                    | SktUDPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID); |

## Variables

|                      | Meaning        | I/O   | Description    | Valid range           | Unit  | Default |
|----------------------|----------------|-------|----------------|-----------------------|-------|---------|
| Socket               | Socket         | Input | Socket         | ---                   | ---   | ---     |
| SendDat[]<br>(array) | Send data      |       | Send data      | Depends on data type. |       |         |
| Size                 | Send data size |       | Send data size | 0 to 2000             | Bytes | 1       |

|                      | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |  |
|----------------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|-------------------|------|--|------|------|-----|----|--------|--|
|                      | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT              | REAL | LRREAL                                       | TIME | DATE | TOD | DT | STRING |  |
| Socket               |             |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |
| SendDat[]<br>(array) |             | OK          |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |
| Size                 |             |             |      |       |       |          | OK   |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |

## Function

The SktUDPSend instruction sends SendDat[] (send data) 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*.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

Transmission of SendDat[] to the send buffer is completed when 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              | Handle for data communications   | UDINT            | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>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.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. |      | ---     |

\*1. These members are not used for this instruction.

## Related System-defined Variables

| Name                             | Meaning | Data type | Description   |
|----------------------------------|---------|-----------|---|
| _EIP_EtnOnlineSta* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.

- 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.
- Except for NX102 CPU Units, 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`, `SktSetOption`, `ModbusTCPcmd`, `ModbusTCPRead`, and `ModbusTCPWrite`.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of a member of *Socket* is outside of the valid range.
  - c) Data transmission is in progress for the socket specified with *Socket*.
  - d) The socket specified with *Socket* is not open.
  - e) The remote node for *Socket* was specified with a domain name and address resolution failed.
  - f) The handle specified by *Socket.Handle* does not exist.
  - g) The value of *Size* exceeds the number of elements in `SendDat[]`.

### Sample Programming

Refer to *Sample Programming* on page 2-1143 for the `SktUDPCreate` instruction.

# SktTCPAccept

The SktTCPAccept instruction requests accepting the TCP socket for the built-in EtherNet/IP.

| Instruction  | Name              | FB/<br>FUN | Graphic expression | ST expression  |
|--------------|-------------------|------------|--------------------|--|
| SktTCPAccept | Accept TCP Socket | FB         |                    | SktTCPAccept_instance(Execute, SrcTcpPort, TimeOut, Done, Busy, Error, ErrorID, Socket); |

## Variables

|            | 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           |        | 0: No timeouts<br>1 to 65535: 0.1 to 6553.5 s | Depends on data type. | 0.1 s | 0       |
| Socket     | Socket                 | Output | Socket  | ---                   | ---   | ---     |

|            | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 <i>Function</i> on page 2-1155 for details on the structure <code>_sSOCKET</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

## Function

The SktTCPAccept instruction requests accepting the port specified with the local TCP port number *SrcTcpPort*. To do this, it executes the Socket(), Bind(), Listen(), and Accept() socket functions. The instruction waits for the period of time set with timeout time *TimeOut* for a connection to be established with the remote node.

The value of *Done* changes to TRUE when processing of the instruction is completed normally. The connection is established when 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              | 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.<br>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.<br>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.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta*2 |         |           |   |
| _EIP2_EtnOnlineSta*3 |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.

- 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.
- Except for NX102 CPU Units, 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, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of *SrcTcpPort* is outside of the valid range.
  - c) Open processing is in progress for the socket specified with *SrcTcpPort*.
  - d) Close processing is in progress for the socket specified with *SrcTcpPort*.
  - e) 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 later            | 30 max. *1        |
| 1.02 or earlier          | 16 max.           |

\*1. For NX102 CPU Units, a maximum of 60 instructions can be opened.

- 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 *Sample Programming* on page 2-1160 for the SktTCPConnect instruction.

# SkdTCPConnect

The SkdTCPConnect instruction connects to a remote TCP port from the built-in EtherNet/IP.

| Instruction   | Name               | FB/<br>FUN | Graphic expression | ST expression  |
|---------------|--------------------|------------|--------------------|--|
| SkdTCPConnect | Connect TCP Socket | FB         |                    | SkdTCPConnect_instance(Execute, SrcTcpPort, DstAdr, DstTcpPort, Done, Busy, Error, ErrorID, Socket); |

## Variables

|            | Meaning                     | I/O    | Description  | Valid range           | Unit | Default |
|------------|-----------------------------|--------|--|-----------------------|------|---------|
| SrcTcpPort | Local TCP port number.      | Input  | Local TCP port number.<br>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   | ---                   | ---  | ---     |

|            | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1158 for details on the structure <code>_sSOCKET</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The SkdTCPConnect 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 value of *Done* changes to TRUE when processing of the instruction is completed normally.

The connection is established when 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   | <code>_sSOCKET</code>         | ---                   | ---  | ---     |
| Handle  | Handle              | Handle for data communications   | UDINT                         | Depends on data type. | ---  | 0       |
| SrcAdr  | Local address       | Local IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo  | Port number         | Port number  | UINT                          | 1 to 65535            | ---  | 0       |
| IpAdr*1 | IP address          | IP address or host name.<br>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   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo  | Port number         | Port number  | UINT                          | 1 to 65535            | ---  | 0       |
| IpAdr   | IP address          | IP address or host name.<br>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   |
|------------------------------------|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> *1  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> *2 |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> *3 |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.

- 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.
- Except for NX102 CPU Units, 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`, `SktSetOption`, `ModbusTCPcmd`, `ModbusTCPRead`, and `ModbusTCPWrite`.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of *DstAdr* is outside of the valid range.
  - c) The value of *DstTcpPort* is outside of the valid range.
  - d) The TCP port that is specified with *SrcTcpPort* is already open.
  - e) The remote node that is specified with *DstAdr* does not exist.
  - f) The remote node that is specified with *DstAdr* and *DstTcpPort* is not waiting for a connection.
  - g) Address resolution failed for the host name that is specified with *DstAdr*.
  - h) 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.

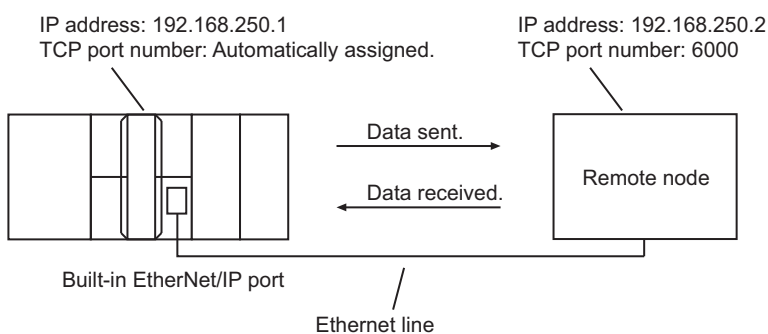
| Unit version of CPU Unit | Number of sockets |
|--------------------------|-------------------|
| 1.03 or later            | 30 max.*1         |
| 1.02 or earlier          | 16 max.           |

\*1. For NX102 CPU Units, a maximum of 60 instructions can be opened.

- 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 CPU Unit and a remote node.



## User program of NJ/NX-series CPU Unit

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           |
|                    | SkTcPcOnnecT_instance      | SkTcPcOnnecT           |   |                     |
|                    | SkTcLearBuf_instance       | SkTcLearBuf            |   |                     |
|                    | SkTcGecTcPcStacUs_instance | SkTcGecTcPcStacUs      |   |                     |
|                    | SkTcTcPcScnd_instance      | SkTcTcPcScnd           |   |                     |
|                    | SkTcTcPcRcv_instance       | SkTcTcPcRcv            |   |                     |
|                    | SkTcClacSe_instance        | SkTcClacSe             |   |                     |

| External Variables | Variable          | Data type | Constant                            | Comment |
|--------------------|-------------------|-----------|-------------------------------------|---------|
|                    | _Eip_EtnOnlineSta | BOOL      | <input checked="" type="checkbox"/> | 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      | <input checked="" type="checkbox"/> | Online  |

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
```

```

Stage:=INT#1;
SkTcPAccept_instance(Execute:=FALSE); // Initialize instance.
SkTcPSend_instance( // Initialize instance.
    Execute:=FALSE,
    SendDat:=SendSocketDat[0]); // Dummy
SkTcPRcv_instance( // Initialize instance.
    Execute:=FALSE,
    RcvDat :=RcvSocketDat[0]); // Dummy
SkTcClose_instance(Execute:=FALSE); // Initialize instance.
END_IF;

IF (DoTCP=TRUE) THEN
CASE Stage OF
1 : // Request accepting a socket connection.
    SkTcPAccept_instance(
        Execute :=TRUE,
        SrcTcpPort:=UINT#6000, // Local TCP port number
        TimeOut :=UINT#0, // Timeout time
        Socket =>WkSocket); // Socket

    IF (SkTcPAccept_instance.Done=TRUE) THEN
        Stage:=INT#2; // Normal end
    ELSIF (SkTcPAccept_instance.Error=TRUE) THEN
        Stage:=INT#10; // Error end
    END_IF;
2 : // Request receiving data
    SkTcPRcv_instance(
        Execute:=TRUE,
        Socket :=WkSocket, // Socket
        TimeOut:=UINT#0, // Timeout time
        Size :=UINT#2000, // Receive data size
        RcvDat :=RcvSocketDat[0]); // Receive data

    IF (SkTcPRcv_instance.Done=TRUE) THEN
        Stage:=INT#3; // Normal end
    ELSIF (SkTcPRcv_instance.Error=TRUE) THEN
        Stage:=INT#20; // Error end
    END_IF;
3 : // Request sending data.
    SendSocketDat:=RcvSocketDat;
    SkTcPSend_instance(
        Execute:=TRUE,
        Socket :=WkSocket, // Socket
        SendDat:=SendSocketDat[0], // Send data
        Size :=UINT#2000); // Send data size

```

```
IF (SkdTCPSend_instance.Done=TRUE) THEN
    Stage:=INT#4;                // Normal end
ELSIF (SkdTCPSend_instance.Error=TRUE) THEN
    Stage:=INT#30;               // Error end
END_IF;

4 :                               // Request closing.
SkdClose_instance(
    Execute:=TRUE,
    Socket :=WkSocket);         // Socket

IF (SkdClose_instance.Done=TRUE) THEN
    Stage:=INT#0;                // Normal end
ELSIF (SkdClose_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/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------------|------------|--------------------|--|
| SkTTCPRcv   | TCP Socket Receive | FB         |                    | SkTTCPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize); |

## Variables

|                  | Meaning           | I/O    | Description   | Valid range           | Unit  | Default |
|------------------|-------------------|--------|---|-----------------------|-------|---------|
| Socket           | Socket            | Input  | Socket  | ---                   | ---   | ---     |
| TimeOut          | Timeout time      |        | 0: No timeouts<br>1 to 65535: 0.1 to 6553.5 s       | 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 | ---     |

|                  | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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 <i>Function</i> on page 2-1167 for details on the structure <code>_sSOCKET</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| 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 period of time that is set with timeout time *TimeOut*.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

Storage of the data to *RcvDat[]* is completed when 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   | <code>_sSOCKET</code>         | ---                   | ---  | ---     |
| Handle               | Handle              | Handle for data communications   | UDINT                         | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT                          | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING                        | Depends on data type. | ---  | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT                          | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING                        | Depends on data type. | ---  | ''      |

\*1. These members are not used for this instruction.

## Related System-defined Variables

| Name   | Meaning | Data type | Description   |
|--|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> * <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> * <sup>2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> * <sup>3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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 the 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.
- Except for NX102 CPU Units, 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*, *SktSetOption*, *ModbusTCPcmd*, *ModbusTCPRead*, and *ModbusTCPWrite*.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of a member of *Socket* is outside of the valid range.
  - c) Data reception is in progress for the socket specified with *Socket*.
  - d) The socket specified with *Socket* is not connected.
  - e) The handle specified by *Socket.Handle* does not exist.
  - f) Data was not received before the time that is specified with *TimeOut* expired.
  - g) The socket was closed with the *SktClose* instruction.

## Sample Programming

Refer to *Sample Programming* on page 2-1160 for the *SktTCPConnect* instruction.

# SktTCPSend

The SktTCPSend instruction sends data from a TCP port for the built-in EtherNet/IP.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------|------------|--------------------|--|
| SktTCPSend  | TCP Socket Send | FB         |                    | SktTCPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID); |

## Variables

|                      | Meaning        | I/O   | Description    | Valid range           | Unit  | Default |
|----------------------|----------------|-------|----------------|-----------------------|-------|---------|
| Socket               | Socket         | Input | Socket         | ---                   | ---   | ---     |
| SendDat[]<br>(array) | Send data      |       | Send data      | Depends on data type. |       |         |
| Size                 | Send data size |       | Send data size | 0 to 2000             | Bytes | 1       |

|                      | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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               |             |             |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |
| SendDat[]<br>(array) |             | OK          |      |       |       |          |      |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |
| Size                 |             |             |      |       |       |          | OK   |       |       |      |     |      |                   |      |  |      |      |     |    |        |  |

## Function

The SktTCPSend instruction sends SendDat[] (send data) 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* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. |      | "       |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. |      | "       |

\*1. These members are not used for this instruction.

## Related System-defined Variables

| Name                             | Meaning | Data type | Description   |
|----------------------------------|---------|-----------|---|
| _EIP_EtnOnlineSta* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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.
- Except for NX102 CPU Units, 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, SktSetOption, ModbusTCPCmd, ModbusTCPRead, and ModbusTCPWrite.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of a member of *Socket* is outside of the valid range.
  - c) Data transmission is in progress for the socket specified with *Socket*.
  - d) The socket specified with *Socket* is not connected.
  - e) The handle specified by *Socket.Handle* does not exist.

### Sample Programming

Refer to *Sample Programming* on page 2-1160 for the SktTCPConnect instruction.

# SktGetTCPStatus

The SktGetTCPStatus instruction reads the status of a TCP socket.

| Instruction          | Name                      | FB/<br>FUN | Graphic expression | ST expression  |
|----------------------|---------------------------|------------|--------------------|--|
| SktGetTCP-<br>Status | Read TCP<br>Socket Status | FB         |                    | SktGetTCPStatus_instance( Exe-<br>cute, Socket, Done, Busy, Error,<br>ErrorID, TcpStatus, DatRcvFlag); |

## Variables

|            | Meaning               | I/O    | Description   | Valid range                | Unit | Default |
|------------|-----------------------|--------|---|----------------------------|------|---------|
| Socket     | Socket                | Input  | Socket  | ---                        | ---  | ---     |
| TcpStatus  | TCP connection status | Output | TCP connection status   | *1                         | ---  | ---     |
| DatRcvFlag | Data Received Flag    |        | TRUE: Data is re-<br>ceived.<br>FALSE: Data is not re-<br>ceived. | Depends on da-<br>ta type. |      |         |

\*1. \_CLOSED, \_LISTEN, \_SYN\_SENT, \_SYN\_RECEIVED, \_ESTABLISHED, \_CLOSE\_WAIT, \_FIN\_WAIT1, \_CLOSING, \_LAST\_ACK, \_FIN\_WAIT2, or \_TIME\_WAIT

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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     |             |             |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| TcpStatus  |             |             |      |       |       |          |      |       |       |      |     |      |      |                   |       |  |      |     |    |        |
| 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 *DatRcvFlag* changes to TRUE.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

Storage of the data to *TcpStatus* and *DatRcvFlag* is completed when 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              | Handle for data communications   | UDINT            | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. | ---  | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. | ---  | ''      |

\*1. These members are not used for this instruction.

The data type of *TcpStatus* is an enumerated type, `_eCONNECTION_STATE`.

The enumerators each indicate the TCP status. The following table describes the TCP status indicated by each enumerator.

| Enumerators   | TCP status   | Description  |
|---------------|--------------|--|
| _CLOSED       | CLOSED       | The connection is closed.  |
| _LISTEN       | LISTEN       | The server is waiting for a connection request (SYN) with a passive open.  |
| _SYN_SENT     | SYN SENT     | The client sent a connection request (SYN) for an active open, and is waiting for an acknowledgment (SYN + ACK).                               |
| _SYN_RECEIVED | SYN RECEIVED | The server sent an acknowledgment (SYN + ACK) in response to the connection request (SYN), and is waiting for an acknowledgment (ACK).         |
| _ESTABLISHED  | ESTABLISHED  | The connection is established.   |
| _CLOSE_WAIT   | CLOSE WAIT   | The server sent an acknowledgment (ACK) to the connection close request (FIN), and is waiting for the server application to be ready to close. |
| _FIN_WAIT1    | FIN WAIT-1   | The client sent a connection close request (FIN), and is waiting for an acknowledgment (ACK).  |
| _CLOSING      | CLOSING      | The client and the server simultaneously received a connection close request (FIN), and are waiting for an acknowledgment (ACK).               |
| _LAST_ACK     | LAST-ACK     | The server sent a connection close request (FIN), and is waiting for an acknowledgment (ACK).  |
| _FIN_WAIT2    | FIN WAIT-2   | The client is waiting for a connection close request (FIN).  |
| _TIME_WAIT    | TIME WAIT    | The client received an acknowledgment (ACK) to the connection close request (FIN) and is waiting for the server process to be completed.       |

## Related System-defined Variables

| Name                             | Meaning | Data type | Description   |
|----------------------------------|---------|-----------|---|
| _EIP_EtnOnlineSta <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta <sup>*2</sup> |         |           |   |
| _EIP2_EtnOnlineSta <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- Except for NX102 CPU Units, you can execute a maximum of 32 of the following instructions at the same time: `SkTUDPCreate`, `SkTUDPRcv`, `SkTUDPSend`, `SkTTCPAccept`, `SkTTCPCConnect`, `SkTTCPRcv`, `SkTTCPSend`, `SkTGetTCPStatus`, `SkTClose`, `SkTClearBuf`, `SkTSetOption`, `ModbusTCPCmd`, `ModbusTCPRead`, and `ModbusTCPWrite`.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) The value of a member of *Socket* is outside of the valid range.
  - b) The handle specified by *Socket.Handle* does not exist.

## Sample Programming

Refer to *Sample Programming* on page 2-1160 for the `SkTTCPCConnect` instruction.

# SketClose

The SketClose instruction closes the specified TCP or UDP socket for the built-in EtherNet/IP.

| Instruction | Name                       | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|----------------------------|------------|--------------------|--|
| SketClose   | Close<br>TCP/UDP<br>Socket | FB         |                    | SketClose_instance(Execute, Socket, Done, Busy, Error, ErrorID); |

## Variables

|        | Meaning | I/O   | Description | Valid range | Unit | Default |
|--------|---------|-------|-------------|-------------|------|---------|
| Socket | Socket  | Input | Socket      | ---         | ---  | ---     |

|        | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1176 for details on the structure <code>_sSOCKET</code> . |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The SketClose 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.

The value of *Done* changes to TRUE when processing of the instruction is completed normally.

Close processing for the TCP and UDP sockets is completed when 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              | Handle of the connection to close.<br>0: Closes all TCP connections that currently use the socket service. | UDINT            | Depends on data type. | ---  | 0       |
| SrcAdr* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name.                         | STRING           | Depends on data type. | ---  | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name.                         | STRING           | Depends on data type. | ---  | ''      |

\*1. These members are not used for this instruction.

## Related System-defined Variables

| Name                             | Meaning | Data type | Description   |
|----------------------------------|---------|-----------|---|
| _EIP_EtnOnlineSta* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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.
- Except for NX102 CPU Units, 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, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) There is a setting error for the local IP address.
  - b) The value of a member of *Socket* is outside of the valid range.
  - c) The handle specified by *Socket.Handle* does not exist.

### Sample Programming

Refer to *Sample Programming* on page 2-1143 for the SktUDPCreate instruction and *Sample Programming* on page 2-1160 for the SktTCPConnect instruction.

# SktClearBuf

The SktClearBuf instruction clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.

| Instruction | Name                                      | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|---|------------|--------------------|--|
| SktClearBuf | Clear TCP/UDP<br>Socket Receive<br>Buffer | FB         |                    | SktClearBuf_instance(Execute, Socket, Done, Busy, Error, ErrorID); |

## Variables

|        | Meaning | I/O   | Description | Valid range | Unit | Default |
|--------|---------|-------|-------------|-------------|------|---------|
| Socket | Socket  | Input | Socket      | ---         | ---  | ---     |

|        | Boo-<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text<br>strings |      |      |     |    |        |
|--------|--------------|--|------|-------|-------|-------|----------|-------|-------|------|-----|------|----------------------|------|---|------|------|-----|----|--------|
|        | BOOL         | BYTE   | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT                 | REAL | LREAL   | TIME | DATE | TOD | DT | STRING |
| Socket |              | Refer to <i>Function</i> on page 2-1179 for details on the structure <code>_sSOCKET</code> . |      |       |       |       |          |       |       |      |     |      |                      |      |   |      |      |     |    |        |

## Function

The SktClearBuf instruction clears the receive buffer for the socket that is specified with *Socket*. The value of *Done* changes to TRUE when processing of the instruction is completed normally. Clear processing of the receive buffer is completed when 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* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. | ---  | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65535            |      | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. | ---  | ''      |

\*1. These members are not used for this instruction.

## Related System-defined Variables

| Name                             | Meaning | Data type | Description   |
|----------------------------------|---------|-----------|---|
| _EIP_EtnOnlineSta* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.

- Except for NX102 CPU Units, 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, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) The value of a member of *Socket* is outside of the valid range.
  - b) The socket that is specified by *Socket* does not exist.
  - c) The handle specified by *Socket.Handle* does not exist.

## Sample Programming

Refer to *Sample Programming* on page 2-1160 for the SktTCPConnect instruction.

# SktSetOption

The SktSetOption instruction sets the option for TCP socket specified for the built-in EtherNet/IP.

| Instruction  | Name                  | FB/<br>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

|             | 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 | ---         | ---  | ---     |

|             | Boo<br>lean | Bit strings   |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1182 for details on the structure <code>_sSOCKET</code> .                        |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OptionType  |             | Refer to <i>Function</i> on page 2-1182 for the enumerators of the enumerated type <code>_eSKT_OPTION_TYPE</code> . |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| OptionParam | OK<br>*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* <sup>1</sup> | Local address       | Local IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65,535           | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. |      | ''      |
| DstAdr* <sup>1</sup> | Destination address | Destination IP address and port number   | _sSOCKET_ADDRESS | ---                   | ---  | ---     |
| PortNo* <sup>1</sup> | Port number         | Port number  | UINT             | 1 to 65,535           | ---  | 0       |
| IpAdr* <sup>1</sup>  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING           | Depends on data type. |      | ''      |

\*1. 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* <sup>1</sup> : TCP_NODELAY option enabled<br>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* <sup>1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta* <sup>2</sup> |         |           |   |
| _EIP2_EtnOnlineSta* <sup>3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* 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*.
- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- You can use this instruction after the socket handle is opened by the *SkdTCPAccept*, or *SkdTCPConnect* instruction and before data transmission is started by *SkdTCPRcv*, *SkdTCPSend*, or *SkdClearBuf* 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 *SkdClose* instruction, execute the *SkdTCPAccept* and *SkdTCPConnect* instructions again to open the handle, and then execute this instruction to set the socket option.
- Except for NX102 CPU Units, you can execute a maximum of 32 of the following instructions at the same time: *SkdUDPCreate*, *SkdUDPRcv*, *SkdUDPSend*, *SkdTCPAccept*, *SkdTCPConnect*, *SkdTCPRcv*, *SkdTCPSend*, *SkdGetTCPStatus*, *SkdClose*, *SkdClearBuf*, *SkdSetOption*, *ModbusTCPcmd*, *ModbusTCPRead*, and *ModbusTCPWrite*.  
For NX102 CPU Units, a maximum of 64 instructions can be executed.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of a member of *Socket* is outside of the valid range.
  - b) The data type specified for *OptionParam* is not supported by *OptionType*.
  - c) The specified handle socket already started transmission.
  - d) The specified socket type is not supported by the handle type. Such a case includes when the *TCP\_NODELAY* is executed for UDP socket.
  - e) 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    |



| Internal Variables | Variable              | Data type              | Initial value   | Comment         |
|--------------------|-----------------------|------------------------|---|-----------------|
|                    | WkSocket              | _sSOCKET               | (Handle:=0, SrcAdr:=(Port-No:=0, IpAdr:="), DstAdr:=(Port-No:=0, IpAdr:=")) | Socket          |
|                    | SendSocketDat         | ARRAY[0..1999] OF BYTE |   | Send data       |
|                    | Nodelay               | BOOL                   | TRUE  | NoDelay setting |
|                    | SkTCPConnect_instance | SkTCPConnect           |   |                 |
|                    | SkSetOption_instance  | SkSetOption            |   |                 |
|                    | SkTCPSend_instance    | SkTCPSend              |   |                 |
|                    | SkClose_instance      | SkClose                |   |                 |

```
// 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;
    SkTCPConnect_instance(Execute:=FALSE); // Initialize instance.
    SkSetOption_instance( // Initialize instance.
        Execute:=FALSE,
        OptionType:=_TCP_NODELAY,
        OptionParam:= Nodelay);
    SkTCPSend_instance( // Initialize instance.
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]); // Dummy
    SkClose_instance(Execute:=FALSE); // Initialize instance.
END_IF;

IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 : // Request a connection.
        SkTCPConnect_instance(
            Execute:=TRUE,
            SrcTcpPort:=UINT#0, // Local UDP port number: Automatically
            DstAdr:='192.168.250.2', // Remote IP address
            DstTcpPort:=UINT#6000, // Destination TCP port number
            Socket =>WkSocket); // Socket
        IF (SkTCPConnect_instance.Done=TRUE) THEN
            Stage:=INT#2; // Normal end
        ELSIF (SkTCPConnect_instance.Error=TRUE) THEN
            Stage:=INT#10; // Error end
        END_IF;
    END_IF;
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;

```

# ModbusTCPcmd

The ModbusTCPcmd instruction sends general commands using Modbus-TCP protocol.

| Instruction  | Name                            | FB/<br>FUN | Graphic expression   | ST expression   |
|--------------|---------------------------------|------------|--|---|
| ModbusTCPcmd | Send Modbus TCP General Command | FB         | <p>The graphic expression shows a box labeled 'ModbusTCPcmd' with inputs: Execute, Socket, UnitIdentifier, CmdDat, CmdSize, RespDat, and TimeOut. It has outputs: Done, Busy, Error, ErrorID, ErrorIDEx, and RespSize.</p> | ModbusTCPcmd_instance(Execute, Socket, UnitIdentifier, CmdDat, CmdSize, RespDat, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize); |



## Version Information

An NX102 CPU Unit is required to use this instruction.

## Variables

|                | Meaning           | I/O    | Description  | Valid range           | Unit        | Default |
|----------------|-------------------|--------|--|-----------------------|-------------|---------|
| Socket         | Socket            | Input  | Socket   | ---                   | ---         | ---     |
| UnitIdentifier | Unit ID           |        | Unit ID*1  | Depends on data type. | ---         | 255     |
| CmdDat         | Command data      |        | Command data   | Depends on data type. | ---         | ---     |
| CmdSize        | Command data size |        | Command data size  | 1 to 253              | Bytes       | 1       |
| TimeOut        | Timeout time      |        | Specify the timeout time in 0.1 seconds. If 0 is specified, it will be treated as 2 seconds. | Depends on data type. | 0.1 seconds | 20      |
| RespDat        | Response data     | In-out | Response data  | Depends on data type. | ---         | ---     |
| RespSize       | Response size     | Output | Response data size   | 1 to 253              | ---         | ---     |

\*1. When you send commands to Modbus-TCP slaves, the default value is used for operation.

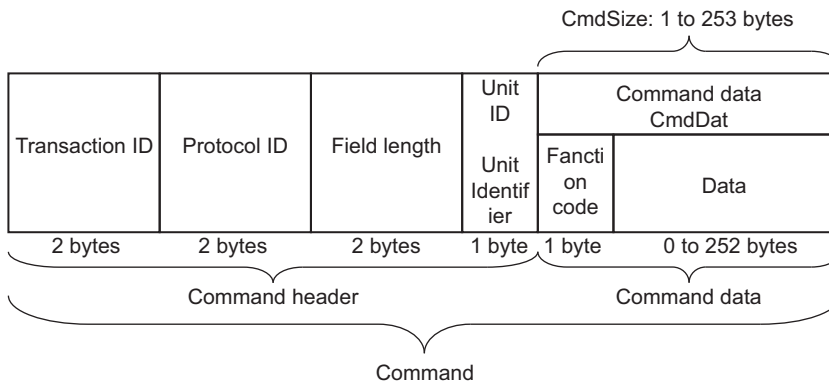
|                     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |  |
|---------------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|---------------------|------|--|------|------|-----|----|--------|--|
|                     | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT                | REAL | LREAL  | TIME | DATE | TOD | DT | STRING |  |
| Socket              |             |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |  |
| UnitIdentifier      |             |             |      |       |       | OK       |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |  |
| CmdDat[]<br>(array) |             | OK          |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |  |
| CmdSize             |             |             |      |       |       |          | OK   |       |       |      |     |      |                     |      |  |      |      |     |    |        |  |

|                      | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|----------------------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                      | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| TimeOut              |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RespDat[]<br>(array) |             | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RespSize             |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |

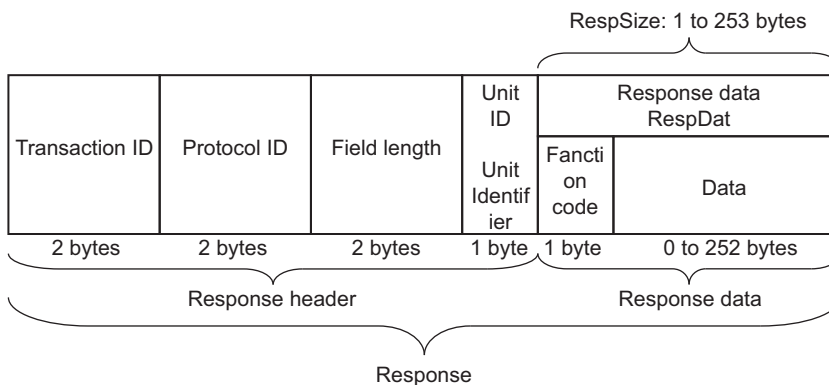
## Function

The ModbusTCPCmd instruction uses the Modbus TCP protocol to send general commands to the destination socket which is established in advance by executing the SktTCPConnect instruction. This instruction ends normally when a normal response to the command is received.

This instruction stores the contents of the CmdDat[] input variable as long as the length of the CmdSize input variable in the command data.



After the instruction sends general commands, it stores the response data received from the destination in the RespDat[] in-out variable.



The instruction outputs, in bytes, the size of the response data contained in the received response to RespSize.

In case of an error response, the values of RespDat[] and RespSize are not changed.

TimeOut input variable specifies the timeout time in 100 milliseconds. If a response does not return within the timeout time, it is determined as a timeout error.

## Data Type of Socket

The data type of *Socket* is structure `_sSOCKET`. The specifications are as follows:

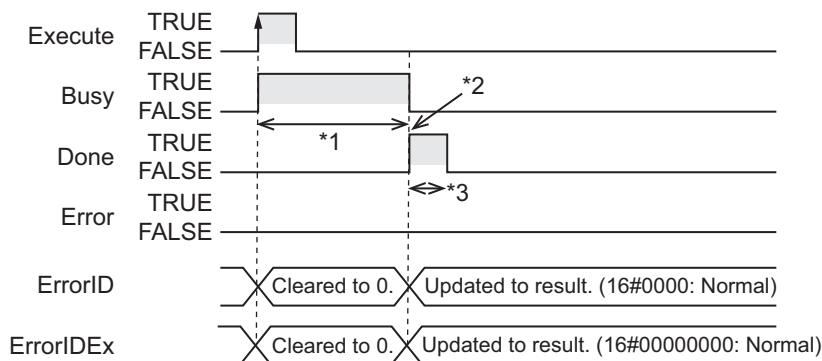
| Name     | Meaning             | Description  | Data type                     | Valid range           | Unit | Default |
|----------|---------------------|--|-------------------------------|-----------------------|------|---------|
| Socket   | Socket              | Socket   | <code>_sSOCKET</code>         | ---                   | ---  | ---     |
| Handle   | Handle              | Handle of the socket to send and receive data                                      | UDINT                         | Depends on data type. | ---  | 0       |
| SrcAdr*1 | Local address       | Local IP address and port number   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo*1 | Port number         | Port number  | UINT                          | 1 to 65,535           |      | 0       |
| IpAdr*1  | IP address          | IP address or host name.<br>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   | <code>_sSOCKET_ADDRESS</code> | ---                   | ---  | ---     |
| PortNo*1 | Port number         | Port number  | UINT                          | 1 to 65,535           |      | 0       |
| IpAdr*1  | IP address          | IP address or host name.<br>A DNS or Hosts setting is required to use a host name. | STRING                        | Depends on data type. | ---  | "       |

\*1. These members are not used for this instruction.

## Timing Charts

The following figures show the timing charts.

### ● Normal end

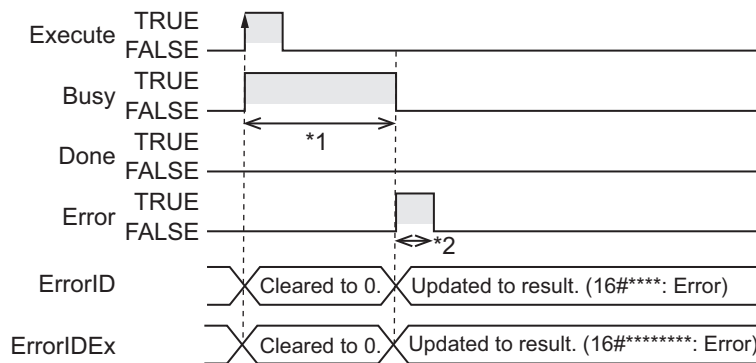


\*1. Processing with Modbus slave

\*2. A response to the command is received.

\*3. Task period

### ● Error end



\*1. Processing with Modbus slave

\*2. Task period

## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* for details on the socket service functions.
- If you execute this instruction in the Simulator, when *Execute* changes from FALSE to TRUE, *Done* immediately changes to TRUE. Data will not be output to the communications line. The value of *RespDat[]* does not change, while the value of *RespSize* changes to 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 normally completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for timing charts for *Execute*, *Done*, *Busy*, and *Error*.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- This instruction can be used only for the built-in EtherNet/IP on NX102 CPU Units.
- For the destination to which to send general commands using this instruction, you need to use the *SkTCPCConnect* instruction to establish a connection in advance. Input the *Socket* that you get there to this instruction. At this time, specify the port number on the Modbus TCP slave side (the port number is set to 502 by default).

- This instruction does not clear the receive buffer for a TCP socket. If you need to clear the buffer, execute the SktClearBuf instruction.
- When you want to set the socket option, execute the SktSetOption instruction.
- You can execute a maximum of 64 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i> | Error  |
|-------------------------|---------------------------|--|
| 16#0400                 | 16#00000000               | The value of <i>CmdSize</i> is outside of the valid range.   |
| 16#0406                 | 16#00000000               | <i>CmdSize</i> exceeded the area of <i>CmdDat[]</i> .  |
| 16#0407                 | 16#00000000               | The number of received bytes exceeded the received data area.  |
| 16#0C10                 | 16#000000XX               | A Exception Response was received by Modbus.<br>XX of 000000xx in <i>ErrorIDEx</i> indicates the Exception Code.<br>Refer to the Modbus Protocol for details on Exception Code.  |
| 16#0C11                 | 16#00000000               | The Modbus Response data is incorrect. <ul style="list-style-type: none"> <li>• <i>FuctionCode</i> is incorrect.</li> <li>• Receive size is incorrect.</li> </ul>  |
| 16#2003                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The socket is being processed.</li> <li>• The socket is closed.</li> </ul>  |
| 16#2006                 | 16#00000000               | No response was received from the destination node within the timeout time.  |
| 16#2007                 | 16#00000000               | The handle value is out of range.  |
| 16#2008                 | 16#00000000               | More than 64 of the following instructions were executed at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite. |

## Sample Programming

The following is a sample program with an NX102 CPU Unit whose IP address is 192.168.250.1. It clears the buffer and then sends a Modbus command to the destination Modbus TCP slave (192.168.250.10, port 502) when *Trigger* changes to TRUE.

It reads a holding register from the read start address 32 (BYTE#16#0020) in the destination. The general command is used to read the variables.

The TCP-NODELAY option is specified in this sample programming because it is recommended in the Modbus Messaging on TCP/IP implementation guide V1.0b.

## ST

| Internal Variables | Variable        | Data type | Initial value | Comment             |
|--------------------|-----------------|-----------|---------------|---------------------|
|                    | Trigger         | BOOL      |               | Execution condition |
|                    | DoModbusTrigger | BOOL      |               | Processing          |
|                    | Nodelay         | BOOL      |               | NoDelay setting     |
|                    | Stage           | INT       |               | State transition    |

| Internal Variables | Variable               | Data type             | Initial value | Comment            |
|--------------------|------------------------|-----------------------|---------------|--------------------|
|                    | Socket                 | _sSOCKET              |               | Socket             |
|                    | ModbusCmdDat           | ARRAY[0..4] OF BYTE   |               | Command data       |
|                    | ModbusDatSize          | UINT                  |               | Command data size  |
|                    | ModbusRespDat          | ARRAY[0..253] OF BYTE |               | Response data      |
|                    | ModbusRespSize         | UINT                  |               | Response data size |
|                    | SkTCPCConnect_instance | SkTCPCConnect         |               |                    |
|                    | SkTSetOption_instance  | SkTSetOption          |               |                    |
|                    | SkClearBuf_instance    | SkClearBuf            |               |                    |
|                    | ModbusTCPCmd_instance  | ModbusTCPCmd          |               |                    |
|                    | SkClose_instance       | SkClose               |               |                    |

| External Variables | Variable           | Data type | Constant | Comment |
|--------------------|--------------------|-----------|----------|---------|
|                    | _EIP1_EtnOnlineSta | BOOL      | ☑        | Online  |

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
  DoModbusTrigger:=TRUE;
  Nodelay:=TRUE;

  SkTCPCConnect_instance(Execute:= FALSE);
  SkTSetOption_instance(
    Execute:=FALSE,
    Socket:=Socket,
    OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
    OptionParam:=Nodelay);

  SkClearBuf_instance(
    Execute:=FALSE,
    Socket:=Socket);
  ModbusTCPCmd_instance(
    Execute:=FALSE,
    Socket:=Socket,
    CmdDat:=ModbusCmdDat[0],
    CmdSize:=ModbusDatSize,
    RespDat:=ModbusRespDat[0]);

  SkClose_instance(
    Execute:=FALSE,
    Socket:=Socket);

  Stage:=1; // Initialization completed.
END_IF;
```



```

IF (DoModbusTrigger=TRUE) THEN
  CASE Stage OF
  1: // Socket connection request
    SktTCPConnect_instance(
      Execute:=TRUE,
      SrcTcpPort:=UINT#502,
      DstAdr:='192.168.250.10',
      DstTcpPort:=UINT#502,
      Socket=>Socket);
    IF (SktTCPConnect_instance.Done=TRUE) THEN
      Stage:=2; // Socket connection is normal end.
    ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
      Stage:=99; // Socket connection is error end.
    END_IF;

  2: //TCP-NODELAY option request
    SktSetOption_instance(
      Execute:=TRUE,
      Socket:=Socket,
      OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
      OptionParam:=Nodelay);
    IF (SktSetOption_instance.Done=TRUE) THEN
      Stage:=3; // Option setting is normal end.
    ELSIF (SktSetOption_instance.Error=TRUE) THEN
      Stage:=99; // Option setting is error end.
    END_IF;

  3: // Buffer clear request
    SktClearBuf_instance(
      Execute:=TRUE,
      Socket:=Socket);
    IF (SktClearBuf_instance.Done=TRUE) THEN
      Stage:=4; // Buffer clear is normal end.
    ELSIF (SktClearBuf_instance.Error=TRUE) THEN
      Stage:=99; // Buffer clear is error end.
    END_IF;

  4: // Modbus Cmd send request
    ModbusCmdDat[0]:=BYTE#16#03; // Function code (read variable)
    ModbusCmdDat[1]:=BYTE#16#00; // Read start address (H)
    ModbusCmdDat[2]:=BYTE#16#20; // Read start address (L)
    ModbusCmdDat[3]:=BYTE#16#00; // Number of data (H)
    ModbusCmdDat[4]:=BYTE#16#01; // Number of data (L)
    ModbusDatSize:=5;
    ModbusTCPCmd_instance(
      Execute:=TRUE,
      Socket:=Socket,

```

```
        CmdDat:=ModbusCmdDat[0],
        CmdSize:=ModbusDatSize,
        RespDat:=ModbusRespDat[0],
        RespSize=>ModbusRespSize);
IF (ModbusTCPCmd_instance.Done=TRUE) THEN
    Stage:=5; // ModbusTCPWrite instruction is normal end.
ELSIF (ModbusTCPCmd_instance.Error=TRUE) THEN
    Stage:=99; // ModbusTCPWrite instruction is error end.
END_IF;

5: // Socket close request
SktClose_instance(
    Execute:=TRUE,
    Socket:=Socket);
IF (SktClose_instance.Done=TRUE) THEN
    Stage:=6; // Socket close is normal end.
ELSIF (SktClose_instance.Error=TRUE) THEN
    Stage:=99; // Socket close is error end.
END_IF;

6: // Processing after the ModbusTCPRead instruction is normal end.
Trigger:=FALSE;
DoModbusTrigger:=FALSE;

99: // Error Processing
Trigger:=FALSE;
DoModbusTrigger:=FALSE;
END_CASE;
END_IF;
```

# ModbusTCPRead

The ModbusTCPRead instruction reads data that is requested by sending read commands using Modbus-TCP protocol.

| Instruction        | Name                               | FB/<br>FUN | Graphic expression | ST expression  |
|--------------------|------------------------------------|------------|--------------------|--|
| ModbusTCP-<br>Read | Send Modbus<br>TCP Read<br>Command | FB         |                    | ModbusTCPRead_instance(Execute, Socket, UnitIdentifier, ReadCmd, ReadDat, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, ReadSize); |



## Version Information

An NX102 CPU Unit is required to use this instruction.

## Variables

|                | Meaning      | I/O    | Description  | Valid range                                       | Unit                                      | Default |
|----------------|--------------|--------|--|---|---|---------|
| Socket         | Socket       | Input  | Socket   | ---   | ---                                       | ---     |
| UnitIdentifier | Unit ID      |        | Unit ID* <sup>1</sup>  | Depends on data type.                             | ---                                       | 255     |
| ReadCmd        | Read command |        | Command data   | Depends on data type.                             | ---                                       | ---     |
| TimeOut        | Timeout time |        | Specify the timeout time in 0.1 seconds. If 0 is specified, it will be treated as 2 seconds. | Depends on data type.                             | 0.1 seconds                               | 20      |
| ReadDat        | Read data    | In-out | Read data  | Depends on data type.                             | ---                                       | ---     |
| ReadSize       | Read size    | Output | Amount of read data  | 1 to 2000* <sup>2</sup><br>1 to 125* <sup>3</sup> | Bits* <sup>2</sup><br>Words* <sup>3</sup> | ---     |

\*1. When you send commands to Modbus-TCP slaves, the default value is used for operation.

\*2. Use this valid range when the data to read is output or input status (BOOL).

\*3. Use this valid range when the data to read is input register or holding register (WORD).

|                | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 <i>Data Type of Socket</i> on page 2-1189 for details on the structure <code>_sSOCKET</code> . |
| UnitIdentifier |             |             |      |       |       | OK       |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |   |

|                      | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|----------------------|---|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                      | BOOL  | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| ReadCmd              | Refer to <i>Data Type of ReadCmd</i> on page 2-1197 for details on the structure <code>_sMODBUS_READ</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| TimeOut              |   |             |      |       |       | OK    |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ReadDat[]<br>(array) | OK  |             | OK   |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ReadSize             |   |             |      |       |       | OK    |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The ModbusTCPRead instruction uses the Modbus TCP protocol to send read commands to the destination socket which is established in advance by executing the SktTCPConnect instruction. This instruction stores the read data and the amount of read data in *ReadDat* and *ReadSize* respectively, and ends normally when a normal response to the command (i.e., requested data) is received.

In case of an error response, the values of *ReadDat* and *ReadSize* do not change.

*TimeOut* input variable specifies the timeout time in 100 milliseconds. If a response does not return within the timeout time, it is determined as a timeout error.

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         |
|--|------------------|
| <code>_MDB_READ_COILS</code>             | 1 to 2,000 (bit) |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | 1 to 2,000 (bit) |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | 1 to 125 (word)  |
| <code>_MDB_READ_INPUT_REGISTERS</code>   | 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      |
|--|----------------|
| <code>_MDB_READ_COILS</code>             | BOOL<br>BOOL[] |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | BOOL<br>BOOL[] |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | WORD<br>WORD[] |
| <code>_MDB_READ_INPUT_REGISTERS</code>   | WORD<br>WORD[] |

## Data Type of Socket

Refer to the *Data Type of Socket* on page 2-1189 for the ModbusTCPcmd instruction.

## Data Type of *ReadCmd*

The data type of *ReadCmd* is structure `_sMODBUS_READ`.

The specifications are as follows:

| Name     | Meaning       | Description        | Data type                  | Valid range  | Unit  | Default                      |
|----------|---------------|--------------------|----------------------------|--|-------|------------------------------|
| ReadCmd  | Read command  | Command data       | <code>_sMODBUS_READ</code> | Depends on data type.  | ---   | ---                          |
| Fun      | Function code | Function code      | <code>_eMDB_FUN</code>     | <code>_MDB_READ_COILS</code><br><code>_MDB_READ_DISCRETE_INPUTS</code><br><code>_MDB_READ_HOLDING_REGISTERS</code><br><code>_MDB_READ_INPUT_REGISTERS</code> | ---   | <code>_MDB_READ_COILS</code> |
| ReadAdr  | Read address  | Read start address | UINT                       | Depends on data type.  | ---   | 0                            |
| ReadSize | Read size     | Read size          | UINT                       | Depends on function code.  | ---*1 | 1                            |

\*1. The unit is the same as the unit of read data specified with *ReadCmd.Fun*.

### ● Data Type of *FUN*

The data type of *Fun* is enumerated type `_eMDB_FUN`.

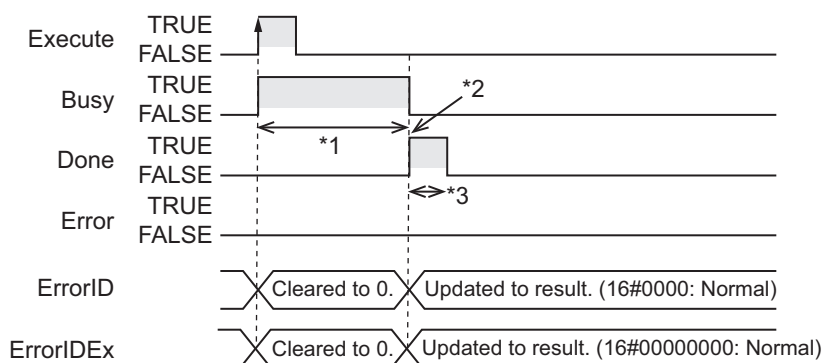
The meanings of the enumerators of enumerated type `_eMDB_FUN` are as follows:

| Enumerator                               | Meaning                       |
|--|-------------------------------|
| <code>_MDB_READ_COILS</code>             | Read outputs (bit)            |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | Read inputs (bit)             |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | Read holding registers (word) |
| <code>_MDB_READ_INPUT_REGISTERS</code>   | Read input registers (word)   |

## Timing Charts

The following figures show the timing charts.

### ● Normal end

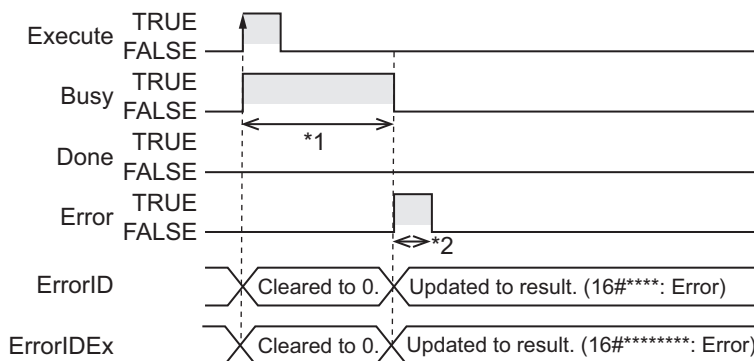


\*1. Processing with Modbus slave

\*2. A response to the command is received.

\*3. Task period

### ● Error end



\*1. Processing with Modbus slave

\*2. Task period

## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* for details on the socket service functions.
- If you execute this instruction in the Simulator, when *Execute* changes from FALSE to TRUE, *Done* immediately changes to TRUE. Data will not be output to the communications line. The value of `ReadDat[]` does not change, while the value of *ReadSize* changes to 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 normally completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for timing charts for *Execute*, *Done*, *Busy*, and *Error*.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- This instruction can be used only for the built-in EtherNet/IP on NX102 CPU Units.
- For the destination to which to send general commands using this instruction, you need to use the `SkTCPCConnect` instruction to establish a connection in advance. Input the *Socket* that you get there

to this instruction. At this time, specify the port number on the Modbus TCP slave side (the port number is set to 502 by default).

- This instruction does not clear the receive buffer for a TCP socket. If you need to clear the buffer, execute the `SktClearBuf` instruction.
- When you want to set the socket option, execute the `SktSetOption` instruction.
- You can execute a maximum of 64 of the following instructions at the same time: `SktUDPCreate`, `SktUDPRcv`, `SktUDPSend`, `SktTCPAccept`, `SktTCPConnect`, `SktTCPRcv`, `SktTCPSend`, `SktGetTCPStatus`, `SktClose`, `SktClearBuf`, `SktSetOption`, `ModbusTCPcmd`, `ModbusTCPRead`, and `ModbusTCPWrite`.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i> | Error   |
|-------------------------|---------------------------|---|
| 16#0400                 | 16#00000000               | The value of <i>ReadCmd.Fun</i> is outside of the valid range.<br>The value of <i>ReadCmd.ReadSize</i> is outside of the valid range.   |
| 16#0406                 | 16#00000000               | <i>ReadCmd.ReadSize</i> exceeded the area of <i>ReadDat[]</i> .   |
| 16#0407                 | 16#00000000               | The number of received bytes exceeded the received data area.   |
| 16#0419                 | 16#00000000               | The data type that is specified with <i>ReadDat[]</i> does not match that of <i>ReadCmd.Fun</i> .   |
| 16#0C10                 | 16#000000XX               | A Exception Response was received by Modbus.<br>XX of 000000xx in <i>ErrorIDEx</i> indicates the Exception Code.<br>Refer to the Modbus Protocol for details on Exception Code.   |
| 16#0C11                 | 16#00000000               | The Modbus Response data is incorrect. <ul style="list-style-type: none"> <li>• <i>FuctionCode</i> is incorrect.</li> <li>• <i>Receive size</i> is incorrect.</li> </ul>  |
| 16#2003                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The socket is being processed.</li> <li>• The socket is closed.</li> </ul>   |
| 16#2006                 | 16#00000000               | No response was received from the destination node within the timeout time.   |
| 16#2007                 | 16#00000000               | The handle value is out of range.   |
| 16#2008                 | 16#00000000               | More than 64 of the following instructions were executed at the same time:<br><code>SktUDPCreate</code> , <code>SktUDPRcv</code> , <code>SktUDPSend</code> , <code>SktTCPAccept</code> , <code>SktTCPConnect</code> , <code>SktTCPRcv</code> , <code>SktTCPSend</code> , <code>SktGetTCPStatus</code> , <code>SktClose</code> , <code>SktClearBuf</code> , <code>SktSetOption</code> , <code>ModbusTCPcmd</code> , <code>ModbusTCPRead</code> , and <code>ModbusTCPWrite</code> . |

## Sample Programming

The following is a sample program with an NX102 CPU Unit whose IP address is 192.168.250.1. It clears the buffer and then sends a Modbus command to the destination Modbus TCP slave (192.168.250.10, port 502) when *Trigger* changes to TRUE.

It reads the status of an output from the read start address 19 in the destination. The read command is used to read the variables.

The TCP-NODELAY option is specified in this sample programming because it is recommended in the Modbus Messaging on TCP/IP implementation guide V1.0b.

## ST

| Internal Variables | Variable               | Data type     | Default | Comment             |
|--------------------|------------------------|---------------|---------|---------------------|
|                    | Trigger                | BOOL          |         | Execution condition |
|                    | DoModbusTrigger        | BOOL          |         | Processing          |
|                    | Nodelay                | BOOL          |         | NoDelay setting     |
|                    | Stage                  | INT           |         | State transition    |
|                    | Socket                 | _sSOCKET      |         | Socket              |
|                    | ModbusReadCmd          | _sMODBUS_READ |         | Read command        |
|                    | ModbusRespDat          | BOOL          |         | Read data           |
|                    | ModbusReadSize         | UINT          |         | Read data size      |
|                    | SkdTCPConnect_instance | SkdTCPConnect |         |                     |
|                    | SkdSetOption_instance  | SkdTSetOption |         |                     |
|                    | SkdClearBuf_instance   | SkdClearBuf   |         |                     |
|                    | ModbusTCPRead_instance | ModbusTCPRead |         |                     |
|                    | SkdClose_instance      | SkdClose      |         |                     |

| External Variables | Variable           | Data type | Constant | Comment |
|--------------------|--------------------|-----------|----------|---------|
|                    | _EIP1_EtnOnlineSta | BOOL      | ☑        | Online  |

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
  DoModbusTrigger:=TRUE;
  Nodelay:=TRUE;

  SkdTCPConnect_instance(Execute:= FALSE);
  SkdSetOption_instance(
    Execute:=FALSE,
    Socket:=Socket,
    OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
    OptionParam:=Nodelay);

  SkdClearBuf_instance(
    Execute:=FALSE,
    Socket:=Socket);
  ModbusTCPRead_instance(
    Execute:=FALSE,
    Socket:=Socket,
    ReadCmd:=ModbusReadCmd,
    ReadDat:=ModbusReadDat);

  SkdClose_instance(
    Execute:=FALSE,
    Socket:=Socket);
```



```

    Stage:=1; // Initialization completed.
END_IF;

IF (DoModbusTrigger=TRUE) THEN
  CASE Stage OF
    1: // Socket connection request
      SktTCPConnect_instance(
        Execute:=TRUE,
        SrcTcPPort:=UINT#502,
        DstAdr:='192.168.250.10',
        DstTcpPort:=UINT#502,
        Socket=>Socket);
      IF (SktTCPConnect_instance.Done=TRUE) THEN
        Stage:=2; // Socket connection is normal end.
      ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
        Stage:=99; // Socket connection is error end.
      END_IF;

    2: //TCP-NODELAY option request
      SktSetOption_instance(
        Execute:=TRUE,
        Socket:=Socket,
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);

      IF (SktSetOption_instance.Done=TRUE) THEN
        Stage:=3; // Option setting is normal end.
      ELSIF (SktSetOption_instance.Error=TRUE) THEN
        Stage:=99; // Option setting is error end.
      END_IF;

    3: // Buffer clear request
      SktClearBuf_instance(
        Execute:=TRUE,
        Socket:=Socket);

      IF (SktClearBuf_instance.Done=TRUE) THEN
        Stage:=4; // Buffer clear is normal end.
      ELSIF (SktClearBuf_instance.Error=TRUE) THEN
        Stage:=99; // Buffer clear is error end.
      END_IF;

    4: // Modbus Read request
      ModbusReadCmd.Fun:=_MDB_READ_COILS; // Function code
      ModbusReadCmd.ReadAdr:=19; // Read address
      ModbusReadCmd.ReadSize:=1; // Read size
  
```

```
ModbusTCPRead_instance(  
    Execute:=TRUE,  
    Socket:=Socket,  
    ReadCmd:=ModbusReadCmd,  
    ReadDat:=ModbusReadDat,  
    ReadSize=>ModbusReadSize);  
  
IF (ModbusTCPRead_instance.Done=TRUE) THEN  
    Stage:=5; // ModbusTCPWrite instruction is normal end.  
ELSIF (ModbusTCPRead_instance.Error=TRUE) THEN  
    Stage:=99; // ModbusTCPWrite instruction is error end.  
END_IF;  
  
5: // Socket close request  
    SktClose_instance(  
        Execute:=TRUE,  
        Socket:=Socket);  
    IF (SktClose_instance.Done=TRUE) THEN  
        Stage:=6; // Socket close is normal end.  
    ELSIF (SktClose_instance.Error=TRUE) THEN  
        Stage:=99; // Socket close is error end.  
    END_IF;  
  
6: // Processing after the ModbusTCPRead instruction is normal end.  
    Trigger:=FALSE;  
    DoModbusTrigger:=FALSE;  
  
99: // Error Processing  
    Trigger:=FALSE;  
    DoModbusTrigger:=FALSE;  
END_CASE;  
END_IF;
```

# ModbusTCPWrite

The ModbusTCPWrite instruction sends write commands using Modbus-TCP protocol.

| Instruction     | Name                          | FB/<br>FUN | Graphic expression | ST expression   |
|-----------------|-------------------------------|------------|--------------------|---|
| ModbusTCP-Write | Send Modbus TCP Write Command | FB         |                    | ModbusTCPWrite_instance(Execute, Socket, UnitIdentifier, WriteCmd, WriteDat, Timeout, Done, Busy, Error, ErrorID, ErrorIDEx); |



## Version Information

An NX102 CPU Unit is required to use this instruction.

## Variables

|                | Meaning       | I/O   | Description  | Valid range           | Unit        | Default |
|----------------|---------------|-------|--|-----------------------|-------------|---------|
| Socket         | Socket        | Input | Socket   | ---                   | ---         | ---     |
| UnitIdentifier | Unit ID       |       | Unit ID*1  | Depends on data type. | ---         | 255     |
| WriteCmd       | Write command |       | Command data   | Depends on data type. | ---         | ---     |
| WriteDat       | Write data    |       | Write data   | Depends on data type. | ---         | ---     |
| Timeout        | Timeout time  |       | Specify the timeout time in 0.1 seconds. If 0 is specified, it will be treated as 2 seconds. | Depends on data type. | 0.1 seconds | 20      |

\*1. When you send commands to Modbus-TCP slaves, the default value is used for operation.

|                       | Boo<br>lean | Bit strings   |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 <i>Data Type of Socket</i> on page 2-1189 for details on the structure <code>_sSOCKET</code> .         |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| UnitIdentifier        |             |   |      |       |       | OK       |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| WriteCmd              |             | Refer to <i>Data Type of WriteCmd</i> on page 2-1204 for details on the structure <code>_sMODBUS_WRITE</code> . |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| WriteDat[]<br>(array) | OK          |   | OK   |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| Timeout               |             |   |      |       |       |          | OK   |       |       |      |     |      |                     |      |  |      |      |     |    |        |

## Function

The ModbusTCPWrite instruction uses the Modbus TCP protocol to send write commands to the destination socket which is established in advance by executing the SktTCPConnect instruction. This instruction ends normally when a normal response to the command (i.e., requested data) is received.

*TimeOut* input variable specifies the timeout time in 100 milliseconds.

If a response does not return within the timeout time, it is determined as a timeout error.

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<br>BOOL[] |
| _MDB_WRITE_SINGLE_REGISTER    | WORD<br>WORD[] |
| _MDB_WRITE_MULTIPLE_COILS     | BOOL<br>BOOL[] |
| _MDB_WRITE_MULTIPLE_REGISTERS | WORD<br>WORD[] |

## Data Type of Socket

Refer to the *Data Type of Socket* on page 2-1189 for the ModbusTCPCmd instruction.

## Data Type of WriteCmd

The data type of *WriteCmd* is structure `_sMODBUS_WRITE`.

The specifications are as follows:

| Name     | Meaning       | Description  | Data type      | Valid range           | Unit | Default |
|----------|---------------|--------------|----------------|-----------------------|------|---------|
| WriteCmd | Write command | Command data | _sMODBUS_WRITE | Depends on data type. | ---  | ---     |

| Name      | Meaning       | Description         | Data type | Valid range  | Unit  | Default                |
|-----------|---------------|---------------------|-----------|--|-------|------------------------|
| Fun       | Function code | Function code       | _eMDB_FUN | _MDB_WRITE_SINGLE_COIL<br>_MDB_WRITE_SINGLE_REGISTER<br>_MDB_WRITE_MULTIPLE_COILS<br>_MDB_WRITE_MULTIPLE_REGISTERS | ---   | _MDB_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.  | ---*1 | 1                      |

\*1. The unit is the same as the unit of read data specified with *WriteCmd.Fun*.

### ● Data Type of FUN

The data type of *Fun* is enumerated type `_eMDB_FUN`.

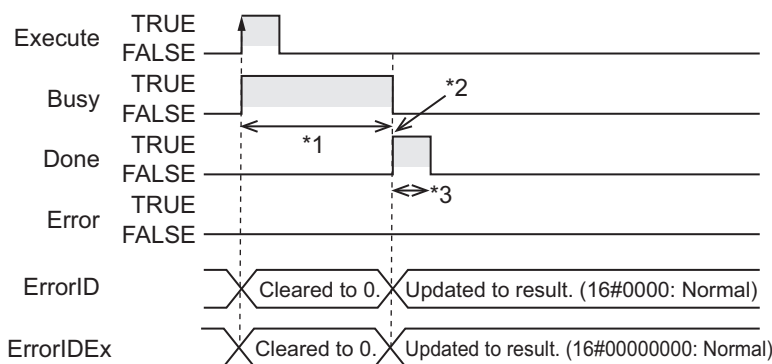
The meanings of the enumerators of enumerated type `_eMDB_FUN` are as follows:

| Enumerator                                 | Meaning                                 |
|--|---|
| <code>_MDB_WRITE_SINGLE_COIL</code>        | Write an output (bit)                   |
| <code>_MDB_WRITE_SINGLE_REGISTER</code>    | Write a holding register (word)         |
| <code>_MDB_WRITE_MULTIPLE_COILS</code>     | Write multiple outputs (bit)            |
| <code>_MDB_WRITE_MULTIPLE_REGISTERS</code> | Write multiple holding registers (word) |

## Timing Charts

The following figures show the timing charts.

### ● Normal end

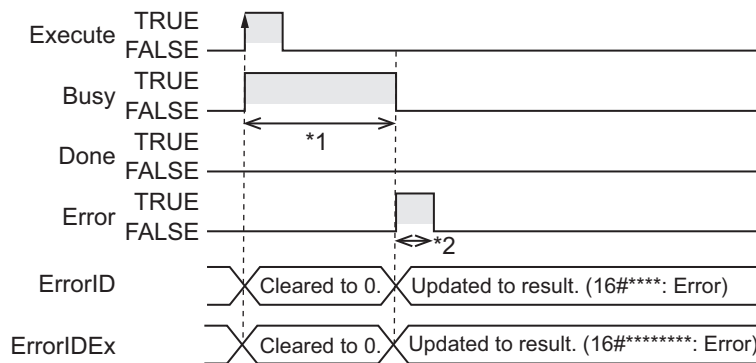


\*1. Processing with Modbus slave

\*2. A response to the command is received.

\*3. Task period

## ● Error end



\*1. Processing with Modbus slave

\*2. Task period

## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual (Cat. No. W506)* for details on the socket service functions.
- If you execute this instruction in the Simulator, when *Execute* changes from FALSE to TRUE, *Done* immediately changes to TRUE. Data will not be output to the communications line.

## 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 normally completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for timing charts for *Execute*, *Done*, *Busy*, and *Error*.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- This instruction can be used only for the built-in EtherNet/IP on NX102 CPU Units.
- For the destination to which to send general commands using this instruction, you need to use the `SktTCPConnect` instruction to establish a connection in advance. Input the *Socket* that you get there to this instruction. At this time, specify the port number on the Modbus TCP slave side (the port number is set to 502 by default).
- This instruction does not clear the receive buffer for a TCP socket. If you need to clear the buffer, execute the `SktClearBuf` instruction.

- When you want to set the socket option, execute the SktSetOption instruction.
- You can execute a maximum of 64 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

| Value of <i>ErrorID</i> | Value of <i>ErrorIDEx</i> | Error   |
|-------------------------|---------------------------|---|
| 16#0400                 | 16#00000000               | The value of <i>WriteCmd.Fun</i> is outside of the valid range.<br>The value of <i>WriteCmd.WriteSize</i> is outside of the valid range.  |
| 16#0406                 | 16#00000000               | <i>WriteCmd.WriteSize</i> exceeded the area of <i>WriteDat[]</i> .  |
| 16#0419                 | 16#00000000               | The data type that is specified with <i>WriteDat[]</i> does not match that of <i>WriteCmd.Fun</i> .   |
| 16#0C10                 | 16#000000XX               | A Exception Response was received by Modbus.<br>XX of 000000xx in <i>ErrorIDEx</i> indicates the Exception Code.<br>Refer to the Modbus Protocol for details on Exception Code.   |
| 16#0C11                 | 16#00000000               | The Modbus Response data is incorrect. <ul style="list-style-type: none"> <li>• FuctionCode is incorrect.</li> <li>• Receive size is incorrect.</li> </ul>  |
| 16#2003                 | 16#00000000               | <ul style="list-style-type: none"> <li>• The socket is being processed.</li> <li>• The socket is closed.</li> </ul>   |
| 16#2006                 | 16#00000000               | No response was received from the destination node within the timeout time.   |
| 16#2007                 | 16#00000000               | The handle value is out of range.   |
| 16#2008                 | 16#00000000               | More than 64 of the following instructions were executed at the same time:<br>SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, SktSetOption, ModbusTCPcmd, ModbusTCPRead, and ModbusTCPWrite. |

## Sample Programming

The following is a sample program with an NX102 CPU Unit whose IP address is 192.168.250.1. It clears the buffer and then sends a Modbus command to the destination Modbus TCP slave (192.168.250.10, port 502) when *Trigger* changes to TRUE.

An output from the write start address 149 in the destination slave is turned ON. The write command is used to write the variables.

The TCP-NODELAY option is specified in this sample programming because it is recommended in the Modbus Messaging on TCP/IP implementation guide V1.0b.

### ST

| Internal Variables | Variable        | Data type | Default | Comment             |
|--------------------|-----------------|-----------|---------|---------------------|
|                    | Trigger         | BOOL      |         | Execution condition |
|                    | DoModbusTrigger | BOOL      |         | Processing          |
|                    | Nodelay         | BOOL      |         | NoDelay setting     |
|                    | Stage           | INT       |         | State transition    |
|                    | Socket          | _sSOCKET  |         | Socket              |

| Internal Variables | Variable                | Data type      | Default | Comment       |
|--------------------|-------------------------|----------------|---------|---------------|
|                    | ModbusWriteCmd          | _sMODBUS_WRITE |         | Write command |
|                    | ModbusWriteDat          | BOOL           |         | Write data    |
|                    | SkTCPCConnect_instance  | SkTCPCConnect  |         |               |
|                    | SkTSetOption_instance   | SkTSetOption   |         |               |
|                    | SkClearBuf_instance     | SkClearBuf     |         |               |
|                    | ModbusTCPWrite_instance | ModbusTCPWrite |         |               |
|                    | SkClose_instance        | SkClose        |         |               |

| External Variables | Variable           | Data type | Constant | Comment |
|--------------------|--------------------|-----------|----------|---------|
|                    | _EIP1_EtnOnlineSta | BOOL      | ☑        | Online  |

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
  DoModbusTrigger:=TRUE;
  Nodelay:=TRUE;

  SkTCPCConnect_instance(Execute:= FALSE);
  SkTSetOption_instance(
    Execute:=FALSE,
    Socket:=Socket,
    OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
    OptionParam:=Nodelay);

  SkClearBuf_instance(
    Execute:=FALSE,
    Socket:=Socket);
  ModbusTCPWrite_instance(
    Execute:=FALSE,
    Socket:=Socket,
    WriteCmd:=ModbusWriteCmd,
    WriteDat:=ModbusWriteDat);

  SkClose_instance(
    Execute:=FALSE,
    Socket:=Socket);

  Stage:=1; // Initialization completed.
END_IF;

IF (DoModbusTrigger=TRUE) THEN
  CASE Stage OF
  1: // Socket connection request
    SkTCPCConnect_instance(
```



```

        Execute:=TRUE,
        SrcTcpPort:=UINT#502,
        DstAdr:='192.168.250.10',
        DstTcpPort:=UINT#502,
        Socket=>Socket);
IF (SktTCPConnect_instance.Done=TRUE) THEN
    Stage:=2; // Socket connection is normal end.
ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
    Stage:=99; // Socket connection is error end.
END_IF;

2: // TCP-NODELAY option request
SktSetOption_instance(
    Execute:=TRUE,
    Socket:=Socket,
    OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
    OptionParam:=Nodelay);
IF (SktSetOption_instance.Done=TRUE) THEN
    Stage:=3; // Option setting is normal end.
ELSIF (SktSetOption_instance.Error=TRUE) THEN
    Stage:=99; // Option setting is error end.
END_IF;

3: // Buffer clear request
SktClearBuf_instance(
    Execute:=TRUE,
    Socket:=Socket);
IF (SktClearBuf_instance.Done=TRUE) THEN
    Stage:=4; // Buffer clear is normal end.
ELSIF (SktClearBuf_instance.Error=TRUE) THEN
    Stage:=99; // Buffer clear is error end.
END_IF;

4: // Modbus Write request
ModbusWriteCmd.Fun:=_MDB_WRITE_SINGLE_COIL; // Function code
ModbusWriteCmd.WriteAdr:=149; // Write address
ModbusWriteCmd.WriteSize:=1; // Write size

ModbusTCPWrite_instance(
    Execute:=TRUE,
    Socket:=Socket,
    WriteCmd:=ModbusWriteCmd,
    WriteDat:=ModbusWriteDat);

IF (ModbusTCPWrite_instance.Done=TRUE) THEN
    Stage:=5; // ModbusTCPWrite instruction is normal end.
ELSIF (ModbusTCPWrite_instance.Error=TRUE) THEN

```

```
        Stage:=99; // ModbusTCPWrite instruction is error end.
    END_IF;

5: // Socket close request
    SktClose_instance(
        Execute:=TRUE,
        Socket:=Socket);
    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=6; // Socket close is normal end.
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=99; // Socket close is error end.
    END_IF;

6: // Processing after the ModbusTCPWrite instruction is normal end.
    Trigger:=FALSE;
    DoModbusTrigger:=FALSE;

99: // Error Processing
    Trigger:=FALSE;
    DoModbusTrigger:=FALSE;
END_CASE;
END_IF;
```

# ChangelPAdr

The ChangelPAdr 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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------------------|------------|--------------------|---|
| ChangelPAdr | Change IP Address | FB         |                    | ChangelPAdr_instance(Execute, UnitNo, BootPControl, IPAdr, SubnetMask, DefaultGateway, Done, Busy, Error, ErrorID); |



## 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.

## Variables

|                             | Meaning   | I/O   | Description  | Valid range  | Unit | Default       |
|-----------------------------|---|-------|--|--|------|---------------|
| UnitNo                      | Unit number                                     | Input | Unit number for which to change the IP address         | $\_CBU\_CPU$ *1,<br>$\_CBU\_CPU\_Po$<br>rt1 *2,<br>$\_CBU\_CPU\_Po$<br>rt2 *3,<br>$\_CBU\_No00$ to<br>$\_CBU\_No15$ *4 | ---  | $\_CBU\_No00$ |
| BootPControl,               | IP address assignment method and setting timing |       | Method to obtain the IP address and the setting timing | 0 to 3 *5  |      | 0             |
| IPAdr[] (array)*6           | IP address                                      |       | IP address   | *7   |      | ---           |
| Subnet Mask[] (array)*6     | Subnet mask                                     |       | Subnet mask  |  |      |               |
| Default Gateway[] (array)*6 | 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. 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 NJ-series CPU Unit.

\*5. The range is 0 to 2 for port 1 on an NX-series CPU Unit and for an NJ-series CPU Unit. The range is 0 to 3 for port 2 on an NX-series CPU Unit.

\*6. This is a 4-element array with element numbers 0 to 3.

- \*7. 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* on page 2-1212, below, for details.

|                                 | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1212 for the enumerators of the enumerated type <code>_eUnitNo</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| BootPCon-<br>trol,              |  |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| IPAdr[] (ar-<br>ray)            |  | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
|                                 | Specify an array.  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Subnet<br>Mask[] (ar-<br>ray)   |  | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
|                                 | Specify an array.  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Default<br>Gateway[]<br>(array) |  | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
|                                 | Specify an array.  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

### Function

The `ChangeIPAdr` instruction changes the IP address of the built-in EtherNet/IP 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 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, 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                                      |
|---|--|
| <code>_CBU_CPU</code> *1                            | Built-in EtherNet/IP port                    |
| <code>_CBU_CPU_Port1</code> *2                      | Built-in EtherNet/IP communications port 1   |
| <code>_CBU_CPU_Port2</code> *3                      | Built-in EtherNet/IP communications port 2   |
| <code>_CBU_No00</code> to <code>_CBU_No15</code> *4 | 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. 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 NJ-series CPU Unit.

The value of *BootPCControl* determines how to obtain the new IP address and when to set it, as described in the following table.

For *BootPCControl*, you can specify a value in the range of 0 to 2 for port 1 on an NX-series CPU Unit, and for an NJ-series CPU Unit.

The range is 0 to 3 for port 2 on an NX-series CPU Unit.

| Value of <i>BootPCControl</i> | 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 *BootPCControl* is set to 0.

| Setting of <i>UnitNo</i>  | Input variable           | Valid range  |
|---------------------------|--------------------------|--|
| Built-in EtherNet/IP port | IPAdr[] (array)          | The following IP addresses cannot be used. All other IP addresses are valid. <ul style="list-style-type: none"> <li>IP addresses that start with 127, 0, or 255</li> <li>IP addresses with a host ID for which all bits are 0's or for which all bits are 1's</li> <li>Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> <li>IP addresses that are reserved for AutoIP*<sup>1</sup> (169.254.0.0 to 169.254.255.255)</li> <li>IP addresses of USB ports (192.168.255.0 to 192.168.255.255)*<sup>2</sup></li> </ul> |
|                           | 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. <ul style="list-style-type: none"> <li>IP addresses that start with 127, 0, or 255</li> <li>There is only one address for which all bits are 1's</li> <li>Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> <li>IP addresses that are reserved for AutoIP*<sup>1</sup> (169.254.0.0 to 169.254.255.255)</li> <li>IP addresses of USB ports (192.168.255.0 to 192.168.255.255)*<sup>2</sup></li> </ul>                                 |

| 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. <ul style="list-style-type: none"> <li>• IP addresses that start with 127</li> <li>• Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>• Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> </ul> |
|                   | Subnet-Mask[] (array)    | <ul style="list-style-type: none"> <li>• 0.0.0.0</li> <li>• 192.0.0.0 to 255.255.255.252</li> </ul>   |
|                   | DefaultGateway[] (array) | The following IP addresses cannot be used. All other IP addresses are valid. <ul style="list-style-type: none"> <li>• IP addresses that start with 127</li> <li>• Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>• Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> </ul> |

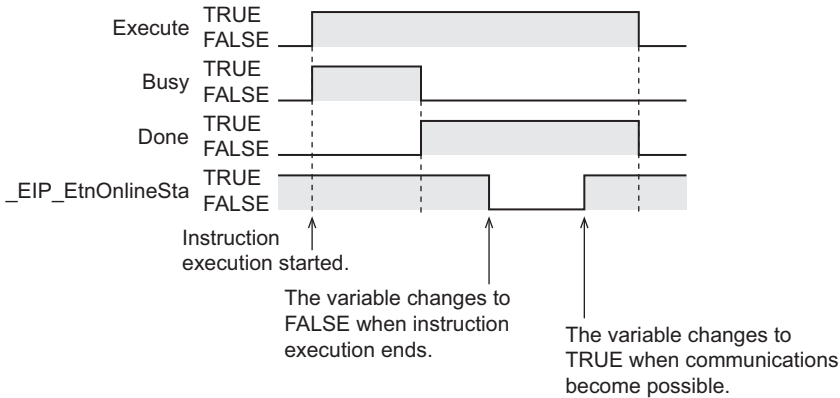
- \*1. AutoIP is an automatic IP address assignment feature of Windows 98 and later operating systems.
- \*2. NX102 CPU Units and NX1P2 CPU Units have no USB port.

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.

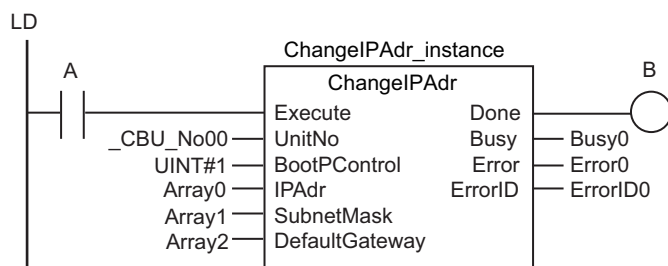
If you specify the built-in EtherNet/IP port for *UnitNo*, you can use the *\_EIP\_EtnOnlineSta*, *\_EIP1\_EtnOnlineSta*, and *\_EIP2\_EtnOnlineSta* system-defined variables to see if communications are possible.

Here, *\_EIP\_EtnOnlineSta* is used as an example, but this information also applies to *\_EIP1\_EtnOnlineSta* and *\_EIP2\_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.

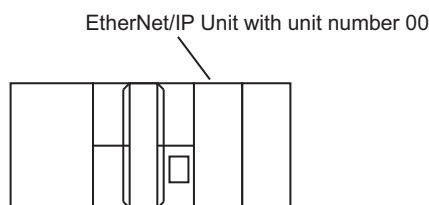


ST

```
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.



IP address is changed to the value that was obtained from the BOOTP server.

## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## 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
- 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 <i>ErrorID</i> | 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. <ul style="list-style-type: none"> <li>• While changing the settings was already in progress</li> <li>• While restarting the built-in EtherNet/IP port was in progress</li> <li>• While downloading tag data link settings from the Network Configurator was in progress</li> </ul> |
| 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.   |

## 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        |

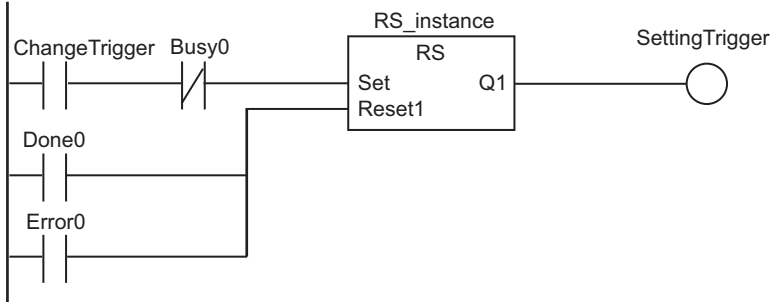
## LD

| 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 |
| 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                  |               |                                      |

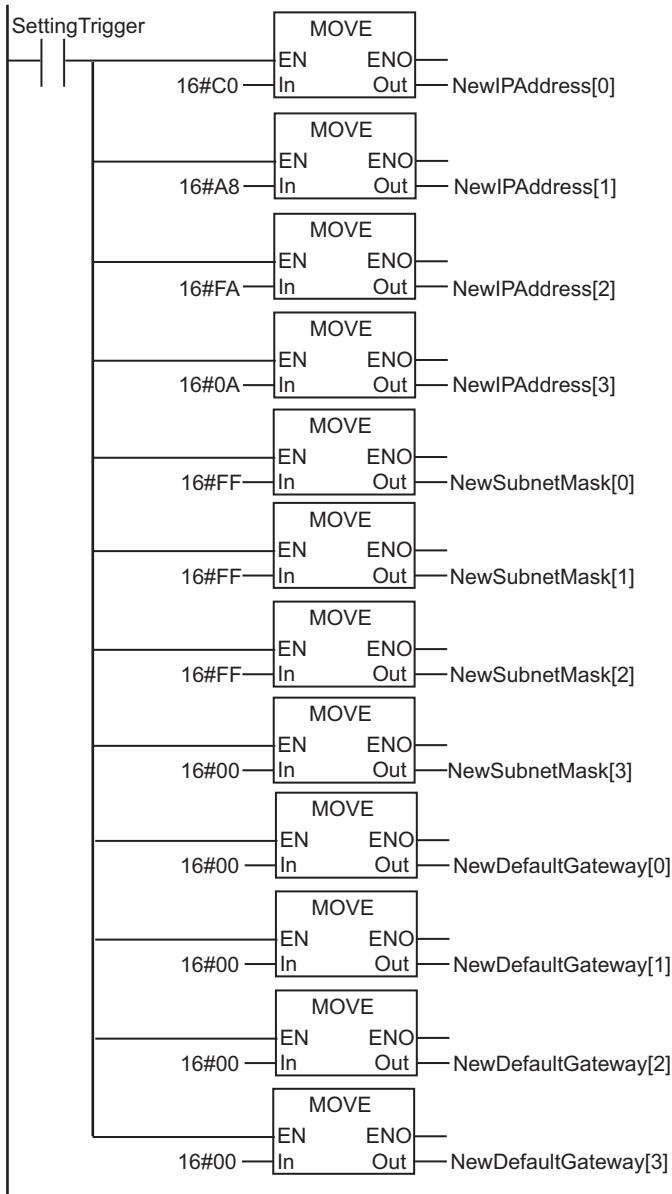


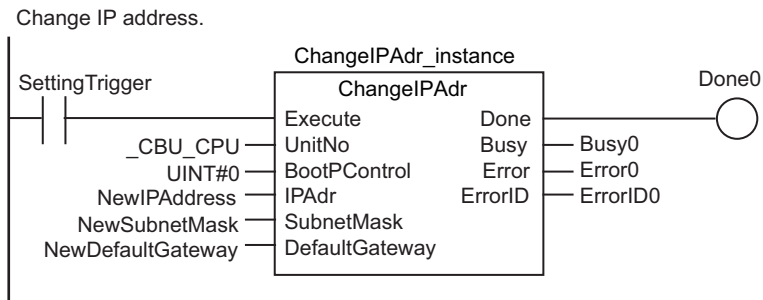
| Variable             | Data type   | Initial value | Comment |
|----------------------|-------------|---------------|---------|
| ChangeIPAdr_instance | ChangeIPAdr |               |         |

Check execution conditions.



Set IP address.





### 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                  |               |                                      |
| ChangelPAdr_instance | ChangelPAdr         |               |                                      |

```

//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/<br>FUN | Graphic expression | ST expression  |
|-------------------|--------------------|------------|--------------------|--|
| Change-FTPAccount | Change FTP Account | FB         |                    | ChangeFTPAccount_instance( Execute, UnitNo, NewUserName, NewPassword, Done, Busy, Error, ErrorID); |



## 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.

## Variables

|             | 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 |
| NewUserName | Login name  |       | Login name  | 1 to 12 single-byte alphanumeric characters (case sensitive) |      | ---       |
| NewPassword | 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 *Valid Range for NewPassword* on page 2-1221, below, for details.

|             | Boo lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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      |          |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |    |
| NewUserName |          |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        | OK |
| NewPassword |          |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        | 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                                      |
|---|--|
| <code>_CBU_CPU</code>                               | Built-in EtherNet/IP port                    |
| <code>_CBU_No00</code> to <code>_CBU_No15</code> *1 | Unit number 00 to 15 of the EtherNet/IP Unit |

\*1. This can be set only for an NJ-series CPU Unit.

## Valid Range for *NewPassword*

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.

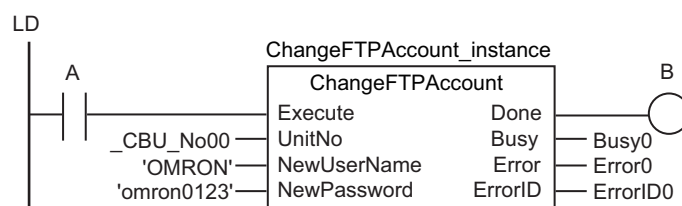
| Setting of <i>UnitNo</i>  | Valid range  |
|---------------------------|--|
| Built-in EtherNet/IP port | 8 to 32 single-byte alphanumeric characters (case sensitive) |
| EtherNet/IP Unit*1        | 1 to 8 single-byte alphanumeric characters (case sensitive)  |

\*1. This can be set only for an NJ-series CPU Unit.

## Notation Example

The following example shows how to change the FTP login name and password of the EtherNet/IP Unit with unit number 00.

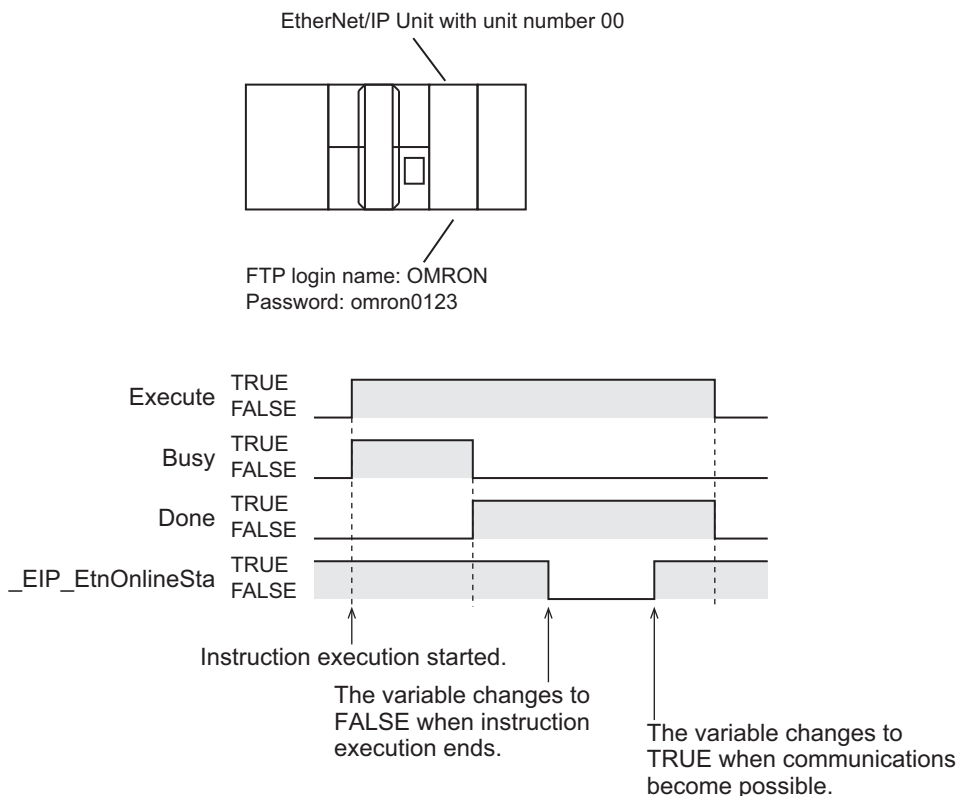
If A (*Execute*) is changed to TRUE from an HMI or any other device, the FTP login name is changed to 'OMRON' and the password is changed to 'omron0123'.



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   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

### 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.
- If you make changes in the FTP login settings with this instruction while you are logged in with FTP, already established FTP sessions will be continuously maintained even after the changes.

## 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 <i>ErrorID</i> | Error name                                   | Description  |
|-------------------------|--|--|
| 16#0400                 | Input Value Out of Range                     | <ul style="list-style-type: none"> <li>• The value of an input variable is outside of the valid range.</li> <li>• The value of an input variable is incorrect.</li> </ul>  |
| 16#2400                 | No Execution Right                           | <p>The instruction was executed when changing the status was not possible.</p> <ul style="list-style-type: none"> <li>• While changing the settings was already in progress</li> <li>• While restarting the built-in EtherNet/IP port was in progress</li> <li>• While downloading tag data link settings from the Network Configurator was in progress</li> </ul> |
| 16#2402                 | Too Many Simultaneous Instruction Executions | Too many ChangeIPAdr, ChangeFTPAccount, and ChangeNTPServerAdr instructions were executed at the same time.  |

# ChangeNTPServerAdr

The ChangeNTPServerAdr instruction changes the NTP server address of the built-in EtherNet/IP port on a CPU Unit or the NTP server address of an EtherNet/IP Unit.

| Instruction         | Name                      | FB/<br>FUN | Graphic expression | ST expression   |
|---------------------|---------------------------|------------|--------------------|---|
| Change-NTPServerAdr | Change NTP Server Address | FB         |                    | ChangeNTPServerAdr_instance(Execute, UnitNo, AdrType, IPAdr, HostName, Done, Busy, Error, ErrorID); |



## 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.

## Variables

|                   | Meaning               | I/O   | Description   | Valid range   | Unit | Default   |
|-------------------|-----------------------|-------|---|---|------|-----------|
| UnitNo            | Unit number           | Input | Unit number for which to change the NTP server address                    | _CBU_CPU or _CBU_No00 to _CBU_No15 *1                                       | ---  | _CBU_No00 |
| AdrType           | Server setting method |       | NTP Server Address Setting Method<br>TRUE: Host name<br>FALSE: IP address | Depends on data type.   |      | FALSE     |
| IPAdr[] (array)*2 | IP address            |       | IP address of the NTP server  | *3  |      | ---       |
| HostName          | Host name             |       | Host name of the address of the NTP server                                | 1 to 200 single-byte alphanumeric characters, "-" (hyphen), and "." (dot)*4 |      | ---       |

\*1. You can set *\_CBU\_No00* to *\_CBU\_No15* only for an NJ-series CPU Unit.

\*2. This is a 4-element array with element numbers 0 to 3.

\*3. 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 *Valid Range for IPAdr[]* on page 2-1225, below, for details.

\*4. There can be between 1 and 63 single-byte alphanumeric characters between "." (dot) and "." (dot). The valid range of *HostName* is valid only when server setting method *AdrType* is FALSE.



|                      | Boo<br>lean  | Bit strings       |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1225 for the enumerators of the enumerated type <code>_eUnitNo</code> . |                   |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| AdrType              | OK   |                   |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| IPAdr[] (ar-<br>ray) | OK   | Specify an array. |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| HostName             |  |                   |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |

## Function

The ChangeNTPServerAdr instruction changes the address of the NTP server of the built-in EtherNet/IP port or EtherNet/IP Unit that is specified with unit number *UnitNo*.

If server setting method *AdrType* is TRUE, the NTP server address is changed to IP address IPAdr[].

If server setting method *AdrType* is FALSE, the NTP server address is changed to host name *HostName*.

When *Execute* changes from FALSE to TRUE, the values of IPAdr[] or *HostName* are written as the NTP server address. 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 NTP server address is enabled when restarting the Unit ends.

You can use this instruction to change the NTP server address 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                                      |
|---|--|
| <code>_CBU_CPU</code>                               | Built-in EtherNet/IP port                    |
| <code>_CBU_No00</code> to <code>_CBU_No15</code> *1 | Unit number 00 to 15 of the EtherNet/IP Unit |

\*1. This can be set only for an NJ-series CPU Unit.

Set the IP address in order in elements 0 to 3 of IPAdr[]. For example, to change the NTP server address to IP address 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.

## Valid Range for IPAdr[]

The valid range of the value of IPAdr[] depends on whether you specify the built-in EtherNet/IP port or an EtherNet/IP Unit for unit number *UnitNo*, as given below. The valid ranges of the values are valid only when *AdrType* is TRUE.

| Setting of <i>UnitNo</i>  | Valid range   |
|---------------------------|---|
| Built-in EtherNet/IP port | The following IP addresses cannot be used. All other IP addresses are valid. <ul style="list-style-type: none"> <li>• IP addresses that start with 127, 0, or 255</li> <li>• Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>• Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> <li>• IP addresses that are reserved for AutoIP*<sup>1</sup> (169.254.0.0 to 169.254.255.255)</li> <li>• IP addresses of USB ports (192.168.255.0 to 192.168.255.255)</li> </ul> |
| EtherNet/IP Unit          | The following IP addresses cannot be used. All other IP addresses are valid. <ul style="list-style-type: none"> <li>• IP addresses that start with 127</li> <li>• Class D IP addresses (224.0.0.0 to 239.255.255.255)</li> <li>• Class E IP addresses (240.0.0.0 to 255.255.255.255)</li> </ul>   |

\*1. AutoIP is an automatic IP address assignment feature of Windows 98 and later operating systems.

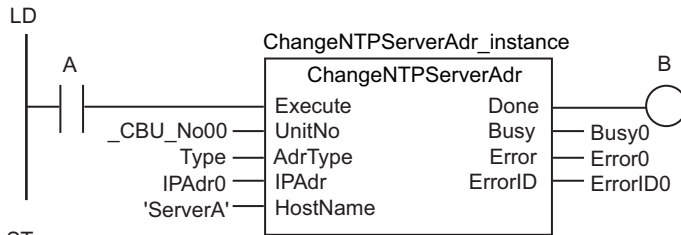
## Notation Example

The following example shows how to change the NTP server address of the EtherNet/IP Unit with unit number 00.

If A (*Execute*) is changed to TRUE from an HMI or other device, the NTP server address is changed. For example, assume that IPAdr0[0] is BYTE#16#C0, IPAdr0[1] is BYTE#16#A8, IPAdr0[2] is BYTE#16#FA, and IPAdr0[3] is BYTE#16#0A.

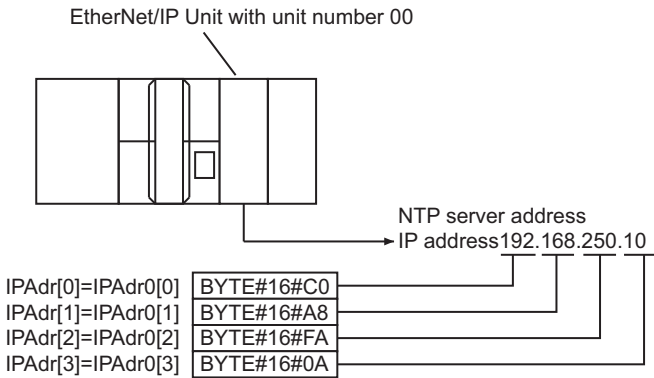
If Type (*AdrType*) is TRUE, the NTP server address is changed to IP address 192.168.250.10.

If Type (*AdrType*) is FALSE, the NTP server address is changed to host name ServerA.



```
LD
A
ChangeNTPServerAdr_instance
ChangeNTPServerAdr
Execute
UnitNo
Type
IPAdr0
'ServerA'
Done
Busy0
Error0
ErrorID0
ST
ChangeNTPServerAdr_instance(A,_CBU_No00,Type,IPAdr0,'ServerA',B,Busy0,Error0,ErrorID0);
```

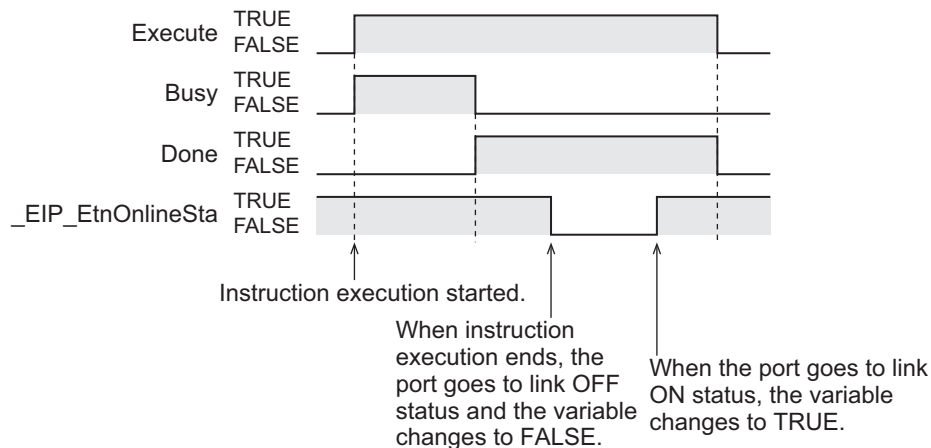
The NTP server address of the EtherNet/IP Unit with a unit number of 00 is changed to an IP address of 192.168.250.10.



You can use the *\_EIP\_EtnOnlineSta*, *\_EIP1\_EtnOnlineSta*, or *\_EIP2\_EtnOnlineSta* system-defined variable to see if the port is in link ON status.

Here, `_EIP_EtnOnlineSta` is used as an example, but this information also applies to `_EIP1_EtnOnlineSta` and `_EIP2_EtnOnlineSta`.

When `Busy` changes to FALSE, the port changes to link OFF status, and `_EIP_EtnOnlineSta` changes to FALSE. When the port then changes to link ON status, `_EIP_EtnOnlineSta` changes to TRUE.



## Related System-defined Variables

| Name                               | Meaning | Data type | Description   |
|------------------------------------|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> *1  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> *2 |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> *3 |         |           |   |

\*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.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Additional Information

- You can change the NTP server address with this instruction even if the CPU Unit is write protected.
- If the NTP server whose address is to change is set to operate at a specified time interval, measuring the specified time interval will start when execution of this instruction ends.

## 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 <code>ErrorID</code> | Error name               | Description   |
|-------------------------------|--------------------------|---|
| 16#0400                       | Input Value Out of Range | The value of <code>IPAdr[]</code> or <code>HostName</code> is not correct*1 |

| Value of ErrorID | Error name                                   | Description  |
|------------------|--|--|
| 16#2400          | No Execution Right                           | <p>The instruction was executed when changing the status was not possible.</p> <ul style="list-style-type: none"> <li>• While changing the settings was already in progress</li> <li>• While restarting the built-in EtherNet/IP port was in progress</li> <li>• While downloading tag data link settings from the Network Configurator was in progress</li> </ul> |
| 16#2402          | Too Many Simultaneous Instruction Executions | <p>Too many ChangeIPAdr, ChangeFTPAccount, and ChangeNTPServerAdr instructions were executed at the same time.</p>   |

\*1. The setting range is checked only for the specified *AdrType*.

# FTPGetFileList

The FTPGetFileList instruction gets a list of the files in the FTP server.

| Instruction    | Name                     | FB/<br>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); |



## 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.

## Variables

|                        | Meaning                       | I/O    | Description   | Valid range  | Unit | Default   |
|------------------------|-------------------------------|--------|---|--|------|-----------|
| ConnectSvr             | Connected FTP server settings | Input  | Setting parameters for the connected FTP server   | ---  | ---  | *1        |
| SvrDirName             | FTP server directory name     |        | 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       |
| GetFileNum             | Number of files to list       |        | Number of files to list   | 1 to 1000  |      | 1         |
| SortOrder*4            | Sort order                    |        | Order to sort the file list   | _NAME_ASC, _NAME_DESC, _DATE_ASC, _DATE_DESC   |      | _NAME_ASC |
| ExecOption             | FTP execution options         |        | Options for FTP execution   | ---  |      | ---       |
| RetryCfg               | Execution retry settings      |        | Instruction execution retry settings  | ---  |      | ---       |
| Cancel                 | Cancel                        | In-out | TRUE: Instruction execution is canceled.<br>FALSE: Instruction execution is not canceled. | Depends on data type.  | ---  | FALSE     |
| FileList[] array*5*6*7 | File details                  |        | Details for the obtained file list  | ---  |      | *1        |

|                  | Meaning                          | I/O    | Description   | Valid range           | Unit | Default |
|------------------|----------------------------------|--------|---|-----------------------|------|---------|
| Command-Canceled | Cancel completed                 | Output | TRUE: Canceling completed.<br>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.

|                  | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1230 for details on the structure <code>_sFTP_CONNECT_SVR</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SvrDirName       |  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| GetFileNum       |  |             |      |       |       |          | OK   |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SortOrder        | Refer to <i>Function</i> on page 2-1230 for the enumerators of the enumerated type <code>_eFILE_SORT_ORDER</code> .                                      |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ExecOption       | Refer to <i>Specifying Options for FTP Server Processing</i> on page 2-1232 for details on the structure <code>_sFTP_EXEC_OPTION</code> .                |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RetryCfg         | Refer to <i>Specifying Retrying Connection Processing with the FTP Server</i> on page 2-1232 for details on the structure <code>_sFTP_RETRY_CFG</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Cancel           | OK   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| FileList[] array | Refer to <i>Function</i> on page 2-1230 for details on the structure <code>_sFTP_FILE_DETAIL</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| 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 <sup>*1</sup>            | STRING            | 1 to 200 bytes <sup>*2</sup>    |      |         |
| PortNo     | Port number                   | TCP port number of FTP server control connection | UINT              | 0 to 65535 <sup>*3</sup>        | ---  | ---     |
| UserName   | User name                     | User name on FTP server                          | STRING            | 33 bytes max. <sup>*4*5*6</sup> |      |         |
| Password   | Password                      | FTP server password                              | STRING            | 33 bytes max. <sup>*4*5*6</sup> |      |         |

- \*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 "\_" (underscore).
- \*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 "\_" (underscore). You can also use "\" (backslash) 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                                 |
|-------------------------|---|
| <code>_NAME_ASC</code>  | Ascending order of names                |
| <code>_NAME_DESC</code> | Descending order of names               |
| <code>_DATE_ASC</code>  | Ascending order of last modified dates  |
| <code>_DATE_DESC</code> | 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     | ---  | Bytes |         |
| Size         | File size           | The file size <sup>*1</sup>  | ULINT             |  |       |         |
| ReadOnly     | Read-only attribute | The read-only attribute of the file or folder<br>TRUE: Read only<br>FALSE: Not read only | BOOL              | Depends on data type.  | ---   |         |
| Folder       | Folder              | TRUE: Folder<br>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. Refer to the instruction, *FTPGetFile* on page 2-1243, for details.

However, the option that is valid for this instruction is *ExecOption.PassiveMode* alone.

## 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:

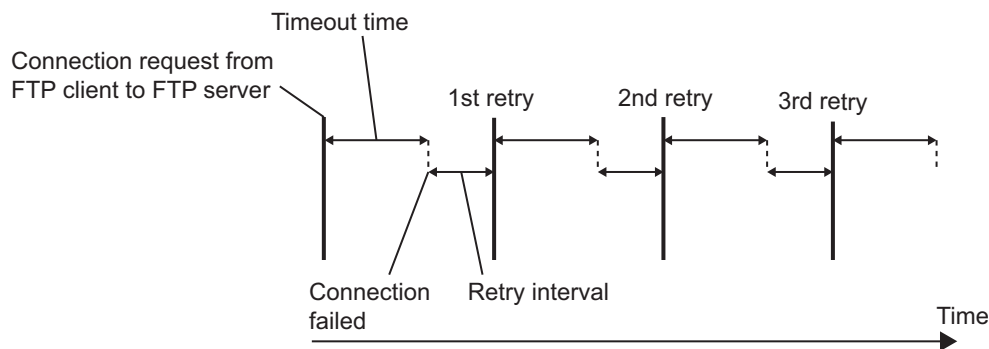


| Name          | Meaning                   | Description                                     | Data type       | Valid range              | Unit    | Default |
|---------------|---------------------------|---|-----------------|--------------------------|---------|---------|
| RetryCfg      | Execution re-try settings | Instruction execution re-try settings           | _sFTP_RETRY_CFG | ---                      | ---     | ---     |
| TimeOut       | Timeout time              | Timeout time for a connection to the FTP server | UINT            | 0 to 60 <sup>*1</sup>    | 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 <sup>*2</sup> | 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.

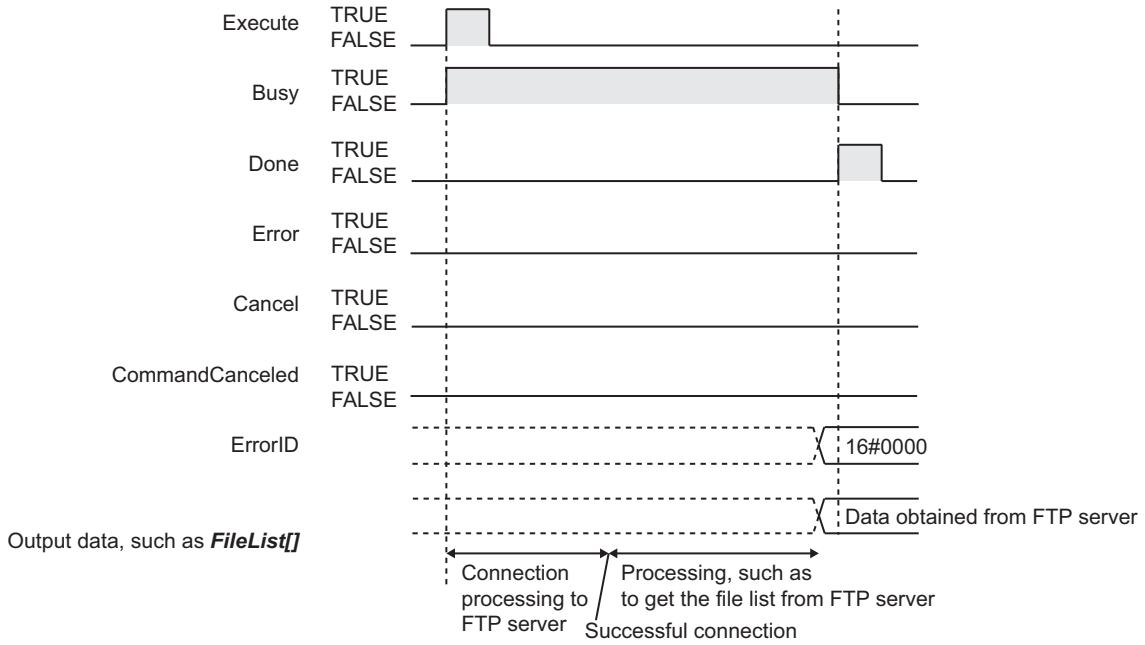


### ● Successfully Connecting to the FTP Server

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.

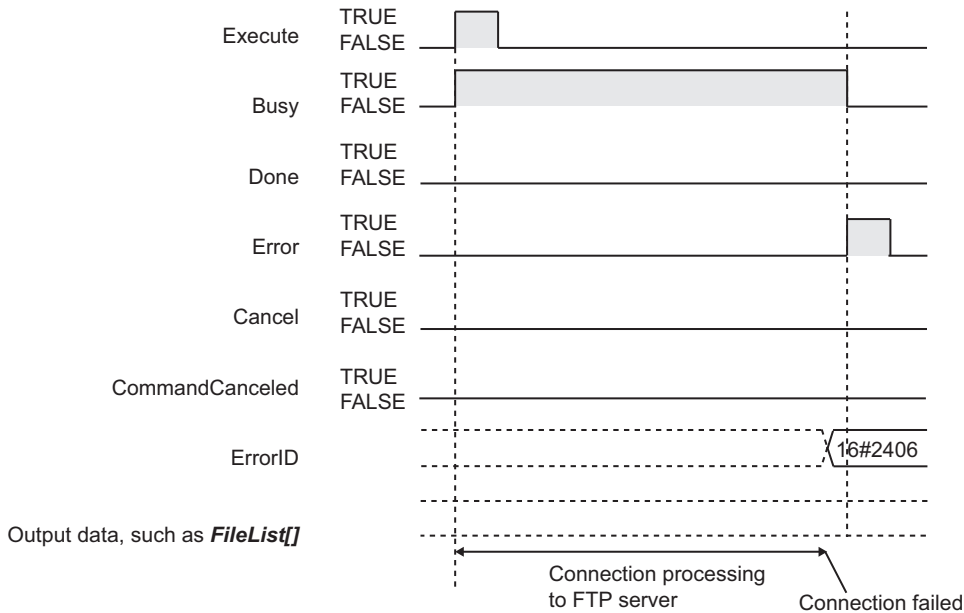


● **Failing to Connect to the FTP Server**

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

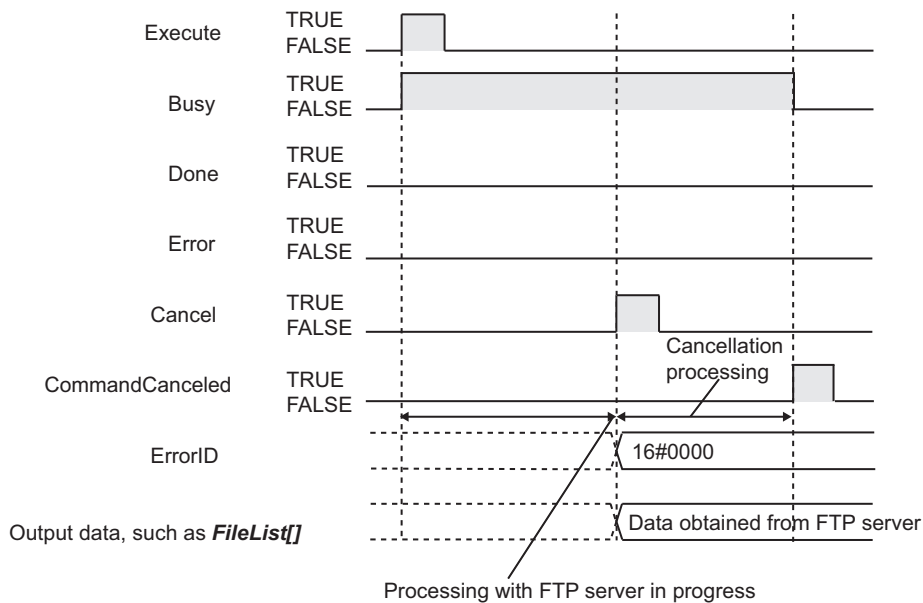
The following occurs if *Cancel* changes to TRUE while the FTPGetFileList instruction is obtaining a file list from the FTP server.

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*.

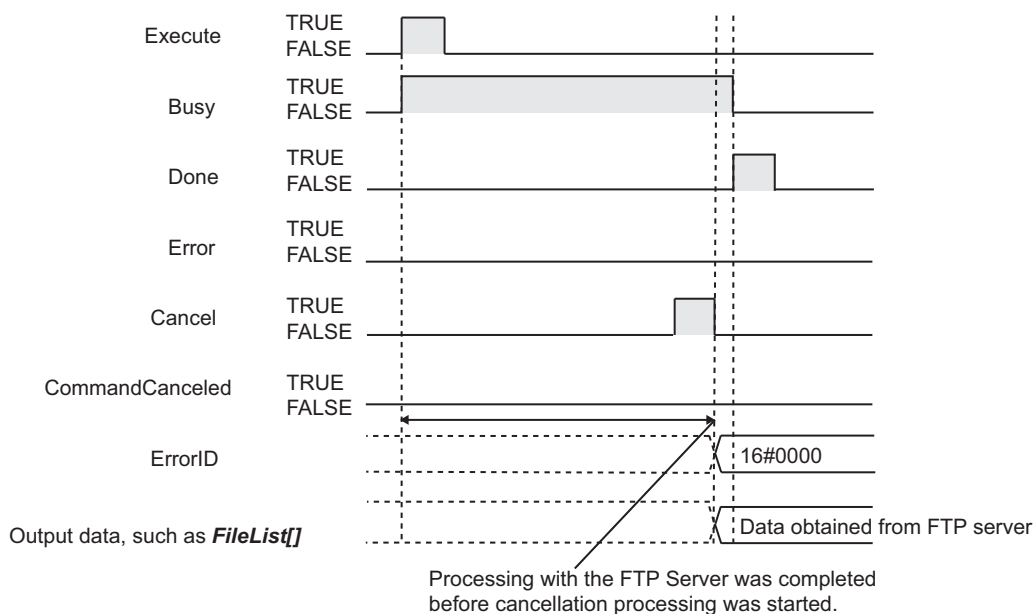
The value of *Done* does not change to TRUE.

The value of *CommandCanceled* changes to TRUE when cancellation is completed. Use this to confirm normal completion of cancellation.



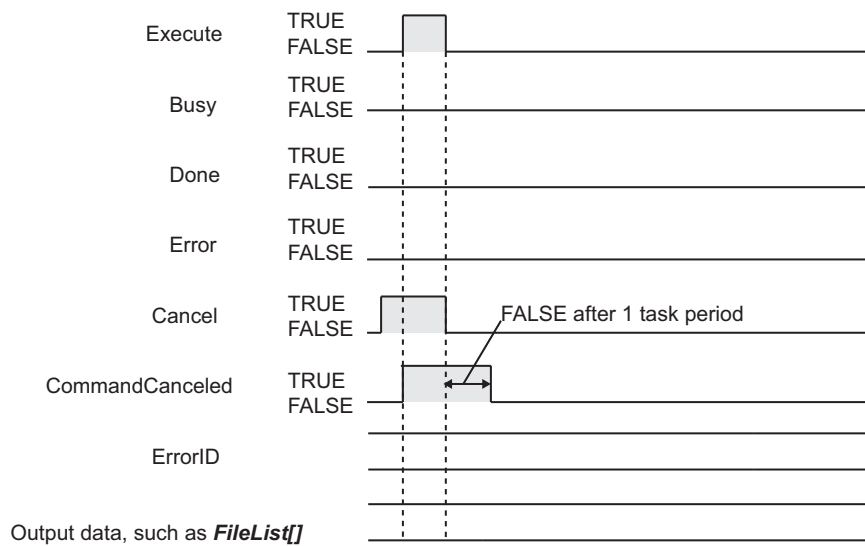
### ● When Processing with the FTP Server Is Completed Before Cancellation Processing Is Started

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   |
|---|---------|-----------|---|
| <i>_EIP_EtnOnlineSta</i> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <i>_EIP1_EtnOnlineSta</i> <sup>*2</sup> |         |           |   |
| <i>_EIP2_EtnOnlineSta</i> <sup>*3</sup> |         |           |   |

\*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.  
You can specify *\_EIP\_EtnOnlineSta* instead of *\_EIP1\_EtnOnlineSta*.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

### Precautions for Correct Use

- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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 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                            |

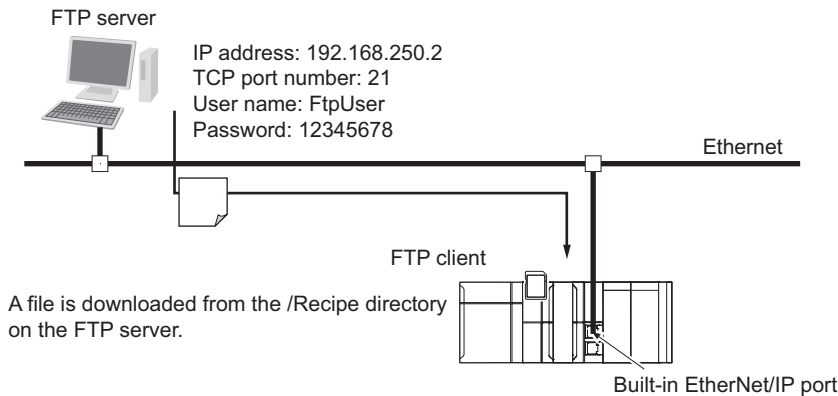
- You can execute a maximum of 3 of the following instructions at the same time: `FTPGetFileList`, `FTPGetFile`, `FTPputFile`, `FTPRemoveFile`, and `FTPRemoveDir`.
- An error will occur in the following cases. `Error` will change to `TRUE`.
  - a) The value of any input parameter is outside of the valid range.
  - b) `".."` is specified for a directory level in `SvrDirName`.
  - c) An incorrect path such as `"//"` is specified for `SvrDirName`.
  - d) The directory specified by `SvrDirName` does not exist on the FTP server.
  - e) The FTP server specified by `ConnectSvr` does not exist on the network or the specified FTP server is not operating.
  - f) More than 3 of the following instructions were executed at the same time: `FTPGetFileList`, `FTPGetFile`, `FTPputFile`, `FTPRemoveFile`, and `FTPRemoveDir`.
  - g) 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 <code>ErrorIDEx</code> | 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.<br>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.  |

## 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.

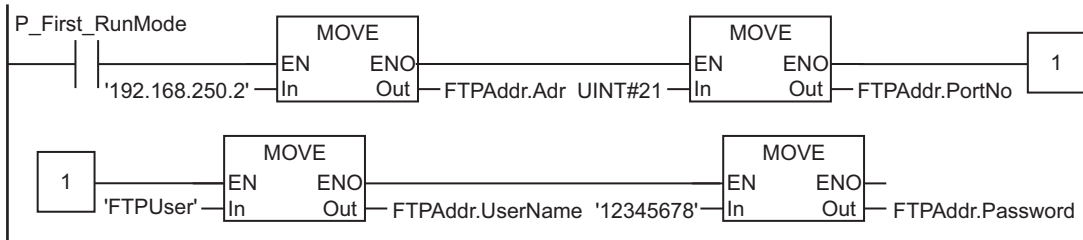
| Item                           | Specification            |
|--------------------------------|--------------------------|
| FTP server directory name      | '/Recipe'                |
| Number of files to list        | 1000                     |
| Sort order                     | Ascending order of 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.

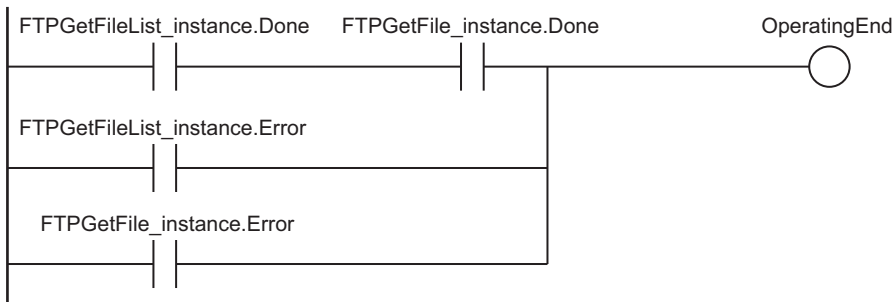
LD

| Internal Variables | Variable                | Data type                          | Initial value   | Comment   |
|--------------------|-------------------------|------------------------------------|---|---|
|                    | FTPGetFileList_instance | FTPGetFileList                     |   | Instance of FTPGetFileList instruction                      |
|                    | FTPGetFile_instance     | FTPGetFile                         |   | Instance of FTPGetFile instruction                          |
|                    | FTPAddr                 | _sFTP_CONNECT_SVR                  | (Adr := ", PortNo := 0, UserName := ", Password := ")   | Connected FTP server settings                               |
|                    | FTPFileList             | ARRAY[0..999] OF _sFTP_FILE_DETAIL | [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_RESULT   | [(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  |

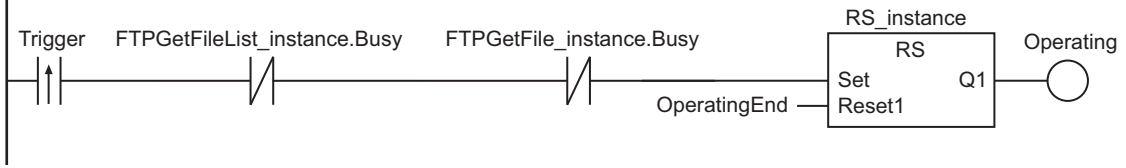
Prepare connected FTP server settings.



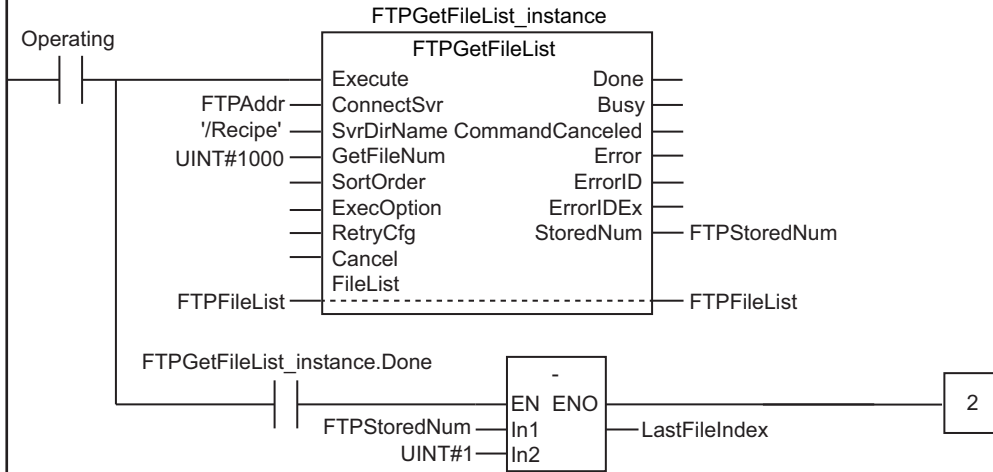
Determine if instruction execution is completed.



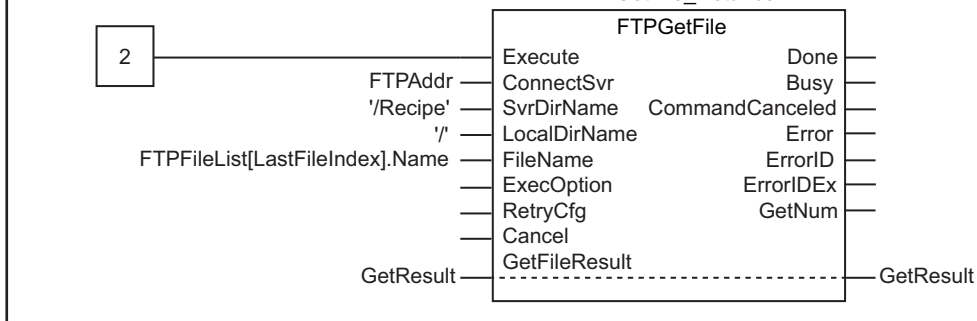
Accept trigger.



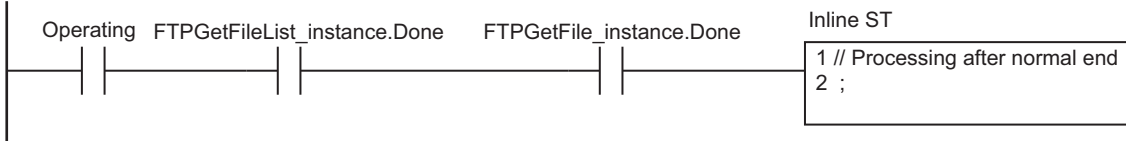
Execute FTPGetFileList instruction.



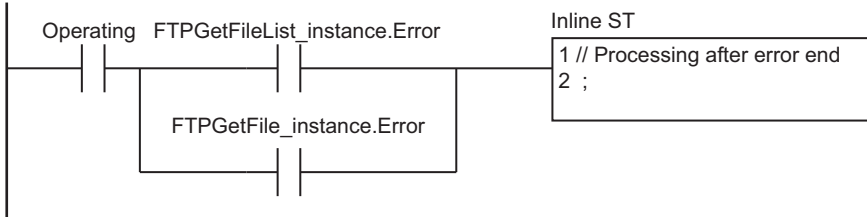
Execute FTPGetFile instruction.



Processing after normal end



Processing after error end





## ST

| Internal Variables | Variable                | Data type                              | Initial value   | Comment   |
|--------------------|-------------------------|--|---|---|
|                    | R_TRIG_instance         | R_TRIG                                 |   | Instance of R_TRIG instruction                        |
|                    | UP_Q                    | BOOL                                   | FALSE   | Trigger output  |
|                    | FTPGetFile_instance     | FTPGetFile                             |   | Instance of FTPGetFile instruction                    |
|                    | FTPGetFileList_instance | FTPGetFileList                         |   | Instance of FTPGetFileList instruction                |
|                    | FTPFileList             | ARRAY[0..999] OF _sFTP_FILE_DE<br>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 FTPGetFile |
|                    | FTPAddr                 | _sFTP_CONNECT_SVR                      | (Adr := ", PortNo := 0, UserName := ", Password := ")   | Connected FTP server settings                         |
|                    | GetResult               | ARRAY[0..0] OF _sFTP_FILE_RESULT       | [(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 directoryname
        FileName := FTPFileList[FTPStoredNum - 1].Name, // File name
        GetFileResult := GetResult); // Downloaded filerresults
    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/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------------------|------------|--------------------|--|
| FTPGetFile  | Get File from<br>FTP Server | FB         |                    | <pre>FTPGetFile_instance(Execute, ConnectSvr, SvrDirName, Local- DirName, FileName, ExecOption, RetryCfg, Cancel, GetFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx,Get- Num);</pre> |



## 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.

## Variables

|                              | Meaning                       | I/O    | Description   | Valid range  | Unit | Default |
|------------------------------|-------------------------------|--------|---|--|------|---------|
| ConnectSvr                   | Connected FTP server settings | Input  | Setting parameters for the connected FTP server   | ---  | ---  | *1      |
| SvrDirName                   | 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      |
| ExecOption                   | FTP execution options         |        | Options for FTP execution   | ---  |      | ---     |
| RetryCfg                     | Execution retry settings      |        | Instruction execution retry settings  | ---  |      | ---     |
| Cancel                       | Cancel                        |        | TRUE: Instruction execution is canceled.<br>FALSE: Instruction execution is not canceled. | Depends on data type.  |      | FALSE   |
| GetFile Result[] array*6*7*8 | Downloaded file results       | In-out | Downloaded file results   | ---  | ---  | *1      |
| Command-Canceled             | Cancel completed              | Output | TRUE: Canceling completed.<br>FALSE: Canceling not completed.                             | Depends on data type.  | ---  | ---     |
| GetNum                       | Number of files to download   |        | 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.

|                            | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1245 for details on the structure <code>_sFTP_CONNECT_SVR</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SvrDirName                 |  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| LocalDir-<br>Name          |  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| FileName                   |  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| ExecOption                 | Refer to <i>Specifying Options for FTP Server Processing</i> on page 2-1247 for details on the structure <code>_sFTP_EXEC_OPTION</code> .                |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RetryCfg                   | Refer to <i>Specifying Retrying Connection Processing with the FTP Server</i> on page 2-1249 for details on the structure <code>_sFTP_RETRY_CFG</code> . |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Cancel                     | OK   |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| GetFileRe-<br>sult[] array | Refer to <i>Function</i> on page 2-1245 for details on the structure <code>_sFTP_FILE_RESULT</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Command-<br>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.

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 "\" (backslash) 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.<br>FALSE: Transfer ended normally. | BOOL                   | Depends on data type.  |      |         |
| RemoveError   | Deletion error          | TRUE: Deletion ended in an error.<br>FALSE: Deletion ended normally. | BOOL                   |  |      |         |
| Reserved      | Reserved                | Reserved by the system.  | ARRAY[0..3]<br>Of Byte | ---  |      |         |

\*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)* <sup>1</sup> |
| *                      | 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      | ---                   | ---  | ---     |
| Passive-Mode | Passive mode specification                       | TRUE: Passive mode<br>FALSE: Active mode   | BOOL                   | Depends on data type. | ---  | FALSE   |
| ASCII Mode   | ASCII mode specification                         | TRUE: ASCII mode<br>FALSE: Binary mode   | BOOL                   |                       |      |         |
| FileRemove   | File deletion after transfer specification<br>*1 | TRUE: Delete files after transfer.<br>FALSE: Do not delete files after transfer.                         | BOOL                   |                       |      |         |
| OverWrite    | Overwrite specification                          | TRUE: Overwrite files at transfer destination.<br>FALSE: Do not overwrite files at transfer destination. | BOOL                   |                       |      |         |
| Reserved     | Reserved   | Reserved by the system.  | ARRAY[0..7]<br>Of Byte |                       |      |         |

\*1. The transfer source files are not deleted when the transfer fails.

### ● 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)* 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.<br>The data connection request is performed from the FTP client. |
| FALSE     | The data connection request with the FTP server is performed in active mode.<br>The data connection request is performed from the FTP server.  |

### ● ASCII Mode (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 *ASCII Mode* 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.<br>Text line feed codes are converted from those for the transfer source system to those for the transfer destination system. |



| Set value | Meaning   |
|-----------|---|
| FALSE     | Binary mode is used as the transfer mode from the transfer source system to the transfer destination system.<br>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. Refer to *Specifying Retrying Connection Processing with the FTP Server* on page 2-1232 for the FTPGetFileList instruction.

## 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. Refer to *Canceling Instruction Execution* on page 2-1234 for the FTPGetFileList instruction.

## Related System-defined Variables

| Name                             | Meaning                   | Data type | Description   |
|----------------------------------|---------------------------|-----------|---|
| _EIP_EtnOnlineSta <sup>*1</sup>  | Online                    | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta <sup>*2</sup> |                           |           |   |
| _EIP2_EtnOnlineSta <sup>*3</sup> |                           |           |   |
| _Card1Ready                      | SD Memory Card Ready Flag | BOOL      | This variable indicates whether the SD Memory Card is recognized and usable.<br>TRUE: Can be used.<br>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.  
You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.
- \*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Precautions for Correct Use

- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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 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 <i>OverWrite</i> | 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.

- b) "." is specified for a directory level in *SvrDirName* or *LocalDirName*.
  - c) An incorrect path such as "/" is specified for *SvrDirName* or *LocalDirName*.
  - d) The directory specified by *SvrDirName* does not exist on the FTP server.
  - e) More than 1,000 files to download exist in the FTP server directory specified with *SvrDirName*.
  - f) The file directory specified with *FileName* does not exist in the download source directory on the FTP server.
  - g) *ExecOption.OverWrite* is FALSE and a file with the same name as the specified file name *FileName* already exists in the specified directory *SvrDirName*.
  - h) *ExecOption.FileRemove* is TRUE but a file with a name that matches *FileName* has a read-only attribute.
  - i) The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - j) Accessing the file specified with *FileName* failed because there is no access right to the file or the file is corrupted.
  - k) More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
  - l) The SD Memory Card is not in a usable condition.
  - m) The SD Memory Card is write protected.
  - n) There is insufficient space available on the SD Memory Card.
  - o) 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 <i>ErrorIDEx</i> | 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.   |

## Sample Programming

Refer to *Sample Programming* on page 2-1238 for the FTPGetFileList instruction.

# FTPPutFile

The FTPPutFile instruction uploads a file to the FTP server.

| Instruction | Name                        | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------------------|------------|--------------------|--|
| FTPPutFile  | Put File onto<br>FTP Server | FB         |                    | FTPPutFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, PutFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, PutNum); |



## 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.

## Variables

|              | Meaning                       | I/O   | Description   | Valid range  | Unit | Default |
|--------------|-------------------------------|-------|---|--|------|---------|
| ConnectSvr   | Connected FTP server settings | Input | Setting parameters for the connected FTP server   | ---  | ---  | *1      |
| SvrDirName   | FTP server directory name     |       | Name of FTP server directory to which to upload a file                                    | 256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2 |      | "*3     |
| LocalDirName | Local directory name          |       | Name of the directory in which to store the file uploaded to the FTP server               | 256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)   |      | '/      |
| FileName     | File name                     |       | Name of file to upload*4  | ---  |      | *1      |
| ExecOption   | FTP execution options         |       | Options for FTP execution   | ---  |      | ---     |
| RetryCfg     | Execution retry settings      |       | Instruction execution retry settings  | ---  |      | ---     |
| Cancel       | Cancel                        |       | TRUE: Instruction execution is canceled.<br>FALSE: Instruction execution is not canceled. | Depends on data type.  |      | FALSE   |

|                             | Meaning                   | I/O    | Description   | Valid range           | Unit | Default |
|-----------------------------|---------------------------|--------|---|-----------------------|------|---------|
| PutFileResult[] array*5*6*7 | Uploaded file results     | In-out | Uploaded file results   | ---                   | ---  | *1      |
| Command-Canceled            | Cancel completed          | Output | TRUE: Canceling completed.<br>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 <i>Function</i> on page 2-1253 for details on the structure <code>_sFTP_CONNECT_SVR</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SvrDirName            |  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    | OK     |
| LocalDirName          |  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    | OK     |
| FileName              |  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    | OK     |
| ExecOption            | Refer to <i>Specifying Options for FTP Server Processing</i> on page 2-1255 for details on the structure <code>_sFTP_EXEC_OPTION</code> .                |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| RetryCfg              | Refer to <i>Specifying Retrying Connection Processing with the FTP Server</i> on page 2-1249 for details on the structure <code>_sFTP_RETRY_CFG</code> . |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Cancel                | OK   |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| PutFileResult[] array | Refer to <i>Function</i> on page 2-1253 for details on the structure <code>_sFTP_FILE_RESULT</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| 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.

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  | <code>_sFTP_CONNECT_SVR</code> | ---                             | ---  | ---     |
| Adr        | Address                       | IP address or host name <sup>*1</sup>            | STRING                         | 1 to 200 bytes <sup>*2</sup>    | ---  | ---     |
| PortNo     | Port number                   | TCP port number of FTP server control connection | UINT                           | 0 to 65535 <sup>*3</sup>        |      |         |
| UserName   | User name                     | User name on FTP server                          | STRING                         | 33 bytes max. <sup>*4*5*6</sup> |      |         |
| Password   | Password                      | FTP server password                              | STRING                         | 33 bytes max. <sup>*4*5*6</sup> |      |         |

\*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 "\" (backslash) 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 <sup>*1</sup> | 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.<br>FALSE: Transfer ended normally. | BOOL                | Depends on data type.  | ---  | ---     |
| RemoveError   | Deletion error          | TRUE: Deletion ended in an error.<br>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. Refer to *Using Wildcards to Specify File Names* on page 2-1246 for the FTPGetFile instruction.

## 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. Refer to *Specifying Options for FTP Server Processing* on page 2-1247 for the FTPGetFile instruction.

## 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. Refer to *Specifying Retrying Connection Processing with the FTP Server* on page 2-1232 for the FTPGetFileList instruction.

## 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. Refer to *Cancelling Instruction Execution* on page 2-1234 for the FTPGetFileList 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.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| _EIP1_EtnOnlineSta *2 |                           |           |   |
| _EIP2_EtnOnlineSta *3 |                           |           |   |
| _Card1Ready           | SD Memory Card Ready Flag | BOOL      | This variable indicates whether the SD Memory Card is recognized and usable.<br>TRUE: Can be used.<br>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.  
You can specify \_EIP\_EtnOnlineSta instead of \_EIP1\_EtnOnlineSta.
- \*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Precautions for Correct Use

- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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 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.

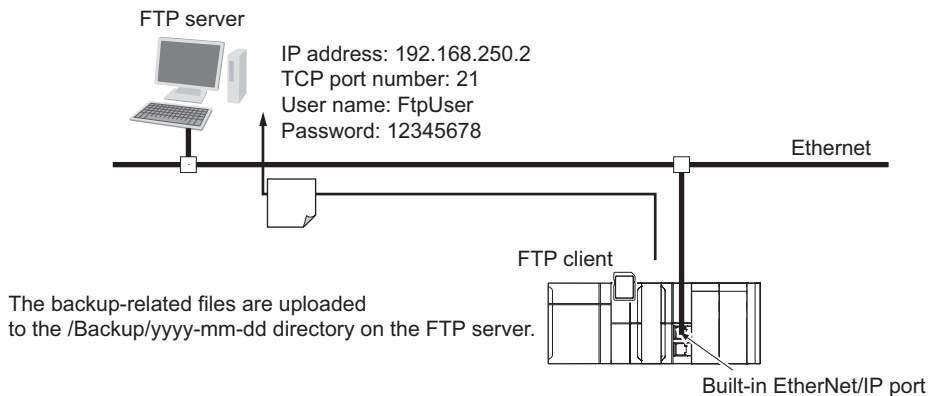


- If *ExecOption.FileRemove* is TRUE and the file specified with *FileName* has a read-only attribute, a deletion error occurs and the value of *PutFileResult[].RemoveError* changes to TRUE.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of any input parameter is outside of the valid range.
  - b) "." is specified for a directory level in *SvrDirName* or *LocalDirName*.
  - c) An incorrect path such as "/" is specified for *SvrDirName* or *LocalDirName*.
  - d) The directory specified by *SvrDirName* does not exist on the FTP server.
  - e) The directory specified by *LocalDirName* does not exist on the FTP client.
  - f) More than 1,000 files to upload exist in the directory specified with *LocalDirName*.
  - g) The file directory specified with *FileName* does not exist in the upload source directory on the SD Memory Card.
  - h) *ExecOption.OverWrite* is FALSE and a file with the same name as the specified file name *FileName* already exists in the specified directory *SvrDirName*.
  - i) *ExecOption.FileRemove* is TRUE but a file with a name that matches *FileName* has a read-only attribute.
  - j) The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - k) Accessing the file specified with *FileName* failed because there is no access right to the file or the file is corrupted.
  - l) More than 3 of the following instructions were executed at the same time: *FTPGetFileList*, *FTPGetFile*, *FTPPutFile*, *FTPRemoveFile*, and *FTPRemoveDir*.
  - m) 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 <i>ErrorIDEx</i> | 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.   |

## 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.

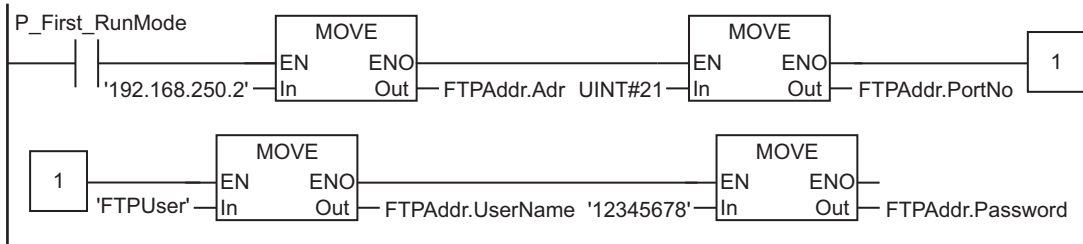
- 1** The BackupToMemoryCard instruction is used to save Controller backup-related files to the root directory on the SD Memory Card.
- 2** The FTPPutFile instruction is used to upload the backup-related files to the '/Backup/yyyy-mm-dd' directory on the FTP server. The wildcard specification '\*.\*' 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

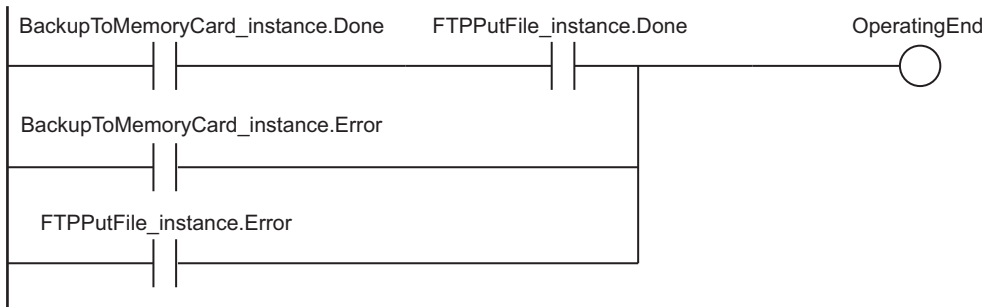
| Internal Variables | Variable            | Data type                        | Initial value   | Comment                            |
|--------------------|---------------------|----------------------------------|---|------------------------------------|
|                    | FTPPutFile_instance | 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              |

| Internal Variables | Variable                    | Data type          | Initial value | Comment                                    |
|--------------------|-----------------------------|--------------------|---------------|--|
|                    | 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 |

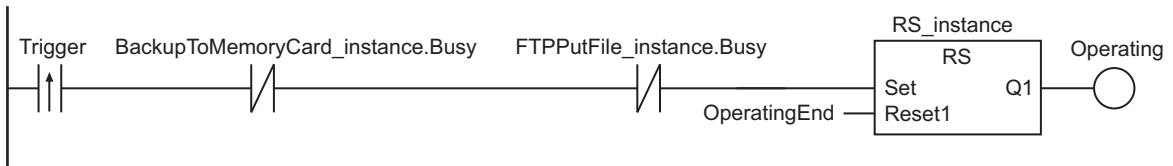
Prepare connected FTP server settings.



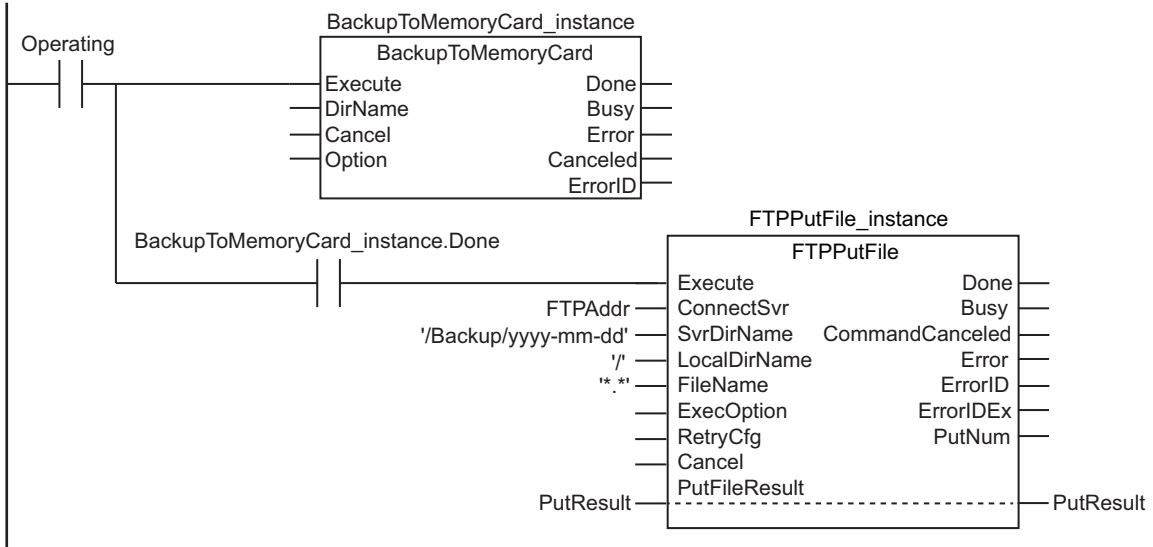
Determine if instruction execution is completed.



Accept trigger.



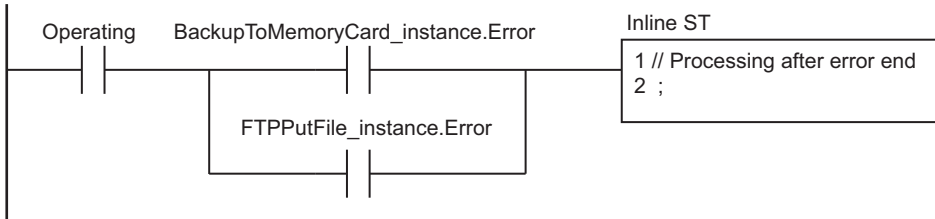
Execute BackupToMemoryCard and FTPPutFile instructions.



Processing after normal end



Processing after error end



## ST

| Internal Variables | Variable            | Data type                        | Initial value  | Comment   |
|--------------------|---------------------|----------------------------------|--|---|
|                    | R_TRIG_instance     | R_TRIG                           |  | Instance of R_TRIG instruction                            |
|                    | UP_Q                | BOOL                             | FALSE  | Trigger output  |
|                    | FTPPutFile_instance | FTPPutFile                       |  | Instance of FTPPutFile instruction                        |
|                    | DoFTPTrigger        | BOOL                             | FALSE  | Execution condition for BackupToMemoryCard and FTPPutFile |
|                    | 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                                     |

| Internal Variables | Variable                    | Data type          | Initial value | Comment                                    |
|--------------------|-----------------------------|--------------------|---------------|--|
|                    | Stage                       | UINT               | 0             | Instruction execution stage                |
|                    | Trigger                     | BOOL               | FALSE         | Execution condition                        |
|                    | BackupToMemoryCard_instance | BackupToMemoryCard |               | Instance of BackupToMemoryCard 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/<br>FUN | Graphic expression | ST expression   |
|--------------------|---------------------------|------------|--------------------|---|
| FTPRemove-<br>File | Delete FTP<br>Server File | FB         |                    | FTPRemoveFile_instance(Execute, ConnectSvr, SvrDirName, FileName, ExecOption, RetryCfg, Cancel, RemoveFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, RemoveNum); |



## 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.

## Variables

|            | Meaning                       | I/O   | Description   | Valid range  | Unit | Default |
|------------|-------------------------------|-------|---|--|------|---------|
| ConnectSvr | Connected FTP server settings | Input | Setting parameters for the connected FTP server   | ---  | ---  | *1      |
| SvrDirName | 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) <sup>*2</sup> |      | "*3     |
| FileName   | File name                     |       | Name of file to delete <sup>*4</sup>  | 256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character) <sup>*5</sup> |      | *1      |
| ExecOption | FTP execution options         |       | Options for FTP execution   | ---  |      |         |
| RetryCfg   | Execution retry settings      |       | Instruction execution retry settings  | ---  |      |         |
| Cancel     | Cancel                        |       | TRUE: Instruction execution is canceled.<br>FALSE: Instruction execution is not canceled. | Depends on data type.  |      | ---     |

|                                 | 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.<br>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 <i>Function</i> on page 2-1264 for details on the structure <code>_sFTP_CONNECT_SVR</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SvrDirName                |  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    | OK     |
| FileName                  |  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    | OK     |
| ExecOption                | Refer to <i>Specifying Options for FTP Server Processing</i> on page 2-1266 for details on the structure <code>_sFTP_EXEC_OPTION</code> .                |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| RetryCfg                  | Refer to <i>Specifying Retrying Connection Processing with the FTP Server</i> on page 2-1249 for details on the structure <code>_sFTP_RETRY_CFG</code> . |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Cancel                    | OK   |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Remove FileResult[] array | Refer to <i>Function</i> on page 2-1264 for details on the structure <code>_sFTP_FILE_RESULT</code> .  |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Command-Canceled          | OK   |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Remove-Num                |  |             |      |       |       |          | 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  | <code>_sFTP_CONNECT_SVR</code> | ---                 | ---  | ---     |
| 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 "\_" (underscore).

\*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 "\_" (underscore). You can also use "\" (backslash) 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 |
|------------------|----------------------|--|--------------------------------|--|------|---------|
| RemoveFileResult | Deleted file results | Transferred file results   | <code>_sFTP_FILE_RESULT</code> | ---  | ---  | ---     |
| 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.<br>FALSE: Transfer ended normally. | BOOL                           | Depends on data type.  |      |         |
| RemoveError      | Deletion error       | TRUE: Deletion ended in an error.<br>FALSE: Deletion ended normally. | BOOL                           |  |      |         |
| Reserved         | Reserved             | Reserved by the system.  | ARRAY[0..3] Of Byte            | ---  |      |         |

\*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. Refer to *Using Wildcards to Specify File Names* on page 2-1246 for the FTPGetFile instruction.

## 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. Refer to *Specifying Options for FTP Server Processing* on page 2-1247 for the FTPGetFileList instruction.

However, the option that is valid for this instruction is *ExecOption.PassiveMode* alone.

## 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. Refer to *Specifying Retrying Connection Processing with the FTP Server* on page 2-1232 for the FTPGetFileList instruction.

## 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. Refer to *Canceling Instruction Execution* on page 2-1234 for the FTPGetFileList instruction.

## Related System-defined Variables

| Name                                    | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <i>_EIP_EtnOnlineSta</i> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <i>_EIP1_EtnOnlineSta</i> <sup>*2</sup> |         |           |   |
| <i>_EIP2_EtnOnlineSta</i> <sup>*3</sup> |         |           |   |

\*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.  
You can specify *\_EIP\_EtnOnlineSta* instead of *\_EIP1\_EtnOnlineSta*.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Precautions for Correct Use

- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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.

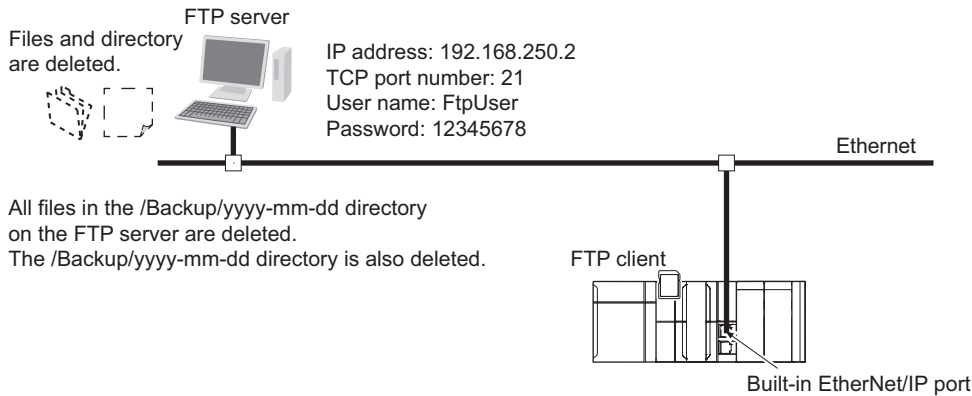
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the number of deleted files exceeds the number of array elements in `RemoveFileResult[]`, the results that will not fit are not stored. In this case, *Error* does not change to TRUE.
- If a file name exceeds 255 characters, the first 255 characters are stored in `Name` in `RemoveFileResult[].Name`. 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.
  - a) The file directory specified with `FileName` does not exist on the FTP server.
  - b) A file specified with `FileName` has a read-only attribute.
  - c) 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.
  - a) The value of any input parameter is outside of the valid range.
  - b) `".."` is specified for a directory level in `SvrDirName`.
  - c) An incorrect path such as `"//"` is specified for `SvrDirName`.
  - d) The directory specified by `SvrDirName` does not exist on the FTP server.
  - e) More than 1,000 files to delete exist in the directory specified with `SvrDirName`.
  - f) A file that matches the file name specified with a wildcard in `FileName` does not exist in the directory on the FTP server.
  - g) A file specified with `FileName` has a read-only attribute.
  - h) The FTP server specified by `ConnectSvr` does not exist on the network or the specified FTP server is not operating.
  - i) 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 <code>ErrorIDEx</code> | 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. |

| Value of ErrorIDEx | Meaning   | Correction  |
|--------------------|---|---|
| 16#00000229        | Execution was not possible because the file name was not correct. | Check the access rights of the specified directory. |

## 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.

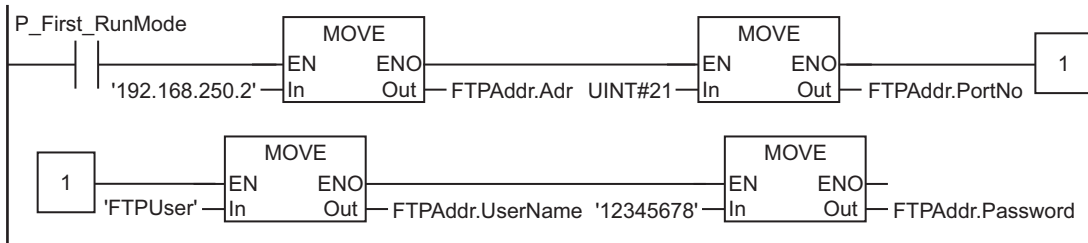
- 1** The FTPRemoveFile instruction is used to delete all of the files in the '/Backup/yyyy-mm-dd' directory on the FTP server. The wildcard specification '\*.\*' is used to specify the names of the files to delete.
- 2** The FTPRemoveDir instruction is used to delete the '/Backup/yyyy-mm-dd' directory 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

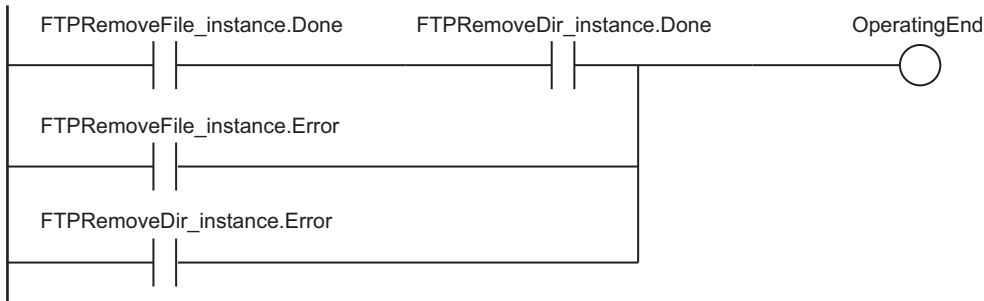
| Internal Variables | Variable                | Data type     | Initial value | Comment                               |
|--------------------|-------------------------|---------------|---------------|---------------------------------------|
|                    | FTPRemove-File_instance | FTPRemoveFile |               | Instance of FTPRemoveFile instruction |

| Internal Variables | Variable              | Data type                        | Initial value  | Comment                              |
|--------------------|-----------------------|----------------------------------|--|--------------------------------------|
|                    | 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                           |

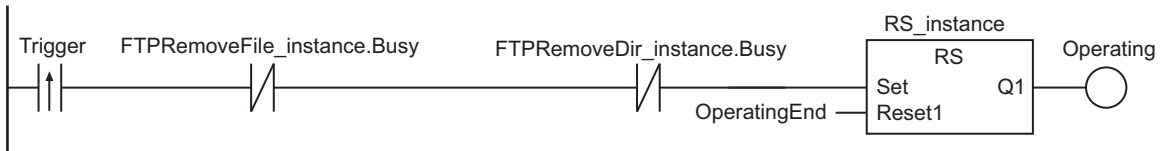
Prepare connected FTP server settings.



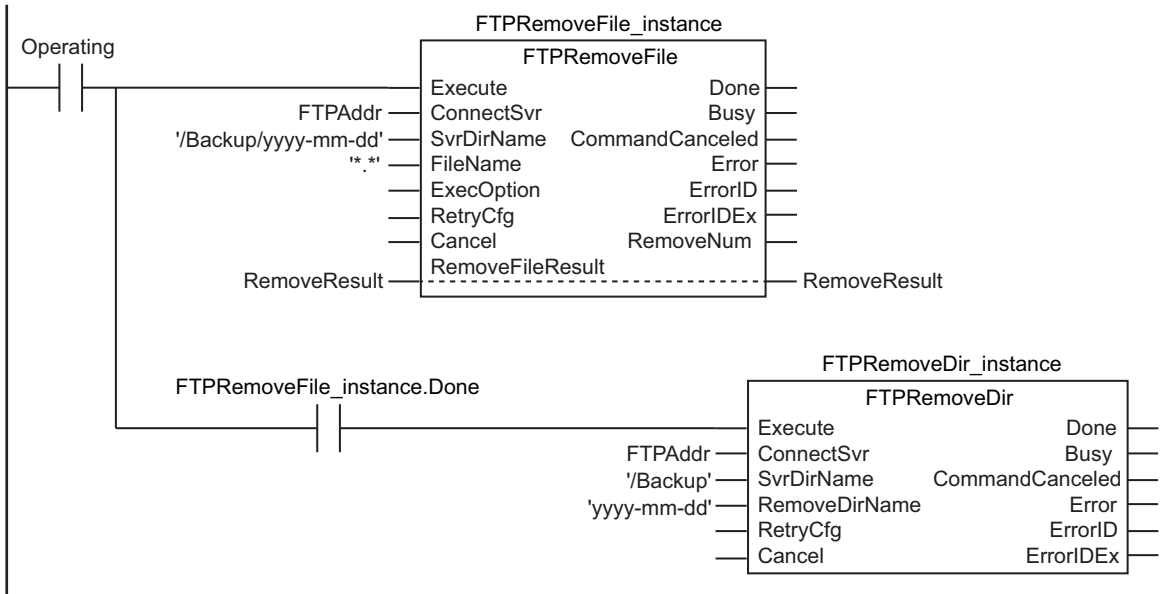
Determine if instruction execution is completed.



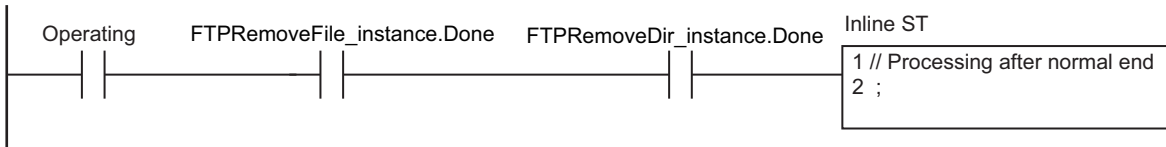
Accept trigger.



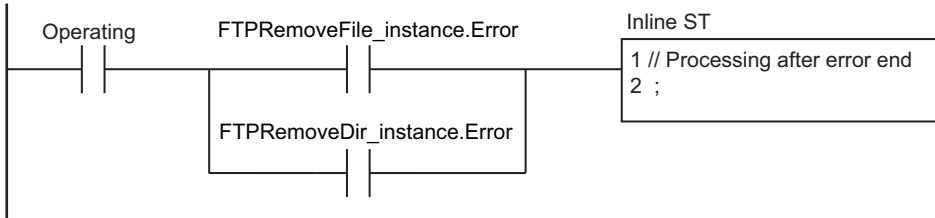
Execute FTPRemoveFile and FTPRemoveDir instructions.



Processing after normal end



Processing after error end



## ST

| Internal Variables | Variable               | Data type         | Initial value   | Comment  |
|--------------------|------------------------|-------------------|---|--|
|                    | R_TRIG_instance        | R_TRIG            |   | Instance of R_TRIG instruction                         |
|                    | UP_Q                   | BOOL              | FALSE   | Trigger output   |
|                    | FTPRemoveFile_instance | FTPRemoveFile     |   | Instance of FTPRemoveFile instruction                  |
|                    | FTPRemoveDir_instance  | FTPRemoveDir      |   | Instance of FTPRemoveDir instruction                   |
|                    | DoFTPTrigger           | BOOL              | FALSE   | Execution condition for FTPRemoveFile and FTPRemoveDir |
|                    | FTPAddr                | _sFTP_CONNECT_SVR | (Adr := ", PortNo := 0, UserName := ", Password := ") | Connected FTP server settings                          |

| Internal Variables | Variable     | Data type                        | Initial value   | Comment                     |
|--------------------|--------------|----------------------------------|---|-----------------------------|
|                    | RemoveResult | ARRAY[0..0] OF _sFTP_FILE_RESULT | [(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
      END_IF;
    END_CASE;
  END_IF;

```

```
        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/<br>FUN | Graphic expression | ST expression  |
|--------------|-----------------------------|------------|--------------------|--|
| FTPRemoveDir | Delete FTP Server Directory | FB         |                    | FTPRemoveDir_instance(Execute, ConnectSvr, SvrDirName, RemoveDirName, Cancel, RetryCfg, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx); |



## 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.

## Variables

|                  | Meaning                       | I/O    | Description   | Valid range  | Unit | Default |
|------------------|-------------------------------|--------|---|--|------|---------|
| ConnectSvr       | Connected FTP server settings | Input  | Setting parameters for the connected FTP server   | ---  | ---  | *1      |
| SvrDirName       | 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           |        | 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.<br>FALSE: Instruction execution is not canceled. | Depends on data type.  |      | FALSE   |
| Command-Canceled | Cancel completed              | Output | TRUE: Canceling completed.<br>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.

|                      | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1274 for details on the structure <code>_sFTP_CONNECT_SVR</code> .  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SvrDirName           |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| RemoveDir-<br>Name   |  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    | OK     |
| RetryCfg             | Refer to <i>Specifying Retrying Connection Processing with the FTP Server</i> on page 2-1249 for details on the structure <code>_sFTP_RETRY_CFG</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Cancel               | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Command-<br>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  | <code>_sFTP_CONNECT_SVR</code> | ---                 | ---  | ---     |
| 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 "\" (backslash) 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 the retry settings is the same as that for the FTPGetFileList instruction. Refer to *Specifying Retrying Connection Processing with the FTP Server* on page 2-1232 for the FTPGetFileList instruction.

## 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. Refer to *Canceling Instruction Execution* on page 2-1234 for the FTPGetFileList instruction.

## Related System-defined Variables

| Name  | Meaning | Data type | Description   |
|---|---------|-----------|---|
| <code>_EIP_EtnOnlineSta</code> <sup>*1</sup>  | Online  | BOOL      | This variable indicates when built-in EtherNet/IP port communications can be used.<br>TRUE: Communications are possible.<br>FALSE: Communications are not possible. |
| <code>_EIP1_EtnOnlineSta</code> <sup>*2</sup> |         |           |   |
| <code>_EIP2_EtnOnlineSta</code> <sup>*3</sup> |         |           |   |

\*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.

You can specify `_EIP_EtnOnlineSta` instead of `_EIP1_EtnOnlineSta`.

\*3. Use this variable name for port 2 on an NX-series CPU Unit.

## Precautions for Correct Use

- You can use this instruction for a built-in EtherNet/IP port on an NJ/NX-series CPU Unit.
- 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*.
- 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, 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.
  - 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.

- i) 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 <i>ErrorIDEx</i> | 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.   |

## Sample Programming

Refer to *Sample Programming* on page 2-1268 for the FTPRemoveFile instruction

# Serial Communications Instructions

| Instruction                    | Name   | Page        |
|--------------------------------|--|-------------|
| ExecPMCR                       | Protocol Macro   | page 2-1278 |
| SerialSend                     | SCU Send Serial  | page 2-1292 |
| SerialRcv and SerialRcvNoClear | SCU Receive Serial/SCU Receive Serial without Receive Buffer Clear | page 2-1303 |
| SendCmd                        | Send Command   | page 2-1318 |
| NX_SerialSend                  | Send No-protocol Data  | page 2-1330 |
| NX_SerialRcv                   | Receive No-protocol Data   | page 2-1343 |
| NX_ModbusRtuCmd                | Send Modbus RTU General Command                                    | page 2-1358 |
| NX_ModbusRtuRead               | Send Modbus RTU Read Command                                       | page 2-1369 |
| NX_ModbusRtuWrite              | Send Modbus RTU Write Command                                      | page 2-1380 |
| NX_SerialSigCtl                | Serial Control Signal ON/OFF Switching                             | page 2-1391 |
| NX_SerialSigRead               | Read Serial Control Signal   | page 2-1399 |
| NX_SerialStatusRead            | Read Serial Port Status  | page 2-1404 |
| NX_SerialBufClear              | Clear Buffer   | page 2-1409 |
| NX_SerialStartMon              | Start Serial Line Monitoring                                       | page 2-1419 |
| NX_SerialStopMon               | Stop Serial Line Monitoring  | page 2-1424 |

# ExecPMCR

The ExecPMCR instruction requests execution of a communications sequence (protocol data) registered in a Serial Communications Unit.

| Instruction | Name           | FB/<br>FUN | Graphic expression   | ST expression   |
|-------------|----------------|------------|--|---|
| ExecPMCR    | Protocol Macro | FB         | <pre> ExecPMCR_instance   ExecPMCR   - Execute  Done   - Port     Busy   - SeqNo   Error   - SrcDat  ErrorID   - DstDat   - ErrorIDEx           </pre> | ExecPMCR_instance(Execute, Port, SeqNo, SrcDat, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units.

## Variables

|                  | Meaning                        | I/O    | Description                    | Valid range           | Unit | Default |
|------------------|--------------------------------|--------|--------------------------------|-----------------------|------|---------|
| Port             | Destination port               | Input  | Destination port               | ---                   | ---  | ---     |
| SeqNo            | Communications sequence number |        | Communications sequence number | 0 to 999              |      | 0       |
| SrcDat[] (array) | Send data array                |        | Send data array                | Depends on data type. |      | *1      |
| DstDat[] (array) | Receive data array             | In-out | Receive data array             | Depends on data type. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, dates, and text strings |      |     |    |        |
|------------------|-------------|--|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|--------------|-------|---|------|-----|----|--------|
|                  | BOOL        | BYTE   | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL         | LREAL | TIME                                      | DATE | TOD | DT | STRING |
| Port             |             | Refer to <i>Function</i> on page 2-1278 for details on the structure <code>_sPORT</code> . |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SeqNo            |             |  |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SrcDat[] (array) |             |  | OK   |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| DstDat[] (array) |             |  | OK   |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

## Function

The ExecPMCR instruction requests execution of the sequence that is specified with communications sequence number *SeqNo* from the specified destination port *Port*.

If data is sent, it is sent from the second element (SrcDat[1]) of send data array SrcDat[]. The number of array elements to send is specified in SrcDat[0].

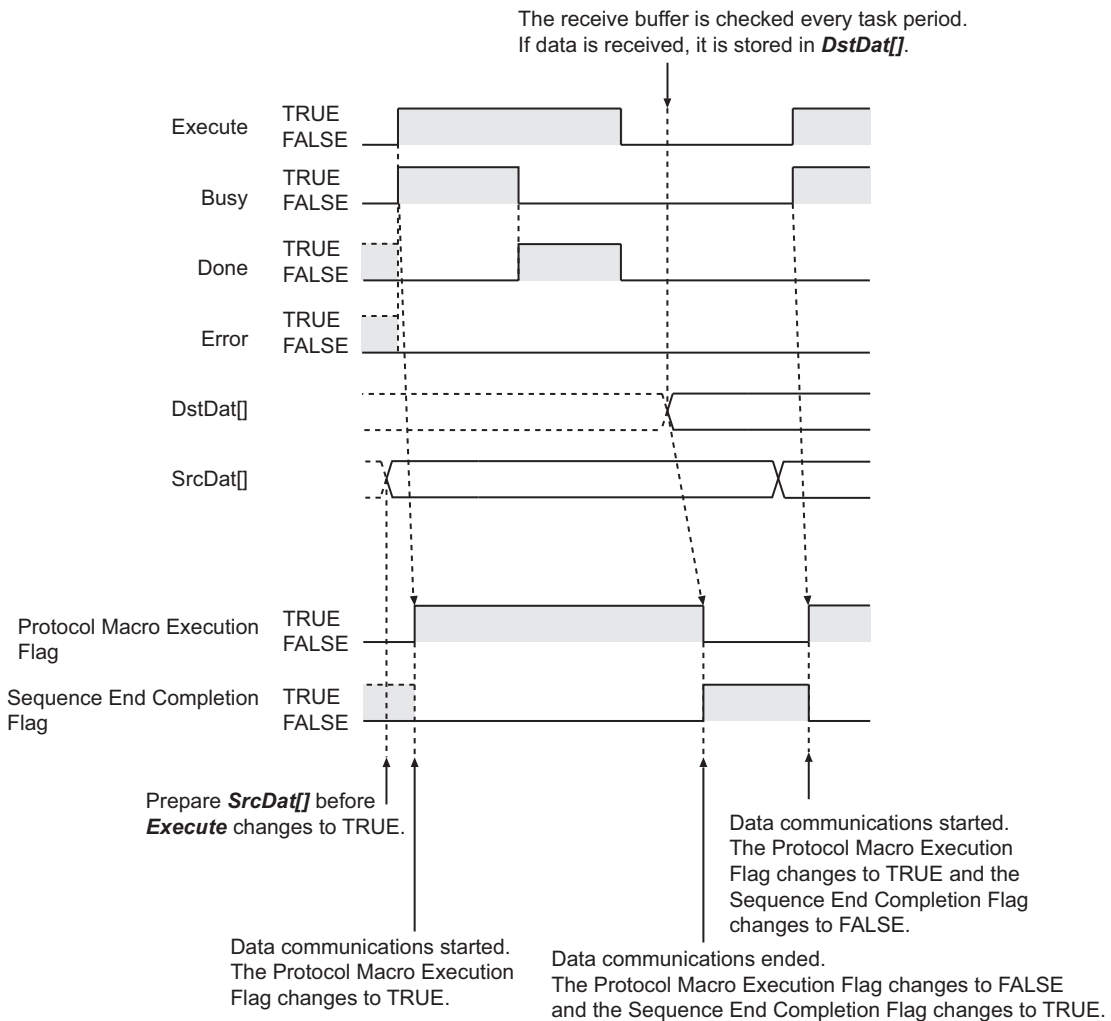
If data is received successfully, the receive data is stored from the second element (DstDat[1]) of receive data array DstDat[]. The number of receive data elements is stored in DstDat[0].

If data is not received successfully, the contents of DstDat[] from before instruction execution is retained for the number of elements specified in DstDat[0].

The data type of destination port *Port* is the structure `_sPORT`. The specifications are as follows:

| Name         | Meaning            | Description                                      | Data type             | Valid range                                      | Unit | Default                |
|--------------|--------------------|--|-----------------------|--|------|------------------------|
| Port         | Destination port   | Destination port                                 | <code>_sPORT</code>   | ---  | ---  | ---                    |
| UnitNo       | Unit number        | Unit number of Serial Communications Unit        | <code>_eUnitNo</code> | <code>_CBU_No00</code> to <code>_CBU_No15</code> | ---  | <code>_CBU_No00</code> |
| PhysicPortNo | Serial port number | Serial port number on Serial Communications Unit | USINT                 | 1 or 2   | ---  | 1                      |

The following figure shows a timing chart. Communications is performed to the end after the value of *Done* changes to TRUE.



## Related System-defined Variables

| Name  | Meaning   | Data type | Description   |
|---|---|-----------|---|
| _Port_numUsingPort                          | Number of Used Ports  | USINT     | This is the number of ports that are currently used.  |
| _Port_isAvailable                           | Network Communications Instruction Enabled Flag               | BOOL      | TRUE: A port is available.<br>FALSE: A port is not available.   |
| _CJB_SCU##P1ChgSta,<br>_CJB_SCU##P2ChgSta*1 | Serial Communications Unit ## Port 1/2 Settings Changing Flag | BOOL      | TRUE: The serial port settings are currently being changed.<br>FALSE: The serial port settings are currently not being changed. |

\*1. "##" denotes the unit number on the Serial Communications Unit.

## Related Semi-user-defined Variables

| Name              | Meaning                        | Data type | Description   |
|-------------------|--------------------------------|-----------|---|
| P#_PmrExecSta*1   | Protocol Macro Execution Flag  | BOOL      | TRUE: Protocol macro execution is in progress.<br>FALSE: Protocol macro execution is not in progress or failed. |
| P#_PmrSeqEndSta*1 | Sequence End Completion Flag   | BOOL      | TRUE: The sequence was completed with an End.<br>FALSE: The sequence was not completed with an End.             |
| P#_PmrSeqAbtSta*1 | Sequence Abort Completion Flag | BOOL      | TRUE: The sequence was completed with an Abort.<br>FALSE: The sequence was not completed with an Abort.         |

\*1. "#" denotes the port number on the Serial Communications Unit.

## Additional Information

Refer to the *SYSMAC CX-Protocol Operation Manual (Cat. No. W344)* for details on protocol macros.

## Precautions for Correct Use

- The ExecPMCR instruction starts execution of a protocol macro. Use the *P#PmrExecSta* (Protocol Macro Execution Flag) system-defined variable to check the status of protocol macro execution.
- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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*.
- An address in memory for CJ-series Units must be specified in the AT Specification attribute of *DstDat[]*.
- Set the value of *SrcDat[0]* and *DstDat[0]* to 0 to use a direct specification and link word specification. An error occurs if you set any other constant or variable, and the instruction is not executed.
- If the value of *DstDat[0]* is 0 or 1 and reception fails, all elements in *DstDat[]* change to 0.
- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *\_Port\_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.
- The *P#\_PmrExecSta* (Protocol Macro Execution Flag) semi-user-defined variable changes to TRUE when instruction execution is started. It changes to FALSE after the communications sequence is



completed and the receive data is stored in `DstDat[]`. You cannot execute this instruction for the same serial port until then. Therefore, use `P#_PmrExecSta` in an N.C. execution condition for the instruction.

- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing may not be possible.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The serial communications mode is not set to Protocol Macro Mode when the instruction is executed.
  - b) The value of `_Port_isAvailable` is FALSE.
  - c) The value of `SeqNo` is outside the valid range.
  - d) The value of `SeqNo` is not registered to a Serial Communications Unit.
  - e) The value of `Port.UnitNo` or `Port.PhysicPortNo` is outside the valid range.
  - f) There is no CJ-series Serial Communications Unit with the specified unit number.
  - g) The value of `SrcDat[0]` exceeds the size of `SrcDat[]`.
  - h) The value of `DstDat[0]` exceeds the size of `DstDat[]`.
  - i) The value of `SrcDat[0]` or `DstDat[0]` exceeds 250 words.
  - j) Communications fail.
  - k) An address in memory for CJ-series Units is not specified in the AT Specification attribute of `DstDat[]`.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000001 | The communications service was interrupted.                                   | <ul style="list-style-type: none"> <li>• Check the data link execution status.</li> <li>• Check the capacity of the transfer destination area on the third node.</li> </ul> |
| 16#00000101 | The local node is not part of the network.                                    | Make the local node part of the network.  |
| 16#00000102 | A token timeout occurred.   | Set the local node address to be within the maximum node address.   |
| 16#00000103 | The retry count was exceeded.   | Perform inter-node tests. If any error is found, check the operating environment.   |
| 16#00000104 | The allowable number of send frames was exceeded.                             | Check the status of events in the network and reduce the number of events in each task period. Or, increase the number of allowable send frames.                            |
| 16#00000105 | The IP address of the local node is out of range.                             | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000106 | The IP address of the local node is also used by another node in the network. | Change one of the node addresses that are duplicated.   |
| 16#00000201 | The remote node is not part of the network.                                   | Make the remote node part of the network.   |
| 16#00000202 | A Unit with the specified unit address does not exist at the destination.     | Correctly set the unit address for the destination network address.   |
| 16#00000203 | The third node is not part of the network.                                    | <ul style="list-style-type: none"> <li>• Check the address of the Unit that is the third node.</li> <li>• Specify only one node for the third node.</li> </ul>              |
| 16#00000204 | The remote node is busy.  | Increase the number of retries or correct the system so that communications traffic is not concentrated on the remote node.   |

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000205 | A response timeout occurred.  | Check the settings of the communications parameters.  |
| 16#00000206 | There is an error in the transmission path.   | <ul style="list-style-type: none"> <li>• Attempt retries.</li> <li>• If this error occurs frequently, check for noise.</li> </ul>   |
| 16#00000301 | A Communications Controller Error occurred.   | Refer to the operation manual for the relevant Unit and make suitable corrections.  |
| 16#00000302 | There is an error in the CPU Unit at the remote node.   | Refer to the manual for the CPU Unit at the remote node and remove the error.   |
| 16#00000303 | There is an error in the relevant Controller and a response is not returned.                                | Check the communications status on the network and restart the relevant Controller. If the error still occurs, replace the relevant Controller.   |
| 16#00000304 | The unit number setting is not correct.   | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000401 | The command that was sent is not supported.   | Set the command array correctly.  |
| 16#00000402 | The Unit model or version is not supported.   | Check the Unit model and version.   |
| 16#00000501 | The remote address setting is wrong.  | Set the destination address in the routing tables.  |
| 16#00000502 | Routing tables are not registered.  | Set the source node, destination node, and relay nodes in the routing tables.   |
| 16#00000503 | There is an error in the routing tables.  | Correct the settings in the routing tables.   |
| 16#00000504 | There are too many relay points.  | Restructure the network or correct the routing tables so that commands are used within a three-layer range.   |
| 16#00001001 | The command is too long.  | Set the command array correctly.  |
| 16#00001002 | The command is too short.   | Set the command array correctly.  |
| 16#00001003 | The number of write elements that is specified in the command does not agree with the number of write data. | Specify the same number of write elements and write data.   |
| 16#00001004 | The command format is incorrect.  | Set the command array correctly.  |
| 16#00001005 | There is an error in the header.  | Correct the settings in the routing tables.   |
| 16#00001101 | The area type does not exist.   | Refer to the command variables and parameter type codes and set the relevant codes.   |
| 16#00001102 | An access size is wrong.  | Correctly set the variable and parameter access sizes.  |
| 16#00001103 | An out-of-range address was specified.  | Specify an address that is within the process range.  |
| 16#00001104 | The address range was exceeded.   | <ul style="list-style-type: none"> <li>• Specify an address that is within the process range.</li> <li>• Correct the settings in the data link table.</li> </ul>                                      |
| 16#00001106 | A communications sequence number that is not registered was specified.                                      | Correct the communications sequence number or add the sequence with the CX-Protocol.  |
| 16#00001109 | An interrelationship error occurred.  | <ul style="list-style-type: none"> <li>• Correct the size relationships in the command data.</li> <li>• Correct the settings in the data link table.</li> </ul>                                       |
| 16#0000110A | Data is redundant.  | <ul style="list-style-type: none"> <li>• Cancel the current process or wait for it to be completed before you execute the command.</li> <li>• Correct the settings in the data link table.</li> </ul> |
| 16#0000110B | The response is too long.   | Set the number of elements in the command array correctly.  |
| 16#0000110C | This is another parameter error.  | Set the command array correctly.  |
| 16#00002002 | The data is protected.  | Execute the command again after clearing the protection.  |
| 16#00002003 | There is no registered table.   | Set the table correctly.  |

| Value       | Error  | Correction  |
|-------------|--|---|
| 16#00002004 | There is no data that matches the search data.   | Set the search data correctly.  |
| 16#00002005 | The relevant program number does not exist.  | Set a valid program number.   |
| 16#00002006 | The relevant file does not exist.  | Set the file name correctly, including the subdirectory names.  |
| 16#00002007 | A verification error occurred.   | <ul style="list-style-type: none"> <li>• Check the contents of memory and write the correct data.</li> <li>• Check the contents of the file.</li> </ul>   |
| 16#00002101 | Access is not possible because the area is read-only.  | Execute the command again after clearing the write protection.  |
| 16#00002102 | The data is protected or the data link table cannot be written.                                  | <ul style="list-style-type: none"> <li>• Execute the command again after clearing the write protection.</li> <li>• Set the system settings in the data link table.</li> </ul>                   |
| 16#00002103 | Registration is not possible.  | <ul style="list-style-type: none"> <li>• Create the file after deleting unnecessary files or prepare new file memory.</li> <li>• Execute the command again after closing open files.</li> </ul> |
| 16#00002105 | The relevant program number does not exist.  | Set a valid program number.   |
| 16#00002106 | The relevant file does not exist.  | Set the file name correctly, including the subdirectory names.  |
| 16#00002107 | A file with the same name already exists.  | Execute the command again after changing the name of the file to write.   |
| 16#00002108 | The change is not allowed because it causes an error.  | Correct the settings.   |
| 16#00002201 | The operation was not possible because a protocol macros is already in execution.                | Use an N.C. program input for the Protocol Macro Execution Flag.  |
| 16#00002202 | The operating mode is wrong.   | Check the operating mode.   |
| 16#00002203 | The operating mode is wrong for the instruction (PROGRAM mode).                                  | Check the operating mode of the Controller.   |
| 16#00002204 | The operating mode is wrong for the instruction (DEBUG mode).                                    | Check the operating mode of the Controller.   |
| 16#00002205 | The operating mode is wrong for the instruction (MONITOR mode).                                  | Check the operating mode of the Controller.   |
| 16#00002206 | The operating mode is wrong for the instruction (RUN mode).                                      | Check the operating mode of the Controller.   |
| 16#00002207 | The specified node is not the polling node.  | Confirm which node is the polling node of the network.  |
| 16#00002208 | The operating mode is wrong for the instruction.   | Check step activation status.   |
| 16#00002211 | The Unit is busy.  | Increase the number of retries or review the system so that communications traffic is not concentrated on the relevant Unit.  |
| 16#00002301 | The file device does not exist.  | Insert the media. Or, format the EM.  |
| 16#00002302 | There is no file memory.   | Check the file memory device.   |
| 16#00002303 | There is no built-in clock.  | Check the specifications of the model.  |
| 16#00002401 | A checksum error occurred in the protocol macro data, or the data transfer is not yet completed. | Transfer the protocol macro data from the CX-Protocol again.  |

| Value       | Error  | Correction  |
|-------------|--|---|
| 16#00002502 | There is an error in the memory.   | Transfer the correct data to the memory.  |
| 16#00002503 | The registered I/O Unit configuration does not agree with the actual Unit configuration. | Check the I/O Unit configuration.   |
| 16#00002504 | There are too many local or remote I/O points.   | Set the number of local and remote I/O points correctly.  |
| 16#00002505 | An error occurred in a data transmission between the CPU Unit and a CPU Bus Unit.        | Check the Units and the connecting cables. After removing the error, execute a command to reset the error.  |
| 16#00002506 | The same rack number, unit number, or I/O address is set more than once.                 | Correct the settings so that each number is unique.   |
| 16#00002507 | An error occurred in a data transmission between the CPU Unit and an I/O Unit.           | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#00002509 | There is an error in SYSMAC BUS/2 data transmission.                                     | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250A | An error occurred in a CPU Bus Unit data transmission.                                   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250D | The same channel setting is used more than once.   | Set the I/O channels correctly.   |
| 16#0000250F | There is an error in memory.   | <ul style="list-style-type: none"> <li>• For internal memory, execute the command again after writing the correct data.</li> <li>• For a Memory Card or EM file memory, execute the expansion memory format command.</li> <li>• If the error cannot be reset with the above corrections, replace the memory.</li> </ul> |
| 16#00002510 | The end station setting is wrong.  | Set the end station correctly.  |
| 16#00002601 | Protection is already cleared.   | You do not need to clear protection.  |
| 16#00002602 | The password was wrong.  | Specify the correct password.   |
| 16#00002604 | The data is protected.   | <ul style="list-style-type: none"> <li>• Execute the command again after clearing the write protection.</li> <li>• Wait for the service that is currently in execution to end, or stop the service and execute the command again.</li> </ul>  |
| 16#00002605 | The service is busy.   | Wait for the service that is currently in execution to end, or stop the service and execute the command again.  |
| 16#00002606 | The service is stopped.  | Execute the relevant service as required.   |
| 16#00002607 | You do not have the execution right.   | <ul style="list-style-type: none"> <li>• Execute the operation from the node that accessed the data link.</li> <li>• If the error still occurs after a restart, replace the Controller.</li> </ul>  |
| 16#00002608 | The environment is not set.  | Make the necessary settings.  |
| 16#00002609 | The required items are not set.  | Set the required items.   |
| 16#0000260A | The specified number is already defined.   | Execute the command again after changing the specified number to an action or transition number that is not already registered.   |
| 16#0000260B | The error cannot be reset.   | Remove the cause of the error and then execute the error reset command.   |
| 16#00003001 | You do not have access rights.   | Wait for the access to be allowed and then execute the command again.   |

| Value       | Error                        | Correction  |
|-------------|------------------------------|---|
| 16#00004001 | The service was interrupted. | Execute the command again after clearing the cause of the service interruption. |

**Note** In addition to the codes in the above table, the values of bits 6, 7, and 15 in the end code can be TRUE. If the value of bit 6 or 7 is TRUE, there is an error in the CPU Unit at the destination. If the value of bit 15 is TRUE, an error occurred during a network relay.

### Sample Programming

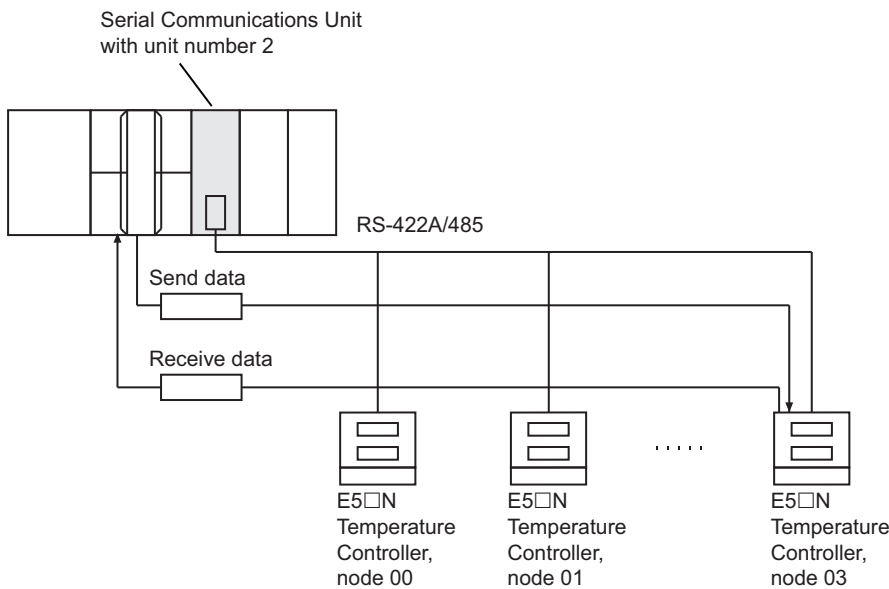
In this sample, a CJ-series Serial Communications Unit is used for data communications with an OMRON Temperature Controller. The present value of the Temperature Controller is read with a protocol macro. CompoWay/F master sequence 610 (Read Variable Area) is used.

The contents of send data array SendData[] is sent from the Controller.

The data received from the Temperature Controller is stored in receive data array RecvData[].

The following communications specifications are used.

| Item                           | Description                |
|--------------------------------|----------------------------|
| Unit used                      | Serial Communications Unit |
| Unit number                    | 2                          |
| Port number                    | 1 (RS-422/485)             |
| Communications sequence number | 610 (Read Variable Area)   |
| Remote node number             | 3                          |
| Data to read                   | Present value              |



The communications data for sequence 610 (Read Variable Area) is allocated as shown below.

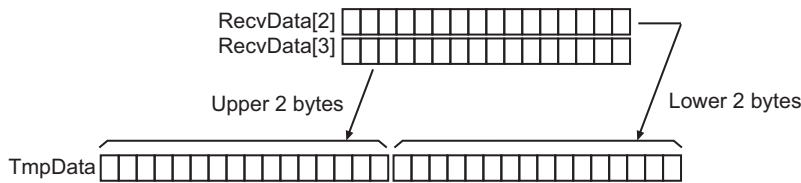
Send Data: WORD Array

|             |                           |
|-------------|---------------------------|
| SendData[0] | Number of send data words |
| SendData[1] | Not used. Node No.        |
| SendData[2] | Variable type             |
| SendData[3] | Read start address        |
| SendData[4] | Number of elements        |

Receive Data: WORD Array

|             |                              |
|-------------|------------------------------|
| RecvData[0] | Number of receive data words |
| RecvData[1] | Response code                |
| RecvData[2] | Receive data                 |
| RecvData[3] |                              |

If the data is received successfully, the lower two bytes (RecvData[2]) and the upper two bytes (RecvData[3]) of the present value of the Temperature Controller are assigned to *TmpData*.



## Send data SendData[] and Receive data RecvData[]

The contents of send data SendData[] and receive data RecvData[] are as follows:

### ● Send Data: WORD Array

| Variable    | Item  | Contents   | Value        |
|-------------|---|--|--------------|
| SendData[0] | Number of send data words                                   | Five words from SendData[0] to SendData[4] are sent.   | WORD#16#0005 |
| SendData[1] | Node number   | Node 3 is used.  | WORD#16#0003 |
| SendData[2] | Variable type + Upper byte of read start address            | To read the present value, the variable type is BYTE#16#C0 and the read start address is WORD#16#00. | WORD#16#C000 |
| SendData[3] | Lower byte of read start address + BYTE#16#00 (fixed value) |  | WORD#16#0000 |
| SendData[4] | Number of elements  | One element is read.   | WORD#16#0001 |

### ● Receive Data: WORD Array

| Variable    | Item                         | Contents   | Value        |
|-------------|------------------------------|--|--------------|
| RecvData[0] | Number of receive data words | Four words from RecvData[0] to RecvData[3] are received.                             | WORD#16#0004 |
| RecvData[1] | Response code                | WORD#16#0000 is returned for a normal end.   |              |
| RecvData[2] | Receive data                 | The lower two bytes of the present value of the Temperature Controller are returned. |              |
| RecvData[3] |                              | The upper two bytes of the present value of the Temperature Controller are returned. |              |

## Definitions of Global Variables

### ● Global Variables

| Name                | Data type | AT specification*1                              | Comment                      |
|---------------------|-----------|---|------------------------------|
| SCU_P1_PmrSeqEndSta | BOOL      | IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqEndSta | Sequence End Completion Flag |

| Name                | Data type | AT specification*1                              | Comment                        |
|---------------------|-----------|---|--------------------------------|
| SCU_P1_PmrSeqAbtSta | BOOL      | IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqAbtSta | Sequence Abort Completion Flag |
| SCU_P1_PmrExecSta   | BOOL      | IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrExecSta   | Protocol Macro Execution Flag  |

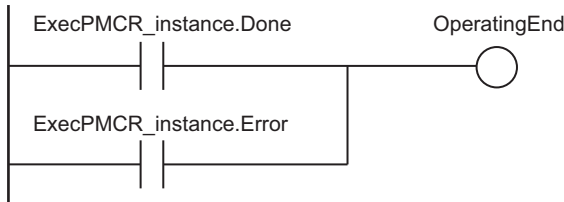
\*1. AT when the Serial Communications Unit is mounted to slot number 0 in rack number 0.

## LD

| Internal Variables | Variable          | Data type           | Initial value                        | AT    | Retain                              | Comment              |
|--------------------|-------------------|---------------------|--------------------------------------|-------|-------------------------------------|----------------------|
|                    | OperatingEnd      | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Processing completed |
|                    | Trigger           | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Execution condition  |
|                    | Operating         | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Processing           |
|                    | InPort            | _sPORT              | (UnitNo:=_CBU_No00, PhysicPortNo:=0) |       | <input type="checkbox"/>            | Port settings        |
|                    | SendData          | ARRAY[0..4] OF WORD | [5(16#0)]                            |       | <input type="checkbox"/>            | Send data            |
|                    | RecvData          | ARRAY[0..3] OF WORD | [4(16#0)]                            | %D200 | <input checked="" type="checkbox"/> | Receive data         |
|                    | TmpData           | DINT                | 0                                    |       | <input type="checkbox"/>            | Present value        |
|                    | RS_instance       | RS                  |                                      |       | <input type="checkbox"/>            |                      |
|                    | ExecPMCR_instance | ExecPMCR            |                                      |       | <input type="checkbox"/>            |                      |

| External Variables | Variable            | Data type | Comment   |
|--------------------|---------------------|-----------|---|
|                    | SCU_P1_PmrSeqEndSta | BOOL      | Sequence End Completion Flag                    |
|                    | SCU_P1_PmrSeqAbtSta | BOOL      | Sequence Abort Completion Flag                  |
|                    | SCU_P1_PmrExecSta   | BOOL      | Protocol Macro Execution Flag                   |
|                    | _Port_isAvailable   | BOOL      | Network Communications Instruction Enabled Flag |

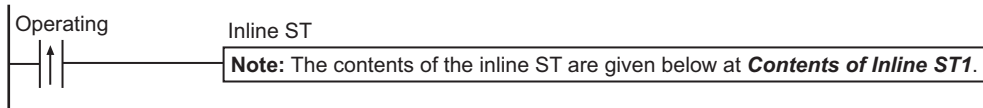
Determine if execution of the ExecPMCR instruction is completed.



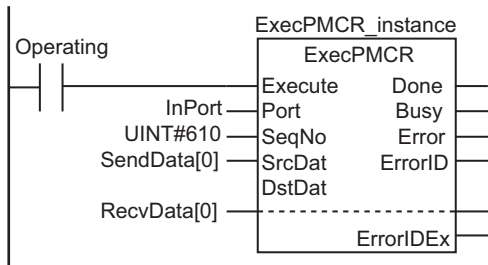
Accept trigger.



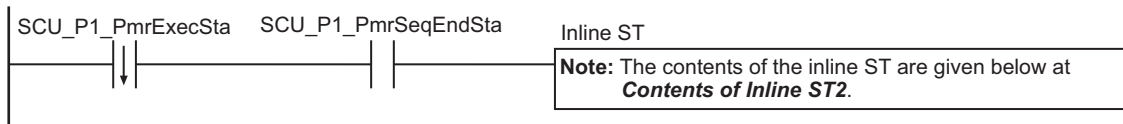
Set communications parameters.



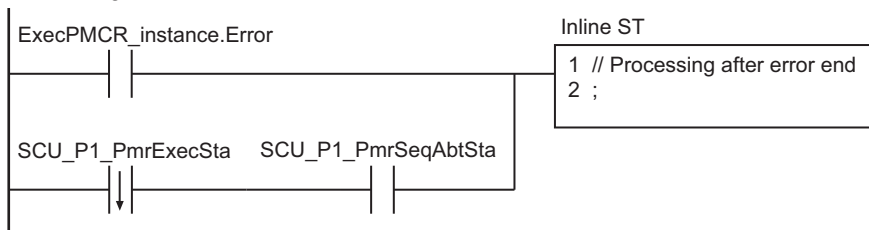
Execute ExecPMCR instruction.



Processing after normal end



Processing after error end



### ● Contents of Inline ST1

```
InPort.UnitNo      :=_CBU_No02;    // Serial Communications Unit with unit number 2
InPort.PhysicPortNo :=USINT#1;    // Port number 1
SendData[0]        :=WORD#16#0005;
SendData[1]        :=WORD#16#0003;
SendData[2]        :=WORD#16#C000;
SendData[3]        :=WORD#16#0000;
SendData[4]        :=WORD#16#0001;
RecvData[0]        :=WORD#16#0004;
```

### ● Contents of Inline ST2

```
// Processing after normal end
TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(
    RecvData[3]), 16) OR WORD_TO_DWORD(RecvData[2]));
```



## ST

| Internal Variables | Variable          | Data type           | Initial value                        | AT    | Retain                              | Comment                                      |
|--------------------|-------------------|---------------------|--------------------------------------|-------|-------------------------------------|--|
|                    | State             | INT                 | 0                                    |       | <input type="checkbox"/>            | Current state                                |
|                    | Trigger           | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Execution condition                          |
|                    | LastTrigger       | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Value of Trigger from previous task period   |
|                    | InPort            | _sPORT              | (UnitNo:=_CBU_No00, PhysicPortNo:=0) |       | <input type="checkbox"/>            | Port settings                                |
|                    | SendData          | ARRAY[0..4] OF WORD | [5(16#0)]                            |       | <input type="checkbox"/>            | Send data                                    |
|                    | RecvData          | ARRAY[0..3] OF WORD | [4(16#0)]                            | %D200 | <input checked="" type="checkbox"/> | Receive data                                 |
|                    | End_ExecPMCR      | BOOL                | FALSE                                |       | <input type="checkbox"/>            | Completion of ExecPMCR instruction execution |
|                    | TmpData           | DINT                | 0                                    |       | <input type="checkbox"/>            | Present value                                |
|                    | RS_instance       | RS                  |                                      |       | <input type="checkbox"/>            |  |
|                    | ExecPMCR_instance | ExecPMCR            |                                      |       | <input type="checkbox"/>            |  |
|                    | F_TRIG_instance   | F_TRIG              |                                      |       | <input type="checkbox"/>            |  |

| External Variables | Variable            | Data type | Comment   |
|--------------------|---------------------|-----------|---|
|                    | SCU_P1_PmrSeqEndSta | BOOL      | Sequence End Completion Flag                    |
|                    | SCU_P1_PmrSeqAbtSta | BOOL      | Sequence Abort Completion Flag                  |
|                    | SCU_P1_PmrExecSta   | BOOL      | Protocol Macro Execution Flag                   |
|                    | _Port_isAvailable   | BOOL      | Network Communications Instruction Enabled Flag |

```

// Accept trigger.
IF (State=INT#0) THEN
    IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE) AND (SCU_P1_PmrExecSta<>TRUE)
        AND (ExecPMCR_instance.Busy<>TRUE) ) THEN
        State:=INT#1;
    END_IF;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize ExecPMCR instruction.
IF (State=INT#1) THEN
    InPort.UnitNo          :=_CBU_No02;    // Serial Communications Unit with unit
number 2
    InPort.PhysicPortNo   :=USINT#1;     // Port number 1

```

```

SendData[0]           :=WORD#16#0005;
SendData[1]           :=WORD#16#0003;
SendData[2]           :=WORD#16#C000;
SendData[3]           :=WORD#16#0000;
SendData[4]           :=WORD#16#0001;
RecvData[0]           :=WORD#16#0004;
ExecPMCR_instance(
    Execute :=FALSE,          // Initialize ExecPMCR instruction.
    SrcDat  :=SendData[0],    // Dummy
    DstDat  :=RecvData[0]);
    State:=INT#2;
END_IF;
// Execute ExecPMCR instruction.
IF (State=INT#2) THEN
    ExecPMCR_instance(
        Execute :=TRUE,
        Port    :=InPort,
        SeqNo   :=UINT#610,
        SrcDat  :=SendData[0],
        DstDat  :=RecvData[0]);

    F_TRIG_instance(SCU_P1_PmrExecSta, End_ExecPMCR);

    IF (End_ExecPMCR=TRUE) THEN
        End_ExecPMCR:=FALSE;
        State:=INT#3;
    END_IF;

    IF (ExecPMCR_instance.Error=TRUE) THEN
        State:=INT#5;
    END_IF;
END_IF;

// Confirm completion of ExecPMCR instruction execution.
IF (State=INT#3) THEN
    IF (SCU_P1_PmrSeqEndSta=TRUE) THEN
        State:=INT#4;
    END_IF;
    IF (SCU_P1_PmrSeqAbtSta=TRUE) THEN
        State:=INT#5;
    END_IF;
END_IF;

IF (State=INT#4) THEN
    // Processing after normal end.
    TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(RecvData[3]), 16)
        OR WORD_TO_DWORD(RecvData[2]));

```

```
    State:=INT#0;  
END_IF;  
  
IF (State=INT#5) THEN  
    // Processing after error end  
    State:=INT#0;  
END_IF;
```

# SerialSend

The SerialSend instruction sends data in No-protocol Mode from a serial port on a Serial Communications Unit.

| Instruction | Name            | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|-----------------|------------|--------------------|--|
| SerialSend  | SCU Send Serial | FB         |                    | SerialSend_instance(Execute, Port, SrcDat, SendSize, Done, Busy, Error, ErrorID, ErrorIDEx); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units.

## Variables

|                  | Meaning          | I/O   | Description                     | Valid range           | Unit  | Default |
|------------------|------------------|-------|---------------------------------|-----------------------|-------|---------|
| Port             | Destination port | Input | Destination port                | ---                   | ---   | ---     |
| SrcDat[] (array) | Send data array  |       | Send data array                 | Depends on data type. |       | *1      |
| SendSize         | Send data size   |       | Data size to send from SrcDat[] | 0 to 256              | Bytes | 1       |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |  |
|------------------|-------------|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|--|
|                  | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT         | REAL | LREAL                                     | TIME | DATE | TOD | DT | STRING |  |
| Port             |             |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |  |
| SrcDat[] (array) |             | OK          |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |  |
| SendSize         |             |             |      |       |       |          | OK   |       |       |      |     |      |              |      |   |      |      |     |    |        |  |

## Function

The SerialSend instruction sends data in No-protocol Mode from the port of the Serial Communications Unit specified with *Port*.

The data to send is contained in SrcDat[] (send data array). The size of the data to send is specified in *SendSize* (send data size).

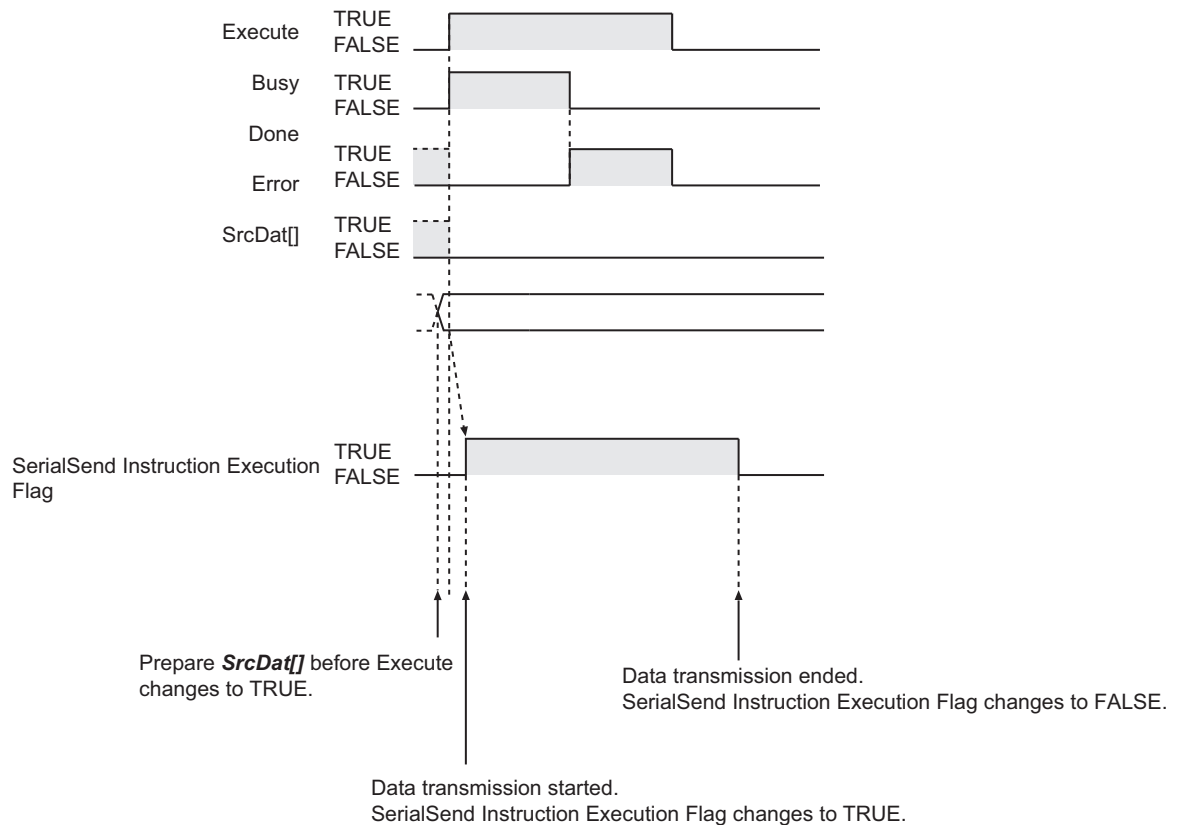
To attach start and end codes to the send data, set them in the DM Area words that are assigned to the Special Unit.

If you add start and end codes, the maximum number of bytes to send is 259 (1-byte start code, 2-byte end code (for CR+LF specification), and 256 bytes of send data).

The data type of destination port *Port* is the structure `_sPORT`. The specifications are as follows:

| Name         | Meaning            | Description                                      | Data type             | Valid range                                      | Unit | Default                |
|--------------|--------------------|--|-----------------------|--|------|------------------------|
| Port         | Destination port   | Destination port                                 | <code>_sPORT</code>   | ---  | ---  | ---                    |
| UnitNo       | Unit number        | Unit number of Serial Communications Unit        | <code>_eUnitNo</code> | <code>_CBU_No00</code> to <code>_CBU_No15</code> | ---  | <code>_CBU_No00</code> |
| PhysicPortNo | Serial port number | Serial port number on Serial Communications Unit | USINT                 | 1 or 2   | ---  | 1                      |

The following figure shows a timing chart. Communications are performed to the end after the value of *Done* changes to TRUE.



## Related System-defined Variables

| Name                            | Meaning   | Data type | Description   |
|---------------------------------|---|-----------|---|
| <code>_Port_numUsingPort</code> | Number of Used Ports                            | USINT     | This is the number of ports that are currently used.          |
| <code>_Port_isAvailable</code>  | Network Communications Instruction Enabled Flag | BOOL      | TRUE: A port is available.<br>FALSE: A port is not available. |

## Related Semi-user-defined Variables

| Name                                  | Meaning                               | Data type | Description   |
|---------------------------------------|---------------------------------------|-----------|---|
| P#_NopSerialSendExecSta* <sup>1</sup> | SerialSend Instruction Execution Flag | BOOL      | TRUE: Execution of the SerialSend instruction is in progress.<br>FALSE: Execution of the SerialSend instruction is not in progress. |
| P#_NopStartCodeYNCfg* <sup>1</sup>    | No-protocol Start Code Enable         | BOOL      | TRUE: Start code<br>FALSE: No start code  |
| P#_NopEndCodeYNCfg* <sup>1</sup>      | No-protocol End Code Enable           | BOOL      | TRUE: End code<br>FALSE: No end code  |
| P#_NopCRLFCfg* <sup>1</sup>           | No-protocol CR LF Specification       | BOOL      | TRUE: CR+LF<br>FALSE: No CR+LF  |
| P#_NopStartCodeCfg* <sup>1</sup>      | No-protocol Start Code                | USINT     | 16#00 to 16#FF  |
| P#_NopEndCodeCfg* <sup>1</sup>        | No-protocol End Code                  | USINT     | 16#00 to 16#FF  |

\*1. "#" denotes the port number on the Serial Communications Unit.

## Additional Information

Refer to the following manual for details on no-protocol communications.

- *CJ-series Serial Communications Units Operation Manual for NJ-series CPU Unit (Cat. No. W494)*

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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 can use this instruction only for a Serial Communications Unit's serial port that is set to No-protocol Mode.
- Nothing is sent if the value of *SendSize* is 0. When the instruction is executed, the value of *Done* changes to TRUE.
- Even when attaching a start or end code, do not include it in the value of *SendSize*.
- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *\_Port\_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.
- You cannot execute this instruction while the SerialSend Instruction Executing Flag (semi-user-defined variable *P#NopSerialSendExecSta*) is TRUE. Use *P#NopSerialSendExecSta* in an N.C. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing may not be possible.
- An error occurs in the following cases. *Error* will change to TRUE.
  - a) The serial communications mode is not set to No-protocol Mode when the instruction is executed.
  - b) The value of *\_Port\_isAvailable* is FALSE.
  - c) The value of *Port.UnitNo* or *Port.PhysicPortNo* is outside the valid range.

- d) There is no CJ-series Serial Communications Unit with the specified unit number.
  - e) The value of *SendSize* is outside of the valid range.
  - f) The value of *SendSize* exceeds the size of *SrcDat[]*.
  - g) Communications fail.
  - h) The instruction is executed during a Unit restart.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000001 | The communications service was interrupted.                                   | <ul style="list-style-type: none"> <li>• Check the data link execution status.</li> <li>• Check the capacity of the transfer destination area on the third node.</li> </ul> |
| 16#00000101 | The local node is not part of the network.                                    | Make the local node part of the network.  |
| 16#00000102 | A token timeout occurred.   | Set the local node address to be within the maximum node address.   |
| 16#00000103 | The retry count was exceeded.   | Perform inter-node tests. If any error is found, check the operating environment.   |
| 16#00000104 | The allowable number of send frames was exceeded.                             | Check the status of events in the network and reduce the number of events in each task period. Or, increase the number of allowable send frames.                            |
| 16#00000105 | The IP address of the local node is out of range.                             | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000106 | The IP address of the local node is also used by another node in the network. | Change one of the node addresses that are duplicated.   |
| 16#00000201 | The remote node is not part of the network.                                   | Make the remote node part of the network.   |
| 16#00000202 | A Unit with the specified unit address does not exist at the destination.     | Correctly set the unit address for the destination network address.   |
| 16#00000203 | The third node is not part of the network.                                    | <ul style="list-style-type: none"> <li>• Check the address of the Unit that is the third node.</li> <li>• Specify only one node for the third node.</li> </ul>              |
| 16#00000204 | The remote node is busy.  | Increase the number of retries or correct the system so that communications traffic is not concentrated on the remote node.   |
| 16#00000205 | A response timeout occurred.  | Check the settings of the communications parameters.  |
| 16#00000206 | There is an error in the transmission path.                                   | <ul style="list-style-type: none"> <li>• Attempt retries.</li> <li>• If this error occurs frequently, check for noise.</li> </ul>   |
| 16#00000301 | A Communications Controller Error occurred.                                   | Refer to the operation manual for the relevant Unit and make suitable corrections.  |
| 16#00000302 | There is an error in the CPU Unit at the remote node.                         | Refer to the manual for the CPU Unit at the remote node and remove the error.   |
| 16#00000303 | There is an error in the relevant Controller and a response is not returned.  | Check the communications status on the network and restart the relevant Controller. If the error still occurs, replace the relevant Controller.                             |
| 16#00000304 | The unit number setting is not correct.                                       | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000401 | The command that was sent is not supported.                                   | Set the command array correctly.  |

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000402 | The Unit model or version is not supported.   | Check the Unit model and version.   |
| 16#00000501 | The remote address setting is wrong.  | Set the destination address in the routing tables.  |
| 16#00000502 | Routing tables are not registered.  | Set the source node, destination node, and relay nodes in the routing tables.   |
| 16#00000503 | There is an error in the routing tables.  | Correct the settings in the routing tables.   |
| 16#00000504 | There are too many relay points.  | Restructure the network or correct the routing tables so that commands are used within a three-layer range.   |
| 16#00001001 | The command is too long.  | Set the command array correctly.  |
| 16#00001002 | The command is too short.   | Set the command array correctly.  |
| 16#00001003 | The number of write elements that is specified in the command does not agree with the number of write data. | Specify the same number of write elements and write data.   |
| 16#00001004 | The command format is incorrect.  | Set the command array correctly.  |
| 16#00001005 | There is an error in the header.  | Correct the settings in the routing tables.   |
| 16#00001101 | The area type does not exist.   | Refer to the command variables and parameter type codes and set the relevant codes.   |
| 16#00001102 | An access size is wrong.  | Correctly set the variable and parameter access sizes.  |
| 16#00001103 | An out-of-range address was specified.  | Specify an address that is within the process range.  |
| 16#00001104 | The address range was exceeded.   | <ul style="list-style-type: none"> <li>Specify an address that is within the process range.</li> <li>Correct the settings in the data link table.</li> </ul>                                      |
| 16#00001106 | A communications sequence number that is not registered was specified.                                      | Correct the communications sequence number or add the sequence with the CX-Protocol.  |
| 16#00001109 | An interrelationship error occurred.  | <ul style="list-style-type: none"> <li>Correct the size relationships in the command data.</li> <li>Correct the settings in the data link table.</li> </ul>                                       |
| 16#0000110A | Data is redundant.  | <ul style="list-style-type: none"> <li>Cancel the current process or wait for it to be completed before you execute the command.</li> <li>Correct the settings in the data link table.</li> </ul> |
| 16#0000110B | The response is too long.   | Set the number of elements in the command array correctly.  |
| 16#0000110C | This is another parameter error.  | Set the command array correctly.  |
| 16#00002002 | The data is protected.  | Execute the command again after clearing the protection.  |
| 16#00002003 | There is no registered table.   | Set the table correctly.  |
| 16#00002004 | There is no data that matches the search data.  | Set the search data correctly.  |
| 16#00002005 | The relevant program number does not exist.   | Set a valid program number.   |
| 16#00002006 | The relevant file does not exist.   | Set the file name correctly, including the subdirectory names.  |
| 16#00002007 | A verification error occurred.  | <ul style="list-style-type: none"> <li>Check the contents of memory and write the correct data.</li> <li>Check the contents of the file.</li> </ul>   |
| 16#00002101 | Access is not possible because the area is read-only.   | Execute the command again after clearing the write protection.  |
| 16#00002102 | The data is protected or the data link table cannot be written.   | <ul style="list-style-type: none"> <li>Execute the command again after clearing the write protection.</li> <li>Set the system settings in the data link table.</li> </ul>                         |



| Value       | Error  | Correction  |
|-------------|--|---|
| 16#00002103 | Registration is not possible.  | <ul style="list-style-type: none"> <li>• Create the file after deleting unnecessary files or prepare new file memory.</li> <li>• Execute the command again after closing open files.</li> </ul> |
| 16#00002105 | The relevant program number does not exist.  | Set a valid program number.   |
| 16#00002106 | The relevant file does not exist.  | Set the file name correctly, including the subdirectory names.  |
| 16#00002107 | A file with the same name already exists.  | Execute the command again after changing the name of the file to write.   |
| 16#00002108 | The change is not allowed because it causes an error.  | Correct the settings.   |
| 16#00002201 | The operation was not possible because a protocol macros is already in execution.                | Use an N.C. program input for the Protocol Macro Execution Flag.  |
| 16#00002202 | The operating mode is wrong.   | Check the operating mode.   |
| 16#00002203 | The operating mode is wrong for the instruction (PROGRAM mode).                                  | Check the operating mode of the Controller.   |
| 16#00002204 | The operating mode is wrong for the instruction (DEBUG mode).                                    | Check the operating mode of the Controller.   |
| 16#00002205 | The operating mode is wrong for the instruction (MONITOR mode).                                  | Check the operating mode of the Controller.   |
| 16#00002206 | The operating mode is wrong for the instruction (RUN mode).                                      | Check the operating mode of the Controller.   |
| 16#00002207 | The specified node is not the polling node.  | Confirm which node is the polling node of the network.  |
| 16#00002208 | The operating mode is wrong for the instruction.   | Check step activation status.   |
| 16#00002211 | The Unit is busy.  | Increase the number of retries or review the system so that communications traffic is not concentrated on the relevant Unit.  |
| 16#00002301 | The file device does not exist.  | Insert the media. Or, format the EM.  |
| 16#00002302 | There is no file memory.   | Check the file memory device.   |
| 16#00002303 | There is no built-in clock.  | Check the specifications of the model.  |
| 16#00002401 | A checksum error occurred in the protocol macro data, or the data transfer is not yet completed. | Transfer the protocol macro data from the CX-Protocol again.  |
| 16#00002502 | There is an error in the memory.   | Transfer the correct data to the memory.  |
| 16#00002503 | The registered I/O Unit configuration does not agree with the actual Unit configuration.         | Check the I/O Unit configuration.   |
| 16#00002504 | There are too many local or remote I/O points.   | Set the number of local and remote I/O points correctly.  |
| 16#00002505 | An error occurred in a data transmission between the CPU Unit and a CPU Bus Unit.                | Check the Units and the connecting cables. After removing the error, execute a command to reset the error.  |
| 16#00002506 | The same rack number, unit number, or I/O address is set more than once.                         | Correct the settings so that each number is unique.   |
| 16#00002507 | An error occurred in a data transmission between the CPU Unit and an I/O Unit.                   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |

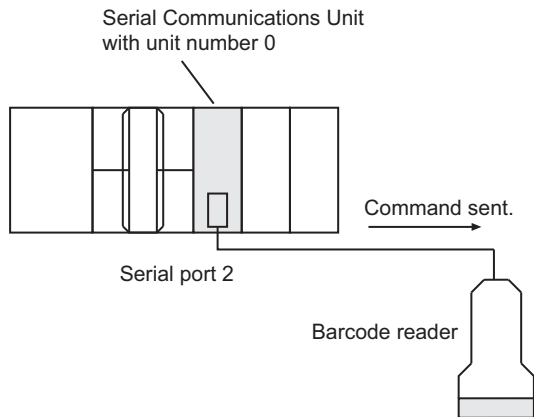
| Value       | Error  | Correction  |
|-------------|--|---|
| 16#00002509 | There is an error in SYSMAC BUS/2 data transmission.   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250A | An error occurred in a CPU Bus Unit data transmission. | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250D | The same channel setting is used more than once.       | Set the I/O channels correctly.   |
| 16#0000250F | There is an error in memory.                           | <ul style="list-style-type: none"> <li>• For internal memory, execute the command again after writing the correct data.</li> <li>• For a Memory Card or EM file memory, execute the expansion memory format command.</li> <li>• If the error cannot be reset with the above corrections, replace the memory.</li> </ul> |
| 16#00002510 | The end station setting is wrong.                      | Set the end station correctly.  |
| 16#00002601 | Protection is already cleared.                         | You do not need to clear protection.  |
| 16#00002602 | The password was wrong.                                | Specify the correct password.   |
| 16#00002604 | The data is protected.                                 | <ul style="list-style-type: none"> <li>• Execute the command again after clearing the write protection.</li> <li>• Wait for the service that is currently in execution to end, or stop the service and execute the command again.</li> </ul>  |
| 16#00002605 | The service is busy.                                   | Wait for the service that is currently in execution to end, or stop the service and execute the command again.  |
| 16#00002606 | The service is stopped.                                | Execute the relevant service as required.   |
| 16#00002607 | You do not have the execution right.                   | <ul style="list-style-type: none"> <li>• Execute the operation from the node that accessed the data link.</li> <li>• If the error still occurs after a restart, replace the Controller.</li> </ul>  |
| 16#00002608 | The environment is not set.                            | Make the necessary settings.  |
| 16#00002609 | The required items are not set.                        | Set the required items.   |
| 16#0000260A | The specified number is already defined.               | Execute the command again after changing the specified number to an action or transition number that is not already registered.   |
| 16#0000260B | The error cannot be reset.                             | Remove the cause of the error and then execute the error reset command.   |
| 16#00003001 | You do not have access rights.                         | Wait for the access to be allowed and then execute the command again.   |
| 16#00004001 | The service was interrupted.                           | Execute the command again after clearing the cause of the service interruption.   |

**Note** In addition to the codes in the above table, the values of bits 6, 7, and 15 in the end code can be TRUE. If the value of bit 6 or 7 is TRUE, there is an error in the CPU Unit at the destination. If the value of bit 15 is TRUE, an error occurred during a network relay.

## Sample Programming

In this sample, a no-protocol command is sent to the barcode reader that is connected to serial port 2 of a CJ-series Serial Communications Unit (unit number 0, device name 'Barcode'). The scene number acquisition command (@READ) is sent.

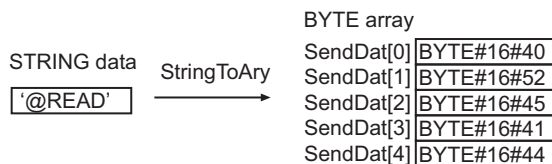
The send data is the contents of the array variable SendDat[]. There is no start code and the end code is 16#OD (CR).



The settings of Serial Communications Unit are given in the following table.

| Item   | Set value                         |
|--|-----------------------------------|
| Port 2: User-specified Setting Inclusion         | User settings                     |
| Port 2: Serial Communications Mode               | No-protocol                       |
| Port 2: Data Length                              | 8 bits                            |
| Port 2: Stop Bits                                | 1 bit                             |
| Port 2: Parity                                   | No                                |
| Port 2: Baud Rate                                | 38,400 bps                        |
| Port 2: No-Protocol End Code                     | D                                 |
| Port 2: No-Protocol Start Code Inclusion Setting | No                                |
| Port 2: No-Protocol End Code Inclusion Setting   | Yes (Specify a desired end code.) |

The text string '@READ' is separated into individual characters and the character codes are stored in the array elements of SendDat[]. Therefore, BYTE#16#40 (@) is stored in SendDat[0], BYTE#16#52(R) is stored in SendData[1], etc. The StringToAry instruction is used to store the character codes.



## Definitions of Global Variables

### ● Global Variables

| Name                            | Data type | AT specification*1                                      | Comment                               |
|---------------------------------|-----------|---|---------------------------------------|
| Barcode_P2_NopSerialSendExecSta | BOOL      | IOBus://rack#0/slot#0/P2_NopSta/P2_NopSerialSendExecSta | SerialSend Instruction Execution Flag |

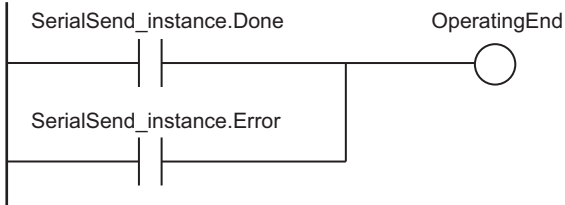
\*1. AT when the Serial Communications Unit is mounted to slot number 0 in rack number 0.

**LD**

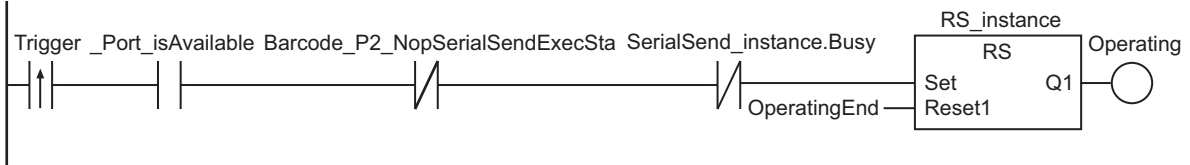
| Internal Variables | Variable            | Data type           | Initial value                        | Comment              |
|--------------------|---------------------|---------------------|--------------------------------------|----------------------|
|                    | OperatingEnd        | BOOL                | FALSE                                | Processing completed |
|                    | Trigger             | BOOL                | FALSE                                | Execution condition  |
|                    | Operating           | BOOL                | FALSE                                | Processing           |
|                    | InPort              | _sPORT              | (UnitNo:=_CBU_No00, PhysicPortNo:=0) | Port settings        |
|                    | SendDat             | ARRAY[0..4] OF BYTE | [5(16#0)]                            | Send data            |
|                    | RS_instance         | RS                  |                                      |                      |
|                    | SerialSend_instance | SerialSend          |                                      |                      |

| External Variables | Variable                        | Data type | Comment   |
|--------------------|---------------------------------|-----------|---|
|                    | _Port_isAvailable               | BOOL      | Network Communications Instruction Enabled Flag |
|                    | Barcode_P2_NopSerialSendExecSta | BOOL      | SerialSend Instruction Execution Flag           |

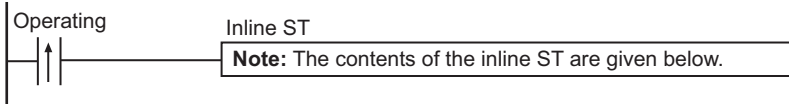
Determine if execution of the SerialSend instruction is completed.



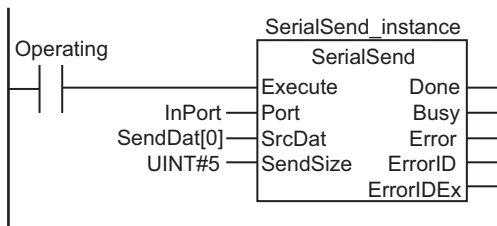
Accept trigger.



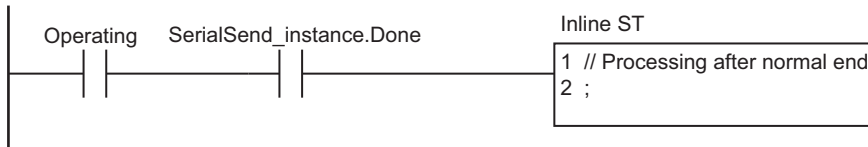
Set communications parameters.



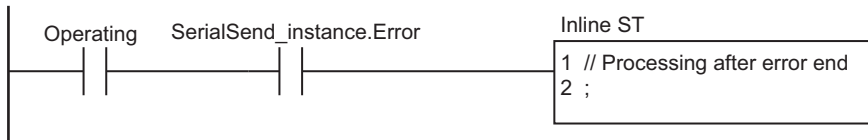
Execute SerialSend instruction.



Processing after normal end



Processing after error end



## ● Contents of Inline ST

```
StringToArray(In:='@READ', AryOut:=SendDat[0]); // Prepare SendDat[].
InPort.UnitNo      :=_CBU_No00;                // Serial Communications Unit with
unit number 0
InPort.PhysicPortNo:=USINT#2;                  // Serial port 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                                 |
|                    | InPort              | _sPORT              | (UnitNo:=_CBU_No00, PhysicPortNo:=0) | Port settings                              |
|                    | SendDat             | ARRAY[0..4] OF BYTE | [5(16#0)]                            | Send data                                  |
|                    | SerialSend_instance | SerialSend          |                                      |  |

| External Variables | Variable                        | Data type | Comment   |
|--------------------|---------------------------------|-----------|---|
|                    | _Port_isAvailable               | BOOL      | Network Communications Instruction Enabled Flag |
|                    | Barcode_P2_NopSerialSendExecSta | BOOL      | SerialSend Instruction Execution Flag           |

```

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
      AND (Barcode_P2_NopSerialSendExecSta=FALSE) AND (SerialSend_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SerialSend instruction.
IF (OperatingStart=TRUE) THEN
    SerialSend_instance(
        Execute:=FALSE,
        SrcDat :=SendDat[0]);
    StringToAry(In:='@READ', AryOut:=SendDat[0]);
    InPort.UnitNo      :=_CBU_No00;           // Serial Communications Unit with unit number 0
    InPort.PhysicPortNo:=USINT#2;           // Serial port 2
    OperatingStart     :=FALSE;
END_IF;

// Execute SerialSend instruction.
IF (Operating=TRUE) THEN
    SerialSend_instance(
        Execute :=TRUE,
        Port    :=InPort,           // Port settings
        SrcDat  :=SendDat[0],       // Send data
        SendSize:=UINT#5);         // Send data size

    IF (SerialSend_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (SerialSend_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;

```

# SerialRcv and SerialRcvNoClear

The SerialRcv and SerialRcvNoClear instructions receive data in No-protocol Mode from a serial port on a Serial Communications Unit.

SerialRcv : Clears the receive buffer after reading the data.

SerialRcvNoClear : Does not clear the receive buffer after reading the data.

| Instruction      | Name  | FB/<br>FUN | Graphic expression | ST expression   |
|------------------|---|------------|--------------------|---|
| SerialRcv        | SCU Receive Serial                              | FB         |                    | SerialRcv_instance(Execute, Port, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);        |
| SerialRcvNoClear | SCU Receive Serial without Receive Buffer Clear | FB         |                    | SerialRcvNoClear_instance(Execute, Port, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize); |



## Precautions for Correct Use

You cannot use these instructions with NX-series CPU Units.



## Version Information

A CPU Unit with unit version 1.03 or later, Sysmac Studio version 1.04 or higher, and a Serial Communications Unit with unit version 2.1 or later are required to use the SerialRcvNoClear instruction.

## Variables

|                  | Meaning                   | I/O    | Description   | Valid range           | Unit  | Default |
|------------------|---------------------------|--------|---|-----------------------|-------|---------|
| Port             | Destination port          | Input  | Destination port  | ---                   | ---   | ---     |
| Size             | Receive data size         |        | Size of receive data stored in DstDat[]                   | 0 to 256              | Bytes | 1       |
| DstDat[] (array) | Receive data array        | In-out | Receive data array  | Depends on data type. | ---   | ---     |
| RcvSize          | Receive data storage size | Output | Size of receive data that was actually stored in DstDat[] | 0 to 256              | Bytes | ---     |

|                  | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------------------|---|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                  | BOOL  | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| Port             | Refer to <i>Data Types of Destination Port Port</i> on page 2-1305 for details on the structure <code>_sPORT</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Size             |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| DstDat[] (array) |   | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RcvSize          |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

First, data which is received in No-protocol Mode from the serial port specified with *Port* is stored in the receive buffer in the Serial Communications Unit.

The `SerialRcv` and `SerialRcvNoClear` instructions transfer data of the size specified with *Size* (receive data size ) from the receive buffer to `DstDat[]` (receive data array).

After the data is transferred, the number of array elements that was actually stored in `DstDat[]` is assigned to *RcvSize* (receive data storage size).

If the size of data in the receive buffer is smaller than *Size*, all of the data in the receive buffer is transferred to `DstDat[]`. The size of the data which was actually transferred and stored in `DstDat[]` is assigned to *RcvSize*.

When the Data In the Receive Buffer Is the Same or Larger than *Size*



When the Data In the Receive Buffer Is Less than *Size*



## Start Code and End Code in Receive Data

Device variables are used in the user program to recognize the start code and end code in the receive data. The start and end codes are deleted from the receive data before it is stored in `DstDat[]`.

| Code to attach       | Device variable for port 1 | Value                       |
|----------------------|----------------------------|-----------------------------|
| Specified start code | P1_NopStartCodeYNCfg       | TRUE                        |
|                      | P1_NopStartCodeCfg         | Start code (16#00 to 16#FF) |



| Code to attach     | Device variable for port 1 | Value                     |
|--------------------|----------------------------|---------------------------|
| Specified end code | P1_NopEndCodeYNCfg         | TRUE                      |
|                    | P1_NopCRLFCfg              | FALSE                     |
|                    | P1_NopEndCodeCfg           | End code (16#00 to 16#FF) |
| CR+LF as end code  | P1_NopEndCodeYNCfg         | TRUE                      |
|                    | P1_NopCRLFCfg              | TRUE                      |

If you add start and end codes, the maximum number of bytes to receive is 259 (1-byte start code, 2-byte end code (for CR+LF specification), and 256 bytes of send data).

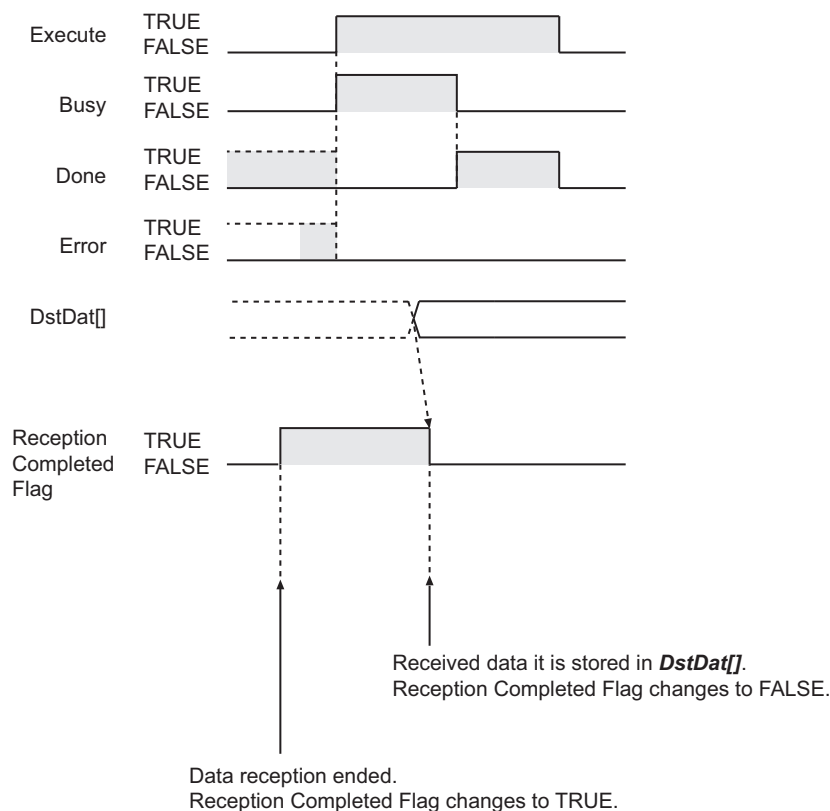
## Data Types of Destination Port *Port*

The data type of destination port *Port* is the structure `_sPORT`. The specifications are as follows:

| Name         | Meaning            | Description                                      | Data type             | Valid range  | Unit | Default                |
|--------------|--------------------|--|-----------------------|--|------|------------------------|
| Port         | Destination port   | Destination port                                 | <code>_sPORT</code>   | ---  | ---  | ---                    |
| UnitNo       | Unit number        | Unit number of Serial Communications Unit        | <code>_eUnitNo</code> | <code>_CBU_No00</code><br>to<br><code>_CBU_No15</code> | ---  | <code>_CBU_No00</code> |
| PhysicPortNo | Serial port number | Serial port number on Serial Communications Unit | USINT                 | 1 or 2   | ---  | 1                      |

## Timing Chart

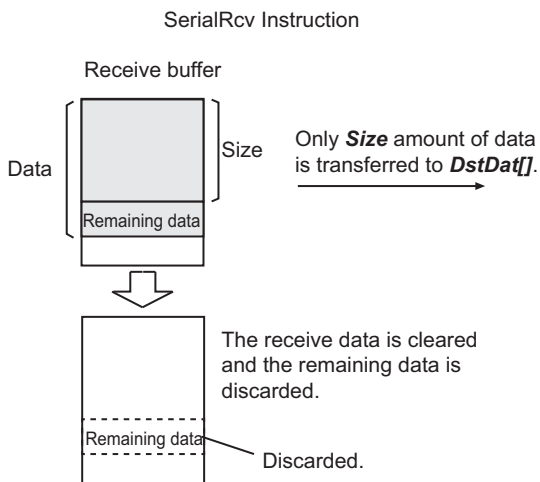
The following figure shows a timing chart.



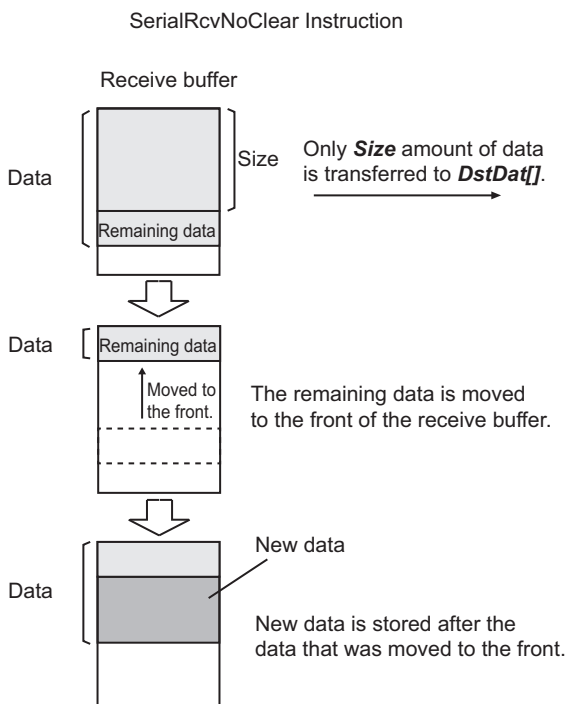
## Difference between the SerialRcv and SerialRcvNoClear Instructions

The SerialRcv and SerialRcvNoClear instructions are different in whether the receive buffer is cleared after data is transferred from the receive buffer to *DstDat[]*.

The SerialRcv instruction clears the receive buffer after it transfers the data. Therefore, if the data in the receive buffer is larger than *Size*, the excess data will be left in the buffer and then discarded after the transfer.



The SerialRcvNoClear instruction clears only the data that was transferred after the data transfer. The data that remains in the receive buffer is moved to the front of the receive buffer. If new data enters the receive buffer after that, it is stored after the data that was moved to the front of the buffer.



## Related System-defined Variables

| Name               | Meaning   | Data type | Description   |
|--------------------|---|-----------|---|
| _Port_numUsingPort | Number of Used Ports                            | USINT     | This is the number of ports that are currently used.          |
| _Port_isAvailable  | Network Communications Instruction Enabled Flag | BOOL      | TRUE: A port is available.<br>FALSE: A port is not available. |

## Related Semi-user-defined Variables

| Name                               | Meaning                                  | Data type | Description  |
|------------------------------------|--|-----------|--|
| P#_NopRcvOvfSta <sup>*1</sup>      | Reception Overflow Flag                  | BOOL      | TRUE: The Unit received more than the specified amount of data.<br>(i.e., data was received after the Reception Completed Flag changed to TRUE.)<br>FALSE: The Unit did not receive more than the specified number of bytes. |
| P#_NopRcvCompleteSta <sup>*1</sup> | Reception Completed Flag                 | BOOL      | TRUE: Reception was completed.<br>FALSE: No data received or currently receiving data.   |
| P#_NopRcvCntSta <sup>*1</sup>      | Reception Counter                        | UINT      | 16#0000 to 16#0100: Number of bytes of receive data  |
| P#_NopStartCodeYNCfg <sup>*1</sup> | No-protocol Start Code Enable            | BOOL      | TRUE: Start code<br>FALSE: None  |
| P#_NopEndCodeYNCfg <sup>*1</sup>   | No-protocol End Code Enable              | BOOL      | TRUE: End code<br>FALSE: None  |
| P#_NopCRLFcfg <sup>*1</sup>        | No-protocol CR LF Specification          | BOOL      | TRUE: CR+LF<br>FALSE: No CR+LF   |
| P#_NopRcvDatSzCfg <sup>*1</sup>    | Number of No-protocol Receive Data Bytes | USINT     | 16#01 to 16#FF: 1 to 255 bytes<br>16#00: 256 bytes   |
| P#_NopStartCodeCfg <sup>*1</sup>   | No-protocol Start Code                   | USINT     | 16#00 to 16#FF   |
| P#_NopEndCodeCfg <sup>*1</sup>     | No-protocol End Code                     | USINT     | 16#00 to 16#FF   |
| P#_TransErr <sup>*1</sup>          | Transmission Error                       | BOOL      | TRUE: Error occurred.<br>FALSE: No error occurred.   |
| P#_OverRunErr <sup>*1</sup>        | Overrun Error                            | BOOL      | TRUE: Error occurred.<br>FALSE: No error occurred.   |

\*1. "#" denotes the port number on the Serial Communications Unit.

## Additional Information

- *P#\_NopRcvCompleteSta* (Reception Completed Flag) changes to TRUE when the following occur.
  - a) The size of received data has reached the size specified with *P#\_NopRcvDatSzCfg* (Number of No-protocol Receive Data Bytes).
  - b) The specified end code is received.
  - c) A total of 256 bytes of data is received.
- *P#\_NopRcvOvfSta* (Reception Overflow Flag) changes to TRUE when the following occur.
  - a) While *P#\_NopRcvCompleteSta* (Reception Completed Flag) is TRUE, further data is received before the SerialRcv or SerialRcvNoClear instruction is executed.

- b) The size of received data has exceeded the size specified with *P#\_NopRcvDatSzCfg* (Number of No-protocol Receive Data Bytes).
- Refer to the *CJ-series Serial Communications Units Operation Manual for NJ-series CPU Unit (Cat. No. W494)* for details on no-protocol communications.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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 these instructions while *P#\_NopRcvCompleteSta* (Reception Completed Flag) is TRUE.
- When data is received, always execute this instruction to transfer the data in the receive buffer to *DstDat[]*. No more data is received until the previous data is transferred.
- Once the size of received data reaches 259 bytes, no more data is received automatically. If further data is received before the *SerialRcv* or *SerialRcvNoClear* instruction is executed, *P#\_OverRunErr* (Overrun Error) changes to TRUE.
- When a start code or end code is attached, do not include it in the value of *Size*.
- You can use this instruction only for a Serial Communications Unit's serial port that is set to No-protocol Mode.
- If the value of *Size* is 0, the data in the receive buffer is not transferred to *DstDat[]*. In this case, *P#\_NopRcvCompleteSta* (Reception Completed Flag) and *P#\_NopRcvOvfSta* (Reception Overflow Flag) will change to FALSE. And the value of *P#\_NopRcvCntSta* (Reception Counter) will be 0.
- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *\_Port\_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing may not be possible.
- The receive buffer in the Serial Communications Unit is cleared when the *SerialRcv* instruction is executed. Therefore, you cannot separate the data in the receive buffer to transfer it to *DstDat[]*.
- As for the *SerialRcv* instruction, if the size of received data exceeds the value of *Size*, the excess data will be discarded when another *SerialRcv* instruction is executed.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The serial communications mode is not set to No-protocol Mode when the instruction is executed.
  - b) The value of *\_Port\_isAvailable* is FALSE.
  - c) The value of *Port.UnitNo* or *Port.PhysicPortNo* is outside the valid range.
  - d) There is no CJ-series Serial Communications Unit with the specified unit number.
  - e) The value of *Size* is outside the valid range.
  - f) The value of *Size* exceeds the size of *DstDat[]*.
  - g) Communications fail.
  - h) The instruction is executed during a Unit restart.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000001 | The communications service was interrupted.                                   | <ul style="list-style-type: none"> <li>• Check the data link execution status.</li> <li>• Check the capacity of the transfer destination area on the third node.</li> </ul> |
| 16#00000101 | The local node is not part of the network.                                    | Make the local node part of the network.  |
| 16#00000102 | A token timeout occurred.   | Set the local node address to be within the maximum node address.   |
| 16#00000103 | The retry count was exceeded.   | Perform inter-node tests. If any error is found, check the operating environment.   |
| 16#00000104 | The allowable number of send frames was exceeded.                             | Check the status of events in the network and reduce the number of events in each task period. Or, increase the number of allowable send frames.                            |
| 16#00000105 | The IP address of the local node is out of range.                             | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000106 | The IP address of the local node is also used by another node in the network. | Change one of the node addresses that are duplicated.   |
| 16#00000201 | The remote node is not part of the network.                                   | Make the remote node part of the network.   |
| 16#00000202 | A Unit with the specified unit address does not exist at the destination.     | Correctly set the unit address for the destination network address.   |
| 16#00000203 | The third node is not part of the network.                                    | <ul style="list-style-type: none"> <li>• Check the address of the Unit that is the third node.</li> <li>• Specify only one node for the third node.</li> </ul>              |
| 16#00000204 | The remote node is busy.  | Increase the number of retries or correct the system so that communications traffic is not concentrated on the remote node.   |
| 16#00000205 | A response timeout occurred.  | Check the settings of the communications parameters.  |
| 16#00000206 | There is an error in the transmission path.                                   | <ul style="list-style-type: none"> <li>• Attempt retries.</li> <li>• If this error occurs frequently, check for noise.</li> </ul>   |
| 16#00000301 | A Communications Controller Error occurred.                                   | Refer to the operation manual for the relevant Unit and make suitable corrections.  |
| 16#00000302 | There is an error in the CPU Unit at the remote node.                         | Refer to the manual for the CPU Unit at the remote node and remove the error.   |
| 16#00000303 | There is an error in the relevant Controller and a response is not returned.  | Check the communications status on the network and restart the relevant Controller. If the error still occurs, replace the relevant Controller.                             |
| 16#00000304 | The unit number setting is not correct.                                       | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000401 | The command that was sent is not supported.                                   | Set the command array correctly.  |
| 16#00000402 | The Unit model or version is not supported.                                   | Check the Unit model and version.   |
| 16#00000501 | The remote address setting is wrong.  | Set the destination address in the routing tables.  |
| 16#00000502 | Routing tables are not registered.  | Set the source node, destination node, and relay nodes in the routing tables.   |
| 16#00000503 | There is an error in the routing tables.                                      | Correct the settings in the routing tables.   |
| 16#00000504 | There are too many relay points.  | Restructure the network or correct the routing tables so that commands are used within a three-layer range.   |
| 16#00001001 | The command is too long.  | Set the command array correctly.  |

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00001002 | The command is too short.   | Set the command array correctly.  |
| 16#00001003 | The number of write elements that is specified in the command does not agree with the number of write data. | Specify the same number of write elements and write data.   |
| 16#00001004 | The command format is incorrect.  | Set the command array correctly.  |
| 16#00001005 | There is an error in the header.  | Correct the settings in the routing tables.   |
| 16#00001101 | The area type does not exist.   | Refer to the command variables and parameter type codes and set the relevant codes.   |
| 16#00001102 | An access size is wrong.  | Correctly set the variable and parameter access sizes.  |
| 16#00001103 | An out-of-range address was specified.  | Specify an address that is within the process range.  |
| 16#00001104 | The address range was exceeded.   | <ul style="list-style-type: none"> <li>Specify an address that is within the process range.</li> <li>Correct the settings in the data link table.</li> </ul>                                      |
| 16#00001106 | A communications sequence number that is not registered was specified.                                      | Correct the communications sequence number or add the sequence with the CX-Protocol.  |
| 16#00001109 | An interrelationship error occurred.  | <ul style="list-style-type: none"> <li>Correct the size relationships in the command data.</li> <li>Correct the settings in the data link table.</li> </ul>                                       |
| 16#0000110A | Data is redundant.  | <ul style="list-style-type: none"> <li>Cancel the current process or wait for it to be completed before you execute the command.</li> <li>Correct the settings in the data link table.</li> </ul> |
| 16#0000110B | The response is too long.   | Set the number of elements in the command array correctly.  |
| 16#0000110C | This is another parameter error.  | Set the command array correctly.  |
| 16#00002002 | The data is protected.  | Execute the command again after clearing the protection.  |
| 16#00002003 | There is no registered table.   | Set the table correctly.  |
| 16#00002004 | There is no data that matches the search data.  | Set the search data correctly.  |
| 16#00002005 | The relevant program number does not exist.   | Set a valid program number.   |
| 16#00002006 | The relevant file does not exist.   | Set the file name correctly, including the subdirectory names.  |
| 16#00002007 | A verification error occurred.  | <ul style="list-style-type: none"> <li>Check the contents of memory and write the correct data.</li> <li>Check the contents of the file.</li> </ul>   |
| 16#00002101 | Access is not possible because the area is read-only.   | Execute the command again after clearing the write protection.  |
| 16#00002102 | The data is protected or the data link table cannot be written.   | <ul style="list-style-type: none"> <li>Execute the command again after clearing the write protection.</li> <li>Set the system settings in the data link table.</li> </ul>                         |
| 16#00002103 | Registration is not possible.   | <ul style="list-style-type: none"> <li>Create the file after deleting unnecessary files or prepare new file memory.</li> <li>Execute the command again after closing open files.</li> </ul>       |
| 16#00002105 | The relevant program number does not exist.   | Set a valid program number.   |
| 16#00002106 | The relevant file does not exist.   | Set the file name correctly, including the subdirectory names.  |
| 16#00002107 | A file with the same name already exists.   | Execute the command again after changing the name of the file to write.   |

| Value       | Error  | Correction   |
|-------------|--|--|
| 16#00002108 | The change is not allowed because it causes an error.  | Correct the settings.  |
| 16#00002201 | The operation was not possible because a protocol macros is already in execution.                | Use an N.C. program input for the Protocol Macro Execution Flag.   |
| 16#00002202 | The operating mode is wrong.   | Check the operating mode.  |
| 16#00002203 | The operating mode is wrong for the instruction (PROGRAM mode).                                  | Check the operating mode of the Controller.  |
| 16#00002204 | The operating mode is wrong for the instruction (DEBUG mode).                                    | Check the operating mode of the Controller.  |
| 16#00002205 | The operating mode is wrong for the instruction (MONITOR mode).                                  | Check the operating mode of the Controller.  |
| 16#00002206 | The operating mode is wrong for the instruction (RUN mode).                                      | Check the operating mode of the Controller.  |
| 16#00002207 | The specified node is not the polling node.  | Confirm which node is the polling node of the network.   |
| 16#00002208 | The operating mode is wrong for the instruction.   | Check step activation status.  |
| 16#00002211 | The Unit is busy.  | Increase the number of retries or review the system so that communications traffic is not concentrated on the relevant Unit. |
| 16#00002301 | The file device does not exist.  | Insert the media. Or, format the EM.   |
| 16#00002302 | There is no file memory.   | Check the file memory device.  |
| 16#00002303 | There is no built-in clock.  | Check the specifications of the model.   |
| 16#00002401 | A checksum error occurred in the protocol macro data, or the data transfer is not yet completed. | Transfer the protocol macro data from the CX-Protocol again.   |
| 16#00002502 | There is an error in the memory.   | Transfer the correct data to the memory.   |
| 16#00002503 | The registered I/O Unit configuration does not agree with the actual Unit configuration.         | Check the I/O Unit configuration.  |
| 16#00002504 | There are too many local or remote I/O points.   | Set the number of local and remote I/O points correctly.   |
| 16#00002505 | An error occurred in a data transmission between the CPU Unit and a CPU Bus Unit.                | Check the Units and the connecting cables. After removing the error, execute a command to reset the error.                   |
| 16#00002506 | The same rack number, unit number, or I/O address is set more than once.                         | Correct the settings so that each number is unique.  |
| 16#00002507 | An error occurred in a data transmission between the CPU Unit and an I/O Unit.                   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.                       |
| 16#00002509 | There is an error in SYSMAC BUS/2 data transmission.   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.                       |
| 16#0000250A | An error occurred in a CPU Bus Unit data transmission.   | Check the Units and connecting cables. After removing the error, execute a command to reset the error.                       |
| 16#0000250D | The same channel setting is used more than once.   | Set the I/O channels correctly.  |



| Value       | Error                                    | Correction  |
|-------------|--|---|
| 16#0000250F | There is an error in memory.             | <ul style="list-style-type: none"> <li>For internal memory, execute the command again after writing the correct data.</li> <li>For a Memory Card or EM file memory, execute the expansion memory format command.</li> <li>If the error cannot be reset with the above corrections, replace the memory.</li> </ul> |
| 16#00002510 | The end station setting is wrong.        | Set the end station correctly.  |
| 16#00002601 | Protection is already cleared.           | You do not need to clear protection.  |
| 16#00002602 | The password was wrong.                  | Specify the correct password.   |
| 16#00002604 | The data is protected.                   | <ul style="list-style-type: none"> <li>Execute the command again after clearing the write protection.</li> <li>Wait for the service that is currently in execution to end, or stop the service and execute the command again.</li> </ul>  |
| 16#00002605 | The service is busy.                     | Wait for the service that is currently in execution to end, or stop the service and execute the command again.  |
| 16#00002606 | The service is stopped.                  | Execute the relevant service as required.   |
| 16#00002607 | You do not have the execution right.     | <ul style="list-style-type: none"> <li>Execute the operation from the node that accessed the data link.</li> <li>If the error still occurs after a restart, replace the Controller.</li> </ul>  |
| 16#00002608 | The environment is not set.              | Make the necessary settings.  |
| 16#00002609 | The required items are not set.          | Set the required items.   |
| 16#0000260A | The specified number is already defined. | Execute the command again after changing the specified number to an action or transition number that is not already registered.   |
| 16#0000260B | The error cannot be reset.               | Remove the cause of the error and then execute the error reset command.   |
| 16#00003001 | You do not have access rights.           | Wait for the access to be allowed and then execute the command again.   |
| 16#00004001 | The service was interrupted.             | Execute the command again after clearing the cause of the service interruption.   |

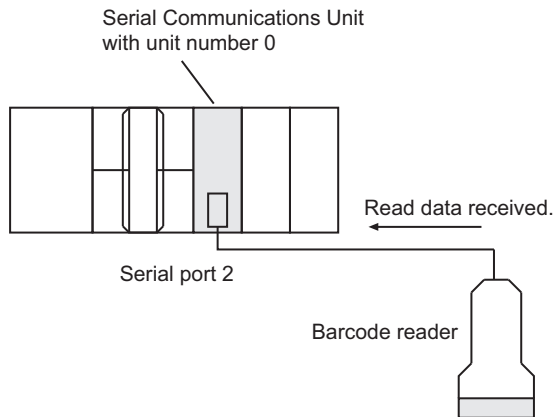
**Note** In addition to the codes in the above table, the values of bits 6, 7, and 15 in the end code can be TRUE. If the value of bit 6 or 7 is TRUE, there is an error in the CPU Unit at the destination. If the value of bit 15 is TRUE, an error occurred during a network relay.

## Sample Programming

In this sample, data that was read by the barcode reader that is connected to serial port 2 of a CJ-series Serial Communications Unit (unit number 0, device name 'Barcode') is received.

The receive data is stored in array variable RecvDat[]. There is no start code and the end code is 16#OD (CR).

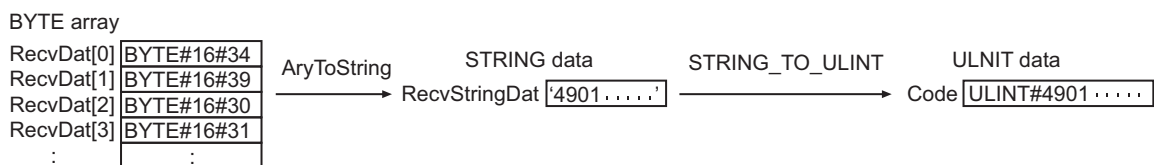




The settings of Serial Communications Unit are given in the following table.

| Item   | Set value                         |
|--|-----------------------------------|
| Port 2: User-specified Setting Inclusion         | User settings                     |
| Port 2: Serial Communications Mode               | No-protocol                       |
| Port 2: Data Length                              | 8 bits                            |
| Port 2: Stop Bits                                | 1 bit                             |
| Port 2: Parity                                   | No                                |
| Port 2: Baud Rate                                | 38,400 bps                        |
| Port 2: No-Protocol End Code                     | D                                 |
| Port 2: No-Protocol Start Code Inclusion Setting | No                                |
| Port 2: No-Protocol End Code Inclusion Setting   | Yes (Specify a desired end code.) |

The number from the barcode reader is separated into individual characters and bit strings for the character codes are stored in `RecvDat[]`. One element of the `RecvDat[]` array corresponds to one character from the barcode. First, the `AryToString` instruction is used to convert the data to a text string, `RecvStringDat`. Next, the `STRING_TO_ULINT` instruction is used to convert the data to an ULINT integer, `Code`.



## Definitions of Global Variables

### ● Global Variables

| Name                         | Data type | AT specification*1                                   | Comment                  |
|------------------------------|-----------|--|--------------------------|
| Barcode_P2_NopRcvCompleteSta | BOOL      | IOBus://rack#0/slot#0/P2_NopSta/P2_NopRcvCompleteSta | Reception Completed Flag |

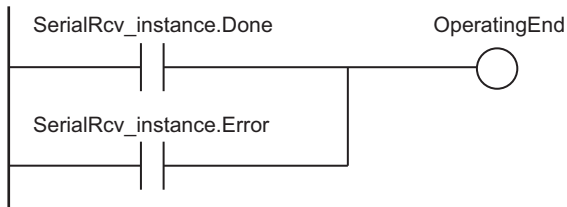
\*1. AT when the Serial Communications Unit is mounted to slot number 0 in rack number 0.

# LD

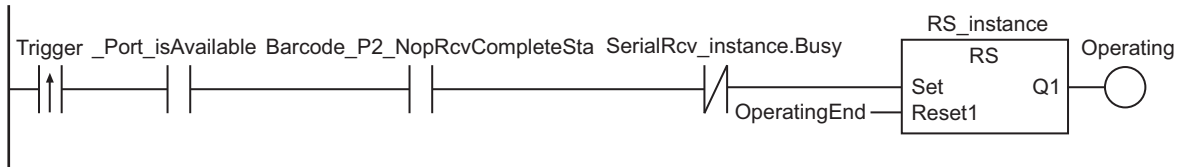
| Internal Variables | Variable           | Data type            | Initial value                        | Comment              |
|--------------------|--------------------|----------------------|--------------------------------------|----------------------|
|                    | OperatingEnd       | BOOL                 | FALSE                                | Processing completed |
|                    | Trigger            | BOOL                 | FALSE                                | Execution condition  |
|                    | Operating          | BOOL                 | FALSE                                | Processing           |
|                    | InPort             | _sPORT               | (UnitNo:=_CBU_No00, PhysicPortNo:=0) | Port settings        |
|                    | RecvDat            | ARRAY[0..12] OF BYTE | [13(16#0)]                           | Receive data         |
|                    | RecvSize           | UINT                 | 0                                    | Receive data size    |
|                    | RecvStringDat      | STRING[255]          | "                                    | Barcode text string  |
|                    | Code               | ULINT                | 0                                    | Barcode integer      |
|                    | RS_instance        | RS                   |                                      |                      |
|                    | SerialRcv_instance | SerialRcv            |                                      |                      |

| External Variables | Variable                     | Data type | Comment   |
|--------------------|------------------------------|-----------|---|
|                    | _Port_isAvailable            | BOOL      | Network Communications Instruction Enabled Flag |
|                    | Barcode_P2_NopRcvCompleteSta | BOOL      | Reception Completed Flag                        |

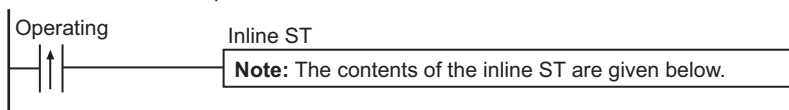
Determine if execution of the SerialRcv instruction is completed.



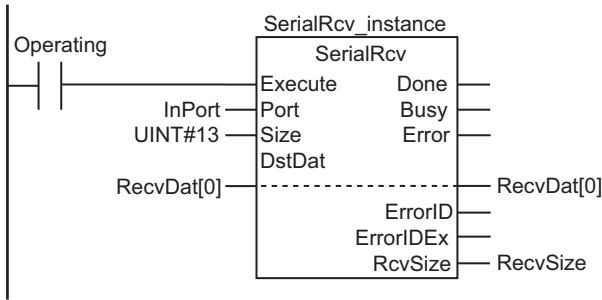
Accept trigger.



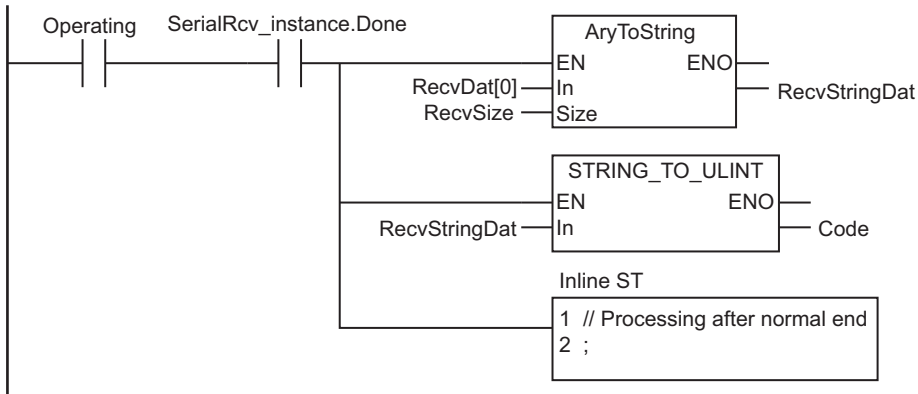
Set communications parameters.



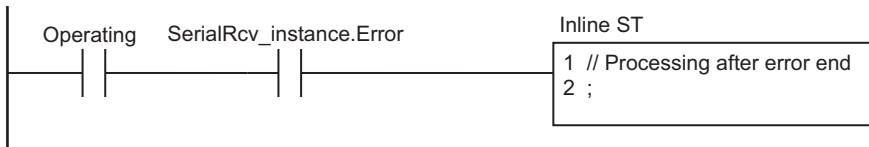
Execute SerialRcv instruction.



Processing after normal end



Processing after error end



● Contents of Inline ST

```
InPort.UnitNo      :=_CBU_No00;    // Serial Communications Unit with unit number 0
InPort.PhysicPortNo:=USINT#2;     // Serial port 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                                 |
|                    | InPort         | _sPORT               | (UnitNo:=_CBU_No00, PhysicPortNo:=0) | Port settings                              |
|                    | RecvDat        | ARRAY[0..12] OF BYTE | [13(16#0)]                           | Receive data                               |

| Internal Variables | Variable           | Data type   | Initial value | Comment             |
|--------------------|--------------------|-------------|---------------|---------------------|
|                    | RecvSize           | UINT        | 0             | Receive data size   |
|                    | RecvStringDat      | STRING[255] | "             | Barcode text string |
|                    | Code               | ULINT       | 0             | Barcode integer     |
|                    | SerialRcv_instance | SerialRcv   |               |                     |

| External Variables | Variable                     | Data type | Comment   |
|--------------------|------------------------------|-----------|---|
|                    | _Port_isAvailable            | BOOL      | Network Communications Instruction Enabled Flag |
|                    | Barcode_P2_NopRcvCompleteSta | BOOL      | Reception Completed Flag                        |

```

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
      AND (Barcode_P2_NopRcvCompleteSta=TRUE) AND (SerialRcv_instance.Busy=FALSE) )
  THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
  END_IF;
  LastTrigger:=Trigger;

// Set communications parameters and initialize SerialRcv instruction.
IF (OperatingStart=TRUE) THEN
  SerialRcv_instance(
    Execute:=FALSE,      // Initialize instance.
    Port   :=InPort,    // Port settings
    Size   :=UINT#13,   // Receive data size
    DstDat :=RecvDat[0], // Receive data
    RcvSize =>RecvSize); // Data size that was actually received
  InPort.UnitNo      :=_CBU_No00; // Serial Communications Unit with unit number
  0
  InPort.PhysicPortNo:=USINT#2;  // Serial port 2
  OperatingStart     :=FALSE;
  END_IF;

// Execute SerialRcv instruction.
IF (Operating=TRUE) THEN
  SerialRcv_instance(
    Execute:=TRUE,
    Port   :=InPort,
    Size   :=UINT#13,
    DstDat :=RecvDat[0],
    RcvSize =>RecvSize);

  IF (SerialRcv_instance.Done=TRUE) THEN

```

```
        // Processing after normal end
        RecvStringDat:=AryToString(In:=RecvDat[0], Size:=RecvSize); // Convert cha
racter codes to a text string.
        Code          :=STRING_TO_ULINT(RecvStringDat);           // Convert tex
t string to an integer.
        Operating     :=FALSE;
    END_IF;
    IF (SerialRcv_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

# SendCmd

The SendCmd instruction uses a serial gateway and sends a command to a Serial Communications Unit.

Or, it sends an explicit command to a DeviceNet Unit or CompoNet Master Unit.

| Instruction | Name         | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|--------------|------------|--------------------|--|
| SendCmd     | Send Command | FB         |                    | SendCmd_instance(Execute, DstNetAdr, CommPort, CmdDat, CmdSize, RespDat, Option, Done, Busy, Error, ErrorID, ErrorIDEx); |



## Precautions for Correct Use

You cannot use this instruction with NX-series CPU Units to which CJ-series configuration Units cannot be connected.

## Variables

|                      | Meaning                     | I/O    | Description                                  | Valid range              | Unit  | Default |
|----------------------|-----------------------------|--------|--|--------------------------|-------|---------|
| DstNetAdr            | Destination network address | Input  | Destination network address                  | ---                      | ---   | ---     |
| CommPort             | Destination serial port     |        | Destination serial port                      | _NONE                    |       | _NONE   |
| CmdDat[]<br>(array)  | Command array               |        | Command to send                              | Depends on data type.    |       | *1      |
| CmdSize              | Command data size           |        | Command data size                            | 0 to max. data length *2 | Bytes | 2       |
| Option               | Response                    |        | Response monitoring and retry specifications | ---                      | ---   | ---     |
| RespDat[]<br>(array) | Response storage array      | In-out | Array to store response                      | Depends on data type.    | ---   | ---     |

\*1. If you omit an input parameter, the default value is not applied. A building error will occur.

\*2. This may vary depending on the network type.

|           | Boo lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, dates, and text strings |      |      |     |    |        |
|-----------|--|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|--------------|------|---|------|------|-----|----|--------|
|           | BOOL   | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT         | REAL | LREAL                                     | TIME | DATE | TOD | DT | STRING |
| DstNetAdr | Refer to <i>Function</i> on page 2-1319 for details on the structure <code>_sDNET_ADR</code> . |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |
| CommPort  | For enumeration <code>_ePORT</code> enumerator, refer to <i>Function</i> on page 2-1319.       |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

|                      | Boo<br>lean  | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|----------------------|--|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|                      | BOOL   | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT                 | REAL | LREAL  | TIME | DATE | TOD | DT | STRING |
| CmdDat[]<br>(array)  |  | OK          |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| CmdSize              |  |             |      |       |       |          | OK   |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Option               | Refer to <i>Function</i> on page 2-1319 for details on the structure <code>_sRESPONSE</code> . |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| RespDat[]<br>(array) |  | OK          |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

## Function

The `SendCmd` instruction sends the contents of command array `CmdDat[]` to the destination specified with destination network address `DstNetAdr` and destination serial port `CommPort`.

The command data size `CmdSize` specifies how many elements of `CmdDat[]` contain the command. The response that is returned is stored in response storage array `RespDat[]`.

The data type of `DstNetAdr` is structure `_sDNET_ADR`. The specifications are as follows:

| Name                   | Meaning                     | Description                 | Data type               | Valid range           | Unit | Default |
|------------------------|-----------------------------|-----------------------------|-------------------------|-----------------------|------|---------|
| <code>DstNetAdr</code> | Destination network address | Destination network address | <code>_sDNET_ADR</code> | ---                   | ---  | ---     |
| <code>NetNo</code>     | Network address             | Network address             | USINT                   | 0 to 127              | ---  | 0       |
| <code>NodeNo</code>    | Node address                | Node address                | USINT                   | Depends on data type. |      |         |
| <code>UnitNo</code>    | Unit address                | Unit address                | BYTE                    | Depends on data type. |      |         |

The data type of `CommPort` is enumerated type `_ePORT`.

The meanings of the enumerators of enumerated type `_ePORT` are as follows:

| Enumerators        | Meaning   |
|--------------------|---|
| <code>_NONE</code> | The destination is not a serial port in Host Link Mode. |

The data type of `Option` is structure `_sRESPONSE`. The specifications are as follows:

| Name                   | Meaning      | Description   | Data type               | Valid range           | Unit      | Default       |
|------------------------|--------------|---|-------------------------|-----------------------|-----------|---------------|
| <code>Option</code>    | Response     | Response monitoring and retry specifications                    | <code>_sRESPONSE</code> | ---                   | ---       | ---           |
| <code>isNonResp</code> | No response  | TRUE: Response is not required.<br>FALSE: Response is required. | BOOL                    | Depends on data type. | ---       | FALSE         |
| <code>TimeOut</code>   | Timeout time | Timeout time<br>0: 2.0 s  | UINT                    |                       | 0.1 s     | 20<br>(2.0 s) |
| <code>Retry</code>     | Retry count  | Retry count   | USINT                   | 0 to 15               | Time<br>s | 0             |

If no response is returned within the timeout time *Option.TimeOut* when the value of the Response Not Necessary Flag *Option.isNonResp* is FALSE, the command is retried until the response is returned. The retry count is specified by *Option.Retry*.

The timeout time is *Option.TimeOut* multiplied by 0.1 s. However, if the value of *Option.TimeOut* is 0, the timeout time is 2.0 s. The default value of *Option.TimeOut* is 2.0 s.

## Related System-defined Variables

| Name               | Meaning   | Data type | Description   |
|--------------------|---|-----------|---|
| _Port_numUsingPort | Number of Used Ports                            | USINT     | This is the number of ports that are currently used.          |
| _Port_isAvailable  | Network Communications Instruction Enabled Flag | BOOL      | TRUE: A port is available.<br>FALSE: A port is not available. |

## Additional Information

- Commands or responses may be lost during communications due to noise or other factors. You can increase reliability by setting *Option.Retry* to other values than 0 so that the retry process can be performed when no response is returned.
- Refer to the *SYSMAC CS/CJ/CP/NSJ-series Communications Commands Reference Manual (Cat. No. W342)* for details on how to specify the destination network address. For the FINS routing specifications, the specification is different between CS/CJ-series CPU Units and NX-series CPU Units. Refer to *Difference Between CS/CJ-series and NX-series in FINS Routing* in the *NX-series CPU Unit FINS Function User's Manual (Cat. No. W596)* for information on the differences in routing specifications.
- To specify a serial port with the serial gateway function, specify the unit address of the serial port for *DstNetAdr.UnitNo*. The unit addresses of the ports on Serial Communications Units are as follows:
  - a) Port 1  
Unit address = BYTE#16#80 + BYTE#16#04 × unit number (hex)  
Example for Unit Number 1  
BYTE#16#80 + BYTE#16#04 × 1 = BYTE#16#84
  - b) Port 2  
Unit address = BYTE#16#81 + BYTE#16#04 × unit number (hex)  
Example for Unit Number 2  
BYTE#16#81 + BYTE#16#04 × 2 = BYTE#16#89

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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*.
- The command is not sent if the value of *CmdSize* is 0. In this case, the value of *Done* changes to TRUE at instruction execution.



- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *\_Port\_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing may not be possible.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The value of *\_Port\_isAvailable* is FALSE.
  - b) The value of *CommPort* is outside the valid range.
  - c) The value of a member of *DstNetAdr* is outside the valid range.
  - d) The value of *CmdSize* is outside the valid range.
  - e) The value of a member of *Option* is outside the valid range.
  - f) The value of *CmdSize* exceeds the size of *CmdDat[]*.
  - g) The response size exceeds the size of *RespDat[]*.
  - h) Communications fail.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000001 | The communications service was interrupted.                                   | <ul style="list-style-type: none"> <li>• Check the data link execution status.</li> <li>• Check the capacity of the transfer destination area on the third node.</li> </ul> |
| 16#00000101 | The local node is not part of the network.                                    | Make the local node part of the network.  |
| 16#00000102 | A token timeout occurred.   | Set the local node address to be within the maximum node address.   |
| 16#00000103 | The retry count was exceeded.   | Perform inter-node tests. If any error is found, check the operating environment.   |
| 16#00000104 | The allowable number of send frames was exceeded.                             | Check the status of events in the network and reduce the number of events in each task period. Or, increase the number of allowable send frames.                            |
| 16#00000105 | The IP address of the local node is out of range.                             | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000106 | The IP address of the local node is also used by another node in the network. | Change one of the node addresses that are duplicated.   |
| 16#00000201 | The remote node is not part of the network.                                   | Make the remote node part of the network.   |
| 16#00000202 | A Unit with the specified unit address does not exist at the destination.     | Correctly set the unit address for the destination network address.   |
| 16#00000203 | The third node is not part of the network.                                    | <ul style="list-style-type: none"> <li>• Check the address of the Unit that is the third node.</li> <li>• Specify only one node for the third node.</li> </ul>              |
| 16#00000204 | The remote node is busy.  | Increase the number of retries or correct the system so that communications traffic is not concentrated on the remote node.   |
| 16#00000205 | A response timeout occurred.  | Check the settings of the communications parameters.  |
| 16#00000206 | There is an error in the transmission path.                                   | <ul style="list-style-type: none"> <li>• Attempt retries.</li> <li>• If this error occurs frequently, check for noise.</li> </ul>   |
| 16#00000301 | A Communications Controller Error occurred.                                   | Refer to the operation manual for the relevant Unit and make suitable corrections.  |

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00000302 | There is an error in the CPU Unit at the remote node.   | Refer to the manual for the CPU Unit at the remote node and remove the error.   |
| 16#00000303 | There is an error in the relevant Controller and a response is not returned.                                | Check the communications status on the network and restart the relevant Controller. If the error still occurs, replace the relevant Controller.   |
| 16#00000304 | The unit number setting is not correct.   | Set the rotary switches on the Serial Communications Unit correctly.  |
| 16#00000401 | The command that was sent is not supported.   | Set the command array correctly.  |
| 16#00000402 | The Unit model or version is not supported.   | Check the Unit model and version.   |
| 16#00000501 | The remote address setting is wrong.  | Set the destination address in the routing tables.  |
| 16#00000502 | Routing tables are not registered.  | Set the source node, destination node, and relay nodes in the routing tables.   |
| 16#00000503 | There is an error in the routing tables.  | Correct the settings in the routing tables.   |
| 16#00000504 | There are too many relay points.  | Restructure the network or correct the routing tables so that commands are used within a three-layer range.   |
| 16#00001001 | The command is too long.  | Set the command array correctly.  |
| 16#00001002 | The command is too short.   | Set the command array correctly.  |
| 16#00001003 | The number of write elements that is specified in the command does not agree with the number of write data. | Specify the same number of write elements and write data.   |
| 16#00001004 | The command format is incorrect.  | Set the command array correctly.  |
| 16#00001005 | There is an error in the header.  | Correct the settings in the routing tables.   |
| 16#00001101 | The area type does not exist.   | Refer to the command variables and parameter type codes and set the relevant codes.   |
| 16#00001102 | An access size is wrong.  | Correctly set the variable and parameter access sizes.  |
| 16#00001103 | An out-of-range address was specified.  | Specify an address that is within the process range.  |
| 16#00001104 | The address range was exceeded.   | <ul style="list-style-type: none"> <li>Specify an address that is within the process range.</li> <li>Correct the settings in the data link table.</li> </ul>                                      |
| 16#00001106 | A communications sequence number that is not registered was specified.                                      | Correct the communications sequence number or add the sequence with the CX-Protocol.  |
| 16#00001109 | An interrelationship error occurred.  | <ul style="list-style-type: none"> <li>Correct the size relationships in the command data.</li> <li>Correct the settings in the data link table.</li> </ul>                                       |
| 16#0000110A | Data is redundant.  | <ul style="list-style-type: none"> <li>Cancel the current process or wait for it to be completed before you execute the command.</li> <li>Correct the settings in the data link table.</li> </ul> |
| 16#0000110B | The response is too long.   | Set the number of elements in the command array correctly.  |
| 16#0000110C | This is another parameter error.  | Set the command array correctly.  |
| 16#00002002 | The data is protected.  | Execute the command again after clearing the protection.  |
| 16#00002003 | There is no registered table.   | Set the table correctly.  |
| 16#00002004 | There is no data that matches the search data.  | Set the search data correctly.  |
| 16#00002005 | The relevant program number does not exist.   | Set a valid program number.   |

| Value       | Error  | Correction  |
|-------------|--|---|
| 16#00002006 | The relevant file does not exist.  | Set the file name correctly, including the subdirectory names.  |
| 16#00002007 | A verification error occurred.   | <ul style="list-style-type: none"> <li>• Check the contents of memory and write the correct data.</li> <li>• Check the contents of the file.</li> </ul>   |
| 16#00002101 | Access is not possible because the area is read-only.  | Execute the command again after clearing the write protection.  |
| 16#00002102 | The data is protected or the data link table cannot be written.                                  | <ul style="list-style-type: none"> <li>• Execute the command again after clearing the write protection.</li> <li>• Set the system settings in the data link table.</li> </ul>                   |
| 16#00002103 | Registration is not possible.  | <ul style="list-style-type: none"> <li>• Create the file after deleting unnecessary files or prepare new file memory.</li> <li>• Execute the command again after closing open files.</li> </ul> |
| 16#00002105 | The relevant program number does not exist.  | Set a valid program number.   |
| 16#00002106 | The relevant file does not exist.  | Set the file name correctly, including the subdirectory names.  |
| 16#00002107 | A file with the same name already exists.  | Execute the command again after changing the name of the file to write.   |
| 16#00002108 | The change is not allowed because it causes an error.  | Correct the settings.   |
| 16#00002201 | The operation was not possible because a protocol macros is already in execution.                | Use an N.C. program input for the Protocol Macro Execution Flag.  |
| 16#00002202 | The operating mode is wrong.   | Check the operating mode.   |
| 16#00002203 | The operating mode is wrong for the instruction (PROGRAM mode).                                  | Check the operating mode of the Controller.   |
| 16#00002204 | The operating mode is wrong for the instruction (DEBUG mode).                                    | Check the operating mode of the Controller.   |
| 16#00002205 | The operating mode is wrong for the instruction (MONITOR mode).                                  | Check the operating mode of the Controller.   |
| 16#00002206 | The operating mode is wrong for the instruction (RUN mode).                                      | Check the operating mode of the Controller.   |
| 16#00002207 | The specified node is not the polling node.  | Confirm which node is the polling node of the network.  |
| 16#00002208 | The operating mode is wrong for the instruction.   | Check step activation status.   |
| 16#00002211 | The Unit is busy.  | Increase the number of retries or review the system so that communications traffic is not concentrated on the relevant Unit.  |
| 16#00002301 | The file device does not exist.  | Insert the media. Or, format the EM.  |
| 16#00002302 | There is no file memory.   | Check the file memory device.   |
| 16#00002303 | There is no built-in clock.  | Check the specifications of the model.  |
| 16#00002401 | A checksum error occurred in the protocol macro data, or the data transfer is not yet completed. | Transfer the protocol macro data from the CX-Protocol again.  |
| 16#00002502 | There is an error in the memory.   | Transfer the correct data to the memory.  |
| 16#00002503 | The registered I/O Unit configuration does not agree with the actual Unit configuration.         | Check the I/O Unit configuration.   |

| Value       | Error   | Correction  |
|-------------|---|---|
| 16#00002504 | There are too many local or remote I/O points.                                    | Set the number of local and remote I/O points correctly.  |
| 16#00002505 | An error occurred in a data transmission between the CPU Unit and a CPU Bus Unit. | Check the Units and the connecting cables. After removing the error, execute a command to reset the error.  |
| 16#00002506 | The same rack number, unit number, or I/O address is set more than once.          | Correct the settings so that each number is unique.   |
| 16#00002507 | An error occurred in a data transmission between the CPU Unit and an I/O Unit.    | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#00002509 | There is an error in SYSMAC BUS/2 data transmission.                              | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250A | An error occurred in a CPU Bus Unit data transmission.                            | Check the Units and connecting cables. After removing the error, execute a command to reset the error.  |
| 16#0000250D | The same channel setting is used more than once.                                  | Set the I/O channels correctly.   |
| 16#0000250F | There is an error in memory.  | <ul style="list-style-type: none"> <li>For internal memory, execute the command again after writing the correct data.</li> <li>For a Memory Card or EM file memory, execute the expansion memory format command.</li> <li>If the error cannot be reset with the above corrections, replace the memory.</li> </ul> |
| 16#00002510 | The end station setting is wrong.   | Set the end station correctly.  |
| 16#00002601 | Protection is already cleared.  | You do not need to clear protection.  |
| 16#00002602 | The password was wrong.   | Specify the correct password.   |
| 16#00002604 | The data is protected.  | <ul style="list-style-type: none"> <li>Execute the command again after clearing the write protection.</li> <li>Wait for the service that is currently in execution to end, or stop the service and execute the command again.</li> </ul>  |
| 16#00002605 | The service is busy.  | Wait for the service that is currently in execution to end, or stop the service and execute the command again.  |
| 16#00002606 | The service is stopped.   | Execute the relevant service as required.   |
| 16#00002607 | You do not have the execution right.  | <ul style="list-style-type: none"> <li>Execute the operation from the node that accessed the data link.</li> <li>If the error still occurs after a restart, replace the Controller.</li> </ul>  |
| 16#00002608 | The environment is not set.   | Make the necessary settings.  |
| 16#00002609 | The required items are not set.   | Set the required items.   |
| 16#0000260A | The specified number is already defined.  | Execute the command again after changing the specified number to an action or transition number that is not already registered.   |
| 16#0000260B | The error cannot be reset.  | Remove the cause of the error and then execute the error reset command.   |
| 16#00003001 | You do not have access rights.  | Wait for the access to be allowed and then execute the command again.   |
| 16#00004001 | The service was interrupted.  | Execute the command again after clearing the cause of the service interruption.   |

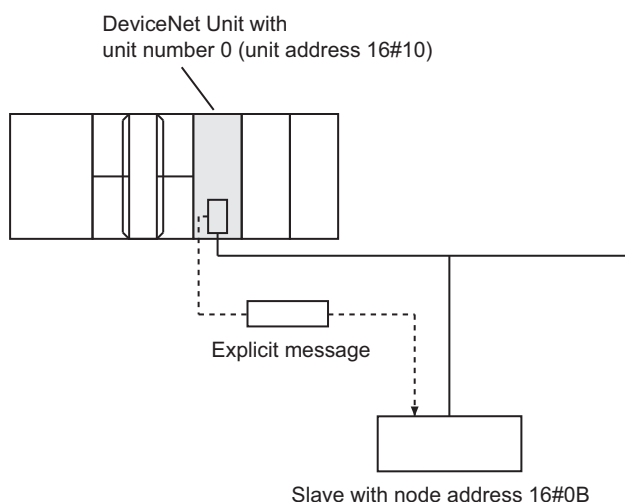
**Note** In addition to the codes in the above table, the values of bits 6, 7, and 15 in the end code can be TRUE. If the value of bit 6 or 7 is TRUE, there is an error in the CPU Unit at the destination. If the value of bit 15 is TRUE, an error occurred during a network relay.

## Sample Programming

In this sample, the SendCmd instruction sends an explicit message via a DeviceNet Unit. This sample reads the vendor ID from the slave with node address 16#0B through the DeviceNet Unit with unit address 16#10.

The following communications specifications are used.

| Item                           | Description |
|--------------------------------|-------------|
| Unit address of DeviceNet Unit | 16#10       |
| Slave node address             | 16#0B       |
| Service code                   | 16#0E       |
| Class ID                       | 1           |
| Instance ID                    | 1           |
| Attribute ID                   | 1           |
| Timeout time                   | 2.0 s       |
| Retry count                    | 2           |



## Command array SendDat[] and Response storage array RecvDat[]

The contents of command array SendDat[] and response storage array RecvDat[] are as follows:

### ● Command Array: BYTE array

| Array element | Item               | Content  | Value      |
|---------------|--------------------|--|------------|
| SendDat[0]    | Command code       | The command code to send an explicit message is 16#2801.                                     | BYTE#16#28 |
| SendDat[1]    |                    |  | BYTE#16#01 |
| SendDat[2]    | Slave node address | The node address is 16#0B.   | BYTE#16#0B |
| SendDat[3]    | Service code       | The service code to read the value of a specified attribute (Get Attribute Single) is 16#0E. | BYTE#16#0E |
| SendDat[4]    | Class ID           | The class ID of the Identity object is 16#0001.  | BYTE#16#00 |
| SendDat[5]    |                    |  | BYTE#16#01 |
| SendDat[6]    | Instance ID        | ---  | BYTE#16#00 |
| SendDat[7]    |                    |  | BYTE#16#01 |

| Array element | Item         | Content   | Value      |
|---------------|--------------|---|------------|
| SendDat[8]    | Attribute ID | The attribute ID of the vendor ID (Vendor ID) is 16#01. | BYTE#16#01 |

### ● Response Storage Array: BYTE Array

| Array element | Item  | Content  |
|---------------|---|--|
| RecvDat[0]    | Command code  | The command code to send an explicit message is 16#2801. |
| RecvDat[1]    |   |  |
| RecvDat[2]    | Completion code                                       | The completion code is 16#0000 for a normal end.         |
| RecvDat[3]    |   |  |
| RecvDat[4]    | Number of bytes received after the slave node address | 4 bytes  |
| RecvDat[5]    |   |  |
| RecvDat[6]    | Slave node address                                    | The node address is 16#0B for a normal end.              |
| RecvDat[7]    | Service code  | The service code for a normal end is 16#8E.              |
| RecvDat[8]    | Vendor ID   | Slave vendor ID.   |
| RecvDat[9]    |   |  |

## Definitions of Global Variables

### ● Global Variables

| Name                | Data type | AT specification *1                      | Comment |
|---------------------|-----------|--|---------|
| DeviceNet_OnlineSta | BOOL      | IOBus://rack#0/slot#0/Unit2Sta/OnlineSta | Online  |

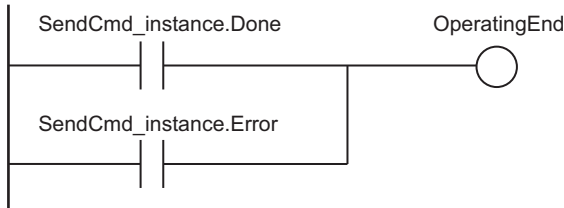
\*1. AT when the Serial Communications Unit is mounted to slot number 0 in rack number 0.

## LD

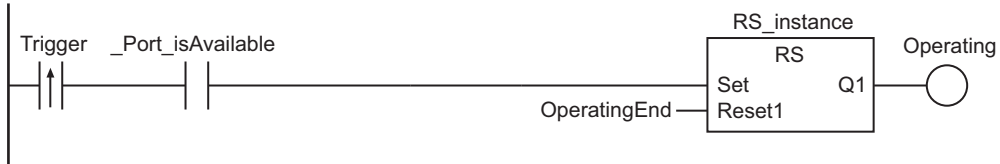
| Internal Variables | Variable         | Data type           | Initial value                            | Comment                     |
|--------------------|------------------|---------------------|--|-----------------------------|
|                    | OperatingEnd     | BOOL                | FALSE                                    | Processing completed        |
|                    | Trigger          | BOOL                | FALSE                                    | Execution condition         |
|                    | Operating        | BOOL                | FALSE                                    | Processing                  |
|                    | InDNetAdr        | _sDNET_ADR          | (NetNo:=0, NodeNo:=0, UnitNo:=16#0)      | Destination network address |
|                    | InOption         | _sRESPONSE          | (isNonResp:=FALSE, TimeOut:=0, Retry:=0) | Response                    |
|                    | SendDat          | ARRAY[0..8] OF BYTE | [9(16#0)]                                | Send data                   |
|                    | RecvDat          | ARRAY[0..9] OF BYTE | [10(16#0)]                               | Receive data                |
|                    | RS_instance      | RS                  |  |                             |
|                    | SendCmd_instance | SendCmd             |  |                             |

| External Variables | Variable          | Data type | Comment   |
|--------------------|-------------------|-----------|---|
|                    | _Port_isAvailable | BOOL      | Network Communications Instruction Enabled Flag |

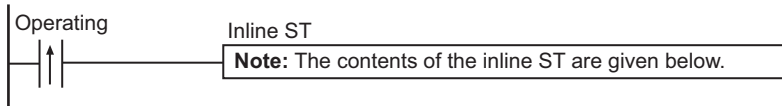
Determine if execution of the SendCmd instruction is completed.



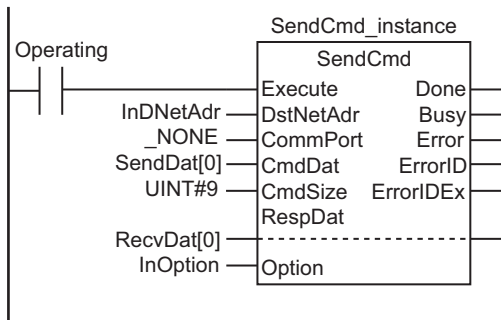
Accept trigger.



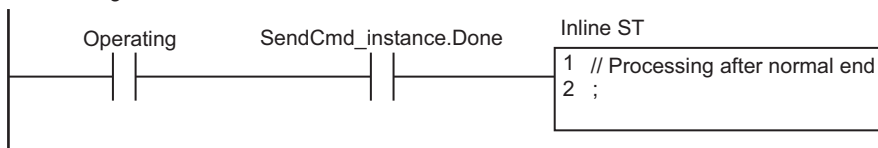
Set communications parameters.



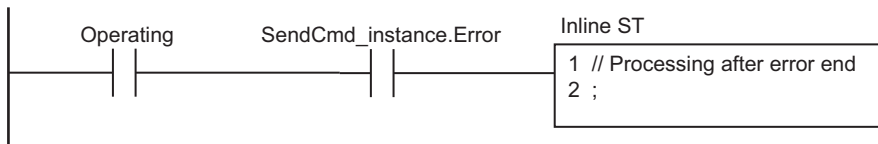
Execute SendCmd instruction.



Processing after normal end



Processing after error end



### ● Contents of Inline ST

```
InDNetAdr.NetNo      :=USINT#0;      // Set network address.
InDNetAdr.NodeNo     :=USINT#0;
InDNetAdr.UnitNo     :=BYTE#16#10;
InOption.isNonResp   :=FALSE;        // Set response.
InOption.TimeOut     :=UINT#20;
```

```

InOption.Retry      :=USINT#2;
SendDat[0]         :=BYTE#16#28; // Set command array.
SendDat[1]         :=BYTE#16#01;
SendDat[2]         :=BYTE#16#0B;
SendDat[3]         :=BYTE#16#0E;
SendDat[4]         :=BYTE#16#00;
SendDat[5]         :=BYTE#16#01;
SendDat[6]         :=BYTE#16#00;
SendDat[7]         :=BYTE#16#01;
SendDat[8]         :=BYTE#16#01;
    
```

### 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                                 |
|                    | InDNetAdr        | _sDNET_ADR          | (NetNo:=0, NodeNo:=0, UnitNo:=16#0)      | Destination network address                |
|                    | InOption         | _sRESPONSE          | (isNonResp:=FALSE, TimeOut:=0, Retry:=0) | Response                                   |
|                    | SendDat          | ARRAY[0..8] OF BYTE | [9(16#0)]                                | Send data                                  |
|                    | RecvDat          | ARRAY[0..9] OF BYTE | [10(16#0)]                               | Receive data                               |
|                    | SendCmd_instance | SendCmd             |  |  |

| External Variables | Variable            | Data type | Comment   |
|--------------------|---------------------|-----------|---|
|                    | DeviceNet_OnlineSta | BOOL      | Online  |
|                    | _Port_isAvailable   | BOOL      | Network Communications Instruction Enabled Flag |

```

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
    AND (DeviceNet_OnlineSta=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SendCmd instruction.
IF (OperatingStart=TRUE) THEN
    SendCmd_instance(
        Execute :=FALSE,
    
```



```

        DstNetAdr:=InDNetAdr,
        CommPort :=_NONE,
        CmdDat   :=SendDat[0],
        CmdSize  :=UINT#9,
        RespDat  :=RecvDat[0],
        Option   :=InOption);
InDNetAdr.NetNo   :=USINT#0;      // Set network address.
InDNetAdr.NodeNo :=USINT#0;
InDNetAdr.UnitNo :=BYTE#16#10;
InOption.isNonResp :=FALSE;      // Set response.
InOption.TimeOut :=UINT#20;
InOption.Retry    :=USINT#2;
SendDat[0]        :=BYTE#16#28;  // Set command array.
SendDat[1]        :=BYTE#16#01;
SendDat[2]        :=BYTE#16#0B;
SendDat[3]        :=BYTE#16#0E;
SendDat[4]        :=BYTE#16#00;
SendDat[5]        :=BYTE#16#01;
SendDat[6]        :=BYTE#16#00;
SendDat[7]        :=BYTE#16#01;
SendDat[8]        :=BYTE#16#01;
OperatingStart   :=FALSE;
END_IF;
// Execute SendCmd instruction.
IF (Operating=TRUE) THEN
    SendCmd_instance(
        Execute :=TRUE,
        DstNetAdr:=InDNetAdr,
        CommPort :=_NONE,
        CmdDat   :=SendDat[0],
        CmdSize  :=UINT#9,
        RespDat  :=RecvDat[0],
        Option   :=InOption);

    IF (SendCmd_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (SendCmd_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;

```

# 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.

| Instruction   | Name                  | FB/<br>FUN | Graphic expression | ST expression   |
|---------------|-----------------------|------------|--------------------|---|
| NX_SerialSend | Send No-protocol Data | FB         |                    | NX_SerialSend_instance(Execute, DevicePort, SendDat, SendSize, SendCfg, Option, Abort, Done, Busy, CommandAborted, 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

|                      | Meaning                          | I/O    | Description                           | Valid range           | Unit  | Default |
|----------------------|----------------------------------|--------|---------------------------------------|-----------------------|-------|---------|
| DevicePort           | Device port                      | Input  | Object that represents a device port  | ---                   | ---   | ---     |
| SendDat[]<br>(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   |
| CommandAborted       | 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           |         |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SendDat[]<br>(array) |         | OK          |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| SendSize             |         |             |      |       |       |       | 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 |
| SendCfg        | Refer to <i>Function</i> on page 2-1331 for details on the structure <code>_sSERIAL_CFG</code> .         |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Option         | Refer to <i>Function</i> on page 2-1331 for details on the structure <code>_sSERIAL_SEND_OPTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| Abort          | OK   |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |
| CommandAborted | 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  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---     |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOptionBoard</code> | ---  | ---     |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---     |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---     |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---     |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---     |
| PortNo     | Port number            | Port number<br>1: Port 1<br>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.

| Position | Port      | Description               | R/W | Data Type           | Variable                     |
|----------|-----------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          |           | Ch1 Output SID            | W   | USINT               |                              |
|          |           | Ch1 Input SID Response    | W   | USINT               |                              |
|          |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|          |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |           | Ch1 Output Data Length    | W   | UINT                |                              |
|          |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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<br>_SERIAL_START_STARTCODE1<br>_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<br>_SERIAL_END_ENDCODE1<br>_SERIAL_END_ENDCODE2<br>_SERIAL_END_TERMINATION_CHAR<br>_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     |
|---------------------------------------|-------------|
| <code>_SERIAL_START_NONE</code>       | None        |
| <code>_SERIAL_START_STARTCODE1</code> | 1-byte code |
| <code>_SERIAL_START_STARTCODE2</code> | 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               |
|---|-----------------------|
| <code>_SERIAL_END_NONE</code>             | None                  |
| <code>_SERIAL_END_ENDCODE1</code>         | 1-byte code           |
| <code>_SERIAL_END_ENDCODE2</code>         | 2-byte code           |
| <code>_SERIAL_END_TERMINATION_CHAR</code> | Termination condition |
| <code>_SERIAL_END_RCV_SIZE</code>         | Receive size          |

Refer to *Operation of Start Code and End Code* on page 2-1347 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       |



#### Precautions for Correct Use

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 <i>StartTrig</i> | Operation   |
|---------------------------|---|
| _SERIAL_START_NONE        | ---   |
| _SERIAL_START_STARTCODE1  | <i>SendDat</i> is sent with start code attached to its beginning.<br>Example: STX |
| _SERIAL_START_STARTCODE2  |   |

| Value of <i>EndTrig</i>      | Operation   |
|------------------------------|---|
| _SERIAL_END_NONE             | ---   |
| _SERIAL_END_ENDCODE1         | <i>SendDat</i> is sent with end code attached to its end.<br>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 instruction execution is completed before an attempt of interruption, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

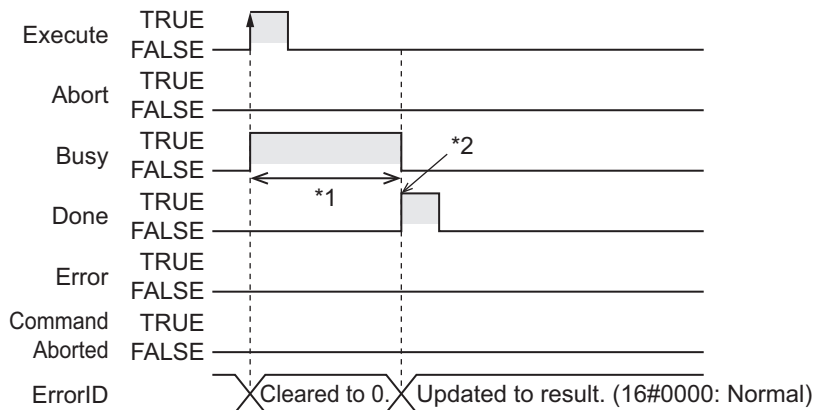
The interruption operation only finishes the *Busy* processing, and it does not clear the send buffer. To clear the buffer, use the instruction, *NX\_SerialBufClear* on page 2-1409.

## Timing Charts

The following figures show the timing charts.

### ● Normal end (when *SendDelay* is 0 (0 s))

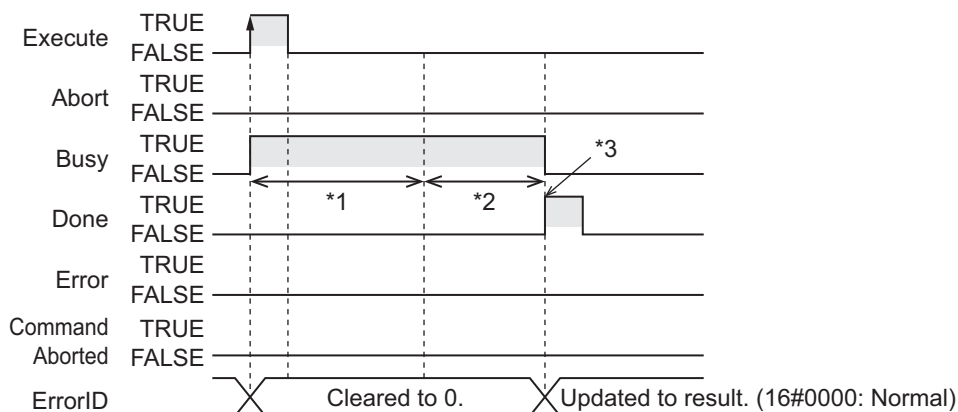
The operation is as follows when *SendDelay* is 0 (0 s).



- \*1. Sending processing
- \*2. Sending completed

### ● Normal end (when *SendDelay* is 100 (1 s))

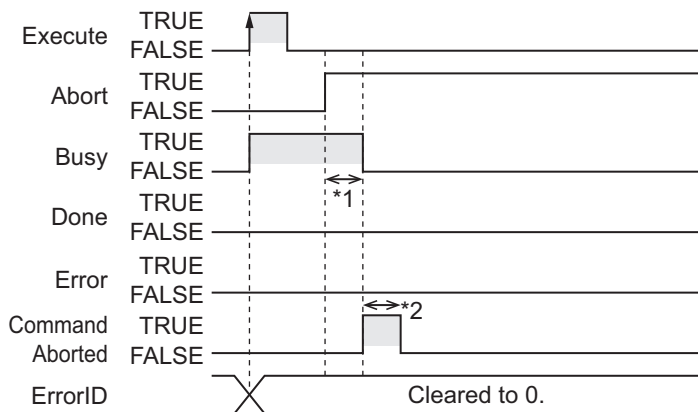
The operation is as follows when *SendDelay* is 100 (1 s).



- \*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.

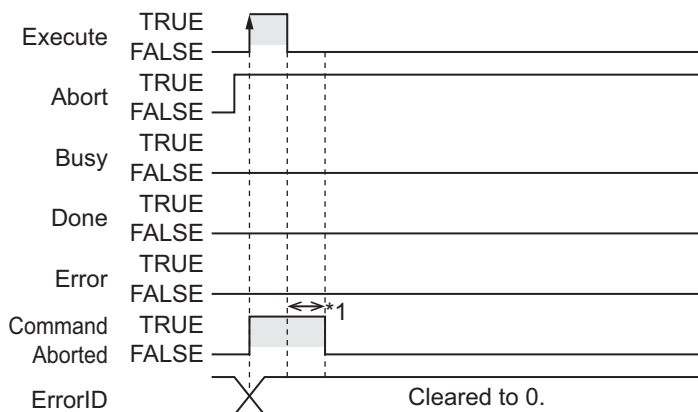


\*1. Interruption processing

\*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

| Name                        | Meaning                           | Data type                        | Description  |
|-----------------------------|-----------------------------------|----------------------------------|--|
| _PLC_OptBoard-<br>Sta*1     | Option Board Sta-<br>tus          | ARRAY[1..2] of<br>_sOPTBOARD_STA | This stores the status of the Option Board.  |
| _NXB_UnitIOAc-<br>tiveTbl*2 | NX Unit I/O Data<br>Active Status | ARRAY[0..32] OF<br>BOOL*3        | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is *ARRAY [0..8] OF BOOL*.

## Precautions for Correct Use

- While *Abort* remains FALSE, execution of this instruction is continued until completed even when *Execute* changes to FALSE or the execution time exceeds the task period.



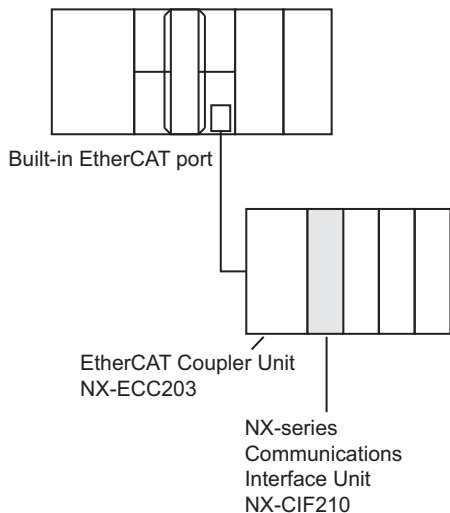
The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.

- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range is set for *SendSize*, *SendCfg.StartTrig*, *SendCfg.EndTrig*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The array variable specified for *SendDat* is smaller than the size specified with *SendSize*.
  - c) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - d) The data type of *DevicePort* is invalid.
  - e) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - f) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
  - g) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - h) The serial communications mode of the specified Option Board is not *No-protocol*.

## Sample Programming

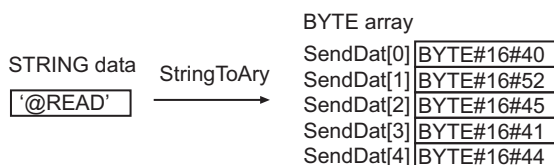
In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

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 StringToArray 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]<br>OF BOOL | ECAT://node#1/NX<br>Unit I/O Data Active<br>Status 125 | Usage of I/O data for 63 NX<br>Units.      |
| N1_Node_location_information           | _sNXUNIT_ID             | ---  | Device variable to specify NX-<br>CIF210*1 |

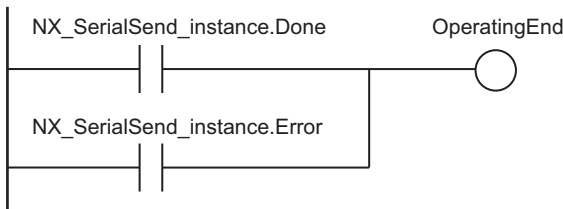
\*1. On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

**LD**

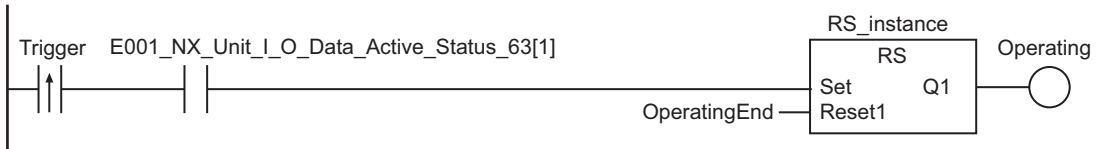
| Internal Variables | Variable               | Data type            | Initial value        | Comment              |
|--------------------|------------------------|----------------------|----------------------|----------------------|
|                    | OperationEnd           | BOOL                 | 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 | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | N1_Node_location_information           | _sNXUNIT_ID          | Device variable to specify NX-CIF210   |

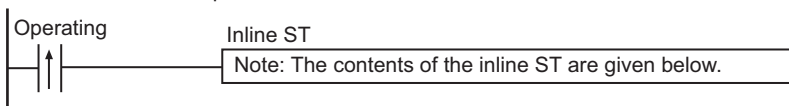
Determine if execution of the NX\_SerialSend instruction has ended.



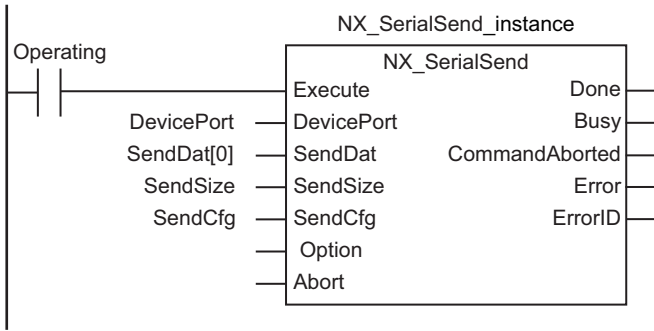
Accept trigger.



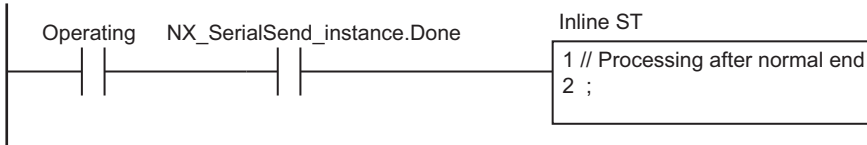
Set communications parameters.



Execute NX\_SerialSend instruction.



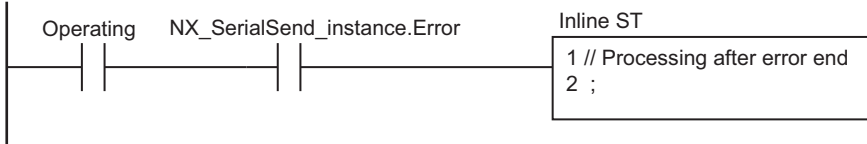
Processing after normal end



```

1 // Processing after normal end
2 ;
    
```

Processing after error end



```

1 // Processing after error end
2 ;
    
```

● 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        |               |  |

| Internal Variables | Variable  | Data type      | Initial value         | Comment            |
|--------------------|-----------|----------------|-----------------------|--------------------|
|                    | StartTrig | _eSERIAL_START | _SERIAL_START_NONE    | Without start code |
|                    | StartCode | BYTE[2]        | [2(16#0)]             |                    |
|                    | EndTrig   | _eSERIAL_END   | _SERIAL_END_END-CODE1 | 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 | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | 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/<br>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

|                     | Meaning                      | I/O    | Description   | Valid range           | Unit  | Default |
|---------------------|------------------------------|--------|---|-----------------------|-------|---------|
| DevicePort          | Device port                  | Input  | Object that represents a device port                    | ---                   | ---   | ---     |
| Size                | Storage size                 |        | Size of <i>RcvDat</i> 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[]<br>(array) | Receive data                 | In-out | Variable to store data received from the receive buffer | Depends on data type. | ---   | ---     |
| CommandAborted      | 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 | ---     |

|            | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |       |          |       |       |      |     |      |      |                  |       |  |      |     |    |        |
| Size       |             |             |      |       |       |       | OK       |       |       |      |     |      |      |                  |       |  |      |     |    |        |

|                       | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----------------------|---|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                       | BOOL  | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| RcvCfg                | Refer to <i>Function</i> on page 2-1344 for details on the structure <code>_sSERIAL_CFG</code> .        |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Option                | Refer to <i>Function</i> on page 2-1344 for details on the structure <code>_sSERIAL_RCV_OPTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Abort                 | OK  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| RcvDat[] (ar-<br>ray) |   | OK          |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| CommandA-<br>borted   | 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 | De-<br>fault |
|------------|-----------------------------|---|----------------------------|--|------|--------------|
| DevicePort | Device port                 | Object that repre-<br>sents a device port | <code>_sDEVICE_PORT</code> | ---  | ---  | ---          |
| DeviceType | Device type                 | Type of the device<br>to specify          | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcat-<br/>Slave</code><br><code>_DeviceOption-<br/>Board</code> | ---  | ---          |
| NxUnit     | Specified Unit              | NX Unit to control                        | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---          |
| EcatSlave  | Specified slave             | EtherCAT slave to<br>control              | <code>_sECAT_ID</code>     | ---  | ---  | ---          |
| OptBoard   | Specified Op-<br>tion Board | Option Board to<br>control                | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---          |
| Reserved   | Reserved                    | Reserved                                  | Reserved                   | ---  | ---  | ---          |
| PortNo     | Port number                 | Port number<br>1: Port 1<br>2: Port 2     | USINT                      | Depends on da-<br>ta 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.

The screenshot shows the I/O Map configuration for an NX-CIF210. The 'Node location information' port is assigned the variable 'N1\_Node\_location\_information'. A red arrow points to this assignment with the text 'Assign a variable.' Below it, several other ports (Ch1 Output SID, Ch1 Input SID Response, Ch1 Output Data Type, Ch1 Output Sub Info, Ch1 Output Data Length, Ch1 Output Data 01-05) are listed with 'W' in the R/W column, and a red arrow points to this section with the text 'Do not assign variables.'

| Position Unit1 | Port      | Description               | R/W | Data Type           | Variable                     |
|----------------|-----------|---------------------------|-----|---------------------|------------------------------|
|                | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|                |           | Ch1 Output SID            | W   | JSINT               |                              |
|                |           | Ch1 Input SID Response    | W   | JSINT               |                              |
|                |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|                |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|                |           | Ch1 Output Data Length    | W   | JSINT               |                              |
|                |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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<br>_SERIAL_START_STARTCODE1<br>_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<br>_SERIAL_END_ENDCODE1<br>_SERIAL_END_ENDCODE2<br>_SERIAL_END_TERMINATION_CHAR<br>_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     |
|---------------------------------------|-------------|
| <code>_SERIAL_START_NONE</code>       | None        |
| <code>_SERIAL_START_STARTCODE1</code> | 1-byte code |
| <code>_SERIAL_START_STARTCODE2</code> | 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               |
|---|-----------------------|
| <code>_SERIAL_END_NONE</code>             | None                  |
| <code>_SERIAL_END_ENDCODE1</code>         | 1-byte code           |
| <code>_SERIAL_END_ENDCODE2</code>         | 2-byte code           |
| <code>_SERIAL_END_TERMINATION_CHAR</code> | Termination condition |
| <code>_SERIAL_END_RCV_SIZE</code>         | Receive size          |

Refer to *Operation of Start Code and End Code* on page 2-1347 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                         | _sSERIAL_RCV_OPTION | ---                     | ---   | ---     |
| 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 (Reserved)   | Receive buffer clear condition | Receive buffer clear condition | BOOL                | Depends on data type.*3 | ---   | 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.

\*3. Receive buffer clear is not executed even if either TRUE or FALSE is specified.



### Precautions for Correct Use

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 <i>RcvSize</i> 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<br>_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 <i>RcvSizeCfg</i> . Processing is performed only after the specified amount of data is accumulated in the buffer.   |

| StartTrig  | EndTrig                      | Operation   |
|--|------------------------------|---|
| _SERIAL_START_STARTCODE1<br>_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.<br>Example: ETX   |
|  | _SERIAL_END_ENDCODE2         |   |
|  | _SERIAL_END_TERMINATION_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 <i>RcvSize</i> .<br>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.<br>Example: ETX   |
|  | _SERIAL_END_ENDCODE2         |  |
|  | _SERIAL_END_TERMINATION_CHAR | Error end, but data is received. *1  |
|  | _SERIAL_END_RCV_SIZE         | <ul style="list-style-type: none"> <li>• Error end for an input value check error</li> <li>• Data cannot be received.</li> </ul> |
| _SERIAL_START_STARTCODE1<br>_SERIAL_START_STARTCODE2 | _SERIAL_END_NONE             | Error end, but data is received.   |
|  | _SERIAL_END_ENDCODE1         | Error end, but data is received.<br>Example: ETX   |
|  | _SERIAL_END_ENDCODE2         |  |
|  | _SERIAL_END_TERMINATION_CHAR | Error end, but data is received. *1  |
|  | _SERIAL_END_RCV_SIZE         | <ul style="list-style-type: none"> <li>• Error end for an input value check error</li> <li>• Data cannot be received.</li> </ul> |

\*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<br><i>RcvDat</i> [ ] |
|----------------|--|---|
| 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 <i>RcvDat</i> . Data is retained in the receive buffer.<br>Receive processing for the data is performed when the next <i>NX_SerialRcv</i> 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. The instruction is interrupted even when the data transmission is in progress.

If the instruction execution is completed before an attempt of interruption, *Done* changes to TRUE and the instruction ends normally.

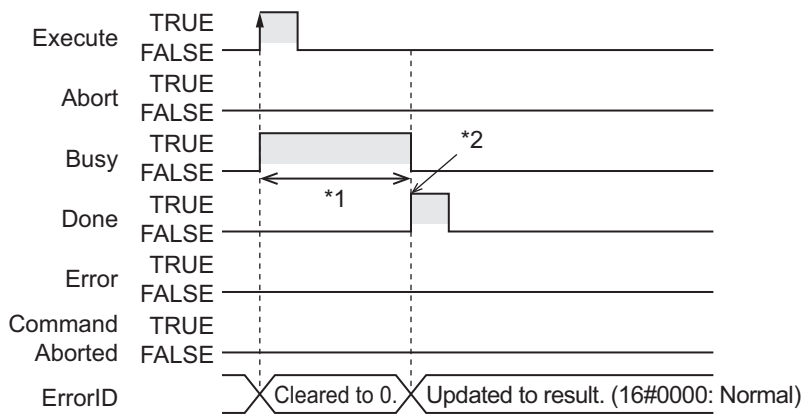
If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

The interruption operation only finishes the *Busy* processing, and it does not clear the send buffer. To clear the buffer, use the instruction, *NX\_SerialBufClear* on page 2-1409.

## Timing Charts

The following figures show the timing charts.

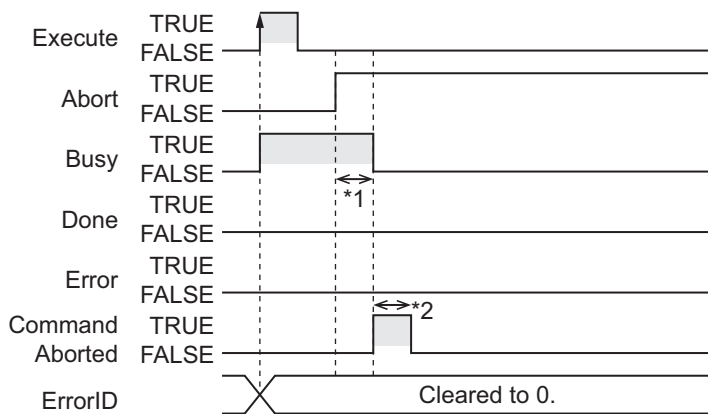
● Normal end



- \*1. Receive processing
- \*2. Data is received in No-protocol mode.

● Interruption executed (when *Busy* is TRUE)

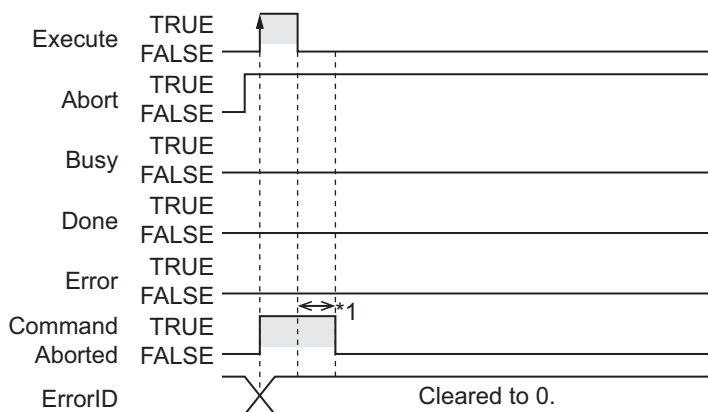
The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.



- \*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_OptBoard-<br>Sta <sup>*1</sup>     | Option Board Sta-<br>tus          | ARRAY[1..2] of<br>_sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOAc-<br>tiveTbl <sup>*2</sup> | NX Unit I/O Data<br>Active Status | ARRAY[0..32] OF<br>BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is *ARRAY [0..8] OF BOOL*.

## Precautions for Correct Use

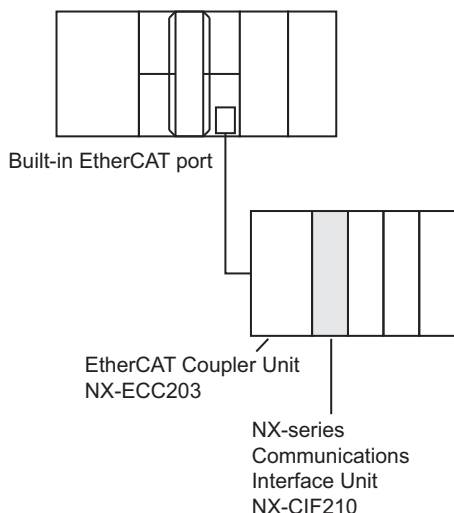
- While *Abort* remains FALSE, execution of this instruction is continued until completed even when *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing. If *Abort* 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.
- A *CIF Unit Initialized* error 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 with *W* in 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 for 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*.
  - More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - The receive buffer is full.

- k) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
- l) A parity error occurred in the data received.
- m) A framing error occurred in the data received.
- n) An overrun error occurred in the data received.
- o) Timeout time elapsed.
- p) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
- q) 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_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | ECAT://node#1/NX<br>Unit I/O Data Active<br>Status 125 | Usage of I/O data for 63 NX<br>Units.      |
| N1_Node_location_information           | _sNXUNIT_ID             | ---  | Device variable to specify NX-<br>CIF210*1 |

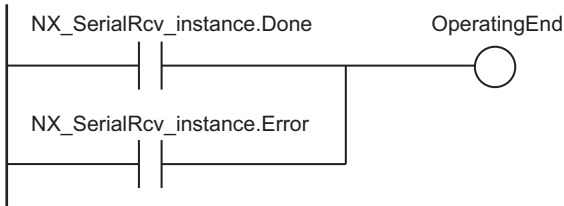
\*1. On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

### LD

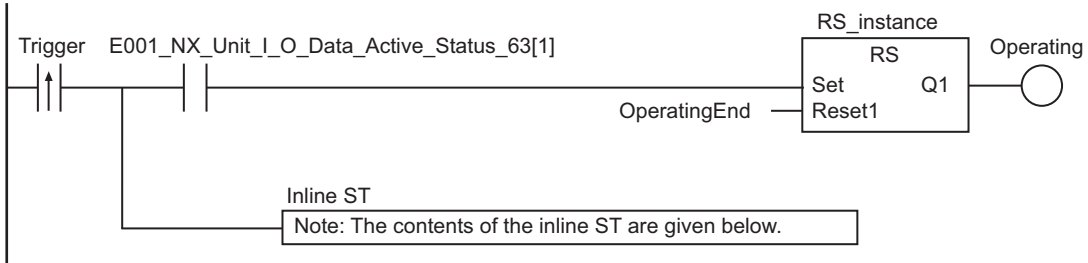
| Internal Variables | Variable              | Data type                | Initial value        | Comment                      |
|--------------------|-----------------------|--------------------------|----------------------|------------------------------|
|                    | OperationEnd          | BOOL                     | FALSE                | Processing completed         |
|                    | Trigger               | BOOL                     | FALSE                | Execution condition          |
|                    | Operating             | BOOL                     | FALSE                | Processing                   |
|                    | DevicePort            | _sDEVICE_PORT            |                      | Port settings                |
|                    | RecvDat               | ARRAY[0..255] OF<br>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#0 s             |                              |
|                    | LastDatRcv            | BOOL                     | FALSE                |                              |

| External Variables | Variable                               | Data type               | Comment  |
|--------------------|--|-------------------------|--|
|                    | E001_NX_Unit_I_O_Data_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | N1_Node_location_information           | _sNXUNIT_ID             | Device variable to specify NX-CIF210   |

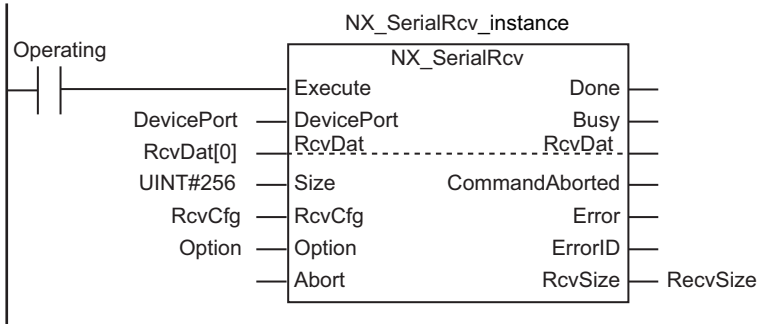
Determine if execution of the NX\_SerialRcv instruction has ended.



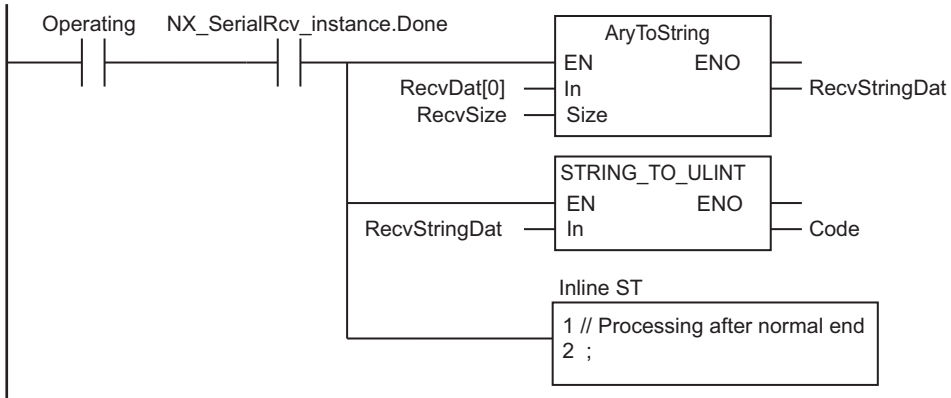
Accept trigger.

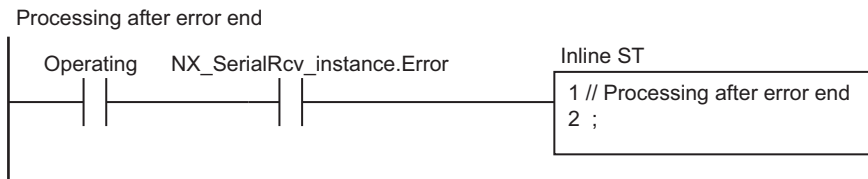


Execute NX\_SerialRcv instruction.



Processing after normal end





## ● 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#0 s             |  |
|                    | LastDatRcv            | BOOL                  | FALSE                |  |

| External Variables | Variable                               | Data type               | Comment  |
|--------------------|--|-------------------------|--|
|                    | E001_NX_Unit_I_O_Data_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | 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_instance.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:=RcvDat,          // Receive data
        RcvSize=>RcvSize);       // Data size that was actually
received
    OperatingStart:=FALSE;
END_IF;
// Execute NX_SerialRcv instruction.
IF (Operating=TRUE) THEN
    NX_SerialRcv_instance(
        Execute:=TRUE,
        DevicePort:=DevicePort,
        Size:=UINT#256,
        RcvDat:=RcvDat,
        RcvSize=>RcvSize);
    IF (NX_SerialRcv_instance.Done=TRUE) THEN
        // Processing after normal end
        RcvStringDat:=AryToString(In:=RcvDat[0],Size:=RcvSize); // Convert
character codes to a text string.
        Code:=STRING_TO_ULINT(RcvDat); // Convert text string to an integer.

        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/<br>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, RespDat, 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

|                      | 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[]<br>(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[]<br>(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.

|                     | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1359 for details on the structure <code>_sDEVICE_PORT</code> .             |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| SlaveAdr            |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| CmdDat[]<br>(array) |   | OK          |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| CmdSize             |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Option              | Refer to <i>Function</i> on page 2-1359 for details on the structure <code>_sSERIAL_MODBUSRTU_OPTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Abort               | OK  |             |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| RespDat[]ar<br>ray  |   | OK          |      |       |       |       |          |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| CommandA-<br>borted | 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 | De-<br>fault |
|------------|------------------------|---------------------------------------|----------------------------|--|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOptionBoard</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---          |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>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.

| Position | Port      | Description               | R/W | Data Type           | Variable                     |
|----------|-----------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          |           | Ch1 Output SID            | W   | USINT               |                              |
|          |           | Ch1 Input SID Response    | W   | USINT               |                              |
|          |           | Ch1 Output Data Type      | W   | WORD                |                              |
|          |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |           | Ch1 Output Data Length    | W   | UINT                |                              |
|          |           | Ch1 Output Data 01        | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | Ch1 Output Data 02        | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | Ch1 Output Data 03        | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | Ch1 Output Data 04        | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | Ch1 Output Data 05        | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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_MODBUSRTU_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    | Timeout time<br>If 0 is set, the timeout time is 2.0 s.   | UINT                      | Depends on data type. | 0.1 s  | 20      |
| NoResponse | No response     | <ul style="list-style-type: none"> <li>Set TRUE when no response is waited for the send command.</li> <li>If TRUE is set, this instruction sends a command and ends normally without waiting for the elapse of the timeout time.</li> </ul> | BOOL                      | Depends on data type. | ---    | FALSE   |
| Retry      | Retry count     | Retry count   | USINT                     | 0 to 15               | ---    | 0       |



### Precautions for Correct Use

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 instruction execution is completed before an attempt of interruption, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

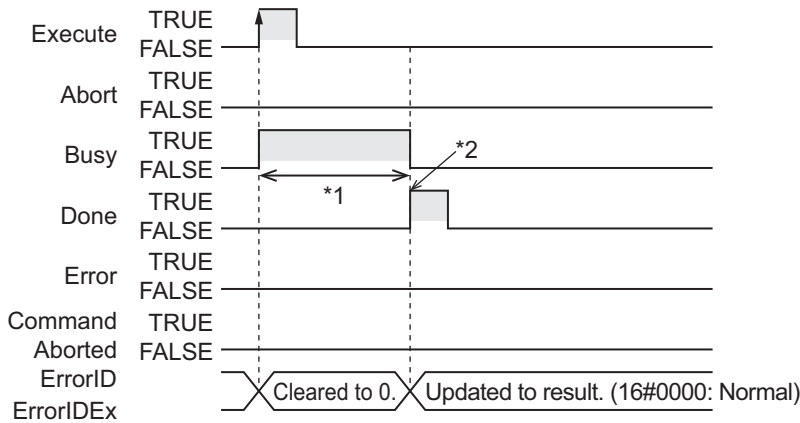
This interruption operation only finishes the *Busy* processing, and it does not clear the send or receive buffer. To clear the buffer, use the instruction, *NX\_SerialBufClear* on page 2-1409.

## Timing Charts

The following figures show the timing charts.

● **Normal end (when *SendDelay* is 0 (0 s))**

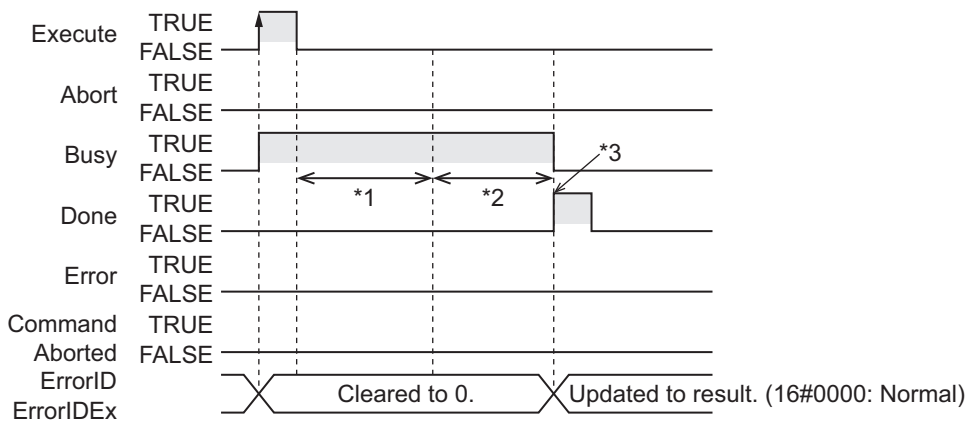
The operation is as follows when *SendDelay* is 0 (0 s).



- \*1. Processing with Modbus-RTU slave
- \*2. A response to the command is received.

● **Normal end (when *SendDelay* is 100 (1 s))**

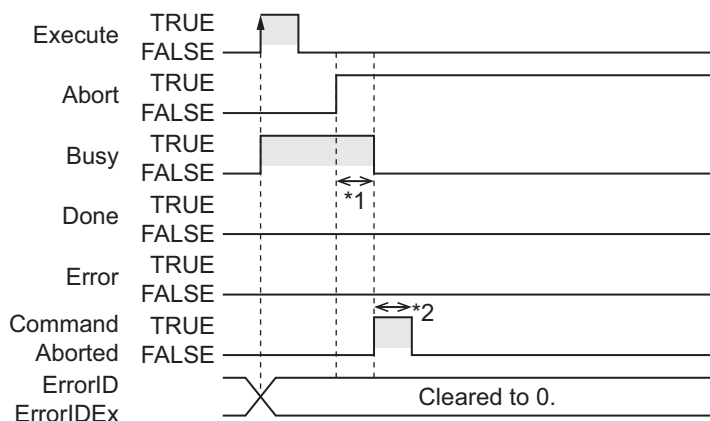
The operation is as follows when *SendDelay* is 100 (1 s).



- \*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.

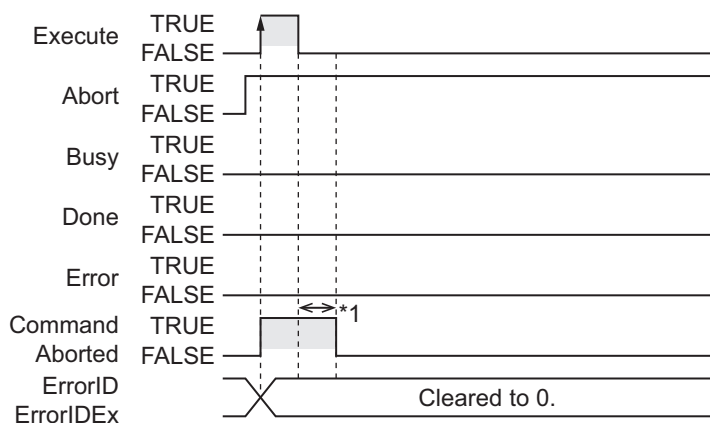


\*1. Interruption processing

\*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_OptBoard-<br>Sta <sup>*1</sup>     | Option Board Sta-<br>tus          | ARRAY[1..2] of<br>_sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOAc-<br>tiveTbl <sup>*2</sup> | NX Unit I/O Data<br>Active Status | ARRAY[0..32] OF<br>BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is *ARRAY [0..8] OF BOOL*.

## 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 completed even when *Execute* changes to FALSE or the execution time exceeds the task period.  
The value of *Done* changes to TRUE when the execution 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.
- A *CIF Unit Initialized* error 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 with *W* in 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 instructions: *NX\_ModbusRtuCmd* instruction, *NX\_ModbusRtuRead* instruction, or *NX\_ModbusRtuWrite* instruction.
  - a) After the operation starts or when you change the operating mode to RUN mode.
  - b) The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
  - c) The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
  - d) 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) A value that is out of range was set for *CmdSize*, *Option.Retry*, *DevicePort.DevicePortType*, *DevicePort.PortNo*, or *SlaveAdr*.
  - b) The variable specified with *CmdDat* is smaller than the size specified with *CmdSize*.
  - c) The size of the received data is larger than the size of the variable set in *RespDat*.
  - d) The Unit or port specified with *DevicePort* does not exist.
  - e) The data type of *DevicePort* is invalid.
  - f) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - g) 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\_Serial-Send instruction, NX\_SerialRcv instruction, NX\_ModbusRtuCmd instruction, NX\_ModbusRtu-Read instruction, and NX\_ModbusRtuWrite instruction.

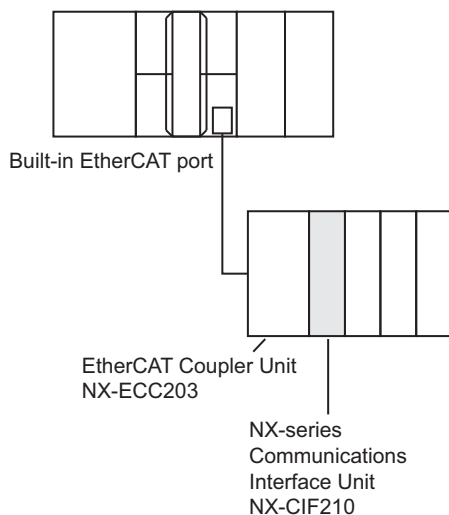
- h) A parity error occurred in the data received.
  - i) A framing error occurred in the data received.
  - j) An overrun error occurred in the data received.
  - k) CRC mismatch occurred for the received data.
  - l) Timeout time elapsed.
  - m) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - n) An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
  - o) There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
  - p) 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=00000XX*. 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.

## ST

| 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                  |               |                                       |
|                    | 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/<br>FUN | Graphic expression | ST expression  |
|------------------|------------------------------|------------|--------------------|--|
| NX_ModbusRtuRead | 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

|                      | 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* <sup>1</sup> | 1 to 247                 | ---               | 1              |
| ReadCmd              | Read command            |        | Read command                              | ---                      | ---               | * <sup>2</sup> |
| Option               | Option                  |        | Option                                    | ---                      | ---               | ---            |
| Abort                | Interruption            |        | Interruption of instruction execution     | Depends on data type.    | ---               | FALSE          |
| ReadDat[]<br>(array) | Read data               | In-out | Variable that stores read data            | Depends on data type.    | ---               | ---            |
| CommandAborted       | Interruption completion | Output | Interruption completion                   | Depends on data type.    | ---               | ---            |
| ReadSize             | Receive size            |        | Receive data size                         | 1 to 2,000* <sup>3</sup> | ---* <sup>4</sup> | ---            |

\*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*.

|                      | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1370 for details on the structure <code>_sDEVICE_PORT</code> .             |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SlaveAdr             |   |             |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ReadCmd              | Refer to <i>Function</i> on page 2-1370 for details on the structure <code>_sSERIAL_MODBUSRTU_READ</code> .   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Option               | Refer to <i>Function</i> on page 2-1370 for details on the structure <code>_sSERIAL_MODBUSRTU_OPTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Abort                | OK  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| ReadDat[]<br>(array) | OK  |             | OK   |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
|                      | An array can also be specified.   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| CommandA-<br>borted  | 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 is 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 | De-<br>fault |
|------------|------------------------|---------------------------------------|----------------------------|--|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOptionBoard</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---          |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>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.

| Position | Port      | Description               | R/W | Data Type           | Variable                     |
|----------|-----------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          |           | ...                       |     |                     |                              |
|          |           | Ch1 Output SID            | W   | JSINT               |                              |
|          |           | Ch1 Input SID Response    | W   | JSINT               |                              |
|          |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|          |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |           | Ch1 Output Data Length    | W   | UINT                |                              |
|          |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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       | <code>_sSERIAL_MODBUSRTU_READ</code> | ---  | ---   | ---                          |
| Fun      | Function code | Function code      | <code>_eMDB_FUN</code>               | <code>_MDB_READ_COILS</code><br><code>_MDB_READ_DISCRETE_INPUTS</code><br><code>_MDB_READ_HOLDING_REGISTERS</code><br><code>_MDB_READ_INPUT_REGISTERS</code> | ---   | <code>_MDB_READ_COILS</code> |
| ReadAdr  | Read address  | Read start address | UINT                                 | Depends on data type.  | ---   | 0                            |
| ReadSize | Read size     | Read size          | UINT                                 | Depends on function code.  | ---*1 | 1                            |

\*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                       |
|--|-------------------------------|
| <code>_MDB_READ_COILS</code>             | Read outputs (bit)            |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | Read inputs (bit)             |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | Read holding registers (word) |
| <code>_MDB_READ_INPUT_REGISTERS</code>   | 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         |
|--|------------------|
| <code>_MDB_READ_COILS</code>             | 1 to 2,000 (bit) |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | 1 to 2,000 (bit) |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | 1 to 125 (word)  |
| <code>_MDB_READ_INPUT_REGISTERS</code>   | 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      |
|--|----------------|
| <code>_MDB_READ_COILS</code>             | BOOL<br>BOOL[] |
| <code>_MDB_READ_DISCRETE_INPUTS</code>   | BOOL<br>BOOL[] |
| <code>_MDB_READ_HOLDING_REGISTERS</code> | WORD<br>WORD[] |

| Function code             | Data type      |
|---------------------------|----------------|
| _MDB_READ_INPUT_REGISTERS | WORD<br>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_MODBUSRTU_OPTION | ---                   | ---    | ---     |
| SendDelay  | Send delay time | Send delay time   | UINT                      | Depends on data type. | 0.01 s | 0       |
| TimeOut    | Timeout time    | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 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       |



#### Precautions for Correct Use

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 instruction execution is completed before an attempt of interruption, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

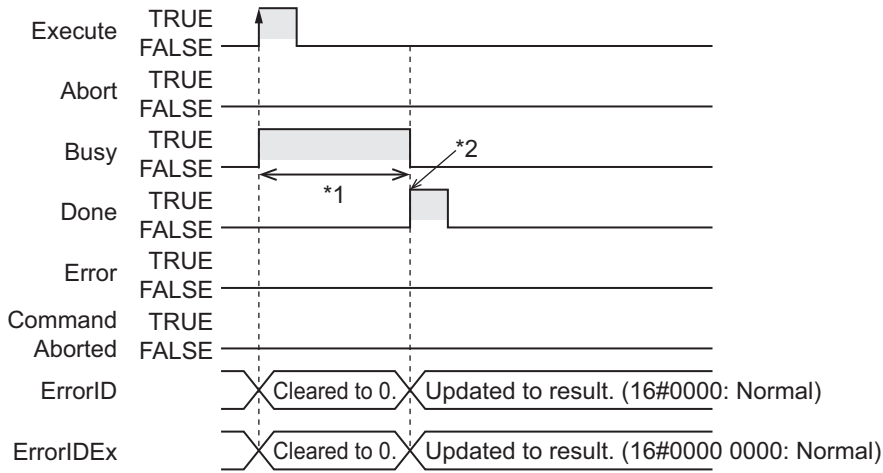
This interruption operation only finishes the *Busy* processing, and it does not clear the send or receive buffer. To clear the buffer, use the instruction, *NX\_SerialBufClear* on page 2-1409.

## Timing Charts

The following figures show the timing charts.

### ● Normal end (when *SendDelay* is 0 (0 s))

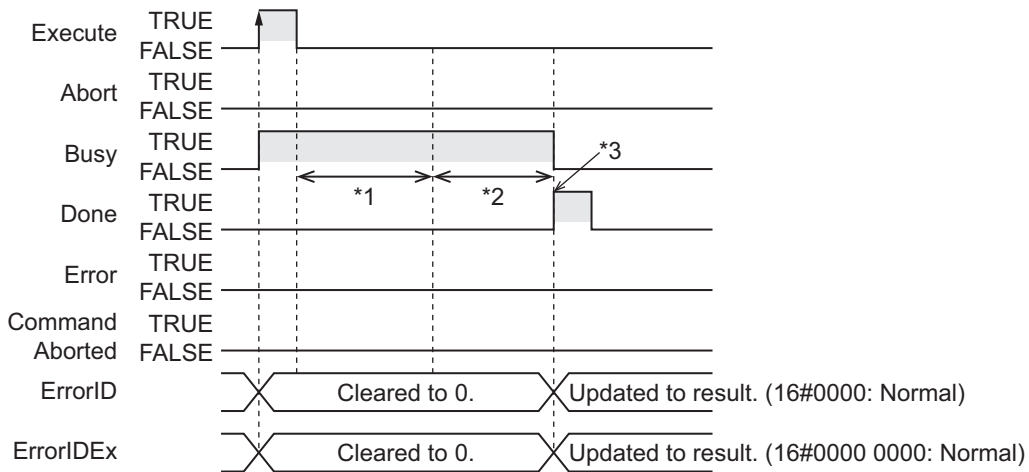
The operation is as follows when *SendDelay* is 0 (0 s).



- \*1. Processing with Modbus-RTU slave
- \*2. A response to the command is received.

● **Normal end (when *SendDelay* is 100 (1 s))**

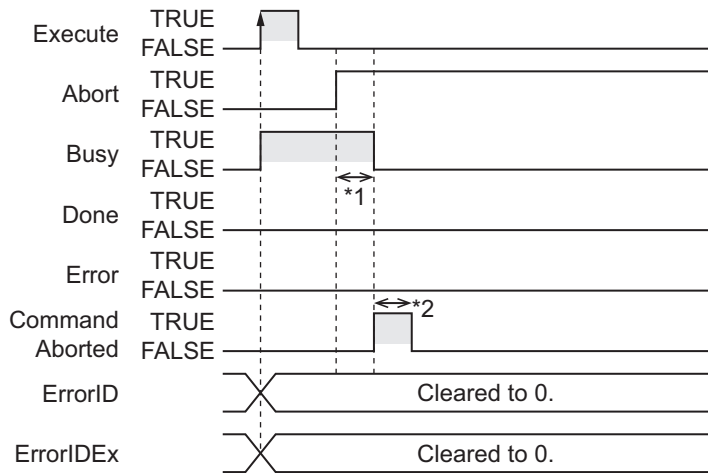
The operation is as follows when *SendDelay* is 100 (1 s).



- \*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)**

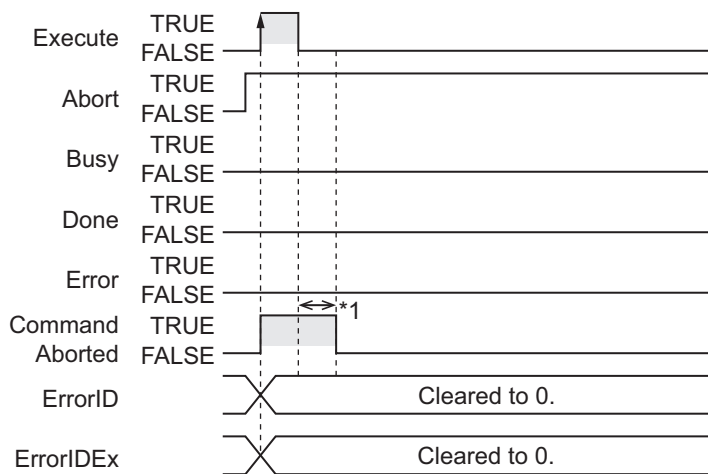
The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.



- \*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 <sup>*1</sup>     | Option Board Status            | ARRAY[1..2] of _sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOActiveTbl <sup>*2</sup> | NX Unit I/O Data Active Status | ARRAY[0..32] OF BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>• This status tells the NX Units whether I/O data communications can be processed.</li> <li>• The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

- \*1. You can use this variable only with NX1P2 CPU Units.
- \*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.
- \*3. For the NX1P2 CPU Units, the data type is ARRAY [0..8] OF BOOL.

## 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 completed even when *Execute* changes to FALSE or the execution time exceeds the task period.  
The value of *Done* changes to TRUE when the execution 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.
- A *CIF Unit Initialized* error 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 with *W* in 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 instructions: *NX\_ModbusRtuCmd* instruction, *NX\_ModbusRtuRead* instruction, or *NX\_ModbusRtuWrite* instruction.
  - a) After the operation starts or when you change the operating mode to RUN mode.
  - b) The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
  - c) The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
  - d) 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) A value that is out of range was set for *SlaveAdr*, *ReadCmd.ReadSize*, *ReadCmd.Fun*, *Option.Retry*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The variable specified for *ReadDat* is smaller than the size specified with *ReadCmd.ReadSize*.
  - c) The Unit or port specified with *DevicePort* does not exist.
  - d) The data type of *DevicePort* or *RespDat* is invalid.
  - e) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - f) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
  - g) A parity error occurred in the data received.
  - h) A framing error occurred in the data received.
  - i) An overrun error occurred in the data received.



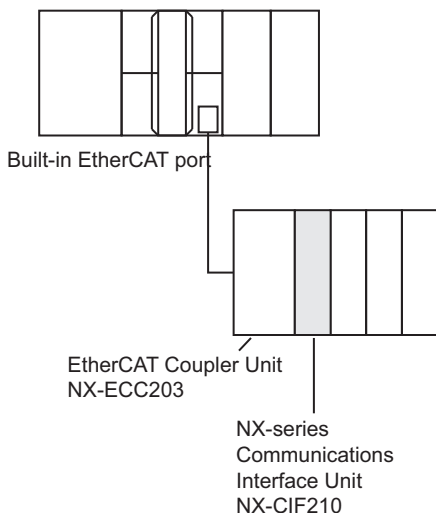
- j) CRC mismatch occurred for the received data.
  - k) Timeout time elapsed. (When the retry is set, timeout time is multiplied by the number of retries.)
  - l) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - m) An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
  - n) There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
  - o) 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=00000XX*. 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.

## ST

| 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) |
|                    | 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/<br>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

|                       | 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[]<br>(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.

|            | Boo<br>lean | Bit strings  |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1381 for details on the structure <code>_sDEVICE_PORT</code> .            |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SlaveAdr   |             |  |      |       |       |       | OK       |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| WriteCmd   |             | Refer to <i>Function</i> on page 2-1381 for details on the structure <code>_sSERIAL_MODBUSRTU_WRITE</code> . |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

|                       | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----------------------|---|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|                       | BOOL  | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL                 | LREAL | TIME   | DATE | TOD | DT | STRING |
| WriteDat[]<br>(array) | OK  |             | OK   |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
|                       | An array can also be specified.   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Option                | Refer to <i>Function</i> on page 2-1381 for details on the structure <code>_sSERIAL_MODBUSRTU_OPTION</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| Abort                 | OK  |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| CommandA-<br>borted   | 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 | De-<br>fault |
|------------|------------------------|---------------------------------------|----------------------------|--|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcat-Slave</code><br><code>_DeviceOption-Board</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---          |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>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.

Assign a variable.

| Position | Port                      | Description               | R/W | Data Type           | Variable                     |
|----------|---------------------------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | ▼ NX-CIF210               |                           |     |                     |                              |
|          | Node location information | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          | ⋮                         |                           |     |                     |                              |
|          | Ch1 Output SID            | Ch1 Output SID            | W   | USINT               |                              |
|          | Ch1 Input SID Response    | Ch1 Input SID Response    | W   | USINT               |                              |
|          | ▶ Ch1 Output Data Type    | Ch1 Output Data Type      | W   | WORD                |                              |
|          | Ch1 Output Sub Info       | Ch1 Output Sub Info       | W   | WORD                |                              |
|          | Ch1 Output Data Length    | Ch1 Output Data Length    | W   | UINT                |                              |
|          | ▶ Ch1 Output Data 01      | Ch1 Output Data 01        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 02      | Ch1 Output Data 02        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 03      | Ch1 Output Data 03        | W   | ARRAY[0..3] OF BYTE |                              |
|          | Ch1 Output Data 04        | Ch1 Output Data 04        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 05      | Ch1 Output Data 05        | W   | ARRAY[0..3] OF BYTE |                              |

Do not assign variables.

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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<br>_MDB_WRITE_SINGLE_REGISTER<br>_MDB_WRITE_MULTIPLE_COILS<br>_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                                 |
|--|---|
| <code>_MDB_WRITE_SINGLE_COIL</code>        | Write an output (bit)                   |
| <code>_MDB_WRITE_SINGLE_REGISTER</code>    | Write a holding register (word)         |
| <code>_MDB_WRITE_MULTIPLE_COILS</code>     | Write multiple outputs (bit)            |
| <code>_MDB_WRITE_MULTIPLE_REGISTERS</code> | 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        |
|--|------------------|
| <code>_MDB_WRITE_SINGLE_COIL</code>        | 1 (bit)          |
| <code>_MDB_WRITE_SINGLE_REGISTER</code>    | 1 (word)         |
| <code>_MDB_WRITE_MULTIPLE_COILS</code>     | 1 to 1,968 (bit) |
| <code>_MDB_WRITE_MULTIPLE_REGISTERS</code> | 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      |
|--|----------------|
| <code>_MDB_WRITE_SINGLE_COIL</code>        | BOOL<br>BOOL[] |
| <code>_MDB_WRITE_SINGLE_REGISTER</code>    | WORD<br>WORD[] |
| <code>_MDB_WRITE_MULTIPLE_COILS</code>     | BOOL<br>BOOL[] |
| <code>_MDB_WRITE_MULTIPLE_REGISTERS</code> | WORD<br>WORD[] |

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_MODBUSRTU_OPTION | ---                   | ---    | ---     |
| SendDelay  | Send delay time | Send delay time   | UINT                      | Depends on data type. | 0.01 s | 0       |
| TimeOut    | Timeout time    | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 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       |



#### Precautions for Correct Use

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 instruction execution is completed before an attempt of interruption, *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 instruction, *NX\_SerialBufClear* on page 2-1409.

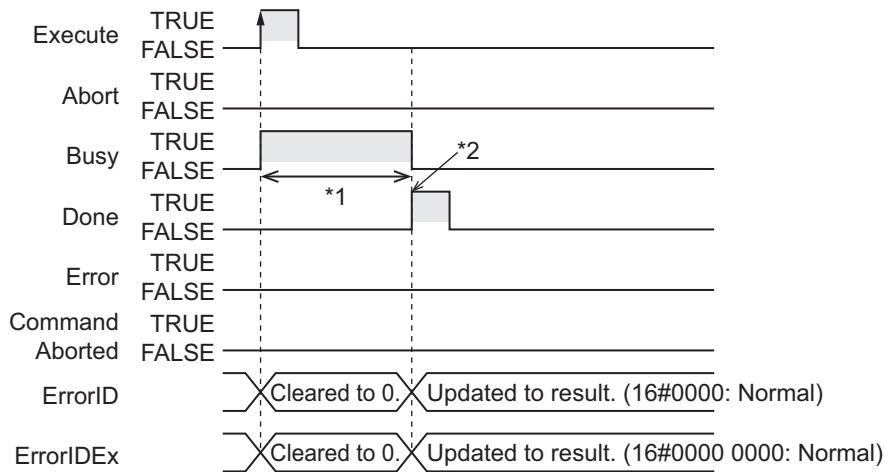
## Timing Charts

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).

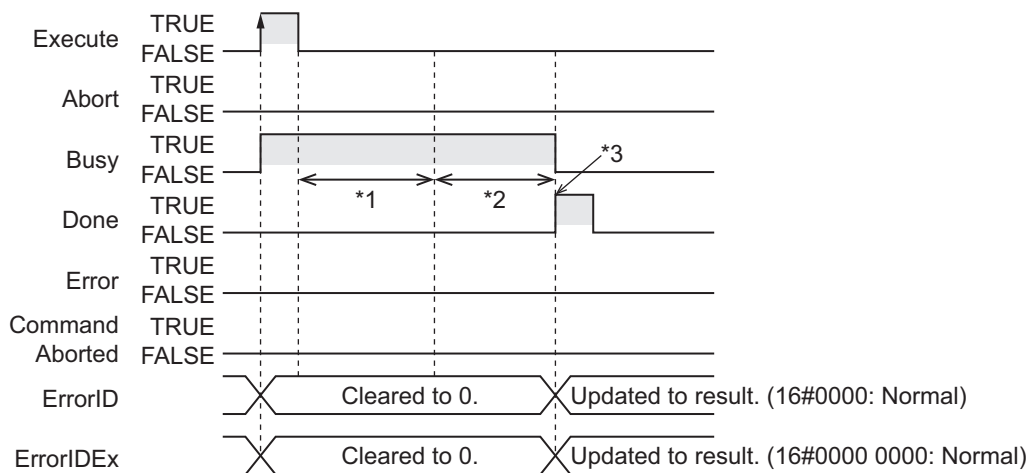




- \*1. Processing with Modbus-RTU slave
- \*2. A response to the command is received.

● Normal end (when *SendDelay* is 100 (1 s))

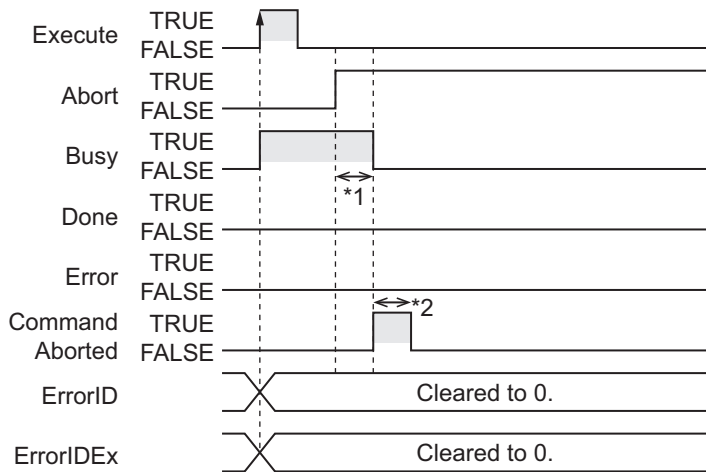
The operation is as follows when *SendDelay* is 100 (1 s).



- \*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.

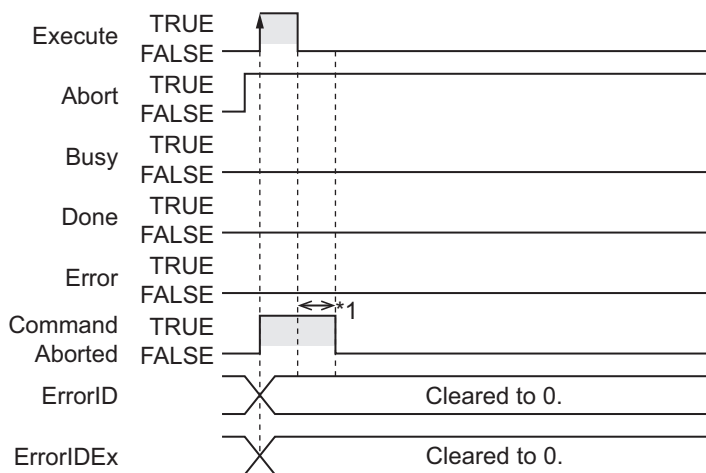


\*1. Interruption processing

\*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 <sup>*1</sup>     | Option Board Status            | ARRAY[1..2] of _sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOActiveTbi <sup>*2</sup> | NX Unit I/O Data Active Status | ARRAY[0..32] OF BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is ARRAY [0..8] OF BOOL.

## 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 completed even when *Execute* changes to FALSE or the execution time exceeds the task period.  
The value of *Done* changes to TRUE when the execution 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.
- A *CIF Unit Initialized* error 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 with *W* in 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 instructions: *NX\_ModbusRtuCmd* instruction, *NX\_ModbusRtuRead* instruction, or *NX\_ModbusRtuWrite* instruction.
  - a) After the operation starts or when you change the operating mode to RUN mode.
  - b) The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
  - c) The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
  - d) 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) A value that is out of range was set for *SlaveAdr*, *WriteCmd.Fun*, *WriteCmd.WriteSize*, *Option.Retry*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The variable specified for *WriteDat* is smaller than the size specified with *WriteCmd.WriteSize*.
  - c) The Unit or port specified with *DevicePort* does not exist.
  - d) The data type of *DevicePort* or *WriteDat* is invalid.
  - e) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - f) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
  - g) A parity error occurred in the data received.
  - h) A framing error occurred in the data received.
  - i) An overrun error occurred in the data received.

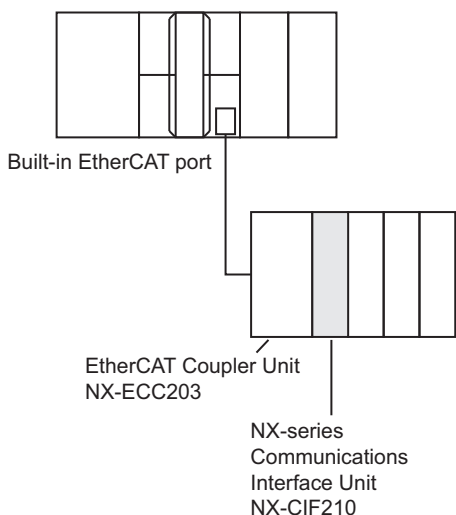
- j) CRC mismatch occurred for the received data.
  - k) Timeout time elapsed.
  - l) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - m) An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
  - n) There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
  - o) 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.

## ST

| 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 |               |                     |

| External Variables | Variable                     | Data type   | Constant                            | Comment |
|--------------------|------------------------------|-------------|-------------------------------------|---------|
|                    | N1_Node_location_information | _sNXUNIT_ID | <input checked="" type="checkbox"/> |         |

```
// 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/<br>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

|            | 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<br>_ER_SIG*1  | ---   | *2      |
| Sig        | ON/OFF command |       | ON/OFF command  | Depends on data type. | ---   | *2      |
| TimeOut    | Timeout time   |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | 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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| Kind       |             |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| 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  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---     |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOptionBoard</code> | ---  | ---     |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---     |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---     |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---     |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---     |
| PortNo     | Port number            | Port number<br>1: Port 1<br>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.



Assign a variable.

| Position Unit1 | Port      | Description               | R/W | Data Type           | Variable                     |
|----------------|-----------|---------------------------|-----|---------------------|------------------------------|
|                | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
| ⋮              |           |                           |     |                     |                              |
|                |           | Ch1 Output SID            | W   | JSINT               |                              |
|                |           | Ch1 Input SID Response    | W   | JSINT               |                              |
|                |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|                |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|                |           | Ch1 Output Data Length    | W   | UINT                |                              |
|                |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|                |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Do not assign variables.

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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   |
|----------------------|-----------|
| <code>_RS_SIG</code> | RS signal |
| <code>_ER_SIG</code> | ER signal |
| <code>_CS_SIG</code> | CS signal |
| <code>_DR_SIG</code> | DR signal |



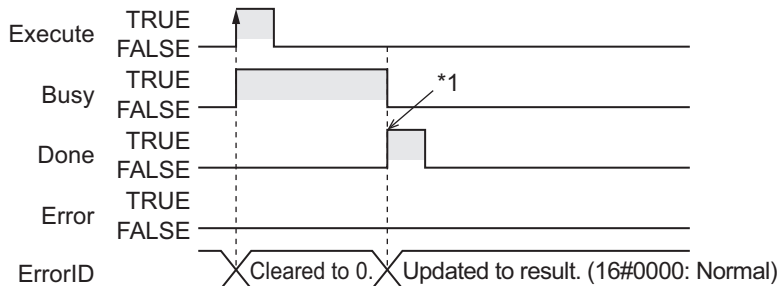
### Precautions for Correct Use

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

## Timing Charts

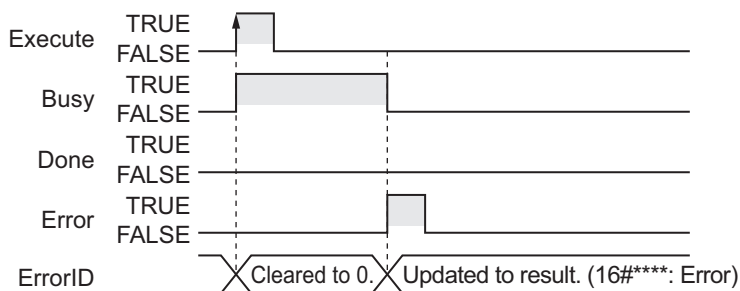
The following figures show the timing charts.

### ● Normal end



\*1. Signal ON/OFF control is completed.

### ● Error end



## Related System-defined Variables

| Name                                    | Meaning                           | Data type                             | Description  |
|---|-----------------------------------|---------------------------------------|--|
| _PLC_OptBoard-<br>Sta <sup>*1</sup>     | Option Board Sta-<br>tus          | ARRAY[1..2] of<br>_sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOAc-<br>tiveTbl <sup>*2</sup> | NX Unit I/O Data<br>Active Status | ARRAY[0..32] OF<br>BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is ARRAY [0..8] OF BOOL.

## Precautions for Correct Use

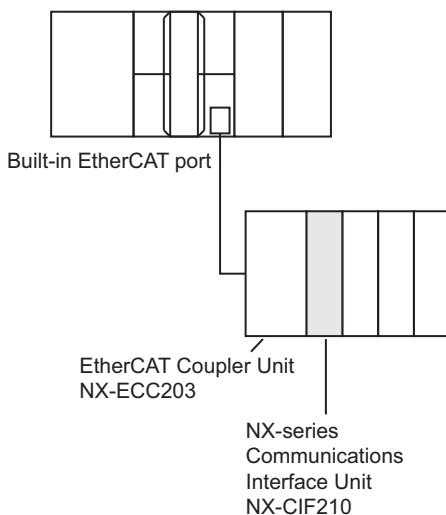
- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- A compiling error will 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. Before use, check the wiring conditions and communication protocol.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *Kind*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - c) An RS-422A/485 serial port is specified with *DevicePort*.
  - d) 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*.
  - e) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - f) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
  - g) Timeout time elapsed.
  - h) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - i) 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]<br>OF BOOL | ECAT://node#1/NX Unit<br>I/O Data Active Status<br>125 | Usage of I/O data for 63 NX Units.     |
| N1_Node_location_information           | _sNXUNIT_ID             | ---  | Device variable to specify NX-CIF210*1 |

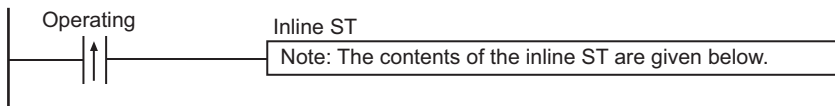
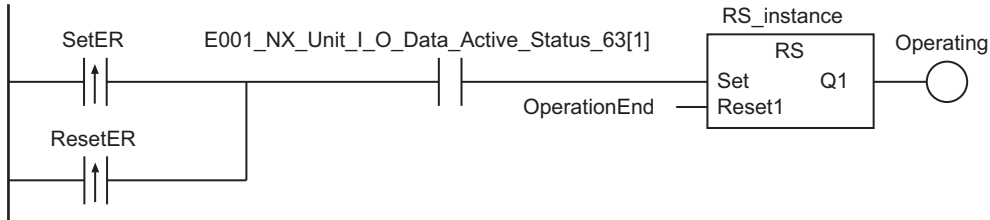
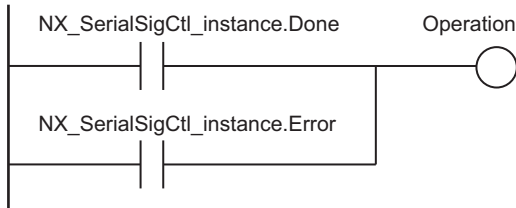
- \*1. On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for details.

## LD

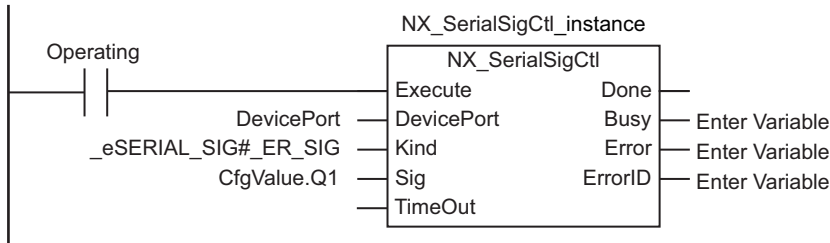
| Internal Variables | Variable                 | Data type       | Initial value | Comment  |
|--------------------|--------------------------|-----------------|---------------|--|
|                    | OperationEnd             | BOOL            | FALSE         | Processing completed                               |
|                    | 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              | ---           | <i>Operating</i> retained                          |
|                    | CfgValue                 | RS              | ---           | Value determined by <i>SetER</i> or <i>ResetER</i> |
|                    | NX_SerialSigCtl_instance | NX_SerialSigCtl | ---           |  |

| External Variables | Variable                               | Data type               | Comment  |
|--------------------|--|-------------------------|--|
|                    | E001_NX_Unit_I_O_Data_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | N1_Node_location_information           | _sNXUNIT_ID             | Device variable to specify NX-CIF210   |

Determine if execution of the NX\_SerialSigCtl instruction has ended.



Execute NX\_SerialSigCtl instruction.



### ● 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 |

| Internal Variables | Variable                 | Data type       | Initial value | Comment  |
|--------------------|--------------------------|-----------------|---------------|--|
|                    | DevicePort               | _sDEVICE_PORT   |               | Port settings                                      |
|                    | CfgValue                 | RS              | ---           | Value determined by <i>SetER</i> or <i>ResetER</i> |
|                    | NX_SerialSigCtl_instance | NX_SerialSigCtl | ---           |  |

| External Variables | Name                                   | Data type               | Comment  |
|--------------------|--|-------------------------|--|
|                    | E001_NX_Unit_I_O_Data_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | 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_Status_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\_SerialSigRead

The NX\_SerialSigRead instruction reads the CS or DR signal of a serial port on an Option Board.

| Instruction           | Name                          | FB/<br>FUN | Graphic expression | ST expression   |
|-----------------------|-------------------------------|------------|--------------------|---|
| NX_SerialSig-<br>Read | Read Serial<br>Control Signal | FB         |                    | NX_SerialSigRead_instance(Execute, DevicePort, Kind, TimeOut, Done, Busy, Error, ErrorID, Sig); |



## Precautions for Correct Use

You can use this instruction for an Option Board for the NX1P2 CPU Unit only.

## Variables

|            | Meaning        | I/O    | Description   | Valid range           | Unit  | Default |
|------------|----------------|--------|---|-----------------------|-------|---------|
| DevicePort | Device port    | Input  | Object that represents a device port                    | ---                   | ---   | ---     |
| Kind       | Signal command |        | Signal command  | _CS_SIG<br>_DR_SIG*1  | ---   | *2      |
| TimeOut    | Timeout time   |        | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | Depends on data type. | 0.1 s | 0       |
| Sig        | Signal         | Output | Outputs the signal that was read.                       | Depends on data type. | ---   | ---     |

\*1. You cannot use \_RS\_SIG or \_ER\_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.

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 |             |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| Kind       |             |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| TimeOut    |             |             |      |       |       |          | OK   |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| Sig        | OK          |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |

## Function

The NX\_SerialSigRead instruction reads the CS or DR signal of a serial port on an Option Board.

If the signal that was read is ON, the Sig output variable is TRUE. If the signal is OFF, Sig is FALSE.

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 | De-<br>fault |
|------------|------------------------|---------------------------------------|----------------------------|--|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcat-Slave</code><br><code>_DeviceOption-Board</code> | ---  | ---          |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---          |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---          |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---          |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---          |
| PortNo     | Port number            | Port number<br>1: Port 1<br>2: Port 2 | USINT                      | Depends on data type.  | ---  | ---          |

Use *DeviceType* to specify the device type.

For an Option Board, set this to `_DeviceOptionBoard`.

The variable used to specify the device is determined by the specified device type.

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.

The figure shows a table representing an I/O Map configuration. The top row is for 'Unit1' and 'Port' 'NX-CIF210'. The first row in the table is 'Node location information' with a description of 'Node location information', R/W 'R', Data Type '`_sNXUNIT_ID`', and Variable '`N1_Node_location_information`'. A red arrow points to this variable cell with the text 'Assign a variable.'. Below this, there are several rows for 'Ch1 Output SID' and 'Ch1 Input SID Response'. The 'Ch1 Output SID' row has R/W 'W', Data Type 'USINT', and an empty Variable cell. A red arrow points to this empty cell with the text 'Do not assign variables.'. Below this are rows for 'Ch1 Output Data Type', 'Ch1 Output Sub Info', 'Ch1 Output Data Length', and five 'Ch1 Output Data' rows (01-05). All these rows have R/W 'W' and Data Types 'WORD', 'WORD', 'UINT', and 'ARRAY[0..3] OF BYTE' respectively. All Variable cells for these rows are empty.



Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | Option Board is specified.   |

In this instruction, you can specify `_DeviceOptionBoard`.

Use the *Kind* input variable to select the CS or DR signal.

The data type of *Kind* is enumerated type `_eSERIAL_SIG`.

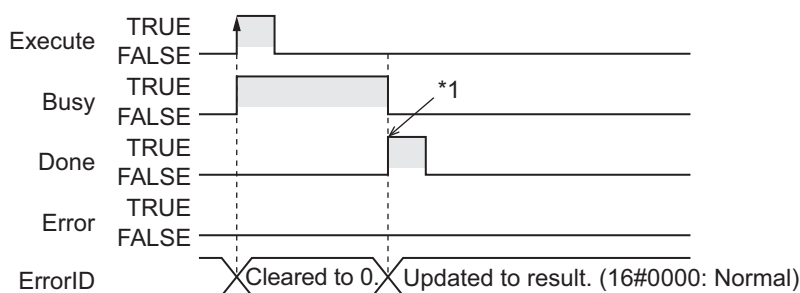
The meanings of the enumerators of enumerated type `_eSERIAL_SIG` are as follows:

| Enumerator           | Meaning   |
|----------------------|-----------|
| <code>_RS_SIG</code> | RS signal |
| <code>_ER_SIG</code> | ER signal |
| <code>_CS_SIG</code> | CS signal |
| <code>_DR_SIG</code> | DR signal |

## Timing Charts

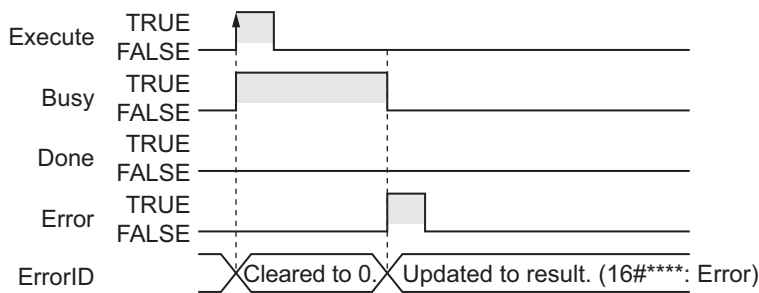
The following figures show the timing charts.

### ● Normal end



\*1. Signal reading is completed.

### ● Error end



### Related System-defined Variables

| Name             | Meaning             | Data type                      | Description                                 |
|------------------|---------------------|--------------------------------|---|
| _PLC_OptBoardSta | Option Board Status | ARRAY[1..2] of _sOPT-BOARD_STA | This stores the status of the Option Board. |

### Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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. Before use, check the wiring conditions and communication protocol.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *Kind*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - c) An RS-422A/485 serial port is specified with *DevicePort*.
  - d) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - e) This instruction is executed with a device port variable that is the same as the one specified for another instruction that is still being executed.  
In this case, the instruction which is still being executed is one of the followings: the *NX\_SerialSigCtl* instruction, *NX\_SerialSigRead* instruction, *NX\_SerialStatusRead* instruction, *NX\_SerialBufClear* instruction, *NX\_SerialStartMon* instruction, and *NX\_SerialStopMon* instruction.
  - f) Timeout time elapsed.
  - g) The instruction is executed for other than Option Boards.
  - h) The serial communications mode of the specified Option Board is not *No-protocol* or *Modbus-RTU master*.

## Sample Programming

Refer to *Sample Programming* on page 2-1395 for the NX\_SerialSigCtl instruction.

# NX\_SerialStatusRead

The NX\_SerialStatusRead instruction reads the status of a serial port on an Option Board.

| Instruction         | Name                    | FB/<br>FUN | Graphic expression | ST expression   |
|---------------------|-------------------------|------------|--------------------|---|
| NX_SerialStatusRead | Read Serial Port Status | FB         |                    | NX_SerialStatusRead_instance(Execute, DevicePort, TimeOut, Done, Busy, Error, ErrorID, PortStatus); |



## Precautions for Correct Use

You can use this instruction for an Option Board for the NX1P2 CPU Unit only.

## Variables

|            | Meaning      | I/O    | Description   | Valid range           | Unit  | Default |
|------------|--------------|--------|---|-----------------------|-------|---------|
| DevicePort | Device port  | Input  | Object that represents a device port                    | ---                   | ---   | ---     |
| TimeOut    | Timeout time |        | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | Depends on data type. | 0.1 s | 0       |
| PortStatus | Port status  | Output | Outputs the port status that was read.                  | ---                   | ---   | ---     |

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1404 for details on the structure <code>_sDEVICE_PORT</code> .        |
| TimeOut    |             |             |      |       |       |          | OK   |       |       |      |     |      |      |                      |       |  |      |     |    |        |  |
| PortStatus |             |             |      |       |       |          |      |       |       |      |     |      |      |                      |       |  |      |     |    |        | Refer to <i>Function</i> on page 2-1404 for details on the structure <code>_sSERIAL_PORT_STATUS</code> . |

## Function

The NX\_SerialStatusRead instruction reads the status of a serial port on an 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<br>_DeviceEcat-Slave<br>_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<br>1: Port 1<br>2: Port 2 | USINT         | Depends on data type.                                     | ---  | ---     |

Use *DeviceType* to specify the device type.

For an Option Board, set this to `_DeviceOptionBoard`.

The variable used to specify the device is determined by the specified device type.

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.

| Position | Port                      | Description               | R/W | Data Type           | Variable                     |
|----------|---------------------------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | ▼ NX-CIF210               |                           |     |                     |                              |
|          | Node location information | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          | ...                       |                           |     |                     |                              |
|          | Ch1 Output SID            | Ch1 Output SID            | W   | USINT               |                              |
|          | Ch1 Input SID Response    | Ch1 Input SID Response    | W   | USINT               |                              |
|          | ▶ Ch1 Output Data Type    | Ch1 Output Data Type      | W   | WORD                |                              |
|          | Ch1 Output Sub Info       | Ch1 Output Sub Info       | W   | WORD                |                              |
|          | Ch1 Output Data Length    | Ch1 Output Data Length    | W   | UINT                |                              |
|          | ▶ Ch1 Output Data 01      | Ch1 Output Data 01        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 02      | Ch1 Output Data 02        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 03      | Ch1 Output Data 03        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 04      | Ch1 Output Data 04        | W   | ARRAY[0..3] OF BYTE |                              |
|          | ▶ Ch1 Output Data 05      | Ch1 Output Data 05        | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | Option Board is specified.   |

In this instruction, you can specify `_DeviceOptionBoard`.

The data type of *PortStatus* output variable is structure `_sSERIAL_PORT_STATUS`.

The specifications are as follows:

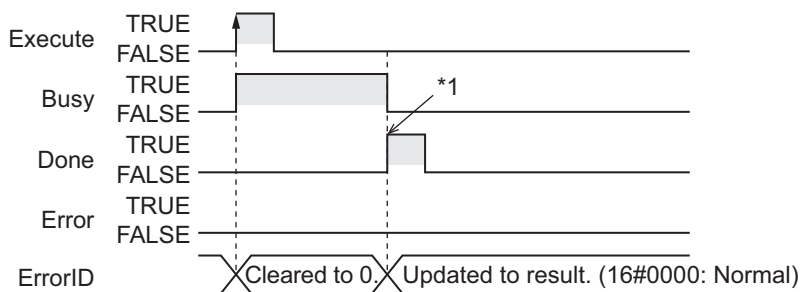
| Name       | Meaning                                   | Description   | Data type                         | Valid range           | Unit | De-<br>fault |
|------------|---|---|-----------------------------------|-----------------------|------|--------------|
| PortStatus | Port status                               | Outputs the port status that was read.                        | <code>_sSERIAL_PORT_STATUS</code> | ---                   | ---  | ---          |
| FullRcvBuf | Data discarded due to full receive buffer | TRUE: Data was discarded.*1<br>FALSE: Data was not discarded. | BOOL                              | Depends on data type. | ---  | ---          |
| Reserved   | Reserved                                  | Reserved  | Reserved                          | ---                   | ---  | ---          |

\*1. Data In the receive buffer may not be complete.

## Timing Charts

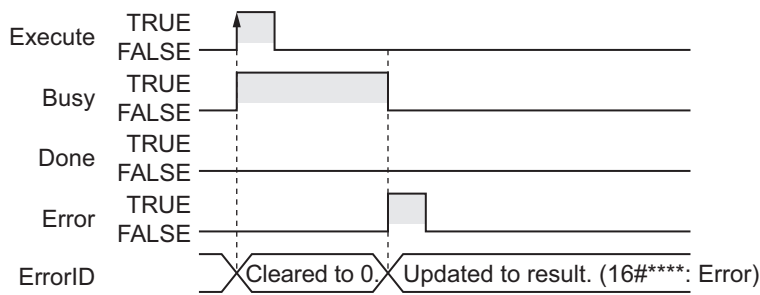
The following figures show the timing charts.

### ● Normal end



\*1. Port status reading is completed.

### ● Error end



## Related System-defined Variables

| Name             | Meaning             | Data type                      | Description                                 |
|------------------|---------------------|--------------------------------|---|
| _PLC_OptBoardSta | Option Board Status | ARRAY[1..2] of _sOPT-BOARD_STA | This stores the status of the Option Board. |

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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. Before use, check the wiring conditions and communication protocol.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *Kind*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - c) More than 32 of the following instructions were executed at the same time: NX\_SerialSend, NX\_SerialRcv, NX\_ModbusRtuCmd, NX\_ModbusRtuRead, NX\_ModbusRtuWrite, NX\_SerialSigCtl, NX\_SerialSigRead, NX\_SerialStatusRead, NX\_SerialBufClear, NX\_SerialStartMon, and NX\_SerialStopMon.
  - d) This instruction is executed with a device port variable that is the same as the one specified for another instruction that is still being executed.  
In this case, the instruction which is still being executed is one of the followings: the NX\_SerialSigCtl instruction, NX\_SerialSigRead instruction, NX\_SerialStatusRead instruction, NX\_SerialBufClear instruction, NX\_SerialStartMon instruction, and NX\_SerialStopMon instruction.
  - e) Timeout time elapsed.
  - f) The instruction is executed for other than Option Boards.
  - g) The serial communications mode of the specified Option Board is not *No-protocol* or *Modbus-RTU master*.

## Sample Programming

Refer to *Sample Programming* on page 2-1395 for the NX\_SerialSigCtl instruction.



# NX\_SerialBufClear

The NX\_SerialBufClear instruction clears the send or receive buffer.

| Instruction       | Name         | FB/<br>FUN | Graphic expression | ST expression  |
|-------------------|--------------|------------|--------------------|--|
| NX_SerialBufClear | Clear Buffer | FB         |                    | 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

|            | 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_SENDR<br>CV<br>_BUF_SEND<br>_BUF_RCV | ---   | _BUF_SENDR<br>CV |
| TimeOut    | Timeout time |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | Depends on data type.                     | 0.1 s | 0                |

|            | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num<br>bers |       | Times, durations,<br>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 |             |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| BufKind    |             |             |      |       |       |          |      |       |       |      |     |      |      |                     |       |  |      |     |    |        |
| 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  | _sDEVICE_PORT | ---   | ---  | ---     |
| DeviceType | Device type            | Type of the device to specify         | _eDEVICE_TYPE | _DeviceNXUnit<br>_DeviceEcat-Slave<br>_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<br>1: Port 1<br>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.

| Position | Port      | Description               | R/W | Data Type           | Variable                     |
|----------|-----------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | NX-CIF210 | Node location information | R   | sNXUNIT_ID          | N1_Node_location_information |
|          |           | Ch1 Output SID            | W   | JSINT               |                              |
|          |           | Ch1 Input SID Response    | W   | JSINT               |                              |
|          |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|          |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |           | Ch1 Output Data Length    | W   | UINT                |                              |
|          |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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                        |
|---------------------------|--------------------------------|
| <code>_BUF_SENDRCV</code> | Send buffer and receive buffer |
| <code>_BUF_SEND</code>    | Send buffer                    |
| <code>_BUF_RCV</code>     | Receive buffer                 |



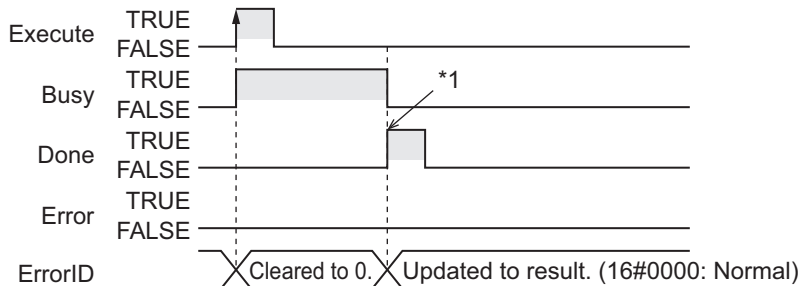
**Precautions for Correct Use**

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

## Timing Charts

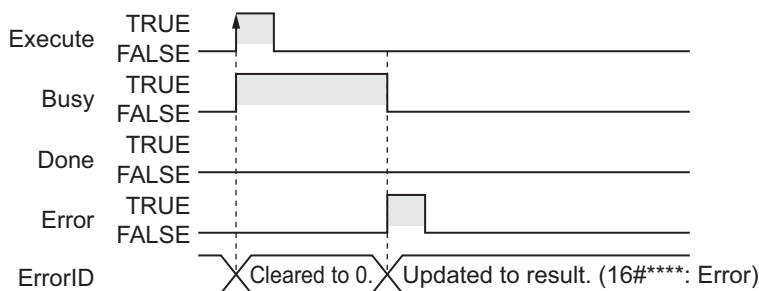
The following figures show the timing charts.

### ● Normal end



\*1. Buffer clear processing is completed.

### ● Error end



## Related System-defined Variables

| Name                                    | Meaning                           | Data type                             | Description  |
|---|-----------------------------------|---------------------------------------|--|
| _PLC_OptBoard-<br>Sta <sup>*1</sup>     | Option Board Sta-<br>tus          | ARRAY[1..2] of<br>_sOPTBOARD_STA      | This stores the status of the Option Board.  |
| _NXB_UnitIOAc-<br>tiveTbl <sup>*2</sup> | NX Unit I/O Data<br>Active Status | ARRAY[0..32] OF<br>BOOL <sup>*3</sup> | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX1P2 CPU Units.

\*2. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*3. For the NX1P2 CPU Units, the data type is ARRAY [0..8] OF BOOL.

## Precautions for Correct Use

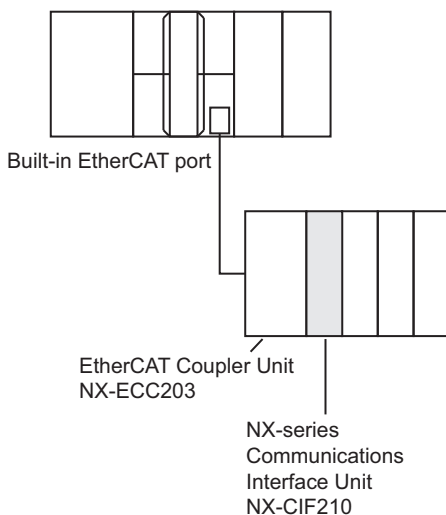
- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- A compiling error will 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. Before use, check the wiring conditions and communication protocol.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *BufKind*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - c) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - d) This instruction is executed with a device port variable that is the same as the one specified for another instruction that 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.
  - e) Timeout time elapsed.
  - f) This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - g) 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)* for details.

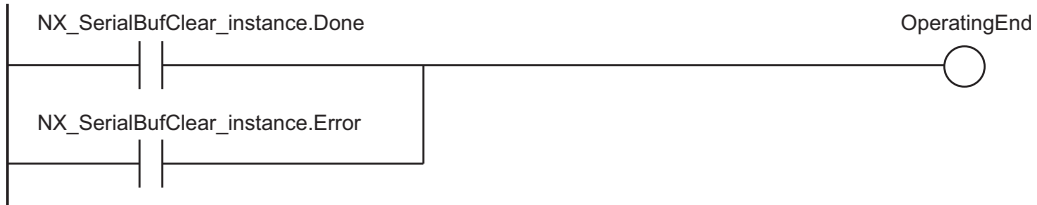
## LD

| 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_SerialBuf-Clear_instance | NX_SerialBufClear   | ---                |                                     |
|                    | RcvngEnd                    | BOOL                |                    | Receive processing completed        |
|                    | Rcvng                       | 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 |                    |                                     |

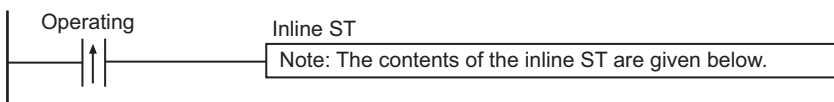
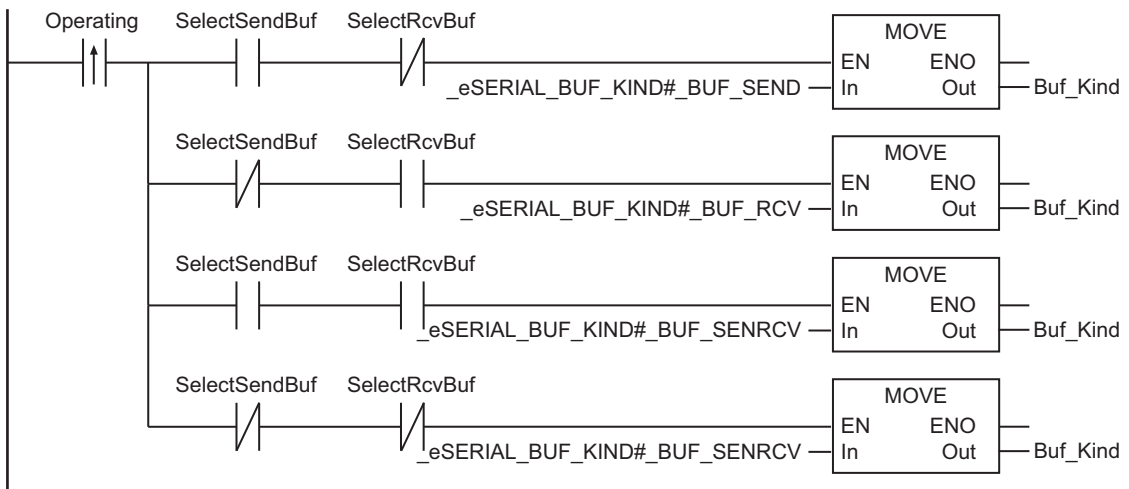
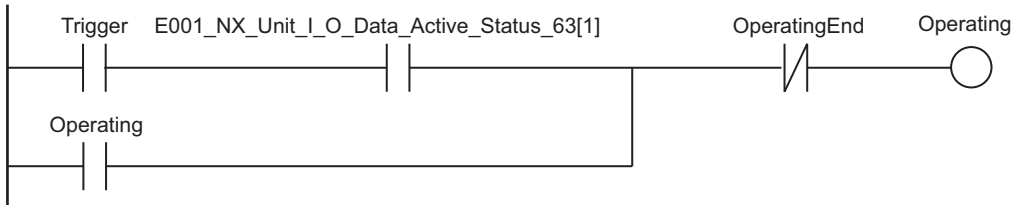
| Internal Variables | Variable   | Data type | Initial value | Comment |
|--------------------|------------|-----------|---------------|---------|
|                    | TimeOut    | TIME      | TIME#0 s      |         |
|                    | LastDatRcv | BOOL      | FALSE         |         |
|                    | ClearBuf   | BOOL      | FALSE         |         |

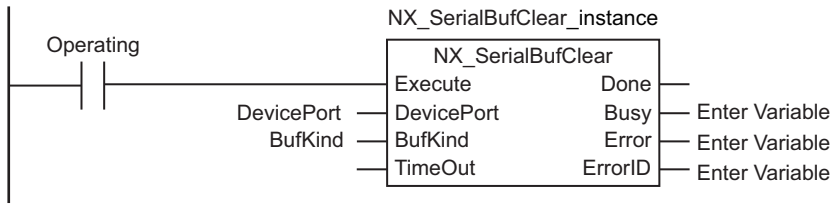
| External Variables | Variable                               | Data type               | Comment  |
|--------------------|--|-------------------------|--|
|                    | E001_NX_Unit_I_O_Data_Active_Status_63 | ARRAY[0..63]<br>OF BOOL | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | N1_Node_location_information           | _sNXUNIT_ID             | Device variable to specify NX-CIF210   |

Determine if execution of the NX\_SerialBufClear instruction has ended.

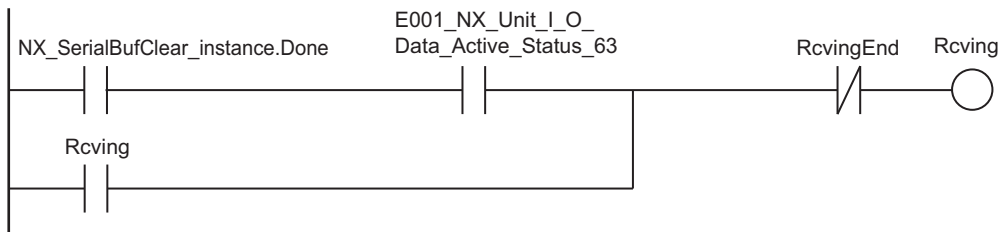
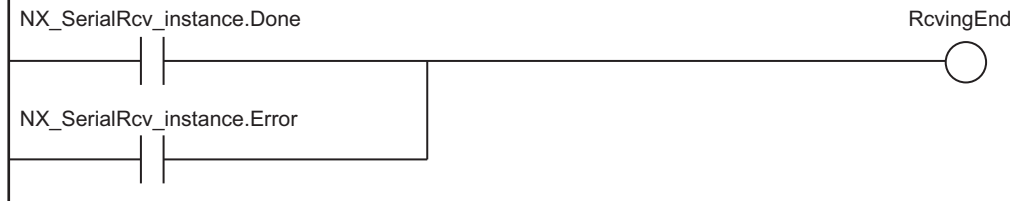


Accept trigger.

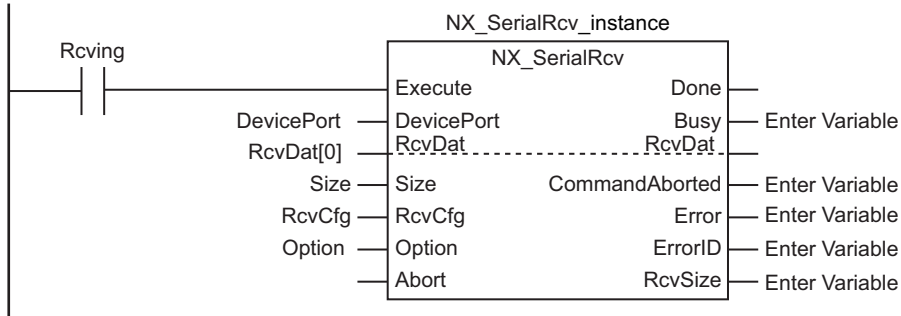




Determine if execution of the NX\_SerialRcv instruction has ended.



Execute NX\_SerialRcv instruction.



● 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 |



| Internal Variables | Variable                   | Data type           | Initial value      | Comment                        |
|--------------------|----------------------------|---------------------|--------------------|--------------------------------|
|                    | 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#0 s           |                                |
|                    | 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 | <ul style="list-style-type: none"> <li>Usage of I/O data for 63 NX Units.</li> <li>If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.</li> </ul> |
|                    | N1_Node_location_information           | _sNXUNIT_ID          | Device variable to specify NX-CIF210   |

```
// Condition setting
RS_instancel(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/<br>FUN | Graphic expression | ST expression   |
|-------------------|------------------------------|------------|--------------------|---|
| NX_SerialStartMon | Start Serial Line Monitoring | 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

|            | Meaning               | I/O   | Description   | Valid range           | Unit  | Default |
|------------|-----------------------|-------|---|-----------------------|-------|---------|
| DevicePort | Device port           | Input | Object that represents a device port  | ---                   | ---   | ---     |
| Continuous | Continuous monitoring |       | Serial line monitor operation method<br>TRUE: Continuous<br>FALSE: One-shot | Depends on data type. | ---   | FALSE   |
| TimeOut    | Timeout time          |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s.                     | Depends on data type. | 0.1 s | 0       |

|            | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1420 for details on the structure <code>_sDEVICE_PORT</code> . |             |      |       |       |          |      |       |       |      |     |      |                     |      |  |      |      |     |    |        |
| 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  | <code>_sDEVICE_PORT</code> | ---  | ---  | ---     |
| DeviceType | Device type            | Type of the device to specify         | <code>_eDEVICE_TYPE</code> | <code>_DeviceNXUnit</code><br><code>_DeviceEcatSlave</code><br><code>_DeviceOptionBoard</code> | ---  | ---     |
| NxUnit     | Specified Unit         | NX Unit to control                    | <code>_sNXUNIT_ID</code>   | ---  | ---  | ---     |
| EcatSlave  | Specified slave        | EtherCAT slave to control             | <code>_sECAT_ID</code>     | ---  | ---  | ---     |
| OptBoard   | Specified Option Board | Option Board to control               | <code>_sOPTBOARD_ID</code> | ---  | ---  | ---     |
| Reserved   | Reserved               | Reserved                              | Reserved                   | ---  | ---  | ---     |
| PortNo     | Port number            | Port number<br>1: Port 1<br>2: Port 2 | USINT                      | Depends on data type.  | ---  | ---     |

Use *DeviceType* to specify the device type.

Set this to `_DeviceNXUnit` for an NX Unit.

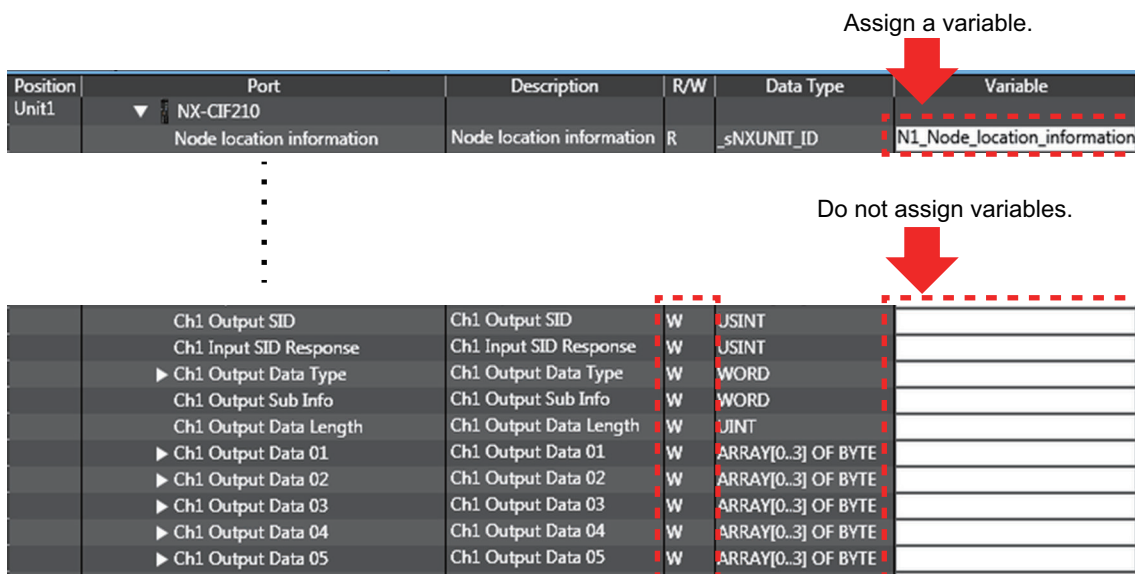
The variable used to specify the device is determined by the specified device type.

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.



| Position | Port      | Description               | R/W | Data Type           | Variable                     |
|----------|-----------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          |           | Ch1 Output SID            | W   | JSINT               |                              |
|          |           | Ch1 Input SID Response    | W   | JSINT               |                              |
|          |           | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|          |           | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |           | Ch1 Output Data Length    | W   | UINT                |                              |
|          |           | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 02      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 04      | W   | ARRAY[0..3] OF BYTE |                              |
|          |           | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | 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.



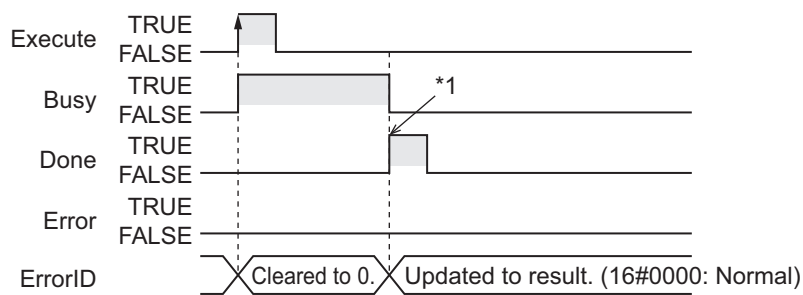
**Precautions for Correct Use**

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units.

## Timing Charts

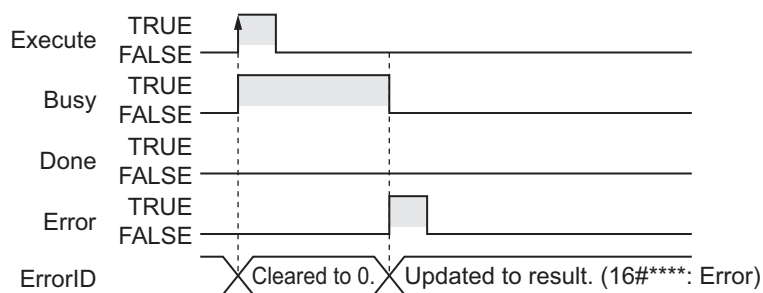
The following figures show the timing charts.

### ● Normal end



\*1. Serial line monitoring is started.

### ● Error end



## Related System-defined Variables

| Name                   | Meaning                        | Data type              | Description  |
|------------------------|--------------------------------|------------------------|--|
| _NXB_UnitIOActiveTbl*1 | NX Unit I/O Data Active Status | ARRAY[0..32] OF BOOL*2 | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*2. For the NX1P2 CPU Units, the data type is *ARRAY [0..8] OF BOOL*.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *DevicePort.DevicePortType* or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.

- c) More than 32 of the following instructions were executed at the same time: NX\_SerialSend, NX\_SerialRcv, NX\_ModbusRtuCmd, NX\_ModbusRtuRead, NX\_ModbusRtuWrite, NX\_SerialSigCtl, NX\_SerialSigRead, NX\_SerialStatusRead, NX\_SerialBufClear, NX\_SerialStartMon, and NX\_SerialStopMon.
- d) 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\_SerialSigCtl instruction, NX\_SerialSigRead instruction, NX\_SerialStatusRead instruction, NX\_SerialBufClear instruction, NX\_SerialStartMon instruction, and NX\_SerialStopMon instruction.
- e) Timeout time elapsed.
- f) 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/<br>FUN | Graphic expression | ST expression  |
|-----------------------|--------------------------------|------------|--------------------|--|
| NX_Serial-<br>StopMon | Stop Serial Line<br>Monitoring | FB         |                    | 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

|            | Meaning      | I/O   | Description   | Valid range           | Unit  | Default |
|------------|--------------|-------|---|-----------------------|-------|---------|
| DevicePort | Device port  | Input | Object that represents a device port                    | ---                   | ---   | ---     |
| TimeOut    | Timeout time |       | Timeout time<br>If 0 is set, the timeout time is 2.0 s. | Depends on data type. | 0.1 s | 0       |

|            | Boo<br>lean | Bit strings   |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1424 for details on the structure <code>_sDEVICE_PORT</code> . |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| 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 | De-<br>fault |
|------------|------------------------|---------------------------------------|---------------|---|------|--------------|
| DevicePort | Device port            | Object that represents a device port  | _sDEVICE_PORT | ---   | ---  | ---          |
| DeviceType | Device type            | Type of the device to specify         | _eDEVICE_TYPE | _DeviceNXUnit<br>_DeviceEcat-Slave<br>_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<br>1: Port 1<br>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.

Assign a variable.

| Position | Port        | Description               | R/W | Data Type           | Variable                     |
|----------|-------------|---------------------------|-----|---------------------|------------------------------|
| Unit1    | ▼ NX-CIF210 | Node location information | R   | _sNXUNIT_ID         | N1_Node_location_information |
|          |             | ...                       |     |                     |                              |
|          |             | Ch1 Output SID            | W   | USINT               |                              |
|          |             | Ch1 Input SID Response    | W   | USINT               |                              |
|          |             | ▶ Ch1 Output Data Type    | W   | WORD                |                              |
|          |             | Ch1 Output Sub Info       | W   | WORD                |                              |
|          |             | Ch1 Output Data Length    | W   | UINT                |                              |
|          |             | ▶ Ch1 Output Data 01      | W   | ARRAY[0..3] OF BYTE |                              |
|          |             | Ch1 Output Data 02        | W   | ARRAY[0..3] OF BYTE |                              |
|          |             | ▶ Ch1 Output Data 03      | W   | ARRAY[0..3] OF BYTE |                              |
|          |             | Ch1 Output Data 04        | W   | ARRAY[0..3] OF BYTE |                              |
|          |             | ▶ Ch1 Output Data 05      | W   | ARRAY[0..3] OF BYTE |                              |

Do not assign variables.

Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* 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                      |
|---------------------------------|------------------------------|
| <code>_DeviceNXUnit</code>      | NX Unit is specified.        |
| <code>_DeviceEcatSlave</code>   | EtherCAT slave is specified. |
| <code>_DeviceOptionBoard</code> | Option Board is specified.   |

In this instruction, you can specify `_DeviceNXUnit`.



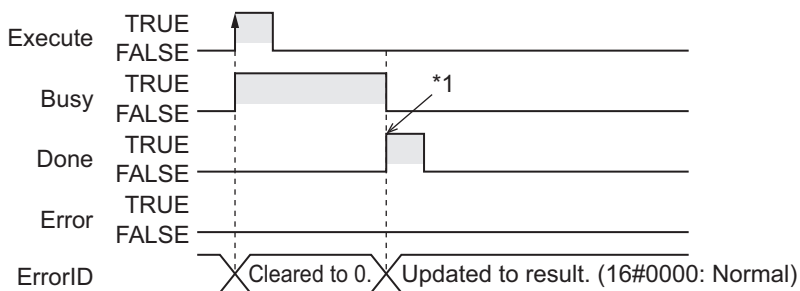
### Precautions for Correct Use

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units.

## Timing Charts

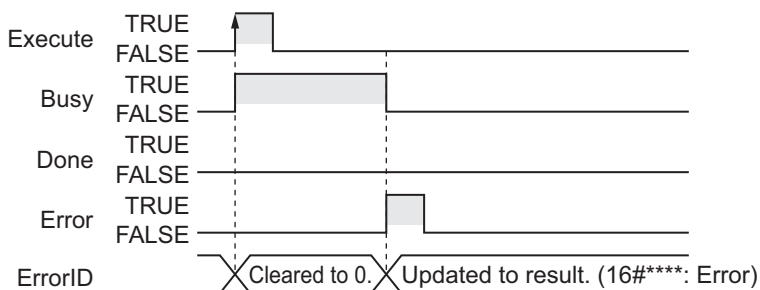
The following figures show the timing charts.

### ● Normal end



\*1. Serial line monitoring is stopped.

### ● Error end



## Related System-defined Variables

| Name                    | Meaning                        | Data type              | Description  |
|-------------------------|--------------------------------|------------------------|--|
| _NXB_UnitI/OActiveTbl*1 | NX Unit I/O Data Active Status | ARRAY[0..32] OF BOOL*2 | <ul style="list-style-type: none"> <li>This status tells the NX Units whether I/O data communications can be processed.</li> <li>The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.</li> </ul> |

\*1. You can use this variable only with NX102 CPU Units and NX1P2 CPU Units.

\*2. For the NX1P2 CPU Units, the data type is *ARRAY [0..8] OF BOOL*.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution 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.
- A *CIF Unit Initialized* error 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 with *W* in 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) A value that is out of range was set for *DevicePort.DevicePortType* or *DevicePort.PortNo*.
  - b) The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - c) More than 32 of the following instructions were executed at the same time: *NX\_SerialSend*, *NX\_SerialRcv*, *NX\_ModbusRtuCmd*, *NX\_ModbusRtuRead*, *NX\_ModbusRtuWrite*, *NX\_SerialSigCtl*, *NX\_SerialSigRead*, *NX\_SerialStatusRead*, *NX\_SerialBufClear*, *NX\_SerialStartMon*, and *NX\_SerialStopMon*.
  - d) This instruction is executed with a device port variable that is the same as the one specified for another instruction that is still being executed.  
In this case, the instruction which is still being executed is one of the followings: the *NX\_SerialSigCtl* instruction, *NX\_SerialSigRead* instruction, *NX\_SerialStatusRead* instruction, *NX\_SerialBufClear* instruction, *NX\_SerialStartMon* instruction, and *NX\_SerialStopMon* instruction.
  - e) Timeout time elapsed.
  - f) This instruction is executed for Units other than NX-series Communications Interface Units.



# SD Memory Card Instructions

| Instruction        | Name                    | Page        |
|--------------------|-------------------------|-------------|
| FileWriteVar       | Write Variable to File  | page 2-1430 |
| FileReadVar        | Read Variable from File | page 2-1436 |
| FileOpen           | Open File               | page 2-1441 |
| FileClose          | Close File              | page 2-1445 |
| FileSeek           | Seek File               | page 2-1448 |
| FileRead           | Read File               | page 2-1451 |
| FileWrite          | Write File              | page 2-1459 |
| FileGets           | Get Text String         | page 2-1467 |
| FilePuts           | Put Text String         | page 2-1475 |
| FileCopy           | Copy File               | page 2-1484 |
| FileRemove         | Delete File             | page 2-1492 |
| FileRename         | Change File Name        | page 2-1497 |
| DirCreate          | Create Directory        | page 2-1503 |
| DirRemove          | Delete Directory        | page 2-1506 |
| BackupToMemoryCard | SD Memory Card Backup   | page 2-1509 |

# 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/<br>FUN | Graphic expression   | ST expression  |
|--------------|------------------------|------------|--|--|
| FileWriteVar | Write Variable to File | FB         | <pre> FileWriteVar_instance ┌───────────────────┐ │ FileWriteVar      │ ├───────────────────┤ │ Execute   Done   │ ├──────────┴────────┤ │ FileName  Busy   │ ├──────────┴────────┤ │ WriteVar  Error  │ ├──────────┴────────┤ │ OverWrite ErrorID│ └───────────────────┘ </pre> | FileWriteVar_instance(Execute, FileName, WriteVar, OverWrite, Done, Busy, Error, ErrorID); |

## Variables

|           | 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.  |      | *1      |
| OverWrite | Overwrite enable |       | TRUE: Enable over-write.<br>FALSE: Prohibit over-write. |  |      | FALSE   |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|           | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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.

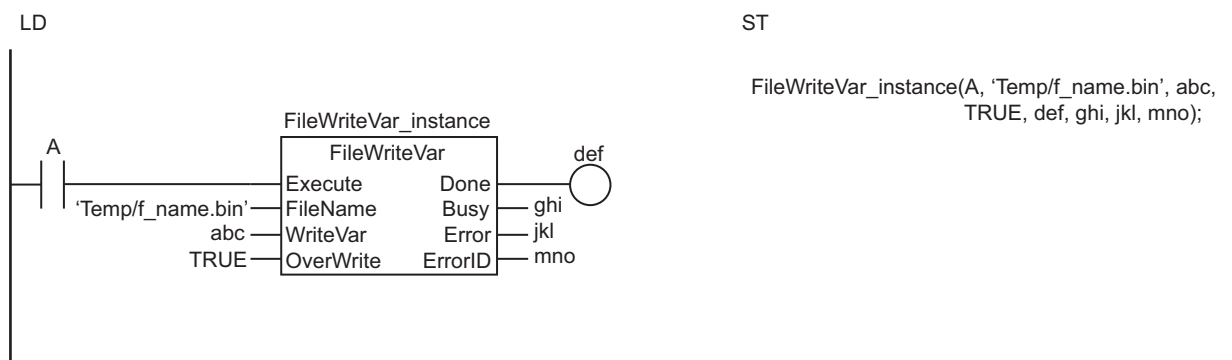
*FileName* includes the path. If a specified directory does not exist in the SD Memory Card, it is created. However, the directory is created only when only the lowest directory level of the specified path does not exist.

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 <i>OverWrite</i>   | Processing                                       |
|-----------------------------|--|
| TRUE (Enable overwrite.)    | The existing file is overwritten.                |
| FALSE (Prohibit overwrite.) | The file is not overwritten and an error occurs. |

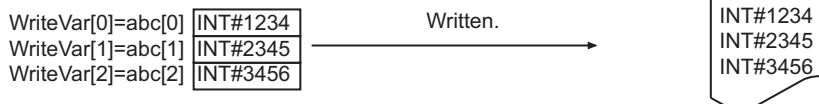
The following figure shows a programming example.

The contents of array variable *abc* is written to a file named 'Temp/f\_name.bin'. Variable *abc* is an INT array variable with three elements.



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.

File **FileName** = 'Temp/f\_name.bin'



## 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<br>TRUE: Can be used.<br>FALSE: Cannot be used.   |
| _Card1Protect | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected. |
| _Card1Err     | SD Memory Card Error Flag           | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.                    |

| Name            | Meaning                                | Data type | Description   |
|-----------------|--|-----------|---|
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) There is insufficient space available on the SD Memory Card.
  - d) The value of *FileName* is not a valid file name.
  - e) A file with the name *FileName* already exists, and the file is being accessed.
  - f) A file with the name *FileName* already exists, and the value of *OverWrite* is FALSE.
  - g) A file with the name *FileName* already exists, and the file is write protected.
  - h) The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - i) The maximum number of files or directories is exceeded.



- j) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename.
- k) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

### Sample Programming

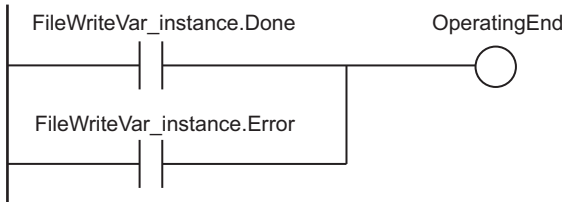
This sample writes all of array variable Var1[] to the file 'File1.dat.'

#### LD

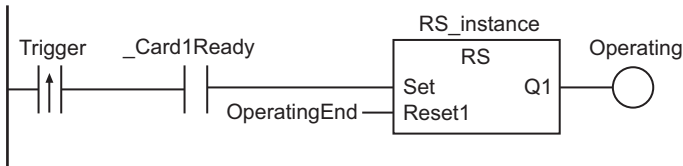
| Internal Variables | Variable              | Data type            | Initial value | Comment               |
|--------------------|-----------------------|----------------------|---------------|-----------------------|
|                    | OperatingEnd          | BOOL                 | FALSE         | 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 |

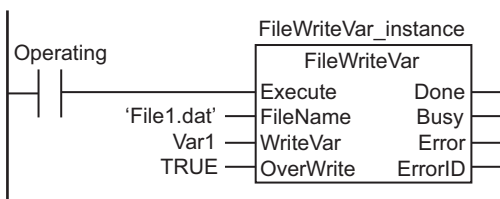
Determine if execution of the FileWriteVar instruction is completed.



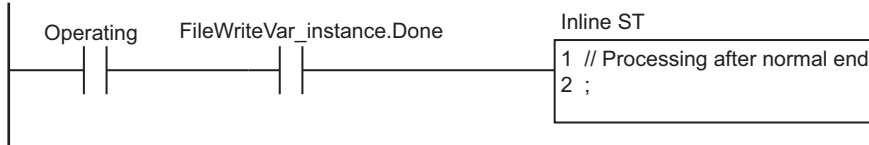
Accept trigger.



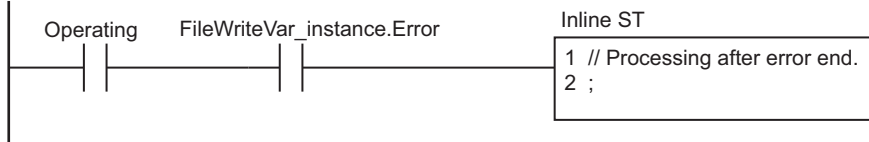
Execute FileWriteVar instruction.



Processing after normal end.



Processing after error end.



### 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/<br>FUN | Graphic expression | ST expression   |
|-------------|-------------------------|------------|--------------------|---|
| FileReadVar | Read Variable from File | FB         |                    | FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID); |

## Variables

|          | Meaning           | I/O    | Description  | Valid range   | Unit | Default |
|----------|-------------------|--------|--|---|------|---------|
| FileName | File name         | Input  | Name of file to read                               | 66 bytes max.<br>(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.   | ---  | ---     |

|          | Boo lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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

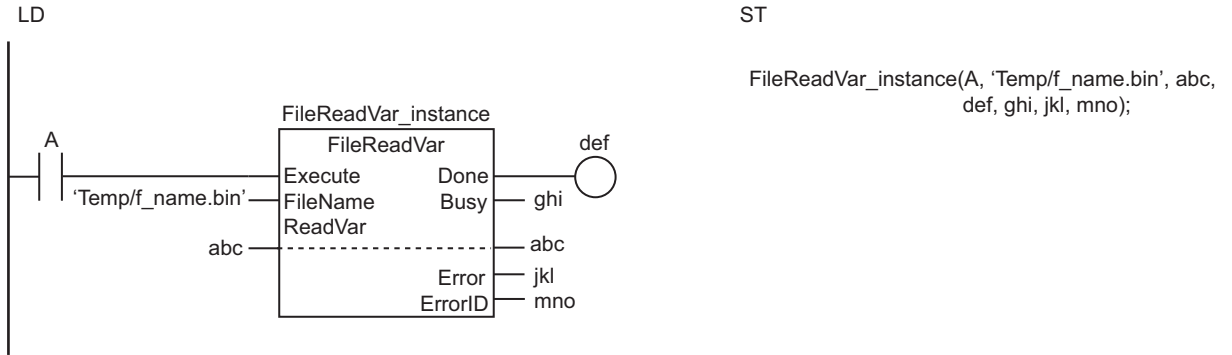
The FileReadVar instruction reads the contents of the file specified by *FileName* from the SD Memory Card as binary data. The contents that is read is assigned to variable to write *ReadVar*.

You can specify an enumeration, array, array element, structure, or structure member for *ReadVar*.

The following figure shows a programming example.

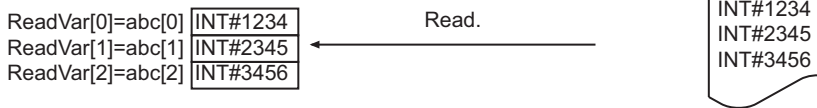
Here, the contents of the file called 'Temp/f\_name.bin' is read and written to the array variable abc[].

Variable *abc* is an INT array variable with three elements.



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**.

File **FileName** = 'Temp/f\_name.bin'



### 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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The file specified by *FileName* does not exist.
  - c) The file specified by *FileName* is being accessed.
  - d) The value of *FileName* is not a valid file name.
  - e) The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - f) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: *FileWriteVar*, *FileReadVar*, *FileCopy*, *DirCreate*, *FileRemove*, *DirRemove*, and *FileRename*.
  - g) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

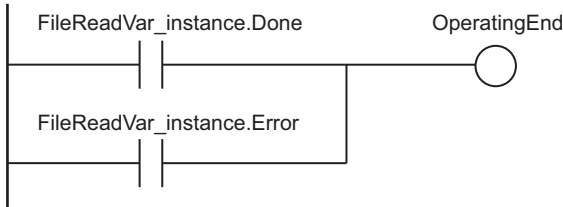
This sample reads the contents of the file 'File1.dat' and stores it in array variable *Var1*.

### LD

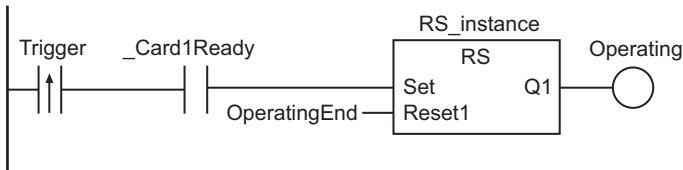
| Internal Variables | Variable             | Data type            | Initial value | Comment               |
|--------------------|----------------------|----------------------|---------------|-----------------------|
|                    | OperatingEnd         | BOOL                 | FALSE         | 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 |

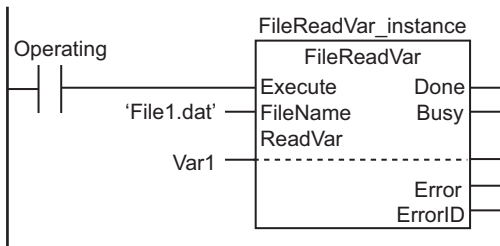
Determine if execution of the FileReadVar instruction is completed.



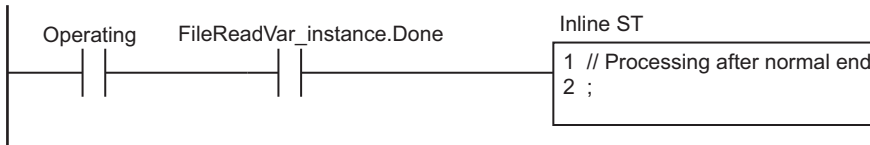
Accept trigger.



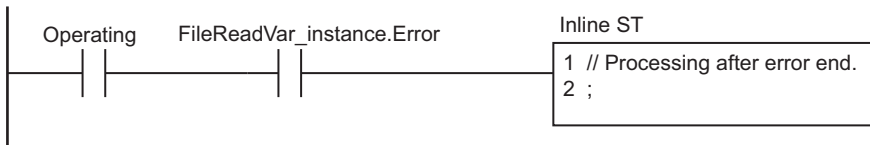
Execute FileReadVar instruction.



Processing after normal end.



Processing after error end.



**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.                        |

| Internal Variables | Variable             | Data type            | Initial value | Comment          |
|--------------------|----------------------|----------------------|---------------|------------------|
|                    | 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/<br>FUN | Graphic expression   | ST expression  |
|-------------|-----------|------------|--|--|
| FileOpen    | Open File | FB         | <pre> graph LR     subgraph FileOpen_instance         FileOpen     end     Execute --- FileOpen     FileName --- FileOpen     Mode --- FileOpen     FileOpen --- Done     FileOpen --- Busy     FileOpen --- Error     FileOpen --- ErrorID     FileOpen --- FileID </pre> | FileOpen_instance(Execute, File-Name, Mode, Done, Busy, Error, ErrorID, FileID); |

## Variables

|          | Meaning   | I/O    | Description                | Valid range   | Unit | Default     |
|----------|-----------|--------|----------------------------|---|------|-------------|
| FileName | File name | Input  | Name of file to open       | 66 bytes max.<br>(65 single-byte alphanumeric characters plus the final NULL character) | ---  | "           |
| Mode     | Open mode |        | Mode in which to open file | *1  | ---  | _READ_EXIST |
| FileID   | File ID   | Output | ID of file that was opened | Depends on data type.   | ---  | ---         |

\*1. \_READ\_EXIST, \_RDWR\_EXIST, \_WRITE\_CREATE, \_RDWR\_CREATE, \_WRITE\_APPEND and \_RDWR\_APPEND

|          | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real num-<br>bers |      | Times, durations,<br>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 <i>Function</i> on page 2-1441 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*.

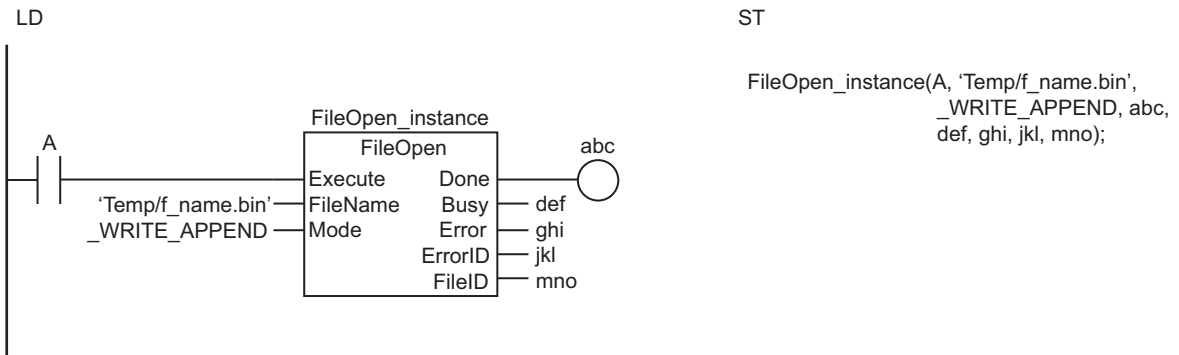
The result is output to file ID *FileID*. *FileID* is used to specify the file in other instructions, such as FileRead and FileWrite.

The data type of *Mode* is enumerated type \_eFOPEN\_MODE. The meanings of the enumerators are as follows:

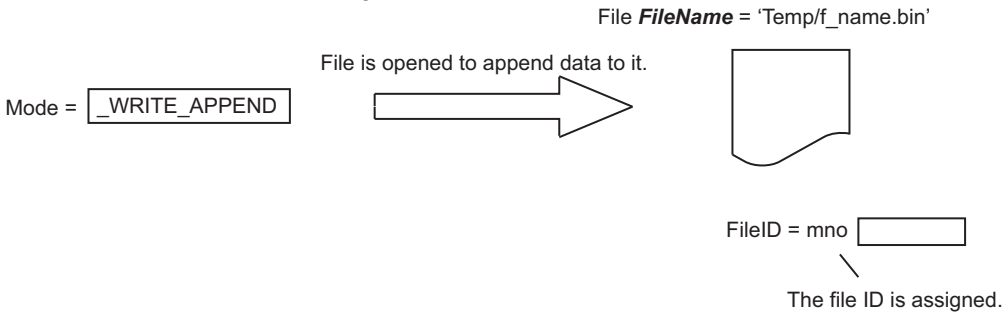
| Enumerator    | Meaning  |
|---------------|--|
| _READ_EXIST   | Use this value to open a text file to read it. The file is read from the beginning.  |
| _RDWR_EXIST   | Use this value to open a file to read and write it. The file is read and written from the beginning.   |
| _WRITE_CREATE | Use this 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.<br>However, if the file already exists and it is write-protected, an error occurs and the file is not opened. |
| _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.<br>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*.



The FileOpen instruction opens the file specified by **FileName** from the SD Memory Card to append data to it.  
The file ID is assigned to variable **FileID**.



## 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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.
- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted. The file, however, will remain open. Use the *FileClose* instruction to close the file.

- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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.
- 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 or 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) The value of *Mode* is `_READ_EXIST` or `_RDWR_EXIST`, and the file specified with *FileName* does not exist.
  - d) The value of *Mode* is outside the valid range.
  - e) The file specified by *FileName* is being accessed.
  - f) The value of *FileName* is not a valid file name.
  - g) The file specified by *FileName* is write protected.
  - h) The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - i) An attempt was made to open more than five files at the same time.
  - j) The maximum number of files or directories is exceeded.
  - k) 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.
  - l) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

### Sample Programming

Refer to *Sample Programming* on page 2-1454 for the FileRead instruction, *Sample Programming* on page 2-1461 for the FileWrite instruction, *Sample Programming* on page 2-1469 for the FileGets instruction, and *Sample Programming* on page 2-1477 for the FilePuts instruction.

# FileClose

The FileClose instruction closes the specified file in the SD Memory Card.

| Instruction | Name       | FB/<br>FUN | Graphic expression | ST expression  |
|-------------|------------|------------|--------------------|--|
| FileClose   | Close File | FB         |                    | FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID); |

## Variables

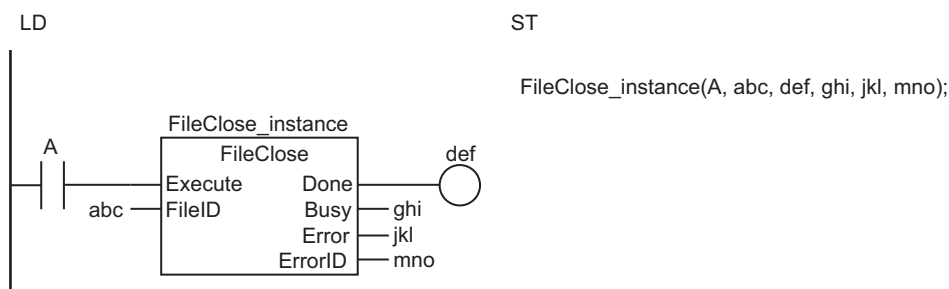
|        | Meaning | I/O   | Description         | Valid range           | Unit | Default |
|--------|---------|-------|---------------------|-----------------------|------|---------|
| FileID | File ID | Input | ID of file to close | Depends on data type. | ---  | 0       |

|        | 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    |       |             |      |       |       |      |     |      |          |      |       |      |      |              |    |   |  |  |  |  |  |

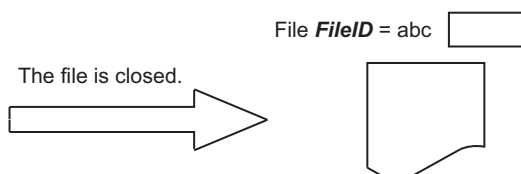
## Function

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.



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.
- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted. The file, however, will remain open. Use the FileClose instruction to close the file.

- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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.
- 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 or 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The file specified by *FileID* does not exist.
  - c) The file specified by *FileID* is already closed.
  - d) The file specified by *FileID* is being accessed.
  - e) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

Refer to *Sample Programming* on page 2-1454 for the FileRead instruction, *Sample Programming* on page 2-1461 for the FileWrite instruction, *Sample Programming* on page 2-1469 for the FileGets instruction, and *Sample Programming* on page 2-1477 for the FilePuts instruction.

# FileSeek

The FileSeek instruction sets a file position indicator in the specified file in the SD Memory Card.

| Instruction | Name      | FB/<br>FUN | Graphic expression   | ST expression   |
|-------------|-----------|------------|--|---|
| FileSeek    | Seek File | FB         | <pre> graph TD     subgraph FileSeek_instance         FileSeek     end     Execute --- FileSeek     FileID --- FileSeek     Offset --- FileSeek     Origin --- FileSeek     FileSeek --- Done     FileSeek --- Busy     FileSeek --- Error     FileSeek --- ErrorID </pre> | FileSeek_instance(Execute, FileID, Offset, Origin, Done, Busy, Error, ErrorID); |

## Variables

|        | 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 <i>Origin</i>                          |                                    | Bytes |           |
| Origin | Reference position |       | Reference position for file position indicator     | _SEEK_SET, _SEEK_CUR, or _SEEK_END | ---   | _SEEK_SET |

|        | Boo<br>lean   | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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 <i>Function</i> on page 2-1448 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*.

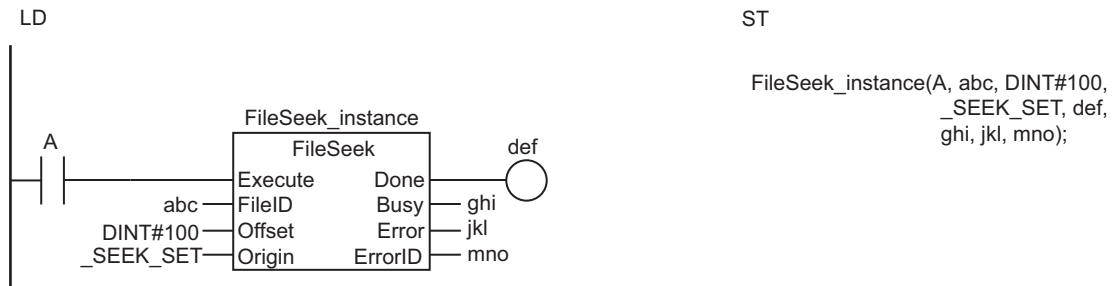
The data type of *Origin* is enumerated type \_eFSEEK\_ORIGIN. The meanings of the enumerators are as follows:

| Enumerator | Meaning                                     |
|------------|---|
| _SEEK_SET  | Beginning of file                           |
| _SEEK_CUR  | Location of current file position indicator |



| Enumerator | Meaning     |
|------------|-------------|
| _SEEK_END  | End of file |

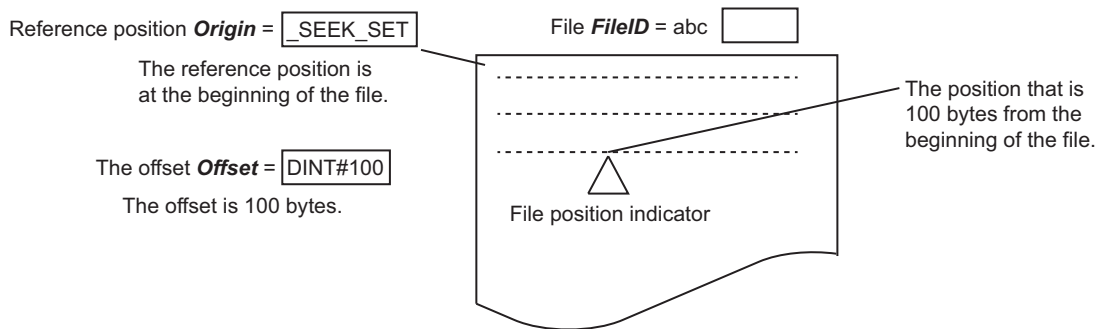
The following figure shows a programming example. A file position indicator is set at 100 bytes from the beginning of the file.



```

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<br>TRUE: Can be used.<br>FALSE: Cannot be used.   |
| _Card1Protect | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected. |
| _Card1Err     | SD Memory Card Error Flag           | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.                    |
| _Card1Access  | SD Memory Card Access Flag          | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>FALSE: Not being accessed.                               |

| Name            | Meaning                                | Data type | Description   |
|-----------------|--|-----------|---|
| _Card1PowerFail | SD Memory Card Power Interruption Flag | BOOL      | This flag indicates if an error occurred in completing processing when power was interrupted during access.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- You need to use the FileOpen instruction to obtain the value of *FileID* before you execute this instruction.
- If you specify **\_WRITE\_APPEND** or **\_RDWR\_APPEND** for *Mode* and execute the FileOpen instruction to append data to a file, the data is always appended to the end of the file.  
If you specify **\_RDWR\_APPEND** for *Mode* to execute the FileOpen instruction, the file position indicator set by the FileSeek instruction will be used only for reading data.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The value of *Origin* is outside the valid range.
  - c) The position specified by *Origin* and *Offset* exceeds the file size.
  - d) The file specified by *FileID* does not exist.
  - e) The file specified by *FileID* is being accessed.
  - f) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

Refer to *Sample Programming* on page 2-1454 for the FileRead instruction, and *Sample Programming* on page 2-1461 for the FileWrite instruction.

# FileRead

The FileRead instruction reads the data from the specified file in the SD Memory Card.

| Instruction | Name      | FB/<br>FUN | Graphic expression   | ST expression   |
|-------------|-----------|------------|--|---|
| FileRead    | Read File | FB         | <pre> FileRead_instance   FileRead   Execute --- Done   FileID --- Busy   ReadBuf ---   Size --- Error   ErrorID --- ReadSize   EOF ---           </pre> | FileRead_instance(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF); |

## Variables

|                      | 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[]<br>(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<br>TRUE: Reached.<br>FALSE: Not reached. |                       |      |         |

|                      | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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[]<br>(array) | OK          | OK          | OK   | OK    | OK    | OK    | OK       | OK    | OK    | OK   | OK  | OK   | OK   | OK           | OK    | OK  | OK   | OK  | OK | OK     |
| ReadSize             |             |             |      |       |       |       | OK       |       |       |      |     |      |      |              |       |   |      |     |    |        |
| EOF                  | OK          |             |      |       |       |       |          |       |       |      |     |      |      |              |       |   |      |     |    |        |

Arrays enumerations or structures can also be specified.

## 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.

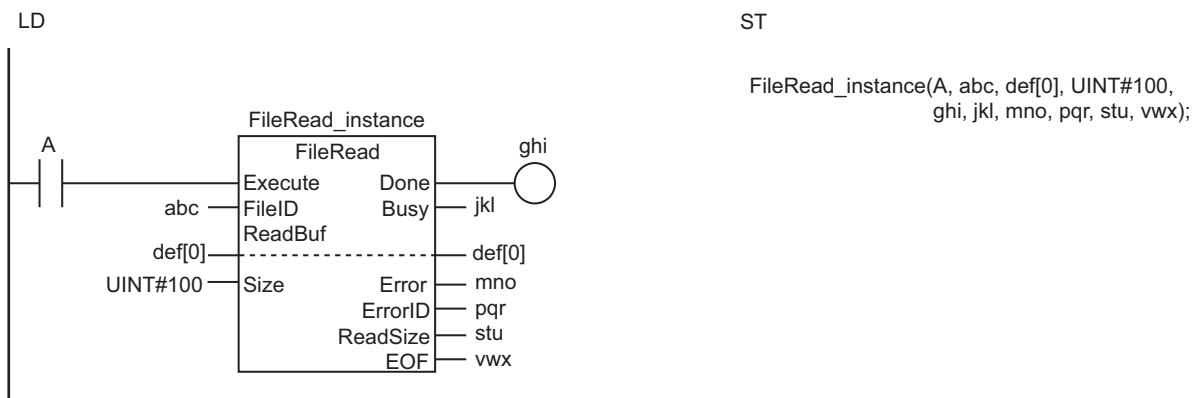
The amount of data to read is *Size* times the size of the `ReadBuf[]` data type. In other words, it is *Size* elements of `ReadBuf[]`.

You can specify an array of enumerations or structures for `ReadBuf[]`.

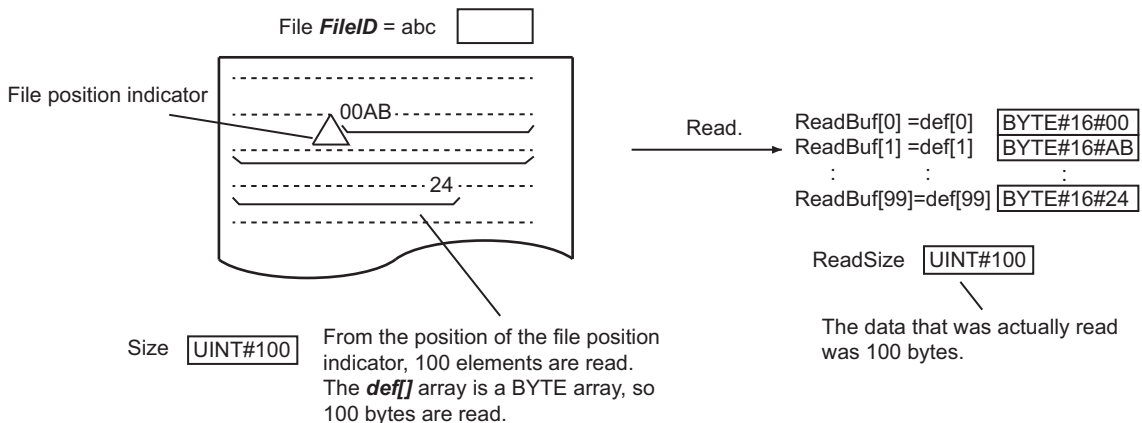
The actual number of elements that were read is stored in *ReadSize*. Normally, *Size* and *ReadSize* will have the same values. If the amount of data from the file position indicator to the end of the file is smaller than *Size*, an error will not occur, and the data to the end of the file is stored in `ReadBuf[]`. In this case, 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.



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   |
|--------------------------|---------------------------|-----------|---|
| <code>_Card1Ready</code> | SD Memory Card Ready Flag | BOOL      | This flag indicates if the SD Memory Card can be accessed by instructions and communications commands. *1<br>TRUE: Can be used.<br>FALSE: Cannot be used. |

| Name            | Meaning                                | Data type | Description   |
|-----------------|--|-----------|---|
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.   |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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 need to use the *FileOpen* instruction to obtain the value of *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 if a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read or write operations in progress are continued up 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.

- b) The number of array elements in ReadBuf[] is smaller than the value of *Size*.
- c) The file specified by *FileID* does not exist.
- d) The file specified by *FileID* is being accessed.
- e) The file specified by *FileID* was not opened in a reading mode.
- f) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## 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:

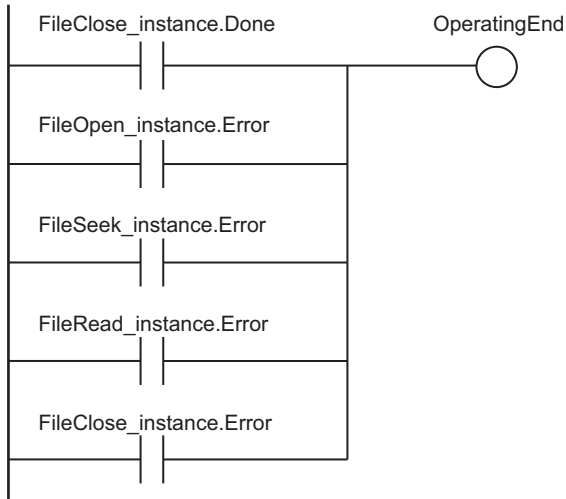
- 1** The FileOpen instruction is used to open the file 'ABC.bin.'
- 2** The FileSeek instruction is used to set a file position indicator at the second byte from the beginning of the file.
- 3** The FileRead instruction is used to read four bytes of data from the position of the file position indicator and store it in array variable InDat[].
- 4** The FileClose instruction is used to close the file 'ABC.bin.'

## LD

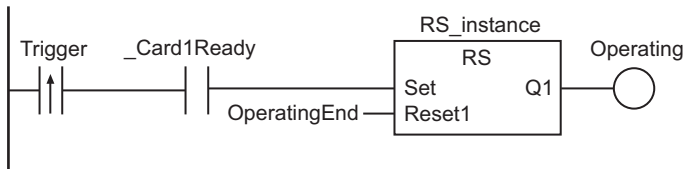
| Internal Variables | Variable           | Data type             | Initial value | Comment               |
|--------------------|--------------------|-----------------------|---------------|-----------------------|
|                    | OperatingEnd       | BOOL                  | FALSE         | Processing 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 |

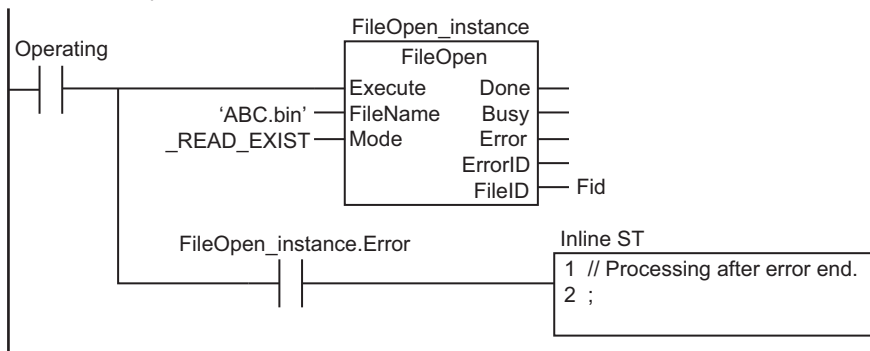
Determine if instruction execution is completed.



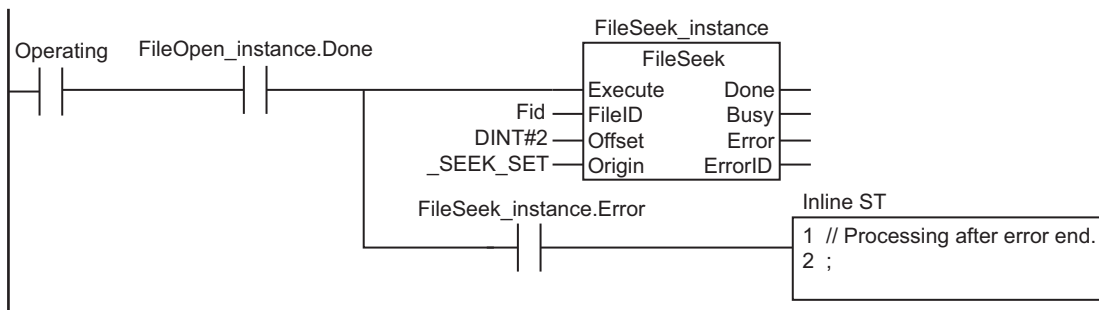
Accept trigger.



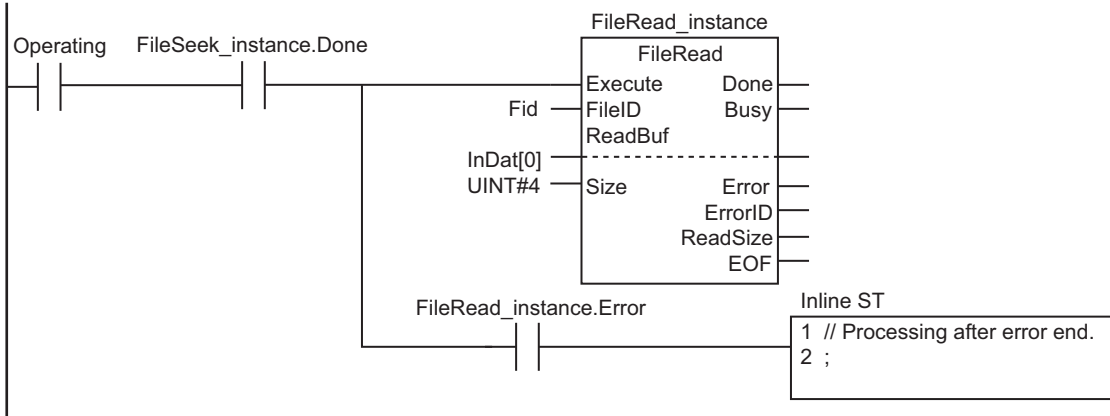
Execute FileOpen instruction.



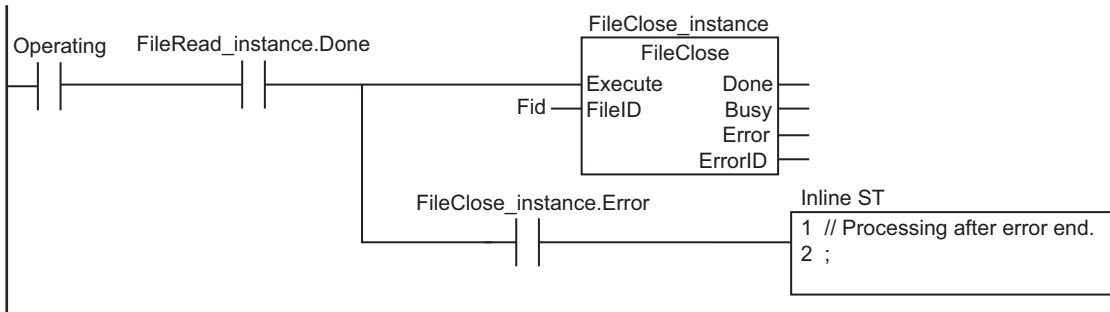
Execute FileSeek instruction.



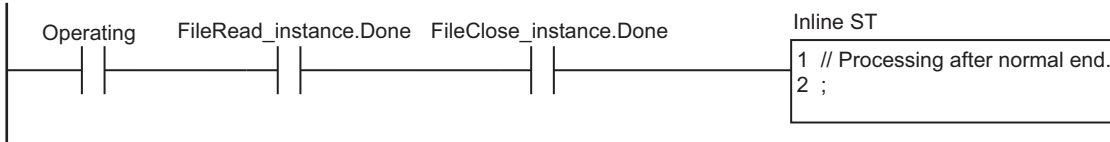
Execute FileRead instruction.



Execute FileClose instruction.



Processing after normal end.



## 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
                Origin :=_SEEK_SET);       //
            nd byte from the beginning.
    
```

```

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/<br>FUN | Graphic expression  | ST expression   |
|-------------|------------|------------|---|---|
| FileWrite   | Write File | FB         | <pre> FileWrite_instance ┌─── FileWrite ───┐ │ Execute  Done   │ │ FileID   Busy   │ │ WriteBuf Error   │ │ Size     ErrorID│ │           WriteSize │ └──────────────────┘ </pre> | FileWrite_instance(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize); |

## Variables

|                       | Meaning                     | I/O    | Description                                   | Valid range           | Unit | Default |
|-----------------------|-----------------------------|--------|---|-----------------------|------|---------|
| FileID                | File ID                     | Input  | ID of file to write                           | Depends on data type. | ---  | 0       |
| WriteBuf[]<br>(array) | Write buffer                |        | Write data                                    |                       |      | *1      |
| 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. | ---  | ---     |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                       | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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[]<br>(array) | OK          | OK          | OK   | OK    | OK    | OK       | OK   | OK    | OK    | OK   | OK  | OK   | OK                   | OK   | OK   | OK   | OK   | OK  | OK | OK     |
| 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.

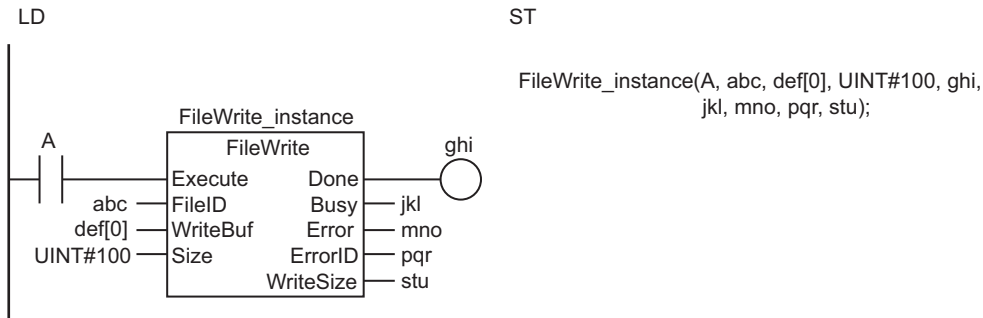
The contents of the write buffer *WriteBuf[]* is written to the file.

The amount of data to be written is the size of the data type of *WriteBuf[]* times *Size*. In other words, it is *Size* elements of *WriteBuf[]*.

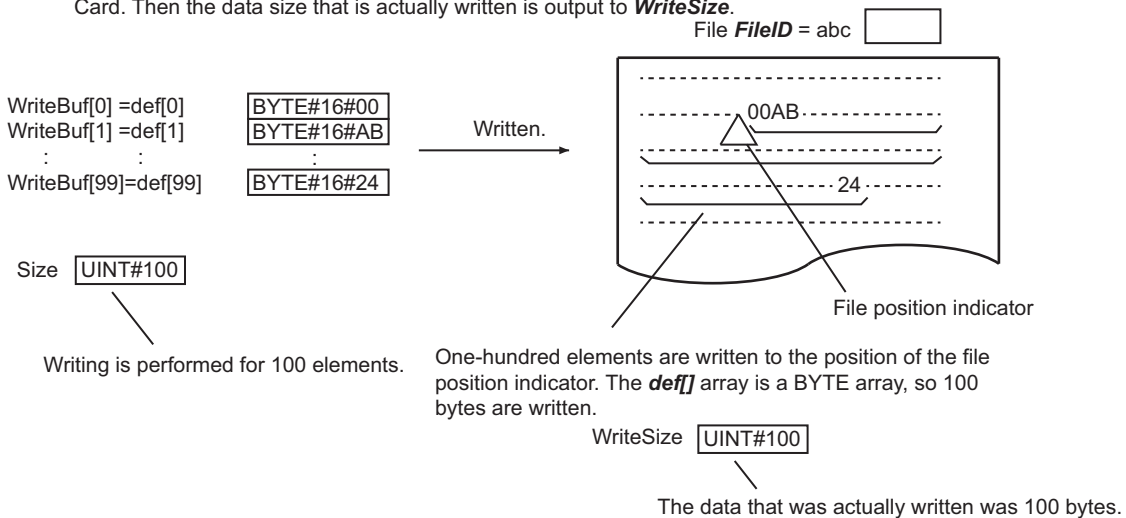
You can specify an array of enumerations or structures for *WriteBuf[]*.

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.



The FileWrite instruction writes the contents of the write buffer **WriteBuf[]** to the position of the file position indicator in the file specified by **FileID** in the SD Memory Card. Then the data size that is actually written is output to **WriteSize**.



### Related System-defined Variables

| Name                       | Meaning                             | Data type | Description   |
|----------------------------|-------------------------------------|-----------|---|
| <code>_Card1Ready</code>   | SD Memory Card Ready Flag           | BOOL      | This flag indicates if the SD Memory Card can be accessed by instructions and communications commands. *1<br>TRUE: Can be used.<br>FALSE: Cannot be used.   |
| <code>_Card1Protect</code> | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected. |
| <code>_Card1Err</code>     | SD Memory Card Error Flag           | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.                    |
| <code>_Card1Access</code>  | SD Memory Card Access Flag          | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>FALSE: Not being accessed.                               |

| Name            | Meaning                                | Data type | Description   |
|-----------------|--|-----------|---|
| _Card1PowerFail | SD Memory Card Power Interruption Flag | BOOL      | This flag indicates if an error occurred in completing processing when power was interrupted during access.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- You need to use the FileOpen instruction to obtain the value of *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 if a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read or write operations in progress are continued up to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) There is insufficient space available on the SD Memory Card.
  - d) The number of array elements in WriteBuf[] is smaller than the value of *Size*.
  - e) The file specified by *FileID* does not exist.
  - f) The file specified by *FileID* is being accessed.
  - g) The file specified by *FileID* was not opened in a writing mode.
  - h) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## 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.

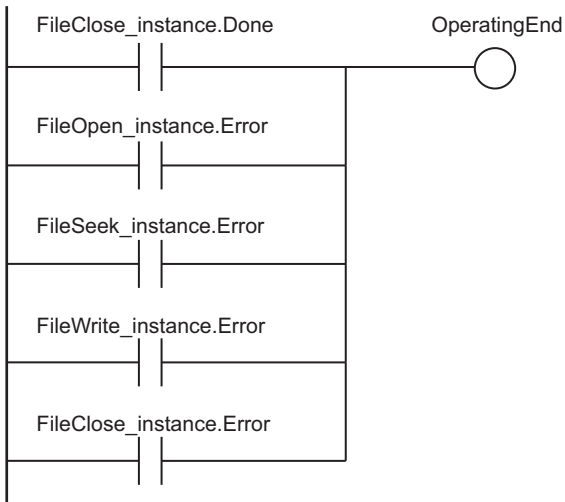
- 3 The FileWrite instruction is used to write four bytes from array variable OutDat[] to the position of the file position indicator.
- 4 The FileClose instruction is used to close the file 'ABC.bin.'

**LD**

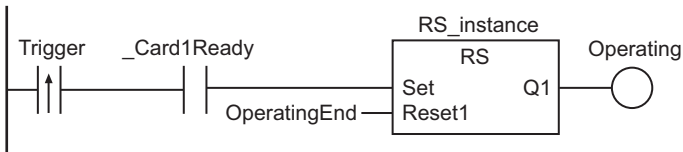
| Internal Variables | Variable           | Data type             | Initial value | Comment               |
|--------------------|--------------------|-----------------------|---------------|-----------------------|
|                    | OperatingEnd       | BOOL                  | FALSE         | Processing 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 |

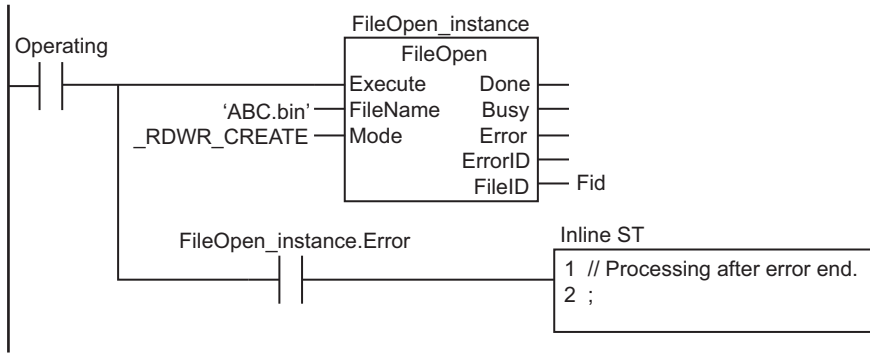
Determine if instruction execution is completed.



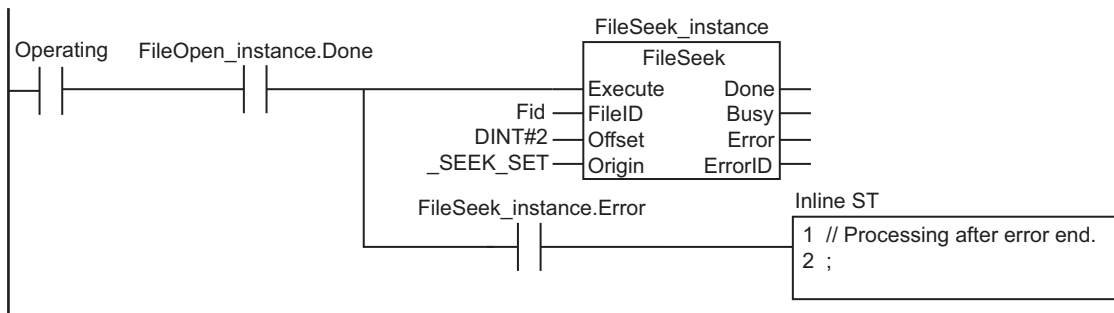
Accept trigger.



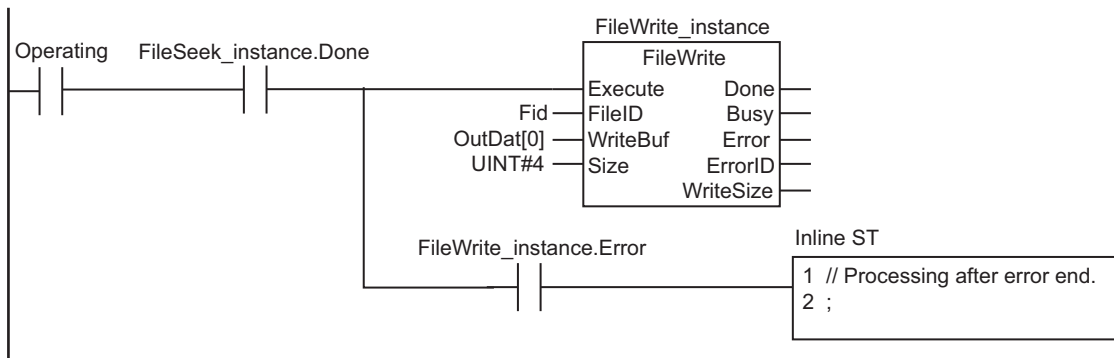
Execute FileOpen instruction.



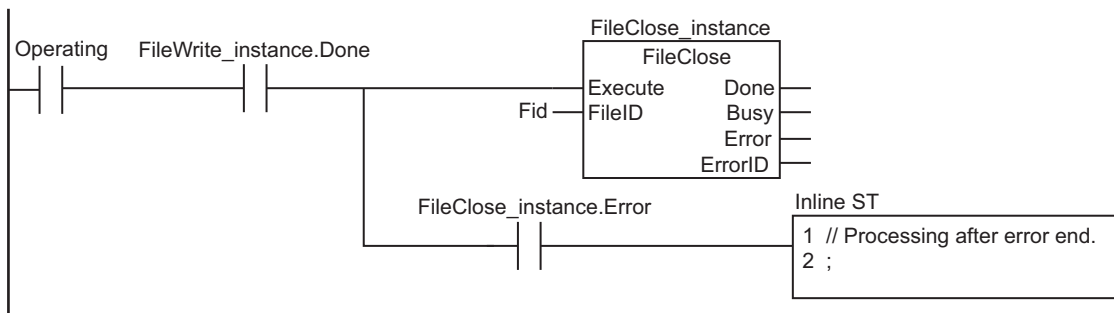
Execute FileSeek instruction.



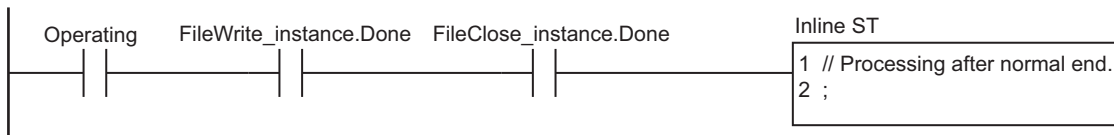
Execute FileWrite instruction.



Execute FileClose instruction.



Processing after normal end.



## 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 by
te 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|-----------------|------------|--|---|
| FileGets    | Get Text String | FB         | <pre> graph LR     subgraph FileGets_instance         FileGets     end     Execute --- FileGets     FileID --- FileGets     TrimLF --- FileGets     Done --- FileGets     Busy --- FileGets     Error --- FileGets     ErrorID --- FileGets     Out --- FileGets     EOF --- FileGets </pre> | FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF); |

## Variables

|        | 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<br>TRUE: Delete.<br>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<br>TRUE: Reached.<br>FALSE: Not reached.                              |                       |      |         |

|        | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real num<br>bers |       | Times, durations,<br>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.

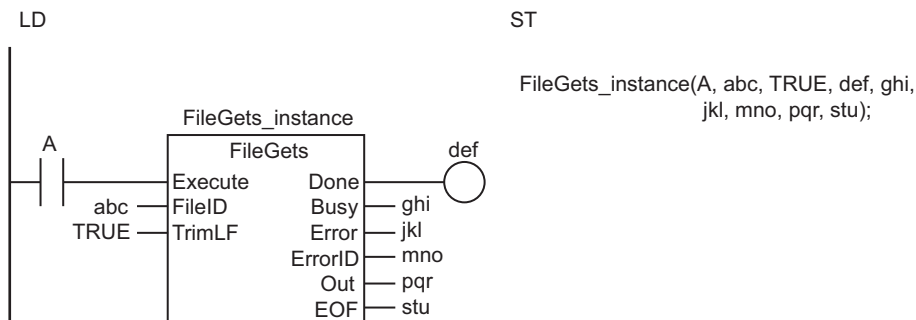
The text string that is read is written to read text string *Out*.

The following three line feeds are automatically detected: CR, LF, and CR+LF.

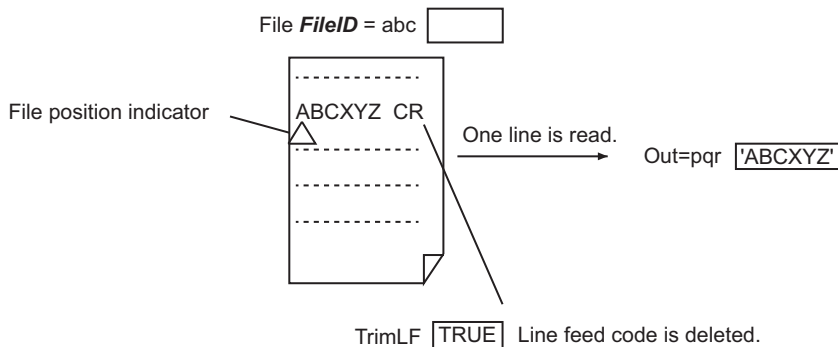
If line feed designation *TrimLF* is TRUE, the line feed code is deleted from the text string before it is written to *Out*.

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*.



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.   |
| _Card1Protect | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected. |
| _Card1Err     | SD Memory Card Error Flag           | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.                    |
| _Card1Access  | SD Memory Card Access Flag          | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>FALSE: Not being accessed.                               |

| Name            | Meaning                                | Data type | Description   |
|-----------------|--|-----------|---|
| _Card1PowerFail | SD Memory Card Power Interruption Flag | BOOL      | This flag indicates if an error occurred in completing processing when power was interrupted during access.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## Precautions for Correct Use

- Execution of this instruction is continued until completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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 need to use the *FileOpen* instruction to obtain the value of *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or if a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read or write operations in progress are continued up to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The file specified by *FileID* does not exist.
  - c) The file specified by *FileID* is being accessed.
  - d) The file specified by *FileID* was not opened in a reading mode.
  - e) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

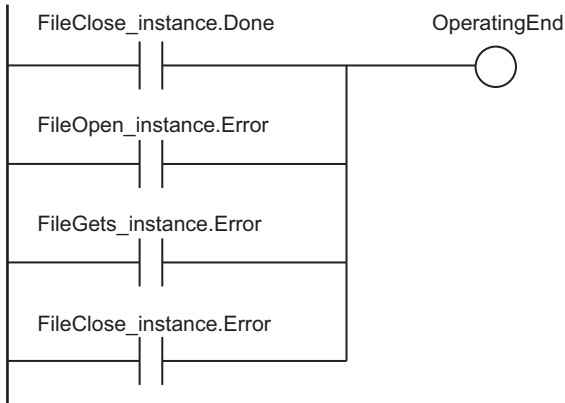
Here, multiple text strings that are separated by CR codes are stored in a file named 'ABC.csv.' All of them are text strings of numbers.

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*[].

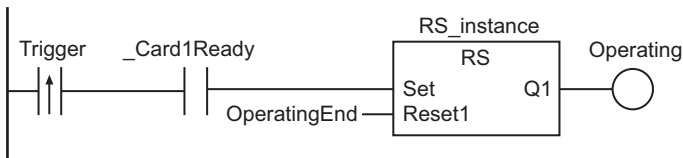
Processing is ended when all of the data to the end of the file is read. It is assumed that this sample programming is in a periodic task.



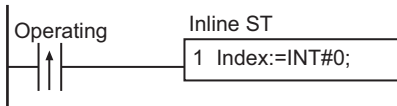
Determine if instruction execution is completed.



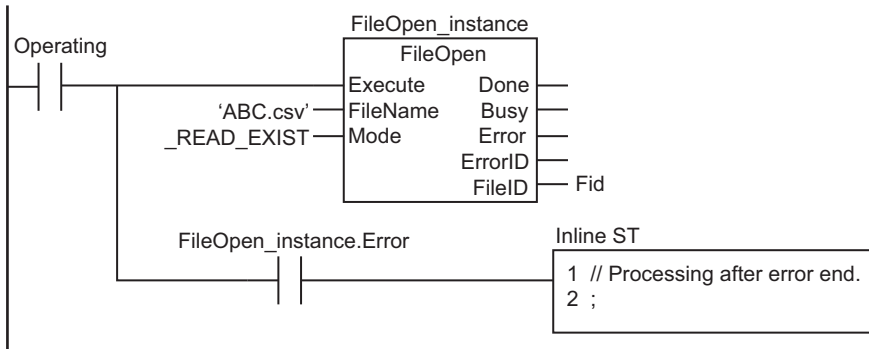
Accept trigger.



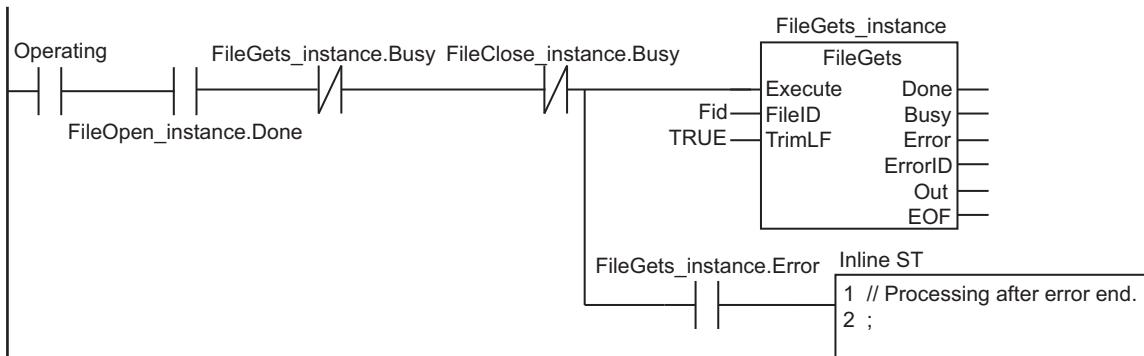
Initialize *InDat[]* element index.



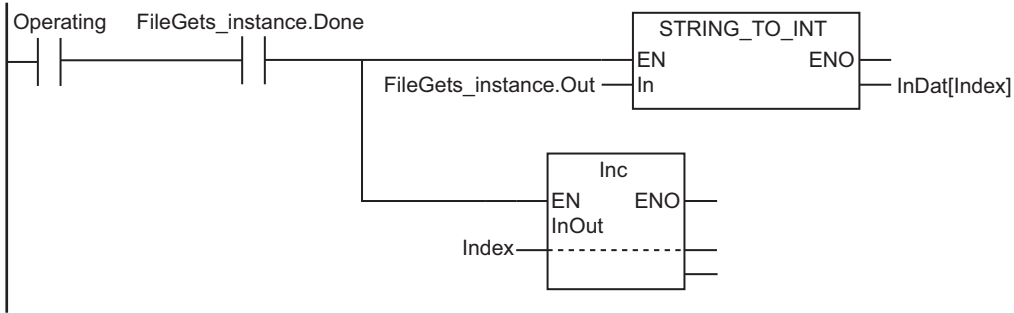
Execute FileOpen instruction.



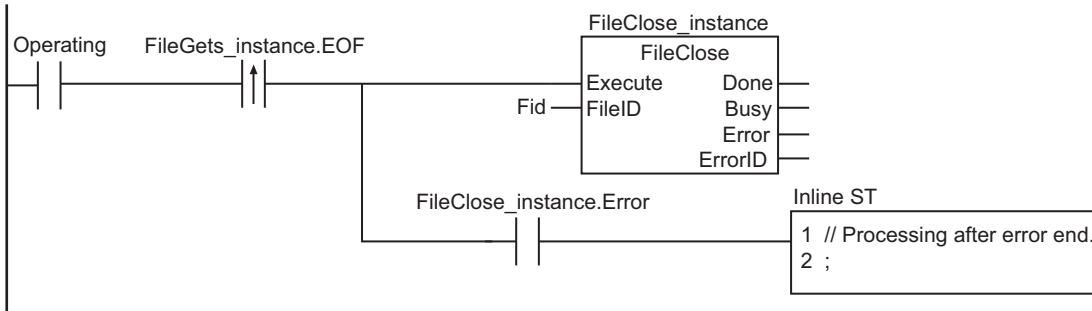
Execute FileGets instruction.



Execute STRING\_TO\_INT instruction.



Execute FileClose when EOF is detected.



Processing after normal end.



### 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

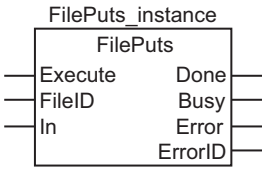
IF (FileClose_instance.Done=TRUE) THEN
    Operating:=FALSE;    // Normal end
END_IF;

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/<br>FUN | Graphic expression  | ST expression   |
|-------------|-----------------|------------|---|---|
| FilePuts    | Put Text String | FB         |  <pre> graph LR     subgraph FilePuts_instance [FilePuts_instance]         subgraph FilePuts [FilePuts]             Execute[Execute]             FileID[FileID]             In[In]             Done[Done]             Busy[Busy]             Error[Error]             ErrorID[ErrorID]         end     end     Execute --- Done     FileID --- Busy     In --- Error     In --- ErrorID           </pre> | FilePuts_instance(Execute, FileID, In, Done, Busy, Error, ErrorID); |

## Variables

|        | 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 |                       |      | "       |

|        | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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     |

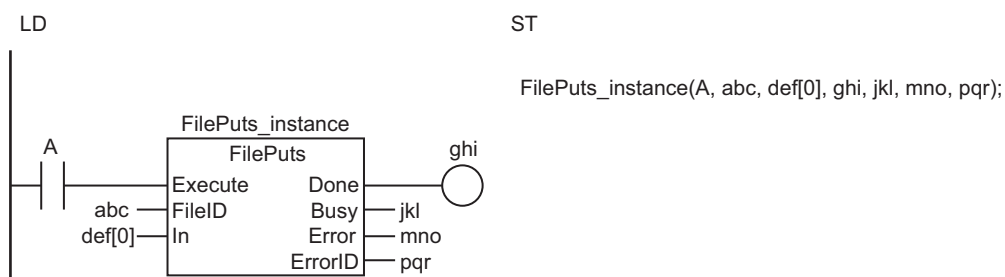
## Function

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.

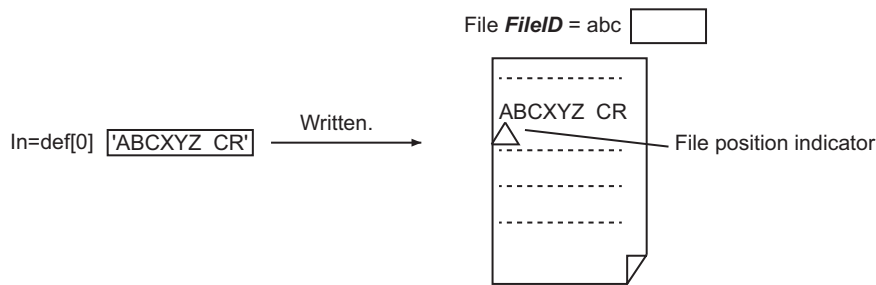
The file position indicator is set at the desired location in advance with the FileSeek instruction.

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.



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   |
|------------------------------|--|-----------|---|
| <code>_Card1Ready</code>     | SD Memory Card Ready Flag              | BOOL      | This flag indicates if the SD Memory Card can be accessed by instructions and communications commands. *1<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| <code>_Card1Protect</code>   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| <code>_Card1Err</code>       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| <code>_Card1Access</code>    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>FALSE: Not being accessed.   |
| <code>_Card1PowerFail</code> | SD Memory Card Power Interruption Flag | BOOL      | This flag indicates if an error occurred in completing processing when power was interrupted during access.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

### 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.

- You need to use the FileOpen instruction to obtain the value of *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or if a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read or write operations in progress are continued up to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) There is insufficient space available on the SD Memory Card.
  - d) The file specified by *FileID* does not exist.
  - e) The file specified by *FileID* is being accessed.
  - f) The file specified by *FileID* was not opened in a writing mode.
  - g) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

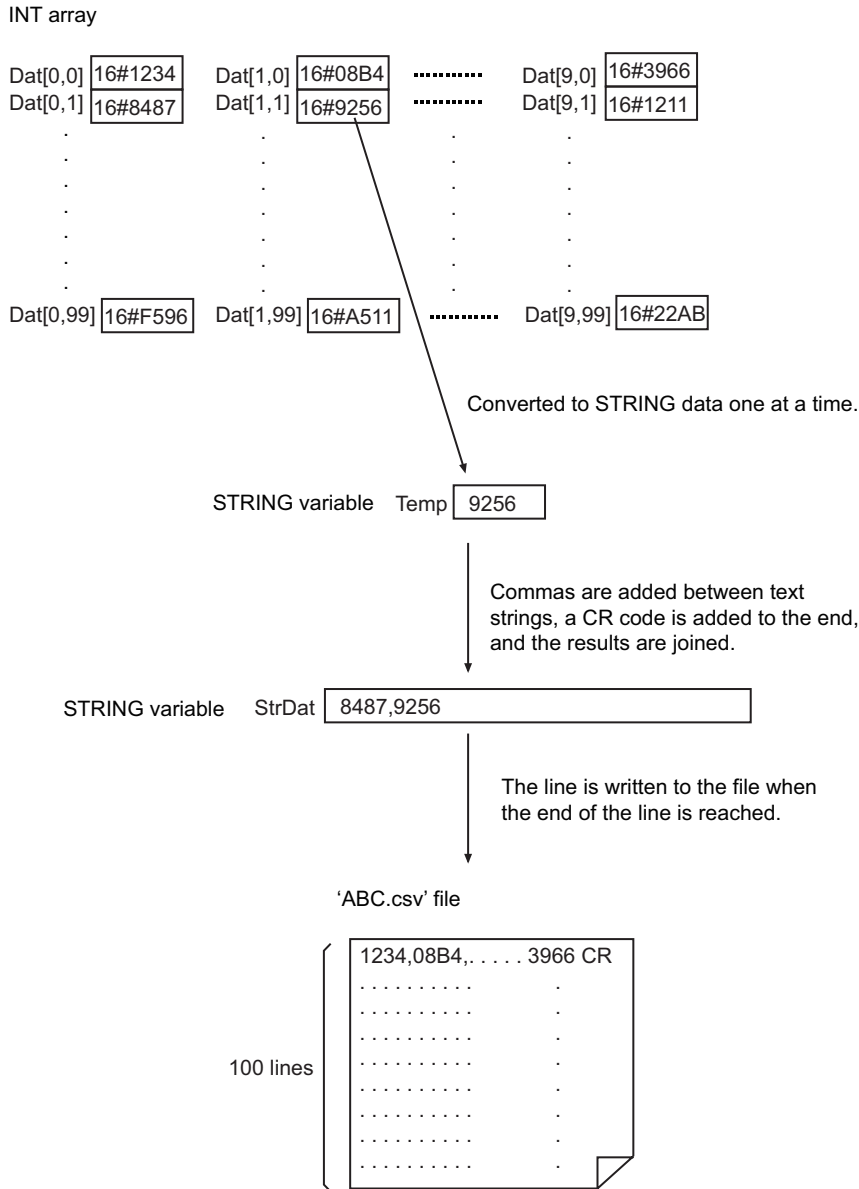
## Sample Programming

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** An element of `Dat[]` is converted into a text string and stored as a STRING variable, *Temp*.
- 2** If the *Temp* is not at the end of the row, insert a comma to join to the STRING variable *StrDat*. If the *Temp* is the last variable at the end of the row, add a CR+LF code to complete the STRING variable *StrDat*.
- 3** When the row is complete, *StrDat* is written into the file.
- 4** Steps 1 to 3 are repeated for 100 lines.



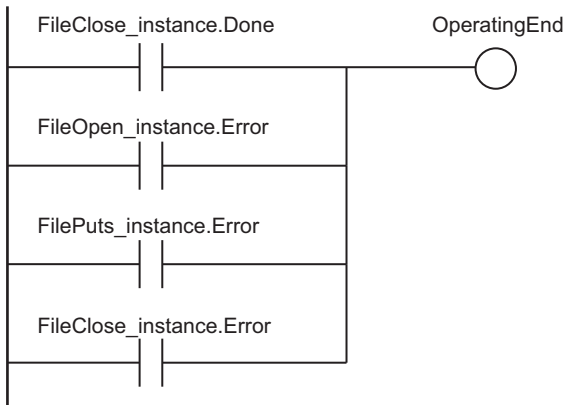
**LD**

| 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                       |               |                       |

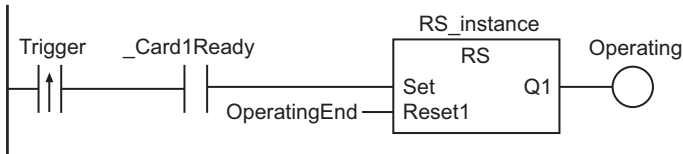
| Internal Variables | Variable           | Data type | Initial value | Comment |
|--------------------|--------------------|-----------|---------------|---------|
|                    | FileOpen_instance  | FileOpen  |               |         |
|                    | FilePuts_instance  | FilePuts  |               |         |
|                    | FileClose_instance | FileClose |               |         |

| External Variables | Variable    | Data type | Comment                   |
|--------------------|-------------|-----------|---------------------------|
|                    | _Card1Ready | BOOL      | SD Memory Card Ready Flag |

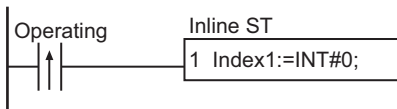
Determine if instruction execution is completed.



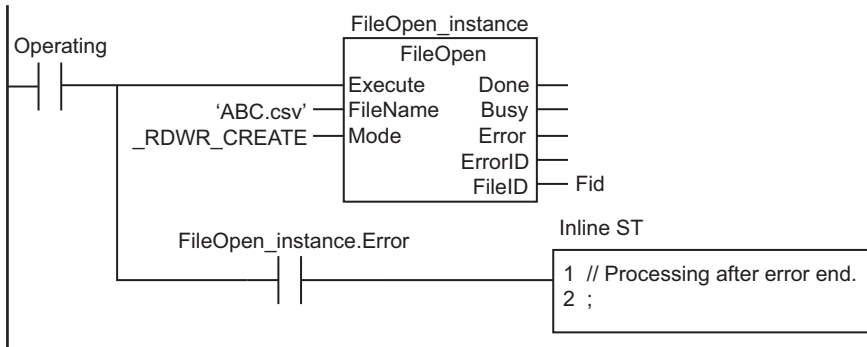
Accept trigger.



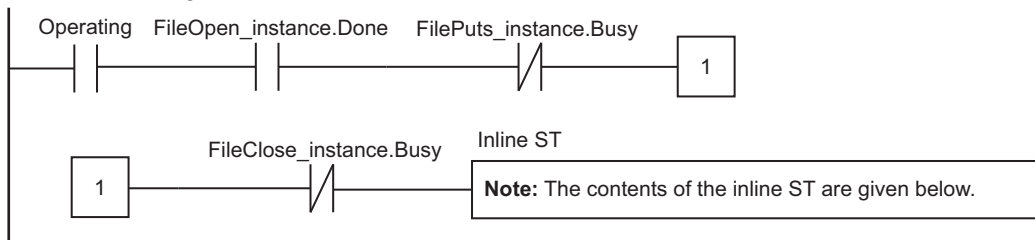
Initialize row index.



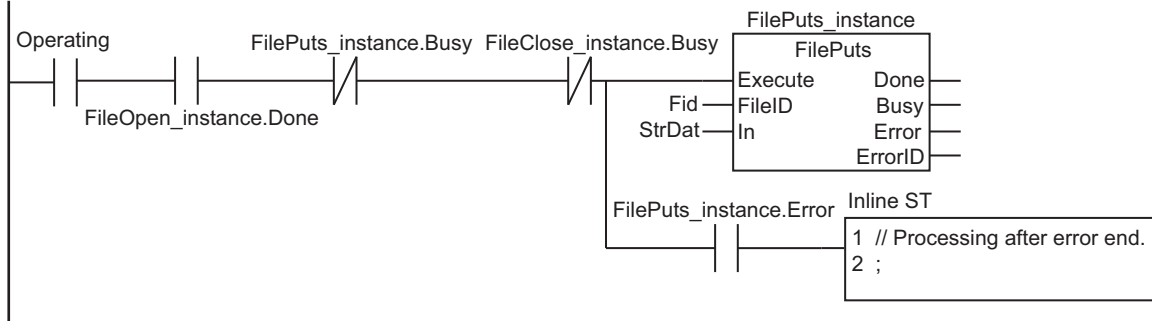
Execute FileOpen instruction.



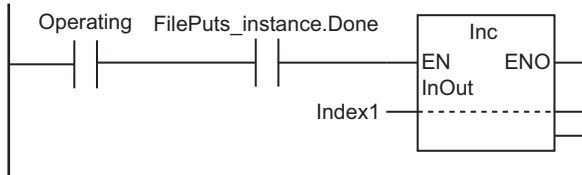
Create a text string for one line.



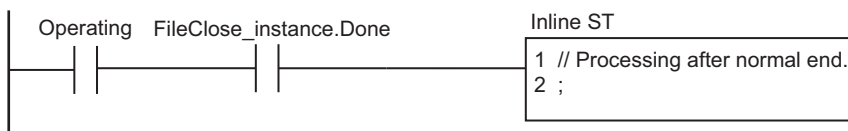
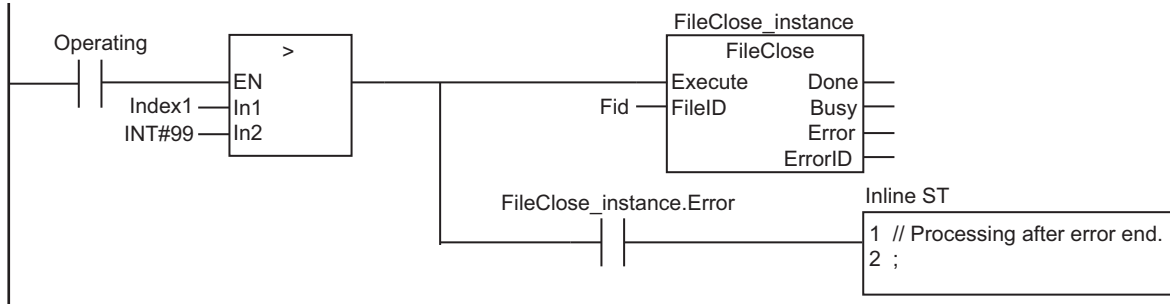
Write a text string for one line to the file.



Increment the line index.



Execute the FileClose instruction after 100 lines are written.



### ● 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');
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$1');
        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/<br>FUN | Graphic expression  | ST expression  |
|-------------|-----------|------------|---|--|
| FileCopy    | Copy File | FB         | <pre> FileCopy_instance FileCopy - Execute      Done - SrcFileName  Busy - DstFileName  Error - OverWrite    ErrorID           </pre> | FileCopy_instance(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID); |

## Variables

|              | Meaning          | I/O   | Description   | Valid range   | Unit | Default |
|--------------|------------------|-------|---|---|------|---------|
| SrcFile-Name | Source file      | Input | Name of file to copy                                  | 66 bytes max.<br>(65 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |
| DstFile-Name | Destination file |       | Name of destination file                              |   |      |         |
| OverWrite    | Overwrite enable |       | TRUE: Enable overwrite.<br>FALSE: Prohibit overwrite. | Depends on data type.   |      | FALSE   |

|              | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

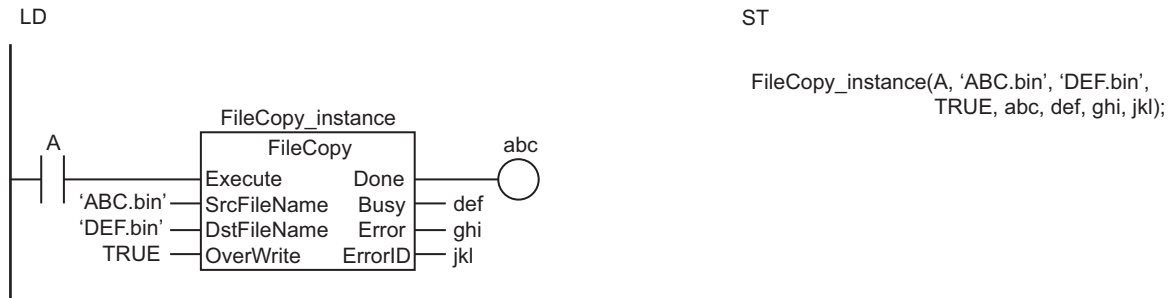
## Function

The FileCopy instruction copies the file specified by source file *SrcFileName* to designation file *DstFileName* in the SD Memory Card.

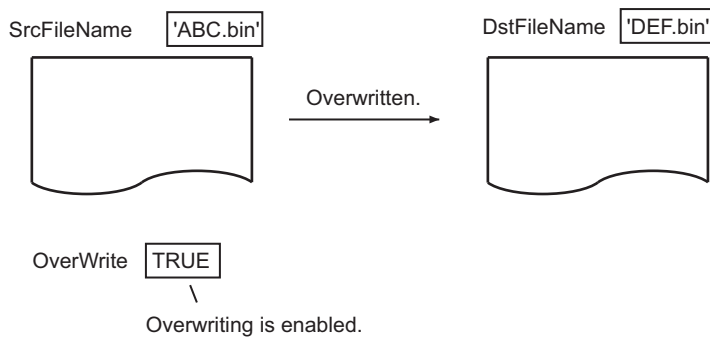
If a file with the name *DstFileName* already exists in the SD Memory Card, the following processing is performed depending on the value of *OverWrite* (overwrite enable).

| Value of <i>OverWrite</i>   | Description                                      |
|-----------------------------|--|
| 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.'



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                  |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                                |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.   |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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. *2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.
- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted.
- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) There is insufficient space available on the SD Memory Card.
  - d) The file specified by *SrcFileName* does not exist.
  - e) The value of *SrcFileName* is not a valid file name.
  - f) The file specified by *SrcFileName* or *DstFileName* is already being accessed.
  - g) The value of *DstFileName* is not a valid file name.
  - h) A file with the name *DstFileName* already exists, and the value of *OverWrite* is FALSE.
  - i) A file with the name *DstFileName* already exists, and the file is write protected.
  - j) The value of *DstFileName* exceeds the maximum number of bytes allowed in a file name.
  - k) The maximum number of files or directories is exceeded.
  - l) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: *FileWriteVar*, *FileReadVar*, *FileCopy*, *DirCreate*, *FileRemove*, *DirRemove*, and *FileRename*.
  - m) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

### Sample Programming

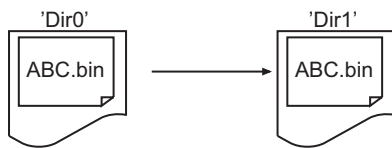
The following procedure is used to move a file.

- 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).

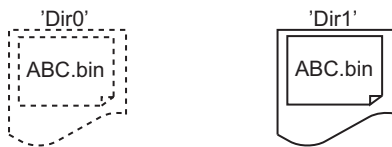
1. Create directory.



2. Copy file.



3. Delete directory.



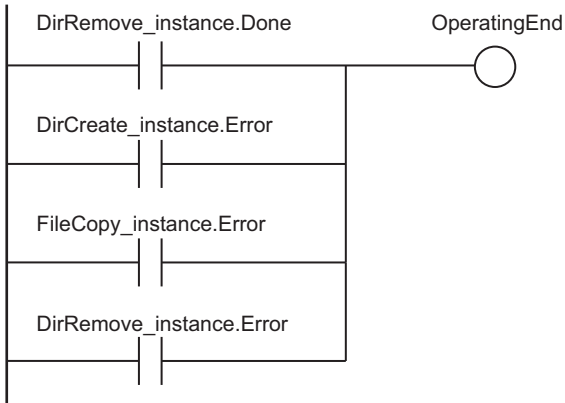
## LD

| 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 |               |                       |

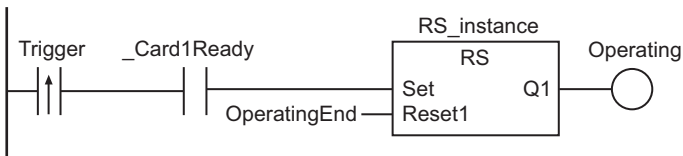
  

| External Variables | Variable    | Data type | Comment                   |
|--------------------|-------------|-----------|---------------------------|
|                    | _Card1Ready | BOOL      | SD Memory Card Ready Flag |

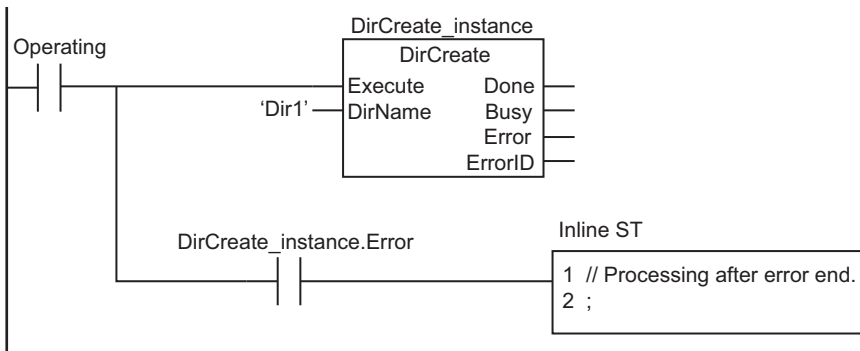
Determine if instruction execution is completed.



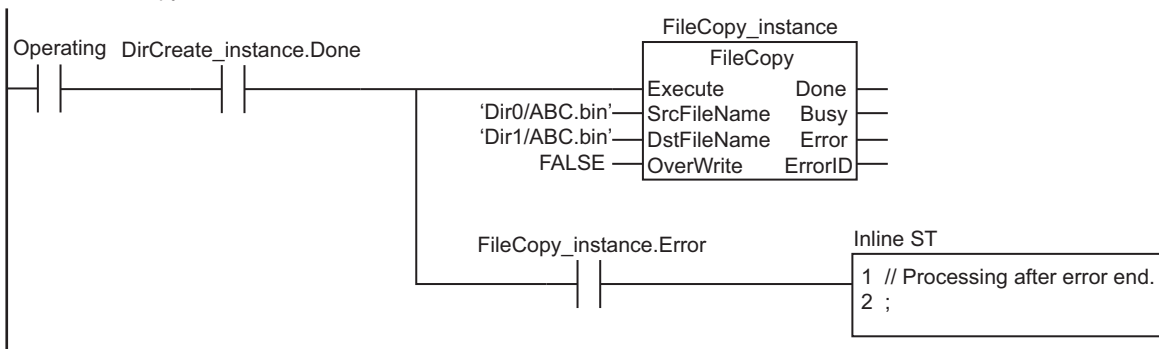
Accept trigger.



Execute DirCreate instruction.

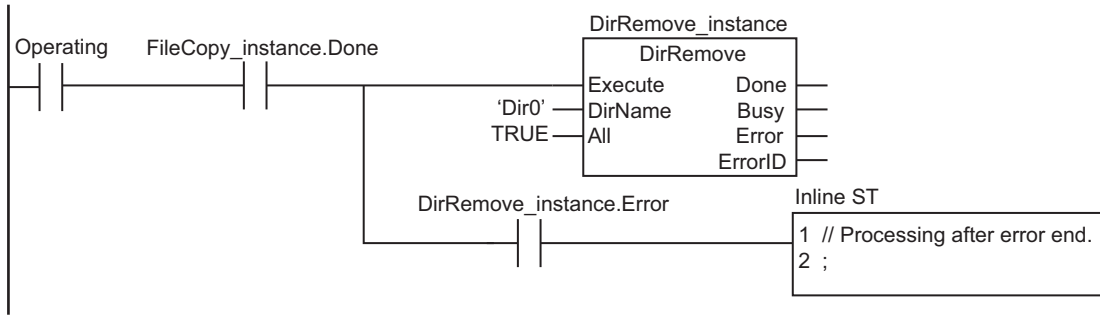


Execute FileCopy instruction.

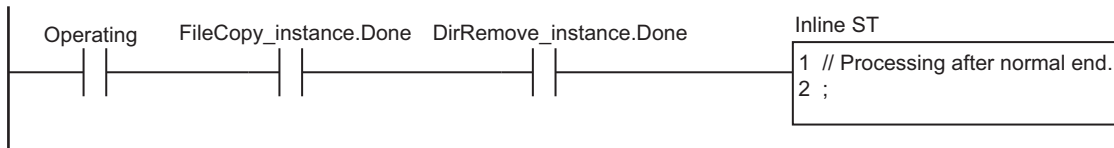




Execute DirRemove instruction.



Processing after normal end.



**ST**

| Internal Variables | Variable           | Data type | Initial value | Comment   |
|--------------------|--------------------|-----------|---------------|---|
|                    | Trigger            | BOOL      | FALSE         | Execution condition                               |
|                    | LastTrigger        | BOOL      | FALSE         | Value of <i>Trigger</i> 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;

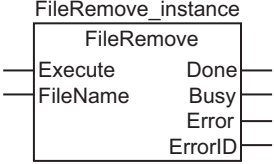
        IF (DirRemove_instance.Error=TRUE) THEN
            Stage:=INT#99;             // Error end
        END_IF;
    END_CASE;
END_IF;

```

```
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/<br>FUN | Graphic expression  | ST expression   |
|-------------|-------------|------------|---|---|
| FileRemove  | Delete File | FB         |  | FileRemove_instance(Execute, FileName, Done, Busy, Error, ErrorID); |

## Variables

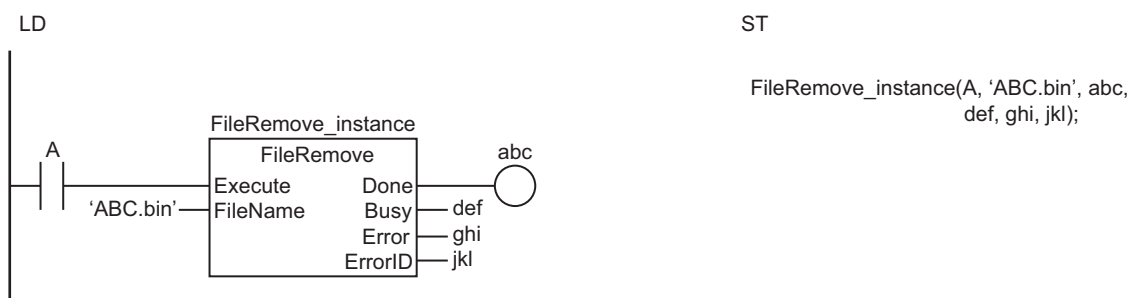
|          | Meaning   | I/O   | Description            | Valid range  | Unit | Default |
|----------|-----------|-------|------------------------|--|------|---------|
| FileName | File name | Input | Name of file to delete | 66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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     |

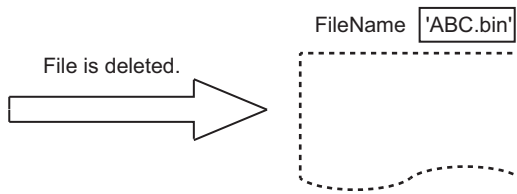
## Function

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.



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.

- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted.
- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) The file specified by *FileName* does not exist.
  - d) The file specified by *FileName* is being accessed.
  - e) A file with the name *FileName* already exists, and the file is write protected.
  - f) The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - g) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename.
  - h) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

### Sample Programming

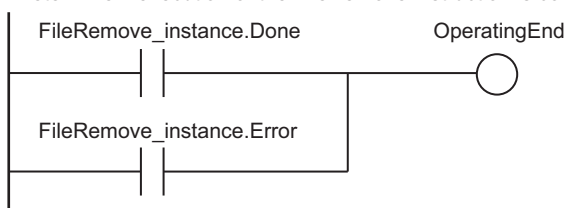
In this sample, the file named 'ABC.bin' is deleted from the SD Memory Card.

#### LD

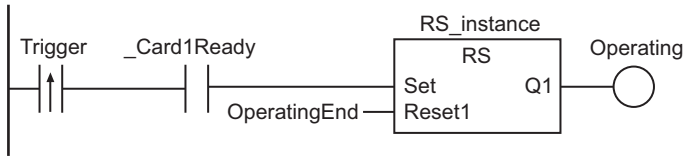
| Internal Variables | Variable            | Data type  | Initial value | Comment               |
|--------------------|---------------------|------------|---------------|-----------------------|
|                    | OperatingEnd        | BOOL       | FALSE         | Processing completed. |
|                    | 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 |

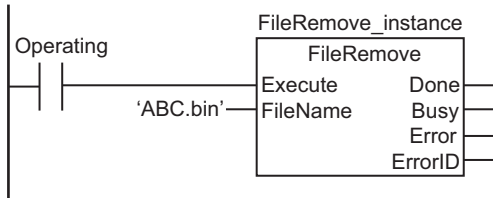
Determine if execution of the FileRemove instruction is completed.



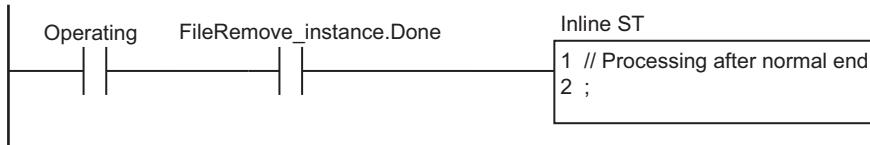
Accept trigger.



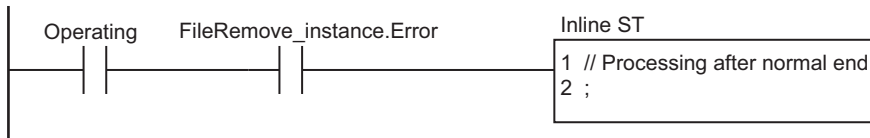
Execute FileRemove instruction.



Processing after normal end.



Processing after error end.



**ST**

| Internal Variables | Variable            | Data type  | Initial value | Comment   |
|--------------------|---------------------|------------|---------------|---|
|                    | Trigger             | BOOL       | FALSE         | Execution condition                               |
|                    | LastTrigger         | BOOL       | FALSE         | Value of <i>Trigger</i> 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|------------------|------------|--|---|
| FileRename  | Change File Name | FB         | <pre> FileRename_instance ┌───────────┐ │ FileRename │ ├───────────┤ │ Execute    │ Done      │ ├───────────┤ │ FileName   │ Busy     │ ├───────────┤ │ NewName    │ Error    │ ├───────────┤ │ OverWrite  │ ErrorID  │ └───────────┘ </pre> | FileRename_instance(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID); |

## Variables

|           | Meaning            | I/O   | Description   | Valid range   | Unit | Default |
|-----------|--------------------|-------|---|---|------|---------|
| FileName  | Original file name | Input | Original file name                                    | 66 bytes max.<br>(65 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |
| NewName   | New file name      |       | New file name   |   |      |         |
| OverWrite | Overwrite enable   |       | TRUE: Enable overwrite.<br>FALSE: Prohibit overwrite. | Depends on data type.   |      | FALSE   |

|           | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                  |       |  |      |     |    |        |

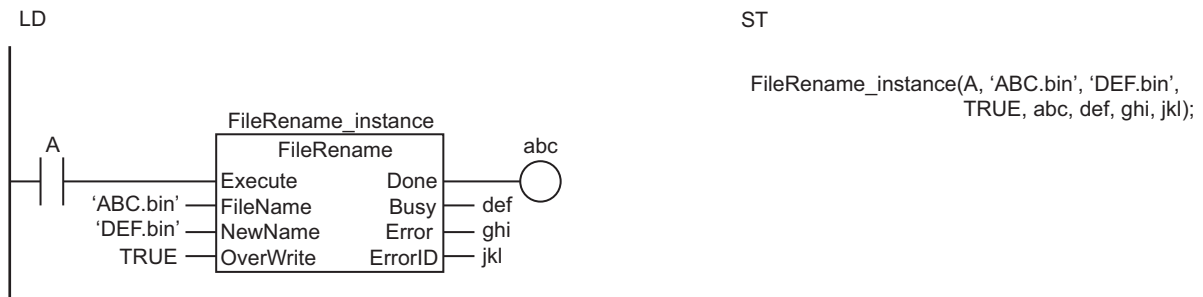
## Function

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.

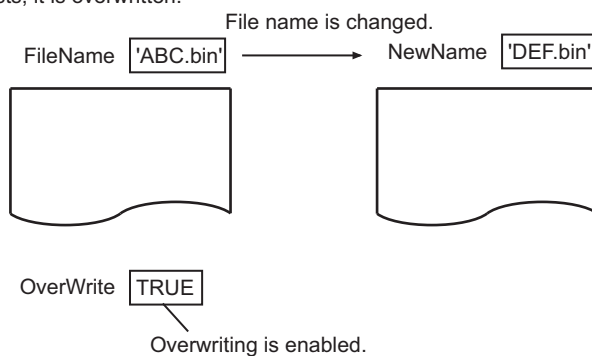
If a file or directory with the name *NewName* already exists in the SD Memory Card, the following processing is performed depending on the value of *OverWrite* (overwrite enable).

| Value of <i>OverWrite</i>   | Description   |
|-----------------------------|---|
| 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.'



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *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*.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.
- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted.
- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) The file directory specified with *FileName* does not exist.
  - d) The value of *FileName* or *NewName* is not a valid file name or directory name.
  - e) The file specified by *FileName* is being accessed.
  - f) There is a subdirectory in the directory that was specified for *FileName*, and the value of *OverWrite* is TRUE.
  - g) A file with the name *NewName* already exists, and the value of *OverWrite* is FALSE.
  - h) A file with the name *NewName* already exists, the file is write protected, and the value of *OverWrite* is TRUE.
  - i) The value of *NewName* exceeds the maximum number of bytes allowed in a file name or directory name.
  - j) The maximum number of directories is exceeded.
  - k) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: *FileWriteVar*, *FileReadVar*, *FileCopy*, *DirCreate*, *FileRemove*, *DirRemove*, and *FileRename*.
  - l) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

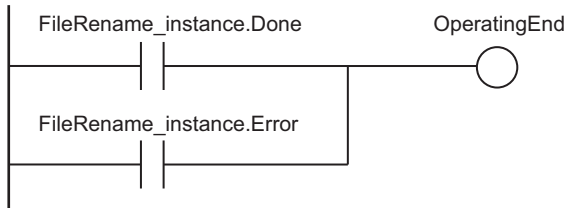
In this sample, the name of the file 'ABC.bin' is changed to 'DEF.bin' on the SD Memory Card.

### LD

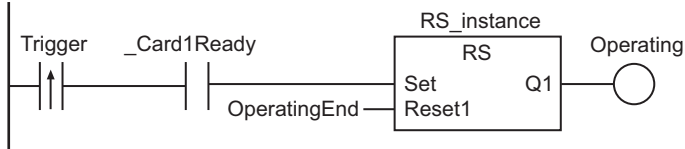
| Internal Variables | Variable            | Data type  | Initial value | Comment               |
|--------------------|---------------------|------------|---------------|-----------------------|
|                    | OperatingEnd        | BOOL       | FALSE         | Processing completed. |
|                    | 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 |

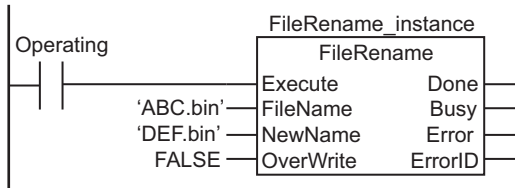
Determine if execution of the FileRename instruction is completed.



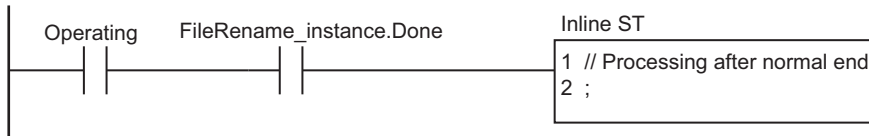
Accept trigger.

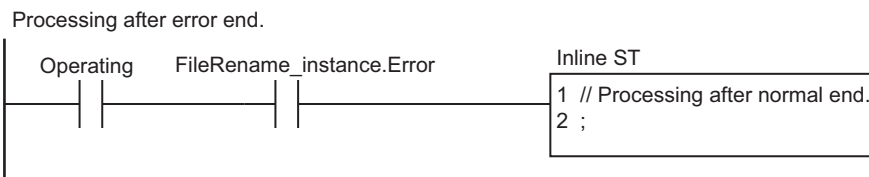


Execute FileRename instruction.



Processing after normal end.



**ST**

| Internal Variables | Variable            | Data type  | Initial value | Comment   |
|--------------------|---------------------|------------|---------------|---|
|                    | Trigger             | BOOL       | FALSE         | Execution condition                               |
|                    | LastTrigger         | BOOL       | FALSE         | Value of <i>Trigger</i> 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/<br>FUN | Graphic expression | ST expression   |
|-------------|------------------|------------|--------------------|---|
| DirCreate   | Create Directory | FB         |                    | DirCreate_instance(Execute, DirName, Done, Busy, Error, ErrorID); |

## Variables

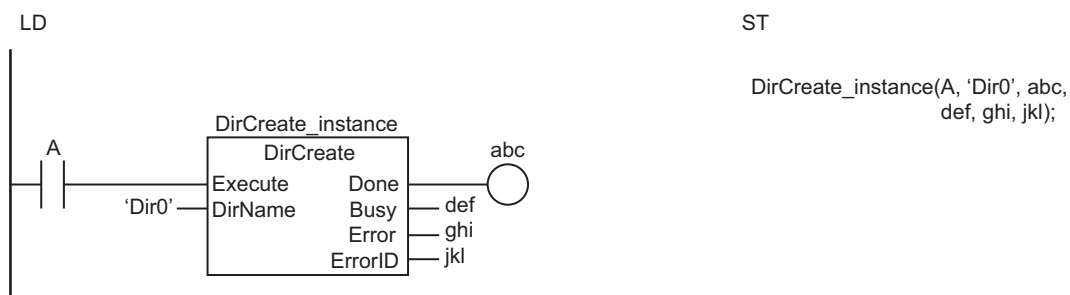
|         | Meaning             | I/O   | Description                 | Valid range  | Unit | Default |
|---------|---------------------|-------|-----------------------------|--|------|---------|
| DirName | Directory to create | Input | Name of directory to create | 66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |

|         | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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     |

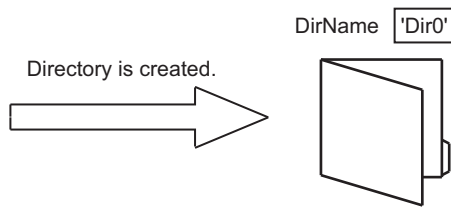
## Function

The DirCreate instruction creates a directory named by *DirName* (directory to create) in the SD Memory Card.

The following figure shows a programming example. Here, a directory named 'Dir0' is created.



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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                 |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                               |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.  |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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.*2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.



- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted.
- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) There is insufficient space available on the SD Memory Card.
  - d) The file specified by *FileName* is being accessed.
  - e) The maximum number of directories is exceeded.
  - f) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename.
  - g) The directory specified by *DirName* already exists.
  - h) The value of *DirName* is not a valid directory name.
  - i) The value of *DirName* exceeds the maximum number of bytes allowed in a directory name.
  - j) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

Refer to *Sample Programming* on page 2-1486 for the FileCopy instruction.

# DirRemove

The DirRemove instruction deletes the specified directory from the SD Memory Card.

| Instruction | Name             | FB/<br>FUN | Graphic expression  | ST expression  |
|-------------|------------------|------------|---|--|
| DirRemove   | Delete Directory | FB         | <pre> DirRemove_instance ┌───────────┐ │ DirRemove │ ├───────────┤ │ Execute   │ Done   │ ├───────────┤ │ DirName   │ Busy  │ ├───────────┤ │ All       │ Error │ ├───────────┤ │           │ ErrorID│ └───────────┘ </pre> | DirRemove_instance(Execute, DirName, All, Done, Busy, Error, ErrorID); |

## Variables

|         | Meaning             | I/O   | Description  | Valid range   | Unit | Default |
|---------|---------------------|-------|--|---|------|---------|
| DirName | Directory to delete | Input | Directory to delete  | 66 bytes max.<br>(65 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |
| All     | All designation     |       | Specifies whether to delete files and subdirectories inside specified directory<br>TRUE: Delete files and subdirectories.<br>FALSE: Do not delete. | Depends on data type.   |      | FALSE   |

|         | Boo lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real numbers |       | Times, durations, 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       |             |      |       |       |          |      |       |       |      |     |      |      |              |       |   |      |     |    |        |

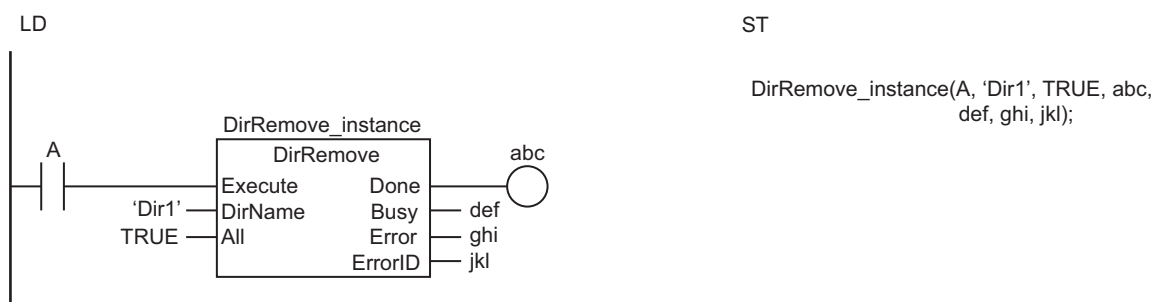
## Function

The DirRemove instruction deletes the directory specified by *DirName* (directory to delete) 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* (all designation).

| Value of <i>All</i> | Description  |
|---------------------|--|
| TRUE                | All files and subdirectories are deleted along with the specified directory. |
| FALSE               | The specified directory is not deleted and an error occurs.                  |

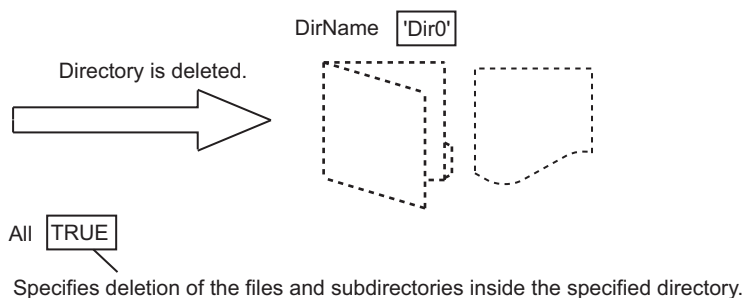
The following figure shows a programming example. Here, a directory named 'Dir1' is deleted.



```

    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<br>TRUE: Can be used.<br>FALSE: Cannot be used.                                  |
| _Card1Protect   | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected.                                |
| _Card1Err       | SD Memory Card Error Flag              | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.   |
| _Card1Access    | SD Memory Card Access Flag             | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>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. *2 This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- When the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, any open file is closed by the system. Any read or write operations in progress are continued up to the end.
- If a file is open when the power supply is stopped by pressing the SD Memory Card power supply switch, the file is not corrupted.
- 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 may be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- 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 will occur in the following cases. *Error* will change to TRUE.
  - a) The SD Memory Card is not in a usable condition.
  - b) The SD Memory Card is write protected.
  - c) The value of *All* is TRUE, and the directory specified with *DirName* is being accessed by another instruction.
  - d) The value of *All* is FALSE, and the directory specified with *DirName* contains files or directories.
  - e) The directory specified by *DirName* is write-protected.
  - f) The directory specified by *DirName* contains write-protected files or write-protected directories.
  - g) The directory specified by *DirName* does not exist.
  - h) The value of *DirName* exceeds the maximum number of bytes allowed in a directory name.
  - i) Five or more of the following SD Memory Card instructions, which do not have *FileID*, are executed at the same time: *FileWriteVar*, *FileReadVar*, *FileCopy*, *DirCreate*, *FileRemove*, *DirRemove*, and *FileRename*.
  - j) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## Sample Programming

Refer to *Sample Programming* on page 2-1486 for the *FileCopy* instruction.

# BackupToMemoryCard

The BackupToMemoryCard instruction backs up data to an SD Memory Card.

| Instruction        | Name                  | FB/<br>FUN | Graphic expression | ST expression   |
|--------------------|-----------------------|------------|--------------------|---|
| BackupToMemoryCard | SD Memory Card Backup | FB         |                    | BackupToMemoryCard _instance(Execute, DirName, Cancel, Option, Done, Busy, Error, Canceled, ErrorID); |



## 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.

## Variables

|          | 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               |        | Canceling the backup<br>TRUE: Cancel<br>FALSE: Do not cancel   | Depends on data type.  |      | FALSE   |
| Option   | For future expansion |        | 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<br>TRUE: Canceling completed<br>FALSE: Canceling failed | Depends on data type.  | ---  | ---     |

|         | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

|          | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|----------|---|-------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|          | BOOL  | BYTE        | WORD | DWORD | LWORD | USINT    | UINT | UDINT | ULINT | SINT | INT | DINT | LINT                 | REAL | LREAL  | TIME | DATE | TOD | DT | STRING |
| 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.

Use *DirName* to specify the name of the directory in which to save the backup data.

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*, data is saved as below.

| Instruction execution    | Directory to save in               |
|--------------------------|------------------------------------|
| 1st execution            | Root directory                     |
| 2nd execution and beyond | The previously specified directory |

If the directory 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* is changed 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 the cancellation is completed, the value of *Canceled* changes to TRUE.

If the change of *Cancel* to TRUE is not completed in time, the value change for cancellation may not be received in time, and the backup process may be continued until completed. If the value change for cancellation is not received 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.

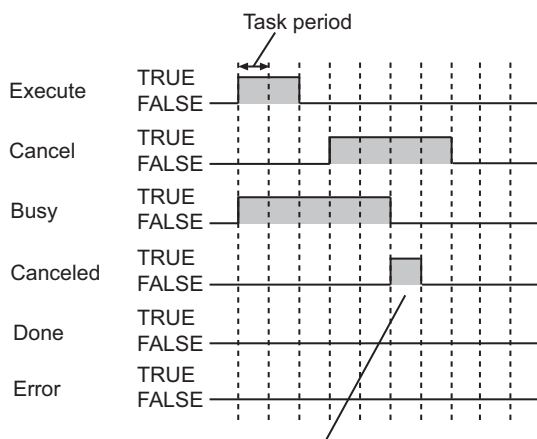
*Option* is for future expansion. Do not connect a parameter to it.

## Timing Chart for Canceling

Timing charts for the instruction variables are provided below for canceling.

### ● To Change *Execute* to FALSE Before *Canceled* Changes to TRUE for a Successful Cancellation

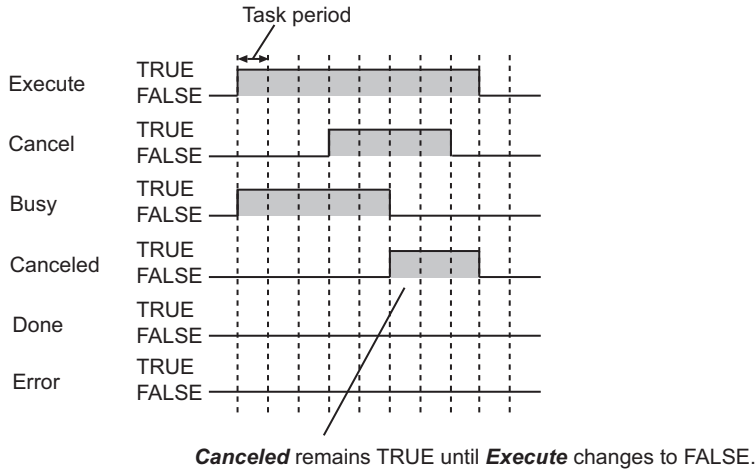
- Backup processing is executed when *Execute* is changed to TRUE. The value of *Busy* changes to TRUE.
- Backup processing is canceled when *Cancel* is changed to TRUE.
- When the cancellation is completed, *Busy* changes to FALSE, and *Canceled* changes to TRUE.
- Change the value of *Execute* to FALSE before *Canceled* changes to TRUE.
- The value of *Canceled* changes to FALSE after one task period.
- Since the cancellation is successfully completed, *Done* remains FALSE.



*Canceled* changes to TRUE for one task period.

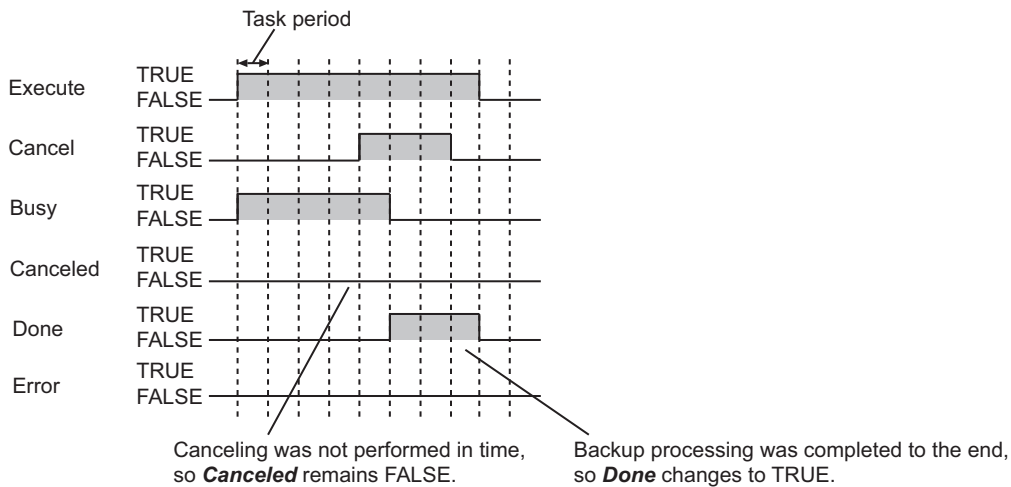
### ● To Change *Execute* to FALSE After *Canceled* Changes to TRUE for a Successful Cancellation

- Backup processing is executed when *Execute* is changed to TRUE. The value of *Busy* changes to TRUE.
- Backup processing is canceled when *Cancel* is changed to TRUE.
- When the cancellation is completed, *Busy* changes to FALSE, and *Canceled* changes to TRUE.
- Change the value of *Execute* to FALSE after *Canceled* changes to TRUE.
- The value of *Canceled* remains TRUE until *Execute* changes to FALSE.
- Since the cancellation is successfully completed, *Done* remains FALSE.



● **When Canceling Is Not Performed in Time**

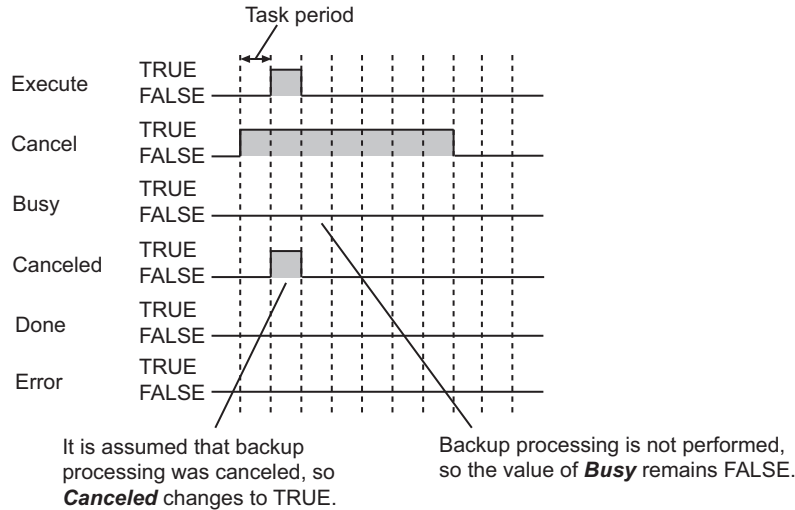
- Backup processing is executed when *Execute* is changed to TRUE. The value of *Busy* changes to TRUE.
- The value of *Cancel* is changed to TRUE. The value change for cancellation is not received in time, and the backup process is continued.
- When the backup process is completed, the value of *Busy* changes to FALSE.
- The backup process was continued to the end, so the value of *Done* changes to TRUE.
- The cancellation was not received in time, so the value of *Canceled* remains FALSE.



● **To Change Execute to TRUE While Cancel Is TRUE**

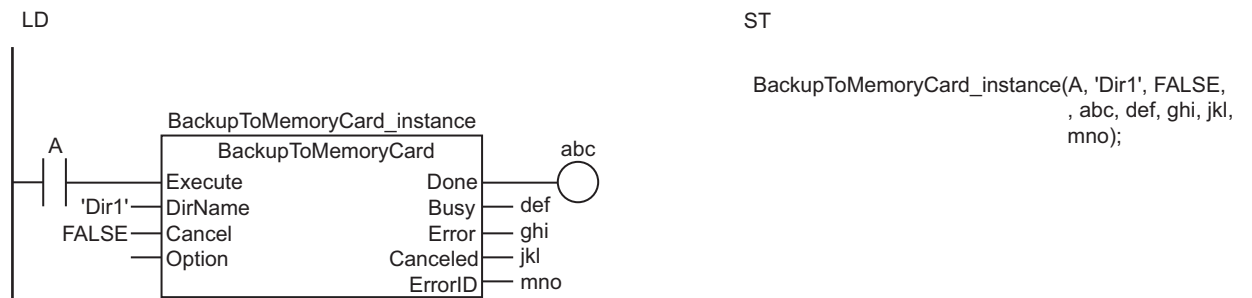
- Change the value of *Cancel* to TRUE.
- Backup processing is not executed even if *Execute* is changed to TRUE. Therefore, the value of *Busy* remains FALSE.
- It is assumed that backup processing was canceled, so *Canceled* changes to TRUE.
- If the value of *Execute* is changed to FALSE, *Canceled* changes to FALSE.





## Notation Example

The following figure shows a programming example. The backup file is saved in a directory called 'Dir1'.



## 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<br>TRUE: Can be used.<br>FALSE: Cannot be used.   |
| _Card1Protect | 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.<br>TRUE: Write protected.<br>FALSE: Not write protected. |
| _Card1Err     | SD Memory Card Error Flag           | BOOL      | This flag indicates if an SD Memory Card that cannot be used is mounted or if a format error occurs.<br>TRUE: Error.<br>FALSE: No error.                    |
| _Card1Access  | SD Memory Card Access Flag          | BOOL      | This flag indicates if the SD Memory Card is currently being accessed.<br>TRUE: Being accessed.<br>FALSE: Not being accessed.                               |

| Name               | Meaning                                | Data type | Description  |
|--------------------|--|-----------|--|
| _Card1Deteriorated | SD Memory Card Life Warning Flag       | BOOL      | This flag indicates if the end of the life of the SD Memory Card is detected.<br>TRUE: End of life detected.<br>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.* <sup>2</sup> This flag is not cleared automatically.<br>TRUE: Error.<br>FALSE: No error.            |
| _BackupBusy        | Backup Function Busy Flag              | BOOL      | This flag indicates if a backup, restoration, or verification is in progress.<br>TRUE: Backup, restore, or compare operation is in progress.<br>FALSE: Backup, restore, or compare operation is not in progress. |

\*1. It is a precondition that the SD Memory Card is physically inserted and mounted normally.

\*2. This indicates an access to the SD Memory Card.

## Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* 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 completed even when the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when the execution is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart of *Execute*, *Done*, *Busy*, and *Error*.
- 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, which are related to backup, do not change when this instruction is executed.
  - a) SD Memory Card Backup Command: `_CardBkupCmd`
  - b) 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.
  - a) The SD Memory Card is not in a usable condition.

- b) The SD Memory Card is write protected.
- c) There is insufficient space available on the SD Memory Card.
- d) The maximum number of files or directories is exceeded.
- e) A file already exists with the name specified with *DirName*.
- f) The value of *DirName* is not a valid directory name.
- g) Another backup operation is already in progress.
- h) Backup processing failed.
- i) While the SD Memory Card is being accessed, an error occurs and causes an access failure.

## 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.   |

## Definitions of Global Variables

### ● 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                  |

| Variable          | Data type | Initial value | Comment                         |
|-------------------|-----------|---------------|---------------------------------|
| 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        |

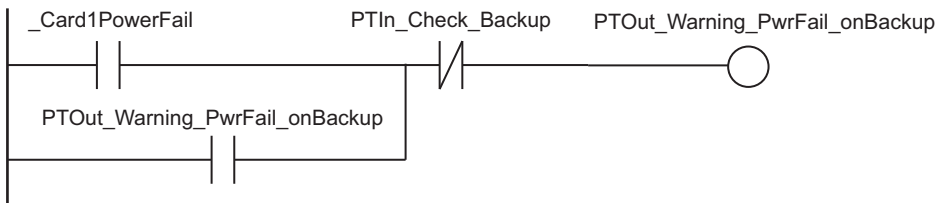
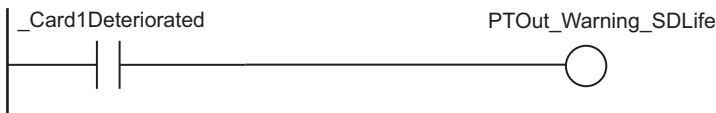
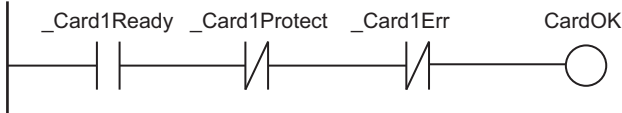
## LD

| Internal Variables | Variable        | Data type          | Initial value   | Comment   |
|--------------------|-----------------|--------------------|---|---|
|                    | CardOK          | BOOL               | FALSE   | SD Memory Card Normal Flag  |
|                    | Backup_inst     | BackupToMemoryCard |   | Instance of BackupToMemoryCard 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.   |

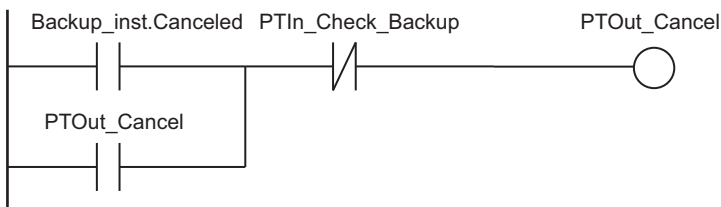
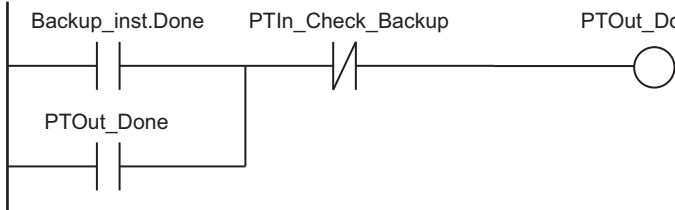
| External Variables | Variable           | Data type | Constant | Comment                                |
|--------------------|--------------------|-----------|----------|--|
|                    | _Card1Ready        | BOOL      | ☑        | SD Memory Card Ready Flag              |
|                    | _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              |

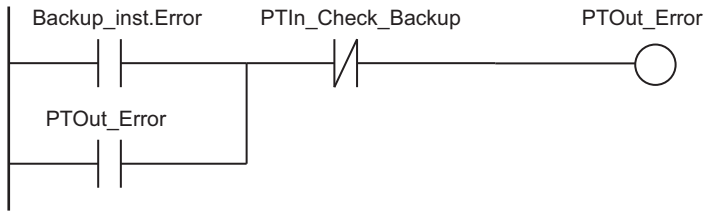
| External Variables | Variable                        | Data type | Constant                 | Comment   |
|--------------------|---------------------------------|-----------|--------------------------|---|
|                    | PTOut_Warning_SDLife            | BOOL      | <input type="checkbox"/> | Output to SD Memory Card life warning lamp      |
|                    | PTOut_Warning_PwrFail_on-Backup | BOOL      | <input type="checkbox"/> | Output to SD Memory Card power interrupted lamp |
|                    | PTOut_Done                      | BOOL      | <input type="checkbox"/> | Output to backup normal end lamp                |
|                    | PTOut_Cancel                    | BOOL      | <input type="checkbox"/> | Output to backup canceled lamp                  |
|                    | PTOut_Error                     | BOOL      | <input type="checkbox"/> | Output to backup error end lamp                 |
|                    | PTIn_Check_Backup               | BOOL      | <input type="checkbox"/> | Input from lamps OFF button                     |
|                    | PTIn_Cancel                     | BOOL      | <input type="checkbox"/> | Input from cancel button                        |

Check status of SD Memory Card.

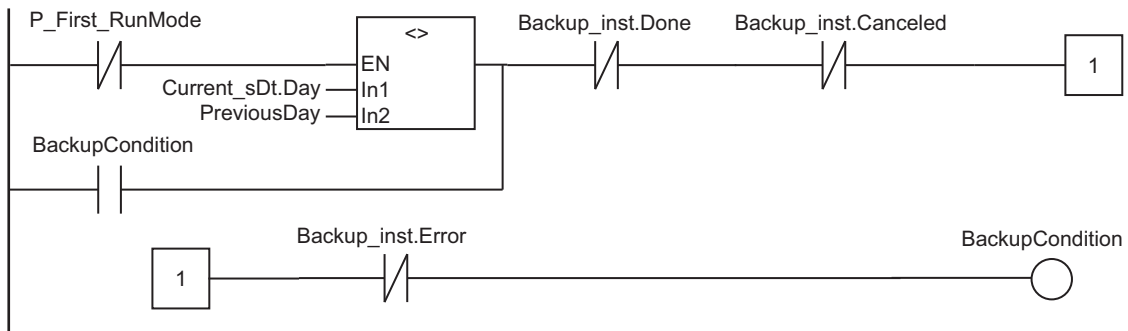
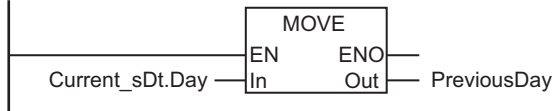


Light the Backup Normal End Lamp, Canceled Lamp, or Error End Lamp as required.

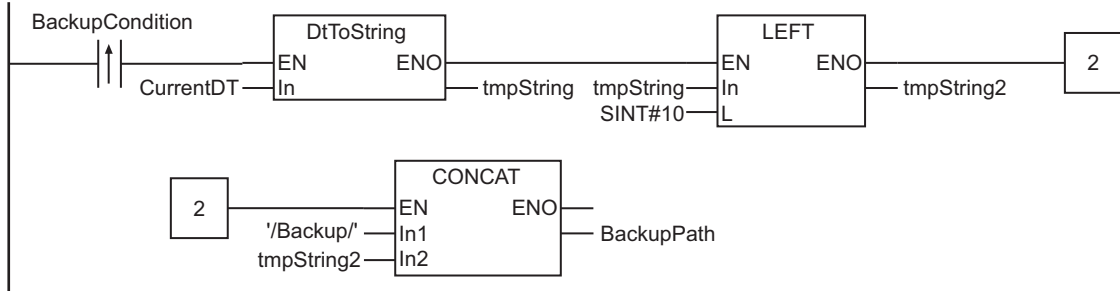




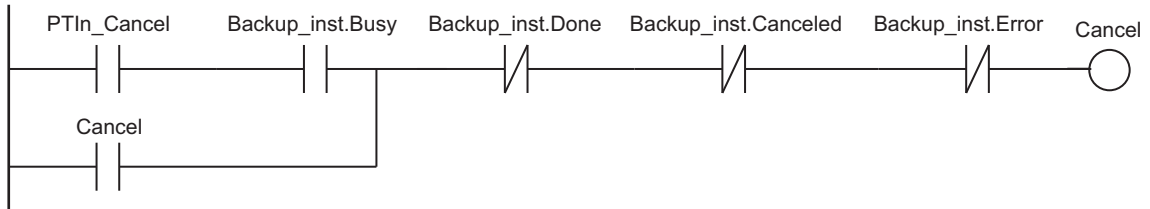
See if date has changed.



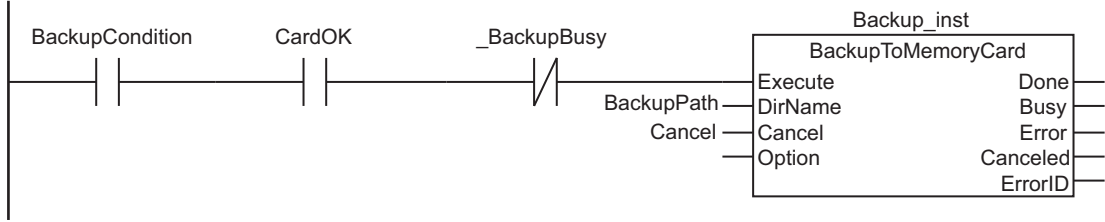
Create directory name.



Detect pressing of the Cancel Button.



Execute BackupToMemoryCard instruction.



**ST**

**2**

| Internal Variables | Variable        | Data type          | Initial value   | Comment   |
|--------------------|-----------------|--------------------|---|---|
|                    | CardOK          | BOOL               | FALSE   | SD Memory Card Normal Flag  |
|                    | Backup_inst     | BackupToMemoryCard |   | Instance of BackupToMemoryCard 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  |
|                    | RS1             | RS                 |   | Instance 1 of Reset-Priority Keep instruction   |
|                    | RS2             | RS                 |   | Instance 2 of Reset-Priority Keep instruction   |
|                    | RS3             | RS                 |   | Instance 3 of Reset-Priority Keep instruction   |
|                    | RS4             | RS                 |   | Instance 4 of Reset-Priority Keep instruction   |
|                    | RS5             | RS                 |   | Instance 5 of Reset-Priority Keep instruction   |
|                    | RS6             | RS                 |   | Instance 6 of Reset-Priority 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_PwrFail_on-Backup | 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

| Instruction         | Name   | Page        |
|---------------------|--|-------------|
| NX_DOutTimeStamp    | Write Digital Output with Specified Time Stamp       | page 2-1524 |
| NX_AryDOutTimeStamp | Write Digital Output Array with Specified Time Stamp | page 2-1530 |

# 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/<br>FUN | Graphic expression | ST expression  |
|------------------|--|------------|--------------------|--|
| NX_DOutTimeStamp | Write Digital Output with Specified Time Stamp | FB         |                    | NX_DOutTimeStamp_instance( Enable, SetDOut, SetTimeStamp, SyncOutTime, DOut, TimeStamp); |



## 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.

## Variables

|              | Meaning                          | I/O    | Description   | Valid range           | Unit | Default |    |
|--------------|----------------------------------|--------|---|-----------------------|------|---------|----|
| Enable       | Enable                           | Input  | TRUE: Value of <i>SetDOut</i> is output.<br>FALSE: Output changes to FALSE when this variable changes to FALSE.           | Depends on data type. | ---  | FALSE   |    |
| SetDOut      | Output value                     |        | Output value  |                       |      |         |    |
| SetTimeStamp | Specified time stamp             |        | Time to output value  |                       |      |         | 0  |
| SyncOutTime  | Time stamp of synchronous output |        | The <i>Time Stamp of Synchronous Output</i> device variable of the EtherCAT Coupler Unit or an NX Unit on the CPU Unit    |                       |      |         | ns |
| DOut         | DOut Unit output bit             | In-out | The <i>Output Bit</i> ** device variable of the Digital Output Unit that supports time stamp refreshing                   | Depends on data type. | ---  | ---     |    |
| TimeStamp    | Time stamp                       |        | The <i>Output Bit</i> ** <i>Time Stamp</i> device variable of the Digital Output Unit that supports time stamp refreshing |                       |      |         | ns |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                   | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| SetTimeS-<br>tamp |             |             |      |       |       |       |          |       | OK    |      |     |      |      |                      |       |  |      |     |    |        |
| SyncOut-<br>Time  |             |             |      |       |       |       |          |       | OK    |      |     |      |      |                      |       |  |      |     |    |        |
| DOut              | OK          |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| TimeStamp         |             |             |      |       |       |       |          |       | OK    |      |     |      |      |                      |       |  |      |     |    |        |

## Function

When the value of *Enable* is TRUE, the NX\_DOutTimeStamp instruction writes *SetDOut* (Output value) at the specified time to the output bit of a Digital Output Unit that supports time stamp refreshing. When *Enable* changes to FALSE, the value of the output bit changes to FALSE from the next task period.

The time difference between the specified time and the output time is  $\pm 1 \mu\text{s}$  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.

Specify *DOut* (DOut Unit output bit) with the *Output Bit \*\** device variable that is assigned to the output bit of the Digital Output Unit that supports time stamp refreshing.

Specify *TimeStamp* with 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 result of the addition to *SetTimeStamp* (Specified time stamp) in the NX\_DOutTimeS-tamp instruction.

For details, refer to *Sample Programming* on page 2-1526 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, writing may not occur at the specified time. In this case, the value is output in the next task period or later.
- If any device variables used for this instruction are modified or accessed for any other instruction or program, perform exclusive control.
- Specify *SyncOutTime* with 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, to which the Digital Output Unit that supports time stamp refreshing is connected. However, an error will not occur even if another variable is specified.
- Specify *DOut* and *TimeStamp* with the device variables of the Digital Output Unit that supports time stamp refreshing. However, an error will not occur even if another variable is specified.
- Specify *DOut* and *TimeStamp* with the device variables for the same channel of the same Unit. However, an error will not occur even if another variable is specified.
- If the value of *TimeStamp* is set as a time in the past, the value of *TimeStamp* becomes 0. In this case, the output of the Digital Output Unit that supports time stamp refreshing is immediately refreshed.

Refer to the *NX-series Digital I/O Units User's Manual (Cat. No. W521)* for more details.

## Sample Programming

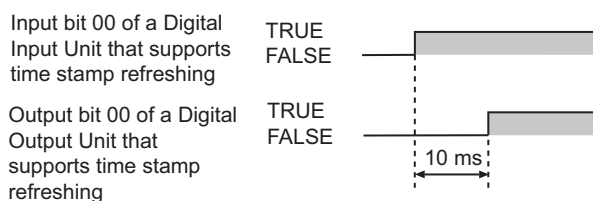
In this sample, 10 ms after the value of input bit 00 of a Digital Input Unit that supports time stamp refreshing changes to TRUE, output bit 00 of a Digital Output Unit that supports time stamp refreshing changes to TRUE.

It is assumed that the value of input bit 00 is TRUE for a longer period 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 a shorter period than the I/O refresh period of the NX bus, the change to TRUE in input bit 00 may not be detected. To solve this problem, for example, you could change the programming to use input changed time of input bit 00 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 when a specified time period expires after a change in a sensor input.



## Network Configuration

The network is configured with Units in the table below. A Slave Terminal configured with the following Units is connected at EtherCAT node address 1. The Units are assigned the device names as shown in the table.

| 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.

| Position | 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_Synchronous_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 |

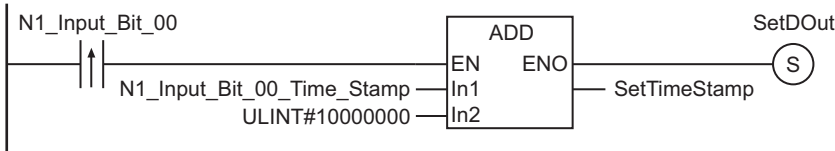
## LD

| Internal Variables | Variable     | Data type | Initial value | Comment              |
|--------------------|--------------|-----------|---------------|----------------------|
|                    | SetTimeStamp | ULINT     | 0             | Specified time stamp |

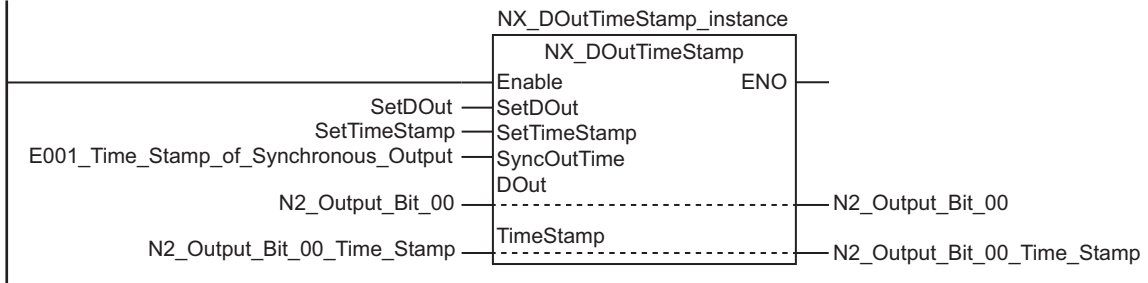
| Internal Variables | Variable                  | Data type        | Initial value | Comment      |
|--------------------|---------------------------|------------------|---------------|--------------|
|                    | SetDOut                   | BOOL             | FALSE         | Output value |
|                    | NX_DOutTimeStamp_instance | NX_DOutTimeStamp |               |              |

| External Variables | Variable                              | Data type | Constant                 | Comment   |
|--------------------|---------------------------------------|-----------|--------------------------|---|
|                    | N1_Input_Bit_00                       | BOOL      | <input type="checkbox"/> | Input bit 00  |
|                    | N1_Input_Bit_00_Time_Stamp            | ULINT     | <input type="checkbox"/> | Input changed time for input bit 00   |
|                    | E001_Time_Stamp_of_Synchronous_Output | ULINT     | <input type="checkbox"/> | Time stamp for the timing of synchronous outputs from the connected NX Unit |
|                    | N2_Output_Bit_00                      | BOOL      | <input type="checkbox"/> | Output bit 00   |
|                    | N2_Output_Bit_00_Time_Stamp           | ULINT     | <input type="checkbox"/> | Specified time for output bit 00  |

Specify the output time stamp.



Output



## 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/<br>FUN | Graphic expression | ST expression  |
|--------------------------|---|------------|--------------------|--|
| NX_AryDOut-<br>TimeStamp | Write Digital<br>Output Array<br>with Specified<br>Time Stamp | FB         |                    | NX_AryDOutTimeStamp_in-<br>stance(Enable, SetDOut, Syn-<br>cOutTime, DOut, TimeStamp); |



## 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.

## Variables

|              | Meaning                          | I/O    | Description  | Valid range           | Unit | Default |
|--------------|----------------------------------|--------|--|-----------------------|------|---------|
| Enable       | Enable                           | Input  | TRUE: Output changes according to the setting of <i>SetDOut</i> .<br>FALSE: Output changes to FALSE when this variable changes to FALSE. | Depends on data type. | ---  | FALSE   |
| SyncOut-Time | Time stamp of synchronous output |        | The <i>Time Stamp of Synchronous Output</i> device variable of the EtherCAT Coupler Unit or an NX Unit on the CPU Unit                   |                       | ns   | *1      |
| SetDOut      | Output pulses                    | In-out | Output pulses  | ---                   | ---  | ---     |
| DOut         | DOut Unit output bit             |        | The <i>Output Bit</i> ** device variable of the Digital Output Unit that supports time stamp refreshing                                  | Depends on data type. |      |         |
| TimeStamp    | Time stamp                       |        | The <i>Output Bit</i> ** <i>Time Stamp</i> device variable of the Digital Output Unit that supports time stamp refreshing                |                       |      |         |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|                  | Boo<br>lean  | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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-<br>Time |  |             |      |       |       |       |          |       | OK    |      |     |      |      |                      |       |  |      |     |    |        |
| SetDOut          | Refer to <i>Specifying the Output Pulses</i> on page 2-1532 for details on the structure <code>_sOUTPUT_REF</code> . |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| DOut             | OK   |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |
| TimeStamp        |  |             |      |       |       |       |          |       | OK    |      |     |      |      |                      |       |  |      |     |    |        |

## Function

When the value of *Enable* is TRUE, the NX\_AryDOutTimeStamp instruction outputs pulses set with *SetDOut* (Output pulses) at the specified time from a Digital Output Unit that supports time stamp refreshing.

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 time difference between the specified time and the output time is  $\pm 1 \mu\text{s}$  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.

Specify *DOut* (DOut Unit output bit) with the *Output Bit \*\** device variable that is assigned to the output bit of the Digital Output Unit that supports time stamp refreshing.

Specify *TimeStamp* with 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 result of the addition 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 result of the addition to SetDOut.OffTime[] in the NX\_AryDOutTimeStamp instruction.

## Specifying the Output Pulses

The data type of the *SetDOut* output pulse is structure `_sOUTPUT_REF`. The specifications are as follows:

| Name            | Meaning       | Description  | Data type                 | Valid range           | Unit | Default                |
|-----------------|---------------|--|---------------------------|-----------------------|------|------------------------|
| SetDOut         | Output pulses | Output pulses  | <code>_sOUTPUT_REF</code> | ---                   | ---  | ---                    |
| EnableOut       | Output enable | Output enable flag<br>TRUE: Enable <i>OnTime</i> and <i>OffTime</i> settings.<br>FALSE: Disable <i>OnTime</i> and <i>OffTime</i> 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 the elements |
| OffTime[] array | OFF times     | Times at which to turn OFF the output bit  | ARRAY[0..15] OF ULINT     |                       |      |                        |

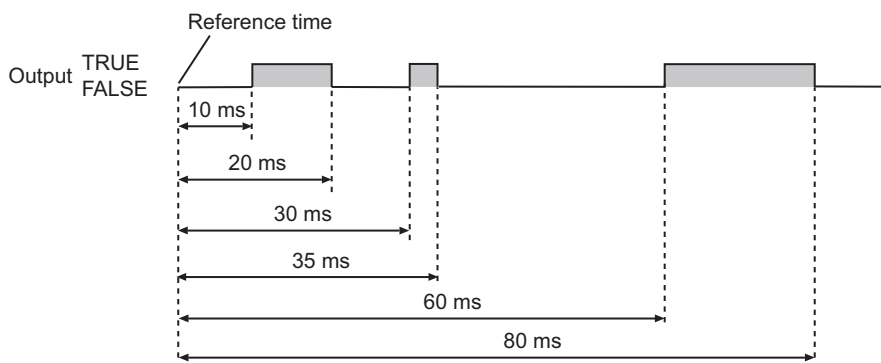
The OnTime[] (ON times) and OffTime[] (OFF times) arrays each have 16 elements.

Each corresponding element of the two arrays represents the ON time and OFF time for one pulse. Accordingly, you can specify up to 16 pulses with the array elements.

If both arrays have 0 at the same element number, the values of all of the subsequent elements are invalid.

For example, the figure below shows the output operation with the following values for 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 |

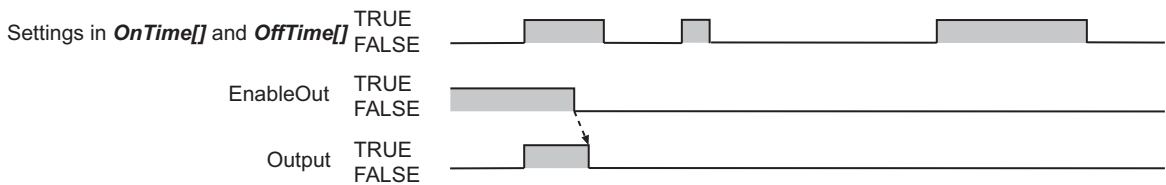
● **EnableOut (Output Enable)**

EnableOut (Output enable) enables the settings in OnTime[] and OffTime[].

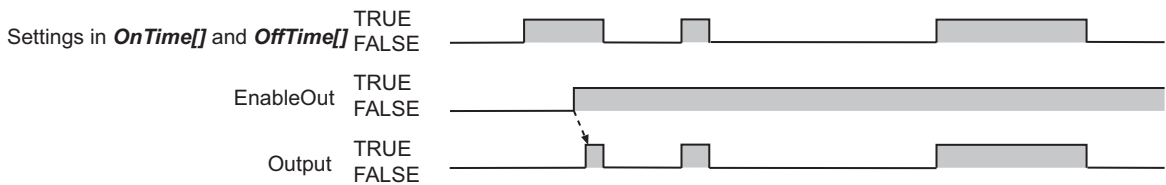
If the value of EnableOut is FALSE, the output value is FALSE regardless of the values in OnTime[] and OffTime[].

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.



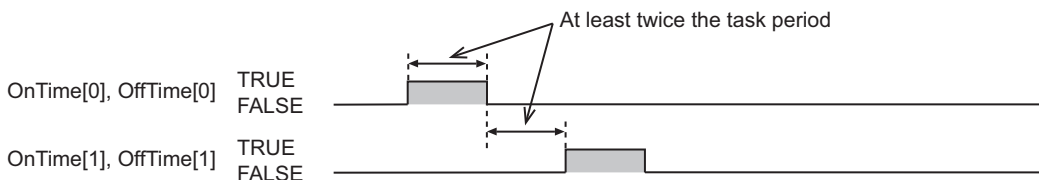
When the value of EnableOut changes to TRUE, the values in OnTime[] and OffTime[] are enabled.



● **Minimum Output Pulse Width**

To output pulses with a time accuracy of 1 μs, set each of the interval between OnTime[] and OffTime[] to at least twice the task period.

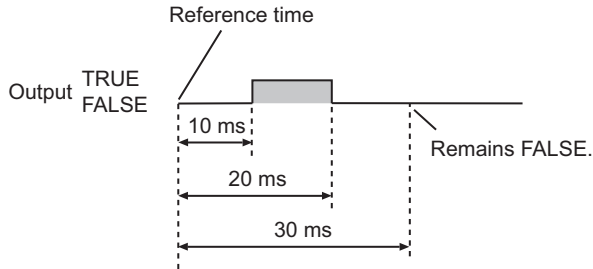
If the interval is less than two task periods, pulse output may not be performed as specified or may be delayed by one task period for the specified ON/OFF time.



● **When OnTime[] and OffTime[] contain the Same Value for the Same Element Number**

If OnTime[] and OffTime[] contain the same value at the same element number, the output will be FALSE. The figure below shows the output operation with the following values for 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 the Value of an Element in OnTime[] Is Larger Than That of OffTime[]**

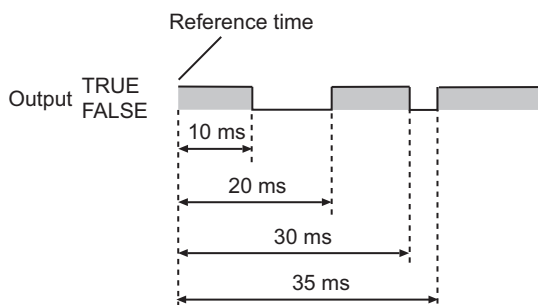
If the value of an element in OnTime[] is larger than that of the corresponding element of OffTime[], the output will change to FALSE and then back to TRUE.

If the lowest value of the elements of OnTime[] is larger than the lowest value of the elements of OffTime[], the output will change to TRUE immediately after execution of this instruction.

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 execution of this instruction.

The figure below shows the output operation with the following values for 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 |

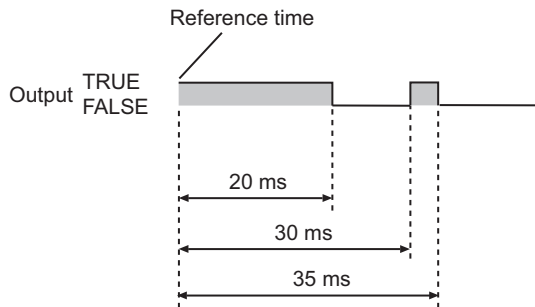


● **When the Value of One Element in Either OnTime[] or OffTime[] Is 0**

If the value of an element in either OnTime[] or OffTime[] is 0, the output will change to TRUE or FALSE immediately after execution of the instruction.

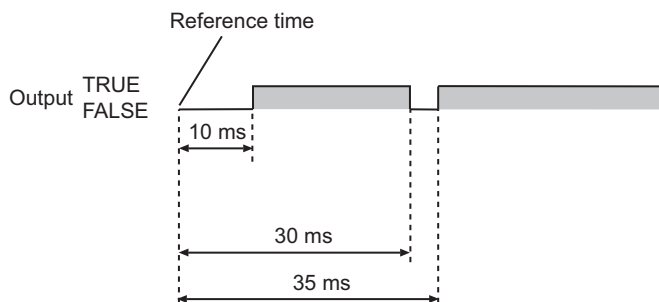
If an array element of OnTime[] is 0, the output will change to TRUE immediately after execution of the instruction. The figure below shows the output operation with the following values for the elements of the two arrays.

| Name      | Element numbers |             |   |
|-----------|-----------------|-------------|---|
|           | 0               | 1           | 2 |
| OnTime[]  | 0               | 30 ms later | 0 |
| OffTime[] | 20 ms later     | 35 ms later | 0 |



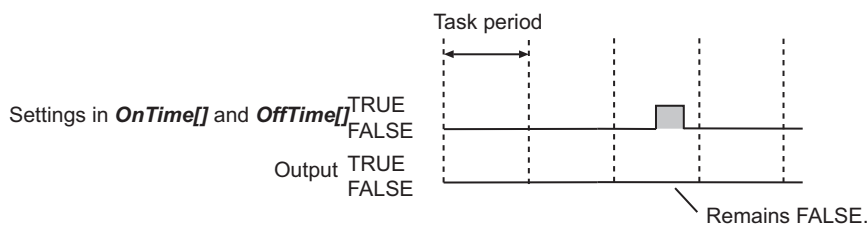
If an array element of OffTime[] is 0, the output will change to FALSE immediately after execution of the instruction. The figure below shows the output operation with the following values for the elements of the two arrays.

| Name      | Element numbers |             |   |
|-----------|-----------------|-------------|---|
|           | 0               | 1           | 2 |
| OnTime[]  | 10 ms later     | 35 ms later | 0 |
| OffTime[] | 0               | 30 ms later | 0 |



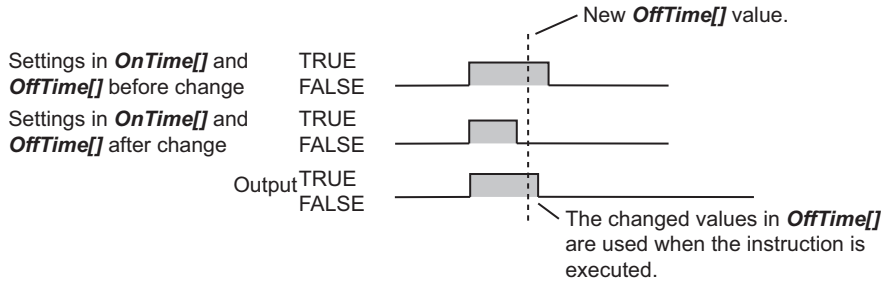
● **When the Output is Set to TRUE and Back to FALSE within One Task Period**

If the output is set to TRUE for one setting and back to FALSE for another setting within one task 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 will become valid the next time the instruction is executed.



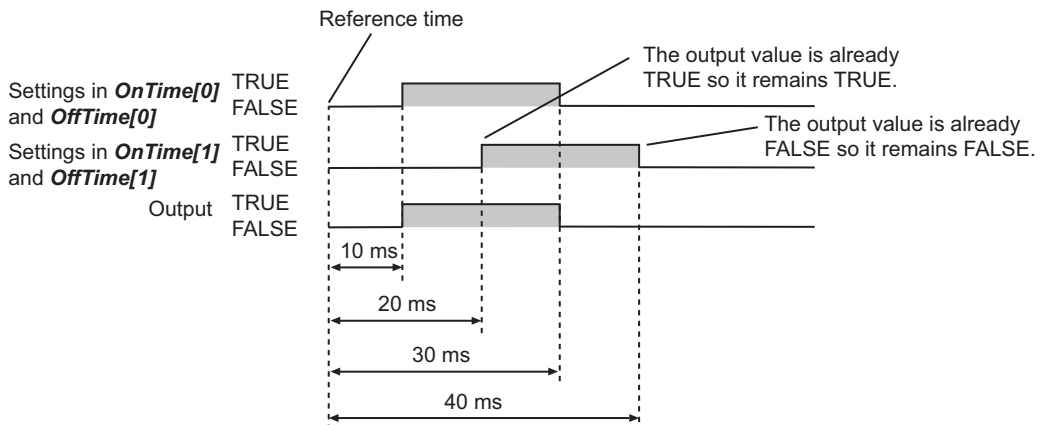
● **Overlapping TRUE Settings for an Output Value**

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.

The figure below shows the output operation with the following values for 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 |



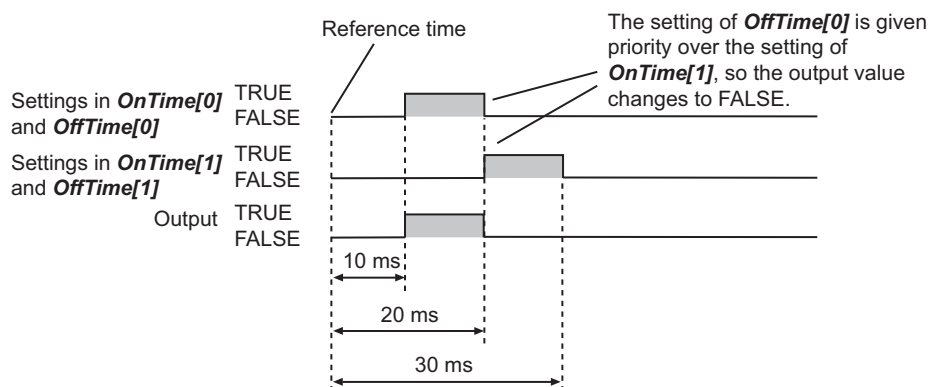
● **Simultaneous TRUE and FALSE Settings for an Output Value**

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.

The figure below shows the output operation with the following values for 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.

Refer to the *NJ/NX-series Motion Control Instructions Reference Manual (Cat. No. W508)* for the detailed specifications of the MC\_DigitalCamSwitch 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 output may not occur at the specified time. In this case, the value is output in the next task period or later.
- If any device variables used for this instruction are modified or accessed for any other instruction or program, perform exclusive control.
- Specify *SyncOutTime* with 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, to which the Digital Output Unit that supports time stamp refreshing is connected. However, an error will not occur even if another variable is specified.
- Specify *DOut* and *TimeStamp* with the device variables of the Digital Output Unit that supports time stamp refreshing. However, an error will not occur even if another variable is specified.
- Specify *DOut* and *TimeStamp* with the device variables for the same channel of the same Unit. However, an error will not occur even if another variable is specified.
- If the value of *TimeStamp* is set as a time in the past, the value of *TimeStamp* becomes 0. In this case, the output of the Digital Output Unit that supports time stamp refreshing is immediately refreshed.

Refer to the *NX-series Digital I/O Units User's Manual (Cat. No. W521)* for more details.

## Sample Programming

For sample programming, refer to the MC\_DigitalCamSwitch instruction in the *NJ/NX-series Motion Control Instructions Reference Manual (Cat. No. W508)*.

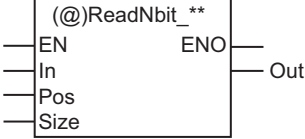


## Other Instructions

| Instruction       | Name  | Page        |
|-------------------|---|-------------|
| ReadNbit_**       | N-bit Read Group                            | page 2-1540 |
| WriteNbit_**      | N-bit Write Group                           | page 2-1542 |
| ChkRange          | Check Subrange Variable                     | page 2-1544 |
| GetMyTaskStatus   | Read Current Task Status                    | page 2-1547 |
| GetMyTaskInterval | Read Current Task Period                    | page 2-1550 |
| Task_IsActive     | Determine Task Status                       | page 2-1552 |
| Lock and Unlock   | Lock Tasks/Unlock Tasks                     | page 2-1554 |
| ActEventTask      | Activate Event Task                         | page 2-1560 |
| Get**Clk          | Get Clock Pulse Group                       | page 2-1567 |
| Get**Cnt          | Get Incrementing Free-running Counter Group | page 2-1569 |

# ReadNbit\_\*\*

The ReadNbit\_\*\* instructions read zero or more bits from a bit string.

| Instruction | Name             | FB/<br>FUN | Graphic expression  | ST expression  |
|-------------|------------------|------------|---|--|
| ReadNbit_** | N-bit Read Group | FUN        |  <p>**** must be a bit string data type.</p> | Out:=ReadNbit_**(In, Pos, Size);<br>**** must be a bit string data type. |

## Variables

|      | 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 <i>In</i> - 1 |      |         |
| Size | Read size     |        | Number of bits to read | 0 to No. of bits in <i>In</i>     |      |         |
| Out  | Read result   | Output | Read result            | Depends on data type.             | ---  | ---     |

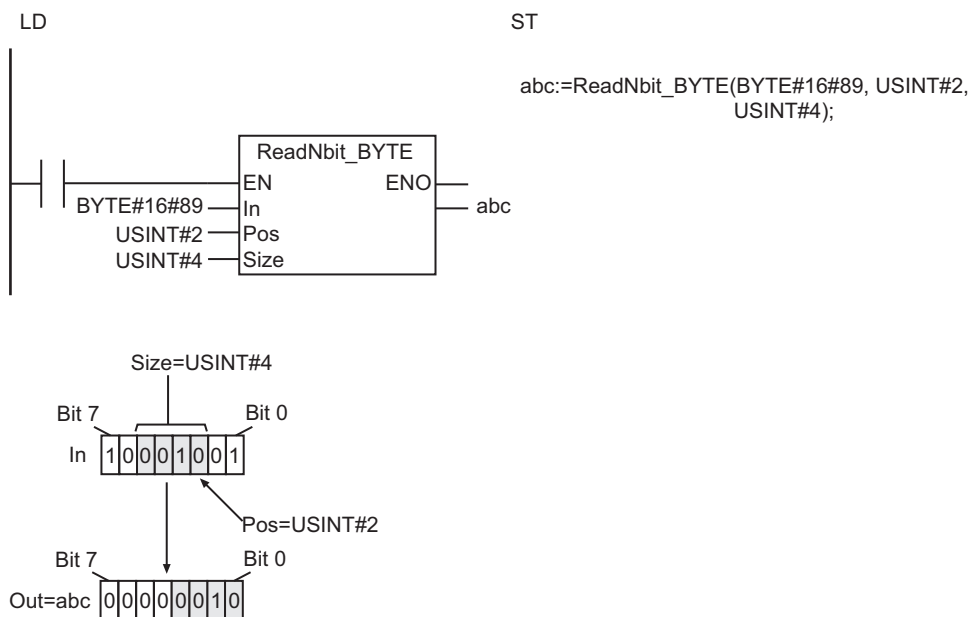
|      | Boo<br>lean                         | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|------|-------------------------------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|      | 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 <i>In</i> |             |      |       |       |       |          |       |       |      |     |      |      |                      |       |  |      |     |    |        |

## Function

The ReadNbit\_\*\* instruction reads the values of the upper *Size* bits from *Pos* (read position) in *In* (read source), and then assigns the values to *Out* (read result).

The name of the instruction is determined by the data type of *In* and *Out*. For example, if *In* and *Out* are both WORD data, the name of 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 the instruction, *WriteNbit\_\*\** on page 2-1542, 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 will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - a) The value of *Size* is outside the valid range.
  - b) The value of *Pos* is outside the valid range.
  - c) *In* does not have so many bits as specified by *Size* after the position specified by *Pos*.

# WriteNbit\_\*\*

The WriteNbit\_\*\* instructions write zero or more bits to a bit string.

| Instruction  | Name              | FB/<br>FUN | Graphic expression                         | ST expression  |
|--------------|-------------------|------------|--|--|
| WriteNbit_** | N-bit Write Group | FUN        | <p>*** must be a bit string data type.</p> | WriteNbit_**(In, InOut, Pos, Size);<br>*** must be a bit string data type. |

## Variables

|       | Meaning        | I/O    | Description   | Valid range                          | Unit | Default |
|-------|----------------|--------|---|--------------------------------------|------|---------|
| In    | Read source    | Input  | Bit string from which to read bits to write to <i>InOut</i> | Depends on data type.                | ---  | 0       |
| Pos   | Write position |        | Bit position to which to write                              | 0 to No. of bits in <i>InOut</i> - 1 |      |         |
| Size  | Write size     |        | Number of bits to write                                     | 0 to No. of bits in <i>In</i>        |      |         |
| InOut | Write target   | In-out | Write result  | Depends on data type.                | ---  | ---     |
| Out   | Return value   | Output | Always TRUE   | TRUE only                            | ---  | ---     |

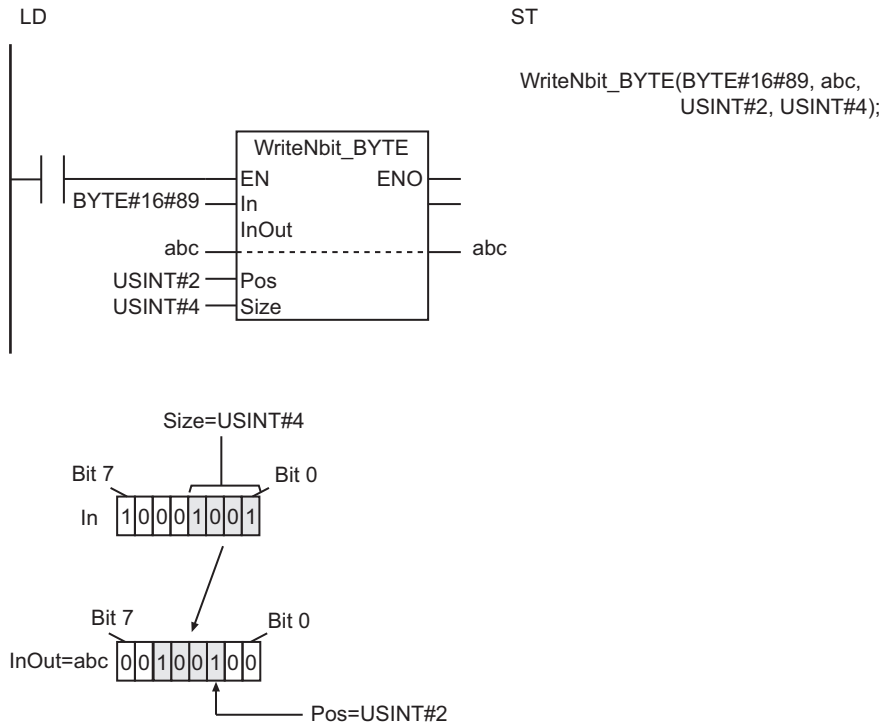
|       | Boo<br>lean | Bit strings                         |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |      |      |     |    |        |
|-------|-------------|-------------------------------------|------|-------|-------|----------|------|-------|-------|------|-----|------|----------------------|------|--|------|------|-----|----|--------|
|       | 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 <i>In</i> |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out   | OK          |                                     |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

## Function

The WriteNbit\_\*\* instruction first reads the values of the lower *Size* bits from *In* (read source). It writes the values to *Pos* (write position) in *InOut* (write target).

The name of the instruction is determined by the data type of *In* and *Out*. For example, if *In* and *Out* are both WORD data, the name of 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.



## Additional Information

Use the instruction, *ReadNbit\_\*\** on page 2-1540, 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.
  - a) The value of *Size* is outside of the valid range.
  - b) The value of *Pos* is outside of the valid range.
  - c) 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 specification.

| Instruction | Name                     | FB/<br>FUN | Graphic expression | ST expression           |
|-------------|--------------------------|------------|--------------------|-------------------------|
| ChkRange    | Check Sub-range Variable | FUN        |                    | Out:=ChkRange(In, Val); |

## Variables

|     | Meaning                      | I/O    | Description                  | Valid range                         | Unit | Default |
|-----|------------------------------|--------|------------------------------|-------------------------------------|------|---------|
| In  | Variable to check            | Input  | Variable to check            | Depends on data type.               | ---  | *1      |
| Val | Range specification variable |        | Range specification variable | Depends on the range specification. |      |         |
| Out | Check result                 | Output | Check result                 | Depends on data type.               | ---  | --      |

\*1. If you omit the input parameter, the default value is not applied. A building error will occur.

|     | Boo<br>lean   | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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 <i>In</i> . |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |
| Out | OK  |             |      |       |       |          |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

## Function

The ChkRange instruction determines if the value of *In* (variable to check) is within the valid range of the range specification variable *Val*.

If the value is within the valid range, the check result *Out* becomes TRUE. If the value is not within the valid range, the check result variable becomes 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.

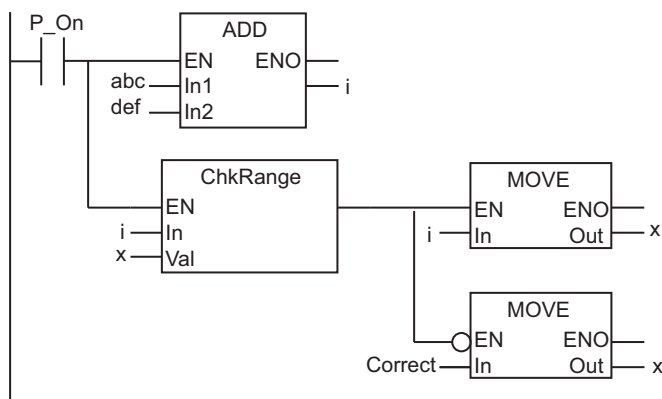
## Sample Programming

In the following examples, 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 within the valid range, the value of *i* is assigned to 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/<br>FUN | Graphic expression | ST expression   |
|----------------------|-----------------------------|------------|--------------------|---|
| GetMyTask-<br>Status | Read Current<br>Task Status | FUN        |                    | GetMyTaskStatus(LastExecTime,<br>MaxExecTime, MinExecTime, Ex-<br>ecCount, Exceeded, Exceed-<br>Count); |

## Variables

|                   | Meaning                          | I/O    | Description   | Valid range                  | Unit | Default |
|-------------------|----------------------------------|--------|---|------------------------------|------|---------|
| Out               | Return value                     | Output | Always TRUE   | TRUE only                    | ---  | ---     |
| LastExec-<br>Time | Last task execution<br>time      |        | Last task execution<br>time of the current task   | Depends on da-<br>ta type.*1 | ns   |         |
| MaxExec-<br>Time  | Maximum task execu-<br>tion time |        | Maximum task execu-<br>tion time of the current<br>task   |                              |      |         |
| MinExec-<br>Time  | Minimum task execu-<br>tion time |        | Minimum task execu-<br>tion time of the current<br>task   |                              |      |         |
| ExecCount         | Task execution count             |        | Number of task execu-<br>tions of the current<br>task   | Depends on da-<br>ta type.   | ---  |         |
| Exceeded          | Task period exceeded<br>flag     |        | TRUE: The last execu-<br>tion of the current task<br>was not completed<br>within the task period.<br>FALSE: The last execu-<br>tion of the current<br>task was completed<br>within the task period. |                              |      |         |
| Exceed-<br>Count  | Task period exceeded<br>count    |        | The number of times<br>the current task has<br>exceeded the task pe-<br>riod.   |                              |      |         |

\*1. Negative numbers are excluded.

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real<br>num-<br>bers |      | Times, durations,<br>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       |      |       |       |      |     |      |                      |      |  |      |      |     |    |        |

|                   | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      |      | Real<br>num-<br>bers |      | Times, durations,<br>dates, and text strings |     |    |        |  |
|-------------------|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|------|----------------------|------|--|-----|----|--------|--|
|                   | BOOL        | BYTE        | WORD | DWORD | LWORD | USINT | UINT     | UDINT | ULINT | SINT | INT | DINT | LINT | REAL | LREAL                | TIME | DATE   | TOD | DT | STRING |  |
| LastExec-<br>Time |             |             |      |       |       |       |          |       |       |      |     |      |      |      |                      | OK   |  |     |    |        |  |
| MaxExec-<br>Time  |             |             |      |       |       |       |          |       |       |      |     |      |      |      |                      | OK   |  |     |    |        |  |
| MinExec-<br>Time  |             |             |      |       |       |       |          |       |       |      |     |      |      |      |                      | OK   |  |     |    |        |  |
| ExecCount         |             |             |      |       |       |       |          | OK    |       |      |     |      |      |      |                      |      |  |     |    |        |  |
| Exceeded          | OK          |             |      |       |       |       |          |       |       |      |     |      |      |      |                      |      |  |     |    |        |  |
| Exceed-<br>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 timing below.

- 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.

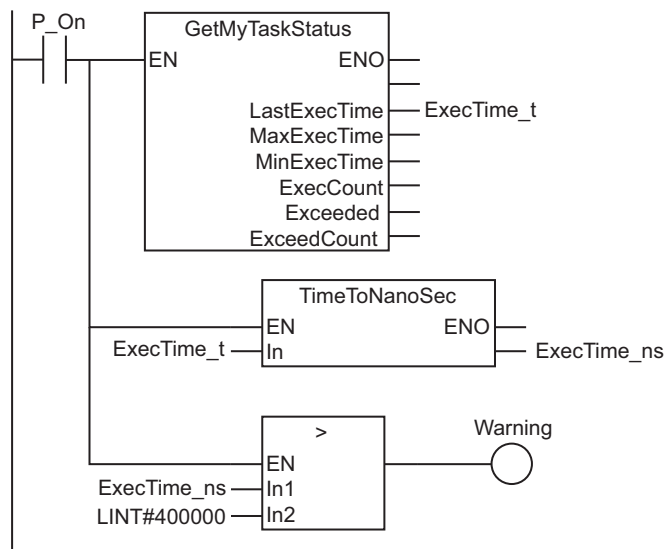
## Sample Programming

In this sample, the `GetMyTaskStatus` reads the status of the current task.

If the previous task execution time exceeds 400  $\mu$ s (400,000 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 | LINT      | 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  |

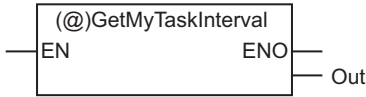
```

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/<br>FUN | Graphic expression   | ST expression             |
|-------------------|--------------------------|------------|--|---------------------------|
| GetMyTaskInterval | Read Current Task Period | FUN        |  | Out:=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.

## Variables

|     | 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.

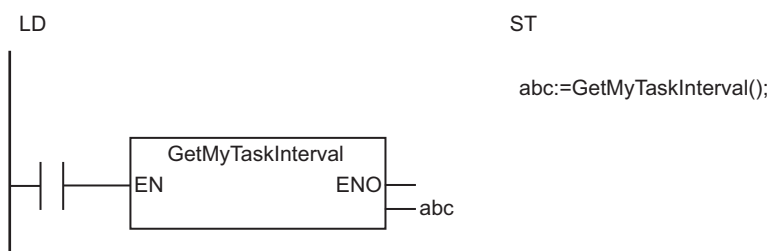
|     | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>dates, and text strings |      |     |    |        |
|-----|-------------|-------------|------|-------|-------|-------|----------|-------|-------|------|-----|------|------|----------------------|-------|--|------|-----|----|--------|
|     | 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#0 s.

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#1 ms.



## 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.

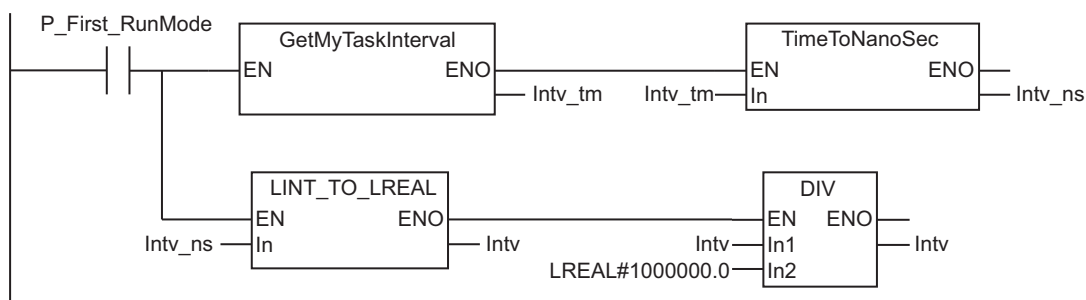
This sample programming can be used, for example, to calculate the axis target position for each task period.

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/<br>FUN | Graphic expression | ST expression                 |
|---------------|-----------------------|------------|--------------------|-------------------------------|
| Task_IsActive | Determine Task Status | FUN        |                    | Out:=Task_IsActive(TaskName); |

## Variables

|          | Meaning   | I/O    | Description  | Valid range   | Unit | Default |
|----------|-----------|--------|--|---|------|---------|
| TaskName | Task name | Input  | Task name  | 64 bytes max.<br>(63 single-byte alphanumeric characters plus the final NULL character) | ---  | "       |
| Out      | Judgement | Output | TRUE: Task is in execution or on standby.<br>FALSE: Not active | Depends on data type.   | ---  | ---     |

|          | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real numbers |       | Times, durations, 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 specify *TaskName* with a variable containing a text string. Directly specify a text string.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE when an error occurs on the preceding rung.
- An error will occur in the following case. The value of *Out* will not change.
  - The task specified with *TaskName* does not exist.

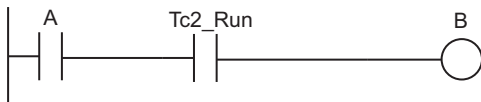
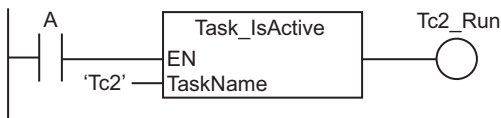


## Sample Programming

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 |

```

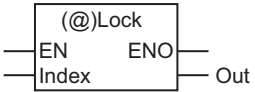
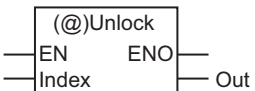
IF (A=TRUE) THEN
    // Determine task status.
    Tc2_Run:=Task_IsActive('Tc2');
    // Make variable B TRUE if Tc2 is running.
    IF (Tc2_Run=TRUE) THEN
        B := TRUE;
    END_IF;
END_IF;

```

# 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/<br>FUN | Graphic expression  | ST expression  |
|-------------|--------------|------------|---|----------------|
| Lock        | Lock Tasks   | FUN        |  | Lock(Index);   |
| Unlock      | Unlock Tasks | FUN        |  | Unlock(Index); |

## Variables

|       | 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             | ---  | ---     |

|       | Boo<br>lean | Bit strings |      |       |       |       | Integers |       |       |      |     |      |      | Real num-<br>bers |       | Times, durations,<br>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          |             |      |       |       |       |          |       |       |      |     |      |      |                   |       |  |      |     |    |        |

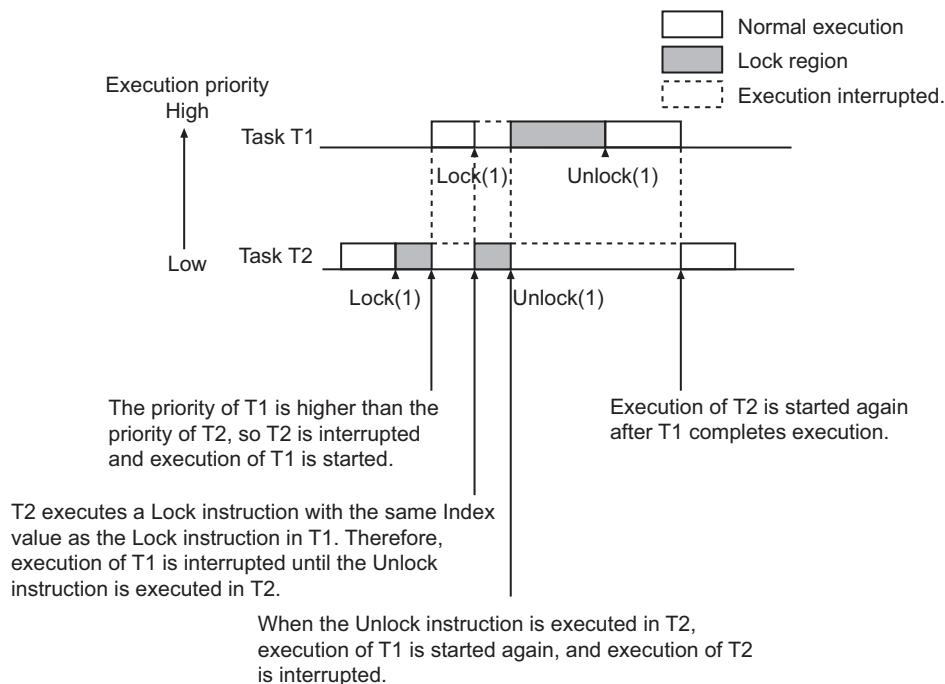
## Function

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.

Task T1 and task T2 each have 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.



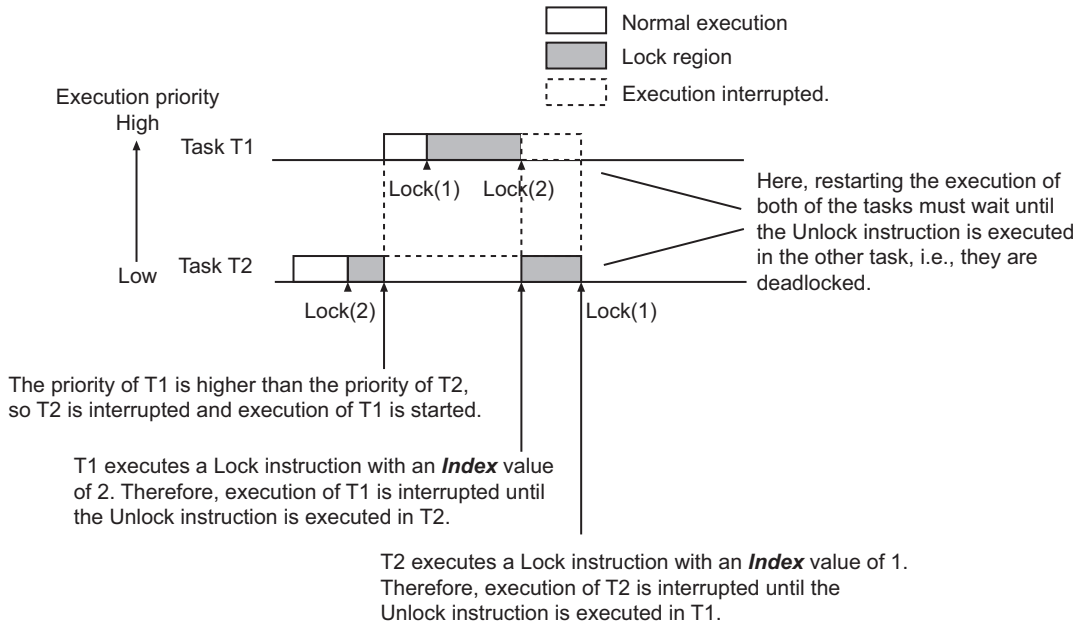
Lock regions with different values for *Index* do not affect each other.

## Additional Information

- The Lock and Unlock instructions are used when the same data is read or written from more than one task. They are used to prevent other tasks from reading or writing the data while a certain task is reading or 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. The following shows an example where a deadlock occurs.



- An error will occur in the following case. The value of *Out* will not change.
  - a) An attempt is made to set up more than 16,777,215 lock regions at the same time.

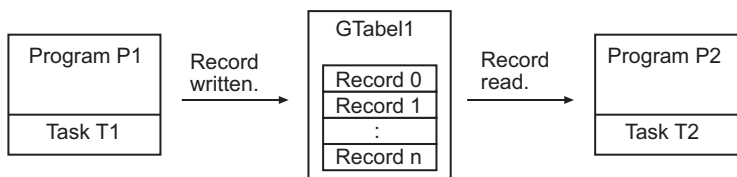
## Sample Programming

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

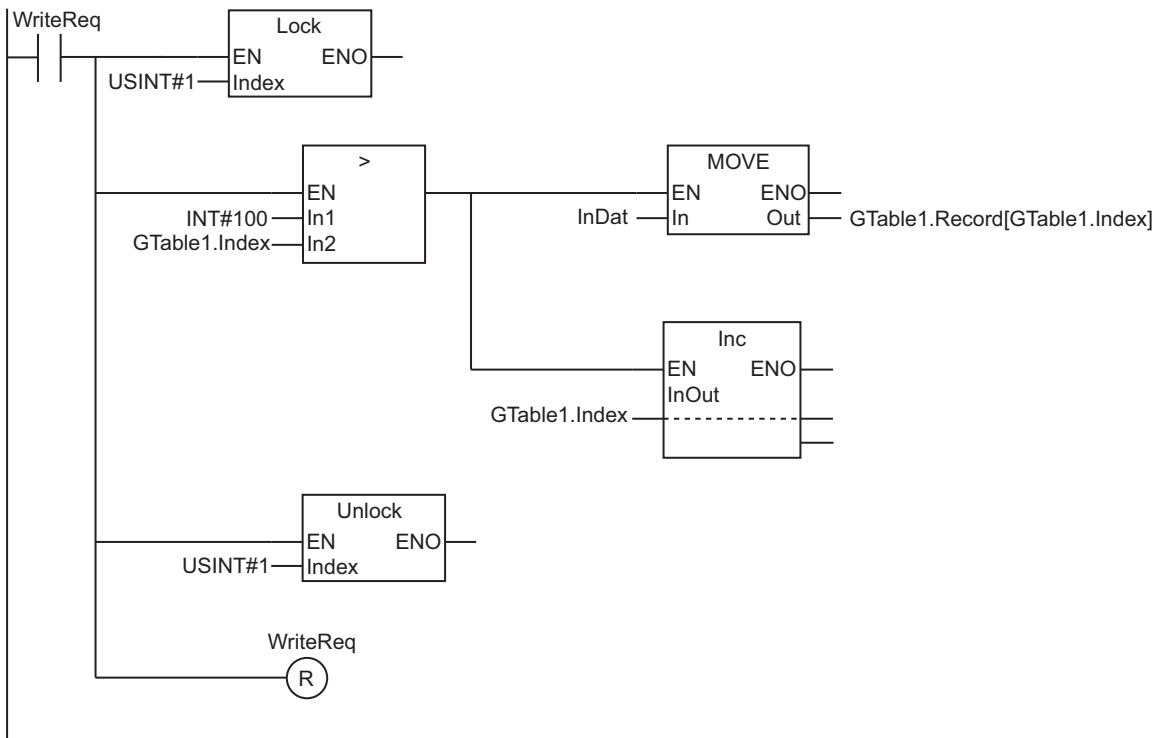
| Variable | Data type | Initial value                 | Comment                  |
|----------|-----------|-------------------------------|--------------------------|
| GTable1  | USERTABLE | (Index:=0,Record:=[100(0.0)]) | Record storage structure |

## 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;

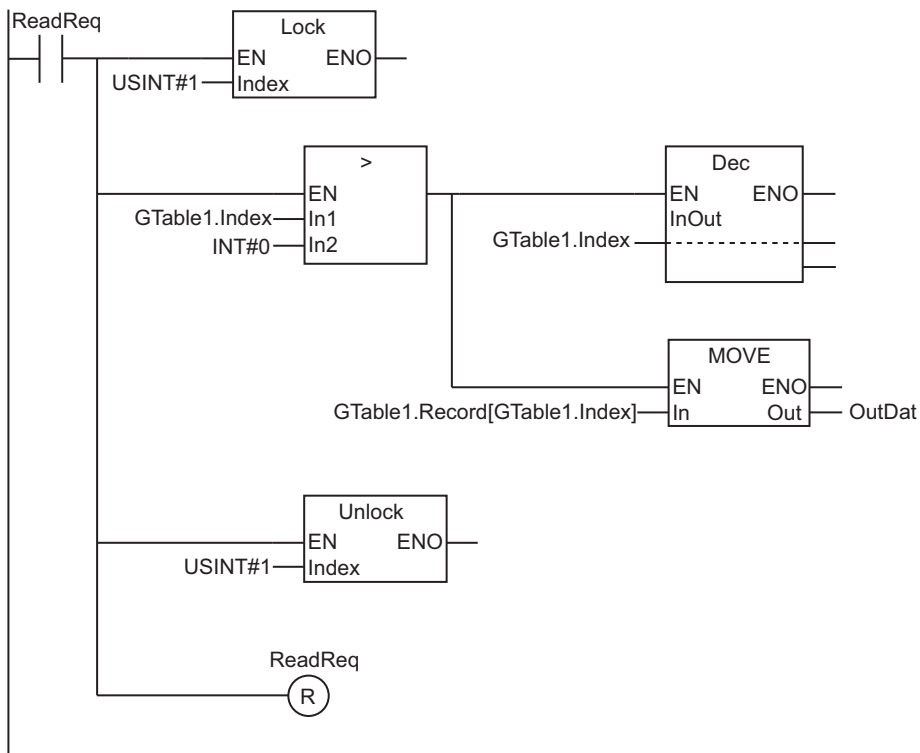
```

### 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/<br>FUN | Graphic expression | ST expression           |
|--------------|---------------------|------------|--------------------|-------------------------|
| ActEventTask | Activate Event Task | FUN        |                    | ActEventTask(TaskName); |



## 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.

## Variables

|          | 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.<br>FALSE: The instruction was not executed or an error occurred. | Depends on data type.  | ---  | ---     |

|          | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      |      | Real<br>num-<br>bers |       | Times, durations,<br>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

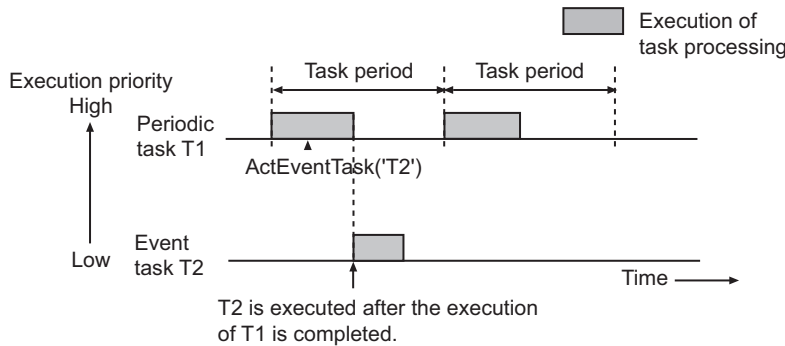
The ActEventTask instruction activates the event task with task name *TaskName*.

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.



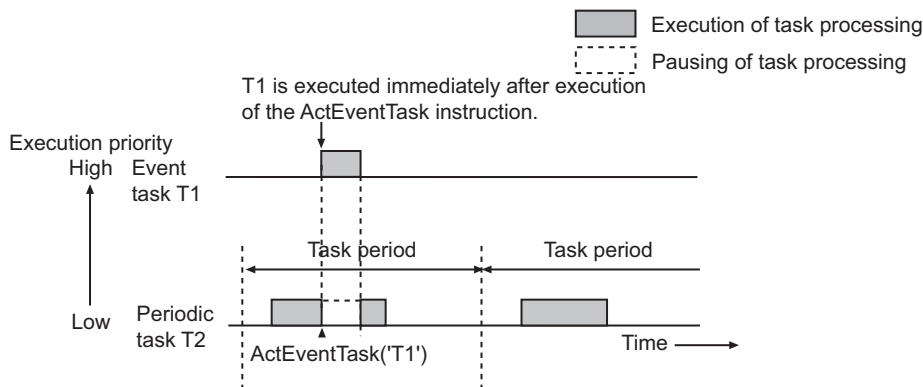
For example, assume that the execution priority of event task T2 is lower than the execution priority of periodic task T1.  
 If the ActEventTask instruction is executed for T2 in T1, the execution of T1 is completed before T2 is executed.



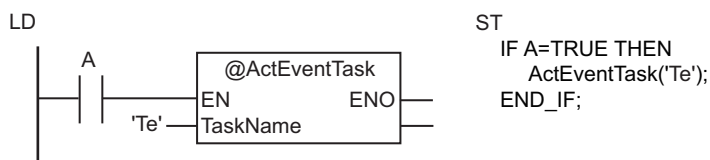
If an event task is started that has an execution priority that is higher than the execution priority of the task in which this instruction was executed, the execution of the task in which this instruction was executed is paused and the event task is executed.

For example, assume that the execution priority of periodic task T2 is lower than the execution priority of event task T1.

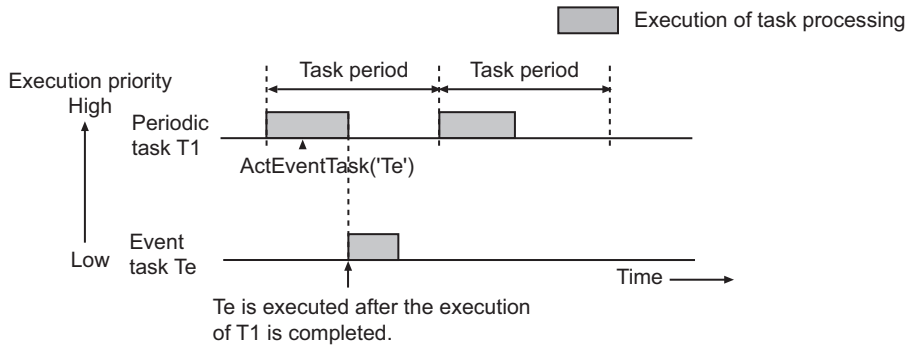
If the ActEventTask instruction is executed for T1 in T2, the execution of T2 is paused to execute T1.



The following figure shows a programming example. When the value of variable A is TRUE, event task 'Te' is executed.



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   |
|----------------------------|------------------|-----------|---|
| <code>_**_Active</code> *1 | Task Active Flag | BOOL      | This variable indicates the execution status of the task. *2<br>TRUE: Execution processing is in progress.<br>FALSE: Stopped. |

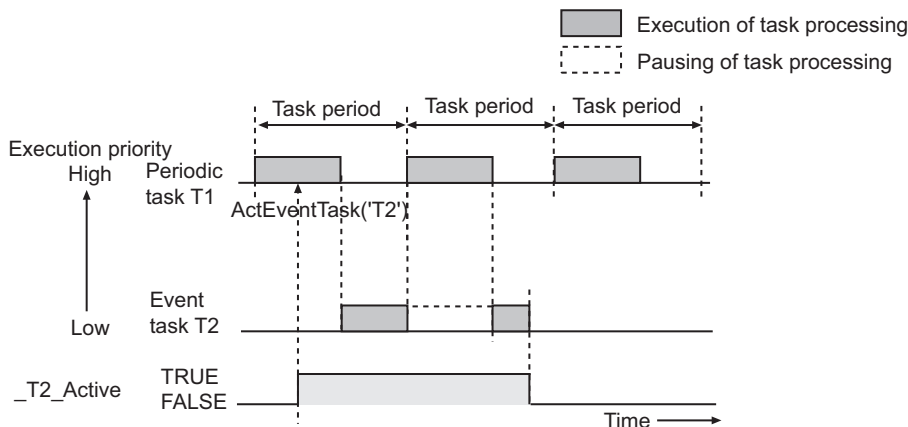
\*1. The asterisks (\*\*) are replaced with the task name.

\*2. Refer to the *NJ/NX-series CPU Unit Software User's Manual (Cat. No. W501)* for details.

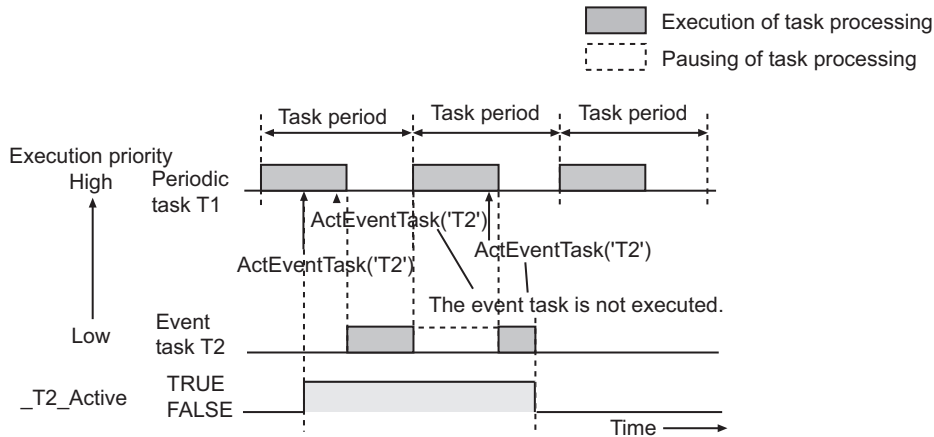
## Additional Information

### 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 activated even if this instruction is executed while the system-defined variable `_**_Active` for the event task is TRUE.

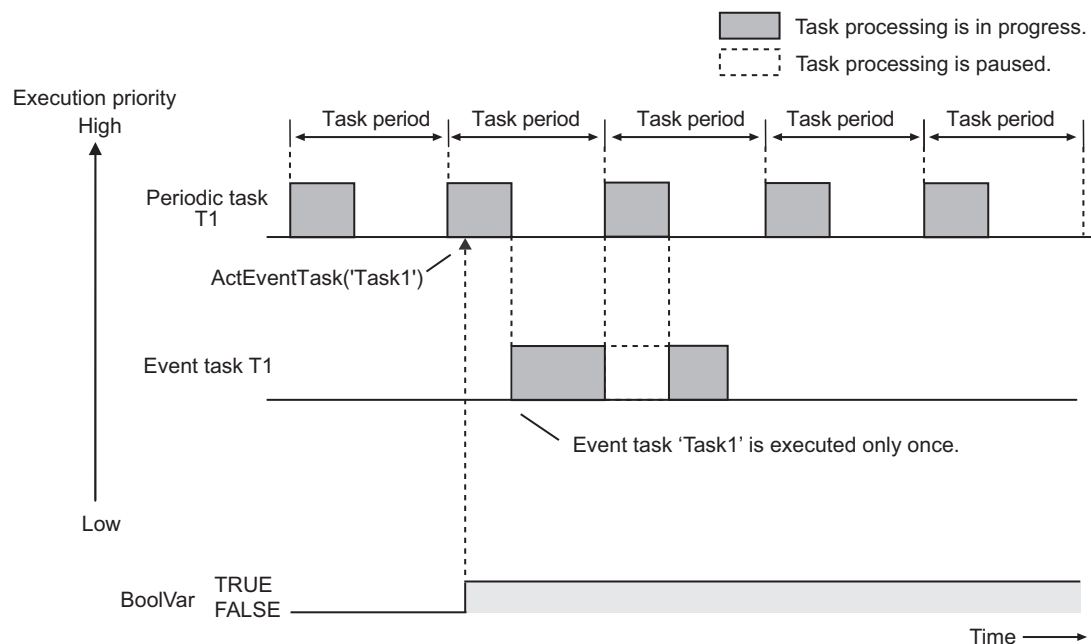
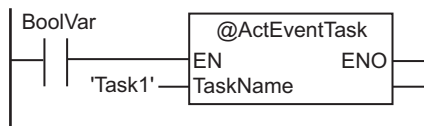


## Executing an Event Task Only Once and Executing It Repeatedly

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.

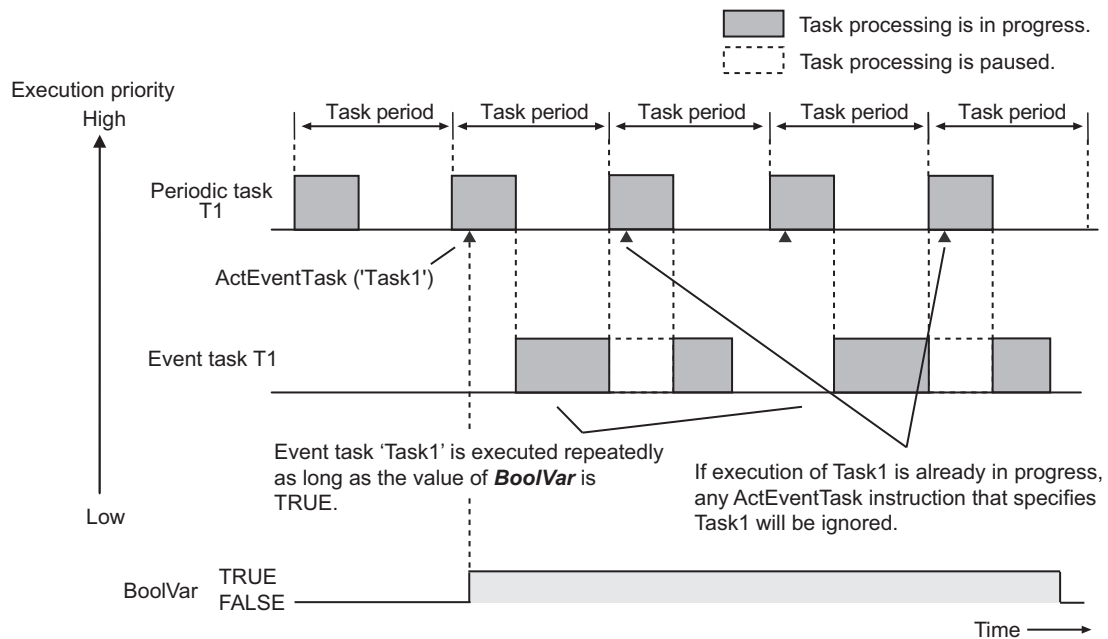
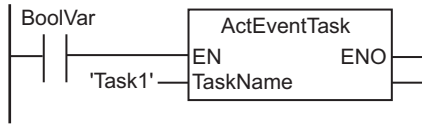
### ● Executing an Event Task Only Once When the Value of a Specified Variable Changes

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.



## ● Executing an Event Task Repeatedly for a Period of Time with a Variable at 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, execution time is required even if the event task is not activated.
- An error will occur if the event task that is specified with *TaskName* does not exist. *ENO* will be FALSE.

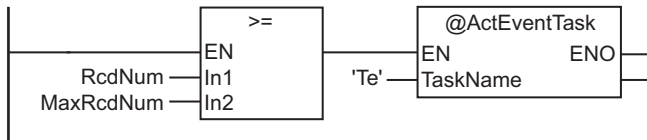
## Sample Programming

### 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;

```

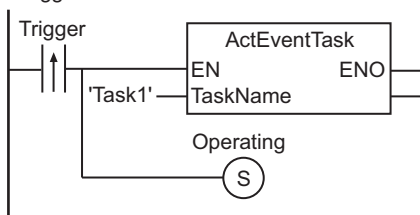
## Example of Confirming Completion of Event Task before Proceeding

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.

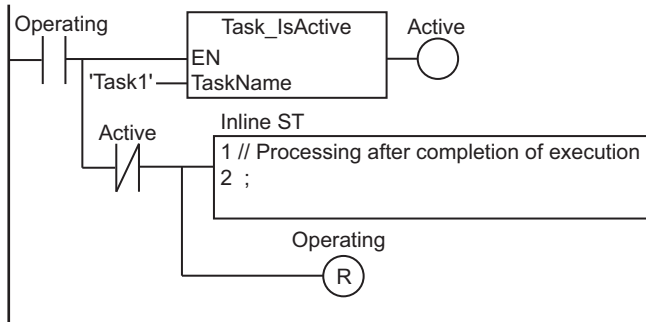
## ● LD

| 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          |

Trigger is received and ActEventTask is executed.



Task\_IsActive is used to see if Task1 execution is in progress.



### ● ST

| Name        | Data type | Initial value | Comment   |
|-------------|-----------|---------------|---|
| Trigger     | BOOL      | FALSE         | Execution condition                               |
| LastTrigger | BOOL      | FALSE         | Value of <i>Trigger</i> 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/<br>FUN | Graphic expression   | ST expression   |
|-------------|-----------------------|------------|--|---|
| Get**Clk    | Get Clock Pulse Group | FUN        |  <p>*** must be 100 <math>\mu</math>s, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.</p> | Out:=Get**Clk();<br>*** must be 100 $\mu$ s, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min. |

## Variables

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

|     | Boo<br>lean | Bit strings |      |       |       | Integers |      |       |       |      |     |      | Real numbers |      | Times, durations, 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          |             |      |       |       |          |      |       |       |      |     |      |              |      |   |      |      |     |    |        |

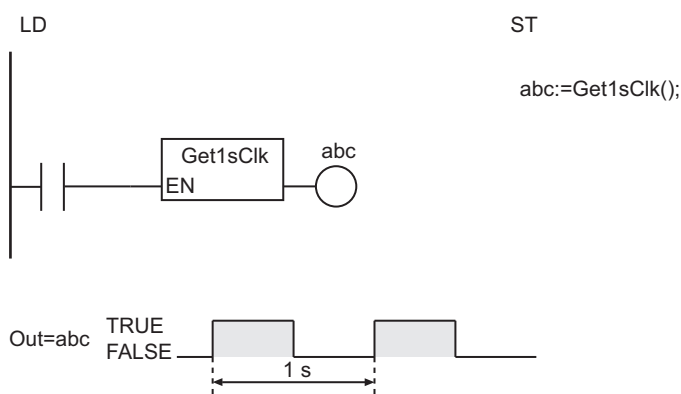
## Function

The Get\*\*Clk instruction outputs a clock pulse at the specified cycle.

The clock pulse period is 100  $\mu$ s, 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.



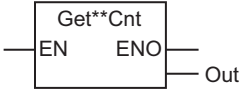
### Precautions for Correct Use

- The first value of *Out* after execution is not defined.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE when an error occurs on the preceding rung.



# Get\*\*Cnt

The Get\*\*Cnt instruction gets free-running counter values at the specified cycle.

| Instruction | Name  | FB/<br>FUN | Graphic expression  | ST expression  |
|-------------|---|------------|---|--|
| Get**Cnt    | Get Incrementing Free-running Counter Group | FUN        |  | Out:=Get**Cnt();<br><br>"***" must be 100 ns, 1 µs, 1 ms, 10 ms, 100 ms, or 1 s. |

## Variables

|     | 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    |      |     |      |          |      |       |      |      |              |    |   |  |  |  |  |

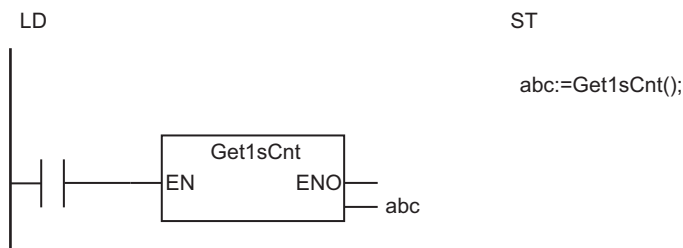
## Function

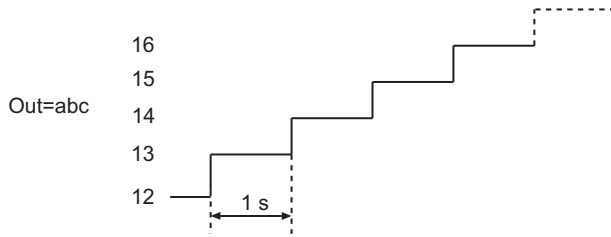
The Get\*\*Cnt instruction gets the values of free-running counters of the specified cycle.

A free-running counter is a counter that is incremented at specified intervals. *Out* contains the current value of the count. The counter period can be set to 100 ns, 1 µs, 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.





### 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 start value of `Out` is not defined. It does not necessarily start from 0.



# Appendices

A

---

|            |  |             |
|------------|--|-------------|
| <b>A-1</b> | <b>Error Codes That You Can Check with ErrorID .....</b>                                       | <b>A-2</b>  |
| <b>A-2</b> | <b>Error Codes .....</b>   | <b>A-31</b> |
| <b>A-3</b> | <b>Instructions You Cannot Use in Event Tasks .....</b>  | <b>A-32</b> |
| <b>A-4</b> | <b>Instructions Related to NX Message Communications Errors .....</b>                          | <b>A-35</b> |
| <b>A-5</b> | <b>SDO Abort Codes .....</b>   | <b>A-36</b> |
| <b>A-6</b> | <b>Version Information .....</b>   | <b>A-37</b> |
| A-6-1      | Instructions with Specifications Changes and New Instructions for<br>Version Upgrades .....    | A-37        |
| A-6-2      | What to Do If An Error Message Says the Instruction May Cause Un-<br>intended Operations ..... | A-41        |

# 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 *NJ/NX-series Troubleshooting Manual (Cat. No. W503)* 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 | ---                             |
| ResetCJBError               |                             | Reset I/O Bus Error                 | 16#0400                    | Input Value Out of Range        |
|                             |                             |                                     | 16#040D                    | Illegal Unit Specified          |
| ResetMCError                |                             | Reset Motion Control Error          | ---                        | ---                             |
| ResetECEError               |                             | Reset EtherCAT Error                | 16#041A                    | Multi-execution of Instructions |
| GetNXUnitError              |                             | Get NX Unit Error Status            | 16#0400                    | Input Value Out of Range        |
|                             |                             |                                     | 16#041A                    | Multi-execution of Instructions |
|                             |                             |                                     | 16#2C00                    | NX Message Error                |
|                             |                             |                                     | 16#2C02                    | NX Message Timeout              |
| ResetUnit                   |                             | Restart Unit                        | 16#0400                    | Input Value Out of Range        |
|                             |                             |                                     | 16#040D                    | Illegal Unit Specified          |
|                             |                             |                                     | 16#040F                    | Unit Restart Failed             |

| Type | Instruction             | Name                             | Error code | Error name   |
|------|-------------------------|----------------------------------|------------|--|
|      | 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 |
|      | 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              |
|      | 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                             |
|      | 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                       |

| Type  | Instruction    | Name                          | Error code                                       | Error name                                       |
|---|----------------|-------------------------------|--|--|
| EtherCAT<br>Communica-<br>tions Instruc-<br>tions | EC_CoESDOWrite | Write EtherCAT CoE SDO        | 16#0400  | Input Value Out of Range                         |
|   |                |                               | 16#1800  | EtherCAT Commu-<br>nications Error               |
|   |                |                               | 16#1801  | EtherCAT Slave<br>Does Not Respond               |
|   |                |                               | 16#1802  | EtherCAT Timeout                                 |
|   |                |                               | 16#1804  | SDO Abort Error                                  |
|   |                |                               | 16#1808  | Communications<br>Resource Overflow              |
|   | EC_CoESDORead  | Read EtherCAT CoE SDO         | 16#0400  | Input Value Out of Range                         |
|   |                |                               | 16#1800  | EtherCAT Commu-<br>nications Error               |
|   |                |                               | 16#1801  | EtherCAT Slave<br>Does Not Respond               |
|   |                |                               | 16#1802  | EtherCAT Timeout                                 |
|   |                |                               | 16#1803  | Reception Buffer<br>Overflow                     |
|   |                |                               | 16#1804  | SDO Abort Error                                  |
|   | EC_StartMon    | Start EtherCAT Packet Monitor | 16#1805  | Saving Packet Mon-<br>itor File                  |
|   |                |                               | 16#1807  | Packet Monitoring<br>Function in Opera-<br>tion  |
|   |                |                               | 16#1808  | Communications<br>Resource Overflow              |
|   |                |                               | 16#1809  | Packet Monitoring<br>Function Not Sup-<br>ported |
|   | EC_StopMon     | Stop EtherCAT Packet Monitor  | 16#1806  | Packet Monitoring<br>Function Not Start-<br>ed   |
|   |                |                               | 16#1808  | Communications<br>Resource Overflow              |
|   |                |                               | 16#1809  | Packet Monitoring<br>Function Not Sup-<br>ported |
|   | EC_SaveMon     | Save EtherCAT Packets         | 16#1805  | Saving Packet Mon-<br>itor File                  |
|   |                |                               | 16#1807  | Packet Monitoring<br>Function in Opera-<br>tion  |
| 16#1808   |                |                               | Communications<br>Resource Overflow              |  |
| 16#1809   |                |                               | Packet Monitoring<br>Function Not Sup-<br>ported |  |



| Type               | Instruction               | Name | Error code | Error name                                     |
|--------------------|---------------------------|------|------------|--|
| 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               |
|                    |                           |      | 16#1809    | Packet Monitoring Function Not Supported       |
| EC_DisconnectSlave | Disconnect EtherCAT Slave |      | 16#1800    | EtherCAT Communications Error                  |
|                    |                           |      | 16#1801    | EtherCAT Slave Does Not Respond                |
|                    |                           |      | 16#1808    | Communications Resource Overflow               |
|                    |                           |      | 16#180A    | Unable to Execute Instruction for Target Slave |
| EC_ConnectSlave    | Connect EtherCAT Slave    |      | 16#1800    | EtherCAT Communications Error                  |
|                    |                           |      | 16#1801    | EtherCAT Slave Does Not Respond                |
|                    |                           |      | 16#1808    | Communications Resource Overflow               |
|                    |                           |      | 16#180A    | Unable to Execute Instruction for Target Slave |

| Type | Instruction            | Name                          | Error code | Error name                                     |
|------|------------------------|-------------------------------|------------|--|
|      | EC_ChangeEnableSetting | Enable/Disable EtherCAT Slave | 16#1800    | EtherCAT Communications Error                  |
|      |                        |                               | 16#1801    | EtherCAT Slave Does Not Respond                |
|      |                        |                               | 16#1808    | Communications Resource Overflow               |
|      |                        |                               | 16#180A    | Unable to Execute Instruction for Target Slave |
|      | 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   |                             |  |  |

| Type   | Instruction                    | Name  | Error code                                      | Error name                                   |
|--|--------------------------------|---|---|--|
| EtherNet/IP<br>Communica-<br>tions Instruc-<br>tions | CIPOpen                        | Open CIP Class 3 Connection<br>(Large_Forward_Open)     | 16#0400   | Input Value Out of Range                     |
|  |                                |   | 16#1C00   | Explicit Message Er-<br>ror                  |
|  |                                |   | 16#1C01   | Incorrect Route<br>Path                      |
|  |                                |   | 16#1C03   | CIP Communica-<br>tions Resource<br>Overflow |
|  |                                |   | 16#1C04   | CIP Timeout                                  |
|  |                                |   | 16#1C05   | Class-3 Connection<br>Not Established        |
|  |                                |   | 16#2000   | Local IP Address<br>Setting Error            |
|  |                                |   | 16#2004   | Local IP Address<br>Not Set                  |
|  | CIPOpenWithDataSize            | Open CIP Class 3 Connection with<br>Specified Data Size | 16#0400   | Input Value Out of Range                     |
|  |                                |   | 16#1C00   | Explicit Message Er-<br>ror                  |
|  |                                |   | 16#1C01   | Incorrect Route<br>Path                      |
|  |                                |   | 16#1C03   | CIP Communica-<br>tions Resource<br>Overflow |
|  |                                |   | 16#1C04   | CIP Timeout                                  |
|  |                                |   | 16#1C05   | Class-3 Connection<br>Not Established        |
| 16#2000  |                                |   | Local IP Address<br>Setting Error               |  |
| 16#2004  |                                |   | Local IP Address<br>Not Set                     |  |
| CIPRead  | Read Variable Class 3 Explicit | 16#0400   | Input Value Out of Range                        |  |
|  |                                | 16#0407   | Data Range Ex-<br>ceeded                        |  |
|  |                                | 16#0419   | Incorrect Data Type                             |  |
|  |                                | 16#1C00   | Explicit Message Er-<br>ror                     |  |
|  |                                | 16#1C02   | CIP Handle Out of<br>Range                      |  |
|  |                                | 16#1C03   | CIP Communica-<br>tions Resource<br>Overflow    |  |
|  |                                | 16#1C04   | CIP Timeout                                     |  |
|  |                                | 16#1C06   | CIP Communica-<br>tions Data Size Ex-<br>ceeded |  |



| Type     | Instruction                     | Name | Error code | Error name                            |
|----------|---------------------------------|------|------------|---------------------------------------|
| 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#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                           |
|-------------|--------------------------------------|------|--------------|--------------------------------------|
| 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 Setting Error       |
|             |                                      |      | 16#2004      | Local IP Address Not Set             |
|             |                                      |      | CIPUCMMWrite | Write Variable UCMM Explicit         |
| 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 Setting Error       |      |              |                                      |
| 16#2004     | Local IP Address Not Set             |      |              |                                      |

| Type | Instruction | Name                       | Error code | Error name                              |
|------|-------------|----------------------------|------------|---|
|      | 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 Setting Error          |
|      |             |                            | 16#2004    | Local IP Address Not Set                |
|      | SkUDPCreate | Create UDP Socket          | 16#0400    | Input Value Out of Range                |
|      |             |                            | 16#2000    | Local IP Address Setting Error          |
|      |             |                            | 16#2001    | TCP/UDP Port Already in Use             |
|      |             |                            | 16#2003    | Socket Status Error                     |
|      |             |                            | 16#2004    | Local IP Address Not Set                |
|      |             |                            | 16#2008    | Socket Communications Resource Overflow |
|      | SkUDPRcv    | 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 |

| Type | Instruction   | Name               | Error code | Error name                              |
|------|---------------|--------------------|------------|---|
|      | SktUDPSend    | UDP Socket Send    | 16#0400    | Input Value Out of Range                |
|      |               |                    | 16#0406    | Illegal Data Position Specified         |
|      |               |                    | 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#2005    | Unable to Use Built-in EtherNet/IP Port |
|      |               |                    | 16#2006    | Socket Timeout                          |
|      |               |                    | 16#2008    | Socket Communications Resource Overflow |



| Type | Instruction     | Name                                | Error code | Error name                              |
|------|-----------------|-------------------------------------|------------|---|
|      | SkdTCPRcv       | 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 |
|      | SkdTCPSend      | TCP Socket Send                     | 16#0400    | Input Value Out of Range                |
|      |                 |                                     | 16#0406    | Illegal Data Position Specified         |
|      |                 |                                     | 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 |
|      | SkdGetTCPStatus | Read TCP Socket Status              | 16#2003    | Socket Status Error                     |
|      |                 |                                     | 16#2007    | Socket Handle Out of Range              |
|      |                 |                                     | 16#2008    | Socket Communications Resource Overflow |
|      | SkdClose        | Close TCP/UDP Socket                | 16#2007    | Socket Handle Out of Range              |
|      |                 |                                     | 16#2008    | Socket Communications Resource Overflow |
|      | SkdClearBuf     | Clear TCP/UDP Socket Receive Buffer | 16#2007    | Socket Handle Out of Range              |
|      |                 |                                     | 16#2008    | Socket Communications Resource Overflow |
|      | SkdSetOption    | 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 |

| Type          | Instruction                     | Name | Error code | Error name                              |
|---------------|---------------------------------|------|------------|---|
| ModbusTCPcmd  | Send Modbus TCP General Command |      | 16#0400    | Input Value Out of Range                |
|               |                                 |      | 16#0406    | Illegal Data Position Specified         |
|               |                                 |      | 16#0407    | Data Range Exceeded                     |
|               |                                 |      | 16#0C10    | Exceptional Modbus Response             |
|               |                                 |      | 16#0C11    | Invalid Modbus Response                 |
|               |                                 |      | 16#2003    | Socket Status Error                     |
|               |                                 |      | 16#2006    | Socket Timeout                          |
|               |                                 |      | 16#2007    | Socket Handle Out of Range              |
| ModbusTCPRead | Send Modbus TCP Read Command    |      | 16#0400    | Input Value Out of Range                |
|               |                                 |      | 16#0406    | Illegal Data Position Specified         |
|               |                                 |      | 16#0407    | Data Range Exceeded                     |
|               |                                 |      | 16#0419    | Incorrect Data Type                     |
|               |                                 |      | 16#0C10    | Exceptional Modbus Response             |
|               |                                 |      | 16#0C11    | Invalid Modbus Response                 |
|               |                                 |      | 16#2003    | Socket Status Error                     |
|               |                                 |      | 16#2006    | Socket Timeout                          |
|               |                                 |      | 16#2007    | Socket Handle Out of Range              |
|               |                                 |      | 16#2008    | Socket Communications Resource Overflow |



| Type | Instruction        | Name                          | Error code | Error name                               |
|------|--------------------|-------------------------------|------------|--|
|      | ModbusTCPWrite     | Send Modbus TCP Write Command | 16#0400    | Input Value Out of Range                 |
|      |                    |                               | 16#0406    | Illegal Data Position Specified          |
|      |                    |                               | 16#0419    | Incorrect Data Type                      |
|      |                    |                               | 16#0C10    | Exceptional Modbus Response              |
|      |                    |                               | 16#0C11    | Invalid Modbus Response                  |
|      |                    |                               | 16#2003    | Socket Status Error                      |
|      |                    |                               | 16#2006    | Socket Timeout                           |
|      |                    |                               | 16#2007    | Socket Handle Out of Range               |
|      | 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                       |
|      | ChangeNTPServerAdr | Change NTP Server Address     | 16#0400    | Input Value Out of Range                 |
|      |                    |                               | 16#040D    | Illegal Unit Specified                   |
|      |                    |                               | 16#2400    | No Execution Right                       |
|      | 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 |

| Type       | Instruction              | Name | Error code | Error name                               |
|------------|--------------------------|------|------------|--|
| 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                               |                                 |
|---------|------------------------------------|-----------------------------|----------------|--|---------------------------------|
|         | 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 | ExecPMCR                    | Protocol Macro | 16#0400                                  | Input Value Out of Range        |
|         |                                    |                             |                | 16#0406                                  | Illegal Data Position Specified |
| 16#0407 |                                    |                             |                | Data Range Exceeded                      |                                 |
| 16#040D |                                    |                             |                | Illegal Unit Specified                   |                                 |
| 16#0413 |                                    |                             |                | Undefined CJ-series Memory Address       |                                 |
| 16#0419 |                                    |                             |                | Incorrect Data Type                      |                                 |
| 16#0C00 |                                    |                             |                | Illegal Serial Communications Mode       |                                 |
| 16#0800 |                                    |                             |                | FINS Error                               |                                 |
| 16#0801 |                                    |                             |                | FINS Port Already in Use                 |                                 |

| Type | Instruction      | Name  | Error code | Error name                         |
|------|------------------|---|------------|------------------------------------|
|      | SerialSend       | SCU Send Serial                                 | 16#0400    | Input Value Out of Range           |
|      |                  |   | 16#0406    | Illegal Data Position Specified    |
|      |                  |   | 16#040D    | Illegal Unit Specified             |
|      |                  |   | 16#0419    | Incorrect Data Type                |
|      |                  |   | 16#0C00    | Illegal Serial Communications Mode |
|      |                  |   | 16#0800    | FINS Error                         |
|      |                  |   | 16#0801    | FINS Port Already in Use           |
|      | SerialRcv        | SCU Receive Serial                              | 16#0400    | Input Value Out of Range           |
|      |                  |   | 16#0407    | Data Range Exceeded                |
|      |                  |   | 16#040D    | Illegal Unit Specified             |
|      |                  |   | 16#0419    | Incorrect Data Type                |
|      |                  |   | 16#0C00    | Illegal Serial Communications Mode |
|      |                  |   | 16#0800    | FINS Error                         |
|      |                  |   | 16#0801    | FINS Port Already in Use           |
|      | SerialRcvNoClear | SCU Receive Serial without Receive Buffer Clear | 16#0400    | Input Value Out of Range           |
|      |                  |   | 16#0407    | Data Range Exceeded                |
|      |                  |   | 16#040D    | Illegal Unit Specified             |
|      |                  |   | 16#0419    | Incorrect Data Type                |
|      |                  |   | 16#0C00    | Illegal Serial Communications Mode |
|      |                  |   | 16#0800    | FINS Error                         |
|      |                  |   | 16#0801    | FINS Port Already in Use           |
|      | SendCmd          | Send Command                                    | 16#0400    | Input Value Out of Range           |
|      |                  |   | 16#0406    | Illegal Data Position Specified    |
|      |                  |   | 16#0407    | Data Range Exceeded                |
|      |                  |   | 16#0419    | Incorrect Data Type                |
|      |                  |   | 16#0800    | FINS Error                         |
|      |                  |   | 16#0801    | FINS Port Already in Use           |

| Type          | Instruction              | Name | Error code | Error name   |
|---------------|--------------------------|------|------------|--|
| 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    | Exceeded Simultaneous Instruction Executed Resources |
|               |                          |      | 16#0C04    | Multi-execution of Ports                             |
|               |                          |      | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|               |                          |      | 16#0C0D    | CIF 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    | Exceeded Simultaneous Instruction Executed Resources |
|               |                          |      | 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    | CIF Unit Initialized                                 |

| Type | Instruction     | Name                            | Error code | Error name   |
|------|-----------------|---------------------------------|------------|--|
|      | 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    | Exceeded Simultaneous Instruction Executed Resources |
|      |                 |                                 | 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    | CIF Unit Initialized                                 |
|      | 16#0C10         | Exceptional Modbus Response     |            |  |
|      | 16#0C11         | Invalid Modbus Response         |            |  |

| Type             | Instruction                  | Name | Error code | Error name   |
|------------------|------------------------------|------|------------|--|
| 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    | Exceeded Simultaneous Instruction Executed Resources |
|                  |                              |      | 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    | CIF Unit Initialized                                 |
| 16#0C10          | Exceptional Modbus Response  |      |            |  |
| 16#0C11          | Invalid Modbus Response      |      |            |  |



| Type              | Instruction                            | Name | Error code | Error name   |
|-------------------|--|------|------------|--|
| 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    | Exceeded Simultaneous Instruction Executed Resources |
|                   |  |      | 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    | CIF 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    | Exceeded Simultaneous Instruction Executed Resources |
|                   |  |      | 16#0C04    | Multi-execution of Ports                             |
|                   |  |      | 16#0C0B    | Serial Communications Timeout                        |
|                   |  |      | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|                   |  |      | 16#0C0D    | CIF Unit Initialized                                 |



| Type | Instruction         | Name                       | Error code | Error name   |
|------|---------------------|----------------------------|------------|--|
|      | NX_SerialSigRead    | Read Serial Control Signal | 16#0400    | Input Value Out of Range                             |
|      |                     |                            | 16#040D    | Illegal Unit Specified                               |
|      |                     |                            | 16#0419    | Incorrect Data Type                                  |
|      |                     |                            | 16#041D    | Exceeded Simultaneous Instruction Executed Resources |
|      |                     |                            | 16#0C04    | Multi-execution of Ports                             |
|      |                     |                            | 16#0C0B    | Serial Communications Timeout                        |
|      |                     |                            | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|      |                     |                            | 16#0C0D    | CIF Unit Initialized                                 |
|      | NX_SerialStatusRead | Read Serial Port Status    | 16#0400    | Input Value Out of Range                             |
|      |                     |                            | 16#040D    | Illegal Unit Specified                               |
|      |                     |                            | 16#0419    | Incorrect Data Type                                  |
|      |                     |                            | 16#041D    | Exceeded Simultaneous Instruction Executed Resources |
|      |                     |                            | 16#0C04    | Multi-execution of Ports                             |
|      |                     |                            | 16#0C0B    | Serial Communications Timeout                        |
|      |                     |                            | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|      |                     |                            | 16#0C0D    | CIF 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    | Exceeded Simultaneous Instruction Executed Resources |
|      |                     |                            | 16#0C04    | Multi-execution of Ports                             |
|      |                     |                            | 16#0C0B    | Serial Communications Timeout                        |
|      |                     |                            | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|      |                     |                            | 16#0C0D    | CIF Unit Initialized                                 |

| Type | Instruction       | Name                         | Error code | Error name   |
|------|-------------------|------------------------------|------------|--|
|      | 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    | Exceeded Simultaneous Instruction Executed Resources |
|      |                   |                              | 16#0C04    | Multi-execution of Ports                             |
|      |                   |                              | 16#0C0B    | Serial Communications Timeout                        |
|      |                   |                              | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|      |                   |                              | 16#0C0D    | CIF 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    | Exceeded Simultaneous Instruction Executed Resources |
|      |                   |                              | 16#0C04    | Multi-execution of Ports                             |
|      |                   |                              | 16#0C0B    | Serial Communications Timeout                        |
|      |                   |                              | 16#0C0C    | Instruction Executed to Inapplicable Port            |
|      |                   |                              | 16#0C0D    | CIF Unit Initialized                                 |



| Type                        | Instruction  | Name                    | Error code | Error name                           |
|-----------------------------|--------------|-------------------------|------------|--------------------------------------|
| SD Memory Card Instructions | FileWriteVar | Write Variable to File  | 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#1409    | That File Name Already Exists        |
|                             |              |                         | 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         |
|                             | FileReadVar  | Read Variable from File | 16#0400    | Input Value Out of Range             |
|                             |              |                         | 16#1400    | SD Memory Card Access Failure        |
|                             |              |                         | 16#1403    | File Does Not Exist                  |
|                             |              |                         | 16#1405    | File Already in Use                  |
|                             |              |                         | 16#140B    | Too Many Files Open                  |
|                             |              |                         | 16#140D    | File or Directory Name Is Too Long   |
|                             |              |                         | 16#140E    | SD Memory Card Access Failed         |

| Type      | Instruction | Name       | Error code | Error name                         |
|-----------|-------------|------------|------------|------------------------------------|
| FileOpen  | Open File   | Open File  | 16#0400    | Input Value Out of Range           |
|           |             |            | 16#1400    | SD Memory Card Access Failure      |
|           |             |            | 16#1401    | SD Memory Card Write-protected     |
|           |             |            | 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       |
| FileClose | Close File  | Close File | 16#1400    | SD Memory Card Access Failure      |
|           |             |            | 16#1403    | File Does Not Exist                |
|           |             |            | 16#1405    | File Already in Use                |
|           |             |            | 16#140E    | SD Memory Card Access Failed       |
| FileSeek  | Seek File   | Seek File  | 16#0400    | Input Value Out of Range           |
|           |             |            | 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#140E    | SD Memory Card Access Failed       |
| FileRead  | Read File   | Read File  | 16#0406    | Illegal Data Position Specified    |
|           |             |            | 16#0419    | Incorrect Data Type                |
|           |             |            | 16#1400    | SD Memory Card Access Failure      |
|           |             |            | 16#1403    | File Does Not Exist                |
|           |             |            | 16#1405    | File Already in Use                |
|           |             |            | 16#1406    | Open Mode Mismatch                 |
|           |             |            | 16#140E    | SD Memory Card Access Failed       |

| Type      | Instruction     | Name | Error code | Error name                           |
|-----------|-----------------|------|------------|--------------------------------------|
| FileWrite | Write File      |      | 16#0406    | Illegal Data Position Specified      |
|           |                 |      | 16#0419    | Incorrect Data Type                  |
|           |                 |      | 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#1405    | File Already in Use                  |
|           |                 |      | 16#1406    | Open Mode Mis-match                  |
| FileGets  | Get Text String |      | 16#1400    | SD Memory Card Access Failure        |
|           |                 |      | 16#1403    | File Does Not Exist                  |
|           |                 |      | 16#1405    | File Already in Use                  |
|           |                 |      | 16#1406    | Open Mode Mis-match                  |
|           |                 |      | 16#140E    | SD Memory Card Access Failed         |
| FilePuts  | Put Text String |      | 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#1405    | File Already in Use                  |
|           |                 |      | 16#1406    | Open Mode Mis-match                  |
|           |                 |      | 16#140E    | SD Memory Card Access Failed         |



| Type     | Instruction                        | Name      | Error code | Error name                           |
|----------|------------------------------------|-----------|------------|--------------------------------------|
| FileCopy | Copy File                          | Copy File | 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#1409    | That File Name Already Exists        |
|          |                                    |           | 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         |
|          |                                    |           | FileRemove | Delete File                          |
| 16#1400  | SD Memory Card Access Failure      |           |            |                                      |
| 16#1401  | SD Memory Card Write-protected     |           |            |                                      |
| 16#1403  | File Does Not Exist                |           |            |                                      |
| 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       |           |            |                                      |

| Type       | Instruction      | Name | Error code | Error name                           |
|------------|------------------|------|------------|--------------------------------------|
| FileRename | Change File Name |      | 16#0400    | Input Value Out of Range             |
|            |                  |      | 16#1400    | SD Memory Card Access Failure        |
|            |                  |      | 16#1401    | SD Memory Card Write-protected       |
|            |                  |      | 16#1403    | File Does Not Exist                  |
|            |                  |      | 16#1404    | Too Many Files/ Directories          |
|            |                  |      | 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#140D    | File or Directory Name Is Too Long   |
|            |                  |      | 16#140E    | SD Memory Card Access Failed         |
| DirCreate  | Create Directory |      | 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#1404    | Too Many Files/ Directories          |
|            |                  |      | 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#140D    | File or Directory Name Is Too Long   |
|            |                  |      | 16#140E    | SD Memory Card Access Failed         |



| Type | Instruction        | Name                  | Error code | Error name                           |
|------|--------------------|-----------------------|------------|--------------------------------------|
|      | DirRemove          | Delete Directory      | 16#0400    | Input Value Out of Range             |
|      |                    |                       | 16#1400    | SD Memory Card Access Failure        |
|      |                    |                       | 16#1401    | SD Memory Card Write-protected       |
|      |                    |                       | 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#140D    | File or Directory Name Is Too Long   |
|      |                    |                       | 16#140E    | SD Memory Card Access Failed         |
|      | BackupToMemoryCard | SD Memory Card Backup | 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#1404    | Too Many Files/ Directories          |
|      |                    |                       | 16#1409    | That File Name Already Exists        |
|      |                    |                       | 16#140C    | Directory Does Not Exist             |
|      |                    |                       | 16#140E    | SD Memory Card Access Failed         |
|      |                    |                       | 16#140F    | Backup Operation Already in Progress |
|      |                    |                       | 16#1410    | Cannot Execute Backup                |
|      |                    |                       | 16#1411    | Unit/Slave Backup Failed             |



## A-2 Error Codes

---

The lower four digits of the event code give the error code for the instruction.

For descriptions of the error codes, refer to the descriptions of the corresponding event codes. For example, if the error code of the instruction is 16#0400, refer to the description of the event with event code 54010400 hex.

Refer to the *NJ/NX-series Troubleshooting Manual (Cat. No. W503)* for event codes.



### Version Information

---

Event codes for instructions are supported by CPU Units with unit version 1.02 or later.

---

# A-3 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                         | page 2-551  |
| Analog Control Instructions          | PIDAT                   | PID Control with Autotuning         | page 2-712  |
|                                      | PIDAT_HeatCool          | Heating/Cooling PID with Autotuning | page 2-742  |
|                                      | AC_StepProgram          | Step Program                        | page 2-830  |
| System Control Instructions          | ResetPLCError           | Reset PLC Controller Error          | page 2-877  |
|                                      | ResetCJBError           | Reset I/O Bus Error                 | page 2-882  |
|                                      | ResetMCError            | Reset Motion Control Error          | page 2-888  |
|                                      | ResetECError            | Reset EtherCAT Error                | page 2-896  |
|                                      | ResetNXBError           | Reset NX Bus Error                  | page 2-901  |
|                                      | GetNXUnitError          | Get NX Unit Error Status            | page 2-905  |
|                                      | ResetUnit               | Restart Unit                        | page 2-914  |
|                                      | RestartNXUnit           | Restart NX Unit                     | page 2-921  |
|                                      | NX_ChangeWriteMode      | Change to NX Unit Write Mode        | page 2-927  |
|                                      | NX_SaveParam            | Save NX Unit Parameters             | page 2-933  |
|                                      | NX_ReadTotalPowerOnTime | Read NX Unit Total Power ON Time    | page 2-942  |
| EtherCAT Communications Instructions | EC_CoESDOWrite          | Write EtherCAT CoE SDO              | page 2-986  |
|                                      | EC_CoESDORead           | Read EtherCAT CoE SDO               | page 2-989  |
|                                      | EC_StartMon             | Start EtherCAT Packet Monitor       | page 2-995  |
|                                      | EC_StopMon              | Stop EtherCAT Packet Monitor        | page 2-1001 |
|                                      | EC_SaveMon              | Save EtherCAT Packets               | page 2-1003 |
|                                      | EC_CopyMon              | Transfer EtherCAT Packets           | page 2-1005 |
|                                      | EC_DisconnectSlave      | Disconnect EtherCAT Slave           | page 2-1007 |
|                                      | EC_ConnectSlave         | Connect EtherCAT Slave              | page 2-1015 |
|                                      | EC_ChangeEnableSetting  | Enable/Disable EtherCAT Slave       | page 2-1017 |
|                                      | NX_WriteObj             | Write NX Unit Object                | page 2-1037 |
| NX_ReadObj                           | Read NX Unit Object     | page 2-1053                         |             |
| IO-Link Communications Instructions  | IOL_ReadObj             | Read IO-Link Device Object          | page 2-1062 |
|                                      | IOL_WriteObj            | Write IO-Link Device Object         | page 2-1071 |



| Type                                    | Instruction                 | Name   | Page        |
|---|-----------------------------|--|-------------|
| EtherNet/IP Communications Instructions | CIPOpen                     | Open CIP Class 3 Connection (Large_Forward_Open)     | page 2-1083 |
|   | CIPOpenWithDataSize         | Open CIP Class 3 Connection with Specified Data Size | page 2-1093 |
|   | CIPRead                     | Read Variable Class 3 Explicit                       | page 2-1097 |
|   | CIPWrite                    | Write Variable Class 3 Explicit                      | page 2-1103 |
|   | CIPSend                     | Send Explicit Message Class 3                        | page 2-1109 |
|   | CIPClose                    | Close CIP Class 3 Connection                         | page 2-1114 |
|   | CIPUCMMRead                 | Read Variable UCMM Explicit                          | page 2-1117 |
|   | CIPUCMMWrite                | Write Variable UCMM Explicit                         | page 2-1123 |
|   | CIPUCMMSend                 | Send Explicit Message UCMM                           | page 2-1130 |
|   | SkUDPCreate                 | Create UDP Socket                                    | page 2-1141 |
|   | SkUDPRcv                    | UDP Socket Receive                                   | page 2-1149 |
|   | SkUDPSend                   | UDP Socket Send                                      | page 2-1152 |
|   | SkTCPAccept                 | Accept TCP Socket                                    | page 2-1155 |
|   | SkTCPConnect                | Connect TCP Socket                                   | page 2-1158 |
|   | SkTCPRcv                    | TCP Socket Receive                                   | page 2-1167 |
|   | SkTCPSend                   | TCP Socket Send                                      | page 2-1170 |
|   | SkGetTCPStatus              | Read TCP Socket Status                               | page 2-1173 |
|   | SkClose                     | Close TCP/UDP Socket                                 | page 2-1176 |
|   | SkClearBuf                  | Clear TCP/UDP Socket Receive Buffer                  | page 2-1179 |
|   | SkSetOption                 | Set TCP Socket Option                                | page 2-1182 |
|   | ChangeIPAdr                 | Change IP Address                                    | page 2-1211 |
|   | ChangeFTPAccount            | Change FTP Account                                   | page 2-1220 |
|   | ChangeNTPServerAdr          | Change NTP Server Address                            | page 2-1224 |
|   | FTPGetFileList              | Get FTP Server File List                             | page 2-1229 |
|   | FTPGetFile                  | Get File from FTP Server                             | page 2-1243 |
|   | FTPPutFile                  | Put File onto FTP Server                             | page 2-1252 |
|   | FTPRemoveFile               | Delete FTP Server File                               | page 2-1263 |
| FTPRemoveDir                            | Delete FTP Server Directory | page 2-1273  |             |
| Serial Communications Instructions      | ExecPMCR                    | Protocol Macro                                       | page 2-1278 |
|   | SerialSend                  | SCU Send Serial                                      | page 2-1292 |
|   | SerialRcv                   | SCU Receive Serial                                   | page 2-1303 |
|   | SerialRcvNoClear            | SCU Receive Serial without Receive Buffer Clear      | page 2-1303 |
|   | SendCmd                     | Send Command   | page 2-1318 |
|   | NX_SerialSend               | Send No-protocol Data                                | page 2-1330 |
|   | NX_SerialRcv                | Receive No-protocol Data                             | page 2-1343 |
|   | NX_ModbusRtuCmd             | Send Modbus RTU General Command                      | page 2-1358 |
|   | NX_ModbusRtuRead            | Send Modbus RTU Read Command                         | page 2-1369 |
|   | NX_ModbusRtuWrite           | Send Modbus RTU Write Command                        | page 2-1380 |
|   | NX_SerialSigCtl             | Serial Control Signal ON/OFF Switching               | page 2-1391 |
|   | NX_SerialSigRead            | Read Serial Control Signal                           | page 2-1399 |
|   | NX_SerialStatusRead         | Read Serial Port Status                              | page 2-1404 |
| NX_SerialBufClear                       | Clear Buffer                | page 2-1409  |             |

| Type                        | Instruction         | Name   | Page        |
|-----------------------------|---------------------|--|-------------|
|                             | NX_SerialStartMon   | Start Serial Line Monitoring                         | page 2-1419 |
|                             | NX_SerialStopMon    | Stop Serial Line Monitoring                          | page 2-1424 |
| SD Memory Card Instructions | FileWriteVar        | Write Variable to File                               | page 2-1430 |
|                             | FileReadVar         | Read Variable from File                              | page 2-1436 |
|                             | FileOpen            | Open File  | page 2-1441 |
|                             | FileClose           | Close File   | page 2-1445 |
|                             | FileSeek            | Seek File  | page 2-1448 |
|                             | FileRead            | Read File  | page 2-1451 |
|                             | FileWrite           | Write File   | page 2-1459 |
|                             | FileGets            | Get Text String                                      | page 2-1467 |
|                             | FilePuts            | Put Text String                                      | page 2-1475 |
|                             | FileCopy            | Copy File  | page 2-1484 |
|                             | FileRemove          | Delete File  | page 2-1492 |
|                             | FileRename          | Change File Name                                     | page 2-1497 |
|                             | DirCreate           | Create Directory                                     | page 2-1503 |
|                             | DirRemove           | Delete Directory                                     | page 2-1506 |
|                             | BackupToMemoryCard  | SD Memory Card Backup                                | page 2-1509 |
| Time Stamp Instructions     | NX_DOutTimeStamp    | Write Digital Output with Specified Time Stamp       | page 2-1524 |
|                             | NX_AryDOutTimeStamp | Write Digital Output Array with Specified Time Stamp | page 2-1530 |

## A-4 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                  | page 2-921  |
|                                      | NX_ChangeWriteMode      | Change to NX Unit Write Mode     | page 2-927  |
|                                      | NX_SaveParam            | Save NX Unit Parameters          | page 2-933  |
|                                      | NX_ReadTotalPowerOnTime | Read NX Unit Total Power ON Time | page 2-942  |
| EtherCAT Communications Instructions | EC_CoESDOWrite          | Write EtherCAT CoE SDO           | page 2-986  |
|                                      | EC_CoESDORead           | Read EtherCAT CoE SDO            | page 2-989  |
|                                      | EC_StartMon             | Start EtherCAT Packet Monitor    | page 2-995  |
|                                      | EC_StopMon              | Stop EtherCAT Packet Monitor     | page 2-1001 |
|                                      | EC_SaveMon              | Save EtherCAT Packets            | page 2-1003 |
|                                      | EC_CopyMon              | Transfer EtherCAT Packets        | page 2-1005 |
|                                      | EC_DisconnectSlave      | Disconnect EtherCAT Slave        | page 2-1007 |
|                                      | EC_ConnectSlave         | Connect EtherCAT Slave           | page 2-1015 |
|                                      | EC_ChangeEnableSetting  | Enable/Disable EtherCAT Slave    | page 2-1017 |
|                                      | NX_WriteObj             | Write NX Unit Object             | page 2-1037 |
|                                      | NX_ReadObj              | Read NX Unit Object              | page 2-1053 |
| IO-Link Communications Instructions  | IOL_ReadObj             | Read IO-Link Device Object       | page 2-1062 |
|                                      | IOL_WriteObj            | Write IO-Link Device Object      | page 2-1071 |



### Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required for an *NX Message Communications Error* to occur.

## A-5 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 <sup>*1</sup> |
| 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                 |

\*1. 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-6 Version Information

This appendix lists the instructions for which specifications were changed and instructions that were added for different unit versions of the CPU Units and for different versions of the Sysmac Studio. It also describes the actions to take if the following error message is displayed for Sysmac Studio version 1.02.

- The instruction may cause unintended operations.

## A-6-1 Instructions with Specifications Changes and New Instructions for Version Upgrades

The instructions that are supported and their specifications depend on the unit version of the CPU Unit and the version of the Sysmac Studio. These are given in the following table.

If a version is given for both the CPU Unit and Sysmac Studio, both versions are required.

| Type                              | Instruction          | Name                   | New/Changed  | Versions  |               | Page       |
|-----------------------------------|----------------------|------------------------|--------------|-----------|---------------|------------|
|                                   |                      |                        |              | CPU Unit  | Sysmac Studio |            |
| ST Statement Instructions         | FOR                  | Repeat Start           | Changed      | ---       | Ver. 1.08     | page 2-46  |
| Sequence Input Instructions       | R_TRIG               | Up Trigger             | Changed      | Ver. 1.02 | ---           | page 2-48  |
| Sequence Output Instructions      | RS                   | Reset-Priority Keep    | Changed      | ---       | Ver. 1.03     | page 2-56  |
|                                   | SR                   | Set-Priority Keep      | Changed      | ---       | Ver. 1.03     | page 2-59  |
| Comparison Instructions           | EQ (=)               | Equal                  | Changed      | ---       | Ver. 1.02     | page 2-102 |
|                                   | NE (<>)              | Not Equal              | Changed      | ---       | Ver. 1.02     | page 2-105 |
|                                   | LT (<)               | Less Than              | Changed      | ---       | Ver. 1.02     | page 2-108 |
|                                   | LE (<=)              | Less Than Or Equal     | Changed      | ---       | Ver. 1.02     | page 2-108 |
|                                   | GT (>)               | Greater Than           | Changed      | ---       | Ver. 1.02     | page 2-108 |
|                                   | GE (>=)              | Greater Than Or Equal  | Changed      | ---       | Ver. 1.02     | page 2-108 |
|                                   | ZoneCmp              | Zone Comparison        | Changed      | Ver. 1.01 | Ver. 1.02     | page 2-120 |
|                                   | Counter Instructions | CTD                    | Down-counter | Changed   | ---           | Ver. 1.03  |
| CTD_**                            |                      | Down-counter Group     | Changed      | ---       | Ver. 1.03     | page 2-158 |
| CTU                               |                      | Up-counter             | Changed      | ---       | Ver. 1.03     | page 2-161 |
| CTU_**                            |                      | Up-counter Group       | Changed      | ---       | Ver. 1.03     | page 2-164 |
| CTUD                              |                      | Up-down Counter        | Changed      | ---       | Ver. 1.03     | page 2-167 |
| CTUD_**                           |                      | Up-down Counter Group  | Changed      | ---       | Ver. 1.03     | page 2-172 |
| Math Instructions                 | EXPT(**)             | Exponentiation         | Changed      | Ver. 1.16 | Ver. 1.20     | page 2-226 |
| Data Type Conversion Instructions | EnumToNum            | Enumeration-to-Integer | New          | Ver. 1.02 | Ver. 1.03     | page 2-331 |
|                                   | NumToEnum            | Integer-to-Enumeration | New          | Ver. 1.02 | Ver. 1.03     | page 2-333 |
| Selection Instructions            | SEL                  | Binary Selection       | Changed      | Ver. 1.02 | Ver. 1.03     | page 2-352 |
|                                   | MUX                  | Multiplexer            | Changed      | Ver. 1.02 | Ver. 1.03     | page 2-354 |

| Type                                      | Instruction           | Name   | New/<br>Change<br>d | Versions  |                  | Page       |
|---|-----------------------|--|---------------------|-----------|------------------|------------|
|   |                       |  |                     | CPU Unit  | Sysmac<br>Studio |            |
|   | AryMax                | Array Maximum  | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-365 |
|   | AryMin                | Array Minimum  | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-365 |
|   | ArySearch             | Array Search   | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-368 |
| Shift Instruc-<br>tions                   | SHL                   | N-bit Left Shift   | Changed             | Ver. 1.02 | Ver. 1.03        | page 2-414 |
|   | SHR                   | N-bit Right Shift  | Changed             | Ver. 1.02 | Ver. 1.03        | page 2-414 |
|   | ROL                   | Rotate N-bits Left                                       | Changed             | Ver. 1.02 | Ver. 1.03        | page 2-420 |
|   | ROR                   | Rotate N-bits Right                                      | Changed             | Ver. 1.02 | Ver. 1.03        | page 2-420 |
| Conversion In-<br>structions              | UTF8ToSJIS            | UTF-8 to SJIS Character Code Conversion                  | New                 | Ver. 1.01 | Ver. 1.02        | page 2-444 |
|   | SJISToUTF8            | SJIS to UTF-8 Character Code Conversion                  | New                 | Ver. 1.01 | Ver. 1.02        | page 2-446 |
|   | PWLApproxNoLineChk    | Broken Line Approximation without Broken Line Data Check | New                 | Ver. 1.03 | Ver. 1.04        | page 2-448 |
|   | PWLLineChk            | Broken Line Data Check                                   | New                 | Ver. 1.03 | Ver. 1.04        | page 2-454 |
|   | PackWord              | 2-byte Join  | New                 | Ver. 1.12 | Ver. 1.16        | page 2-514 |
|   | PackDword             | 4-byte Join  | New                 | Ver. 1.12 | Ver. 1.16        | page 2-516 |
|   | LOWER_BOUND           | Get First Number of Array                                | New                 | Ver.1.18  | Ver.1.22         | page 2-518 |
|   | UPPER_BOUND           | Get Last Number of Array                                 | New                 | Ver.1.18  | Ver.1.22         | page 2-518 |
| Stack and Ta-<br>ble Instruc-<br>tions    | RecSearch             | Record Search  | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-541 |
|   | RecRangeSearch        | Range Record Search                                      | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-546 |
|   | RecSort               | Record Sort  | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-551 |
|   | RecNum                | Get Number of Records                                    | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-557 |
|   |                       |  |                     | Ver. 1.02 | Ver. 1.03        |            |
|   | RecMax                | Maximum Record Search                                    | Changed             | Ver. 1.01 | Ver. 1.02        | page 2-560 |
| RecMin                                    | Minimum Record Search | Changed  | Ver. 1.01           | Ver. 1.02 | page 2-560       |            |
| Text String In-<br>structions             | AddDelimiter          | Put Text Strings with Delimiters                         | New                 | Ver. 1.02 | Ver. 1.03        | page 2-606 |
|   | SubDelimiter          | Get Text Strings Minus Delimiters                        | New                 | Ver. 1.02 | Ver. 1.03        | page 2-618 |
| Time and Time<br>of Day Instruc-<br>tions | TruncTime             | Truncate Time  | New                 | Ver. 1.01 | Ver. 1.02        | page 2-698 |
|   | TruncDt               | Truncate Date and Time                                   | New                 | Ver. 1.01 | Ver. 1.02        | page 2-702 |
|   | TruncTod              | Truncate Time of Day                                     | New                 | Ver. 1.01 | Ver. 1.02        | page 2-706 |
| Analog Control<br>Instructions            | PIDAT_HeatCool        | Heating/Cooling PID with Autotuning                      | New                 | Ver. 1.08 | Ver. 1.09        | page 2-742 |
|   | TimeProportional-Out  | Time-proportional output                                 | New                 | Ver. 1.02 | Ver. 1.03        | page 2-779 |
|   | LimitAlarm_**         | Upper/Lower Limit Alarm Group                            | New                 | Ver. 1.02 | Ver. 1.03        | page 2-799 |



| Type                                 | Instruction                      | Name  | New/<br>Change<br>d | Versions                |                        | Page        |
|--------------------------------------|----------------------------------|---|---------------------|-------------------------|------------------------|-------------|
|                                      |                                  |   |                     | CPU Unit                | Sysmac Studio          |             |
|                                      | LimitAlarmDv_**                  | Upper/Lower Deviation Alarm Group                       | New                 | Ver. 1.02               | Ver. 1.03              | page 2-804  |
|                                      | LimitAlarmDvStby-Seq_**          | Upper/Lower Deviation Alarm with Standby Sequence Group | New                 | Ver. 1.02               | ---                    | page 2-809  |
|                                      | ScaleTrans                       | Scale Transformation                                    | New                 | Ver. 1.05               | Ver. 1.06              | page 2-827  |
|                                      | AC_StepProgram                   | Step Program  | New                 | Ver. 1.06               | Ver. 1.07              | page 2-830  |
|                                      |                                  |   | Changed             | Ver.1.21/<br>Ver.1.32   | Ver.1.28               |             |
| System Control Instructions          | ResetMCErr                       | Reset Motion Control Error                              | Changed             | Ver. 1.02               | Ver. 1.03              | page 2-888  |
|                                      |                                  |   |                     | Ver. 1.10               | Ver. 1.12              |             |
|                                      | GetECErr                         | Get EtherCAT Error Status                               | Changed             | Ver. 1.02               | Ver. 1.03              | page 2-898  |
|                                      | ResetNXBErr                      | Reset NX Bus Error                                      | New                 | Ver. 1.13               | Ver. 1.17              | page 2-901  |
|                                      | GetNXBErr                        | Get NX Bus Error Status                                 | New                 | Ver. 1.13               | Ver. 1.17              | page 2-903  |
|                                      | GetNXUnitErr                     | Get NX Unit Error Status                                | New                 | Ver. 1.13               | Ver. 1.17              | page 2-905  |
|                                      | RestartNXUnit                    | Restart NX Unit   | New                 | Ver. 1.05               | Ver. 1.06              | page 2-921  |
|                                      |                                  |   |                     | Changed                 | Ver. 1.07              |             |
|                                      | NX_ChangeWriteMode               | Change to NX Unit Write Mode                            | New                 | Ver. 1.05               | Ver. 1.06              | page 2-927  |
|                                      | NX_SaveParam                     | Save NX Unit Parameters                                 | New                 | Ver. 1.05               | Ver. 1.06              | page 2-933  |
| PLC_ReadTotalPowerOnTime             | Read PLC Total Power ON Time     | New   | Ver. 1.13           | Ver. 1.17               | page 2-939             |             |
| NX_ReadTotalPowerOnTime              | Read NX Unit Total Power ON Time | New   | Ver. 1.10           | Ver. 1.12               | page 2-942             |             |
| Program Control Instructions         | PrgStart                         | Enable Program  | New                 | Ver. 1.08               | Ver. 1.09              | page 2-952  |
|                                      | PrgStop                          | Disable Program   | New                 | Ver. 1.08               | Ver. 1.09              | page 2-961  |
|                                      | PrgStatus                        | Read Program Status                                     | New                 | Ver. 1.08               | Ver. 1.09              | page 2-980  |
| EtherCAT Communications Instructions | EC_StartMon                      | Start EtherCAT Packet Monitor                           | Changed             | Ver. 1.10               | Ver.1.12* <sup>1</sup> | page 2-995  |
|                                      |                                  |   |                     | Ver.1.40 * <sup>2</sup> | Ver.1.29               |             |
|                                      | EC_StopMon                       | Stop EtherCAT Packet Monitor                            | Changed             | Ver. 1.10               | Ver.1.12* <sup>1</sup> | page 2-1001 |
|                                      |                                  |   |                     | Ver.1.40 * <sup>2</sup> | Ver.1.29               |             |
|                                      | EC_SaveMon                       | Save EtherCAT Packets                                   | Changed             | Ver. 1.10               | Ver.1.12* <sup>1</sup> | page 2-1003 |
|                                      |                                  |   |                     | Ver.1.40 * <sup>2</sup> | Ver.1.29               |             |
|                                      | EC_CopyMon                       | Transfer EtherCAT Packets                               | Changed             | Ver. 1.10               | Ver.1.12* <sup>1</sup> | page 2-1005 |
|                                      |                                  |   |                     | Ver.1.40 * <sup>2</sup> | Ver.1.29               |             |
| EC_ChangeEnableSetting               | Enable/Disable EtherCAT Slave    | New   | Ver. 1.04           | Ver. 1.05               | page 2-1017            |             |
| NX_WriteObj                          | Write NX Unit Object             | New   | Ver. 1.05           | Ver. 1.06               | page 2-1037            |             |
| NX_ReadObj                           | Read NX Unit Object              | New   | Ver. 1.05           | Ver. 1.06               | page 2-1053            |             |

| Type                                    | Instruction                 | Name   | New/<br>Change<br>d | Versions               |                        | Page        |
|---|-----------------------------|--|---------------------|------------------------|------------------------|-------------|
|   |                             |  |                     | CPU Unit               | Sysmac<br>Studio       |             |
| IO-Link Communications Instructions     | IOL_ReadObj                 | Read IO-Link Device Object                           | New                 | Ver. 1.12              | Ver. 1.16              | page 2-1062 |
|   | IOL_WriteObj                | Write IO-Link Device Object                          | New                 | Ver. 1.12              | Ver. 1.16              | page 2-1071 |
| EtherNet/IP Communications Instructions | CIPOpenWithDataSize         | Open CIP Class 3 Connection with Specified Data Size | New                 | Ver. 1.06              | Ver. 1.07              | page 2-1093 |
|   | CIPSend                     | Send Explicit Message Class 3                        | Changed             | Ver. 1.11              | Ver. 1.15              | page 2-1109 |
|   | CIPUCMMSend                 | Send Explicit Message UCMM                           | Changed             | Ver. 1.11              | Ver. 1.15              | page 2-1130 |
|   | SktUDPCreate                | Create UDP Socket                                    | Changed             | Ver. 1.03              | ---                    | page 2-1141 |
|   |                             |  | Changed             | Ver. 1.10              | Ver. 1.13              |             |
|   | SktTCPAccept                | Accept TCP Socket                                    | Changed             | Ver. 1.03              | ---                    | page 2-1155 |
|   | SktTCPConnect               | Connect TCP Socket                                   | Changed             | Ver. 1.03              | ---                    | page 2-1158 |
|   | SktSetOption                | Set TCP Socket Option                                | New                 | Ver.1.12* <sup>3</sup> | Ver.1.16* <sup>3</sup> | page 2-1182 |
|   | ModbusTCPcmd                | Send Modbus TCP General Command                      | New                 | Ver.1.30               | Ver.1.23               | page 2-1187 |
|   | ModbusTCPRead               | Send Modbus TCP Read Command                         | New                 | Ver.1.30               | Ver.1.23               | page 2-1195 |
|   | ModbusTCPWrite              | Send Modbus TCP Write Command                        | New                 | Ver.1.30               | Ver.1.23               | page 2-1203 |
|   | ChangeIPAdr                 | Change IP Address                                    | New                 | Ver. 1.02              | Ver. 1.03              | page 2-1211 |
|   |                             |  | Changed             | Ver. 1.10              | Ver. 1.13              |             |
|   | ChangeFTPAccount            | Change FTP Account                                   | New                 | Ver. 1.02              | Ver. 1.03              | page 2-1220 |
|   |                             |  | Changed             | Ver. 1.10              | Ver. 1.13              |             |
|   | ChangeNTPServerAdr          | Change NTP Server Address                            | New                 | Ver. 1.02              | Ver. 1.03              | page 2-1224 |
|   |                             |  | Changed             | Ver. 1.10              | Ver. 1.13              |             |
|   | FTPGetFileList              | Get FTP Server File List                             | New                 | Ver. 1.08              | Ver. 1.09              | page 2-1229 |
|   |                             |  | Changed             | Ver. 1.09              | Ver. 1.10              |             |
|   |                             |  | Changed             | Ver. 1.16              | ---                    |             |
|   | FTPGetFile                  | Get File from FTP Server                             | New                 | Ver. 1.08              | Ver. 1.09              | page 2-1243 |
|   |                             |  | Changed             | Ver. 1.09              | Ver. 1.10              |             |
|   |                             |  | Changed             | Ver. 1.16              | ---                    |             |
|   | FTPputFile                  | Put File onto FTP Server                             | New                 | Ver. 1.08              | Ver. 1.09              | page 2-1252 |
| Changed                                 |                             |  | Ver. 1.09           | Ver. 1.10              |                        |             |
| Changed                                 |                             |  | Ver. 1.16           | ---                    |                        |             |
| FTPRemoveFile                           | Delete FTP Server File      | New  | Ver. 1.08           | Ver. 1.09              | page 2-1263            |             |
|   |                             | Changed  | Ver. 1.09           | Ver. 1.10              |                        |             |
|   |                             | Changed  | Ver. 1.16           | ---                    |                        |             |
| FTPRemoveDir                            | Delete FTP Server Directory | New  | Ver. 1.08           | Ver. 1.09              | page 2-1273            |             |
|   |                             | Changed  | Ver. 1.09           | Ver. 1.10              |                        |             |
|   |                             | Changed  | Ver. 1.16           | ---                    |                        |             |

| Type                               | Instruction         | Name   | New/<br>Change<br>d | Versions  |               | Page        |
|------------------------------------|---------------------|--|---------------------|-----------|---------------|-------------|
|                                    |                     |  |                     | CPU Unit  | Sysmac Studio |             |
| Serial Communications Instructions | SerialRcvNoClear*4  | SCU Receive Serial without Receive Buffer Clear      | New                 | Ver. 1.03 | Ver. 1.04     | page 2-1303 |
|                                    | NX_SerialSend       | Send No-protocol Data                                | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1330 |
|                                    | NX_SerialRcv        | Receive No-protocol Data                             | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1343 |
|                                    | NX_ModbusRtuCmd     | Send Modbus RTU General Command                      | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1358 |
|                                    | NX_ModbusRtuRead    | Send Modbus RTU Read Command                         | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1369 |
|                                    | NX_ModbusRtuWrite   | Send Modbus RTU Write Command                        | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1380 |
|                                    | NX_SerialSigCtl     | Serial Control Signal ON/OFF Switching               | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1391 |
|                                    | NX_SerialSigRead    | Read Serial Control Signal                           | New                 | Ver. 1.13 | Ver. 1.17     | page 2-1399 |
|                                    | NX_SerialStatusRead | Read Serial Port Status                              | New                 | Ver. 1.13 | Ver. 1.17     | page 2-1404 |
|                                    | NX_SerialBufClear   | Clear Buffer   | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1409 |
|                                    | NX_SerialStartMon   | Start Serial Line Monitoring                         | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1419 |
|                                    | NX_SerialStopMon    | Stop Serial Line Monitoring                          | New                 | Ver. 1.11 | Ver. 1.15     | page 2-1424 |
| SD Memory Card Instructions        | BackupToMemoryCard  | SD Memory Card Backup                                | New                 | Ver. 1.08 | Ver. 1.09     | page 2-1509 |
| Time Stamp Instructions            | NX_DOutTimeStamp    | Write Digital Output with Specified Time Stamp       | New                 | Ver. 1.06 | Ver. 1.07     | page 2-1524 |
|                                    | NX_AryDOutTimeStamp | Write Digital Output Array with Specified Time Stamp | New                 | Ver. 1.06 | Ver. 1.07     | page 2-1530 |
| Other Instructions                 | GetMyTaskInterval   | Read Current Task Period                             | New                 | Ver. 1.08 | Ver. 1.09     | page 2-1550 |
|                                    | ActEventTask        | Activate Event Task                                  | New                 | Ver. 1.03 | Ver. 1.04     | page 2-1560 |

- \*1. For an NJ101 CPU Unit, Sysmac Studio version 1.13 or higher is required.
- \*2. You cannot use this instruction for project unit version 1.40 or later.
- \*3. 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 this instruction.
- \*4. A CPU Unit with unit version 1.03 or later, Sysmac Studio version 1.04 or higher, and a Serial Communications Unit with unit version 2.1 or later are required to use the SerialRcvNoClear instruction.

## A-6-2 What to Do If An Error Message Says the Instruction May Cause Unintended Operations

The Sysmac Studio may display the following error message:

- The instruction may cause unintended operations. Refer to the Instruction Reference Manual for details.

The message is the results of restrictions in the user program. The user program may need to be corrected.

This appendix describes the conditions under which this error message may be displayed and how to correct the user program



**Version Information**

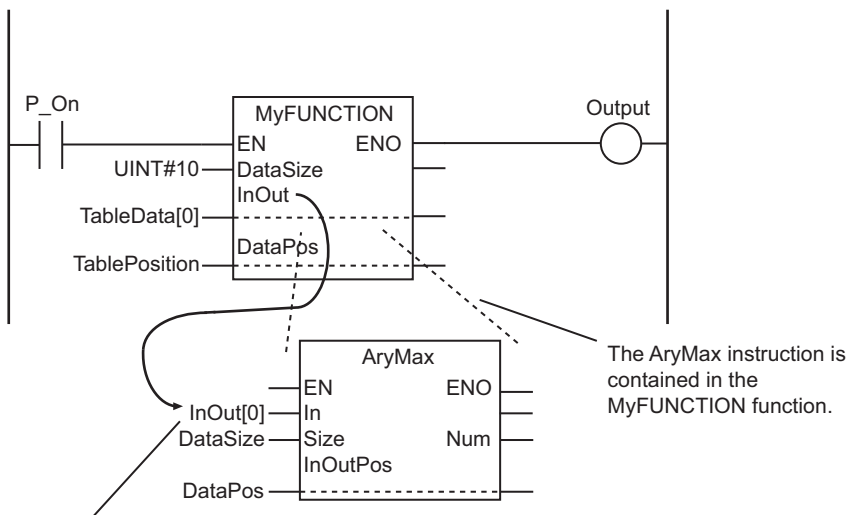
This error message is displayed only for Sysmac Studio version 1.02.

**Conditions for Display of the Error Message**

The error message is displayed when an in-out variable in a function or function block is passed to a specific variable of a specific instruction within the same function or function block.

A table of the instructions and variables for which the error message is displayed is given later in this appendix.

Programming Example for Which the Error Message Is Displayed



In this example, the **InOut** in-out variable of MyFUNCTION is passed to the **In** variable of the AryMax instruction.

**Instructions and Variables for Which the Error Message Is Displayed**

This error message is displayed only for specific variables of specific instructions. These are listed in the following table.

| Type                    | Instruction | Name                       | Function             | Page       |
|-------------------------|-------------|----------------------------|----------------------|------------|
| Comparison Instructions | TableCmp    | Table Comparison           | Table and AryOut     | page 2-122 |
|                         | AryCmpEQ    | Array Comparison Equal     | In1, In2, and AryOut | page 2-125 |
|                         | AryCmpNE    | Array Comparison Not Equal | In1, In2, and AryOut | page 2-125 |
|                         | AryCmpLT    | Array Comparison Less Than | In1, In2, and AryOut | page 2-127 |

| Type                                     | Instruction   | Name   | Function             | Page       |
|--|---------------|--|----------------------|------------|
|  | AryCmpLE      | Array Comparison Less Than Or Equal          | In1, In2, and AryOut | page 2-127 |
|  | AryCmpGT      | Array Comparison Greater Than                | In1, In2, and AryOut | page 2-127 |
|  | AryCmpGE      | Array Comparison Greater Than Or Equal       | In1, In2, and AryOut | page 2-127 |
|  | AryCmpEQV     | Array Value Comparison Equal                 | In1 and AryOut       | page 2-130 |
|  | AryCmpNEV     | Array Value Comparison Not Equal             | In1 and AryOut       | page 2-130 |
|  | AryCmpLTV     | Array Value Comparison Less Than             | In1 and AryOut       | page 2-132 |
|  | AryCmpLEV     | Array Value Comparison Less Than Or Equal    | In1 and AryOut       | page 2-132 |
|  | AryCmpGTV     | Array Value Comparison Greater Than          | In1 and AryOut       | page 2-132 |
| Math In-structions                       | AryCmpGEV     | Array Value Comparison Greater Than Or Equal | In1 and AryOut       | page 2-132 |
|  | AryAdd        | Array Addition                               | In1, In2, and AryOut | page 2-236 |
|  | AryAddV       | Array Value Addition                         | In1 and AryOut       | page 2-238 |
|  | ArySub        | Array Subtraction                            | In1, In2, and AryOut | page 2-240 |
|  | ArySubV       | Array Value Subtraction                      | In1 and AryOut       | page 2-242 |
|  | AryMean       | Array Mean                                   | In                   | page 2-244 |
| BCD Con-<br>version In-<br>structions    | ArySD         | Array Element Standard Devia-<br>tion        | In                   | page 2-246 |
|  | AryToBCD      | Array BCD Conversion                         | In1 and AryOut       | page 2-271 |
| Bit String<br>Processing<br>Instructions | AryToBin      | Array Unsigned Integer Conver-<br>sion       | In1 and AryOut       | page 2-273 |
|  | AryAnd        | Array Logical AND                            | In1, In2, and AryOut | page 2-347 |
|  | AryOr         | Array Logical OR                             | In1, In2, and AryOut | page 2-347 |
|  | AryXor        | Array Logical Exclusive OR                   | In1, In2, and AryOut | page 2-347 |
| Selection<br>Instructions                | AryXorN       | Array Logical Exclusive NOR                  | In1, In2, and AryOut | page 2-347 |
|  | AryMax        | Array Maximum                                | In                   | page 2-365 |
|  | AryMin        | Array Minimum                                | In                   | page 2-365 |
| Data<br>Movement<br>Instructions         | ArySearch     | Array Search                                 | In                   | page 2-368 |
|  | TransBits     | Move Bits                                    | InOut                | page 2-379 |
|  | AryExchange   | Array Data Exchange                          | InOut1 and InOut2    | page 2-387 |
|  | AryMove       | Array Move                                   | In1 and AryOut       | page 2-389 |
| Shift In-<br>structions                  | Clear         | Initialize                                   | InOut                | page 2-391 |
|  | AryShiftReg   | Shift Register                               | InOut                | page 2-406 |
|  | AryShiftRegLR | Reversible Shift Register                    | InOut                | page 2-408 |
|  | ArySHL        | Array N-element Left Shift                   | InOut                | page 2-411 |
|  | ArySHR        | Array N-element Right Shift                  | InOut                | page 2-411 |



| Type                                 | Instruction    | Name                            | Function                  | Page       |
|--------------------------------------|----------------|---------------------------------|---------------------------|------------|
|                                      | NSHLC          | Shift N-bits Left with Carry    | InOut                     | page 2-417 |
|                                      | NSHRC          | Shift N-bits Right with Carry   | InOut                     | page 2-417 |
| Conversion Instructions              | Decoder        | Bit Decoder                     | InOut                     | page 2-429 |
|                                      | Encoder        | Bit Encoder                     | In                        | page 2-432 |
|                                      | ColmToLine_**  | Column to Line Conversion Group | In                        | page 2-435 |
|                                      | LineToColm     | Line to Column Conversion       | InOut                     | page 2-437 |
|                                      | PWLApprox      | Broken Line Approximation       | Line                      | page 2-448 |
|                                      | MovingAverage  | Moving Average                  | Buf                       | page 2-457 |
|                                      | StringToAry    | Text String-to-Array Conversion | AryOut                    | page 2-488 |
|                                      | AryToString    | Array-to-Text String Conversion | In                        | page 2-490 |
|                                      | DispartDigit   | Four-bit Separation             | AryOut                    | page 2-492 |
|                                      | UniteDigit_**  | Four-bit Join Group             | In                        | page 2-494 |
|                                      | Dispart8Bit    | Byte Data Separation            | AryOut                    | page 2-496 |
|                                      | Unite8Bit_**   | Byte Data Join Group            | In                        | page 2-498 |
|                                      | ToAryByte      | Conversion to Byte Array        | In1 and AryOut            | page 2-500 |
|                                      | AryByteTo      | Conversion from Byte Array      | In and OutVal             | page 2-506 |
|                                      | SizeOfAry      | Get Number of Array Elements    | In                        | page 2-512 |
| Stack and Table Instructions         | StackPush      | Push onto Stack                 | InOut                     | page 2-524 |
|                                      | StackFIFO      | First In First Out              | InOut and OutVal          | page 2-533 |
|                                      | StackLIFO      | Last In First Out               | InOut and OutVal          | page 2-533 |
|                                      | StackIns       | Insert into Stack               | InOut                     | page 2-536 |
|                                      | StackDel       | Delete from Stack               | InOut                     | page 2-539 |
|                                      | RecSearch      | Record Search                   | In, Member, and In-OutPos | page 2-541 |
|                                      | RecRangeSearch | Range Record Search             | In, Member, and In-OutPos | page 2-546 |
|                                      | RecSort        | Record Sort                     | InOut and Member          | page 2-551 |
|                                      | RecNum         | Get Number of Records           | In and Member             | page 2-557 |
|                                      | RecMax         | Maximum Record Search           | In, Member, and In-OutPos | page 2-560 |
|                                      | RecMin         | Minimum Record Search           | In, Member, and In-OutPos | page 2-560 |
| FCS Instructions                     | AryLRC_**      | Calculate Array LRC Group       | In                        | page 2-574 |
|                                      | AryCRCCCITT    | Calculate Array CRC-CCITT       | In                        | page 2-576 |
|                                      | AryCRC16       | Calculate Array CRC-16          | In                        | page 2-578 |
| System Control Instructions          | SetAlarm       | Create User-defined Error       | Info1 and Info2           | page 2-868 |
|                                      | SetInfo        | Create User-defined Information | Info1 and Info2           | page 2-912 |
| EtherCAT Communications Instructions | EC_CoESDOWrite | Write EtherCAT CoE SDO          | WriteDat                  | page 2-986 |
|                                      | EC_CoESDORead  | Read EtherCAT CoE SDO           | ReadDat                   | page 2-989 |

| Type                                    | Instruction            | Name                            | Function                      | Page        |
|---|------------------------|---------------------------------|-------------------------------|-------------|
| EtherNet/IP Communications Instructions | CIPRead                | Read Variable Class 3 Explicit  | DstDat                        | page 2-1097 |
|   | CIPWrite               | Write Variable Class 3 Explicit | SrcDat                        | page 2-1103 |
|   | CIPSend                | Send Explicit Message Class 3   | ServiceDat and RespServiceDat | page 2-1109 |
|   | CIPUCMMRead            | Read Variable UCMM Explicit     | DstDat                        | page 2-1117 |
|   | CIPUCMMWrite           | Write Variable UCMM Explicit    | SrcDat                        | page 2-1123 |
|   | CIPUCMMSend            | Send Explicit Message UCMM      | ServiceDat and RespServiceDat | page 2-1130 |
|   | SktUDPRcv              | UDP Socket Receive              | RcvDat                        | page 2-1149 |
|   | SktUDPSend             | UDP Socket Send                 | SendDat                       | page 2-1152 |
|   | SktTCPRcv              | TCP Socket Receive              | RcvDat                        | page 2-1167 |
| SktTCPSend                              | TCP Socket Send        | SendDat                         | page 2-1170                   |             |
| Serial Communications Instructions      | ExecPMCR               | Protocol Macro                  | SrcDat and DstDat             | page 2-1278 |
|   | SerialSend             | SCU Send Serial                 | SrcDat                        | page 2-1292 |
|   | SerialRcv              | SCU Receive Serial              | DstDat                        | page 2-1303 |
|   | SendCmd                | Send Command                    | CmdDat and RespDat            | page 2-1318 |
| SD Memory Card Instructions             | FileWriteVar           | Write Variable to File          | WriteVar                      | page 2-1430 |
|   | FileReadVar            | Read Variable from File         | ReadVar                       | page 2-1436 |
|   | FileRead               | Read File                       | ReadBuf                       | page 2-1451 |
|   | FileWrite              | Write File                      | WriteBuf                      | page 2-1459 |
| Motion Control Instructions             | MC_SetCamTableProperty | Set Cam Table Properties        | CamTable                      | *1          |
|   | MC_SaveCamTable        | Save Cam Table                  | CamTable                      | *1          |
|   | MC_Write               | Write MC Setting                | Target and Setting-Value      | *1          |
|   | MC_CamIn               | Start Cam Operation             | CamTable                      | *1          |
|   | MC_ChangeAxesInGroup   | Change Axes in Group            | Axes                          | *1          |
| Other Instructions                      | ChkRange               | Check Subrange Variable         | Val                           | page 2-1544 |

\*1. Refer to the *NJ/NX-series Motion Control Instructions Reference Manual (Cat. No. W508)* for details.

## Correcting the User Program

The user program must be corrected so that the error message is not displayed. There are two ways to correct the user program.

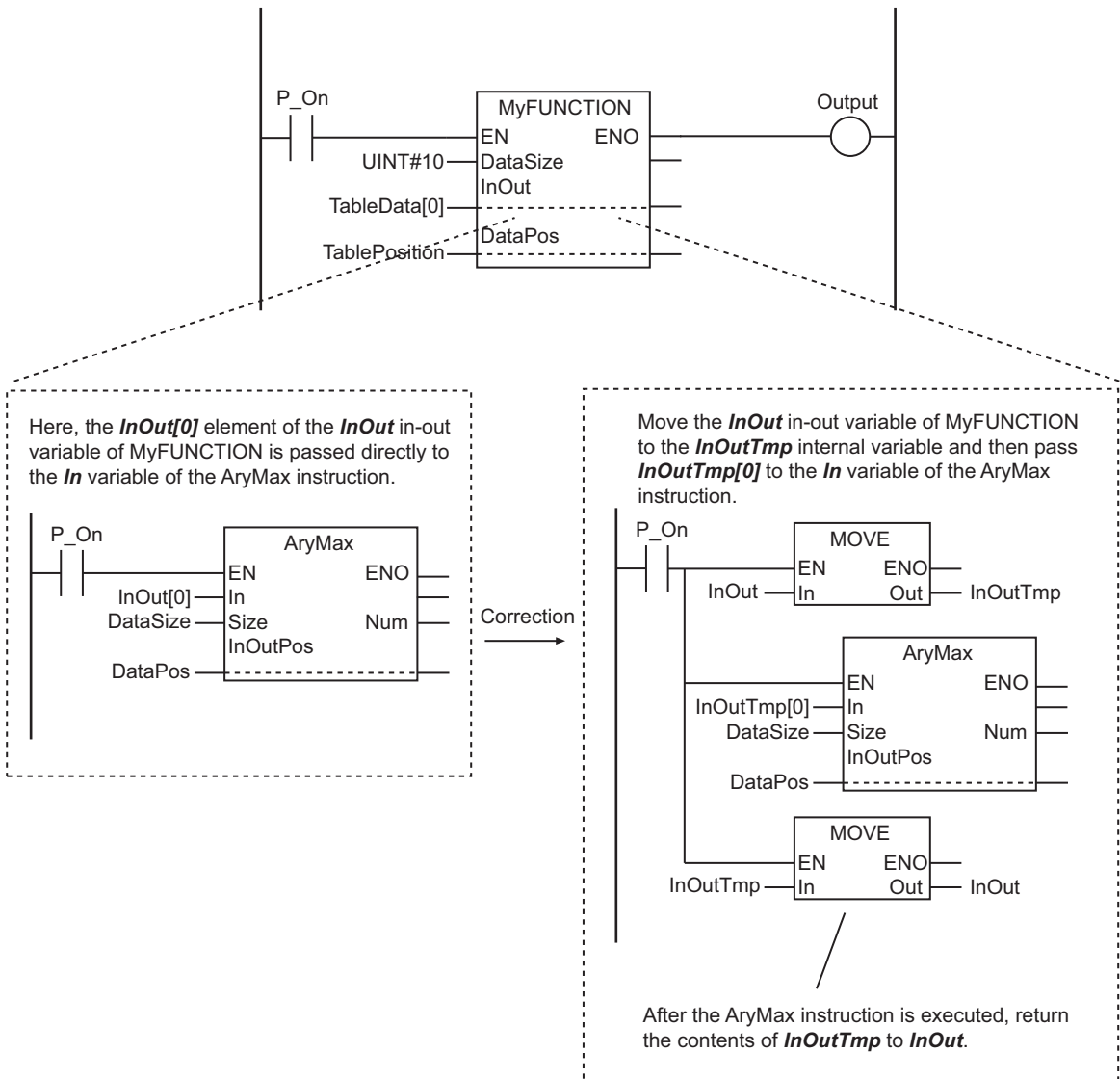
- Copy the in-out variable to an internal variable in the function or function block and then pass the internal variable to the instruction.
- Place the instruction outside of the function or function block.

● **Passing the In-Out Variable to an Internal Variable**

Do not pass the in-out variable of the function or function block directly to the instruction. Copy it to an internal variable first.

If the value of the internal variable changes when the instruction is executed, the internal variable is then copied back to the in-out variable after the instruction is executed.

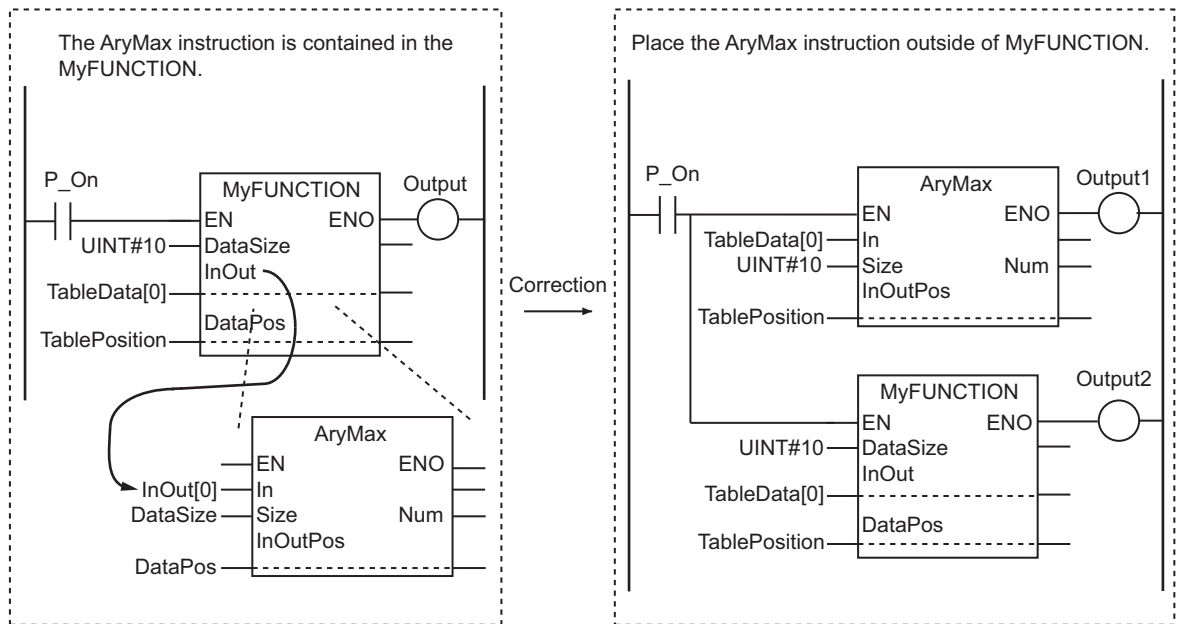
However, this method cannot be used for the Clear instruction. To use the Clear instruction, place it outside the function or function block.



● **Placing the Instruction Outside of the Function or Function Block**

You can correct the problem by placing the instruction outside of the function or function block instead of inside it.





## When the Error Message Can Be Ignored

Even if the error message is displayed, there are cases when the instruction can be used without any problems. Whether the instruction can be used depends on the parameter that is passed to the in-out variable of the function or function block.

These conditions are listed in the following table.

| Applicability of instruction | Parameter passed to in-out variable of function or function block |
|------------------------------|---|
| Can be used.                 | Basic data type, enumeration, array, structure, or union          |
| Cannot be used.              | One element of an array, or one member of a structure or union    |

### ● Example in Which the Instruction Can Be Used

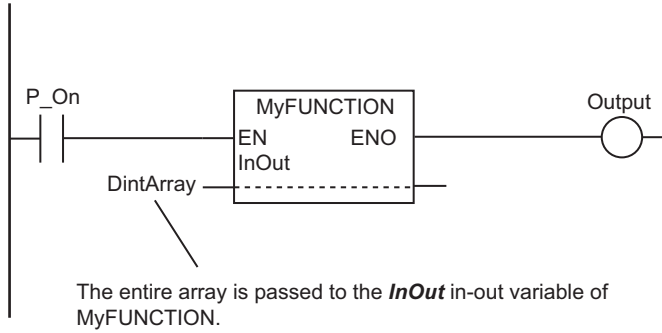
In this example, an array is passed to the in-out variable of a function or function block and then used in an instruction inside the function or function block.

Variable Called by MyFUNCTION

| Variable  | Data type           |
|-----------|---------------------|
| DintArray | ARRAY[0..9] OF DINT |

Variable in MyFUNCTION

| Variable | Data type           |
|----------|---------------------|
| InOut    | ARRAY[0..9] OF DINT |



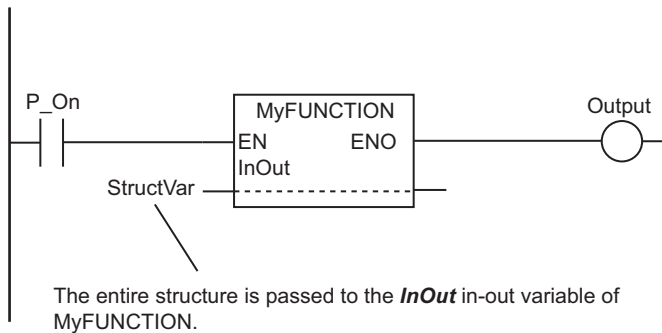
In this example, an structure is passed to the in-out variable of a function or function block and then used in an instruction inside the function or function block.

Variable Called by MyFUNCTION

| Variable  | Data type |
|-----------|-----------|
| StructVar | STRUCT    |

Variable in MyFUNCTION

| Variable | Data type |
|----------|-----------|
| InOut    | STRUCT    |



● **Example in Which the Instruction Cannot Be Used**

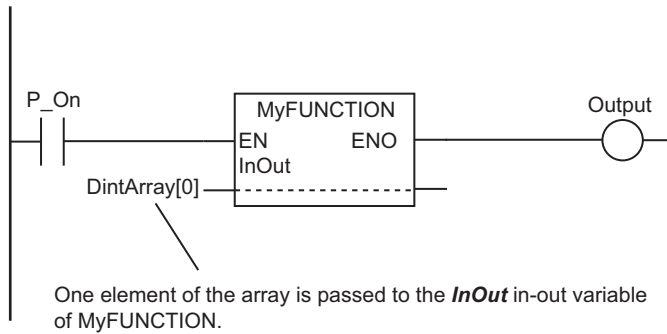
In the following example, one element of an array is passed to the in-out variable of a function or function block, so the instruction cannot be used inside the function or function block.

Variable Called by MyFUNCTION

| Variable  | Data type           |
|-----------|---------------------|
| DintArray | ARRAY[0..9] OF DINT |

Variable in MyFUNCTION

| Variable | Data type |
|----------|-----------|
| InOut    | DINT      |



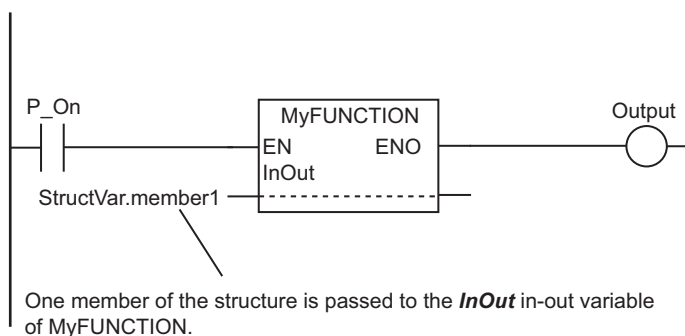
In the following example, one member of a structure is passed to the in-out variable of a function or function block, so the instruction cannot be used inside the function or function block.

Variable Called by MyFUNCTION

| Variable  | Data type |
|-----------|-----------|
| StructVar | STRUCT    |

Variable in MyFUNCTION

| Variable | Data type |
|----------|-----------|
| InOut    | DINT      |



## Option Setting for Error Detection

As described above, there are cases when the instruction can be used even if the error message is displayed. There is an option in the Sysmac Studio that you can use to enable or disable detection of this error.

To prevent detection of this error, display the option settings on the Sysmac Studio and clear the selection of the **Detect an error when an in-out variable is passed to a specific instruction argument** Check Box in the **Program Check Area**. Refer to the *Sysmac Studio Version 1 Operation Manual (Cat. No. W504)* for the specific procedure.

However, before you disable detection of this error, make sure that all of the instructions in the user program can be used. Even if detection of this error is disabled, the same message will be displayed as a warning.

 **Caution**

If you clear the selection of this option, unexpected operation may occur for the instructions and the system may be affected. Always confirm that the conditions for use described in *When the Error Message Can Be Ignored* on page A-47 are met before you clear the selection of this option.

**Version Information**

---

This error message is displayed and the above option setting is available only for Sysmac Studio version 1.02.

---



# Index



# Index

## Numerics

|   |       |
|---|-------|
| - (Subtraction).....  | 2-187 |
| -OU (Subtraction with Overflow Check).....                      | 2-190 |
| * (Multiplication).....   | 2-194 |
| ** (Exponentiation).....  | 2-226 |
| **_BCD_TO_*** (BCD-to-Unsigned Integer Conversion Group).....   | 2-256 |
| **_TO_*** (Bit String-to-Bit String Conversion Group).....      | 2-288 |
| **_TO_*** (Bit String-to-Integer Conversion Group).....         | 2-285 |
| **_TO_*** (Bit String-to-Real Number Conversion Group).....     | 2-290 |
| **_TO_*** (Integer-to-Bit String Conversion Group).....         | 2-280 |
| **_TO_*** (Integer-to-Integer Conversion Group).....            | 2-277 |
| **_TO_*** (Integer-to-Real Number Conversion Group).....        | 2-283 |
| **_TO_*** (Real Number-to-Bit String Conversion Group).....     | 2-295 |
| **_TO_*** (Real Number-to-Integer Conversion Group).....        | 2-292 |
| **_TO_*** (Real Number-to-Real Number Conversion Group).....    | 2-297 |
| **_TO_BCD_*** (Unsigned Integer-to-BCD Conversion Group).....   | 2-259 |
| **_TO_STRING (Bit String-to-Text String Conversion Group).....  | 2-301 |
| **_TO_STRING (Integer-to-Text String Conversion Group).....     | 2-299 |
| **_TO_STRING (Real Number-to-Text String Conversion Group)..... | 2-303 |
| *OU (Multiplication with Overflow Check).....                   | 2-198 |
| / (Division).....   | 2-202 |
| & (Logical AND).....  | 2-340 |
| + (Addition).....   | 2-179 |
| +OU (Addition with Overflow Check).....                         | 2-183 |
| < (Less Than).....  | 2-108 |
| <= (Less Than Or Equal).....                                    | 2-108 |
| <> (Not Equal).....   | 2-105 |
| = (Equal).....  | 2-102 |
| > (Greater Than).....   | 2-108 |
| >= (Greater Than Or Equal).....                                 | 2-108 |
| 2-byte Join.....  | 2-514 |
| 4-byte Join.....  | 2-516 |
| ADD (Addition).....   | 2-179 |
| Add Time.....   | 2-633 |
| Add Time to Date and Time.....                                  | 2-637 |
| Add Time to Time of Day.....                                    | 2-635 |
| ADD_DT_TIME (Add Time to Date and Time).....                    | 2-637 |
| ADD_TIME (Add Time).....  | 2-633 |
| ADD_TOD_TIME (Add Time to Time of Day).....                     | 2-635 |
| AddDelimiter (Put Text Strings with Delimiters).....            | 2-606 |
| Addition.....   | 2-179 |
| Addition with Overflow Check.....                               | 2-183 |
| AddOU (Addition with Overflow Check).....                       | 2-183 |
| AND.....  | 2-17  |
| AND (AND).....  | 2-17  |
| AND (Logical AND).....  | 2-340 |
| AND NOT.....  | 2-17  |
| ANDN (AND NOT).....   | 2-17  |
| Array Addition.....   | 2-236 |
| Array BCD Conversion.....                                       | 2-271 |
| Array Comparison Equal.....                                     | 2-125 |
| Array Comparison Greater Than.....                              | 2-127 |
| Array Comparison Greater Than Or Equal.....                     | 2-127 |
| Array Comparison Less Than.....                                 | 2-127 |
| Array Comparison Less Than Or Equal.....                        | 2-127 |
| Array Comparison Not Equal.....                                 | 2-125 |
| Array Data Exchange.....  | 2-387 |
| Array Element Standard Deviation.....                           | 2-246 |
| Array Logical AND.....  | 2-347 |
| Array Logical Exclusive NOR.....                                | 2-347 |
| Array Logical Exclusive OR.....                                 | 2-347 |
| Array Logical OR.....   | 2-347 |
| Array Maximum.....  | 2-365 |
| Array Mean.....   | 2-244 |
| Array Minimum.....  | 2-365 |
| Array Move.....   | 2-389 |
| Array N-element Left Shift.....                                 | 2-411 |
| Array N-element Right Shift.....                                | 2-411 |
| Array Search.....   | 2-368 |
| Array Subtraction.....  | 2-240 |
| Array Unsigned Integer Conversion.....                          | 2-273 |
| Array Value Addition.....                                       | 2-238 |
| Array Value Comparison Equal.....                               | 2-130 |
| Array Value Comparison Greater Than.....                        | 2-132 |
| Array Value Comparison Greater Than Or Equal.....               | 2-132 |
| Array Value Comparison Less Than.....                           | 2-132 |
| Array Value Comparison Less Than Or Equal.....                  | 2-132 |
| Array Value Comparison Not Equal.....                           | 2-130 |
| Array Value Subtraction.....                                    | 2-242 |
| Array-to-Text String Conversion.....                            | 2-490 |
| AryAdd (Array Addition).....                                    | 2-236 |
| AryAddV (Array Value Addition).....                             | 2-238 |
| AryAnd (Array Logical AND).....                                 | 2-347 |
| AryByteTo (Conversion from Byte Array).....                     | 2-506 |
| AryCmpEQ (Array Comparison Equal).....                          | 2-125 |
| AryCmpEQV (Array Value Comparison Equal).....                   | 2-130 |

## A

|   |        |
|---|--------|
| ABS (Absolute Value).....                             | 2-207  |
| Absolute Value.....                                   | 2-207  |
| AC_StepProgram (Step Program).....                    | 2-830  |
| Accept TCP Socket.....                                | 2-1155 |
| Accumulation Timer.....                               | 2-148  |
| AccumulationTimer (Accumulation Timer).....           | 2-148  |
| ACOS (Principal Arc Cosine (COS <sup>-1</sup> ))..... | 2-214  |
| ActEventTask (Activate Event Task).....               | 2-1560 |
| Activate Event Task.....                              | 2-1560 |

- AryCmpGE (Array Comparison Greater Than Or Equal)..... 2-127
- AryCmpGEV (Array Value Comparison Greater Than Or Equal).....2-132
- AryCmpGT (Array Comparison Greater Than)..... 2-127
- AryCmpGTV (Array Value Comparison Greater Than)..2-132
- AryCmpLE (Array Comparison Less Than Or Equal).... 2-127
- AryCmpLEV (Array Value Comparison Less Than Or Equal).  
..... 2-132
- AryCmpLT (Array Comparison Less Than).....2-127
- AryCmpLTV (Array Value Comparison Less Than)..... 2-132
- AryCmpNE (Array Comparison Not Equal)..... 2-125
- AryCmpNEV (Array Value Comparison Not Equal)..... 2-130
- AryCRC16 (Calculate Array CRC-16)..... 2-578
- AryCRCCITT (Calculate Array CRC-CCITT)..... 2-576
- AryExchange (Array Data Exchange).....2-387
- AryLRC\_\*\* (Calculate Array LRC Group)..... 2-574
- AryMax (Array Maximum).....2-365
- AryMean (Array Mean).....2-244
- AryMin (Array Minimum).....2-365
- AryMove (Array Move).....2-389
- AryOr (Array Logical OR)..... 2-347
- ArySD (Array Element Standard Deviation)..... 2-246
- ArySearch (Array Search)..... 2-368
- AryShiftReg (Shift Register)..... 2-406
- AryShiftRegLR (Reversible Shift Register).....2-408
- ArySHL (Array N-element Left Shift)..... 2-411
- ArySHR (Array N-element Right Shift)..... 2-411
- ArySub (Array Subtraction).....2-240
- ArySubV (Array Value Subtraction)..... 2-242
- AryToBCD (Array BCD Conversion)..... 2-271
- AryToBin (Array Unsigned Integer Conversion).....2-273
- AryToString (Array-to-Text String Conversion)..... 2-490
- AryXor (Array Logical Exclusive OR).....2-347
- AryXorN (Array Logical Exclusive NOR)..... 2-347
- ASIN (Principal Arc Sine ( $\text{SIN}^{-1}$ ))..... 2-214
- ATAN (Principal Arc Tangent ( $\text{TAN}^{-1}$ ))..... 2-214
- B**
- 
- BackupToMemoryCard (SD Memory Card Backup).... 2-1509
- Band (Deadband Control).....2-359
- BCD Data Type-to-Unsigned Integer Conversion Group.....  
..... 2-262
- BCD\_TO\_\*\* (BCD Data Type-to-Unsigned Integer Conversion Group)..... 2-262
- BCD-to-Unsigned Integer Conversion Group..... 2-256
- BCDsToBin (Signed BCD-to-Signed Integer Conversion).....  
..... 2-265
- Binary Code-to-Gray Code Conversion.....2-485
- Binary Selection.....2-352
- BinToBCDs\_\*\* (Signed Integer-to-BCD Conversion Group)..  
..... 2-268
- BinToGray\_\*\* (Binary Code-to-Gray Code Conversion)2-485
- Bit Counter.....2-434
- Bit Decoder.....2-429
- Bit Encoder.....2-432
- Bit Pattern Copy (Bit String to Real Number) Group..... 2-395
- Bit Pattern Copy (Bit String to Signed Integer) Group... 2-393
- Bit Pattern Copy (Real Number to Bit String) Group..... 2-401
- Bit Pattern Copy (Real Number to Signed Integer) Group.....  
..... 2-403
- Bit Pattern Copy (Signed Integer to Bit String) Group... 2-397
- Bit Pattern Copy (Signed Integer to Real Number) Group.....  
..... 2-399
- Bit Reversal.....2-345
- Bit String Conversion Group.....2-327
- Bit String-to-Bit String Conversion Group.....2-288
- Bit String-to-Integer Conversion Group..... 2-285
- Bit String-to-Real Number Conversion Group..... 2-290
- Bit String-to-Text String Conversion Group..... 2-301
- BitCnt (Bit Counter)..... 2-434
- Block Set.....2-383
- BREAK (Break Loop).....2-98
- Break Down Date and Time.....2-692
- Break Loop..... 2-42, 2-98
- Broken Line Approximation with Broken Line Data Check....  
..... 2-448
- Broken Line Approximation without Broken Line Data Check  
..... 2-448
- Broken Line Data Check.....2-454
- Byte Data Join Group..... 2-498
- Byte Data Separation..... 2-496
- C**
- 
- Calculate Array CRC-16..... 2-578
- Calculate Array CRC-CCITT..... 2-576
- Calculate Array LRC Group..... 2-574
- Calculate Text String CRC-16.....2-572
- Calculate Text String CRC-CCITT..... 2-570
- Calculate Text String LRC..... 2-568
- Case.....2-32
- CASE (Case)..... 2-32
- Change File Name.....2-1497
- Change FTP Account..... 2-1220
- Change IP Address.....2-1211
- Change NTP Server Address..... 2-1224
- Change to NX Unit Write Mode..... 2-927
- ChangeFTPAccount (Change FTP Account).....2-1220
- ChangeIPAdr (Change IP Address).....2-1211
- ChangeNTPServerAdr (Change NTP Server Address).....  
..... 2-1224
- Check for Leap Year..... 2-682
- Check Subrange Variable.....2-1544
- CheckReal (Real Number Check)..... 2-252
- Checksum Calculation.....2-566
- ChkLeapYear (Check for Leap Year)..... 2-682
- ChkRange (Check Subrange Variable)..... 2-1544
- CIPClose (Close CIP Class 3 Connection)..... 2-1114
- CIPOpen (Open CIP Class 3 Connection)..... 2-1083
- CIPOpenWithDataSize (Open CIP Class 3 Connection with  
Specified Data Size)..... 2-1093
- CIPRead (Read Variable Class 3 Explicit).....2-1097
- CIPSend (Send Explicit Message Class 3).....2-1109
- CIPUCMM Read (Read Variable UCMM Explicit)..... 2-1117
- CIPUCMMSend (Send Explicit Message UCMM)..... 2-1130
- CIPUCMMWrite (Write Variable UCMM Explicit)..... 2-1123

- CIPWrite (Write Variable Class 3 Explicit).....2-1103  
Clear (Initialize).....2-391  
Clear Buffer.....2-1409  
Clear String.....2-600  
Clear TCP/UDP Socket Receive Buffer.....2-1179  
ClearString (Clear String).....2-600  
Close CIP Class 3 Connection.....2-1114  
Close File.....2-1445  
Close TCP/UDP Socket.....2-1176  
Cmp (Compare).....2-118  
ColmToLine\_\*\* (Column to Line Conversion Group).....2-435  
Column to Line Conversion Group.....2-435  
Combine Real Number Mantissa and Exponent.....2-467  
Compare.....2-118  
CONCAT (Concatenate String).....2-582  
CONCAT\_DATE\_TOD (Concatenate Date and Time of Day)  
.....2-654  
Concatenate Date and Time of Day.....2-654  
Concatenate String.....2-582  
Connect EtherCAT Slave.....2-1015  
Connect TCP Socket.....2-1158  
Conversion from Byte Array.....2-506  
Conversion to Byte Array.....2-500  
Convert Date and Time to Seconds.....2-664  
Convert Date to Seconds.....2-666  
Convert Days to Month.....2-686  
Convert Nanoseconds to Time.....2-679  
Convert Seconds to Date.....2-672  
Convert Seconds to Date and Time.....2-670  
Convert Seconds to Time.....2-680  
Convert Seconds to Time of Day.....2-674  
Convert Time of Day to Seconds.....2-668  
Convert Time to Nanoseconds.....2-676  
Convert Time to Seconds.....2-677  
Convert to Lowercase.....2-602  
Convert to Uppercase.....2-602  
Copy File.....2-1484  
Copy\*\*To\*\*\* (Bit Pattern Copy (Bit String to Real Number)  
Group).....2-395  
Copy\*\*To\*\*\* (Bit Pattern Copy (Real Number to Bit String)  
Group).....2-401  
Copy\*\*ToNum (Bit Pattern Copy (Bit String to Signed Inte-  
ger) Group).....2-393  
Copy\*\*ToNum (Bit Pattern Copy (Real Number to Signed In-  
teger) Group).....2-403  
CopyNumTo\*\* (Bit Pattern Copy (Signed Integer to Bit  
String) Group).....2-397  
CopyNumTo\*\* (Bit Pattern Copy (Signed Integer to Real  
Number) Group).....2-399  
COS (Cosine in Radians).....2-211  
Cosine in Radians.....2-211  
Create Directory.....2-1503  
Create UDP Socket.....2-1141  
Create User-defined Error.....2-868  
Create User-defined Information.....2-912  
CTD (Down-counter).....2-156  
CTD\_\*\* (Down-counter Group).....2-158  
CTU (Up-counter).....2-161  
CTU\_\*\* (Up-counter Group).....2-164  
CTUD (Up-down Counter).....2-167  
CTUD\_\*\* (Up-down Counter Group).....2-172
- ## D
- Data Exchange.....2-385  
Data Trace Sampling.....2-859  
Data Trace Trigger.....2-862  
Date and Time-to-Text String Conversion.....2-479  
Date-to-Text String Conversion.....2-481  
DateStructToDt (Join Time).....2-695  
DateToSec (Convert Date to Seconds).....2-666  
DateToString (Date-to-Text String Conversion).....2-481  
DaysToMonth (Convert Days to Month).....2-686  
Dead Zone Control.....2-361  
Deadband Control.....2-359  
Dec (Decrement).....2-232  
Decoder (Bit Decoder).....2-429  
Decrement.....2-232  
Degrees to Radians.....2-209  
DegToRad (Degrees to Radians).....2-209  
DELETE (Delete String).....2-595  
Delete Directory.....2-1506  
Delete File.....2-1492  
Delete from Stack.....2-539  
Delete FTP Server Directory.....2-1273  
Delete FTP Server File.....2-1263  
Delete String.....2-595  
Determine Task Status.....2-1552  
DirCreate (Create Directory).....2-1503  
DirRemove (Delete Directory).....2-1506  
Disable Program.....2-961  
Disconnect EtherCAT Slave.....2-1007  
Dispart8Bit (Byte Data Separation).....2-496  
DispartDigit (Four-bit Separation).....2-492  
DispartReal (Separate Mantissa and Exponent).....2-464  
DIV (Division).....2-202  
Divide Time.....2-652  
Division.....2-202  
DIVTIME (Divide Time).....2-652  
Down (Down Trigger).....2-48  
Down Trigger.....2-48  
Down-counter.....2-156  
Down-counter Group.....2-158  
DT\_TO\_DATE (Extract Date from Date and Time).....2-658  
DT\_TO\_TOD (Extract Time of Day from Date and Time).....  
.....2-656  
DtToDateStruct (Break Down Date and Time).....2-692  
DtToSec (Convert Date and Time to Seconds).....2-664  
DtToString (Date and Time-to-Text String Conversion).....2-479
- ## E
- EC\_ChangeEnableSetting (Enable/Disable EtherCAT Slave)  
.....2-1017  
EC\_CoESDORed (Read EtherCAT CoE SDO).....2-989  
EC\_CoESDOWrite (Write EtherCAT CoE SDO).....2-986  
EC\_ConnectSlave (Connect EtherCAT Slave).....2-1015  
EC\_CopyMon (Transfer EtherCAT Packets).....2-1005  
EC\_DisconnectSlave (Disconnect EtherCAT Slave).....2-1007



- EC\_SaveMon (Save EtherCAT Packets).....2-1003  
 EC\_StartMon (Start EtherCAT Packet Monitor).....2-995  
 EC\_StopMon (Stop EtherCAT Packet Monitor).....2-1001  
 Enable Program.....2-952  
 Enable/Disable EtherCAT Slave.....2-1017  
 Encoder (Bit Encoder).....2-432  
 End.....2-74  
 End (End).....2-74  
 Enumeration-to-Integer.....2-331  
 EnumToNum (Enumeration-to-Integer).....2-331  
 EQ (Equal).....2-102  
 EQascii (Text String Comparison Equal).....2-111  
 Equal.....2-102  
 Exchange (Data Exchange).....2-385  
 ExecPMCR (Protocol Macro).....2-1278  
 EXIT (Break Loop).....2-42  
 EXP (Natural Exponential Operation).....2-224  
 Exponentiation.....2-226  
 EXPT (Exponentiation).....2-226  
 Extract Date from Date and Time.....2-658  
 Extract Time of Day from Date and Time.....2-656
- ## F
- F\_TRIG (Down Trigger).....2-48  
 FileClose (Close File).....2-1445  
 FileCopy (Copy File).....2-1484  
 FileGets (Get Text String).....2-1467  
 FileOpen (Open File).....2-1441  
 FilePuts (Put Text String).....2-1475  
 FileRead (Read File).....2-1451  
 FileReadVar (Read Variable from File).....2-1436  
 FileRemove (Delete File).....2-1492  
 FileRename (Change File Name).....2-1497  
 FileSeek (Seek File).....2-1448  
 FileWrite (Write File).....2-1459  
 FileWriteVar (Write Variable to File).....2-1430  
 FIND (Find String).....2-589  
 Find String.....2-589  
 First In First Out.....2-533  
 Fixed-decimal Number-to-Text String Conversion.....2-474  
 Fixed-length Decimal Text String Conversion.....2-469  
 Fixed-length Hexadecimal Text String Conversion.....2-469  
 FixNumToString (Fixed-decimal Number-to-Text String Conversion).....2-474  
 FOR (Repeat Start).....2-91  
 Four-bit Join Group.....2-494  
 Four-bit Separation.....2-492  
 Fraction (Real Number Fraction).....2-250  
 FTPGetFile (Get File from FTP Server).....2-1243  
 FTPGetFileList (Get FTP Server File List).....2-1229  
 FTPPutFile (Put File onto FTP Server).....2-1252  
 FTPRemoveDir (Delete FTP Server Directory).....2-1273  
 FTPRemoveFile (Delete FTP Server File).....2-1263
- ## G
- GE (Greater Than Or Equal).....2-108  
 GEascii (Text String Comparison Greater Than or Equal).....2-115
- Get Byte Length.....2-599  
 Get Clock Pulse Group.....2-1567  
 Get Day of Week.....2-688  
 Get Days in Month.....2-683  
 Get EtherCAT Error Status.....2-898  
 Get EtherNet/IP Error Status.....2-886  
 Get File from FTP Server.....2-1243  
 Get First Number of Array.....2-518  
 Get FTP Server File List.....2-1229  
 Get I/O Bus Error Status.....2-884  
 Get Incrementing Free-running Counter Group.....2-1569  
 Get Last Number of Array.....2-518  
 Get Motion Control Error Status.....2-894  
 Get Number of Array Elements.....2-512  
 Get Number of Records.....2-557  
 Get NX Bus Error Status.....2-903  
 Get NX Unit Error Status.....2-905  
 Get PLC Controller Error Status.....2-880  
 Get String Any.....2-587  
 Get String Left.....2-584  
 Get String Right.....2-584  
 Get Text String.....2-1467  
 Get Text Strings Minus Delimiters.....2-618  
 Get Time of Day.....2-662  
 Get User-defined Error Status.....2-875  
 Get Week Number.....2-690  
 Get\*\*Clk (Get Clock Pulse Group).....2-1567  
 Get\*\*Cnt (Get Incrementing Free-running Counter Group).....2-1569  
 GetAlarm (Get User-defined Error Status).....2-875  
 GetByteLen (Get Byte Length).....2-599  
 GetCJBError (Get I/O Bus Error Status).....2-884  
 GetDayOfWeek (Get Day of Week).....2-688  
 GetDaysOfMonth (Get Days in Month).....2-683  
 GetECError (Get EtherCAT Error Status).....2-898  
 GetEIPErr (Get EtherNet/IP Error Status).....2-886  
 GetMCError (Get Motion Control Error Status).....2-894  
 GetMyTaskInterval (Read Current Task Period).....2-1550  
 GetMyTaskStatus (Read Current Task Status).....2-1547  
 GetNTPStatus (Read NTP Status).....2-919  
 GetNXBError (Get NX Bus Error Status).....2-903  
 GetNXUnitError (Get NX Unit Error Status).....2-905  
 GetPLCError (Get PLC Controller Error Status).....2-880  
 GetTime (Get Time of Day).....2-662  
 GetTraceStatus (Read Data Trace Status).....2-865  
 GetWeekOfYear (Get Week Number).....2-690  
 Gray (Gray Code Conversion).....2-439  
 Gray Code Conversion.....2-439  
 Gray Code-to-Binary Code Conversion Group.....2-485  
 GrayToBin\_\*\* (Gray Code-to-Binary Code Conversion Group).....2-485  
 Greater Than.....2-108  
 Greater Than Or Equal.....2-108  
 GT (Greater Than).....2-108  
 GTascii (Text String Comparison Greater Than).....2-115
- ## H
- Heating/Cooling PID with Autotuning.....2-742

|   |       |
|---|-------|
| Hexadecimal Text String-to-Number Conversion Group.....                     |       |
| .....   | 2-472 |
| HexStringToNum_** (Hexadecimal Text String-to-Number Conversion Group)..... | 2-472 |
| Hundred-ms Timer.....   | 2-152 |

**I**

|   |        |
|---|--------|
| If.....   | 2-28   |
| IF (If).....                                    | 2-28   |
| Inc (Increment).....                            | 2-232  |
| Increment.....                                  | 2-232  |
| Initialize.....                                 | 2-391  |
| INSERT (Insert String).....                     | 2-597  |
| Insert into Stack.....                          | 2-536  |
| Insert String.....                              | 2-597  |
| Integer Conversion Group.....                   | 2-325  |
| Integer-to-Bit String Conversion Group.....     | 2-280  |
| Integer-to-Enumeration.....                     | 2-333  |
| Integer-to-Integer Conversion Group.....        | 2-277  |
| Integer-to-Real Number Conversion Group.....    | 2-283  |
| Integer-to-Text String Conversion Group.....    | 2-299  |
| IOL_ReadObj (Read IO-Link Device Object).....   | 2-1062 |
| IOL_WriteObj (Write IO-Link Device Object)..... | 2-1071 |

**J**

|                 |       |
|-----------------|-------|
| JMP (Jump)..... | 2-89  |
| Join Time.....  | 2-695 |
| Jump.....       | 2-89  |

**L**

|   |        |
|---|--------|
| Last In First Out.....  | 2-533  |
| LD (Load).....  | 2-14   |
| LDN (Load NOT).....   | 2-14   |
| LE (Less Than Or Equal).....  | 2-108  |
| LEascii (Text String Comparison Less Than or Equal)..                                 | 2-115  |
| LEFT (Get String Left).....   | 2-584  |
| LEN (String Length).....  | 2-591  |
| Less Than.....  | 2-108  |
| Less Than Or Equal.....   | 2-108  |
| LIMIT (Limiter).....  | 2-357  |
| LimitAlarm_** (Upper/Lower Limit Alarm Group).....                                    | 2-799  |
| LimitAlarmDv_** (Upper/Lower Deviation Alarm Group).....                              | 2-804  |
| LimitAlarmDvStbySeq_** (Upper/Lower Deviation Alarm with Standby Sequence Group)..... | 2-809  |
| Limiter.....  | 2-357  |
| Line to Column Conversion.....  | 2-437  |
| LineToColm (Line to Column Conversion).....   | 2-437  |
| LN (Natural Logarithm).....   | 2-220  |
| Load.....   | 2-14   |
| Load NOT.....   | 2-14   |
| Lock (Lock Tasks).....  | 2-1554 |
| Lock Tasks.....   | 2-1554 |
| LOG (Logarithm Base 10).....  | 2-220  |
| Logarithm Base 10.....  | 2-220  |
| Logical AND.....  | 2-340  |
| Logical Exclusive NOR.....  | 2-343  |

|   |       |
|---|-------|
| Logical Exclusive OR.....                                 | 2-340 |
| Logical OR.....   | 2-340 |
| LOWER_BOUND (Get First Number of Array).....              | 2-518 |
| LREAL-to-Formatted Text String.....                       | 2-311 |
| LrealToFormatString (LREAL-to-Formatted Text String)..... | 2-311 |
| LT (Less Than).....                                       | 2-108 |
| LTascii (Text String Comparison Less Than).....           | 2-115 |

**M**

|   |        |
|---|--------|
| Master Control End.....                             | 2-76   |
| Master Control Start.....                           | 2-76   |
| MAX (Maximum).....                                  | 2-363  |
| Maximum.....  | 2-363  |
| Maximum Record Search.....                          | 2-560  |
| MC (Master Control Start).....                      | 2-76   |
| MCR (Master Control End).....                       | 2-76   |
| MemCopy (Memory Copy).....                          | 2-381  |
| Memory Copy.....                                    | 2-381  |
| MID (Get String Any).....                           | 2-587  |
| MIN (Minimum).....                                  | 2-363  |
| Minimum.....  | 2-363  |
| Minimum Record Search.....                          | 2-560  |
| MOD (Modulo-division).....                          | 2-205  |
| ModbusTCPCmd (Send Modbus TCP General Command)..... | 2-1187 |
| ModbusTCPRead (Send Modbus TCP Read Command).....   | 2-1195 |
| ModbusTCPWrite (Send Modbus TCP Write Command)..... | 2-1203 |
| ModReal (Real Number Modulo-division).....          | 2-248  |
| Modulo-division.....                                | 2-205  |
| Move.....   | 2-372  |
| MOVE (Move).....                                    | 2-372  |
| Move Bit.....                                       | 2-375  |
| Move Bits.....                                      | 2-379  |
| Move Digit.....                                     | 2-377  |
| MoveBit (Move Bit).....                             | 2-375  |
| MoveDigit (Move Digit).....                         | 2-377  |
| Moving Average.....                                 | 2-457  |
| MovingAverage (Moving Average).....                 | 2-457  |
| MUL (Multiplication).....                           | 2-194  |
| MuIOU (Multiplication with Overflow Check).....     | 2-198  |
| MULTIME (Multiply Time).....                        | 2-650  |
| Multiplexer.....                                    | 2-354  |
| Multiplication.....                                 | 2-194  |
| Multiplication with Overflow Check.....             | 2-198  |
| Multiply Time.....                                  | 2-650  |
| MUX (Multiplexer).....                              | 2-354  |

**N**

|  |        |
|--|--------|
| N-bit Left Shift.....                            | 2-414  |
| N-bit Read Group.....                            | 2-1540 |
| N-bit Right Shift.....                           | 2-414  |
| N-bit Write Group.....                           | 2-1542 |
| NanoSecToTime (Convert Nanoseconds to Time)..... | 2-679  |
| Natural Exponential Operation.....               | 2-224  |
| Natural Logarithm.....                           | 2-220  |
| NE (Not Equal).....                              | 2-105  |

NEascii (Text String Comparison Not Equal)..... 2-113  
 Neg (Reverse Sign)..... 2-427  
 NEXT (Repeat End)..... 2-91  
 NOT (Bit Reversal)..... 2-345  
 Not Equal..... 2-105  
 NSHLC (Shift N-bits Left with Carry)..... 2-417  
 NSHRC (Shift N-bits Right with Carry)..... 2-417  
 NumToDecString (Fixed-length Decimal Text String Conversion)..... 2-469  
 NumToEnum (Integer-to-Enumeration)..... 2-333  
 NumToHexString (Fixed-length Hexadecimal Text String Conversion)..... 2-469  
 NX\_AryDOutTimeStamp (Write Digital Output Array with Specified Time Stamp)..... 2-1530  
 NX\_ChangeWriteMode (Change to NX Unit Write Mode)..... 2-927  
 NX\_DOutTimeStamp (Write Digital Output with Specified Time Stamp)..... 2-1524  
 NX\_ModbusRtuCmd (Send Modbus RTU General Command)..... 2-1358  
 NX\_ModbusRtuRead (Send Modbus RTU Read Command)..... 2-1369  
 NX\_ModbusRtuWrite (Send Modbus RTU Write Command)..... 2-1380  
 NX\_ReadObj (Read NX Unit Object)..... 2-1053  
 NX\_ReadTotalPowerOnTime (Read NX Unit Total Power ON Time)..... 2-942  
 NX\_SaveParam (Save NX Unit Parameters)..... 2-933  
 NX\_SerialBufClear (Clear Buffer)..... 2-1409  
 NX\_SerialRcv (Receive No-protocol Data)..... 2-1343  
 NX\_SerialSend (Send No-protocol Data)..... 2-1330  
 NX\_SerialSigCtl (Serial Control Signal ON/OFF Switching)..... 2-1391  
 NX\_SerialSigRead (Read Serial Control Signal)..... 2-1399  
 NX\_SerialStartMon (Start Serial Line Monitoring)..... 2-1419  
 NX\_SerialStatusRead (Read Serial Port Status)..... 2-1404  
 NX\_SerialStopMon (Stop Serial Line Monitoring)..... 2-1424  
 NX\_WriteObj (Write NX Unit Object)..... 2-1037

## O

Off-Delay Timer..... 2-142  
 On-Delay Timer..... 2-136  
 Open CIP Class 3 Connection (Large\_Forward\_Open)..... 2-1083  
 Open CIP Class 3 Connection with Specified Data Size..... 2-1093  
 Open File..... 2-1441  
 OR..... 2-20  
 OR (Logical OR)..... 2-340  
 OR NOT..... 2-20  
 ORN (OR NOT)..... 2-20  
 Out (Output)..... 2-23  
 OutABit (Output A Bit)..... 2-71  
 OutNot (Output NOT)..... 2-23  
 Output..... 2-23  
 Output A Bit..... 2-71  
 Output NOT..... 2-23

## P

PackDword (4-byte Join)..... 2-516  
 PackWord (2-byte Join)..... 2-514  
 PID Control with Autotuning..... 2-712  
 PIDAT (PID Control with Autotuning)..... 2-712  
 PIDAT\_HeatCool (Heating/Cooling PID with Autotuning)..... 2-742  
 PLC\_ReadTotalPowerOnTime (Read PLC Total Power ON Time)..... 2-939  
 PrgStart(Enable Program)..... 2-952  
 PrgStatus (Read Program Status)..... 2-980  
 PrgStop (Disable Program)..... 2-961  
 Principal Arc Cosine (COS<sup>-1</sup>)..... 2-214  
 Principal Arc Sine (SIN<sup>-1</sup>)..... 2-214  
 Principal Arc Tangent (TAN<sup>-1</sup>)..... 2-214  
 Production Information..... 27  
 Protocol Macro..... 2-1278  
 Push onto Stack..... 2-524  
 Put File onto FTP Server..... 2-1252  
 Put Text String..... 2-1475  
 Put Text Strings with Delimiters..... 2-606  
 PWLApprox (Broken Line Approximation with Broken Line Data Check)..... 2-448  
 PWLApproxNoLineChk (Broken Line Approximation without Broken Line Data Check)..... 2-448  
 PWLLineChk (Broken Line Data Check)..... 2-454

## R

R\_TRIG (Up Trigger)..... 2-48  
 Radians to Degrees..... 2-209  
 RadToDeg (Radians to Degrees)..... 2-209  
 Rand (Random Number)..... 2-234  
 Random Number..... 2-234  
 Range Record Search..... 2-546  
 Read Current Task Period..... 2-1550  
 Read Current Task Status..... 2-1547  
 Read Data Trace Status..... 2-865  
 Read EtherCAT CoE SDO..... 2-989  
 Read File..... 2-1451  
 Read IO-Link Device Object..... 2-1062  
 Read NTP Status..... 2-919  
 Read NX Unit Object..... 2-1053  
 Read NX Unit Total Power ON Time..... 2-942  
 Read PLC Total Power ON Time..... 2-939  
 Read Program Status..... 2-980  
 Read Serial Control Signal..... 2-1399  
 Read Serial Port Status..... 2-1404  
 Read TCP Socket Status..... 2-1173  
 Read Variable Class 3 Explicit..... 2-1097  
 Read Variable from File..... 2-1436  
 Read Variable UCMM Explicit..... 2-1117  
 ReadNbit\_\*\* (N-bit Read Group)..... 2-1540  
 Real Number Check..... 2-252  
 Real Number Conversion Group..... 2-329  
 Real Number Fraction..... 2-250  
 Real Number Modulo-division..... 2-248  
 Real Number-to-Bit String Conversion Group..... 2-295

|   |        |   |        |
|---|--------|---|--------|
| Real Number-to-Integer Conversion Group.....            | 2-292  | Save EtherCAT Packets.....  | 2-1003 |
| Real Number-to-Real Number Conversion Group.....        | 2-297  | Save NX Unit Parameters.....  | 2-933  |
| Real Number-to-Text String Conversion Group.....        | 2-303  | Scale Transformation.....   | 2-827  |
| REAL-to-Formatted Text String.....                      | 2-305  | ScaleTrans (Scale Transformation).....                                  | 2-827  |
| RealToFormatString (REAL-to-Formatted Text String)..... | 2-305  | SCU Receive Serial.....   | 2-1303 |
| Receive No-protocol Data.....                           | 2-1343 | SCU Receive Serial without Receive Buffer Clear.....                    | 2-1303 |
| RecMax (Maximum Record Search).....                     | 2-560  | SCU Send Serial.....  | 2-1292 |
| RecMin (Minimum Record Search).....                     | 2-560  | SD Memory Card Backup.....  | 2-1509 |
| RecNum (Get Number of Records).....                     | 2-557  | SecToDate (Convert Seconds to Date).....                                | 2-672  |
| Record Search.....                                      | 2-541  | SecToDt (Convert Seconds to Date and Time).....                         | 2-670  |
| Record Sort.....  | 2-551  | SecToTime (Convert Seconds to Time).....                                | 2-680  |
| RecRangeSearch (Range Record Search).....               | 2-546  | SecToTod (Convert Seconds to Time of Day).....                          | 2-674  |
| RecSearch (Record Search).....                          | 2-541  | Seek File.....  | 2-1448 |
| RecSort (Record Sort).....                              | 2-551  | SEL (Binary Selection).....   | 2-352  |
| Repeat.....   | 2-39   | Send Command.....   | 2-1318 |
| REPEAT (Repeat).....                                    | 2-39   | Send Explicit Message Class 3.....                                      | 2-1109 |
| Repeat End.....   | 2-91   | Send Explicit Message UCMM.....   | 2-1130 |
| Repeat Start.....                                       | 2-91   | Send Modbus RTU General Command.....                                    | 2-1358 |
| REPLACE (Replace String).....                           | 2-593  | Send Modbus RTU Read Command.....                                       | 2-1369 |
| Replace String.....                                     | 2-593  | Send Modbus RTU Write Command.....                                      | 2-1380 |
| Reset.....  | 2-62   | Send Modbus TCP General Command.....                                    | 2-1187 |
| Reset A Bit.....  | 2-69   | Send Modbus TCP Read Command.....                                       | 2-1195 |
| Reset Bits.....   | 2-66   | Send Modbus TCP Write Command.....                                      | 2-1203 |
| Reset CJ Bus Controller Error.....                      | 2-882  | Send No-protocol Data.....  | 2-1330 |
| Reset EtherCAT Error.....                               | 2-896  | SendCmd (Send Command).....   | 2-1318 |
| Reset Motion Control Error.....                         | 2-888  | Separate Mantissa and Exponent.....                                     | 2-464  |
| Reset NX Bus Error.....                                 | 2-901  | Serial Control Signal ON/OFF Switching.....                             | 2-1391 |
| Reset PLC Controller Error.....                         | 2-877  | SerialRcv (SCU Receive Serial).....                                     | 2-1303 |
| Reset User-defined Error.....                           | 2-873  | SerialRcvNoClear (SCU Receive Serial without Receive Buffer Clear)..... | 2-1303 |
| Reset-Priority Keep.....                                | 2-56   | SerialSend (SCU Send Serial).....                                       | 2-1292 |
| ResetABit (Reset A Bit).....                            | 2-69   | Set.....  | 2-62   |
| ResetAlarm (Reset User-defined Error).....              | 2-873  | Set A Bit.....  | 2-69   |
| ResetBits (Reset Bits).....                             | 2-66   | Set Bits.....   | 2-66   |
| ResetCJBError (Reset CJ Bus Controller Error).....      | 2-882  | Set TCP Socket Option.....  | 2-1182 |
| ResetECError (Reset EtherCAT Error).....                | 2-896  | Set Time.....   | 2-660  |
| ResetMCError (Reset Motion Control Error).....          | 2-888  | Set-Priority Keep.....  | 2-59   |
| ResetNXBError (Reset NX Bus Error).....                 | 2-901  | SetABit (Set A Bit).....  | 2-69   |
| ResetPLCError (Reset PLC Controller Error).....         | 2-877  | SetAlarm (Create User-defined Error).....                               | 2-868  |
| ResetUnit (Restart Unit).....                           | 2-914  | SetBits (Set Bits).....   | 2-66   |
| Restart NX Units.....                                   | 2-921  | SetBlock (Block Set).....   | 2-383  |
| Restart Unit.....                                       | 2-914  | SetInfo (Create User-defined Information).....                          | 2-912  |
| RestartNXUnit (Restart NX Units).....                   | 2-921  | SetTime (Set Time).....   | 2-660  |
| Return.....   | 2-75   | Shift N-bits Left with Carry.....                                       | 2-417  |
| RETURN (Return).....                                    | 2-75   | Shift N-bits Right with Carry.....                                      | 2-417  |
| Reverse Sign.....                                       | 2-427  | Shift Register.....   | 2-406  |
| Reversible Shift Register.....                          | 2-408  | SHL (N-bit Left Shift).....   | 2-414  |
| RIGHT (Get String Right).....                           | 2-584  | SHR (N-bit Right Shift).....  | 2-414  |
| ROL (Rotate N-bits Left).....                           | 2-420  | Signed BCD-to-Signed Integer Conversion.....                            | 2-265  |
| ROR (Rotate N-bits Right).....                          | 2-420  | Signed Integer-to-BCD Conversion Group.....                             | 2-268  |
| Rotate N-bits Left.....                                 | 2-420  | SIN (Sine in Radians).....  | 2-211  |
| Rotate N-bits Right.....                                | 2-420  | Sine in Radians.....  | 2-211  |
| Round (Round Off Real Number).....                      | 2-336  | SizeOfAry (Get Number of Array Elements).....                           | 2-512  |
| Round Off Real Number.....                              | 2-336  | SJIS to UTF-8 Character Code Conversion.....                            | 2-446  |
| Round Up Real Number.....                               | 2-336  | SJISToUTF8 (SJIS to UTF-8 Character Code Conversion).....               | 2-446  |
| RoundUp (Round Up Real Number).....                     | 2-336  | .....   | 2-446  |
| RS (Reset-Priority Keep).....                           | 2-56   | SktClearBuf (Clear TCP/UDP Socket Receive Buffer).....                  | 2-1179 |
|   |        | SktClose (Close TCP/UDP Socket).....                                    | 2-1176 |
|   |        | SktGetTCPStatus (Read TCP Socket Status).....                           | 2-1173 |

---

**S**



|   |        |  |        |
|---|--------|--|--------|
| SkSetOption (Set TCP Socket Option).....                            | 2-1182 | TAN (Tangent in Radians).....                            | 2-211  |
| SkTCPAccept (Accept TCP Socket).....                                | 2-1155 | Tangent in Radians.....                                  | 2-211  |
| SkTCPConnect (Connect TCP Socket).....                              | 2-1158 | Task_IsActive (Determine Task Status).....               | 2-1552 |
| SkTCPRcv (TCP Socket Receive).....                                  | 2-1167 | TCP Socket Receive.....                                  | 2-1167 |
| SkTCPSend (TCP Socket Send).....                                    | 2-1170 | TCP Socket Send.....                                     | 2-1170 |
| SkUDPCreate (Create UDP Socket).....                                | 2-1141 | Test A Bit.....  | 2-52   |
| SkUDPRcv (UDP Socket Receive).....                                  | 2-1149 | Test A Bit NOT.....                                      | 2-52   |
| SkUDPSend (UDP Socket Send).....                                    | 2-1152 | TestABit (Test A Bit).....                               | 2-52   |
| SQRT (Square Root).....   | 2-217  | TestABitN (Test A Bit NOT).....                          | 2-52   |
| Square Root.....  | 2-217  | Text String Comparison Equal.....                        | 2-111  |
| SR (Set-Priority Keep).....   | 2-59   | Text String Comparison Greater Than.....                 | 2-115  |
| StackDel (Delete from Stack).....                                   | 2-539  | Text String Comparison Greater Than or Equal.....        | 2-115  |
| StackFIFO (First In First Out).....                                 | 2-533  | Text String Comparison Less Than.....                    | 2-115  |
| StackIns (Insert into Stack).....                                   | 2-536  | Text String Comparison Less Than or Equal.....           | 2-115  |
| StackLIFO (Last In First Out).....                                  | 2-533  | Text String Comparison Not Equal.....                    | 2-113  |
| StackPush (Push onto Stack).....                                    | 2-524  | Text String-to-Array Conversion.....                     | 2-488  |
| Start EtherCAT Packet Monitor.....                                  | 2-995  | Text String-to-Bit String Conversion Group.....          | 2-319  |
| Start Serial Line Monitoring.....                                   | 2-1419 | Text String-to-Fixed-decimal Conversion.....             | 2-476  |
| Step Program.....   | 2-830  | Text String-to-Integer Conversion Group.....             | 2-317  |
| Stop EtherCAT Packet Monitor.....                                   | 2-1001 | Text String-to-Real Number Conversion Group.....         | 2-321  |
| Stop Serial Line Monitoring.....                                    | 2-1424 | Time of Day-to-Text String Conversion.....               | 2-483  |
| String Length.....  | 2-591  | Time-proportional Output.....                            | 2-779  |
| STRING_TO_** (Text String-to-Bit String Conversion Group)<br>.....  | 2-319  | TimeProportionalOut (Time-proportional Output).....      | 2-779  |
| STRING_TO_** (Text String-to-Integer Conversion Group)..<br>.....   | 2-317  | Timer (Hundred-ms Timer).....                            | 2-152  |
| STRING_TO_** (Text String-to-Real Number Conversion<br>Group).....  | 2-321  | Timer Pulse.....   | 2-145  |
| StringCRC16 (Calculate Text String CRC-16).....                     | 2-572  | TimeToNanoSec (Convert Time to Nanoseconds).....         | 2-676  |
| StringCRCCCITT (Calculate Text String CRC-CCITT).....               | 2-570  | TimeToSec (Convert Time to Seconds).....                 | 2-677  |
| StringLRC (Calculate Text String LRC).....                          | 2-568  | TO_** (Bit String Conversion Group).....                 | 2-327  |
| StringSum (Checksum Calculation).....                               | 2-566  | TO_** (Integer Conversion Group).....                    | 2-325  |
| StringToAry (Text String-to-Array Conversion).....                  | 2-488  | TO_** (Real Number Conversion Group).....                | 2-329  |
| StringToFixNum (Text String-to-Fixed-decimal Conversion)..<br>..... | 2-476  | ToAryByte (Conversion to Byte Array).....                | 2-500  |
| SUB (Subtraction).....  | 2-187  | TodToSec (Convert Time of Day to Seconds).....           | 2-668  |
| SUB_DATE_DATE (Subtract Date).....                                  | 2-645  | TodToString (Time of Day-to-Text String Conversion)..... | 2-483  |
| SUB_DT_DT (Subtract Date and Time).....                             | 2-646  | TOF (Off-Delay Timer).....                               | 2-142  |
| SUB_DT_TIME (Subtract Time from Date and Time).....                 | 2-648  | ToLCase (Convert to Lowercase).....                      | 2-602  |
| SUB_TIME (Subtract Time).....                                       | 2-639  | TON (On-Delay Timer).....                                | 2-136  |
| SUB_TOD_TIME (Subtract Time from Time of Day).....                  | 2-641  | ToUCase (Convert to Uppercase).....                      | 2-602  |
| SUB_TOD_TOD (Subtract Time of Day).....                             | 2-643  | TP (Timer Pulse).....                                    | 2-145  |
| SubDelimiter (Get Text Strings Minus Delimiters).....               | 2-618  | TraceSamp (Data Trace Sampling).....                     | 2-859  |
| SubOU (Subtraction with Overflow Check).....                        | 2-190  | TraceTrig (Data Trace Trigger).....                      | 2-862  |
| Subtract Date.....  | 2-645  | TransBits (Move Bits).....                               | 2-379  |
| Subtract Date and Time.....   | 2-646  | Transfer EtherCAT Packets.....                           | 2-1005 |
| Subtract Time.....  | 2-639  | Trim String Left.....                                    | 2-604  |
| Subtract Time from Date and Time.....                               | 2-648  | Trim String Right.....                                   | 2-604  |
| Subtract Time from Time of Day.....                                 | 2-641  | TrimL (Trim String Left).....                            | 2-604  |
| Subtract Time of Day.....   | 2-643  | TrimR (Trim String Right).....                           | 2-604  |
| Subtraction.....  | 2-187  | TRUNC (Truncate).....                                    | 2-336  |
| Subtraction with Overflow Check.....                                | 2-190  | Truncate.....  | 2-336  |
| Swap (Swap Bytes).....  | 2-425  | Truncate Date and Time.....                              | 2-702  |
| Swap Bytes.....   | 2-425  | Truncate Time.....                                       | 2-698  |
|   |        | Truncate Time of Day.....                                | 2-706  |
|   |        | TruncDt (Truncate Date and Time).....                    | 2-702  |
|   |        | TruncTime (Truncate Time).....                           | 2-698  |
|   |        | TruncTod (Truncate Time of Day).....                     | 2-706  |

## T

|                                  |       |
|----------------------------------|-------|
| Table Comparison.....            | 2-122 |
| TableCmp (Table Comparison)..... | 2-122 |

## U

|                         |        |
|-------------------------|--------|
| UDP Socket Receive..... | 2-1149 |
| UDP Socket Send.....    | 2-1152 |

|   |        |
|---|--------|
| Unite8Bit_** (Byte Data Join Group).....                          | 2-498  |
| UniteDigit_** (Four-bit Join Group).....                          | 2-494  |
| UniteReal (Combine Real Number Mantissa and Exponent).<br>.....   | 2-467  |
| Unlock (Unlock Tasks).....  | 2-1554 |
| Unlock Tasks.....   | 2-1554 |
| Unsigned Integer-to-BCD Conversion Group.....                     | 2-259  |
| Up (Up Trigger).....  | 2-48   |
| Up Trigger.....   | 2-48   |
| Up-counter.....   | 2-161  |
| Up-counter Group.....   | 2-164  |
| Up-down Counter.....  | 2-167  |
| Up-down Counter Group.....  | 2-172  |
| UPPER_BOUND (Get Last Number of Array).....                       | 2-518  |
| Upper/Lower Deviation Alarm Group.....                            | 2-804  |
| Upper/Lower Deviation Alarm with Standby Sequence<br>Group.....   | 2-809  |
| Upper/Lower Limit Alarm Group.....                                | 2-799  |
| UTF-8 to SJIS Character Code Conversion.....                      | 2-444  |
| UTF8ToSJIS (UTF-8 to SJIS Character Code Conversion)....<br>..... | 2-444  |

## V

---

|              |    |
|--------------|----|
| Version..... | 25 |
|--------------|----|

## W

---

|  |        |
|--|--------|
| While.....   | 2-36   |
| WHILE (While).....                                   | 2-36   |
| Write Digital Output Array with Specified Time Stamp | 2-1530 |
| Write Digital Output with Specified Time Stamp.....  | 2-1524 |
| Write EtherCAT CoE SDO.....                          | 2-986  |
| Write File.....                                      | 2-1459 |
| Write IO-Link Device Object.....                     | 2-1071 |
| Write NX Unit Object.....                            | 2-1037 |
| Write Variable Class 3 Explicit.....                 | 2-1103 |
| Write Variable to File.....                          | 2-1430 |
| Write Variable UCMM Explicit.....                    | 2-1123 |
| WriteNbit_** (N-bit Write Group).....                | 2-1542 |

## X

---

|                                   |       |
|-----------------------------------|-------|
| XOR (Logical Exclusive OR).....   | 2-340 |
| XORN (Logical Exclusive NOR)..... | 2-343 |

## Z

---

|                                |       |
|--------------------------------|-------|
| Zone (Dead Zone Control).....  | 2-361 |
| Zone Comparison.....           | 2-120 |
| ZoneCmp (Zone Comparison)..... | 2-120 |



**OMRON Corporation** Industrial Automation Company  
Kyoto, JAPAN

Contact: [www.ia.omron.com](http://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 2011-2020 All Rights Reserved.  
In the interest of product improvement,  
specifications are subject to change without notice.

**Cat. No. W502-E1-30**

0820