OMRON

産業用PCプラットフォーム
NYシリーズ

コマンドリファレンスマニュアル
基本編

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

SYSMAC
always in control

SBCA-437J

おことわり

(1) 本マニュアルの内容の一部または全部を無断で複写、複製、転載することを禁じます。

(2) 本マニュアルの内容に関しては、改良のため予告なしに仕様などを変更することがあります。
あらかじめご了承ください。

(3) 本マニュアルの内容に関しては、万全を期しておりますが、万一ご不審な点や誤りなど、お気づきのことがありましたら、お手数ですが、弊社支店または営業所までご連絡ください。
その際、巻末記載の Man.No.（マニュアル No.）も併せてお知らせください。

商標

- Sysmac は、オムロン株式会社 FA 機器製品の日本およびその他の国における商標または登録商標です。
- Microsoft、Windows、Excel、Visual Basic、Microsoft Edge は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
- EtherCAT ® は、ドイツ Beckhoff Automation GmbH によりライセンスされた特許取得済み技術であり登録商標です。
- ODVA、CIP、CompoNet、DeviceNet、EtherNet/IP は ODVA の商標です。
- SD、SDHC ロゴは、SD-3C,LLC の商標です。
- Intel、Intel Core は、米国およびその他の国における Intel Corporation の商標です。

その他、本文中に掲載している会社名および製品名は、それぞれ各社の商標または登録商標です。

著作権

- スクリーンショットはマイクロソフトの許諾を得て使用しています。
- 本製品にはサードパーティ製のソフトウェアが組み込まれています。このソフトウェアに関連するライセンスと著作権については、http://www.fa.omron.co.jp/nj_info_j/ をご覧ください。

はじめに

このたびは、NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC をお買い上げいただきまして、ありがとうございます。

本マニュアルでは、対象となる製品の産業用パネル型 PC と産業用ボックス型 PC を総称して「NY シリーズ 産業用 PC」と記載する場合があります。また、NY シリーズ 産業用 PC のリアルタイム OS に組み込まれたコントローラ機能が直接制御する範囲を「コントローラ」と記載する場合があります。

本マニュアルは、NY シリーズ コントローラを使用する上で、必要な情報を記載しています。お使いになる前に本マニュアルをよく読んで、機能・性能などを十分ご理解いただき、システムの構築にご活用ください。
また、お読みになった後も本マニュアルは大切に保管して、いつも手元においてお使いください。

対象となる読者の方々

本マニュアルは、電気の知識（電気工事士あるいは同等の知識）を有する、次の方を対象に記述しています。
- FA 機器の導入を担当される方
- FA システムを設計される方
- FA 機器を設置、接続される方
- FA 現場を管理される方

また、プログラミング言語に関して、国際標準規格 IEC 61131-3 もしくは国内標準規格 JIS B 3503 に規定される内容をご理解いただいている方を対象に記述しています。

対象となる製品

本マニュアルは以下の製品を対象にしています。
- NY シリーズ IPC マシンコントローラ 産業用パネル型 PC
  - 形 NY532-15□□
  - 形 NY532-14□□
  - 形 NY532-13□□
  - 形 NY532-5400

- NY シリーズ IPC マシンコントローラ 産業用ボックス型 PC
  - 形 NY512-15□□
  - 形 NY512-14□□
  - 形 NY512-13□□

各製品の仕様の一部または制約事項が、他のマニュアルに記載されている場合があります。
「分冊構成 (P.2)」および「関連マニュアル (P.24)」をご確認ください。

分冊構成

本製品のマニュアルは、次表のとおりに分冊されています。目的に応じてお読みいただき、本製品を活用してください。
本製品の操作には、おもにオートメーションソフトウェア Sysmac Studio を使用します。Sysmac Studio については、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』 を参照してください。

使用目的	マニュアル										
	基本情報										
	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編	NY シリーズ IPC マシンコントローラ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル セットアップ編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編	NY シリーズ コマンドリファレンスマニュアル 基本編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル モーション制御編	NY シリーズ コマンドリファレンスマニュアル モーション編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート編	NJ/NY シリーズ NC 統合コントローラ ユーザーズマニュアル	NY シリーズ トラブルシューティングマニュアル
NY シリーズ パネル型 PC の概要を知る	○										
NY シリーズ ボックス型 PC の概要を知る		○									
装着、設置、ハードウェア設定を行う	○	○									
モーション制御を行う場合						○					
EtherCAT を使用する場合								○			
EtherNet/IP を使用する場合									○		
セットアップを行う *1			○								
初期設定を行う場合											
コントローラの使用準備を行う場合											
ソフトウェア設定を行う				○							
モーション制御を行う場合						○					
EtherCAT を使用する場合								○			
EtherNet/IP を使用する場合									○		
数値制御を行う場合										○	
ユーザプログラムを作成する				○	○						
モーション制御を行う場合						○	○				
EtherCAT を使用する場合								○			
EtherNet/IP を使用する場合									○		
数値制御を行う場合										○	
異常処理を組み込む場合											○
動作確認やデバッグを行う				○							
モーション制御を行う場合						○					
EtherCAT を使用する場合								○			
EtherNet/IP を使用する場合									○		
数値制御を行う場合										○	
異常管理機能やトラブル発生時の対処方法を知る *2										△	○

使用目的		マニュアル									
		基本情報									
		NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ハードウェア編	NY シリーズ IPC マシンコントローラ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル セットアップ編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編	NY シリーズ コマンドリファレンスマニュアル 基本編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル モーション制御編	NY シリーズ コマンドリファレンスマニュアル モーション編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート編	NY シリーズ IPC マシンコントローラ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート編	NJ/NY シリーズ NC 統合コントローラ ユーザーズマニュアル	NY シリーズ トラブルシューティングマニュアル
保守作業について知る		○	○									
	モーション制御を行う場合						○					
	EtherCAT を使用する場合								○			
	EtherNet/IP を使用する場合									○		

*1. Windows 側のセットアップ方法やユーティリティなどの使い方については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル セットアップ編(SBCA-441)』を参照してください。

*2. 異常管理の考え方と異常項目の概要については、『NY シリーズ トラブルシューティングマニュアル(SBCA-438)』を参照してください。ただし、△ マークがついているマニュアルの製品に特有の異常の詳細については、△ マークがついているマニュアルを参照してください。

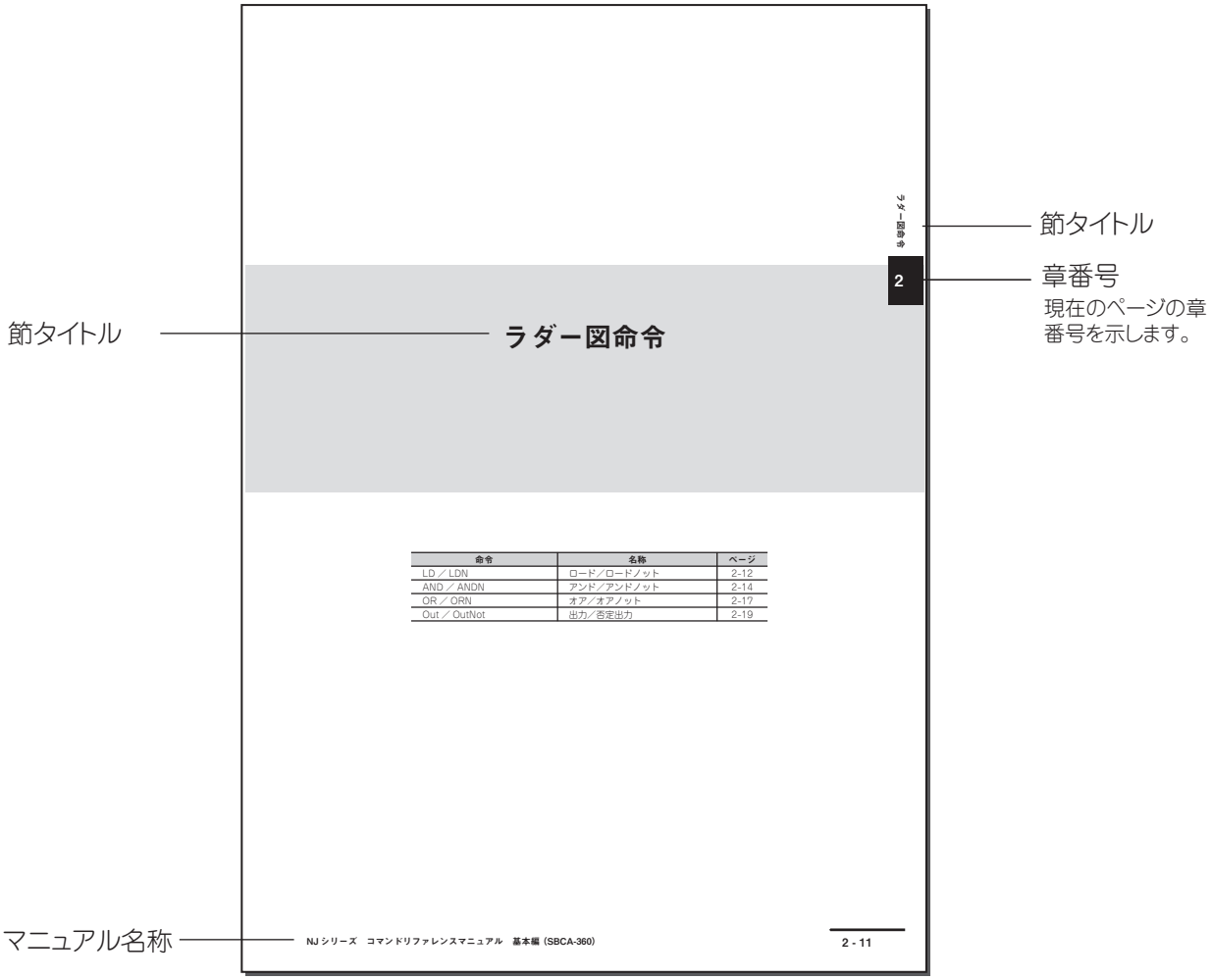
マニュアルの読み方

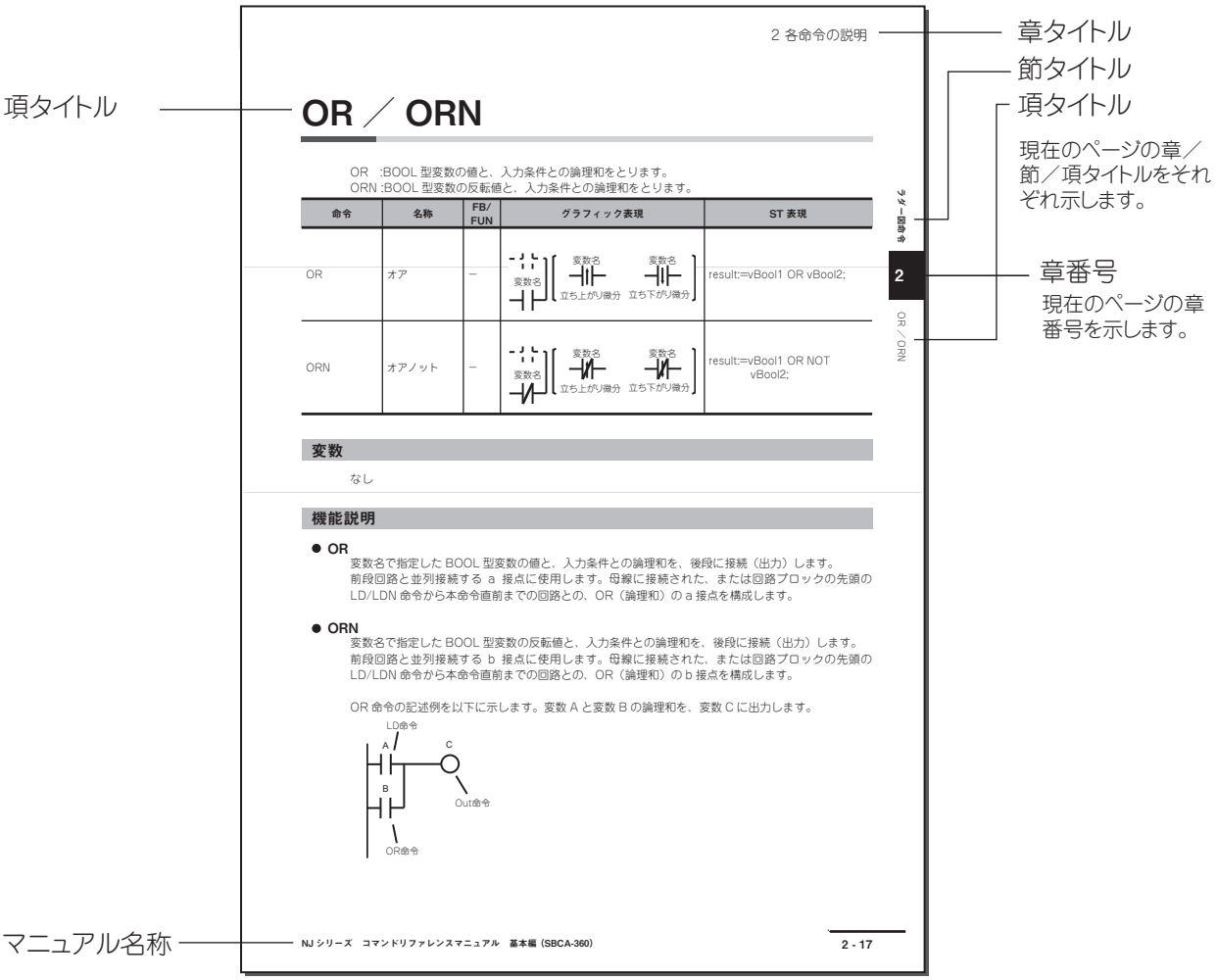
このマニュアルに記載している各命令は、NJ/NX シリーズと共通で書かれている部分があります。
そのため、以下の点に注意してください。

- NY シリーズ コントローラは、CJ ユニットを接続できません。各命令で、CJ ユニットについて記載している項目やサンプルプログラムについては、読み飛ばしてください。
- 各命令の説明で、「CPU ユニット」と記載しているところは、「NY シリーズ コントローラ」と読み替えてください。
- NY シリーズ コントローラには、SD メモリカードは装着できませんが、Windows の共有フォルダを使って、仮想 SD メモリカード機能を提供しています。そのため、「SD メモリカード」と記載しているところは、「仮想 SD メモリカード」と読み替えてください。
  仮想 SD メモリカード機能の詳細については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』、および『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル セットアップ編(SBCA-441)』を参照してください。

ページ構成

このマニュアルの各ページの構成は次のとおりです。





（注）各ページの構成は、説明用の見本です。実際の内容とは異なります。

アイコン

このマニュアルで使われているアイコンには、次のような意味があります。

安全上の要点

製品を安全に使用するために実施または回避すべきことを示します。

使用上の注意

製品の動作不能、誤動作、または性能・機能への悪影響を予防するために実施または回避すべきことを示します。

参考

必要に応じて読んでいただきたい項目です。
知っておくと便利な情報や、使用する上で参考となる内容について説明しています。

バージョン関連情報

コントローラや Sysmac Studio の、バージョンによる性能や機能の違いについて説明しています。

目次構成

1 命令一覧

2 各命令の説明

A 付録

I 索引

目次

はじめに **1**
対象となる読者の方々 1
対象となる製品 1

分冊構成 **2**

マニュアルの読み方 **4**
ページ構成 4
アイコン 5

目次構成 **7**

ご承諾事項 **16**

安全上の注意 **18**

安全上の要点 **19**

使用上の注意 **20**

法規と規格 **21**

バージョン **22**
バージョンの確認方法 22

関連マニュアル **24**

マニュアル改訂履歴 **27**

第 1 章　命令一覧

命令一覧 **1-2**
ラダー図命令 1-2
ST 構文命令 1-2
シーケンス入力命令 1-2
シーケンス出力命令 1-3
シーケンス制御命令 1-3
比較命令 1-3
タイマ命令 1-4
カウンタ命令 1-4
算術命令 1-5
BCD 変換命令 1-5
データ型変換命令 1-6
ビット列演算命令 1-7
選択命令 1-7
データ転送命令 1-8
シフト命令 1-8
データ変換命令 1-9
スタック／テーブル命令 1-10
FCS 命令 1-10
文字列命令 1-11
時間／時刻命令 1-11
アナログ制御命令 1-12
システム制御命令 1-13

プログラム制御命令 1-13
EtherCAT 通信命令 1-14
IO-Link 通信命令 1-14
EtherNet/IP 通信命令 1-14
シリアル通信命令 1-16
SD メモリカード命令 1-16
タイムスタンプ命令 1-17
OS 制御命令 1-17
その他の命令 1-17

第 2 章　各命令の説明

本章の見方 2-3
記載項目 2-3
共通変数 2-4
変数の有効範囲とデフォルトの初期値 2-8
派生データ型（列挙型、構造体、共用体） 2-9
配列指定 2-10
その他 2-11

ラダー図命令 2-13
LD / LDN 2-14
AND / ANDN 2-17
OR / ORN 2-20
Out / OutNot 2-23

ST 構文命令 2-27
IF 2-28
CASE 2-32
WHILE 2-36
REPEAT 2-38
EXIT 2-40
RETURN 2-43
FOR 2-44

シーケンス入力命令 2-45
R_TRIG, Up / F_TRIG, Down 2-46
TestABit / TestABitN 2-50

シーケンス出力命令 2-53
RS 2-54
SR 2-57
Set / Reset 2-60
SetBits / ResetBits 2-63
SetABit / ResetABit 2-65
OutABit 2-67

シーケンス制御命令 2-69
End 2-70
RETURN 2-71
MC / MCR 2-72
JMP 2-84
FOR / NEXT 2-86
BREAK 2-93

比較命令 2-95
EQ, = ... 2-96
NE, <> 2-98
LT,< / LE,<= / GT,> / GE,>= 2-100
EQascii 2-103
NEascii 2-105
LTascii / LEascii / GTascii / GEascii 2-107
Cmp 2-110
ZoneCmp 2-112
TableCmp 2-114
AryCmpEQ / AryCmpNE 2-117

AryCmpLT / AryCmpLE / AryCmpGT / AryCmpGE 2-119
AryCmpEQV / AryCmpNEV 2-121
AryCmpLTV / AryCmpLEV / AryCmpGTV / AryCmpGEV 2-123

タイマ命令 2-125
TON 2-126
TOF 2-131
TP 2-134
AccumulationTimer 2-137
Timer 2-141

カウンタ命令 2-145
CTD 2-146
CTD_** 2-148
CTU 2-151
CTU_** 2-153
CTUD 2-156
CTUD_** 2-161

算術命令 2-167
ADD, + ... 2-169
AddOU, +OU 2-173
SUB, - 2-176
SubOU, -OU 2-179
MUL, * 2-183
MulOU, *OU 2-186
DIV, / 2-190
MOD 2-193
ABS 2-195
RadToDeg / DegToRad 2-197
SIN / COS / TAN 2-199
ASIN / ACOS / ATAN 2-202
SQRT 2-205
LN / LOG 2-207
EXP 2-211
EXPT, ** 2-213
Inc / Dec 2-217
Rand 2-219
AryAdd 2-221
AryAddV 2-223
ArySub 2-225
ArySubV 2-227
AryMean 2-229
ArySD 2-231
ModReal 2-233
Fraction 2-235
CheckReal 2-237

BCD 変換命令 2-241
_BCD_TO_* 2-242
_TO_BCD_* 2-245
BCD_TO_** 2-248
BCDsToBin 2-251
BinToBCDs_** 2-254
AryToBCD 2-257
AryToBin 2-259

データ型変換命令 2-261
_TO_*（整数→整数変換グループ） 2-262
_TO_*（整数→ビット列変換グループ） 2-265
_TO_*（整数→実数変換グループ） 2-268
_TO_*（ビット列→整数変換グループ） 2-270
_TO_*（ビット列→ビット列変換グループ） 2-273
_TO_*（ビット列→実数変換グループ） 2-275
_TO_*（実数→整数変換グループ） 2-277
_TO_*（実数→ビット列変換グループ） 2-280
_TO_*（実数→実数変換グループ） 2-282

**_TO_STRING（整数→文字列変換グループ） 2-284
**_TO_STRING（ビット列→文字列変換グループ） 2-286
**_TO_STRING（実数→文字列変換グループ） 2-288
RealToFormatString 2-290
LrealToFormatString 2-296
STRING_TO_**（文字列→整数変換グループ） 2-302
STRING_TO_**（文字列→ビット列変換グループ） 2-304
STRING_TO_**（文字列→実数変換グループ） 2-306
TO_**（整数変換グループ） 2-309
TO_**（ビット列変換グループ） 2-311
TO_**（実数変換グループ） 2-313
EnumToNum 2-315
NumToEnum 2-317
TRUNC / Round / RoundUp 2-319

ビット列演算命令 2-323
AND, & / OR / XOR 2-324
XORN 2-327
NOT 2-329
AryAnd / AryOr / AryXor / AryXorN 2-331

選択命令 2-335
SEL 2-336
MUX 2-338
LIMIT 2-341
Band 2-343
Zone 2-345
MAX / MIN 2-347
AryMax / AryMin 2-349
ArySearch 2-352

データ転送命令 2-355
MOVE 2-356
MoveBit 2-359
MoveDigit 2-361
TransBits 2-363
MemCopy 2-365
SetBlock 2-367
Exchange 2-369
AryExchange 2-371
AryMove 2-373
Clear 2-375
Copy**ToNum（ビット列→符号付き整数） 2-377
Copy**To***（ビット列→実数） 2-379
CopyNumTo**（符号付き整数→ビット列） 2-381
CopyNumTo**（符号付き整数→実数） 2-383
Copy**To***（実数→ビット列） 2-385
Copy**ToNum（実数→符号付き整数） 2-386

シフト命令 2-389
AryShiftReg 2-390
AryShiftRegLR 2-392
ArySHL / ArySHR 2-395
SHL / SHR 2-398
NSHLC / NSHRC 2-400
ROL / ROR 2-402

データ変換命令 2-405
Swap 2-407
Neg 2-408
Decoder 2-410
Encoder 2-413
BitCnt 2-415
ColmToLine_** 2-416
LineToColm 2-418
Gray 2-420
UTF8ToSJIS 2-425

SJISToUTF8 2-427
PWLApprox / PWLApproxNoLineChk 2-429
PWLLineChk 2-434
MovingAverage 2-437
DispartReal 2-443
UniteReal 2-446
NumToDecString / NumToHexString 2-448
HexStringToNum_** 2-451
FixNumToString 2-453
StringToFixNum 2-455
DtToString 2-458
DateToString 2-460
TodToString 2-461
GrayToBin_** / BinToGray_** 2-463
StringToAry 2-466
AryToString 2-468
DispartDigit 2-470
UniteDigit_** 2-472
Dispart8Bit 2-474
Unite8Bit_** 2-476
ToAryByte 2-478
AryByteTo 2-484
SizeOfAry 2-489
PackWord 2-491
PackDword 2-493
LOWER_BOUND / UPPER_BOUND 2-495

スタック／テーブル命令 2-499
StackPush 2-500
StackFIFO / StackLIFO 2-509
StackIns 2-512
StackDel 2-514
RecSearch 2-516
RecRangeSearch 2-520
RecSort 2-525
RecNum 2-531
RecMax / RecMin 2-533

FCS 命令 2-537
StringSum 2-538
StringLRC 2-540
StringCRCCCITT 2-542
StringCRC16 2-544
AryLRC_** 2-546
AryCRCCCITT 2-548
AryCRC16 2-550

文字列命令 2-553
CONCAT 2-554
LEFT / RIGHT 2-556
MID 2-558
FIND 2-560
LEN 2-562
REPLACE 2-563
DELETE 2-565
INSERT 2-567
GetByteLen 2-569
ClearString 2-570
ToUCase / ToLCase 2-571
TrimL / TrimR 2-573
AddDelimiter 2-575
SubDelimiter 2-586

時間／時刻命令 2-597
ADD_TIME 2-599
ADD_TOD_TIME 2-601
ADD_DT_TIME 2-603

SUB_TIME 2-605
SUB_TOD_TIME 2-607
SUB_TOD_TOD 2-609
SUB_DATE_DATE 2-610
SUB_DT_DT 2-611
SUB_DT_TIME 2-613
MULTIME 2-615
DIVTIME 2-617
CONCAT_DATE_TOD 2-619
DT_TO_TOD 2-621
DT_TO_DATE 2-623
GetTime 2-625
DtToSec 2-627
DateToSec 2-629
TodToSec 2-631
SecToDt 2-633
SecToDate 2-635
SecToTod 2-637
TimeToNanoSec 2-639
TimeToSec 2-640
NanoSecToTime 2-642
SecToTime 2-643
ChkLeapYear 2-645
GetDaysOfMonth 2-646
DaysToMonth 2-648
GetDayOfWeek 2-650
GetWeekOfYear 2-652
DtToDateStruct 2-654
DateStructToDt 2-656
TruncTime 2-658
TruncDt 2-662
TruncTod 2-666

アナログ制御命令 2-671
PIDAT 2-672
PIDAT_HeatCool 2-699
TimeProportionalOut 2-733
LimitAlarm_** 2-750
LimitAlarmDv_** 2-754
LimitAlarmDvStbySeq_** 2-758
ScaleTrans 2-774
AC_StepProgram 2-777

システム制御命令 2-801
TraceSamp 2-802
TraceTrig 2-805
GetTraceStatus 2-808
SetAlarm 2-811
ResetAlarm 2-816
GetAlarm 2-818
ResetPLCError 2-820
GetPLCError 2-823
GetEIPError 2-825
ResetMCError 2-827
GetMCError 2-832
ResetECError 2-834
GetECError 2-836
SetInfo 2-838
RestartNXUnit 2-840
NX_ChangeWriteMode 2-845
NX_SaveParam 2-850
NX_ReadTotalPowerOnTime 2-855

プログラム制御命令 2-863
PrgStart 2-864
PrgStop 2-872

PrgStatus 2-889

EtherCAT 通信命令 2-895
EC_CoESDOWrite 2-896
EC_CoESDORead 2-899
EC_StartMon 2-904
EC_StopMon 2-910
EC_SaveMon 2-912
EC_CopyMon 2-914
EC_DisconnectSlave 2-916
EC_ConnectSlave 2-923
EC_ChangeEnableSetting 2-925
EC_GetMasterStatistics 2-943
EC_ClearMasterStatistics 2-946
EC_GetSlaveStatistics 2-948
EC_ClearSlaveStatistics 2-951
NX_WriteObj 2-953
NX_ReadObj 2-968

IO-Link 通信命令 2-975
IOL_ReadObj 2-976
IOL_WriteObj 2-984

EtherNet/IP 通信命令 2-993
CIPOpen 2-995
CIPOpenWithDataSize 2-1004
CIPRead 2-1007
CIPWrite 2-1012
CIPSend 2-1018
CIPClose 2-1023
CIPUCMMRead 2-1025
CIPUCMMWrite 2-1030
CIPUCMMSend 2-1036
SktUDPCreate 2-1047
SktUDPRcv 2-1055
SktUDPSend 2-1058
SktTCPAccept 2-1061
SktTCPConnect 2-1064
SktTCPRcv 2-1072
SktTCPSend 2-1075
SktGetTCPStatus 2-1078
SktClose 2-1081
SktClearBuf 2-1084
SktSetOption 2-1087
ChangeIPAdr 2-1092
ChangeFTPAccount 2-1100
FTPGetFileList 2-1104
FTPGetFile 2-1117
FTPPutFile 2-1125
FTPRemoveFile 2-1135
FTPRemoveDir 2-1144

シリアル通信命令 2-1149
NX_SerialSend 2-1150
NX_SerialRcv 2-1161
NX_ModbusRtuCmd 2-1174
NX_ModbusRtuRead 2-1184
NX_ModbusRtuWrite 2-1195
NX_SerialSigCtl 2-1206
NX_SerialBufClear 2-1213
NX_SerialStartMon 2-1222
NX_SerialStopMon 2-1226

SD メモリカード命令 2-1231
FileWriteVar 2-1232
FileReadVar 2-1237
FileOpen 2-1242

FileClose 2-1246
FileSeek 2-1249
FileRead 2-1252
FileWrite 2-1260
FileGets 2-1268
FilePuts 2-1275
FileCopy 2-1284
FileRemove 2-1291
FileRename 2-1295
DirCreate 2-1300
DirRemove 2-1303
BackupToMemoryCard 2-1306

タイムスタンプ命令 2-1319
NX_DOutTimeStamp 2-1320
NX_AryDOutTimeStamp 2-1325

OS 制御命令 2-1333
IPC_GetOSStatus 2-1334
IPC_RebootOS 2-1337
IPC_Shutdown 2-1340

その他の命令 2-1343
ReadNbit_** 2-1344
WriteNbit_** 2-1346
ChkRange 2-1348
GetMyTaskStatus 2-1350
GetMyTaskInterval 2-1353
Task_IsActive 2-1355
Lock / Unlock 2-1357
ActEventTask 2-1363
Get**Clk 2-1370
Get**Cnt 2-1371

付録

A-1 ErrorID で確認できるエラーコード一覧 A-2
A-2 イベントタスクで使用できない命令 A-20
A-3 NX メッセージ通信異常に関連する命令 A-22
A-4 SDO アボートコード一覧 A-23
A-5 バージョン関連情報 A-24

索引

ご承諾事項

「当社商品」について特別の合意がない場合には、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。

定義

本ご承諾事項中の用語の定義は次のとおりです。

- 「当社商品」:「当社」の FA システム機器、汎用制御機器、センシング機器、電子・機構部品
- 「カタログ等」:「当社商品」に関する、ベスト制御機器オムロン、電子・機構部品総合カタログ、その他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含みます。
- 「利用条件等」:「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取り扱い方法、利用上の注意、禁止事項その他
- 「お客様用途」:「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み、または利用を含みます。
- 「適合性等」:「お客様用途」での「当社商品」の(a)適合性、(b)動作、(c)第三者の知的財産の非侵害、(d)法令の遵守および(e)各種規格の遵守

記載事項のご注意

「カタログ等」の記載内容については次の点をご理解ください。

- 定格値および性能値は、単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。
- 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。
- 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。
- 「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

ご利用にあたってのご注意

ご採用およびご利用に際しては次の点をご理解ください。

- 定格・性能ほか「利用条件等」を遵守しご利用ください。
- お客様ご自身にて「適合性等」をご確認いただき、「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」を一切保証いたしかねます。
- 「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で、必ず事前に確認してください。
- 「当社商品」をご使用の際には、(i)定格および性能に対し余裕のある「当社商品」のご利用、冗長設計などの安全設計、(ii)「当社商品」が故障しても、「お客様用途」の危険を最小にする安全設計、(iii)利用者に危険を知らせるための、安全対策のシステム全体としての構築、(iv)「当社商品」および「お客様用途」の定期的な保守、の各事項を実施してください。
- 「当社」は DDoS 攻撃（分散型 DoS 攻撃）、コンピュータウイルスその他の技術的な有害プログラム、不正アクセスにより、「当社商品」、インストールされたソフトウェア、またはすべてのコンピュータ機器、コンピュータプログラム、ネットワーク、データベースが感染したとしても、そのことにより直接または間接的に生じた損失、損害その他の費用について一切責任を負わないものとします。

お客様ご自身にて、①アンチウイルス保護、②データ入出力、③紛失データの復元、④「当社商品」またはインストールされたソフトウェアに対するコンピュータウイルス感染防止、⑤「当社商品」に対する不正アクセス防止についての十分な措置を講じてください。

- 「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。ただし、次に掲げる用途であっても「当社」の意図した特別な商品用途の場合や特別の合意がある場合は除きます。
  a）高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、娯楽設備、医用機器、安全装置、その他生命・身体に危険が及びうる用途）
  b）高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24 時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
  c）厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
  d)「カタログ等」に記載のない条件や環境での用途
- 上記の(a)から(d)に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないで下さい。自動車搭載用商品については当社営業担当者にご相談ください。

保証条件

「当社商品」の保証条件は次のとおりです。

- 保証期間　ご購入後 1 年間といたします。
  （ただし「カタログ等」に別途記載がある場合を除きます。）
- 保証内容　故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
  a）当社保守サービス拠点における故障した「当社商品」の無償修理
  （ただし、電子・機構部品については、修理対応は行いません。）
  b）故障した「当社商品」と同数の代替品の無償提供
- 保証対象外　故障の原因が次のいずれかに該当する場合は、保証いたしません。
  a)「当社商品」本来の使い方以外のご利用
  b)「利用条件等」から外れたご利用
  c）本ご承諾事項「ご利用にあたってのご注意」に反するご利用
  d)「当社」以外による改造、修理による場合
  e)「当社」以外の者によるソフトウェアプログラムによる場合
  f)「当社」からの出荷時の科学・技術の水準では予見できなかった原因
  g）上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

責任の制限

本ご承諾事項に記載の保証が、「当社商品」に関する保証のすべてです。
「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

輸出管理

「当社商品」または技術資料を、輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。お客様が法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。

安全上の注意

安全上のご注意の内容については、以下のマニュアルを参照してください。
- 『NY シリーズ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編(SBCA-434)』
- 『NY シリーズ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編(SBCA-435)』
- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』

安全上の要点

安全上の要点の内容については、以下のマニュアルを参照してください。

- 『NY シリーズ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編(SBCA-434)』
- 『NY シリーズ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編(SBCA-435)』
- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』

使用上の注意

使用上の注意の内容については、以下のマニュアルを参照してください。
- 『NY シリーズ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編(SBCA-434)』
- 『NY シリーズ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編(SBCA-435)』
- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』

法規と規格

法規と規格の内容については、以下のマニュアルを参照してください。
- 『NY シリーズ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編(SBCA-434)』
- 『NY シリーズ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編(SBCA-435)』

バージョン

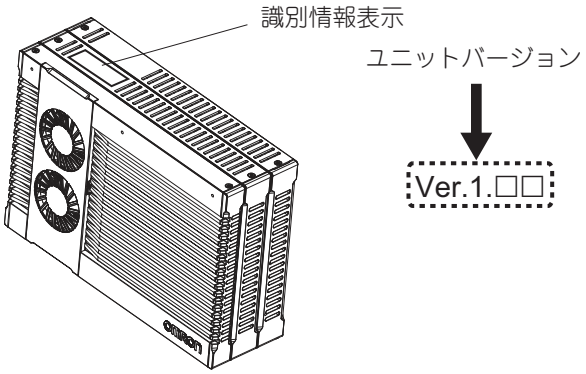
NY シリーズおよび各 EtherCAT スレーブは、ハードウェアやソフトウェアが、ハードウェアリビジョンやユニットバージョンという改変番号でバージョン管理されています。ハードウェアまたはソフトウェアに仕様変更があるごとに、ハードウェアリビジョンまたはユニットバージョンが更新されます。したがって、同じ形式のユニットや EtherCAT スレーブでも、ハードウェアリビジョンまたはユニットバージョンが異なると搭載機能や性能が異なります。

バージョンの確認方法

バージョンは、識別情報表示または Sysmac Studio で確認できます。

識別情報表示による確認

製品側面の識別情報表示で、バージョンが確認できます。
NY シリーズコントローラ 形 NY5□2-□□□□の識別情報表示は、下図のようになります。



Sysmac Studio による確認方法

Sysmac Studio で、バージョンが確認できます。ユニットと EtherCAT スレーブでは、確認方法が異なります。

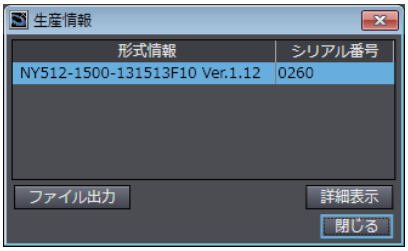
● NY シリーズコントローラでのバージョン確認方法

ユニットのバージョンは、オンライン状態の［生産情報］で確認できます。ただし、バージョンが確認できるユニットは、コントローラ機能だけです。

1 **マルチビューエクスプローラで［構成・設定］–［CPU・増設ラック］–［CPU ラック］を右クリックし、［生産情報表示］を選択します。**
［生産情報］ダイアログボックスが表示されます。

● 生産情報の表示内容の切り替え

1 **［生産情報］ダイアログボックスの右下の、［簡易表示］または［詳細表示］を選択します。**
［生産情報］の簡易表示と詳細表示が、切り替わります。





簡易表示　　詳細表示

表示内容は、簡易表示と詳細表示とで異なります。詳細表示では、ユニットバージョン、ハードウェアリビジョン、および各種のバージョンが表示されます。簡易表示では、ユニットバージョンだけが表示されます。

● EtherCAT スレーブのバージョン確認方法

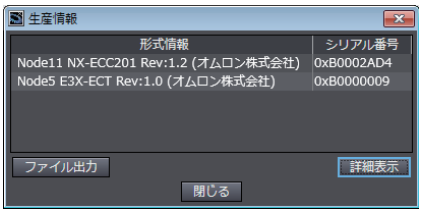
EtherCAT スレーブのバージョンは、オンライン状態の［生産情報］で確認することができます。確認方法は、以下のとおりです。

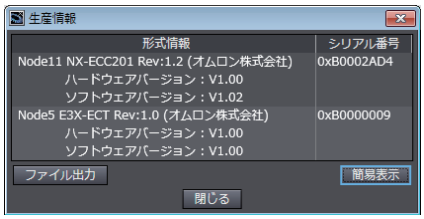
1 **マルチビューエクスプローラで［構成・設定］の［EtherCAT］をダブルクリックします。または、［構成・設定］の［EtherCAT］を右クリックして［編集］を選択します。**
EtherCAT 構成編集画面が表示されます。

2 **EtherCAT 構成編集画面でマスタを右クリックして［生産情報表示］を選択します。**
生産情報ダイアログボックスが表示されます。
ユニットバージョンは、「Rev」を付けて表示されます。

● 生産情報の表示内容の切り替え

1 **［生産情報］ダイアログボックスの右下の、［簡易表示］または［詳細表示］を選択します。**
［生産情報］の簡易表示と詳細表示が、切り替わります。





簡易表示　　詳細表示

関連マニュアル

本マニュアルに関連するマニュアルは、下表のとおりです。あわせてご覧ください。

マニュアル名称	Man.No.	形式	用途	内容
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ユーザーズマニュアル ハードウェア編	SBCA-435	形 NY532-□□□□	NY シリーズ産業用パネル型 PC の概要／設計／取付／保守などの基本的な仕様について知りたいとき。 おもにハードウェアに関する情報。	NY シリーズのシステム全体概要、および産業用パネル型 PC に関して、以下の内容を説明します。 • 特長やシステム構成 • 概要 • 各部の名称と機能 • 一般仕様 • 設置と配線 • 保守点検
NY シリーズ IPC マシンコントローラ 産業用ボックス型 PC ユーザーズマニュアル ハードウェア編	SBCA-434	形 NY512-□□□□	NY シリーズ産業用ボックス型 PC の概要／設計／取付／保守などの基本的な仕様について知りたいとき。 おもにハードウェアに関する情報。	NY シリーズのシステム全体概要、および産業用ボックス型 PC に関して、以下の内容を説明します。 • 特長やシステム構成 • 概要 • 各部の名称と機能 • 一般仕様 • 設置と配線 • 保守点検
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ／産業用ボックス型 PC ユーザーズマニュアル セットアップ編	SBCA-441	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC の初期設定やコントローラの使用準備について知りたいとき。	NY シリーズのシステム全体概要に関して、以下の内容を説明します。 • 2 つの OS のしくみ • 初期設定 • Industrial PC Support Utility • NYCompolet • Industrial PC API • バックアップとリカバリ
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編	SBCA-436	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC のコントローラ機能のプログラミング／システムの立ち上げについて知りたいとき。	NY シリーズのコントローラ機能に関して、以下の内容を説明します。 • コントローラの動作 • コントローラの機能 • コントローラの設定 • IEC 61131-3 ベースの言語仕様とプログラミング
NY シリーズ コマンドリファレンス マニュアル 基本編	SBCA-437	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC の基本命令仕様の詳細について知りたいとき。	各命令（IEC 61131-3 仕様）の詳細を説明します。
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ／産業用ボックス型 PC ユーザーズマニュアル モーション制御編	SBCE-379	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC のモーション制御の設定やプログラミングの考え方について知りたいとき。	モーション制御のためのコントローラの設定や動作、プログラミングの考え方について説明します。
NY シリーズ コマンドリファレンス マニュアル モーション編	SBCE-380	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC のモーション命令仕様の詳細について知りたいとき。	各モーション制御命令の詳細を説明します。
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT® ポート編	SBCD-368	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC の内蔵 EtherCAT ポートを使用するとき。	内蔵 EtherCAT ポートに関して説明します。 概要、構成、機能、セットアップについて記述しています。

マニュアル名称	Man.No.	形式	用途	内容
NY シリーズ IPC マシンコントローラ 産業用パネル型 PC ／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP™ポート編	SBCD-369	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC の内蔵 EtherNet/IP ポートを使用するとき。	内蔵 EtherNet/IP ポートに関して説明します。 基本設定、タグデータリンク、その他の機能について記述しています。
NJ/NY シリーズ NC 統合コントローラ ユーザーズマニュアル	SBCE-428	形 NJ501-5300 形 NY532-5400	NJ/NY シリーズで数値制御を行うとき。	数値制御機能について説明します。
NJ/NY シリーズ コマンドリファレンスマニュアル G コード編	SBCE-429	形 NJ501-5300 形 NY532-5400	G コード／M コード命令仕様の詳細について知りたいとき。	G コード／M コード命令の詳細を説明します。
NY シリーズ トラブルシューティングマニュアル	SBCA-438	形 NY532-□□□□ 形 NY512-□□□□	NY シリーズ産業用 PC で検出する異常の詳細について知りたいとき。	NY シリーズ システムにて検出する異常管理の考え方と各異常項目について説明します。
Sysmac Studio Version 1 オペレーションマニュアル	SBCA-470	形 SYSMAC -SE2□□□	Sysmac Studio の操作方法、機能について知りたいとき。	Sysmac Studio の操作方法について説明します。
CNC Operator オペレーションマニュアル	SBCE-430	形 SYSMAC- RTNC0□□□D	CNC Operator の概要、使用方法について知りたいとき。	CNC Operator の概要、インストール方法、基本操作、接続操作、主な機能の操作方法などについて説明します。
NX シリーズ EtherCAT®カプラユニット ユーザーズマニュアル	SBCD-361	形 NX-ECC□□□	NX シリーズ EtherCAT カプラユニット、および EtherCAT スレーブターミナルの使用方法について知りたいとき。	NX シリーズ EtherCAT カプラユニットと NX ユニットで構成される EtherCAT スレーブターミナルのシステム概要や構成方法、および EtherCAT を介して NX ユニットを設定、制御、モニタするための、EtherCAT カプラユニットのハードウェアや設定方法、機能について説明します。
NX シリーズ NX ユニット ユーザーズマニュアル	SBCA-407	形 NX-ID□□□□ 形 NX-IA□□□□ 形 NX-OC□□□□ 形 NX-OD□□□□ 形 NX-MD□□□□	NX ユニットの使用方法について知りたいとき。	NX ユニットのハードウェアや設定方法、機能について説明します。 以下のユニットのマニュアルがあります。 デジタル I/O ユニット、アナログ I/O ユニット、システムユニット、位置インタフェースユニット、コミュニケーションインタフェースユニット、ロードセル入力ユニット、IO-Link マスタユニット
	SBCA-408	形 NX-AD□□□□ 形 NX-DA□□□□		
	SBCA-461	形 NX-HAD□□□		
	SBCA-440	形 NX-TS□□□□ 形 NX-HB□□□□		
	SBCA-409	形 NX-PD1□□□ 形 NX-PF0□□□ 形 NX-PC0□□□ 形 NX-TBX01		
	SBCE-374	形 NX-EC0□□□ 形 NX-ECS□□□ 形 NX-PG0□□□		
	SBCA-422	形 NX-CIF□□□		
	SBCA-439	形 NX-RS□□□□		
	SBCD-370	形 NX-ILM□□□		
NX シリーズ データリファレンスマニュアル	SBCA-410	形 NX-□□□□□□	NX シリーズの各ユニットのシステム構成に必要となるデータを一覧で閲覧したいとき。	NX シリーズの各ユニットの「消費電力」、「質量」など、システム構築に必要となるデータを集めて記載しています。

マニュアル名称	Man.No.	形式	用途	内容
EtherCAT リモート I/O ターミナル GX シリーズ EtherCAT スレーブ ユーザーズマニュアル	SBCD-350	形 GX-ID□□□□ 形 GX-OD□□□□ 形 GX-OC□□□□ 形 GX-MD□□□□ 形 GX-AD□□□□ 形 GX-DA□□□□ 形 GX-EC□□□□ 形 XWT-ID□□ 形 XWT-OD□□	EtherCAT リモート I/O ターミナルの使用方法について知りたいとき。	EtherCAT リモート I/O ターミナルのハードウェアや設定方法、機能について説明します。
AC サーボモータ／ドライバ 1S シリーズ EtherCAT®通信内蔵タイプ ユーザーズマニュアル	SBCE-377	形 R88M-1□ 形 R88D-1SN□-ECT	EtherCAT 通信内蔵タイプのサーボモータ／ドライバの使用方法について知りたいとき。	EtherCAT 通信内蔵タイプのサーボモータ／ドライバのハードウェアや設定方法、機能について説明します。
	SBCE-438	形 R88M-1AL□/-1AM□ 形 R88D-1SAN□-ECT		
AC サーボモータ／ドライバ G5 シリーズ EtherCAT®通信内蔵タイプ ユーザーズマニュアル	SBCE-365	形 R88M-K□ 形 R88D-KN□-ECT	EtherCAT 通信内蔵タイプのサーボモータ／ドライバの使用方法について知りたいとき。	EtherCAT 通信内蔵タイプのサーボモータ／ドライバのハードウェアや設定方法、機能について説明します。 G5 シリーズには、リニアモータタイプや位置制御限定モデルがあります。
	SBCE-366	形 R88L-EC-□ 形 R88D-KN□-ECT-L		

マニュアル改訂履歴

マニュアル改訂記号は、表表紙・裏表紙に記載されている Man.No.の後尾に付記されます。



改訂記号	改訂年月	改訂理由・改訂ページ
A	2016 年 9 月	初版
B	2016 年 11 月	誤記訂正による改訂
C	2017 年 4 月	誤記訂正による改訂
D	2017 年 10 月	• CPU ユニット Ver.1.16、Sysmac Studio Ver.1.20 対応による改訂 • 誤記訂正
E	2018 年 4 月	• CPU ユニット Ver.1.18、Sysmac Studio Ver.1.22 対応による改訂 • 誤記訂正
F	2019 年 1 月	誤記訂正による改訂
G	2019 年 7 月	• ユニットバージョン Ver.1.21 へのバージョンアップに伴う改訂 • 誤記訂正
H	2020 年 7 月	誤記訂正による改訂
J	2024 年 7 月	誤記訂正による改訂

1

命令一覧

この章では、NY シリーズ コントローラで使用できる命令の一覧を示します。

命令一覧 **1-2**

命令一覧

- モーション制御命令の仕様は、『NY シリーズ コマンドリファレンスマニュアル モーション編 (SBCE-380)』を参照してください。
- シミュレーション命令の仕様は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。

ラダー図命令

命令	名称	機能概要	ページ
LD	ロード	BOOL 型変数の値を読み込みます。	P.2-14
LDN	ロードノット	BOOL 型変数の反転値を読み込みます。	P.2-14
AND	アンド	BOOL 型変数の値と、入力値との論理積をとります。	P.2-17
ANDN	アンドノット	BOOL 型変数の反転値と、入力値との論理積をとります。	P.2-17
OR	オア	BOOL 型変数の値と、入力条件との論理和をとります。	P.2-20
ORN	オアノット	BOOL 型変数の反転値と、入力条件との論理和をとります。	P.2-20
Out	出力	前段までの論理演算処理の結果を、BOOL 型変数に出力します。	P.2-23
OutNot	否定出力	前段までの論理演算処理の結果の反転値を、BOOL 型変数に出力します。	P.2-23

ST 構文命令

命令	名称	機能概要	ページ
IF	選択	指定した条件式の評価結果に応じて、二者択一的に、実行する文を選択します。	P.2-28
CASE	多方向分岐	指定した整数式の値に応じて、複数の文の中から、実行する文を選択します。	P.2-32
WHILE	繰り返し	指定した条件式の評価結果が TRUE のあいだ、ある文を繰り返し実行します。	P.2-36
REPEAT	繰り返し	ある文を一度実行し、指定した条件式が TRUE になるまで、その文を繰り返し実行します。	P.2-38
EXIT	ループブレイク	最も内側の、FOR 命令、WHILE 命令または REPEAT 命令の繰り返し処理を中断します。	P.2-40
RETURN	復帰	ファンクションまたはファンクションブロックを終了し、処理を呼出元に戻します。	P.2-43
FOR	繰り返し	繰り返しの条件を指定して、FOR 文から END_FOR 文までにある文を、繰り返し実行します。	P.2-44

シーケンス入力命令

命令	名称	機能概要	ページ
R_TRIG, Up	立ち上がり微分	入力信号の立ち上がり（FALSE→TRUE）時、1 タスク周期だけ TRUE を出力します。	P.2-46
F_TRIG, Down	立ち下がり微分	入力信号の立ち下がり（TRUE→FALSE）時、1 タスク周期だけ TRUE を出力します。	P.2-46
TestABit	ビットテスト	ビット列の、指定ビットの値を出力します。	P.2-50
TestABitN	ビットテスト否定	ビット列の、指定ビットの反転値を出力します。	P.2-50

シーケンス出力命令

命令	名称	機能概要	ページ
RS	リセット優先キープ	BOOL 型変数の値を保持します。 セット入力とリセット入力が同時に TRUE の場合、リセット入力が優先されます。	P.2-54
SR	セット優先キープ	BOOL 型変数の値を保持します。 セット入力とリセット入力が同時に TRUE の場合、セット入力が優先されます。	P.2-57
Set	セット	BOOL 型変数に TRUE をセットします	P.2-60
Reset	リセット	BOOL 型変数に FALSE をセットします。	P.2-60
SetBits	多ビットセット	ビット列データの連続複数ビットに、TRUE をセットします。	P.2-63
ResetBits	多ビットリセット	ビット列データの連続複数ビットに、FALSE をセットします。	P.2-63
SetABit	1 ビットセット	ビット列データの指定ビットに、TRUE をセットします。	P.2-65
ResetABit	1 ビットリセット	ビット列データの指定ビットに、FALSE をセットします。	P.2-65
OutABit	1 ビットアウト	ビット列データの指定ビットに、TRUE/FALSE をセットします。	P.2-67

シーケンス制御命令

命令	名称	機能概要	ページ
End	エンド	プログラムの、現在のタスク周期での実行を終了します。	P.2-70
RETURN	復帰	ファンクションまたはファンクションブロックを終了し、処理を呼出元に戻します。	P.2-71
MC	マスタコントロール開始	マスタコントロール領域の開始点を示し、マスタコントロールによるリセットを実行します。	P.2-72
MCR	マスタコントロール終了	マスタコントロール領域の終了点を示します。	P.2-72
JMP	ジャンプ	指定したジャンプ先に、処理が移ります。	P.2-84
FOR	繰り返し開始	繰り返し処理の開始位置を示すとともに、繰り返しの条件を指定します。	P.2-86
NEXT	繰り返し終了	繰り返し処理の終了位置を示します。	P.2-86
BREAK	ループブレイク	最も内側の FOR 命令から NEXT 命令までの繰り返し処理を中断します。	P.2-93

比較命令

命令	名称	機能概要	ページ
EQ, =	比較 EQ	複数のデータが、すべて等しいかどうかを判定します。	P.2-96
NE, <>	比較 NE	2 つのデータが、異なっているかどうかを判断します。	P.2-98
LT, <	比較 LT	複数の数値の大小比較(<)を行います。	P.2-100
LE, <=	比較 LE	複数の数値の大小比較(<=)を行います。	P.2-100
GT, >	比較 GT	複数の数値の大小比較(>)を行います。	P.2-100
GE, >=	比較 GE	複数の数値の大小比較(>=)を行います。	P.2-100
EQascii	文字列比較 EQ	複数の文字列が、すべて等しいかどうかを判定します。	P.2-103
NEascii	文字列比較 NE	2 つの文字列が、異なっているかどうかを判定します。	P.2-105
LTascii	文字列比較 LT	複数の文字列の大小比較(<)を行います。	P.2-107
LEascii	文字列比較 LE	複数の文字列の大小比較(<=)を行います。	P.2-107
GTascii	文字列比較 GT	複数の文字列の大小比較(>)を行います。	P.2-107
GEascii	文字列比較 GE	複数の文字列の大小比較(>=)を行います。	P.2-107
Cmp	比較	2 つの数値を比較します。	P.2-110

命令	名称	機能概要	ページ
ZoneCmp	領域比較	比較データが、指定した上限値と下限値の間にあるかどうかを判定します。	P.2-112
TableCmp	テーブル比較	比較データと、比較テーブルで指定された複数の定義区間とを比較します。	P.2-114
AryCmpEQ	配列一括比較 EQ	2 つの配列の各要素が、等しいかどうかを判定します。	P.2-117
AryCmpNE	配列一括比較 NE	2 つの配列の各要素が、異なるかどうかを判定します。	P.2-117
AryCmpLT	配列一括比較 LT	2 つの配列の各要素の大小比較（<）を行います。	P.2-119
AryCmpLE	配列一括比較 LE	2 つの配列の各要素の大小比較（<=）を行います。	P.2-119
AryCmpGT	配列一括比較 GT	2 つの配列の各要素の大小比較（>）を行います。	P.2-119
AryCmpGE	配列一括比較 GE	2 つの配列の各要素の大小比較（>=）を行います。	P.2-119
AryCmpEQV	配列要素比較 EQ	配列の各要素と数値が、等しいかどうかを判定します。	P.2-121
AryCmpNEV	配列要素比較 NE	配列の各要素と数値が、異なるかどうかを判定します。	P.2-121
AryCmpLTV	配列要素比較 LT	配列の各要素と数値との大小比較（<）を行います。	P.2-123
AryCmpLEV	配列要素比較 LE	配列の各要素と数値との大小比較（<=）を行います。	P.2-123
AryCmpGTV	配列要素比較 GT	配列の各要素と数値との大小比較（>）を行います。	P.2-123
AryCmpGEV	配列要素比較 GE	配列の各要素と数値との大小比較（>=）を行います。	P.2-123

タイマ命令

命令	名称	機能概要	ページ
TON	オンディレータイマ	起動から設定時間経過後に TRUE を出力するタイマです。	P.2-126
TOF	オフディレータイマ	起動から設定時間経過後に FALSE を出力するタイマです。	P.2-131
TP	パルス出力	起動から設定時間だけ TRUE を出力するタイマです。	P.2-134
AccumulationTimer	積算タイマ	タイマ入力が TRUE である時間を、積算するタイマです。	P.2-137
Timer	100ms タイマ	起動から設定時間経過後に TRUE を出力するタイマです。設定時間の最小単位は 100ms です。	P.2-141

カウンタ命令

命令	名称	機能概要	ページ
CTD	減算カウンタ	カウンタ入力信号が入力されるごとに、減算していくカウンタです。プリセット値、カウンタ値のデータ型は、INT です。	P.2-146
CTD_**	減算カウンタグループ	カウンタ入力信号が入力されるごとに、減算していくカウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。	P.2-148
CTU	加算カウンタ	カウンタ入力信号が入力されるごとに、加算していくカウンタです。プリセット値、カウンタ値のデータ型は、INT です。	P.2-151
CTU_**	加算カウンタグループ	カウンタ入力信号が入力されるごとに、加算していくカウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。	P.2-153
CTUD	可逆カウンタ	加算カウンタ入力および減算カウンタ入力に従い、加減算するカウンタです。プリセット値、カウンタ値のデータ型は、INT です。	P.2-156
CTUD_**	可逆カウンタグループ	加算カウンタ入力および減算カウンタ入力に従い、加減算するカウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。	P.2-161

算術命令

命令	名称	機能概要	ページ
ADD, +	加算	整数、実数を加算します。または、文字列を結合します。	P.2-169
AddOU, +OU	加算（オーバーフローチェック付き）	整数、実数を加算します。さらに、整数の加算結果に対して、オーバーフローのチェックを行います。	P.2-173
SUB, -	減算	整数、実数を減算します。	P.2-176
SubOU, -OU	減算（オーバーフローチェック付き）	整数、実数を減算します。さらに、整数の減算結果に対して、オーバーフローのチェックを行います。	P.2-179
MUL, *	乗算	整数、実数を乗算します。	P.2-183
MulOU, *OU	乗算（オーバーフローチェック付き）	整数、実数を乗算した結果を出力します。さらに、整数の乗算結果に対して、オーバーフローのチェックを行います。	P.2-186
DIV, /	除算	整数、実数を除算します。	P.2-190
MOD	剰余	整数を除算したときの剰余を求めます。	P.2-193
ABS	絶対値	整数、実数の絶対値を求めます。	P.2-195
RadToDeg	ラジアン→角度変換	実数をラジアン(rad)単位から度（°）単位に変換します。	P.2-197
DegToRad	角度→ラジアン変換	実数を度（°）単位からラジアン(rad)単位に変換します。	P.2-197
SIN	SIN 演算	実数の sin（正弦値）演算を行います。	P.2-199
COS	COS 演算	実数の cos（余弦値）演算を行います。	P.2-199
TAN	TAN 演算	実数の tan（正接値）演算を行います。	P.2-199
ASIN	SIN^{-1} 演算	実数の sin^{-1}（逆正弦値）演算を行います。	P.2-202
ACOS	COS^{-1} 演算	実数の cos^{-1}（逆余弦値）演算を行います。	P.2-202
ATAN	TAN^{-1} 演算	実数の tan^{-1}（逆正接値）演算を行います。	P.2-202
SQRT	平方根演算	実数の平方根（$\sqrt{}$）を求めます。	P.2-205
LN	自然対数演算	実数の自然対数を求めます。	P.2-207
LOG	常用対数演算	実数の常用対数を求めます。	P.2-207
EXP	自然指数演算	自然対数の底 e の指数関数を求めます。	P.2-211
EXPT, **	べき乗演算	2 つの実数のべき乗を求めます。	P.2-213
Inc	インクリメント	整数値をインクリメントします。	P.2-217
Dec	デクリメント	整数値をデクリメントします。	P.2-217
Rand	乱数発生	疑似乱数を発生します。	P.2-219
AryAdd	配列一括加算	2 つの配列の、要素ごとの加算を行います。	P.2-221
AryAddV	配列要素加算	配列の各要素に、同一数値を加算します。	P.2-223
ArySub	配列一括減算	2 つの配列の、要素ごとの減算を行います。	P.2-225
ArySubV	配列要素減算	配列の各要素から、同一数値を減算します。	P.2-227
AryMean	配列要素の平均値算出	配列要素の平均値を算出します。	P.2-229
ArySD	配列要素の標準偏差	配列要素の標準偏差を算出します。	P.2-231
ModReal	実数剰余	実数を除算したときの剰余を算出します。	P.2-233
Fraction	実数小数部抽出	実数の、小数部を抽出します。	P.2-235
CheckReal	実数チェック	実数が、無限大または非数であるかどうかを判定します。	P.2-237

BCD 変換命令

命令	名称	機能概要	ページ
_BCD_TO_*	BCD→符号なし整数変換グループ	BCD のビット列を、符号なし整数に変換します。	P.2-242
_TO_BCD_*	符号なし整数→BCD 変換グループ	符号なし整数を、BCD のビット列に変換します。	P.2-245
BCD_TO_**	BCD 群→符号なし整数変換グループ	BCD のビット列を、符号なし整数に変換します。	P.2-248

命令	名称	機能概要	ページ
BCDsToBin	符号付き BCD→符号付き整数変換	符号付き BCD のビット列を、符号付き整数に変換します。	P.2-251
BinToBCDs_**	符号付き整数→BCD 変換グループ	符号付き整数を、符号付き BCD のビット列に変換します。	P.2-254
AryToBCD	配列一括 BCD 変換	符号なし整数配列の各要素を、BCD のビット列に変換します。	P.2-257
AryToBin	配列一括符号なし整数変換	BCD のビット列配列の各要素を、符号なし整数に変換します。	P.2-259

データ型変換命令

命令	名称	機能概要	ページ
_TO_*（整数→整数変換グループ）	整数→整数変換グループ	整数を、異なるデータ型の整数に変換します。	P.2-262
_TO_*（整数→ビット列変換グループ）	整数→ビット列変換グループ	整数を、ビット列に変換します。	P.2-265
_TO_*（整数→実数変換グループ）	整数→実数変換グループ	整数を、実数に変換します。	P.2-268
_TO_*（ビット列→整数変換グループ）	ビット列→整数変換グループ	ビット列を、整数に変換します。	P.2-270
_TO_*（ビット列→ビット列変換グループ）	ビット列→ビット列変換グループ	ビット列を、異なるデータ型のビット列に変換します。	P.2-273
_TO_*（ビット列→実数変換グループ）	ビット列→実数変換グループ	ビット列を、実数に変換します。	P.2-275
_TO_*（実数→整数変換グループ）	実数→整数変換グループ	実数を、整数に変換します。	P.2-277
_TO_*（実数→ビット列変換グループ）	実数→ビット列変換グループ	実数を、ビット列に変換します。	P.2-280
_TO_*（実数→実数変換グループ）	実数→実数変換グループ	実数を、異なるデータ型の実数に変換します。	P.2-282
**_TO_STRING（整数→文字列変換グループ）	整数→文字列変換グループ	整数を、文字列に変換します。	P.2-284
**_TO_STRING（ビット列→文字列変換グループ）	ビット列→文字列変換グループ	ビット列を、文字列に変換します。	P.2-286
**_TO_STRING（実数→文字列変換グループ）	実数→文字列変換グループ	実数を、文字列に変換します。	P.2-288
RealToFormatString	実数（REAL）→文字列変換書式付き	REAL 型変数を、書式指定付きで、文字列に変換します。	P.2-290
LrealToFormatString	実数（LREAL）→文字列変換書式付き	LREAL 型変数を、書式指定付きで、文字列に変換します。	P.2-296
STRING_TO_**（文字列→整数変換グループ）	文字列→整数変換グループ	文字列を、整数に変換します。	P.2-302
STRING_TO_**（文字列→ビット列変換グループ）	文字列→ビット列変換グループ	文字列を、ビット列に変換します。	P.2-304

命令	名称	機能概要	ページ
STRING_TO_**（文字列→実数変換グループ）	文字列→実数変換グループ	文字列を、実数に変換します。	P.2-306
TO_**（整数変換グループ）	整数変換グループ	整数、ビット列、実数、文字列を、整数に変換します。	P.2-309
TO_**（ビット列変換グループ）	ビット列変換グループ	整数、ビット列、実数、文字列を、ビット列に変換します。	P.2-311
TO_**（実数変換グループ）	実数変換グループ	整数、ビット列、実数、文字列を、実数に変換します。	P.2-313
EnumToNum	列挙型→整数変換	列挙型を DINT 型に変換します。	P.2-315
NumToEnum	整数→列挙型変換	DINT 型を列挙型に変換します。	P.2-317
TRUNC	実数切り捨て	実数の小数第 1 位を切り捨てて、整数にします。	P.2-319
Round	実数丸め	実数の小数第 1 位を五捨五入して、整数にします。	P.2-319
RoundUp	実数切り上げ	実数の小数第 1 位を切り上げて、整数にします。	P.2-319

ビット列演算命令

命令	名称	機能概要	ページ
AND, &	論理積	複数のブール、ビット列の 1 ビットごとの論理積演算を行います。	P.2-324
OR	論理和	複数のブール、ビット列の 1 ビットごとの論理和演算を行います。	P.2-324
XOR	排他的論理和	複数のブール、ビット列の 1 ビットごとの排他的論理和演算を行います。	P.2-324
XORN	排他的論理和否定	複数のブール、ビット列の 1 ビットごとの排他的論理和否定演算を行います。	P.2-327
NOT	ビット反転	ブール、ビット列の各ビットを反転します。	P.2-329
AryAnd	配列論理積	配列間で、各要素のブール、ビット列の 1 ビットごとの論理積演算を行います。	P.2-331
AryOr	配列論理和	配列間で、各要素のブール、ビット列の 1 ビットごとの論理和演算を行います。	P.2-331
AryXor	配列排他的論理和	配列間で、各要素のブール、ビット列の 1 ビットごとの排他的論理和演算を行います。	P.2-331
AryXorN	配列排他的論理和否定	配列間で、各要素のブール、ビット列の 1 ビットごとの排他的論理和否定演算を行います。	P.2-331

選択命令

命令	名称	機能概要	ページ
SEL	ビット選択	2 個の選択肢のうちの 1 つを選択します。	P.2-336
MUX	マルチプレクサ	2～5 個の選択肢のうちの 1 つを選択します。	P.2-338
LIMIT	上下限リミット制限	設定された上下限値によって、入力変数の値を制限します。	P.2-341
Band	デッドバンド制御	デッドバンド（不感帯）制御します。	P.2-343
Zone	デッドゾーン制御	入力値にバイアス値を付加します。	P.2-345
MAX	最大値検索	2～5 個の数値のうちの、最大値を検索します。	P.2-347
MIN	最小値検索	2～5 個の数値のうちの、最小値を検索します。	P.2-347
AryMax	配列変数の最大値検索	1 次元配列要素の最大値を検索します。	P.2-349
AryMin	配列変数の最小値検索	1 次元配列要素の最小値を検索します。	P.2-349
ArySearch	配列検索	1 次元配列から、指定した値を検索します。	P.2-352

データ転送命令

命令	名称	機能概要	ページ
MOVE	転送	変数や定数の値を、別の変数に転送します。	P.2-356
MoveBit	ビット転送	ビット列内の 1 ビットを転送します。	P.2-359
MoveDigit	ディジット転送	ビット列内の複数けた（1 けたは 4 ビット）を転送します。	P.2-361
TransBits	多ビット転送	ビット列内の複数ビットを転送します。	P.2-363
MemCopy	メモリコピー	複数の配列要素を転送します。転送元と転送先は、同じデータ型に限られます。	P.2-365
SetBlock	ブロック設定	複数の配列要素に、変数や定数の値を転送します。	P.2-367
Exchange	データ交換	2 つの変数の値を交換します。	P.2-369
AryExchange	配列データ交換	2 つの配列間の要素を交換します。	P.2-371
AryMove	配列の転送	複数の配列要素を転送します。転送元と転送先のデータ型が、異なっていてもかまいません。	P.2-373
Clear	初期化	変数を初期化します。	P.2-375
Copy**ToNum（ビット列→符号付き整数）	ビットパターンコピー（ビット列→符号付き整数）グループ	ビット列の内容を、そのまま符号付き整数にコピーします。	P.2-377
Copy**To***（ビット列→実数）	ビットパターンコピー（ビット列→実数）グループ	ビット列の内容を、そのまま実数にコピーします。	P.2-379
CopyNumTo**（符号付き整数→ビット列）	ビットパターンコピー（符号付き整数→ビット列）グループ	符号付き整数の内容を、そのままビット列にコピーします。	P.2-381
CopyNumTo**（符号付き整数→実数）	ビットパターンコピー（符号付き整数→実数）グループ	符号付き整数の内容を、そのまま実数にコピーします。	P.2-383
Copy**To***（実数→ビット列）	ビットパターンコピー（実数→ビット列）グループ	実数の内容を、そのままビット列にコピーします。	P.2-385
Copy**ToNum（実数→符号付き整数）	ビットパターンコピー（実数→符号付き整数）グループ	実数の内容を、そのまま符号付き整数にコピーします。	P.2-386

シフト命令

命令	名称	機能概要	ページ
AryShiftReg	シフトレジスタ	配列要素からなるビット列全体を 1 ビット左シフトし、最下位ビットに入力値を挿入します。	P.2-390
AryShiftRegLR	左右シフトレジスタ	配列要素からなるビット列全体を左（または右）方向に 1 ビットシフトし、最下位（または最上位）ビットに入力値を挿入します。	P.2-392
ArySHL	配列の N 要素左シフト	配列要素を、複数要素分、左方向（上位方向）にシフトします。	P.2-395
ArySHR	配列の N 要素右シフト	配列要素を、複数要素分、右方向（下位方向）にシフトします。	P.2-395
SHL	N ビット左シフト	ビット列を複数ビット、左方向（上位方向）にシフトします。	P.2-398
SHR	N ビット右シフト	ビット列を複数ビット、右方向（下位方向）にシフトします。	P.2-398
NSHLC	N ビットデータ N ビット左シフト CY 付き	ビット列配列をキャリー(CY)フラグ付きで複数ビット、左方向（上位方向）にシフトします。	P.2-400

命令	名称	機能概要	ページ
NSHRC	N ビットデータ N ビット右シフト CY 付き	ビット列配列をキャリー(CY)フラグ付きで複数ビット、右方向（下位方向）にシフトします。	P.2-400
ROL	N ビット左回転	ビット列を複数ビット、左方向（上位方向）に回転します。	P.2-402
ROR	N ビット右回転	ビット列を複数ビット、右方向（下位方向）に回転します。	P.2-402

データ変換命令

命令	名称	機能概要	ページ
Swap	バイトスワップ	16 ビットの値の上位バイトと下位バイトを交換します。	P.2-407
Neg	符号反転	数値の符号を反転します。	P.2-408
Decoder	ビットデコーダ	最大 256 ビットからなる配列要素の、指定した 1 ビットを TRUE、その他のビットを FALSE にセットします。	P.2-410
Encoder	ビットエンコーダ	最大 256 ビットからなる配列要素のうちの、値が TRUE であるビットの位置を求めます。	P.2-413
BitCnt	ビットカウンタ	ビット列内の、値が TRUE のビットの総数をカウントします。	P.2-415
ColmToLine_**	ビット列→ビット行変換グループ	各配列要素の、指定した位置にあるビットの値を抽出し、ビット列として出力します。	P.2-416
LineToColm	ビット行→ビット列変換	ビット列を分解して、配列要素の指定したビット位置に出力します。	P.2-418
Gray	グレイコード変換	グレイコードを角度に変換します。	P.2-420
UTF8ToSJIS	文字コード変換（UTF-8→SJIS）	文字コード UTF-8 の文字列を、文字コード SJIS の BYTE 型配列に変換します。	P.2-425
SJISToUTF8	文字コード変換（SJIS→UTF-8）	文字コード SJIS の BYTE 型配列を、文字コード UTF-8 の文字列に変換します。	P.2-427
PWLApprox	折れ線近似変換（折れ線データチェックあり）	折れ線データが有効かどうかのチェックありで、整数、実数の折れ線近似計算を行います。	P.2-429
PWLApproxNoLineChk	折れ線近似変換（折れ線データチェックなし）	折れ線データが有効かどうかのチェックなしで、整数、実数の折れ線近似計算を行います。	P.2-429
PWLLineChk	折れ線データチェック	折れ線近似変換（折れ線データチェックなし）命令で使用する折れ線データが、X 座標の昇順に並んでいるかどうかを判定します。	P.2-434
MovingAverage	移動平均	移動平均値を算出します。	P.2-437
DispartReal	実数の仮数・指数分離	実数を、符号付き仮数部と、指数部に分離します。	P.2-443
UniteReal	仮数・指数の実数への結合	符号付き仮数部と、指数部を結合して実数にします。	P.2-446
NumToDecString	固定長 10 進文字列変換	整数を、固定長 10 進文字列表現に変換します。	P.2-448
NumToHexString	固定長 16 進文字列変換	整数を、固定長 16 進文字列表現に変換します。	P.2-448
HexStringToNum_**	16 進文字列→数値変換グループ	16 進文字列表現を、整数に変換します。	P.2-451
FixNumToString	固定小数点数→文字列変換	符号付き固定小数点数を、10 進文字列表現に変換します。	P.2-453
StringToFixNum	文字列→固定小数点数変換	10 進文字列を、符号付き固定小数点数表現に変換します。	P.2-455
DtToString	日時→文字列変換	日時を、文字列表現に変換します。	P.2-458
DateToString	日付→文字列変換	日付を、文字列表現に変換します。	P.2-460
TodToString	時刻→文字列変換	時刻を、文字列表現に変換します。	P.2-461

命令	名称	機能概要	ページ
GrayToBin_**	グレイコード→BIN コード変換グループ	グレイコードをビット列に変換します。	P.2-463
BinToGray_**	BIN コード→グレイコード変換グループ	ビット列をグレイコードに変換します。	P.2-463
StringToAry	文字列→配列変換	文字列を、BYTE 型の配列に変換します。	P.2-466
AryToString	配列→文字列変換	BYTE 型配列を、文字列に変換します。	P.2-468
DispartDigit	4 ビット分離	ビット列を、4 ビット単位に分離します。	P.2-470
UniteDigit_**	4 ビット結合グループ	4 ビット単位のデータを、ビット列に結合します。	P.2-472
Dispart8Bit	バイト単位データ分離	ビット列を、1 バイト単位に分離します。	P.2-474
Unite8Bit_**	バイト単位データ結合グループ	1 バイト単位のデータを、ビット列に結合します。	P.2-476
ToAryByte	バイト配列への変換	変数を 1 バイト単位に分割して、BYTE 型配列に格納します。	P.2-478
AryByteTo	バイト配列からの変換	BYTE 型配列要素を結合して、変数に格納します。	P.2-484
SizeOfAry	配列要素数取得	配列要素数を、取得します。	P.2-489
PackWord	2 バイト結合	2 つの 1 バイトデータを 1 つの 2 バイトデータに結合します。	P.2-491
PackDword	4 バイト結合	4 つの 1 バイトデータを 1 つの 4 バイトデータに結合します。	P.2-493
LOWER_BOUND	配列開始番号取得	配列次元の開始番号を取得します。	P.2-495
UPPER_BOUND	配列終了番号取得	配列次元の終了番号を取得します。	P.2-495

スタック／テーブル命令

命令	名称	機能概要	ページ
StackPush	スタックデータ格納	スタックに、値を格納します。	P.2-500
StackFIFO	先入れ先出し	スタックの最下位の値を取り出します。	P.2-509
StackLIFO	後入れ先出し	スタックの最上位の値を取り出します。	P.2-509
StackIns	スタックデータ挿入	スタックの任意の位置に、値を挿入します。	P.2-512
StackDel	スタックデータ削除	スタックの任意の位置の、値を削除します。	P.2-514
RecSearch	レコード検索	構造体を要素とする配列から、検索キーと一致する要素を、指定された方法で検索します。	P.2-516
RecRangeSearch	範囲指定レコード検索	構造体を要素とする配列から、検索条件の範囲に一致する要素を、指定された方法で検索します。	P.2-520
RecSort	レコードソート	構造体を要素とする配列の要素を、ソートします。	P.2-525
RecNum	レコード数取得	構造体を要素とする配列の、終端データまでのレコード数を求めます。	P.2-531
RecMax	レコード最大値検索	構造体を要素とする配列から、指定されたメンバの最大値を検索します。	P.2-533
RecMin	レコード最小値検索	構造体を要素とする配列から、指定されたメンバの最小値を検索します。	P.2-533

FCS 命令

命令	名称	機能概要	ページ
StringSum	サム値算出	文字列のサム値を算出します。	P.2-538
StringLRC	LRC 値算出（文字列）	文字列の LRC 値（水平パリティ）を算出します。	P.2-540
StringCRCCCITT	CRC-CCITT 値算出（文字列）	文字列の CRC-CCITT 値を、XMODEM 方式で算出します。	P.2-542

命令	名称	機能概要	ページ
StringCRC16	CRC-16 値算出（文字列）	文字列の CRC-16 値を、MODBUS 方式で算出します。	P.2-544
AryLRC_**	LRC 値算出（配列）グループ	配列の LRC 値を算出します。	P.2-546
AryCRCCCITT	CRC-CCITT 値算出（配列）	配列の CRC-CCITT 値を、XMODEM 方式で算出します。	P.2-548
AryCRC16	CRC-16 値算出（配列）	配列の CRC-16 値を、MODBUS 方式で算出します。	P.2-550

文字列命令

命令	名称	機能概要	ページ
CONCAT	文字列・連結	2～5 個の文字列を連結します。	P.2-554
LEFT	文字列・左からの取出	文字列の左（先頭）から、指定数の文字列を取り出します。	P.2-556
RIGHT	文字列・右からの取出	文字列の右（末尾）から、指定数の文字列を取り出します。	P.2-556
MID	文字列・任意位置からの取出	文字列の任意位置から、指定数の文字列を取り出します。	P.2-558
FIND	文字列・検索	文字列の中の指定文字列の位置を検索します。	P.2-560
LEN	文字列・長さ検出	文字列の文字数を求めます。	P.2-562
REPLACE	文字列・置換	文字列の一部を、別の文字列で置換します。	P.2-563
DELETE	文字列・削除	文字列の一部または全部を削除します。	P.2-565
INSERT	文字列・挿入	文字列の中に、別の文字列を挿入します。	P.2-567
GetByteLen	文字列・バイト数取得	文字列のバイト数をカウントします。	P.2-569
ClearString	文字列・クリア	文字列をクリアします。	P.2-570
ToUCase	文字列・大文字変換	文字列中の半角アルファベットをすべて大文字に変換します。	P.2-571
ToLCase	文字列・小文字変換	文字列中の半角アルファベットをすべて小文字に変換します。	P.2-571
TrimL	文字列・左トリム	文字列中の先頭の空白を削除します。	P.2-573
TrimR	文字列・右トリム	文字列中の末尾の空白を削除します。	P.2-573
AddDelimiter	デリミタ付加文字列書込	構造体全体の値を、デリミタを付加した文字列に変換します。	P.2-575
SubDelimiter	文字列読出デリミタ削除	文字列からデリミタで区切られたデータを読み出し、構造体のメンバの値として格納します。	P.2-586

時間／時刻命令

命令	名称	機能概要	ページ
ADD_TIME	時間加算	時間と時間を加算します。	P.2-599
ADD_TOD_TIME	時刻と時間の加算	時刻に時間を加算します。	P.2-601
ADD_DT_TIME	日付時刻と時間の加算	日付時刻に時間を加算します。	P.2-603
SUB_TIME	時間減算	時間から時間を減算します。	P.2-605
SUB_TOD_TIME	時刻と時間の減算	時刻から時間を減算します。	P.2-607
SUB_TOD_TOD	時刻減算	時刻から時刻を減算します。	P.2-609
SUB_DATE_DATE	日付減算	日付から日付を減算します。	P.2-610
SUB_DT_DT	日付時刻減算	日付時刻から日付時刻を減算します。	P.2-611
SUB_DT_TIME	日付時刻と時間の減算	日付時刻から時間を減算します。	P.2-613

命令	名称	機能概要	ページ
MULTIME	時間乗算	時間を指定乗数で乗算します。	P.2-615
DIVTIME	時間除算	時間を指定除数で除算します。	P.2-617
CONCAT_DATE_TOD	日付と時刻の結合	日付と時刻を結合します。	P.2-619
DT_TO_TOD	日付時刻からの時刻切出	日付時刻から時刻を切り出します。	P.2-621
DT_TO_DATE	日付時刻からの日付切出	日付時刻から日付を切り出します。	P.2-623
GetTime	時刻取得	現在時刻を読み出します。	P.2-625
DtToSec	日付時刻→秒変換	日付時刻を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。	P.2-627
DateToSec	日付→秒変換	日付を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。	P.2-629
TodToSec	時刻→秒変換	時刻を、0 時 0 分 0 秒からの秒数に変換します。	P.2-631
SecToDt	秒→日付時刻変換	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付時刻に変換します。	P.2-633
SecToDate	秒→日付変換	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付に変換します。	P.2-635
SecToTod	秒→時刻変換	0 時 0 分 0 秒からの秒数を、時刻に変換します。	P.2-637
TimeToNanoSec	時間→ナノ秒変換	時間を、ナノ秒数に変換します。	P.2-639
TimeToSec	時間→秒変換	時間を、秒数に変換します。	P.2-640
NanoSecToTime	ナノ秒→時間変換	ナノ秒数を、時間に変換します。	P.2-642
SecToTime	秒→時間変換	秒数を、時間に変換します。	P.2-643
ChkLeapYear	うるう年判別	指定された年が、うるう年かどうかを判定します。	P.2-645
GetDaysOfMonth	月の日数取得	指定された月の、日数を取得します。	P.2-646
DaysToMonth	日数→月変換	1 月 1 日からの日数から、その日が何月かを算出します。	P.2-648
GetDayOfWeek	曜日取得	指定された年月日の、曜日を取得します。	P.2-650
GetWeekOfYear	週取得	指定された年月日が、その年の第何週であるかを算出します。	P.2-652
DtToDateStruct	時刻分解	日付時刻を、年、月、日、時、分、秒、ナノ秒に分解します。	P.2-654
DateStructToDt	時刻結合	年、月、日、時、分、秒、ナノ秒に分解された日付時刻を、結合します。	P.2-656
TruncTime	時間切り捨て	TIME 型変数の、指定した単位未満の値を切り捨てます。	P.2-658
TruncDt	日付時刻切り捨て	DT 型変数の、指定した単位未満の値を切り捨てます。	P.2-662
TruncTod	時刻切り捨て	TOD 型変数の、指定した単位未満の値を切り捨てます。	P.2-666

アナログ制御命令

命令	名称	機能概要	ページ
PIDAT	オートチューニング付き PID 演算	オートチューニング付きの PID 演算（目標値フィルタ型 2 自由度 PID 制御） を行います。	P.2-672
PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	オートチューニング付きの加熱冷却 PID 演算（目標値フィルタ型 2 自由度 PID 制御） を行います。	P.2-699
TimeProportionalOut	時分割比例出力	操作量を時分割比例出力に変換します。	P.2-733
LimitAlarm_**	上下限警報グループ	入力値が、下限設定値未満の値、または上限設定値を超える値の場合に警報を出力します。	P.2-750
LimitAlarmDv_**	上下限偏差警報グループ	基準値からの入力値の偏差が、下限偏差設定値または上限偏差設定値を超える値の場合に警報を出力します。	P.2-754

命令	名称	機能概要	ページ
LimitAlarmDvStby Seq_**	待機シーケンス付き上下限偏差警報グループ	待機シーケンス付きで上下限偏差警報を出力します。	P.2-758
ScaleTrans	スケール変換	入力値を入力レンジから出力レンジに変換します。	P.2-774
AC_StepProgram	ステッププログラム	指定したプログラムパターンに従い、タスク周期ごとに現在目標値と予測目標値を算出します。	P.2-777

システム制御命令

命令	名称	機能概要	ページ
TraceSamp	データトレースサンプリング	データトレースのサンプリングを実行します。	P.2-802
TraceTrig	データトレーストリガ条件成立	データトレースのトリガを発生させます。	P.2-805
GetTraceStatus	データトレースステータス読出	データトレースの実行ステータスを、読み出します。	P.2-808
SetAlarm	ユーザ異常発生	ユーザ異常を発生させます。	P.2-811
ResetAlarm	ユーザ異常解除	ユーザ異常を解除します。	P.2-816
GetAlarm	ユーザ異常ステータス取得	発生中のユーザ異常の、最重要のイベント重要度（ユーザ異常レベル 1～8）と、最重要のイベントコードを取得します。	P.2-818
ResetPLCError	PLC 異常解除	PLC 機能モジュールで発生中のコントローラ異常を解除します。	P.2-820
GetPLCError	PLC 異常ステータス取得	PLC 機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。	P.2-823
GetEIPError	EtherNet/IP 異常ステータス取得	EtherNet/IP 機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。	P.2-825
ResetMCError	モーション制御異常解除	モーション制御機能モジュールで発生中のコントローラ異常を解除します。	P.2-827
GetMCError	モーション制御異常ステータス取得	モーション制御機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。	P.2-832
ResetECError	EtherCAT 異常解除	EtherCAT マスタ機能モジュールで発生中のコントローラ異常を解除します。	P.2-834
GetECError	EtherCAT 異常ステータス取得	EtherCAT マスタ機能モジュールで発生中の異常を検出します。	P.2-836
SetInfo	ユーザ情報発生	ユーザ情報を発生させます。	P.2-838
RestartNXUnit	NX ユニットリスタート	EtherCAT カプラユニット、または NX ユニットをリスタートします。	P.2-840
NX_ChangeWriteMode	NX ユニット書込モード変更	EtherCAT カプラユニット、または NX ユニットを、データの書込ができるモードに変更します。	P.2-845
NX_SaveParam	NX ユニットパラメータ保存	EtherCAT カプラユニット、または NX ユニットに書き込んだデータを、保存します。	P.2-850
NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	通信カプラユニット、および NX ユニットが保有する累積通電時間を読み出します。	P.2-855

プログラム制御命令

命令	名称	機能概要	ページ
PrgStart	プログラム実行指令	指定したプログラムの実行を指令します。	P.2-864
PrgStop	プログラム停止指令	指定したプログラムの停止を指令します。	P.2-872
PrgStatus	プログラム状態読出	指定したプログラムの状態を呼び出します。	P.2-889

EtherCAT 通信命令

命令	名称	機能概要	ページ
EC_CoESDOWrite	CoE SDO 書込	EtherCAT ネットワーク上の、指定したスレーブが持つ CoE のオブジェクトに、値を書き込みます。	P.2-896
EC_CoESDORead	CoE SDO 読出	EtherCAT ネットワーク上の、指定したスレーブが持つ CoE のオブジェクトから、値を読み出します。	P.2-899
EC_StartMon	EtherCAT パケットモニタ機能開始	EtherCAT 通信の、パケットモニタ機能を実行開始します。	P.2-904
EC_StopMon	EtherCAT パケットモニタ機能停止	EtherCAT 通信の、パケットモニタ機能を停止します。	P.2-910
EC_SaveMon	EtherCAT パケット保存	EtherCAT 通信のパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存します。	P.2-912
EC_CopyMon	EtherCAT パケット転送	EtherCAT 通信の、CPU ユニットのメインメモリ上の内部ファイルにあるパケットデータを、SD メモリカードに転送します。	P.2-914
EC_DisconnectSlave	EtherCAT スレーブ離脱	EtherCAT ネットワーク上の、指定したスレーブを、ネットワークから離脱させます。	P.2-916
EC_ConnectSlave	EtherCAT スレーブ再加入	EtherCAT ネットワーク上の、指定したスレーブを、ネットワークに再加入させます。	P.2-923
EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	EtherCAT スレーブの、有効／無効切替を行います。	P.2-925
EC_GetMasterStatistics	EtherCAT マスタ診断・統計情報読出	EtherCAT マスタの診断・統計情報を読み出します。	P.2-943
EC_ClearMasterStatistics	EtherCAT マスタ診断・統計情報クリア	EtherCAT マスタの診断・統計情報をクリアします。	P.2-946
EC_GetSlaveStatistics	EtherCAT スレーブ診断・統計情報読出	EtherCAT スレーブの診断・統計情報を読み出します。	P.2-948
EC_ClearSlaveStatistics	EtherCAT スレーブ診断・統計情報クリア	EtherCAT スレーブの診断・統計情報をクリアします。	P.2-951
NX_WriteObj	NX オブジェクト書込	EtherCAT カプラユニット、または NX ユニットの NX オブジェクトに、データを書き込みます。	P.2-953
NX_ReadObj	NX オブジェクト読出	EtherCAT カプラユニット、または NX ユニットの NX オブジェクトから、データを読み出します。	P.2-968

IO-Link 通信命令

命令	名称	機能概要	ページ
IOL_ReadObj	IO-Link デバイスオブジェクト読出	IO-Link デバイスのオブジェクトからデータを読み出します。	P.2-976
IOL_WriteObj	IO-Link デバイスオブジェクト書込	IO-Link デバイスのオブジェクトにデータを書き込みます。	P.2-984

EtherNet/IP 通信命令

命令	名称	機能概要	ページ
CIPOpen	CIPClass3 (Large_Forward_Open) コネクション確立	指定した相手ノードに対して、データ長 1994 バイトで、CIPClass3 (Large_Forward_Open) コネクションを確立します。	P.2-995
CIPOpenWithDataSize	CIP Class3 コネクション確立（サイズ指定あり）	指定した相手ノードに対して、指定したデータ長以下の Class3 Explicit メッセージを送受信可能な CIP の Class3 コネクションを確立します。	P.2-1004

命令	名称	機能概要	ページ
CIPRead	変数読出（Class3 Explicit メッセージ）	CIP ネットワーク上の、他局コントローラの変数の値を読み出します（Class3 Explicit メッセージ）。	P.2-1007
CIPWrite	変数書込（Class3 Explicit メッセージ）	CIP ネットワーク上の、他局コントローラの変数に、値を書き込みます（Class3 Explicit メッセージ）。	P.2-1012
CIPSend	任意 Explicit メッセージ送信（Class3）	CIP ネットワーク上の指定したデバイスに、Class3 の CIP メッセージを送信します。	P.2-1018
CIPClose	CIP Class3 コネクションの切断	指定したハンドルの、CIP Class3 の コネクション,.を切断します。	P.2-1023
CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	指定した CIP ネットワーク上の、他局コントローラの変数の値を読み出します（UCMM Explicit メッセージ）。	P.2-1025
CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	CIP ネットワーク上の、他局コントローラの変数に、値を書き込みます（UCMM Explicit メッセージ）。	P.2-1030
CIPUCMMSend	任意 Explicit メッセージ送信(UCMM)	CIP ネットワーク上の指定したデバイスに、UCMM の CIP メッセージを送信します。	P.2-1036
SktUDPCreate	UDP ソケットクリエイト	EtherNet/IP の、UDP ソケットのクリエイト要求（サーバポートのオープン）を行います。	P.2-1047
SktUDPRcv	UDP ソケット受信	EtherNet/IP の、UDP ソケットの受信バッファ内のデータを読み出します。	P.2-1055
SktUDPSend	UDP ソケット送信	EtherNet/IP の UDP ポートから、データを送信します。	P.2-1058
SktTCPAccept	TCP ソケットアクセプト	EtherNet/IP の、TCP ソケットのアクセプト要求を行います。	P.2-1061
SktTCPConnect	TCP ソケットコネクト	EtherNet/IP から、相手 TCP ポートにコネクト（接続）します。	P.2-1064
SktTCPRcv	TCP ソケット受信	EtherNet/IP の、指定した TCP ソケットの受信バッファ内のデータを読み出します。	P.2-1072
SktTCPSend	TCP ソケット送信	EtherNet/IP の、指定した TCP ポートから、データを送信します。	P.2-1075
SktGetTCPStatus	TCP ソケットのステータス読出	TCP ソケットの、ステータスを読み出します。	P.2-1078
SktClose	TCP/UDP ソケットクローズ	EtherNet/IP の指定した、TCP ソケットまたは UDP ソケットを、クローズします。	P.2-1081
SktClearBuf	TCP/UDP ソケット受信バッファクリア	EtherNet/IP の指定した、TCP ソケットまたは UDP ソケットの、受信バッファをクリアします。	P.2-1084
SktSetOption	TCP ソケットオプション設定	EtherNet/IP の、指定した TCP ソケットのソケットオプションを設定します。	P.2-1087
ChangeIPAdr	IP アドレス変更	EtherNet/IP ポート、または内部通信ポートの IP アドレスを変更します。	P.2-1092
ChangeFTPAccount	FTP アカウント変更	EtherNet/IP ポートの FTP のログイン名とパスワードを変更します。	P.2-1100
FTPGetFileList	FTP サーバのファイルリスト取得	FTP サーバ内のファイルのリストを取得します。	P.2-1104
FTPGetFile	FTP サーバからのファイルダウンロード	FTP からファイルをダウンロードします。	P.2-1117
FTPPutFile	FTP サーバへのファイルアップロード	FTP サーバへファイルをアップロードします。	P.2-1125
FTPRemoveFile	FTP サーバのファイル削除	FTP サーバのファイルを削除します。	P.2-1135
FTPRemoveDir	FTP サーバのディレクトリ削除	FTP サーバのディレクトリを削除します。	P.2-1144

シリアル通信命令

命令	名称	機能概要	ページ
NX_SerialSend	無手順データの送信	NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、無手順でデータを送信します。	P.2-1150
NX_SerialRcv	無手順データの受信	NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、無手順でデータを読み出します。	P.2-1161
NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに対して汎用コマンドを送信します。	P.2-1174
NX_ModbusRtuRead	Modbus RTU Read コマンド送信	NX シリーズ コミュニケーションインタフェースユニットから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに読み出しコマンドを送信します。	P.2-1184
NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	NX シリーズ コミュニケーションインタフェースユニットから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに書き込みコマンドを送信します。	P.2-1195
NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	NX シリーズ コミュニケーションインタフェースユニットのシリアルポートの ER 信号または RS 信号を ON または OFF にします。	P.2-1206
NX_SerialBufClear	バッファクリア	送受信バッファをクリアします。	P.2-1213
NX_SerialStartMon	シリアルラインモニタの開始	NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを開始します。	P.2-1222
NX_SerialStopMon	シリアルラインモニタの停止	NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを停止します。	P.2-1226

SD メモリカード命令

命令	名称	機能概要	ページ
FileWriteVar	変数ファイル書込	SD メモリカード内の指定したファイルに、1 個の変数の値を、バイナリ形式で書き込みます。	P.2-1232
FileReadVar	変数ファイル読出	SD メモリカード内の、指定したファイルの値をバイナリ形式で読み出し、変数に書き込みます。	P.2-1237
FileOpen	ファイルオープン	SD メモリカード内の指定したファイルを、オープンします。	P.2-1242
FileClose	ファイルクローズ	SD メモリカード内の指定したファイルを、クローズします。	P.2-1246
FileSeek	ファイルシーク	SD メモリカード内の指定したファイルに、ファイル位置指示子を設定します。	P.2-1249
FileRead	ファイル読出	SD メモリカード内の、指定したファイルのデータを読み出します。	P.2-1252
FileWrite	ファイル書込	SD メモリカード内の、指定したファイルにデータを書き込みます。	P.2-1260
FileGets	文字列読出	SD メモリカード内の指定したファイルから、1 行分の文字列を読み出します。	P.2-1268
FilePuts	文字列書込	SD メモリカード内の指定したファイルに、文字列を書き込みます。	P.2-1275
FileCopy	ファイルコピー	SD メモリカード内の指定したファイルを、コピーします。	P.2-1284
FileRemove	ファイル削除	SD メモリカード内の指定したファイルを、削除します。	P.2-1291
FileRename	ファイル名変更	SD メモリカード内の指定したファイルやディレクトリの、名称を変更します。	P.2-1295
DirCreate	ディレクトリ作成	SD メモリカード内に、指定した名称のディレクトリを作成します。	P.2-1300
DirRemove	ディレクトリ削除	SD メモリカード内の、指定したディレクトリを削除します。	P.2-1303

命令	名称	機能概要	ページ
BackupToMemoryCard	SD メモリカードバックアップ	SD メモリカードバックアップを実行します。	P.2-1306

タイムスタンプ命令

命令	名称	機能概要	ページ
NX_DOutTimeStamp	タイムスタンプデジタル出力書込	タイムスタンプ方式対応のデジタル出力ユニットの出力接点に、値を書き込みます。	P.2-1320
NX_AryDOutTimeStamp	タイムスタンプデジタル出力配列書込	タイムスタンプ方式対応のデジタル出力ユニットから、パルスを出力させます。	P.2-1325

OS 制御命令

命令	名称	機能概要	ページ
IPC_GetOSStatus	OS ステータス読み出し	産業用 PC のオペレーティングシステム(Windows)のステータスを読み出します。	P.2-1334
IPC_RebootOS	OS 再起動	OS 再起動産業用 PC のオペレーティングシステム(Windows)を再起動します。	P.2-1337
IPC_Shutdown	シャットダウン	産業用 PC のシステムでのシャットダウン処理を開始し、完了後に Windows 側にシャットダウン要求を通知します。	P.2-1340

その他の命令

命令	名称	機能概要	ページ
ReadNbit_**	N ビット読出グループ	ビット列内の、複数ビットを読み出します。	P.2-1344
WriteNbit_**	N ビット書込グループ	ビット列内に、複数ビットを書き込みます。	P.2-1346
ChkRange	範囲型変数チェック	変数の値が、範囲型指定の有効範囲内にあるかどうかを判定します。	P.2-1348
GetMyTaskStatus	自タスク状態読出	自タスクのタスク状態を読み出します。	P.2-1350
GetMyTaskInterval	自タスク設定周期読出	自タスクのタスク周期を読み出します。	P.2-1353
Task_IsActive	タスク実行中判定	指定したタスクが、実行中かどうかを判定します。	P.2-1355
Lock	タスク間排他ロック	タスク間排他ロックを開始します。他のタスクの同じロック番号の区間は、実行できなくなります。	P.2-1357
Unlock	タスク間排他ロック解除	タスク間排他ロックを終了します。	P.2-1357
ActEventTask	イベントタスク起動	イベントタスクを起動します。	P.2-1363
Get**Clk	クロックパルス取得グループ	指定した周期の、クロックパルスを出力します。	P.2-1370
Get**Cnt	加算フリーランニングカウンタ取得グループ	指定した周期の、フリーランニングカウンタの値を取得します。	P.2-1371

2

各命令の説明

この章では、NY シリーズ コントローラで使用できる命令の仕様を、説明します。

本章の見方 2-3
ラダー図命令 2-13
ST 構文命令 2-27
シーケンス入力命令 2-45
シーケンス出力命令 2-53
シーケンス制御命令 2-69
比較命令 2-95
タイマ命令 2-125
カウンタ命令 2-145
算術命令 2-167
BCD 変換命令 2-241
データ型変換命令 2-261
ビット列演算命令 2-323
選択命令 2-335
データ転送命令 2-355
シフト命令 2-389
データ変換命令 2-405
スタック／テーブル命令 2-499
FCS 命令 2-537
文字列命令 2-553
時間／時刻命令 2-597
アナログ制御命令 2-671
システム制御命令 2-801
プログラム制御命令 2-863
EtherCAT 通信命令 2-895
IO-Link 通信命令 2-975

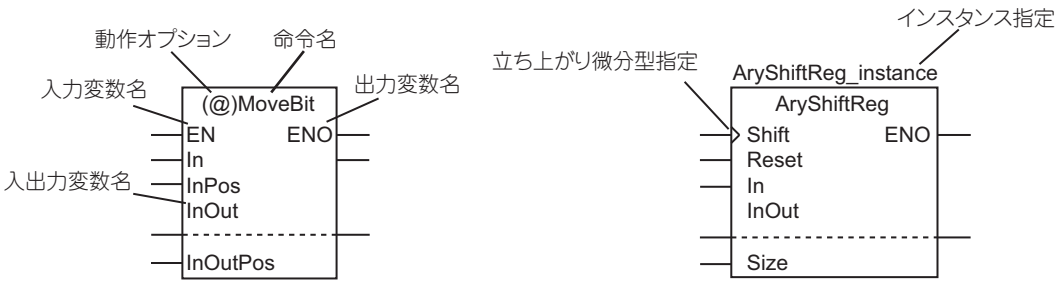
EtherNet/IP 通信命令 2-993
シリアル通信命令 2-1149
SD メモリカード命令 2-1231
タイムスタンプ命令 2-1319
OS 制御命令 2-1333
その他の命令 2-1343

本章の見方

ここでは、本章（各命令の説明）の記載項目と、見方を説明します。

記載項目

記載項目は、以下のとおりです。

項目	内容
命令	命令名を示します。例: MoveBit
名称	命令の日本語名称を示します。例: ビット転送
FB/FUN	命令がファンクションブロック(FB)型か、ファンクション(FUN)型かを示します。 FB 型の命令は、プログラムまたは FB からしか呼び出せません。 FUN 型の命令は、プログラム、FB、FUN のいずれからでも呼び出せます。
グラフィック表現	その命令を、ラダー図プログラム内で表現する際の図を示します。 ●FUN 型の例　●FB 型の例  このうち、「動作オプション」、「立ち上がり微分型指定」、「インスタンス指定」の意味は、つぎのとおりです。 動作オプション　：FUN 型の命令名の前に書かれている(@)のことです。 動作オプションがある命令は、命令名の前に@を付加することにより、入力立ち上がり微分型にすることができます。 入力立ち上がり微分型とは、実行条件の入力変数(EN)の値が、前回のタスク周期で FALSE、現在のタスク周期で TRUE の場合に実行される命令のことです。 立ち上がり微分型指定　：入力変数のピンに書かれている矢印（->）のことです。この指定がある命令は、入力立ち上がり微分型として動作します。 インスタンス指定　：FB 型の命令の上に書かれている「XX_instance」のことです。インスタンス指定が書かれている命令には、任意のインスタンス名を付ける必要があります。
ST 表現	その命令を、ST プログラム内で表現する際の記述を示します。 ST プログラム内で命令を記述するには、以下の 2 つの方法があります。 (1) 変数（入力変数、出力変数、入出力変数）名とパラメータ名の対応を明示する方法 例: MoveBit(In:=abc, InPos:=def, InOut:=ghi, InOutPos:=jkl); (2) 変数（入力変数、出力変数、入出力変数）名は省略し、パラメータ名のみを記述する方法 例: MoveBit(In, InPos, InOut, InOutPos); 本章では、(2)の方法で記述しています。 XX_instance（変数名）と書かれている命令には、任意のインスタンス名を付ける必要があります。 例: TON_instance(In, PT, Q, ET);

項目	内容
変数	• 変数名 入力変数、出力変数、入出力変数の名称を示します。例: In1 ただし、複数の命令に共通に使われる変数については、各命令の説明ページには記載していません。共通に使われる変数とは、以下の 8 つの変数です。これらの仕様は、後述します。 (EN, ENO, Execute, Done, Busy, Error, ErrorID, ErrorIDEx) • 名称 変数の日本語名称を示します。例: 加算カウンタ • 入／出力 入力変数、出力変数、入出力変数のいずれであるかを示します。 • 内容 その変数の意味や、制限を示します。 • 有効範囲 その変数の取り得る値の範囲を示します。 「データ型に従う」と書かれている場合は、その変数のデータ型の有効範囲に従います。各データ型の有効範囲は、後述します。 • 単位 その変数の単位を示します。「－」と書かれている場合は、単位なしです。例: バイト • 初期値 変数にパラメータを代入せずに命令を実行した場合、その変数に自動的に設定される値のことです。 「－」と書かれている場合は、以下の意味になります。 入力変数の場合 : その入力変数のデータ型のデフォルトの初期値が代入されます。各データ型のデフォルトの初期値は、後述します。 入力変数が構造体の場合、機能説明中の構造体仕様の表で、初期値を示します。 出力変数の場合 : 初期値は設定されません。 入出力変数の場合 : 初期値は設定されません。 • データ型 その変数のデータ型を示します。 列挙型、配列、構造体、共用体が適用される場合は、そのように記述しています。
機能	その命令の機能を示します。 文中で、変数名はかぎかっこ「 」付きで記述します。例: 「In1」 また、配列名は角かっこ[]付きで記述します。例: InOut[]
関連するシステム定義変数	その命令に関連するシステム定義変数の、名称と概略を示します。 システム定義変数の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。
関連する準ユーザ定義変数	その命令に関連する準ユーザ定義変数の、名称と概略を示します。 準ユーザ定義変数の詳細は、そのページに示した参照マニュアルを、参照してください。
参考	その命令の機能の補足情報を示します。 関連する命令や、その命令の便利な使い方などを記述しています。
使用上の注意	その命令を使用するうえでの、注意事項を示します。 どんな場合に異常が発生するかも、ここに記述しています。
サンプルプログラム	その命令を含む、短いアプリケーションプログラムの例を示します。 同じ処理内容で、ラダー図プログラムと ST プログラムの両方を、記述しています。

共通変数

複数の命令に共通に使われる変数(EN, ENO, Execute, Done, Busy, Error, ErrorID, ErrorIDEx)の仕様を説明します。

これらの共通変数は、各命令の変数の表には記載していません。各命令がこれらの共通変数を持つかどうかは、グラフィック表現および ST 表現の欄で確認してください。

EN

「EN」は、FUN 型の命令の、実行条件を示す入力変数です。
ラダー図プログラムで、FUN 型の命令を使用する場合は、実行条件を「EN」に入力（結線）してください。

	名称	入／出力	内容	データ型	有効範囲	初期値
EN	イネーブル（実行条件）	入力	TRUE: 命令を実行する。*1 FALSE: 命令を実行しない。	BOOL	TRUE, FALSE	TRUE

*1. 動作オプションで入力立ち上がり微分型(@)に指定されている場合は、「EN」の値が FALSE から TRUE に変化することが、実行条件となります。

- FB 型の命令には、「EN」はありません。
- ST プログラム FUN 型の命令を呼び出す場合は、「EN」の記述を省略してください。命令の実行条件は、ST プログラムの動作シーケンスで決定するので、「EN」は不要です。

ENO

「ENO」は、命令の実行結果を示す出力変数です。
ラダー図プログラムにおいては、実行条件を後段の命令に伝えるために使用することができます。
通常は、命令が正常終了した時点で「ENO」の値が TRUE になります。それを受けて、後段の命令が実行を開始します。

	名称	入／出力	内容	データ型	有効範囲	初期値
ENO	イネーブルアウト	出力	TRUE: 正常終了 *1 FALSE: 異常終了、実行中、または実行条件非成立	BOOL	TRUE, FALSE	−

*1. 実行条件が成立している場合に限ります。正常終了後、実行条件が非成立になったら、その時点で「ENO」の値は FALSE になります。

- 「ENO」は、FUN 型の命令にも FB 型の命令にも存在します。しかし、命令によっては、「ENO」を持たないものもあります。

Execute, Done, Busy

「Execute」は、FB 型の命令の一部が持つ、実行条件を示す入力変数です。
「Execute」の値が FALSE から TRUE に変化すると、命令が実行されます。いったん命令が実行されると、「Execute」の値が FALSE になっても、また命令実行時間がタスク周期を超えても、処理を最後まで続けます。

「Done」は、FB 型の命令の一部が持つ、命令が正常終了したことを示す出力変数です。
「Busy」は、FB 型の命令の一部が持つ、命令が実行中であることを示す出力変数です。

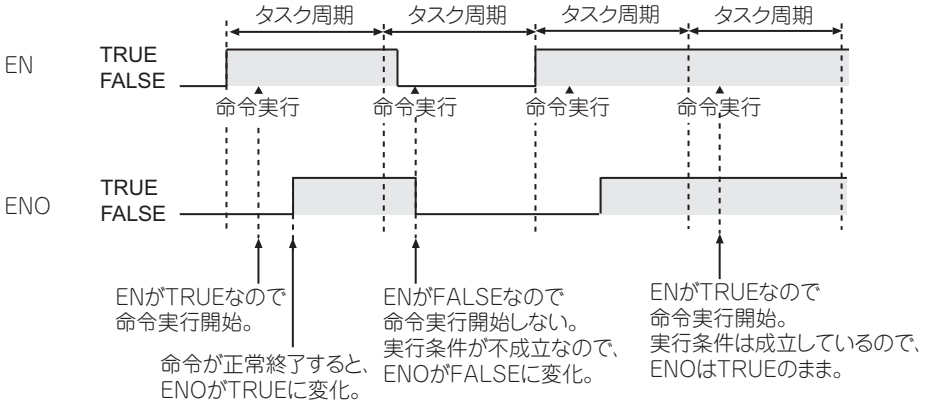
	名称	入／出力	内容	データ型	有効範囲	初期値
Execute	起動	入力	TRUE: 命令を実行する。*1 FALSE: 命令を実行しない。*2	BOOL	TRUE, FALSE	FALSE
Done	完了	出力	TRUE: 正常終了 *3 *4 FALSE: 異常終了、実行中、または実行条件非成立	BOOL	TRUE, FALSE	−
Busy	実行中		TRUE: 実行中 FALSE: 実行中ではない			

*1. 運転開始時にすでに「Execute」の値が TRUE の場合は、命令を実行しません。命令を実行させるためには、「Execute」の値を、いったん FALSE にしてください。

*2. 命令実行中に値が FALSE になっても、処理を最後まで続けます。
*3. 正常終了後、実行条件が非成立になったら、その時点で「Done」の値は FALSE になります。
*4. 正常終了した時点ですでに実行条件が非成立だったら、1 タスク周期分だけ「Done」の値が TRUE になり、その後 FALSE になります。

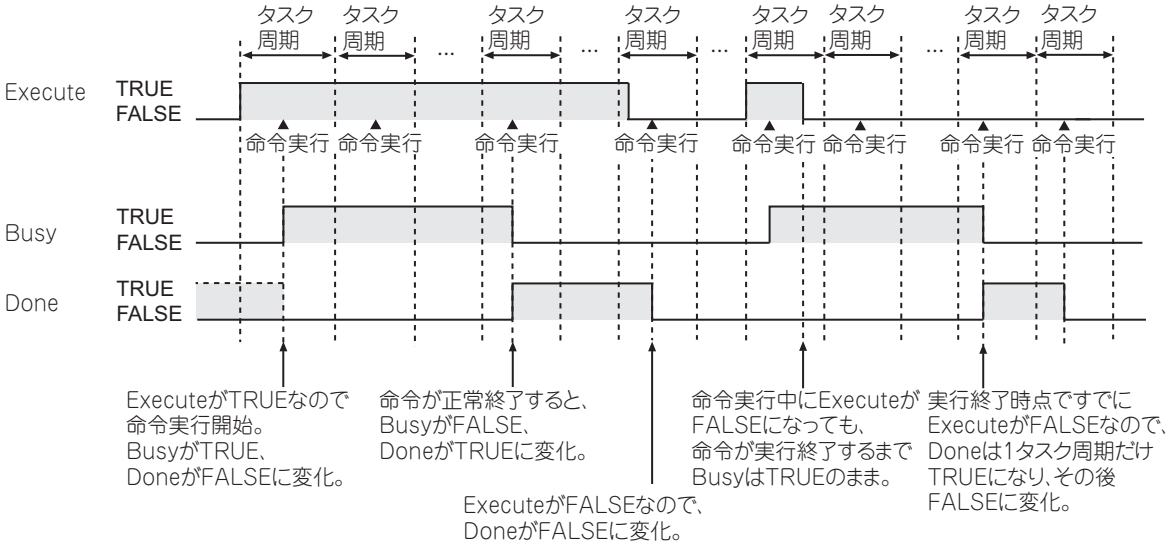
● 1 タスク周期で処理が完了する命令

「EN」や「ENO」を持つ命令、つまり 1 タスク周期で処理が完了する命令のタイミングチャートを、以下に示します。



● 複数タスク周期にまたがって処理が行われる命令

「Execute」や「Busy」を持つ命令、つまり複数のタスク周期にまたがって処理が行われる命令のタイミングチャートを、以下に示します。



Error, ErrorID, ErrorIDEx

「Error」、「ErrorID」、「ErrorIDEx」は、FB 型の命令の一部が持つ、命令が異常終了したことを示す出力変数です。

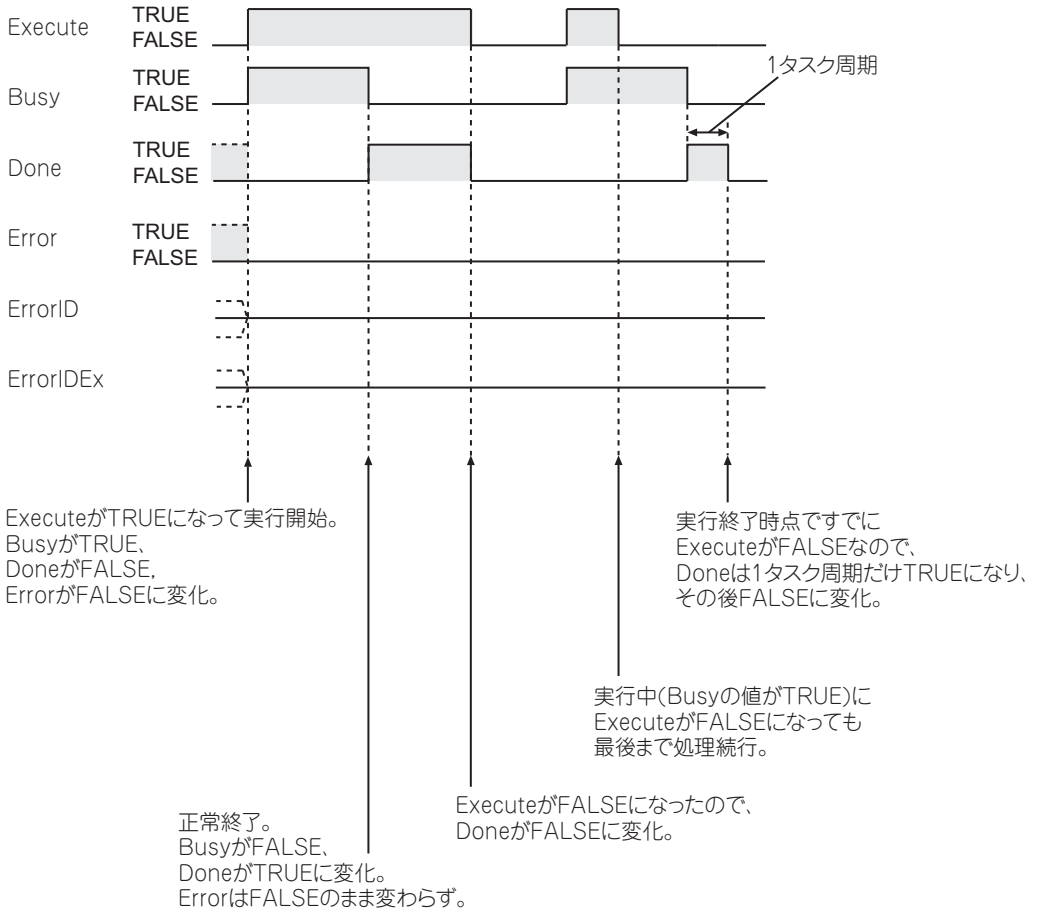
	名称	入／出力	内容	データ型	有効範囲	初期値
Error	エラー	出力	TRUE: 異常終了 *1 *2 FALSE: 正常終了、実行中、または実行条件非成立	BOOL	TRUE, FALSE	−
ErrorID	エラーコード		異常終了時の異常 ID 正常終了時は、WORD#16#0	WORD	命令によって異なる	
ErrorIDEx	拡張エラーコード		異常終了時の拡張異常 ID 正常終了時は、DWORD#16#0	DWORD		

*1. 異常終了後、実行条件が非成立になったら、その時点で「Error」の値は FALSE になります。
*2. 異常終了した時点ですでに実行条件が非成立だったら、1 タスク周期分だけ「Error」の値が TRUE になり、その後 FALSE になります。

「ErrorID」で確認できるエラーコードの一覧は、「A-1 ErrorID で確認できるエラーコード一覧 (P.A-2)」を参照してください。
エラーコードの内容は、『NY シリーズ トラブルシューティングマニュアル(SBCA-438)』を参照してください。

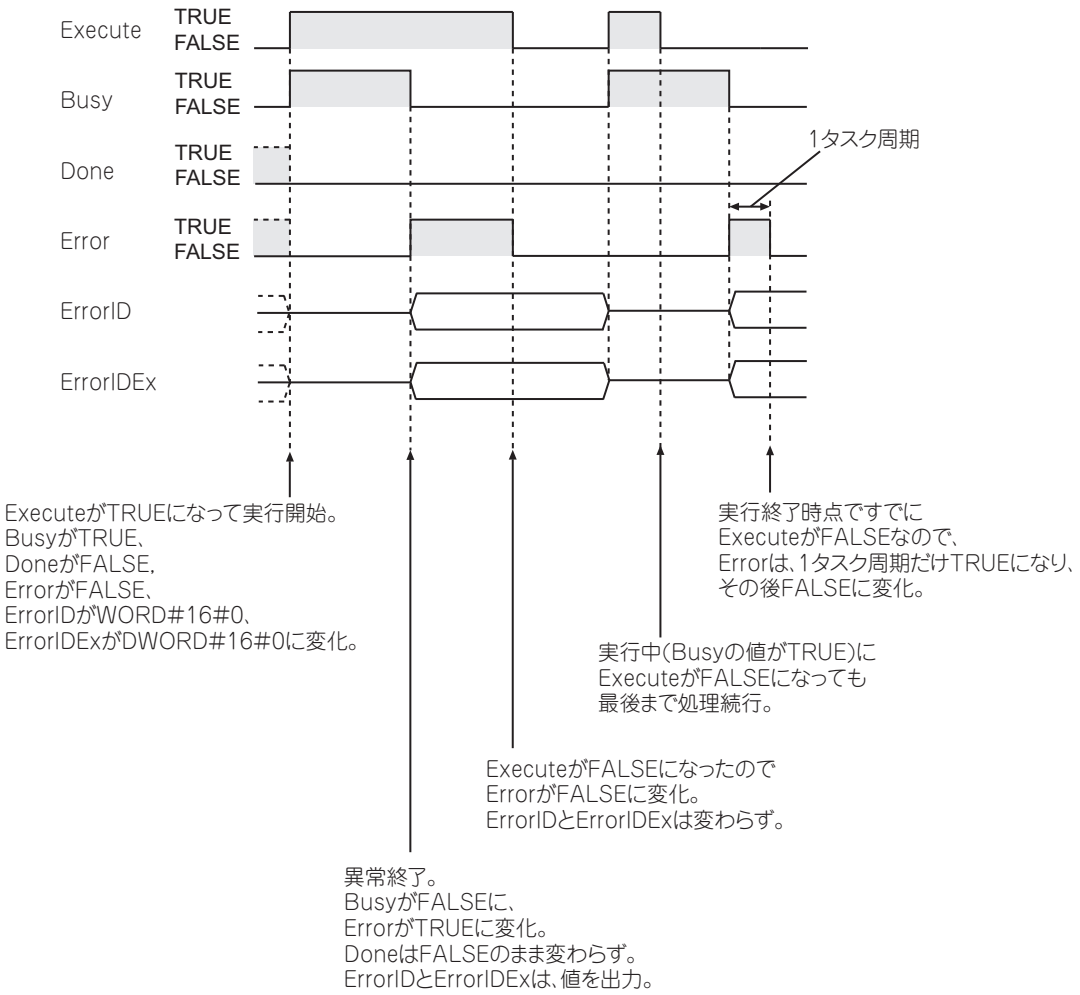
● 正常終了の場合

「Execute」、「Done」、「Busy」、「Error」、「ErrorID」、「ErrorIDEx」のタイミングチャートを以下に示します。



● 異常終了の場合

「Execute」、「Done」、「Busy」、「Error」、「ErrorID」、「ErrorIDEx」のタイミングチャートを以下に示します。



変数の有効範囲とデフォルトの初期値

変数の有効範囲とは、その変数が取り得る値の範囲のことです。また、変数のデフォルトの初期値とは、入力変数にパラメータを代入せずに命令を実行した場合に、その入力変数に代入される値のことです。
これらは、データ型ごとに値が決められています。命令として特に指定のない場合、変数の有効範囲と初期値は、その値に従います。
本章の変数の表で、有効範囲欄に「データ型に従う」、入力変数の初期値欄に「－」と書いてあるものが、それに該当します。

各データ型の有効範囲と初期値を、以下に示します。

分類	データ型	有効範囲	初期値
ブール	BOOL	FALSE, TRUE	FALSE
ビット列	BYTE	BYTE#16#00 ～ FF	BYTE#16#00
	WORD	WORD#16#0000 ～ FFFF	WORD#16#0000
	DWORD	DWORD#16#00000000 ～ FFFFFFFF	DWORD#16#00000000
	LWORD	LWORD#16#0000000000000000 ～ FFFFFFFFFFFFFFFF	LWORD#16#0000000000000000

分類	データ型	有効範囲	初期値
整数	USINT	USINT#0 ～ +255	USINT#0
	UINT	UINT#0 ～ +65535	UINT#0
	UDINT	UDINT#0 ～ +4294967295	UDINT#0
	ULINT	ULINT#0 ～ +18446744073709551615	ULINT#0
	SINT	SINT#-128 ～ +127	SINT#0
	INT	INT#-32768 ～ +32767	INT#0
	DINT	DINT#-2147483648 ～ +2147483647	DINT#0
	LINT	LINT#-9223372036854775808 ～ +9223372036854775807	LINT#0
実数	REAL	REAL#-3.402823e+38 ～ -1.175495e-38, 0, +1.175495e-38 ～ +3.402823e+38, +∞/-∞	REAL#0
	LREAL	LREAL#-1.79769313486231e+308 ～ -2.22507385850721e-308, 0, +2.22507385850721e-308 ～ +1.79769313486231e+308, +∞/-∞	LREAL#0
時刻、持続時間、日付、文字列	TIME	T#-9223372036854.775808ms (T#-106751d_23h_47m_16s_854.775808ms) ～ T#9223372036854.775807ms (T#+106751d_23h_47m_16s_854.775807ms)	T#0s
	DATE	D#1970-01-01 ～ D#2106-02-06 (1970 年 1 月 1 日 ～ 2106 年 2 月 6 日)	D#1970-01-01
	TOD	TOD#00:00:00.000000000 ～ TOD#23:59:59.999999999 (0 時 0 分 0.000000000 秒 ～ 23 時 59 分 59.999999999 秒)	TOD#00:00:00.000000000
	DT	DT#1970-01-01-00:00:00.000000000 ～ DT#2106-02-06-23:59:59.999999999 (1970 年 1 月 1 日 0 時 0 分 0.000000000 秒 ～ 2106 年 2 月 6 日 23 時 59 分 59.999999999 秒)	DT#1970-01-01-00:00:00.000000000
	STRING	文字コード: UTF-8 0～1986 バイト (半角英数字で 0 ～ 1985 文字+終端 NULL 文字)	''

派生データ型（列挙型、構造体、共用体）

派生データ型（列挙型、構造体、共用体）を使用する変数については、そのことを、変数のデータ型の表に記述しています。記述方法について、説明します。

列挙型

列挙型変数は、表内に列挙のデータ型を記載しています。
以下に、例を示します。この場合、変数「Out」のデータ型は、列挙型_eDAYOFWEEK です。列挙子の詳細は、機能説明の中で説明します。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out	列挙型_eDAYOFWEEK　列挙子は機能説明参照																			

構造体、共用体

構造体変数、共用体変数は、表内に構造体、共用体のデータ型を記載しています。
以下に例を示します。この場合、変数「In1」のデータ型は、構造体_sPORT です。構造体、共用体のメンバの詳細は、機能説明の中で説明します。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	構造体_sPORT　詳細は機能説明参照																			

構造体全体、構造体の 1 メンバ、共用体全体、共用体の 1 メンバを、パラメータとして指定することが可能な変数は、表内にそのことを記載しています。
下の例の場合、変数「In1」には、基本データ型のほかに、構造体全体、構造体の 1 メンバ、共用体全体、共用体の 1 メンバを指定することが可能です。構造体全体、共用体全体を指定する場合は、構造体名または共用体名だけをパラメータとしてください。構造体の 1 メンバ、共用体の 1 メンバを指定する場合は、メンバ名まで明記したものをパラメータとしてください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	構造体全体、構造体の 1 メンバ、共用体全体、共用体の 1 メンバも指定可能																			

配列指定

配列指定変数には、変数名の後に角かっこ[]を付け、「配列」と記載しています。この記述がある場合は、添え字付きの、配列の要素をパラメータとしてください。
以下に例を示します。この場合、In1[]は BYTE 型の配列であることを示します。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列		○																		

構造体や共用体を要素とする配列は、以下の例のように、表内にそのことを記載しています。この記述の場合も、添え字付きの、配列の要素をパラメータとしてください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列	構造体を要素とする配列、共用体を要素とする配列を指定																			

配列全体、配列の 1 要素を、パラメータとして指定することが可能な変数は、表内にそのことを記載しています。
下の例の場合、変数「In1」には、基本データ型のほかに、配列全体、配列の 1 要素を指定することが可能です。配列全体を指定する場合は、配列名だけをパラメータとしてください。配列の 1 要素を指定する場合は、添え字で要素番号まで明記したものをパラメータとしてください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	配列全体、配列の 1 要素も指定可能																			

その他

全命令共通の異常検知

使用上の注意には、各命令で発生する可能性がある異常の種類が書かれています。しかし、以下の異常は、全命令共通で検知を行うため、使用上の注意には書かれていないので、注意してください。

- 配列指定変数の、配列領域を超えた要素を読み書きした場合。
  （例） 配列指定変数 a[0..3]に対して、a[4]を入力変数に設定した。
- 入力変数、出力変数、入出力変数として配列指定変数が定義されている命令に対して、配列ではないパラメータを渡した場合。
- STRING 型変数に、定義されているバイト数よりも長い文字列を代入した場合。
- STRING 型変数に、NULL 文字で終端されていない文字列を代入した場合。
- 整数型変数を、0 で除算した場合。

全命令共通の注意事項

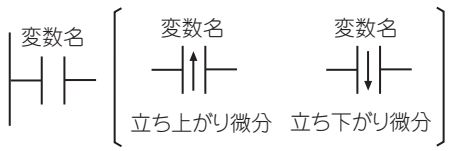
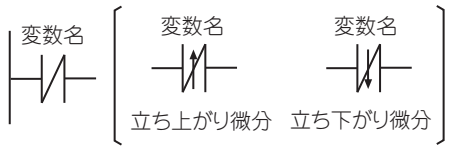
命令の中には、接続するパラメータによって、処理量が可変のものがあります。
処理量が大きすぎると、命令実行時間が長くなり、タスク周期を超えてしまう場合があります。その場合、タスク周期オーバとなってしまうので、処理量は適切に調整してください。

ラダー図命令

命令	名称	ページ
LD / LDN	ロード／ロードノット	P.2-14
AND / ANDN	アンド／アンドノット	P.2-17
OR / ORN	オア／オアノット	P.2-20
Out / OutNot	出力／否定出力	P.2-23

LD / LDN

LD　：BOOL 型変数の値を読み込みます。
LDN ：BOOL 型変数の反転値を読み込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LD	ロード	–		なし
LDN	ロードノット	–		なし

変数

なし

機能

LD

変数名で指定した BOOL 型変数の値を読み込み、後段に接続（出力）します。
指定した変数の値が TRUE なら TRUE を出力、FALSE なら FALSE を出力します。
母線から始まる最初の a 接点、または回路ブロックの最初の a 接点に使用します。

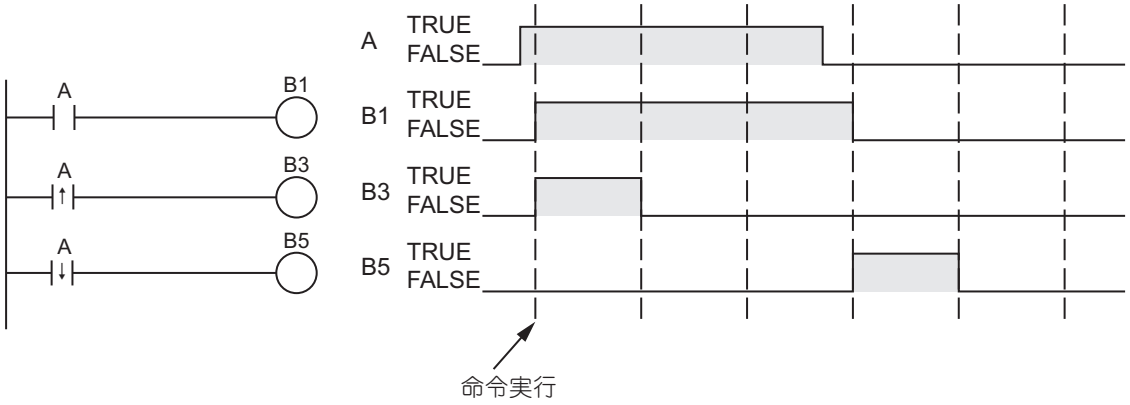
立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値	出力
LD	TRUE	TRUE
	FALSE	FALSE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と、現在の変数の値によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値	出力
LD	立ち上がり微分	前回実行時 FALSE → 現在 TRUE	TRUE
		上記以外	FALSE
	立ち下がり微分	前回実行時 TRUE → 現在 FALSE	TRUE
		上記以外	FALSE

記述例およびタイミングチャートを、以下に示します。



LDN

変数名で指定した BOOL 型変数の反転値を読み込み、後段に接続（出力）します。
指定した変数の値が TRUE なら FALSE を出力、FALSE なら TRUE を出力します。
母線から始まる最初の b 接点、または回路ブロックの最初の b 接点に使用します。

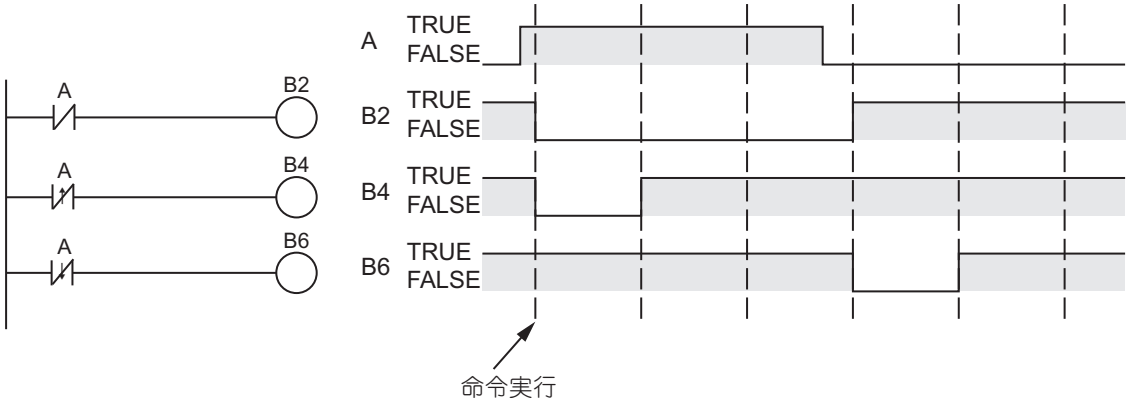
立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値	出力
LDN	TRUE	FALSE
	FALSE	TRUE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と、現在の変数の値によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値	出力
LDN	立ち上がり微分	前回実行時 FALSE → 現在 TRUE	FALSE
		上記以外	TRUE
	立ち下がり微分	前回実行時 TRUE → 現在 FALSE	FALSE
		上記以外	TRUE

記述例およびタイミングチャートを、以下に示します。



使用上の注意

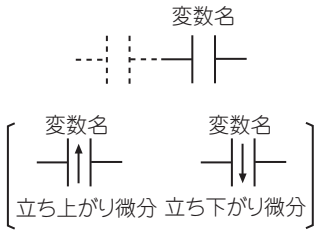
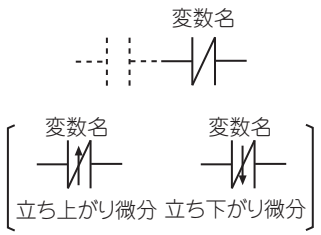
- つぎの場合、異常となり、本命令から後段回路への出力は不定となります。
  a）変数の値に配列要素を指定し、かつ、その要素が存在しない場合。

（例）BOOL 型配列 a[0..5]が定義されていて、変数 a[10]を指定して本命令を実行しようとした場合

- 上記のように、本命令から後段回路への出力が不定となる場合、以下のいずれかの対応を推奨します。
  a）配列領域を超えた要素の読み込みをしないよう、必ず領域に収まる要素番号を設定するプログラムにします。
  b）事前に要素番号をチェックし配列領域を超えている場合は、JMP 命令を使用するなどにより異常の発生を回避するプログラムにします。
- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。

AND / ANDN

AND　：BOOL 型変数の値と、入力条件との論理積をとります。
ANDN　：BOOL 型変数の反転値と、入力条件との論理積をとります。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AND	アンド	−		result:=vBool1 AND vBool2; result:=vBool1 & vBool2;
ANDN	アンドノット	−		result:=vBool1 AND NOT vBool2;

変数

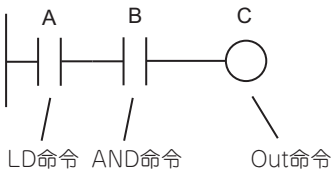
なし

機能

AND

変数名で指定した BOOL 型変数の値と、入力条件との論理積を、後段に接続（出力）します。
前段回路と直列接続する a 接点に使用します。

AND 命令の記述例を以下に示します。変数「A」と変数「B」の論理積を、変数「C」に出力します。





変数Aと変数Bの論理積を、変数Cに出力します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値と、入力条件の組み合わせ	出力
AND	変数の値: TRUE 入力条件: TRUE	TRUE
	上記以外	FALSE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と現在の変数の値、および入力条件によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値、および入力条件の組み合わせ	出力
AND	立ち上がり微分	変数の値: 前回実行時 FALSE → 現在 TRUE 入力条件: TRUE	TRUE
		上記以外	FALSE
	立ち下がり微分	変数の値: 前回実行時 TRUE → 現在 FALSE 入力条件: TRUE	TRUE
		上記以外	FALSE

ANDN

変数名で指定した BOOL 型変数の反転値と、入力条件との論理積を、後段に接続（出力）します。前段回路と直列接続する b 接点に使用します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値と、入力条件の組み合わせ	出力
ANDN	変数の値: FALSE 入力条件: TRUE	TRUE
	上記以外	FALSE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と現在の変数の値、および入力条件によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値、および入力条件の組み合わせ	出力
ANDN	立ち上がり微分	変数の値: 前回実行時 FALSE → 現在 TRUE 入力条件: TRUE	FALSE
		変数の値: 無関係 入力条件: FALSE	
		上記以外	TRUE
	立ち下がり微分	変数の値: 前回実行時 TRUE → 現在 FALSE 入力条件: TRUE	FALSE
		変数の値: 無関係 入力条件: FALSE	
		上記以外	TRUE

使用上の注意

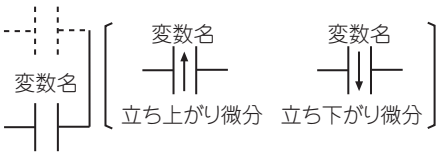
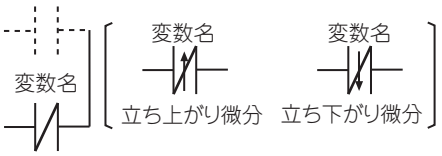
- つぎの場合、異常となり、ラダー図プログラムでは本命令から後段回路への出力は不定となります。なお、ST 言語では本命令を含む行を実行しません。詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「ST 言語で記述された POU での構文エラー発生時の動作」を参照してください。
  a）変数の値に配列要素を指定し、かつ、その要素が存在しない場合。
  （例）BOOL 型配列 a[0..5]が定義されていて、変数 a[10]を指定して本命令を実行しようとした場合
- 上記のように、ラダー図プログラムで本命令から後段回路への出力が不定となる場合、以下のいずれかの対応を推奨します。

a）配列領域を超えた要素の読み込みをしないよう、必ず領域に収まる要素番号を設定するラダー図プログラムにします。
b）事前に要素番号をチェックし配列領域を超えている場合は、JMP 命令を使用するなどにより異常の発生を回避するラダー図プログラムにします。

- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。
- 本命令は、母線に直接接続することはできません。

OR / ORN

OR　：BOOL 型変数の値と、入力条件との論理和をとります。
ORN ：BOOL 型変数の反転値と、入力条件との論理和をとります。

命令	名称	FB/FUN	グラフィック表現	ST 表現
OR	オア	−		result:=vBool1 OR vBool2;
ORN	オアノット	−		result:=vBool1 OR NOT vBool2;

変数

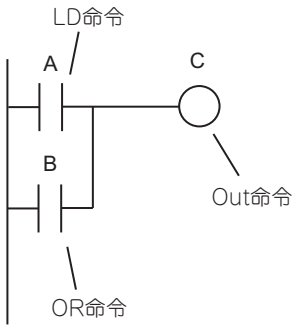
なし

機能

OR

変数名で指定した BOOL 型変数の値と、入力条件との論理和を、後段に接続（出力）します。
前段回路と並列接続する a 接点に使用します。母線に接続された、または回路ブロックの先頭の LD/LDN 命令から本命令直前までの回路との、OR（論理和）の a 接点を構成します。

OR 命令の記述例を以下に示します。変数「A」と変数「B」の論理和を、変数「C」に出力します。





変数Aと変数Bの論理和を、変数Cに出力します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値と入力条件の組み合わせ	出力
OR	変数の値: FALSE 入力条件: FALSE	FALSE
	上記以外	TRUE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と現在の変数の値、および入力条件によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値、および入力条件の組み合わせ	出力
OR	立ち上がり微分	変数の値: 前回実行時 FALSE → 現在 TRUE 入力条件: 無関係	TRUE
		変数の値: 無関係 入力条件: TRUE	
		上記以外	FALSE
	立ち下がり微分	変数の値: 前回実行時 TRUE → 現在 FALSE 入力条件: 無関係	TRUE
		変数の値: 無関係 入力条件: TRUE	
		上記以外	FALSE

ORN

変数名で指定した BOOL 型変数の反転値と、入力条件との論理和を、後段に接続（出力）します。前段回路と並列接続する b 接点に使用します。母線に接続された、または回路ブロックの先頭の LD/LDN 命令から本命令直前までの回路との、OR（論理和）の b 接点を構成します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	変数の値と入力条件の組み合わせ	出力
ORN	変数の値: TRUE 入力条件: FALSE	FALSE
	上記以外	TRUE

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時の変数の値と現在の変数の値、および入力条件によって、つぎのように動作します。

命令	微分指定	前回実行時の変数の値と現在の変数の値、および入力条件の組み合わせ	出力
ORN	立ち上がり微分	変数の値: 前回実行時 FALSE → 現在 TRUE 入力条件: FALSE	FALSE
		上記以外	TRUE
	立ち下がり微分	変数の値: 前回実行時 TRUE → 現在 FALSE 入力条件: FALSE	FALSE
		上記以外	TRUE

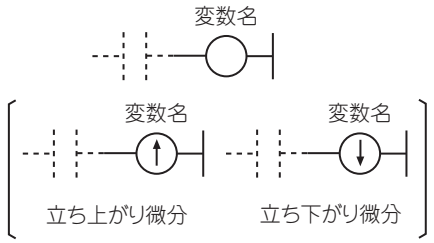
使用上の注意

- つぎの場合、異常となり、ラダー図プログラムでは本命令から後段回路への出力は不定となります。なお、ST 言語では本命令を含む行を実行しません。詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「ST 言語で記述された POU での構文エラー発生時の動作」を参照してください。
  a）変数の値に配列要素を指定し、かつ、その要素が存在しない場合。
  （例）BOOL 型配列 a[0..5]が定義されていて、変数 a[10]を指定して本命令を実行しようとした場合

- 上記のように、ラダー図プログラムで本命令から後段回路への出力が不定となる場合、以下のいずれかの対応を推奨します。
  a）配列領域を超えた要素の読み込みをしないよう、必ず領域に収まる要素番号を設定するラダー図プログラムにします。
  b）事前に要素番号をチェックし配列領域を超えている場合は、JMP 命令を使用するなどにより異常の発生を回避するラダー図プログラムにします。
- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。

Out / OutNot

Out　　：前段までの論理演算処理の結果を、BOOL 型変数に出力します。
OutNot ：前段までの論理演算処理の結果の反転値を、BOOL 型変数に出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Out	出力	−		変数名:=（前段までの論理演算式）;
OutNot	否定出力	−		変数名:=NOT（前段までの論理演算式）;

変数

なし

機能

Out

前段までの論理演算処理の結果を、変数名で指定した BOOL 型変数に出力します。

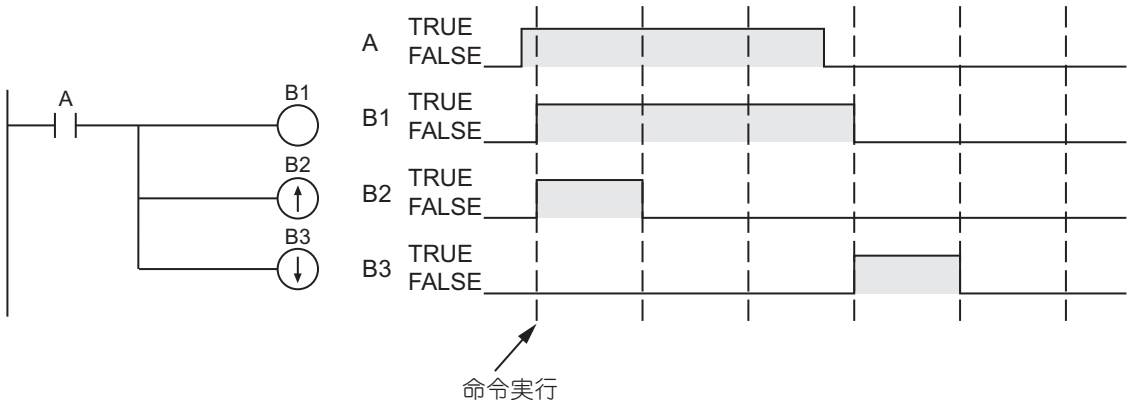
立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

前段までの論理演算処理の結果	出力
TRUE	TRUE
FALSE	FALSE

Out 命令には、立ち上がり微分、立ち下がり微分の動作指定をすることができます。その場合、前段までの論理演算処理の結果が、前回実行時と現在とでどう変化したかで、出力値が決まります。現在の前段までの論理演算処理の結果によって、つぎのように動作します。

微分指定	前回実行時の、前段までの論理演算処理の結果と、現在の、前段までの論理演算処理の結果	出力
立ち上がり微分	前回実行時 FALSE　→　現在 TRUE	TRUE
	上記以外	FALSE
立ち下がり微分	前回実行時 TRUE　→　現在 FALSE	TRUE
	上記以外	FALSE

記述例およびタイミングチャートを、以下に示します。

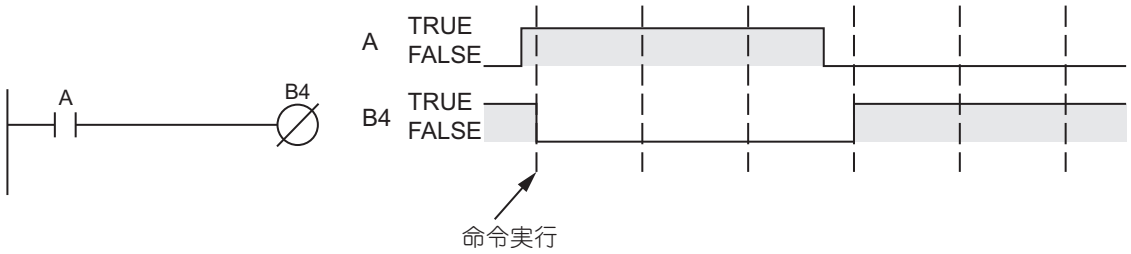


OutNot

前段までの論理演算処理の結果の反転値を、変数名で指定した BOOL 型変数に出力します。

前段までの論理演算処理の結果	出力
TRUE	FALSE
FALSE	TRUE

記述例およびタイミングチャートを、以下に示します。



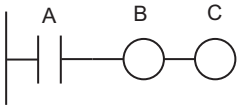
参考

Set/Reset 命令と Out/OutNot 命令の違い

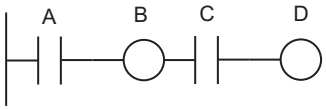
- Set/Reset 命令は、入力が TRUE になったときに動作し、入力が FALSE のときは動作しません。つまり、入力が FALSE のときは、出力に変化はありません。
- Out/OutNot 命令は、前段までの論理演算処理の結果が TRUE でも FALSE でも、それに応じた出力を行います。

使用上の注意

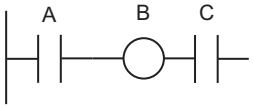
- つぎの場合、異常となり、出力を行いません。
  a）変数の値に配列要素を指定し、かつ、その要素が存在しない場合。
  （例）BOOL 型配列 a[0..5]が定義されていて、変数 a[10]を指定して本命令を実行しようとした場合
- 以下のような接続は可能です。
  a）Out 命令の後段に、別の Out 命令を接続。



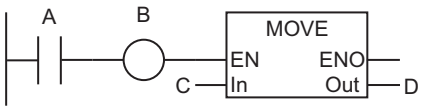
b）Out 命令の後段に、LD 命令と Out 命令を接続。



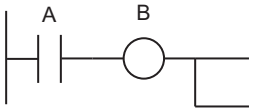
・以下のような接続はできません。
a）Out 命令の後段に、LD 命令だけを接続。

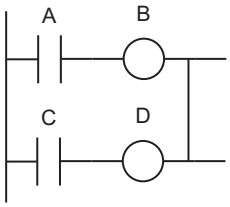


b）Out 命令の後段に、ファンクション、ファンクションブロックを接続。



c）Out 命令の後段での、分岐や合流。





ST 構文命令

命令	名称	ページ
IF	選択	P.2-28
CASE	多方向分岐	P.2-32
WHILE	繰り返し	P.2-36
REPEAT	繰り返し	P.2-38
EXIT	ループブレイク	P.2-40
RETURN	復帰	P.2-43
FOR	繰り返し	P.2-44

IF

指定した条件式の評価結果に応じて、二者択一的に、実行する文を選択します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
IF	選択	−	なし	IF 条件式 THEN 　文; ELSIF　条件式　THEN 　文; ELSE 　文; END_IF;

変数

なし

機能

指定した条件式の評価結果に応じて、二者択一的に実行する文を選択します。条件式には、評価結果として TRUE または FALSE の値を持つ以下のものを記述します。

条件式として記述できるもの	記述例	評価結果
論理式	a>3	変数「a」の値が 3 より大きければ TRUE、それ以外は FALSE
	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
BOOL 型変数	abc	「abc」の値が TRUE なら TRUE、FALSE なら FALSE
BOOL 型定数	TRUE	TRUE
BOOL 型の戻り値を持つファンクション	FUN 名	そのファンクションの戻り値が TRUE なら TRUE、FALSE なら FALSE

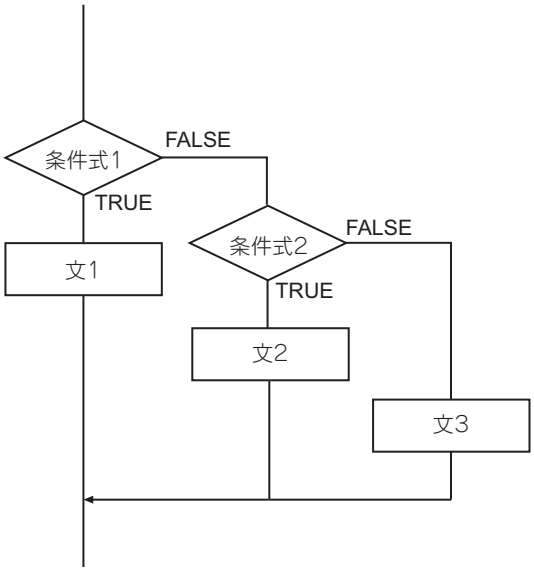
論理式には、以下の演算子が使用できます。

演算子	意味	記述例	評価結果
=	一致	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
<>	不一致	a<>b	変数「a」、「b」の値が等しくなければ TRUE、それ以外は FALSE
<	比較	a<b	変数「a」の値が「b」より小さければ TRUE,それ以外は FALSE
<=		a<=b	変数「a」の値が「b」以下なら TRUE、それ以外は FALSE
>		a>b	変数「a」の値が「b」より大きければ TRUE、それ以外は FALSE
>=		a>=b	変数「a」の値が「b」以上なら TRUE、それ以外は FALSE
AND, &	論理積	a AND b a & b	BOOL 型変数「a」、「b」の論理積
OR	論理和	a OR b	BOOL 型変数「a」、「b」の論理和
XOR	排他的論理和	a XOR b	BOOL 型変数「a」、「b」の排他的論理和
NOT	否定	NOT a	BOOL 型変数「a」の否定

下の記述の場合の処理フローは、条件式 1、条件式 2 の評価結果によって、図のようになります。文 n には、複数の文を記述することができます。

```
IF 条件式 1 THEN
    文 1;
ELSIF 条件式 2 THEN
    文 2;
```

```
ELSE
    文 3;
END_IF;
```



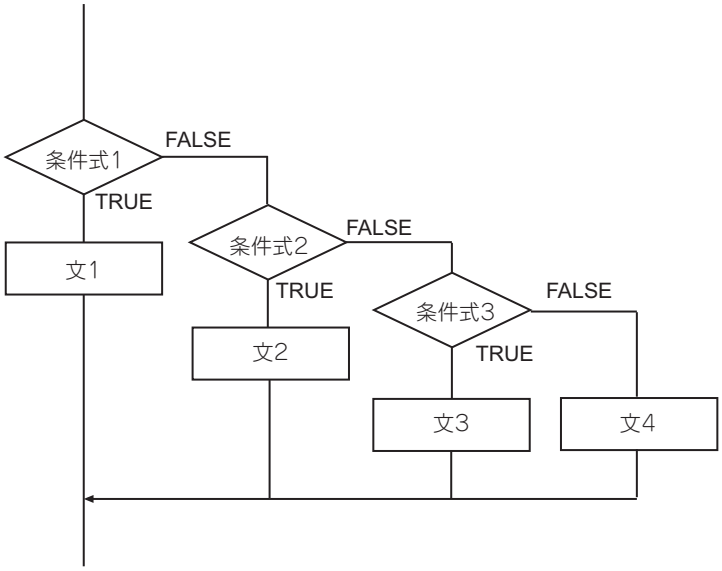
参考

- 本命令は、階層構造にすることが可能です。以下の記述の場合、文 11 は、条件式 1 の評価結果が TRUE で、かつ条件式 11 の評価結果も TRUE のときに実行されます。

```
IF 条件式 1 THEN
    IF 条件式 11 THEN
        文 11;
    ELSIF 条件式 12 THEN
        文 12;
    ELSE
        文 13;
    END_IF;
ELSIF 条件式 2 THEN
    文 2;
ELSE
    文 3;
END_IF;
```

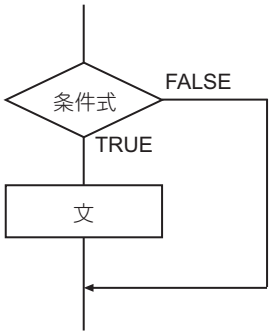
ELSIF は、複数記述することが可能です。以下の記述の場合、処理フローは図のようになります。

```
IF 条件式 1 THEN
    文 1;
ELSIF 条件式 2 THEN
    文 2;
ELSIF 条件式 3 THEN
    文 3;
ELSE
    文 4;
END_IF;
```



- 条件式が 1 つしかない場合、ELSIF は不要です。また、いずれの条件式にもあてはまらなければ何も処理をしない場合には、ELSE が不要です。以下の記述の場合、処理フローは図のようになります。

```
IF 条件式 THEN
    文;
END_IF;
```



- 文に書ける内容に、特に制限はありません。ファンクションブロックの呼び出しや FOR 命令など、IF 命令の外で書けることは何でも書けます。

使用上の注意

- IF と END_IF は、省略できません。また、IF と END_IF は必ず対にして、使用してください。
- IF 命令、CASE 命令、FOR 命令、WHILE 命令、REPEAT 命令を含めて、階層の深さは 15 までです。

サンプルプログラム

変数「abc」の値が INT#0 より小さければ変数「def」に INT#0 を代入します。「abc」の値が INT#0 ならば変数「def」に INT#1 を、変数「ghi」に INT#2 を代入します。「abc」の値がそのどちらでもなければ、変数「def」に INT#3 を代入します。

名称	データ型	初期値
abc	INT	0

名称	データ型	初期値
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

指定した整数式の値に応じて、複数の文の中から、実行する文を選択します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CASE	多方向分岐	−	なし	CASE 整数式 OF 　選択値: 　　文; 　選択値: 　　文; 　　: 　ELSE 　　文; END_CASE;

変数

なし

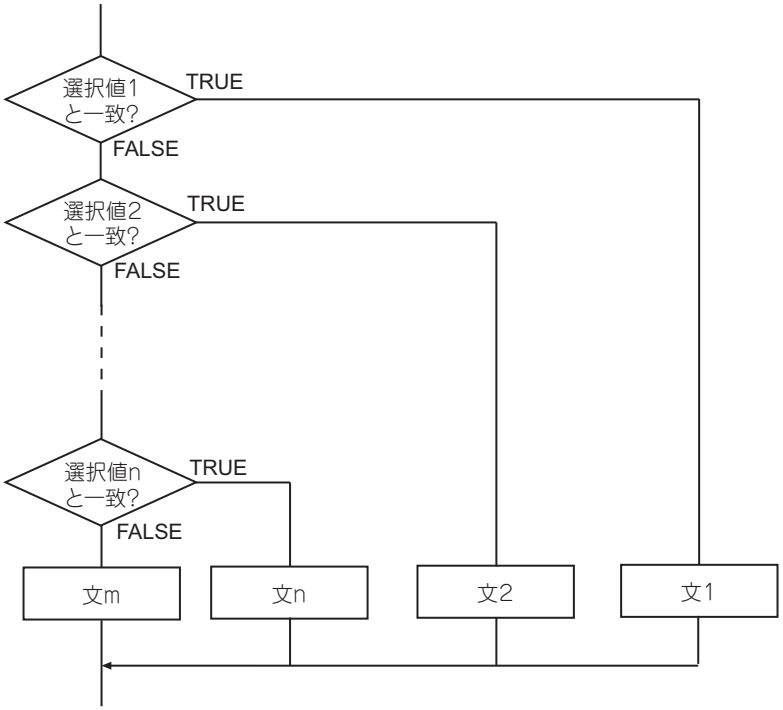
機能

指定した整数式の値に応じて、複数の文の中から、実行する文を選択します。
整数式と選択値には、以下のものを記述します。

	記述できるもの
整数式	整数型変数、整数型定数、整数型の式、整数型の戻り値を持つファンクション、列挙型変数、列挙型の式、列挙型の戻り値を持つファンクション
選択値	整数型定数

下の記述の場合の処理フローは、整数式の値によって、図のようになります。文と書かれているところには、複数の文を記述することができます。

```
CASE 整数式 OF
    1:
        文 1;
    2:
        文 2;
    :
    n:
        文 n;
    ELSE
        文 m;
END_CASE;
```



参考

- 本命令は、階層構造にすることが可能です。以下の記述の場合、文 12 は、整数式 1 の値が 1 で、かつ整数式 11 の値が 2 のときに実行されます。

```
CASE 整数式1 OF
    1:
        CASE 整数式11 OF
            1:
                文11;
            2:
                文12;
            ELSE
                文1m;
        END_CASE;
    2:
        文2;
    3:
        文3;
    ELSE
        文m;
END_CASE;
```

- 選択値には、複数の値を記述することが可能です。複数の値は、「,」（カンマ）で区切って記述します。以下の記述の場合、整数式の値が 1 または 2 のときに文 1 が実行されます。

```
CASE 整数式1 OF
    1,2:
        文1;
    3:
        文2;
```

```
    4:
        文 3;
    ELSE
        文 m;
END_CASE;
```

- 選択値に、連続値を記述することも可能です。連続値は、「..」（ピリオド 2 つ）で区切って記述します。以下の記述の場合、整数式の値が 10 以上 15 以下のときに文 1 が実行されます。

```
CASE 整数式 1 OF
    10..15:
        文 1;
    16:
        文 2;
    17:
        文 3;
    ELSE
        文 m;
END_CASE;
```

- ELSE は、省略可能です。その場合、整数式の値がいずれの選択値とも等しくなければ、どの文も実行されません。
- 文に書ける内容に、特に制限はありません。ファンクションブロックの呼び出しや FOR 命令など、CASE 命令の外で書けることは何でも書けます。
- C 言語の「switch」文とは、つぎの仕様が異なります。C 言語の「switch」文は、「break」文を使わない限り、整数式と等しい選択値以降の文がすべて実行されます。これに対して CASE 文は、整数式と等しい選択値に該当する文だけが実行されます。たとえば、以下の例の場合、C 言語の「switch」文は、文 1 から文 3 までがすべて実行されます。これに対して CASE 命令は、文 1 だけが実行されます。

C言語のswitch文

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

CASE命令

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

使用上の注意

- CASE と END_CASE は、省略できません。また、CASE と END_CASE は必ず対にして、使用してください。
- 整数式と選択値は、データ型が異なっていてもかまいません。
- 選択値には、同じ値が重複するような記述はできません。
- IF 命令、CASE 命令、FOR 命令、WHILE 命令、REPEAT 命令を含めて、階層の深さは 15 までです。

サンプルプログラム

変数「abc」の値が INT#1 なら変数「def」に INT#10 を代入、INT#2 なら INT#20 を代入、INT#3 なら INT#30 を代入します。そのいずれでもなければ変数「ghi」の値を代入します。

名称	データ型	初期値
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;
```

変数「abc」の値が INT#1 なら変数「def」に INT#10 を、INT#2 または INT#5 なら INT#20 を、INT#6 以上 INT#10 以下なら INT#30 を代入します。そのいずれでもない場合、何もしません。

名称	データ型	初期値
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

指定した条件式の評価結果が TRUE のあいだ、ある文を繰り返し実行します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
WHILE	繰り返し	−	なし	WHILE 条件式 DO 　文; END_WHILE;

変数

なし

機能

指定した条件式の評価結果が TRUE のあいだ、ある文を繰り返し実行します。条件式には、評価結果として TRUE または FALSE の値を持つ以下のものを記述します。

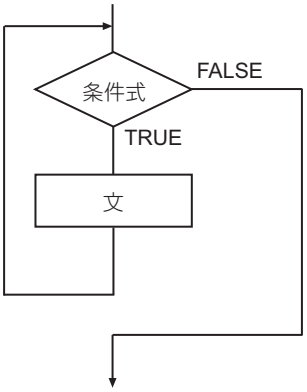
条件式として記述できるもの	記述例	評価結果
論理式	a>3	変数「a」の値が 3 より大きければ TRUE、それ以外は FALSE
	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
BOOL 型変数	abc	「abc」の値が TRUE なら TRUE、FALSE なら FALSE
BOOL 型定数	TRUE	TRUE
BOOL 型の戻り値を持つファンクション	FUN 名	そのファンクションの戻り値が TRUE なら TRUE、FALSE なら FALSE

論理式には、以下の演算子が使用できます。

演算子	意味	記述例	評価結果
=	一致	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
<>	不一致	a<>b	変数「a」、「b」の値が等しくなければ TRUE、それ以外は FALSE
<	比較	a<b	変数「a」の値が「b」より小さければ TRUE,それ以外は FALSE
<=		a<=b	変数「a」の値が「b」以下なら TRUE、それ以外は FALSE
>		a>b	変数「a」の値が「b」より大きければ TRUE、それ以外は FALSE
>=		a>=b	変数「a」の値が「b」以上なら TRUE、それ以外は FALSE
AND, &	論理積	a AND b a & b	BOOL 型変数「a」、「b」の論理積
OR	論理和	a OR b	BOOL 型変数「a」、「b」の論理和
XOR	排他的論理和	a XOR b	BOOL 型変数「a」、「b」の排他的論理和
NOT	否定	NOT a	BOOL 型変数「a」の否定

下の記述の場合の処理フローは、図のようになります。図中の文と書かれているところには、複数の文を記述することができます。

```
WHILE 条件式 DO
    文;
END_WHILE;
```



参考

- 条件式を最初に評価した結果が FALSE の場合、文は一度も実行されません。
- 文に書ける内容に、特に制限はありません。ファンクションブロックの呼び出しや FOR 命令など、WHILE 命令の外で書けることは何でも書けます。
- 繰り返し処理を中断するには、EXIT 命令を実行してください。その場合、EXIT 命令から END_WHILE 命令までの処理は実行されません。

使用上の注意

- WHILE と END_WHILE は、省略できません。また、WHILE と END_WHILE は必ず対にして、使用してください。
- IF 命令、CASE 命令、FOR 命令、WHILE 命令、REPEAT 命令を含めて、階層の深さは 15 までです。

サンプルプログラム

変数「abc」の値が INT#1000 以下であるあいだ、「abc」に INT#7 を加え続けます。

名称	データ型	初期値
abc	INT	0

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

REPEAT

ある文を一度実行し、指定した条件式が TRUE になるまで、その文を繰り返し実行します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
REPEAT	繰り返し	−	なし	REPEAT 　文; UNTIL　条件式 END_REPEAT;

変数

なし

機能

ある文を一度実行し、指定した条件式が TRUE になるまで、その文を繰り返し実行します。条件式には、評価結果として TRUE または FALSE の値を持つ以下のものを記述します。

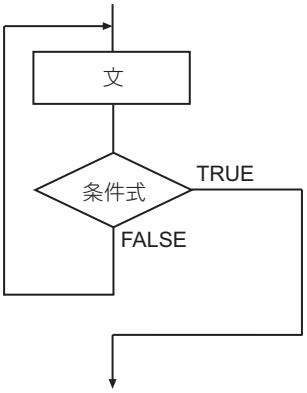
条件式として記述できるもの	記述例	評価結果
論理式	a>3	変数「a」の値が 3 より大きければ TRUE、それ以外は FALSE
	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
BOOL 型変数	abc	「abc」の値が TRUE なら TRUE、FALSE なら FALSE
BOOL 型定数	TRUE	TRUE
BOOL 型の戻り値を持つファンクション	FUN 名	そのファンクションの戻り値が TRUE なら TRUE、FALSE なら FALSE

論理式には、以下の演算子が使用できます。

演算子	意味	記述例	評価結果
=	一致	a=b	変数「a」、「b」の値が等しければ TRUE、それ以外は FALSE
<>	不一致	a<>b	変数「a」、「b」の値が等しくなければ TRUE、それ以外は FALSE
<	比較	a<b	変数「a」の値が「b」より小さければ TRUE,それ以外は FALSE
<=		a<=b	変数「a」の値が「b」以下なら TRUE、それ以外は FALSE
>		a>b	変数「a」の値が「b」より大きければ TRUE、それ以外は FALSE
>=		a>=b	変数「a」の値が「b」以上なら TRUE、それ以外は FALSE
AND, &	論理積	a AND b a & b	BOOL 型変数「a」、「b」の論理積
OR	論理和	a OR b	BOOL 型変数「a」、「b」の論理和
XOR	排他的論理和	a XOR b	BOOL 型変数「a」、「b」の排他的論理和
NOT	否定	NOT a	BOOL 型変数「a」の否定

下の記述の場合の処理フローは、図のようになります。図中の文と書かれているところには、複数の文を記述することができます。

```
REPEAT
    文;
UNTIL 条件式
END_REPEAT;
```



参考

- 文を一度実行してから条件式を評価します。したがって、文は必ず一度は実行されます。
- 文に書ける内容に、特に制限はありません。ファンクションブロックの呼び出しや FOR 命令など、REPEAT 命令の外で書けることは何でも書けます。
- 繰り返し処理を中断するには、EXIT 命令を実行してください。その場合、EXIT 命令から END_REPEAT 命令までの処理は実行されません。

使用上の注意

- REPEAT、UNTIL、END_REPEAT は、いずれも省略できません。また、REPEAT、UNTIL、END_REPEAT は必ず一組にして、使用してください。
- IF 命令、CASE 命令、FOR 命令、WHILE 命令、REPEAT 命令を含めて、階層の深さは 15 までです。

サンプルプログラム

変数「abc」の値が INT#10 より大きくなるまで、「abc」に INT#1 を加え続けます。

名称	データ型	初期値
abc	INT	0

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

EXIT

最も内側の、FOR 命令、WHILE 命令または REPEAT 命令の繰り返し処理を中断します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EXIT	ループブレイク	－	なし	FOR Index:=0 TO 9 BY 1 DO IF Error[Index] THEN EXIT; END_IF; END_FOR;

変数

なし

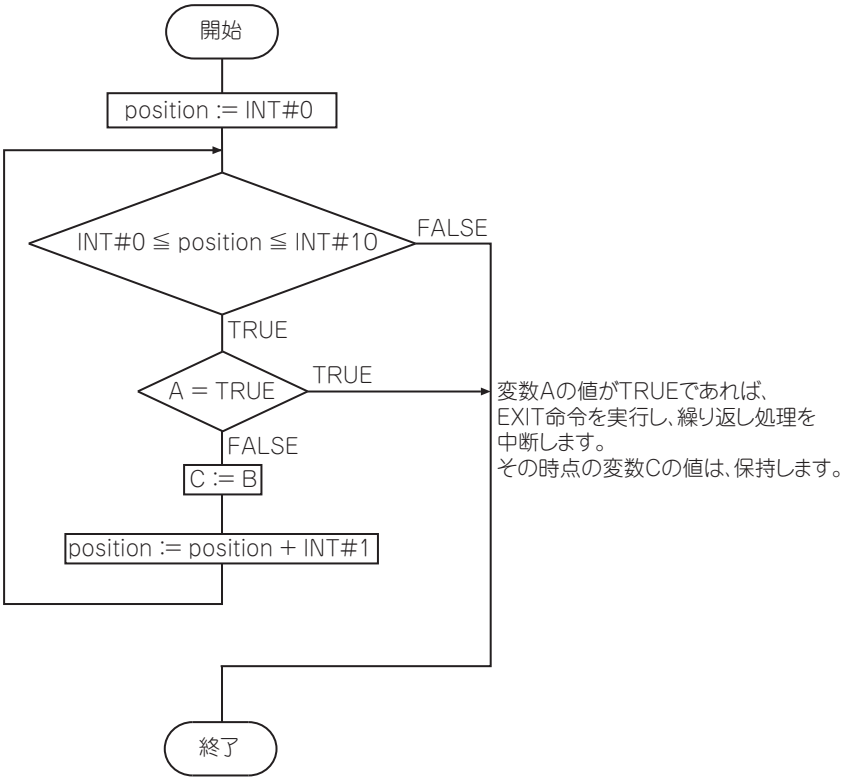
機能

最も内側の、FOR 命令、WHILE 命令または REPEAT 命令の繰り返し処理を中断します。処理は、繰り返し処理のつぎに移ります。

以下のユーザプログラムの場合、FOR 命令の繰り返し処理を実行中、毎回、変数「A」の値を確認します。変数「A」の値が TRUE であれば、EXIT 命令を実行し、繰り返し処理を終了します。その場合、END_IF のつぎの「C:=B;」は実行されず、変数「C」は値を保持します。

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

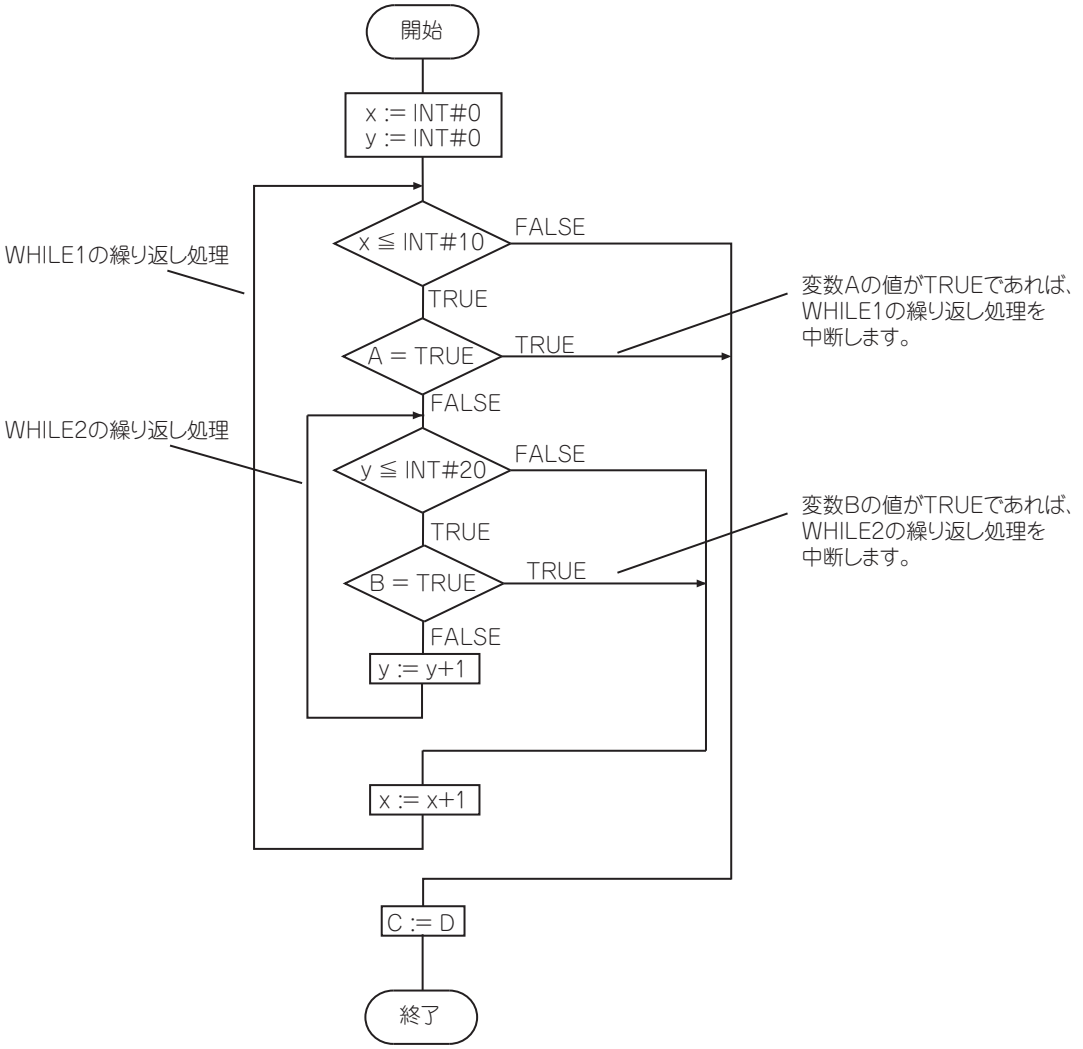
フローチャートは、以下のようになります。



本命令の実行により中断するのは、最も内側の繰り返し処理だけです。したがって、以下のユーザプログラムの場合、変数「B」の値が TRUE であれば EXIT 命令 2 を実行し、WHILE 命令 2 の繰り返し処理を中断します。その結果、処理は「x:=x+1;」に移ります。この場合、1 階層上の WHILE 命令 1 の繰り返し処理は続行します。
変数「A」の値が TRUE であれば EXIT 命令 1 を実行し、WHILE 命令 1 の繰り返し処理を中断します。その結果、処理は「C:=D;」に移ります。

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

フローチャートは、以下のようになります。



使用上の注意

- 本命令は、必ず FOR と END_FOR の間、WHILE と END_WHILE の間、または REPEAT と END_REPEAT の間で使用してください。
- 繰り返し処理を階層的に使用している（ネスティング）場合、すべての繰り返し処理を中断するには、階層の数だけ本命令が必要です。

RETURN

プログラム、ファンクション、またはファンクションブロックを終了します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RETURN	復帰	－	なし	RETURN;

変数

なし

機能

プログラム、ファンクション、またはファンクションブロックを終了します。

- 本命令をファンクションまたはファンクションブロック内で実行した場合は、ファンクションまたはファンクションブロックを終了し、処理を呼出元に戻します。
- 本命令をセクションがあるプログラム内で実行した場合は、セクションとプログラムを終了し、つぎのプログラムを実行します。
- 本命令をセクションがないプログラム内で実行した場合は、プログラムを終了し、つぎのプログラムを実行します。

以下のユーザプログラムの場合、RETURN 命令が実行されると、そのつぎにある SR 命令は実行されません。

```
RETURN;
SR_Instance(Set1:=a, Reset:=b, Q1=>abc);
```

使用上の注意

- 本命令をファンクションまたはファンクションブロック内で使用する場合は、本命令を実行する前に、当該 POU の戻り値、出力変数、「ENO」の値をセットしてください。
- 本命令をプログラム内で実行すると、当該プログラムを終了し、つぎのプログラムを実行します。注意してください。
- 本命令を多用すると、処理の流れがわかりにくくなります。注意してください。
- ラダー図プログラムの RETURN 命令は、シーケンス制御命令の、「RETURN (P.2-71)」命令を参照してください。

FOR

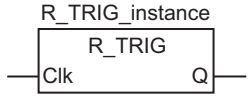
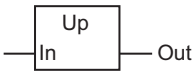
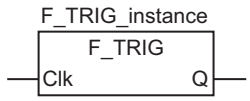
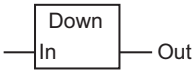
FOR 命令の説明は、シーケンス制御命令の、「FOR / NEXT (P.2-86)」命令を参照してください。

シーケンス入力命令

命令	名称	ページ
R_TRIG, Up / F_TRIG, Down	立ち上がり微分／立ち下がり微分	P.2-46
TestABit / TestABitN	ビットテスト／ビットテスト否定	P.2-50

R_TRIG, Up / F_TRIG, Down

R_TRIG, Up　：入力信号の立ち上がり（FALSE→TRUE）時、1 タスク周期だけ TRUE を出力します。
F_TRIG, Down ：入力信号の立ち下がり（TRUE→FALSE）時、1 タスク周期だけ TRUE を出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
R_TRIG	立ち上がり微分	FB		R_TRIG_instance(Clk, Q);
Up		FUN		なし
F_TRIG	立ち下がり微分	FB		F_TRIG_instance(Clk, Q);
Down		FUN		なし

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Clk, In	入力信号	入力	入力信号	データ型に従う	−	−
Q, Out	出力信号	出力	出力信号	データ型に従う	−	−

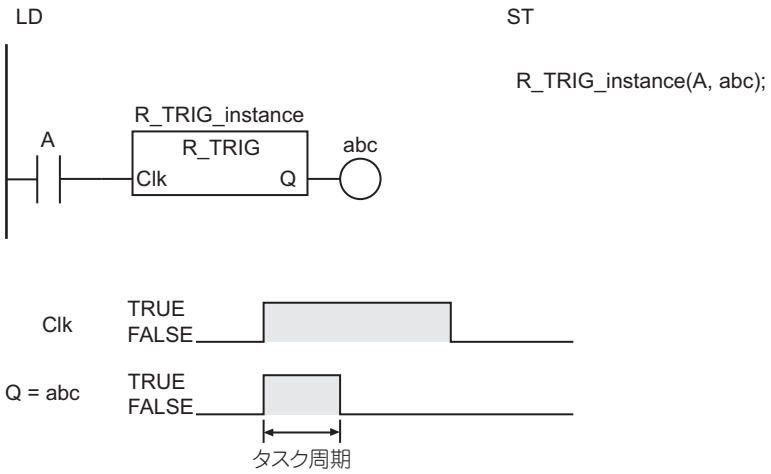
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Clk, In	○																			
Q, Out	○																			

機能

R_TRIG

入力信号「Clk」の立ち上がり（FALSE→TRUE）時、1 タスク周期だけ、出力信号「Q」に TRUE を代入します。それ以外は、「Q」の値は FALSE になります。
R_TRIG 命令と Up 命令の機能は同じです。

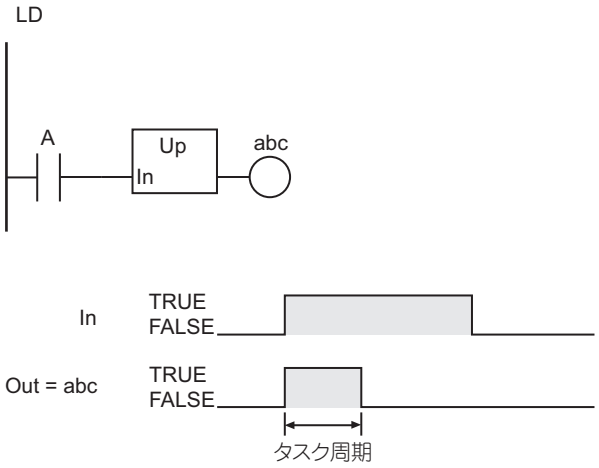
以下に、記述例とタイミングチャートを示します。



Up

入力信号「In」の立ち上がり（FALSE→TRUE）時、1 タスク周期だけ、出力信号「Out」に TRUE を代入します。それ以外は、「Out」の値は FALSE になります。
R_TRIG 命令と Up 命令の機能は同じです。
ただし、本命令を最初に実行するタスク周期での動作は R_TRIG 命令の動作と異なります。本命令を最初に実行するタスク周期での動作は、「使用上の注意 (P.2-48)」を参照してください。

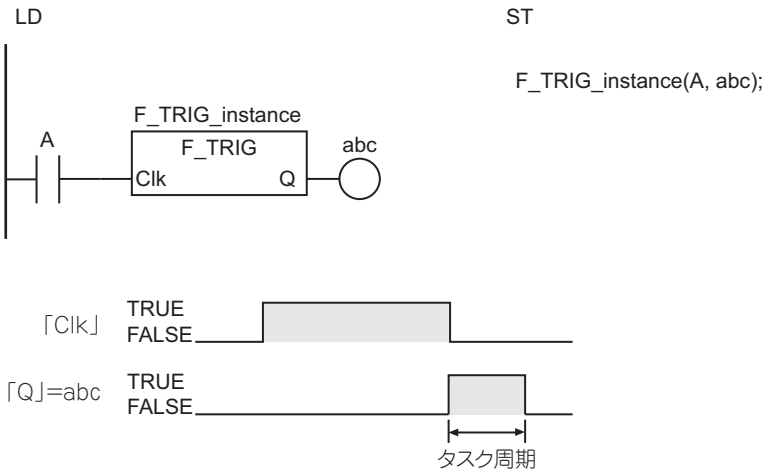
以下に、記述例とタイミングチャートを示します。



F_TRIG

入力信号「Clk」の立ち下がり（TRUE→FALSE）時、1 タスク周期だけ、出力信号「Q」に TRUE を代入します。それ以外は、「Q」の値は FALSE になります。
F_TRIG 命令と Down 命令の機能は、まったく同じです。

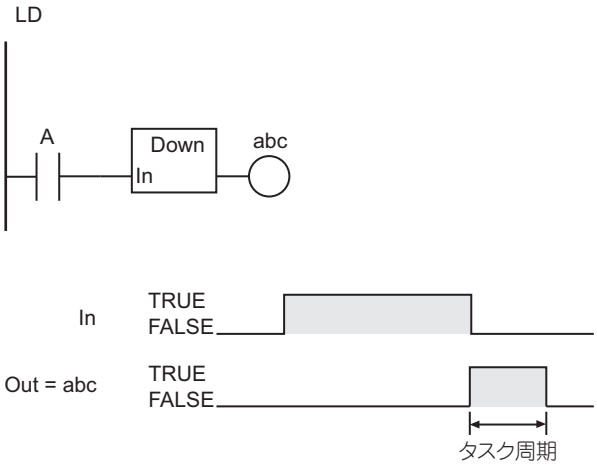
以下に、記述例とタイミングチャートを示します。



Down

入力信号「In」の立ち下がり（TRUE→FALSE）時、1 タスク周期だけ、出力信号「Out」に TRUE を代入します。それ以外は、「Out」の値は FALSE になります。
F_TRIG 命令と Down 命令の機能は、まったく同じです。

以下に、記述例とタイミングチャートを示します。



使用上の注意

- 立ち上がり／立ち下がり検知は、前回の本命令実行時と現在の「Clk」または「In」の値の違いで行います。したがって、JMP 命令などを使用して本命令を毎回のタスク周期では実行しない場合は、注意してください。
- 電源断が発生しても、「Clk」または「In」の値が FALSE になったとは認識しません。「Clk」、「In」の値が FALSE になったと認識するのは、「Clk」、「In」の値が FALSE の状態で、本命令が「Clk」、「In」の値を評価したときだけです。
- Up 命令を最初に実行したタスク周期では、「In」の値に関係なく、「Out」の値は必ず FALSE になります。
- 電源 ON 時に、Up 命令の「In」の値が TRUE の場合は、「In」の値がいったん FALSE になってから TRUE になるまで、「Out」の値は FALSE になります。
- R_TRIG 命令を最初に実行するタスク周期では、「Clk」の値が TRUE の場合は、「Q」の値は TRUE になります。

- F_TRIG 命令を最初に実行するタスク周期では、「Clk」の値に関係なく、「Q」の値は必ず FALSE になります。
- 電源 ON 時に、R_TRIG 命令の「Clk」の値が TRUE の場合は、「Q」の値は TRUE になります。
- 電源 ON 時に、F_TRIG 命令の「Clk」の値が FALSE の場合は、「Clk」の値がいったん TRUE になってから FALSE になるまで、「Q」の値は FALSE になります。
- Down 命令を最初に実行したタスク周期では、「In」の値に関係なく、「Out」の値は必ず FALSE になります。
- 電源 ON 時に、Down 命令の「In」の値が FALSE の場合は、「In」の値がいったん TRUE になってから FALSE になるまで、「Out」の値は FALSE になります。

TestABit / TestABitN

TestABit　：ビット列の、指定ビットの値を出力します。
TestABitN　：ビット列の、指定ビットの反転値を出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TestABit	ビットテスト	FUN	(@)TestABit EN In Pos Out	Out:=TestABit (In, Pos);
TestABitN	ビットテスト否定	FUN	(@)TestABitN EN In Pos Out	Out:=TestABitN (In, Pos);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	ビット列	入力	ビット列	データ型に従う	－	*1
Pos	ビット位置		指定ビット位置	0～「In」のビット数-1		0
Out	ビット値	出力	• TestABit 指定ビットの値 • TestABitN 指定ビットの反転値	データ型に従う	－	－

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Out	○																			

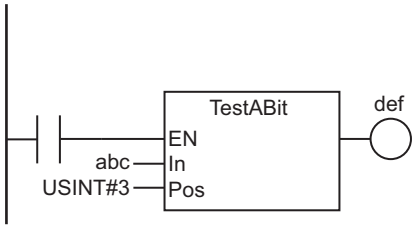
機能

TestABit

「EN」が TRUE のとき、ビット列「In」の、ビット位置「Pos」の値を、ビット値「Out」に代入します。
EN が FALSE のとき、「Out」の値は FALSE になります。

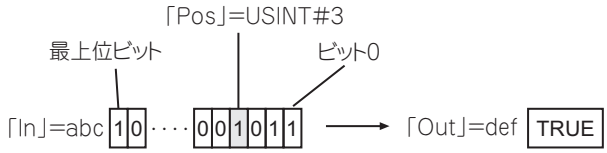
「Pos」＝「USINT#3」の場合の例を、以下に示します。

LD



ST

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



TestABitN

「EN」が TRUE のとき、ビット列「In」の、ビット位置「Pos」の反転値を、ビット値「Out」に代入します。
EN が FALSE のとき、「Out」の値は FALSE になります。

使用上の注意

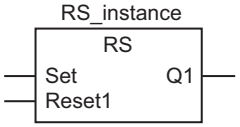
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- つぎの場合、異常となり、「Out」の値は FALSE になります。
  a)「Pos」の値が、「In」のビット数-1 より大きい場合。

シーケンス出力命令

命令	名称	ページ
RS	リセット優先キープ	P.2-54
SR	セット優先キープ	P.2-57
Set / Reset	セット／リセット	P.2-60
SetBits / ResetBits	多ビットセット／多ビットリセット	P.2-63
SetABit / ResetABit	1 ビットセット／1 ビットリセット	P.2-65
OutABit	1 ビットアウト	P.2-67

RS

BOOL 型変数の値を保持します。セット入力とリセット入力が同時に TRUE の場合、リセット入力が優先されます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RS	リセット優先キープ	FB		RS_instance(Set, Reset1, Q1);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Set *1	セット	入力	セット入力	データ型に従う	−	0
Reset1 *1	リセット		リセット入力			
Q1	キープ	出力	キープ出力	データ型に従う	−	−

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Set」を「S」、「Reset1」を「R1」と記述することができます。
たとえば、RS_instance(S:=A, R1:=B, Q1=>abc);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set	○																			
Reset1	○																			
Q1	○																			

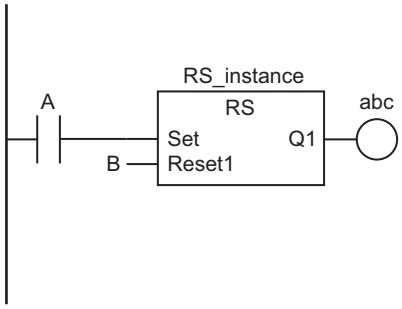
機能

リセット優先の自己保持回路です。入出力の関係を、以下に示します。

「Set」の値	「Reset1」の値	「Q1」の値
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	保持

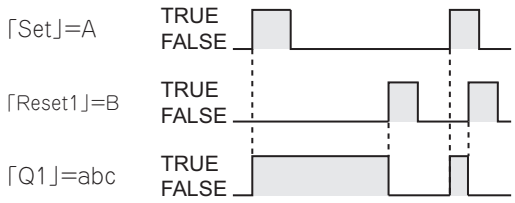
記述例とタイミングチャートを、以下に示します。

LD



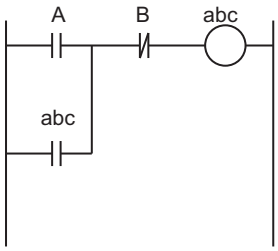
ST

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



参考

- RS 命令は、下に示す自己保持回路と同じ振る舞いをします。



- ただし、RS 命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合は、上記の自己保持回路と異なる振る舞いをします。

回路	B の値	abc の値
RS 命令	TRUE	保持
	FALSE	FALSE
自己保持回路	TRUE	FALSE
	FALSE	

使用上の注意

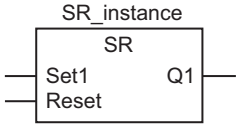
- 「Reset1」入力を、b 接点の外部機器から直接取り込まないでください。AC 電源断や瞬時停電のときは、コントローラの内部電源がすぐに OFF せず、入力ユニットの入力が先に ON となる場合があります。その結果、「Reset1」入力の値が TRUE となってしまいます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Q1」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「Q1」は、前回実行時の値を保持します。
- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、つぎのように動作します。

a)「Reset1」の値が TRUE ならば「Q1」の値を保持し、FALSE ならば「Q1」の値は FALSE になります。
b)「Q1」の後段に接続している命令には、「Q1」の値が TRUE であっても、FALSE が入力されます。

・「Q1」に保持属性ありのパラメータを接続しても、電源断発生時には値が保持されません。電源復帰後に動作モードを運転モードにして本命令を実行した時点で、「Q1」の値は FALSE になります。ただし、「参考 (P.2-55)」の自己保持回路の場合は、電源復帰後も値が保持されます。

SR

BOOL 型変数の値を保持します。セット入力とリセット入力が同時に TRUE の場合、セット入力が優先されます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SR	セット優先キープ	FB		SR_instance(Set1, Reset, Q1);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Set1 *1	セット	入力	セット入力	データ型に従う	−	0
Reset	リセット		リセット入力			
Q1	キープ	出力	キープ出力	データ型に従う	−	−

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Set1」を「S1」、「Reset」を「R」と記述することができます。
たとえば、SR_instance(S1:=A, R:=B, Q1=>abc);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set1	○																			
Reset	○																			
Q1	○																			

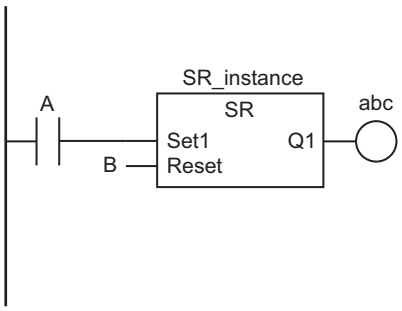
機能

セット優先の自己保持回路です。入出力の関係を、以下に示します。

「Set1」の値	「Reset」の値	「Q1」の値
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	保持

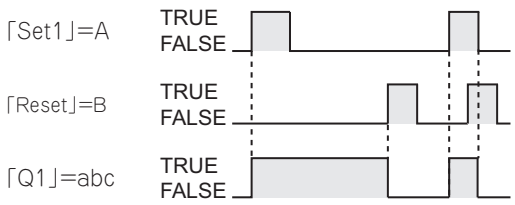
記述例とタイミングチャートを、以下に示します。

LD



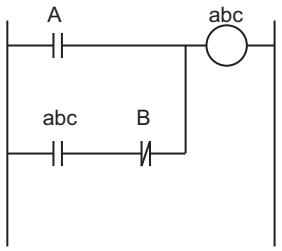
ST

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



参考

- SR 命令は、下に示す自己保持回路と同じ振る舞いをします。



- ただし、SR 命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合は、上記の自己保持回路と異なる振る舞いをします。

回路	B の値	abc の値
SR 命令	TRUE	保持
	FALSE	FALSE
自己保持回路	TRUE	FALSE
	FALSE	

使用上の注意

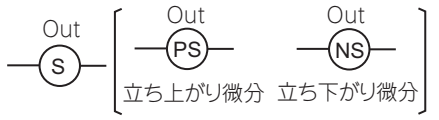
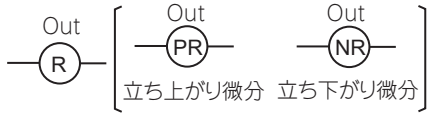
- 「Reset」入力を、b 接点の外部機器から直接取り込まないでください。AC 電源断や瞬時停電のときは、コントローラの内部電源がすぐに OFF せず、入力ユニットの入力が先に ON となる場合があります。その結果、「Reset」入力の値が TRUE となってしまいます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Q1」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「Q1」は、前回実行時の値を保持します。
- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、つぎのように動作します。

a)「Reset」の値が TRUE ならば「Q1」の値を保持し、FALSE ならば「Q1」の値は FALSE になります。
b)「Q1」の後段に接続している命令には、「Q1」の値が TRUE であっても、FALSE が入力されます。

・「Q1」に保持属性ありのパラメータを接続しても、電源断発生時には値が保持されません。電源復帰後に動作モードを運転モードにして本命令を実行した時点で、「Q1」の値は FALSE になります。ただし、参考欄の自己保持回路の場合は、電源復帰後も値が保持されます。

Set / Reset

Set　：BOOL 型変数に TRUE をセットします。
Reset ：BOOL 型変数に FALSE をセットします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Set	セット	−		なし
Reset	リセット	−		なし

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	出力	出力	出力	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			

機能

Set

本命令への入力が TRUE の場合、「Out」に TRUE をセットします。
いったん「Out」を TRUE にしたら、入力が FALSE になっても、「Out」は FALSE になりません。
「Out」を FALSE に戻すには、Reset 命令を使用します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	入力	出力
Set	TRUE	TRUE
	FALSE	保持

立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時と現在の入力の値によって、つぎのように動作します。

命令	微分指定	前回実行時と現在の入力の値	出力
Set	立ち上がり微分	前回実行時 FALSE → 現在 TRUE	TRUE
		上記以外	保持
	立ち下がり微分	前回実行時 TRUE → 現在 FALSE	TRUE
		上記以外	保持

Reset

本命令への入力が TRUE の場合、「Out」に FALSE をセットします。
いったん「Out」を FALSE にしたら、入力が FALSE になっても、「Out」は TRUE になりません。
「Out」を TRUE に戻すには、Set 命令を使用します。

立ち上がり微分、立ち下がり微分の指定がない場合の動作は、つぎのようになります。

命令	入力	出力
Reset	TRUE	FALSE
	FALSE	保持

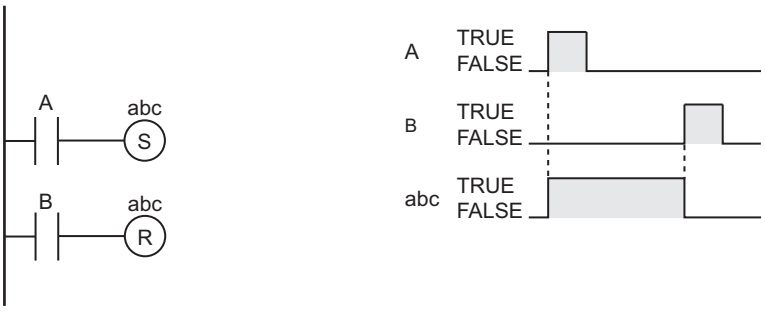
立ち上がり微分、立ち下がり微分の指定がある場合は、前回実行時と現在の入力の値によって、つぎのように動作します。

命令	微分指定	前回実行時と現在の入力の値	出力
Reset	立ち上がり微分	前回実行時 FALSE → 現在 TRUE	FALSE
		上記以外	保持
	立ち下がり微分	前回実行時 TRUE → 現在 FALSE	FALSE
		上記以外	保持

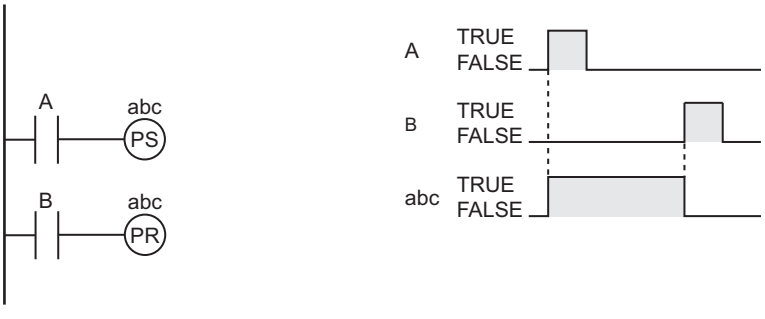
記述例とタイミングチャート

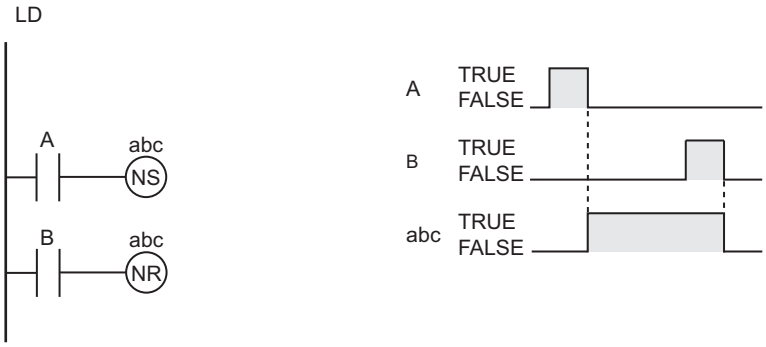
記述例とタイミングチャートを、以下に示します。

LD



LD





参考

Set/Reset 命令と Out 命令の違い

- Set/Reset 命令は、入力が TRUE になったときに動作し、入力が FALSE のときは動作しません。つまり、入力が FALSE のときは、出力に変化はありません。
- Out 命令は、前段までの結果が TRUE のときに指定変数を TRUE にし、前段までの結果が FALSE のときに指定変数を FALSE にします。つまり、入力が TRUE でも FALSE でも動作します。

Set/Reset 命令と SR/RS 命令の違い

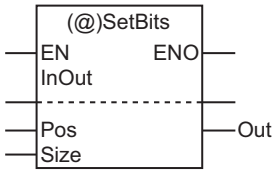
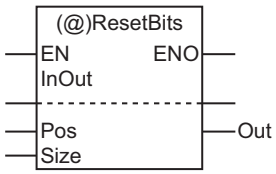
SR/RS 命令は、「Set」入力と「Reset」入力をプログラム内の同じ位置に記述しなければなりませんが、Set/Reset 命令は、それらを別の位置に記述することができます。

使用上の注意

- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、「Out」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「Out」の値を保持します。
- 立ち上がり微分指定ありで電源起動時に入力が TRUE の場合、そのままでは本命令は実行されません。入力がいったん FALSE になり、再度 TRUE になったときに実行されます。
- 立ち上がり微分指定なしで電源起動時に入力が TRUE の場合、入力をいったん FALSE にしなくても、本命令は実行されます。

SetBits / ResetBits

SetBits　：ビット列データの連続複数ビットに、TRUE をセットします。
ResetBits ：ビット列データの連続複数ビットに、FALSE をセットします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SetBits	多ビットセット	FUN		SetBits(InOut, Pos, Size);
ResetBits	多ビットリセット	FUN		ResetBits(InOut, Pos, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	ビット列	入出力	ビット列	データ型に従う	−	−
Pos	ビット位置	入力	指定ビットの位置	0〜「InOut」のビット数-1	−	0
Size	ビット数		ビット数	0〜「InOut」のビット数		1
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		○	○	○	○															
Pos						○														
Size						○														
Out	○																			

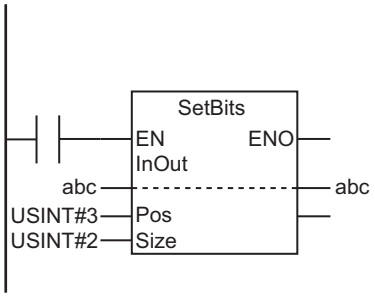
機能

SetBits

ビット列「InOut」の、ビット位置「Pos」から「Size」個のビットの値を、TRUE にセットします。それ以外のビットは変化しません。

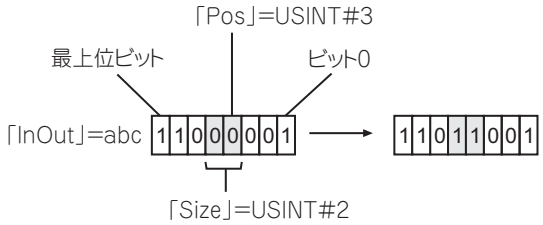
「Pos」＝「USINT#3」、「Size」＝「USINT#2」の場合の例を、以下に示します。

LD



ST

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



ResetBits

ビット列「InOut」の、ビット位置「Pos」から「Size」個のビットの値を、FALSE にセットします。それ以外のビットは変化しません。

参考

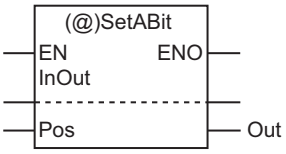
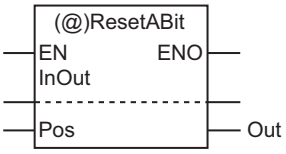
本命令を使用すると、データメモリなどのチャネル（ワード）単位でデータを扱うメモリエリアに AT 指定した変数に対して、一括に TRUE/FALSE をセット／リセットすることができます。

使用上の注意

- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、「InOut」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「InOut」の値を保持します。
- 「Size」の値が「0」の場合、「InOut」の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、「InOut」は変化しません。
  a）「Pos」の値が、「InOut」のビット数-1 より大きい場合。
  b）「Size」の値が、有効範囲外の場合。
  c）「Pos」と「Size」の指定が、「InOut」のビット数を超える場合。

SetABit / ResetABit

SetABit　：ビット列データの指定ビットに、TRUE をセットします。
ResetABit ：ビット列データの指定ビットに、FALSE をセットします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SetABit	1 ビットセット	FUN		SetABit(InOut, Pos);
ResetABit	1 ビットリセット	FUN		ResetABit(InOut, Pos);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	ビット列	入出力	ビット列	データ型に従う	−	−
Pos	ビット位置	入力	指定ビットの位置	0～「InOut」のビット数-1	−	0
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		○	○	○	○															
Pos						○														
Out	○																			

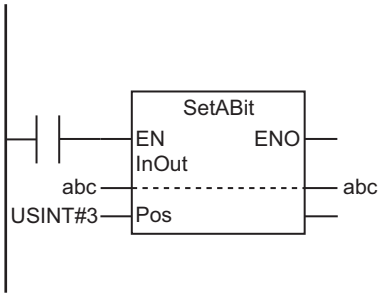
機能

SetABit

ビット列「InOut」の、ビット位置「Pos」の値を、TRUE にセットします。それ以外のビットは、変化しません。
実行後に「EN」が FALSE になっても、「InOut」の「Pos」ビットは変化しません。

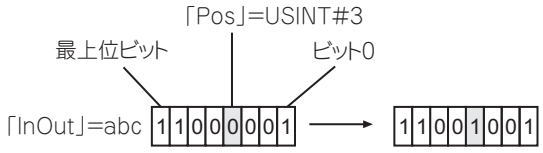
「Pos」＝「USINT#3」の場合の例を、以下に示します。

LD



ST

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



ResetABit

ビット列「InOut」の、ビット位置「Pos」の値を、FALSE にセットします。それ以外のビットは、変化しません。
実行後に「EN」が FALSE になっても、「InOut」の「Pos」ビットは変化しません。

参考

SetABit/ResetABit 命令と OutABit 命令の違い

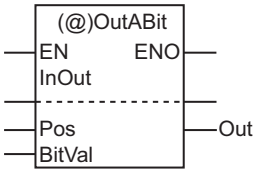
- SetABit/ResetABit 命令は、指定ビットにセットする値が、TRUE または FALSE に固定されています。
- OutABit 命令は、指定ビットにセットする値を、動的に変えることができます。

使用上の注意

- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、「InOut」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「InOut」の値を保持します。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、「InOut」は変化しません。
  a)「Pos」の値が、「In」のビット数-1 より大きい場合。

OutABit

ビット列データの指定ビットに、TRUE/FALSE をセットします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
OutABit	1 ビットアウト	FUN		OutABit(InOut, Pos, BitVal);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	ビット列	入出力	ビット列	データ型に従う	−	−
Pos	ビット位置	入力	指定ビットの位置	0〜「InOut」のビット数-1	−	0
BitVal	セット値		セットする値	データ型に従う		TRUE
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

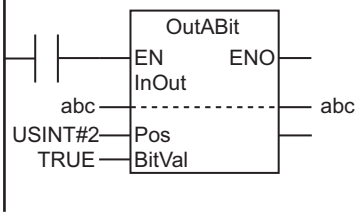
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		○	○	○	○															
Pos						○														
BitVal	○																			
Out	○																			

機能

ビット列「InOut」の、ビット位置「Pos」に、セット値「BitVal」の値を格納します。「Pos」以外のビット位置の値は、変化しません。

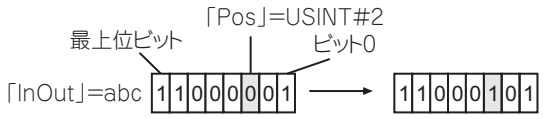
「Pos」＝「USINT#2」、「BitVal」＝TRUE の場合の例を、以下に示します。

LD



ST

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



参考

SetABit/ResetABit 命令と OutABit 命令の違い

- SetABit/ResetABit 命令は、指定ビットにセットする値が、TRUE または FALSE に固定されています。
- OutABit 命令は、指定ビットにセットする値を、「BitVal」の値を変えることによって動的に変えることができます。

使用上の注意

- 本命令がマスタコントロール領域にあって、マスタコントロールによるリセットが行われた場合、「InOut」の値を保持します。
- ジャンプ系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「InOut」の値を保持します。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、「InOut」は変化しません。
  a)「Pos」の値が、「InOut」のビット数-1 より大きい場合。

シーケンス制御命令

命令	名称	ページ
End	エンド	P.2-70
RETURN	復帰	P.2-71
MC / MCR	マスタコントロール開始／マスタコントロール終了	P.2-72
JMP	ジャンプ	P.2-84
FOR / NEXT	繰り返し開始／繰り返し終了	P.2-86
BREAK	ループブレイク	P.2-93

End

プログラムの、現在のタスク周期での実行を終了します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
End	エンド	FUN	End EN ENO	なし

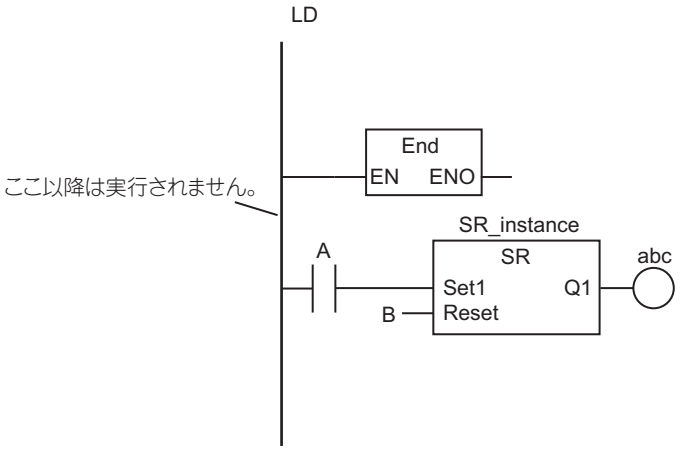
変数

なし

機能

プログラムの、現在のタスク周期での実行を終了します。

記述例を、以下に示します。この例の場合、End 命令が実行されると、そのつぎにある SR 命令は実行されません。

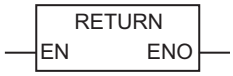


使用上の注意

- 本命令は、プログラム内だけで、使用してください。
- 本命令を、ファンクション、ファンクションブロック、インライン ST 内で使用した場合、ビルド時に異常になります。
- 本命令は、必ず直接左母線に接続してください。

RETURN

ファンクションまたはファンクションブロックを終了し、処理を呼出元に戻します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RETURN	復帰	FUN		なし

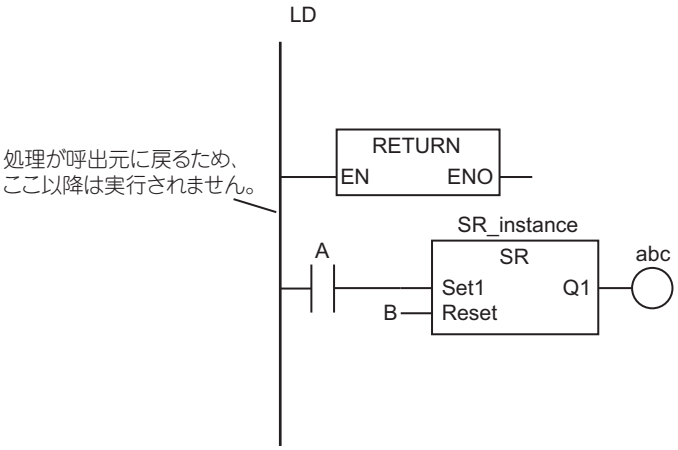
変数

なし

機能

ファンクションまたはファンクションブロックを終了し、処理を呼出元に戻します。

記述例を、以下に示します。この例の場合、RETURN 命令が実行されると、そのつぎにある SR 命令は実行されません。

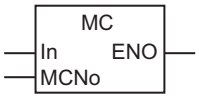
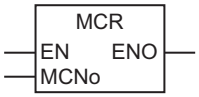


使用上の注意

- 本命令を、ラダー図プログラムで使用する場合、以下のことに注意してください。
  a）本命令は、ファンクションまたはファンクションブロック内で使用してください。プログラム内で使用すると、ビルド時に異常になります。
  b）本命令は、必ず左母線に直接接続してください。
- 本命令を実行する前に、当該 POU の戻り値、出力変数、「ENO」の値をセットしてください。
- 本命令を多用すると、処理の流れがわかりにくくなります。注意してください。
- ST プログラムやインライン ST の RETURN 命令は、ST 構文命令の、「RETURN (P.2-43)」命令を参照してください。

MC / MCR

MC　：マスタコントロール領域の開始点を示し、マスタコントロールによるリセットを実行します。
MCR ：マスタコントロール領域の終了点を示します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MC	マスタコントロール開始	–		なし
MCR	マスタコントロール終了	–		なし

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In（MC 命令だけ）	マスタコントロール入力	入力	FALSE: マスタコントロールによるリセット実行	データ型に従う	–	–
MCNo	マスタコントロール No.		マスタコントロール No.	0～14 *1		0

*1.　Sysmac Studio によって自動登録されます。ユーザは設定不要です。

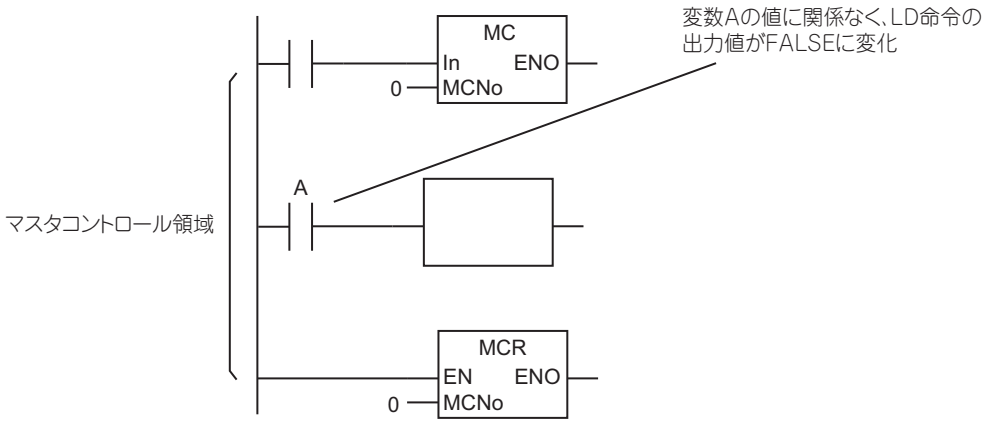
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In（MC 命令だけ）	○																			
MCNo							○													

機能

マスタコントロールとは、ユーザプログラム内の指定した領域に存在するすべての POU を、処理停止またはそれに準ずる状態にする制御のことです。
マスタコントロール機能を使うと、比較的長い一連の処理の実行条件を、容易に制御することが可能です。

マスタコントロールで制御するユーザプログラム内の領域のことを、マスタコントロール領域と呼びます。マスタコントロール領域の開始位置に MC 命令を、終了位置に MCR 命令を配置します。
MC 命令のマスタコントロール入力「In」の値が FALSE になると、下図のように、マスタコントロール領域内の左母線に接続している LD 命令の出力が、強制的に FALSE になります。この状態を、マスタコントロールによるリセットと呼びます。

マスタコントロールによるリセットが実行されると、LD 命令の後段の POU は、原則として、実行条件が FALSE になった場合の動作をします。ただし、POU によって、原則どおりの動作をしないものがあります。それについては、後述します。



「In」の値が TRUE の場合は、マスタコントロールによるリセットは起こりません。マスタコントロール領域にある POU は、通常の動作をします。

マスタコントロールによるリセット実行中の、各 POU の動作

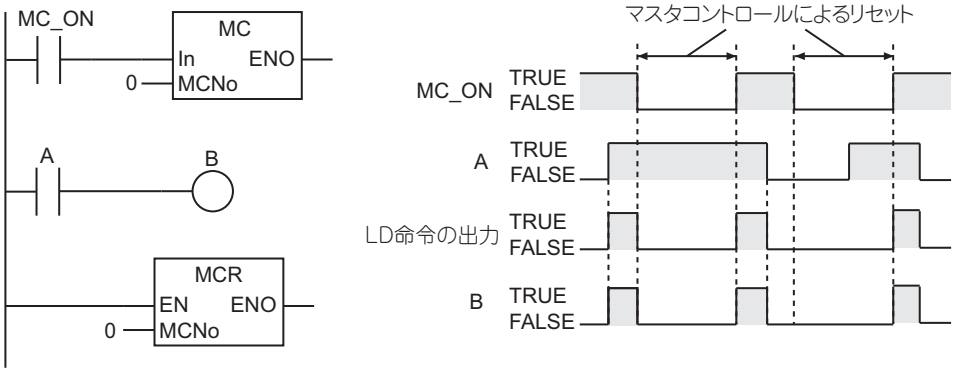
マスタコントロールによるリセットが実行中の動作は、POU によって、以下のように異なります。

POU	動作
Out/OutABit 命令	指定された変数に、FALSE を出力します。
OutNot 命令	指定された変数に、FALSE を出力します。
Set/Reset 命令	マスタコントロールによるリセット実行前の出力を保持します。
TON 命令	タイマ入力「In」の値が FALSE の場合の動作をします。つまり、タイマがリセットされます。 経過時間「ET」の値が 0、タイマ出力「Q」の値が FALSE になります。
TOF 命令	タイマ入力「In」の値が TRUE の場合の動作をします。つまり、タイマがリセットされます。 経過時間「ET」の値が 0、タイマ出力「Q」の値が TRUE になります。 ただし、「Q」を後段の Out 命令に接続している場合、Out 命令には FALSE が入力されます。
TP 命令	タイマ入力「In」の値が FALSE の場合の動作をします。つまりタイマがリセットされます。 計時中　：経過時間「ET」の値は計時終了まで増加し、終了後に 0 になります。タイマ出力「Q」の値は計時終了まで TRUE、終了後に FALSE になります。 計時中以外　：「ET」の値は 0、「Q」の値は FALSE になります。 ただし、「Q」を後段の Out 命令に接続している場合、計時中であっても Out 命令には FALSE が入力されます。
AccumulationTimer 命令	タイマ入力「In」の値が FALSE の場合の動作をします。つまり、タイマが停止します。 積算時間「ET」もタイマ出力「Q」も、値を保持します。 ただし、「Q」を後段の Out 命令に接続している場合、「Q」の値が TRUE であっても、Out 命令には FALSE が入力されます。 また、リセット「Reset」は、有効です。
Timer 命令	タイマ入力「In」の値が FALSE の場合の動作をします。つまり、タイマがリセットされます。 残り時間「ET」に設定時間「PT」の値がセットされ、タイマ出力「Q」の値が FALSE になります。
CTU/CTD/CTUD 命令	実行しません。マスタコントロールによるリセット実行前に、これらの命令が実行中だった場合、カウント値はリセット実行前の値を保持します。 カウントアップフラグ「Q」を、後段の Out 命令に接続している場合、その Out 命令には FALSE が入力されます。
JMP 命令	実行しません。

POU	動作
FOR/NEXT 命令	実行しません。
BREAK 命令	実行しません。
複数のタスク周期にまたがって実行されるファンクションブロック（出力変数に「Done」、「Busy」、「Error」を持つ命令）	左母線に接続された実行条件の値が、FALSE になります。 マスタコントロールによるリセット実行前にこの命令が実行中だった場合、処理が完了するまでは継続して実行します。その場合、「Done」、「Busy」、「Error」の出力は行いますが、その次段にコイルを接続している場合には必ず FALSE を出力します。「Done」、「Busy」、「Error」に変数を直接接続している場合は、その変数には命令の仕様どおりの値が代入されます。 「Done」、「Busy」、「Error」の値を、「インスタンス名.出力変数」の形式で取得することも可能です。
上記以外のファンクション	実行しません。
上記以外のファンクションブロック	左母線に接続された実行条件の値が、FALSE になります。

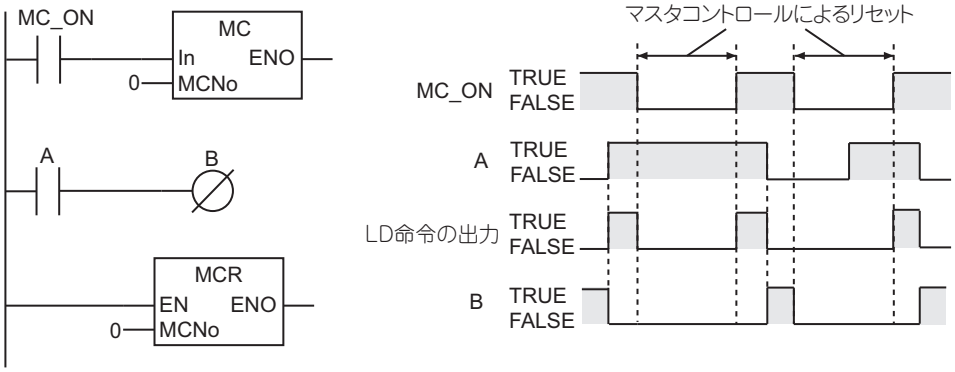
● Out

マスタコントロールによるリセット実行中は、FALSE を出力します。



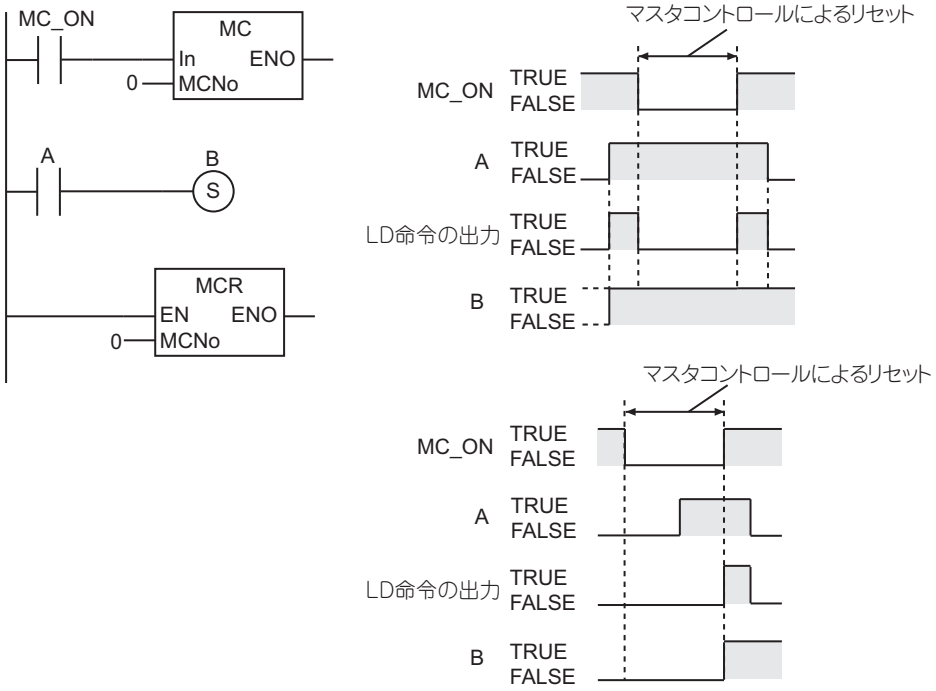
● OutNot

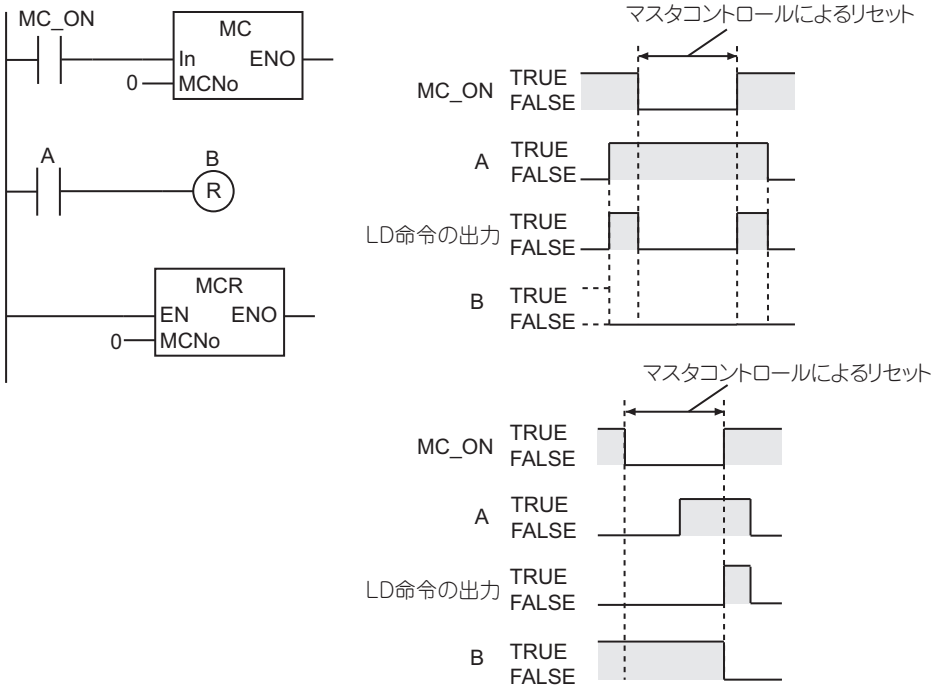
マスタコントロールによるリセット実行中は、FALSE を出力します。
前段の LD 命令の出力が FALSE になった場合の、通常の OutNot 命令の動作とは異なるので、注意してください。



● Set/Reset

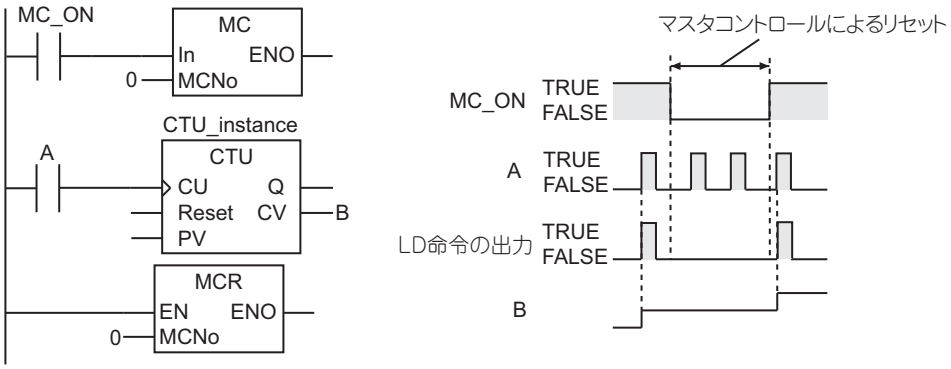
マスタコントロールによるリセット実行中は、それ以前の出力値を保持します。





● CTU/CTD/CTUD

マスタコントロールによるリセット実行中は、それ以前のカウンタ値を保持します。マスタコントロールによるリセット解除後は、保持していたカウンタ値からカウントを続行します。



入力立ち上がり微分、入力立ち下がり微分指定ありの POU の動作

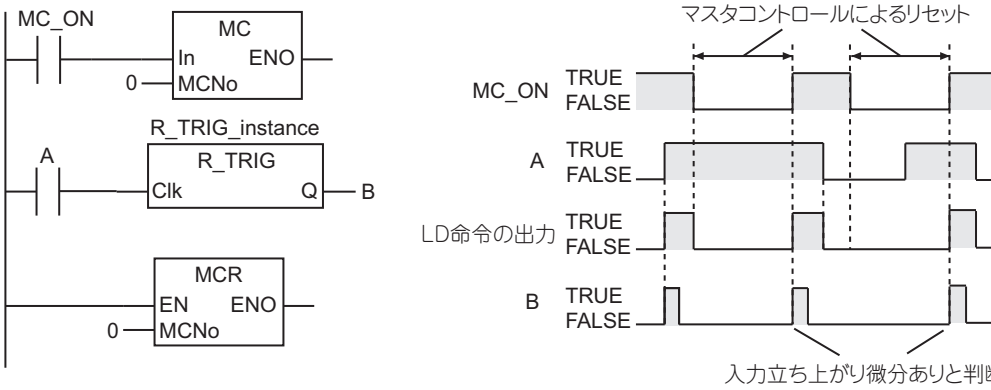
入力立ち上がり微分指定、入力立ち下がり微分指定ありの POU には、以下のものがあります。

微分型	命令
入力立ち上がり微分型	• 立ち上がり微分指定付きの LD, LDN, AND, ANDN, OR, ORN, Out • R_TRIG, Up • 入力立ち上がり微分型の動作オプション(@)付きのファンクション • 入力立ち上がり微分型指定付きのファンクションブロック（カウンタ命令など）
入力立ち下がり微分型	• 立ち下がり微分指定付きの LD, LDN, AND, ANDN, OR, ORN, Out • F_TRIG, Down

マスタコントロールによるリセットが実行または解除されると、これらの POU の実行条件の値が変化します。それによって、これらの POU の実行条件の、立ち上がり微分や立ち下がり微分条件が成立する場合があります。このようにして発生する実行条件の立ち上がり微分や立ち下がり微分も、有効に機能します。

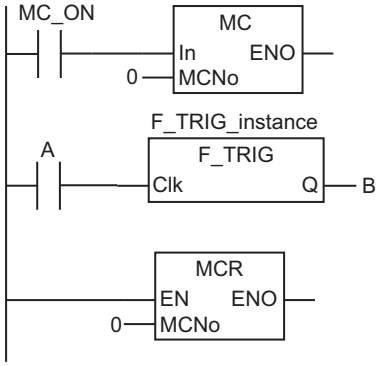
● R_TRIG, Up

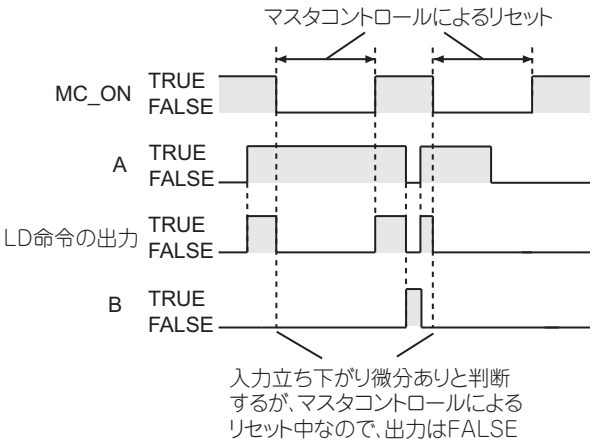
マスタコントロールによるリセットが実行されると、実行条件の値が FALSE になります。その後、マスタコントロールによるリセット解除時に、実行条件の値が TRUE ならば、入力立ち上がり微分が発生したと判断して動作します。



● F_TRIG, Down

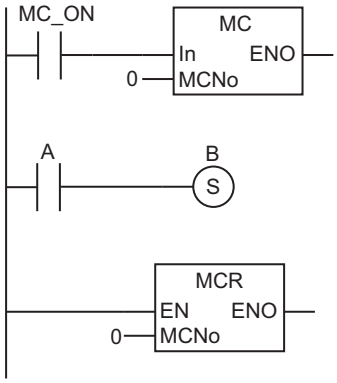
マスタコントロールによるリセットが実行されると、実行条件の値が FALSE になります。その時点までの実行条件の値が TRUE であれば、それにより、入力立ち下がり微分が発生したと判断します。しかし、マスタコントロールによるリセット実行中の F_TRIG、Down の出力値は、強制的に FALSE となるので、結果として出力値は FALSE となります。

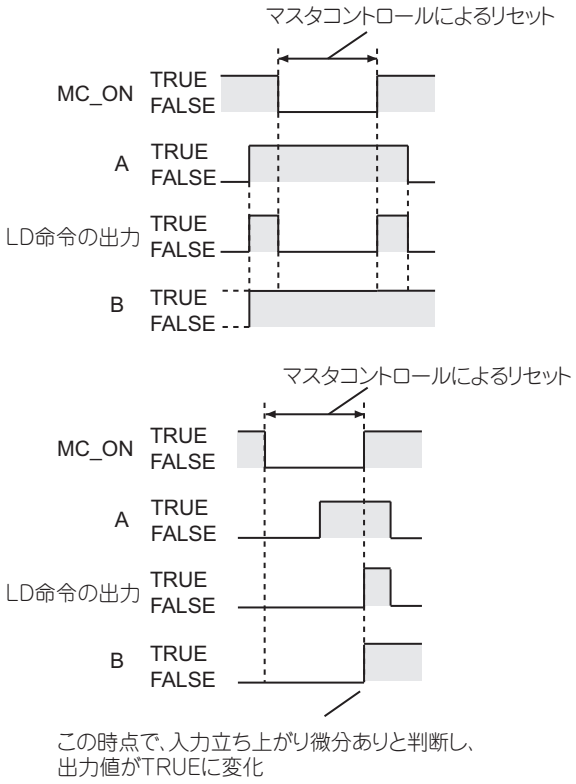


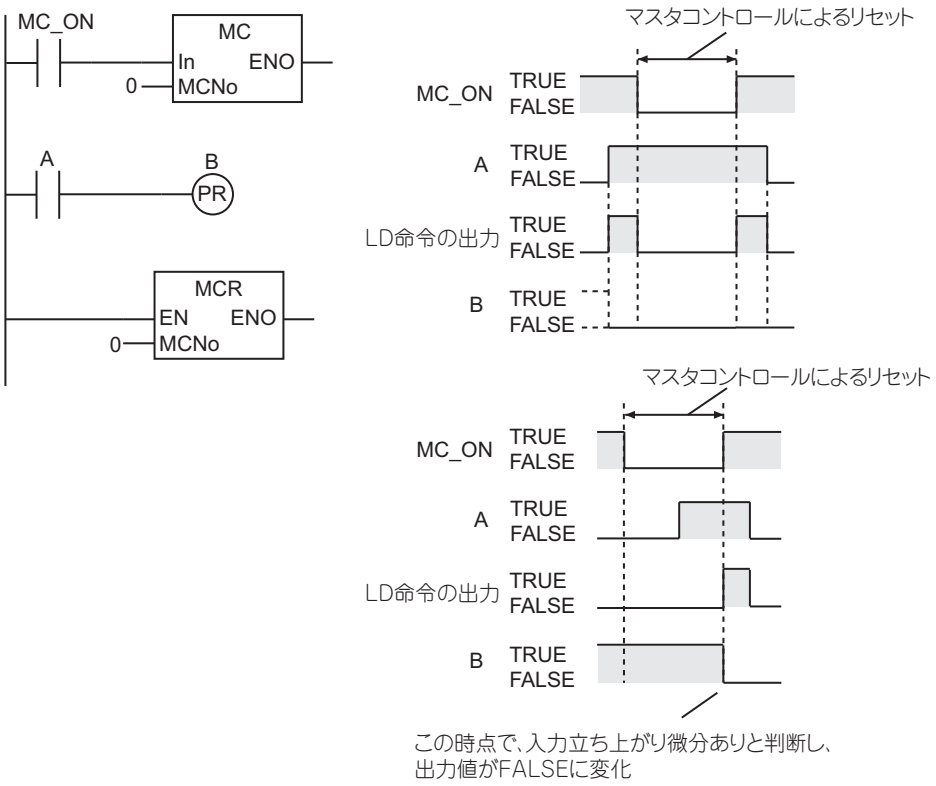


● 入力立ち上がり微分指定ありの Set/Reset

マスタコントロールによるリセット実行中は、それ以前の出力値を保持します。
マスタコントロールによるリセット解除時に、実行条件が FALSE→TRUE になったら、その時点で動作します。

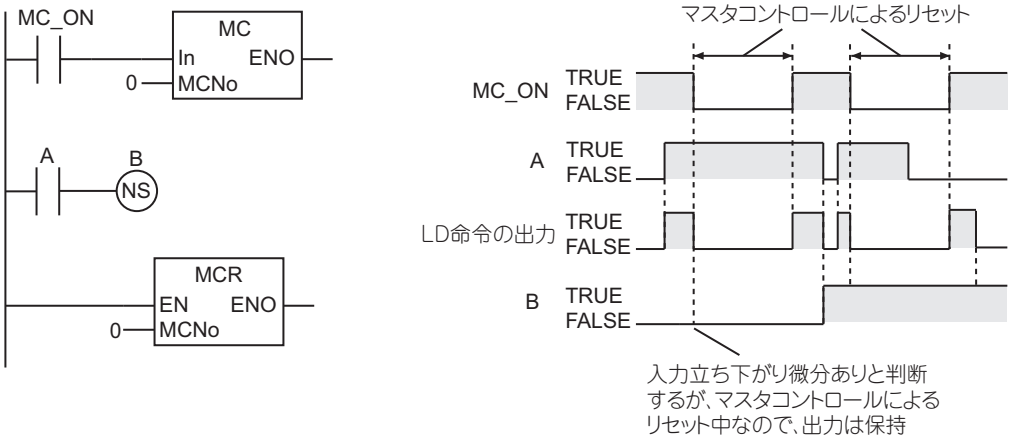


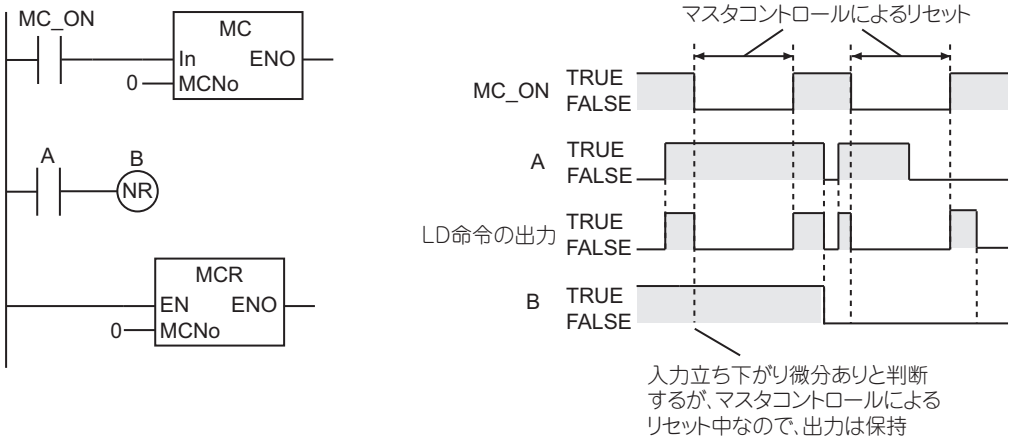




● 入力立ち下がり微分指定ありの Set/Reset

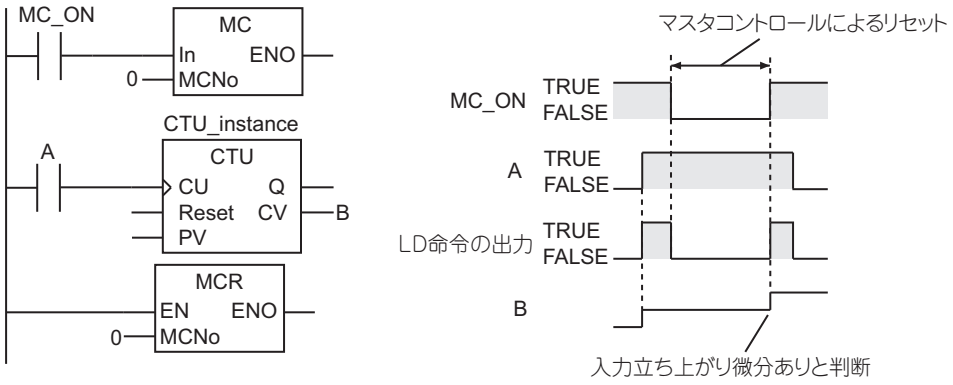
マスタコントロールによるリセットが実行されると、実行条件の値が FALSE になります。その時点までの実行条件の値が TRUE であれば、それにより、入力立ち下がり微分が発生したと判断します。しかし、マスタコントロールによるリセット実行中の出力値は、それ以前の値を保持するので、結果的に出力値は保持となります。





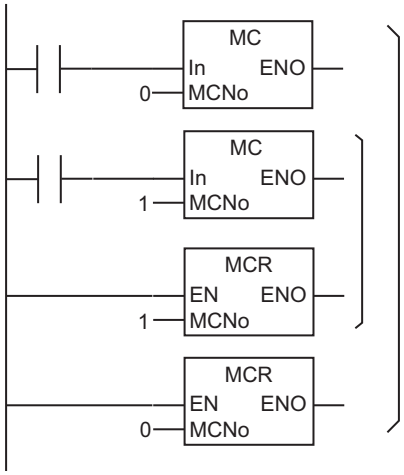
● CTU/CTD/CTUD

マスタコントロールによるリセットが実行されると、カウンタ入力の値が FALSE になります。その後、マスタコントロールによるリセットが解除されたときに、カウンタ入力の値が TRUE ならば、入力立ち上がり微分が発生したと判断してカウントします。



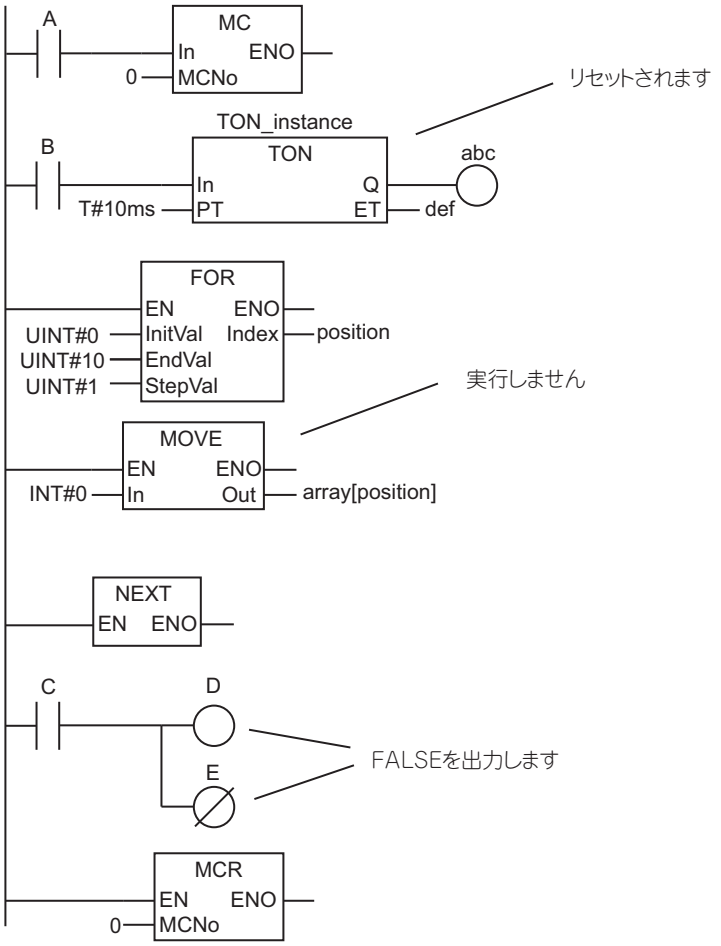
MC 命令と MCR 命令は、必ず同一 POU 内で対にして使用します。相対する MC 命令と MCR 命令には、マスタコントロール No.「MCNo」に、同じ値が割り当てられます。「MCNo」の値は、ユーザが任意に決めるものではなく、Sysmac Studio によって自動登録されます。

MC 命令と MCR 命令は、15 段までネスティングすることが可能です。



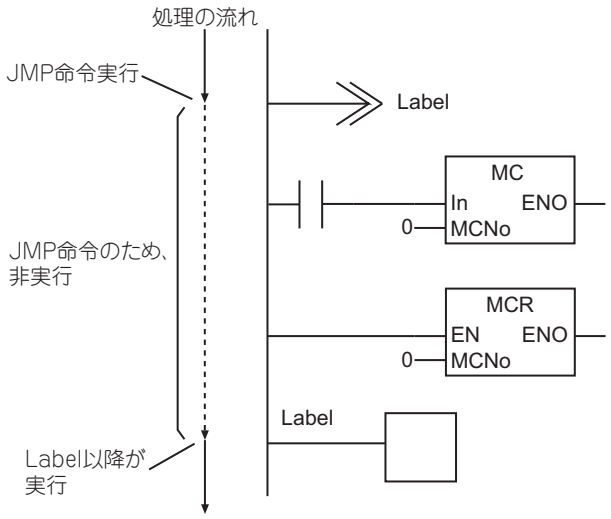
記述例を、以下に示します。

変数 A の値が FALSE になると、マスタコントロールによるリセットを実行します。その間、TON 命令はリセットされます。MOVE 命令は実行されません。また、Out 命令も OutNot 命令も、変数 D と E に、FALSE を出力します。



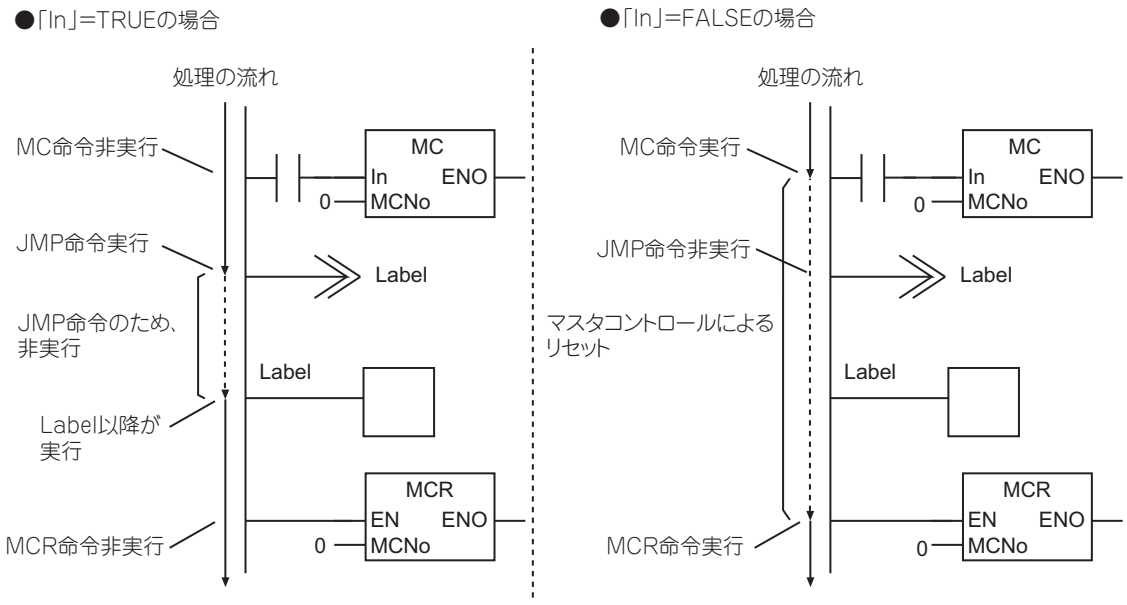
使用上の注意

- 本命令は、ラダー図プログラムだけで使用可能です。ST プログラムでは、使用できません。ラダー図プログラム中のインライン ST 内でも、使用できません。
- 「In」は、必ず左母線に接続してください。「In」に、変数や定数を渡すことはできません。
- MC 命令と MCR 命令は、必ず同一 POU 内で、対にして使用してください。
- MCR 命令は、必ず MC 命令の後ろに配置してください。
- MC 命令と MCR 命令は、16 段以上ネスティングすることはできません。
- マスタコントロール領域にインライン ST が含まれる場合、マスタコントロールによるリセットが実行されると、そのインライン ST の記述は実行されません。
- MC/MCR 命令と JMP 命令を組み合わせて使用する場合、動作は以下のようになります。
  a）下図は、JMP-Label の内側に MC-MCR があります。したがって、「In」の値に関係なくジャンプが実行されます。

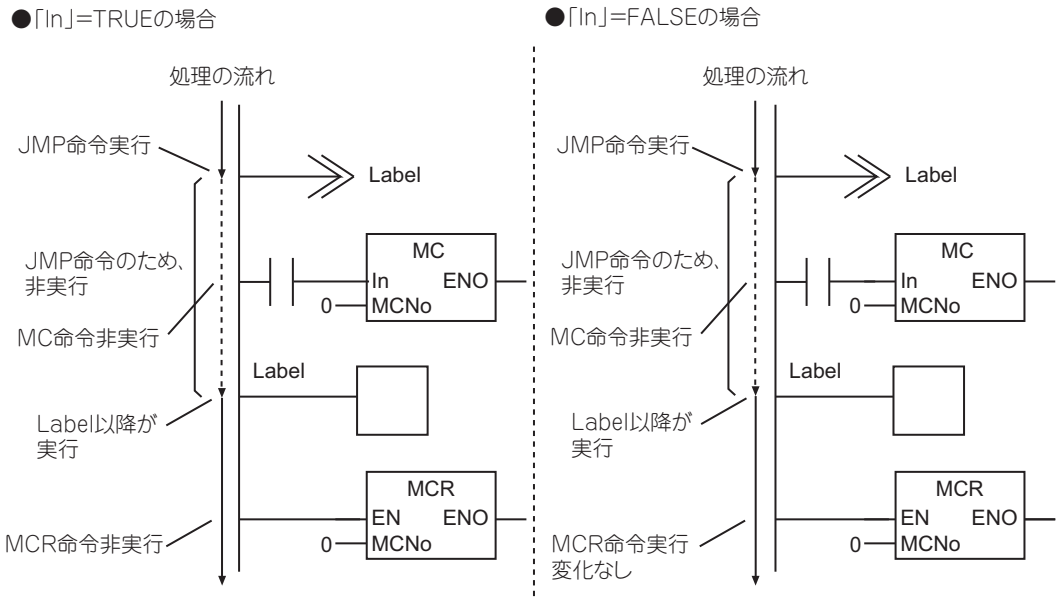


b）下図は、MC-MCR の内側に JMP-Label があります。したがって、動作は下表のようになります。

「In」の値	動作
TRUE	マスタコントロールによるリセットは、実行されません。 ジャンプは実行されます。
FALSE	マスタコントロールによるリセットが、実行されます。 ジャンプは、実行されません。

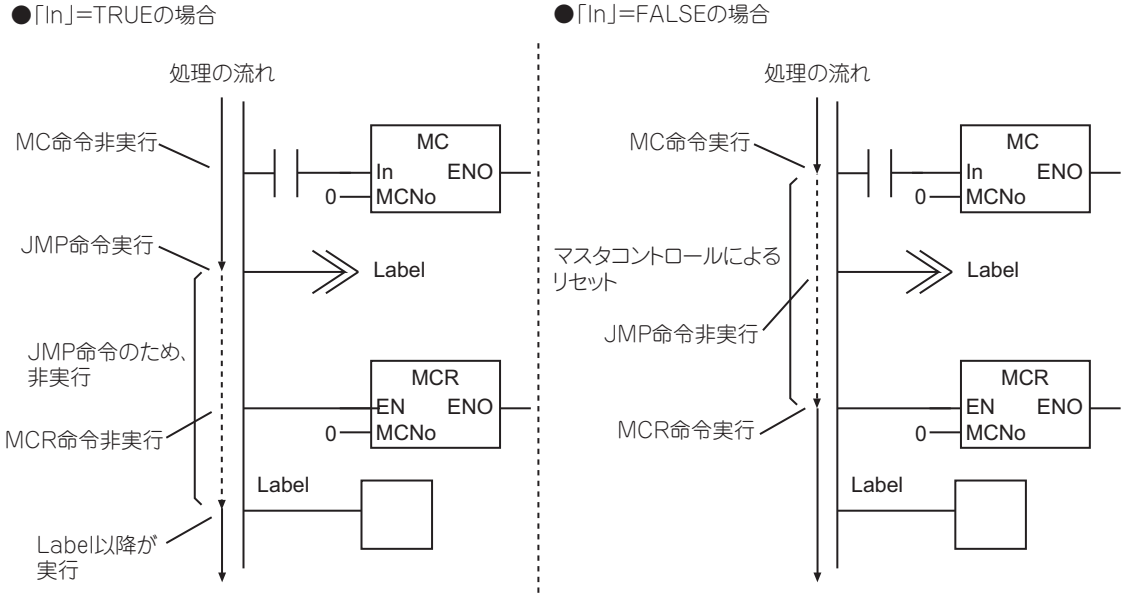


c）下図は、JMP 命令、MC 命令、Label、MCR 命令の順に並んでいます。したがって、まずジャンプが実行されます。その結果、MC 命令は実行されません。よって、Label 以降は実行されます。「In」の値が FALSE ならば MCR 命令は実行されますが、変化はありません。



d）下図は、MC 命令、JMP 命令、MCR 命令、Label の順に並んでいます。したがって、動作は下表のようになります。

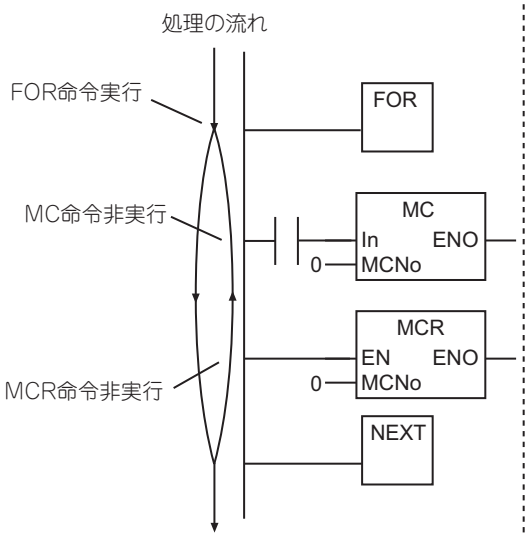
「In」の値	動作
TRUE	マスタコントロールによるリセットは、実行されません。 ジャンプは実行されます。
FALSE	マスタコントロールによるリセットが、実行されます。 ジャンプは、実行されません。



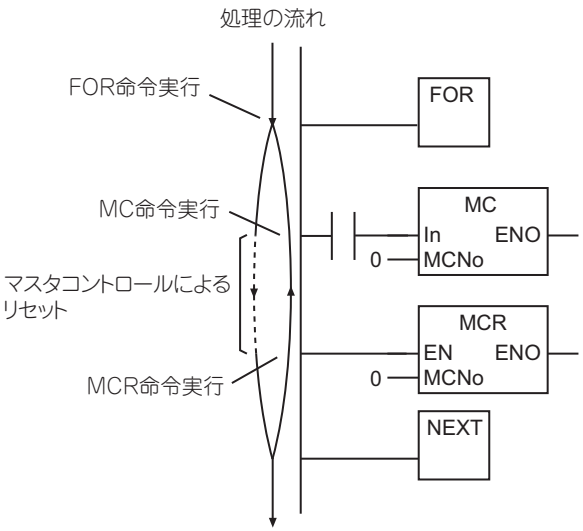
・ MC/MCR 命令と FOR/NEXT 命令を組み合わせて使用する場合、動作は以下のようになります。
a）下図は、FOR-NEXT の内側に MC-MCR があります。したがって、動作は下表のようになります。

「In」の値	動作
TRUE	マスタコントロールによるリセットは、実行されません。 FOR ループは実行されます。
FALSE	マスタコントロールによるリセットが、実行されます。 FOR ループは実行されますが、MC-MCR の内側は実行されません。

●「In」=TRUEの場合



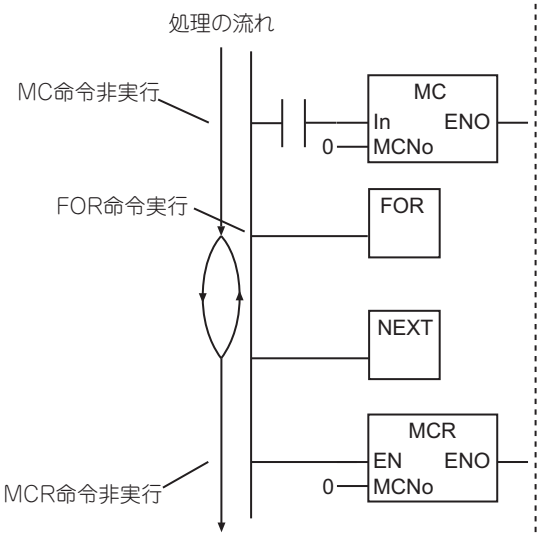
●「In」=FALSEの場合



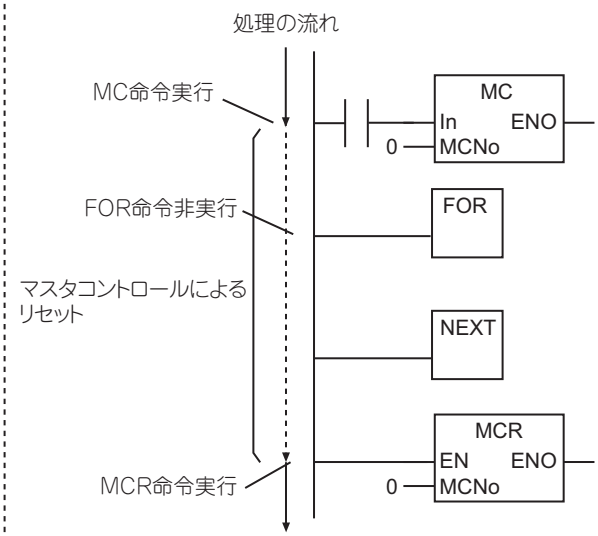
b）下図は、MC-MCR の内側に FOR-NEXT があります。したがって、動作は下表のようになります。

「In」の値	動作
TRUE	マスタコントロールによるリセットは、実行されません。 FOR ループは実行されます。
FALSE	マスタコントロールによるリセットが、実行されます。 FOR ループは実行されません。

●「In」=TRUEの場合



●「In」=FALSEの場合



c）FOR 命令、NEXT 命令、MC 命令、MCR 命令が、つぎのように互い違いに配置されていると、ビルド時に異常になります。
FOR、MC、NEXT、MCR の順、または MC、FOR、MCR、NEXT の順

JMP

指定したジャンプ先に、処理が移ります。

命令	名称	FB/FUN	グラフィック表現	ST 表現
JMP	ジャンプ	FUN	Label	なし

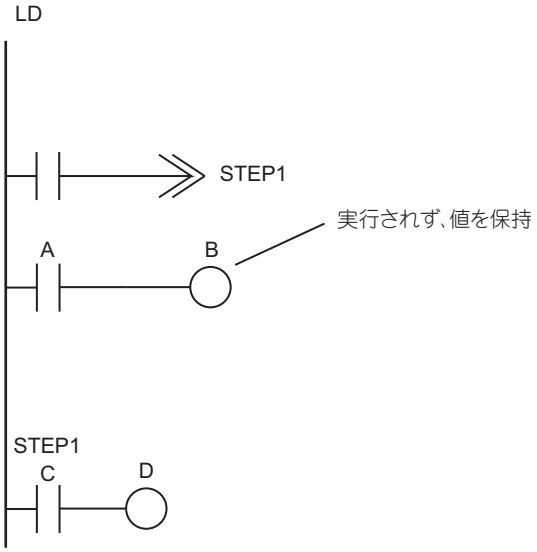
変数

なし

機能

入力条件が TRUE の場合、ラダー図プログラム上に、Label で指定したジャンプ先に、処理を移します。Label は、任意の文字列です。

記述例を、以下に示します。この例では、Label として「STEP1」という文字列を使用しています。JMP 命令が実行されると、処理が「STEP1」と書かれた位置に移ります。この場合、JMP 命令と Label の間にある Out 命令は実行されず、変数「B」は値を保持します。



参考

- セクションの上方向にジャンプすることも可能です。
- 複数の JMP 命令が、同じ Label をジャンプ先に指定することも可能です。

使用上の注意

- Label を省略することはできません。省略すると、ビルド時に異常になります。
- JMP 命令と Label は、同じセクション内に設定してください。
- 同一セクション内に、同じ Label を複数設定することはできません。

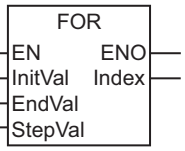
- FOR-NEXT ループの外から中にジャンプすることはできません。
- Label として使用可能な文字の制限は、以下のとおりです。

項目	仕様
最大バイト数	127 バイト ASCII コード換算で 127 文字 日本語文字（半角カナ含む）換算で 31 文字
文字コード	UTF-8
使用可能な文字	CASE インセンシティブ 英数字、各国言語文字 記号: _（アンダーバー）、~（チルダ）
組み合わせで使用できない文字	• ASCII コードの、数字 0～9（文字コード 16#30～16#39）で始まる文字列。 • ASCII コードの、「_」（アンダーバー）が 1 文字だけの文字列。 • 2 個以上連続した、ASCII コードの、「_」（アンダーバー）を含む文字列。 • 最初の 1 文字が、ASCII コードの「_」（アンダーバー）である文字列。 • 最後の 1 文字が、ASCII コードの「_」（アンダーバー）である文字列。 • 最初の 2 文字が、「P_」である文字列。
禁則文字	空白、!（感嘆符）、"（ダブルクォート）、#、$、&、'（シングルクォート）、(、)、*、+、,（カンマ）、-、.（ピリオド）、/、:（コロン）、;（セミコロン）、<、=、>、?、@、[、]、^、`（逆クォート）、%

- Label として、変数名を指定することはできません。

FOR / NEXT

FOR　：繰り返し処理の開始位置を示すとともに、繰り返しの条件を指定します。
NEXT　：繰り返し処理の終了位置を示します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FOR	繰り返し開始	FUN		FOR Index:=InitVal TO EndVal BY StepVal DO 　　式; END_FOR*; *ST 言語では、繰り返し処理の終了位置を示すのに「NEXT」は使わず、「END_FOR」を使います。
NEXT	繰り返し終了	FUN		

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InitVal	初期値	入力	繰り返し開始時に、「Index」にセットされる値	データ型に従う *1	−	*2
EndVal	最終値		繰り返し終了時の「Index」の値			
StepVal	増減値		処理が繰り返されるごとに、「Index」に加算される値	データ型に従う *3		*4
Index	制御変数	出力	ループインデックス	データ型に従う	−	−

*1.　ラダー図プログラムの場合、「InitVal」＜「EndVal」となるように、設定してください。
*2.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*3.　ラダー図プログラムの場合、「0」および負数は含みません。ST プログラムの場合、「0」は含みません。
*4.　ラダー図プログラムの場合、入力パラメータが省略されても、初期値は適用されません。ビルド時に異常になります。
　　ST プログラムの場合、入力パラメータが省略されると、初期値として「1」が適用されます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InitVal						○	○	○	○	○	○	○	○							
	列挙型、配列の 1 要素、構造体の 1 メンバも指定可能 *1																			
EndVal						○	○	○	○	○	○	○	○							
	配列の 1 要素、構造体の 1 メンバも指定可能																			
StepVal						○	○	○	○	○	○	○	○							
	配列の 1 要素、構造体の 1 メンバも指定可能																			
Index						○	○	○	○	○	○	○	○							
	配列の 1 要素、構造体の 1 メンバも指定可能																			

*1.　ラダー図言語の場合、列挙型は指定できません。

機能

FOR から NEXT まで（ST 言語の場合は FOR から END_FOR まで）の処理を、繰り返し行います。FOR/NEXT の処理の手順は、つぎのとおりです。

1 **制御変数「Index」に、「InitVal」の値をセットします。**

2 **「StepVal」、「Index」、「EndVal」の値が、つぎの表の条件を満たすかどうかを判断します。条件を満たしていれば、3 に進みます。満たしていなければ繰り返し処理を行わず、NEXT 命令（ST 言語の場合は END_FOR）のつぎの処理に移ります。**

プログラム言語	繰り返し処理を続ける条件
ラダー図	「StepVal」≧0　かつ　「Index」≦「EndVal」
ST	「StepVal」≧0　かつ　「Index」≦「EndVal」
	「StepVal」＜0　かつ　「Index」≧「EndVal」

3 **「Index」、「EndVal」の値が、つぎの表の条件を満たすかどうかを判断します。条件を満たしていれば 4 に進みます。満たしていなければ、繰り返し処理を終了し、NEXT 命令（ST 言語の場合は END_FOR）のつぎの処理に移ります。**

プログラム言語	繰り返し処理を続ける条件
ラダー図	「Index」≦「EndVal」
ST	「StepVal」≧0 の場合、「Index」≦「EndVal」
	「StepVal」＜0 の場合、「Index」≧「EndVal」

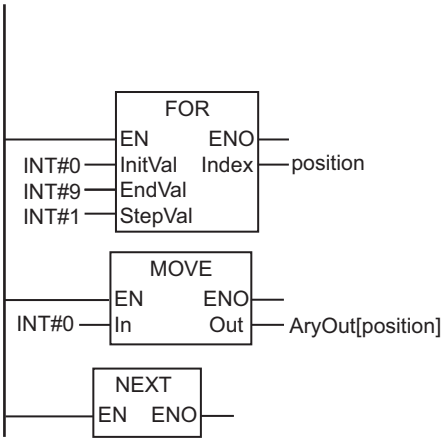
4 **FOR 命令と NEXT 命令（ST 言語の場合は END_FOR）の間にある処理を 1 回実行します。**

5 **「Index」に、「StepVal」の値を加算します。**

6 **3 に戻ります。**

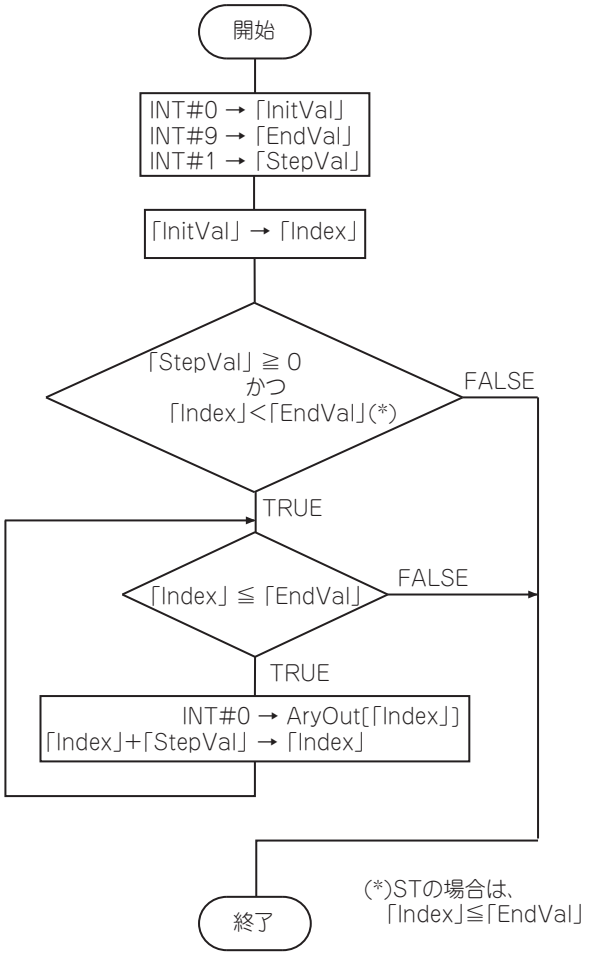
「InitVal」＝「INT#0」、「EndVal」＝「INT#9」、「StepVal」＝「INT#1」の場合の例を、以下に示します。MOVE 命令が 10 回実行され、配列変数 AryOut[0]～[9]に値 INT#0 が代入されます。

LD



ST

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



AryOut[0]からAryOut[9]に、
順番にINT#0が代入されます。

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

入力変数に演算式やファンクションを用いたST プログラムの記述例

本命令を ST プログラムで使用する場合、「InitVal」、「EndVal」、「StepVal」の入力変数には、下記の記述ができます。

- 演算結果が整数値になる演算式
- 整数値を戻すファンクション
- 列挙子を戻すファンクション

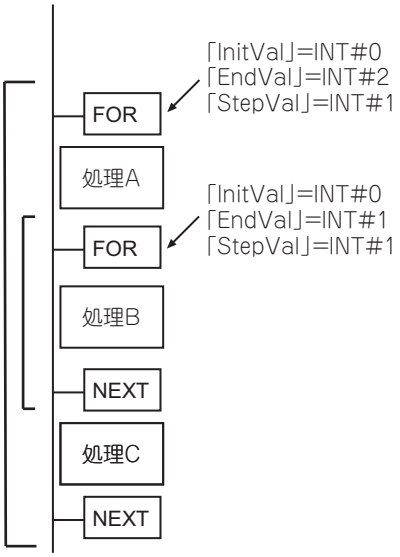
以下は、「EndVal」をファンクション、「StepVal」を演算式で記述した例です。

```
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;
```

参考

- 繰り返し処理を中断するには、BREAK 命令（ST 言語の場合は EXIT）を実行してください。その場合、BREAK 命令から NEXT 命令までの処理は実行されません。
- FOR から NEXT（ST 言語の場合は FOR から END_FOR）の間に、さらに FOR/NEXT 命令を配置する（ネスティング）ことも可能です。下図の場合、処理の順序は、つぎのようになります。

処理 A→処理 B→処理 B→処理 C→処理 A→処理 B→処理 B→処理 C→処理 A→処理 B→処理 B→処理 C



使用上の注意

- 本命令をラダー図プログラムで使用する場合、FOR 命令も NEXT 命令も、必ず左母線に直接接続してください。
- FOR 命令と NEXT 命令（ST 言語の場合は FOR と END_FOR）は、必ず対にして使用してください。両者の数が同じでないと、プログラム異常になります。
- 対になる FOR 命令と NEXT 命令は、同一セクション内に設定してください。
- 処理の繰り返しが無限に行われる（無限ループ）ことのないよう、繰り返し終了条件は十分注意して設定してください。無限ループになると、タスク実行タイムアウトが発生します。

  各変数の入力パラメータ値が以下の表の場合、SINT 型の最大値は 127 ですから、「Index」の値が「EndVal」の値より大きくなることはありません。したがって、無限ループになります。
  「EndVal」には、データ型の最大値を設定しないようにしてください。

変数	入力パラメータ値
InitVal	SINT#0
EndVal	SINT#127
StepVal	SINT#1
Index	−

- 「StepVal」、「InitVal」、「EndVal」の値によって、動作は以下のようになります。

プログラム言語	「StepVal」の値	「InitVal」、「EndVal」の値	動作
ラダー図	StepVal ＞ 0	InitVal ≦ EndVal	正常に動作します。
		InitVal ＞ EndVal	FOR 命令と NEXT 命令の間の処理が、一度も実行されません。 異常は発生しません。
	StepVal ＜ 0	InitVal ＜ EndVal	FOR 命令と NEXT 命令の間の処理が、不定回実行されます。 このような設定は、しないでください。異常は発生しません。
		InitVal ≧ EndVal	FOR 命令と NEXT 命令の間の処理が、一度も実行されません。 異常は発生しません。
	StepVal ＝ 0	InitVal ＜ EndVal	無限ループになり、タスク実行タイムアウトが発生します。
		InitVal ≧ EndVal	FOR 命令と NEXT 命令の間の処理が、一度も実行されません。 異常は発生しません。
ST	StepVal ＞ 0	InitVal ≦ EndVal	正常に動作します。
		InitVal ＞ EndVal	FOR 命令と END_FOR 命令の間の処理が、一度も実行されません。 異常は発生しません。
	StepVal ＜ 0	InitVal ＜ EndVal	FOR 命令と END_FOR 命令の間の処理が、一度も実行されません。 異常は発生しません。
		InitVal ≧ EndVal	正常に動作します。
	StepVal ＝ 0	InitVal ≦ EndVal	無限ループになり、タスク実行タイムアウトが発生します。
		InitVal ＞ EndVal	FOR 命令と END_FOR 命令の間の処理が、一度も実行されません。 異常は発生しません。

- ネスティング（FOR/NEXT 命令の階層構造）の深さは、IF 命令、CASE 命令、FOR 命令、WHILE 命令、REPEAT 命令を含めて、15 階層までです。
- ネスティングしている場合、繰り返し処理を中断するには、階層の数だけ BREAK 命令（ST 言語の場合は EXIT）が必要です。
- 繰り返し処理を中断するために、ジャンプ系命令（JMP 命令など）を使用しないでください。繰り返し処理の中断には、必ず BREAK 命令（ST 言語の場合は EXIT）を使用してください。
- 繰り返し処理中に、「InitVal」、「EndVal」、「StepVal」の値を変更した場合の動作が、ラダー図言語と ST 言語では異なります。

変数	動作	
	ラダー図言語	ST 言語
InitVal	繰り返し処理が終了するまで、変更後の値が反映されません。	繰り返し処理が終了するまで、変更後の値が反映されません。
EndVal	繰り返し処理の途中でも、変更後の値が反映されます。	
StepVal	意図どおりの動作をしない場合があります。繰り返し処理中の値変更はしないでください。	

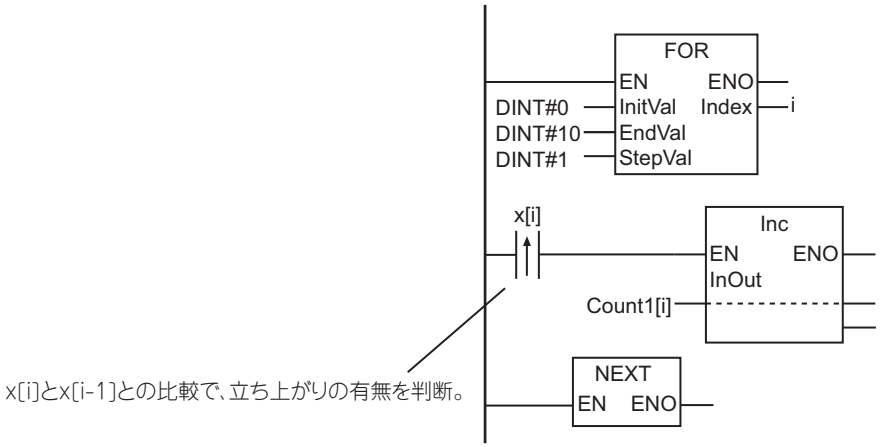
- ラダー図言語の場合、「InitVal」、「EndVal」、「StepVal」、「Index」は、すべて同じデータ型にしてください。すべて同じにしないと、ビルド時に異常となります。
- 「Index」のデータ型は、「InitVal」、「EndVal」、「StepVal」の有効範囲を包含するようにしてください。包含していないと、ビルド時に異常となります。
- 繰り返し処理終了後の「Index」の値は、ラダー図言語と ST 言語とで異なります。ラダー図言語は、最後に「Index」に「StepVal」を加えずに終了します。ST 言語は、最後に「Index」に「StepVal」を加えて終了します。繰り返し回数に、違いはありません。

（例）「InitVal」=1、「EndVal」=100、「StepVal」=1 の場合

ラダー図言語　：　繰り返し回数は 100 回、繰り返し終了後の「Index」の値は 100
ST 言語　　：　繰り返し回数は 100 回、繰り返し終了後の「Index」の値は 101

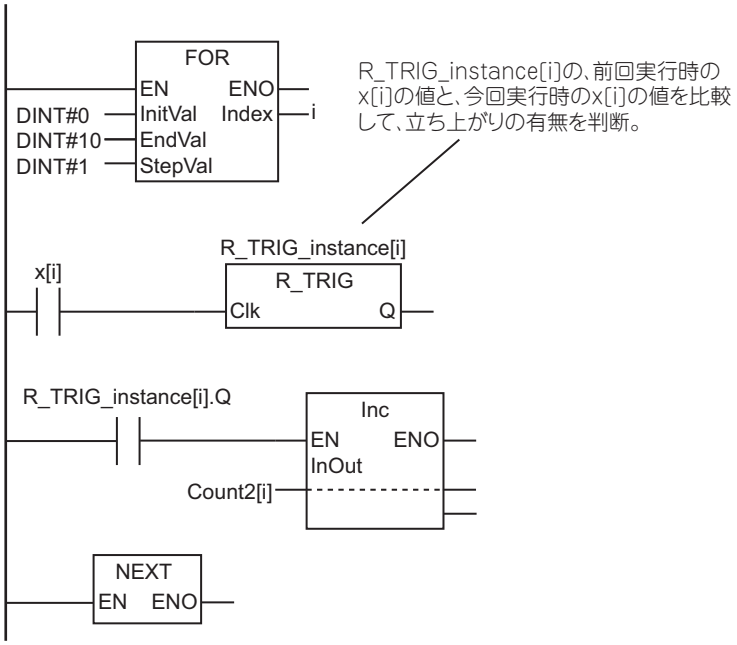
- LD プログラムの FOR ループ中で、LD 命令（または AND 命令、OR 命令）を立ち上がり微分指定（または立ち下がり微分指定）ありで、かつ配列とともに使用する場合は、注意が必要です。
  立ち上がり微分指定は、その命令の前回実行時の指定変数の値と今回実行時の指定変数の値を比較して、立ち上がりの有無を判断します。通常、命令実行のたびに指定変数が変わることはありませんが、FOR ループ中で、かつ配列を指定した場合は、命令実行のたびに配列の要素が変わります。このため、配列の要素の値どうしの比較で、立ち上がりの有無が判断されることになります。
  下のユーザプログラムの場合、x[i-1]の値、x[i]の値、および Count1[i]へのインクリメント処理の関係は、下表のようになります。

x[i-1]の値	x[i]の値	Count1[i]へのインクリメント処理
TRUE	TRUE	実行されない。
TRUE	FALSE	実行されない。
FALSE	TRUE	実行される。
FALSE	FALSE	実行されない。



- 一方、下のユーザプログラムの場合は、x[i]の立ち上がりの有無を、R_TRIG 命令で検知します。
  x[i]の各要素に対応した R_TRIG インスタンスが設けられているので、x[i]の要素ごとの値の変化で、立ち上がりの有無が判断されます。したがって、R_TRIG_instance[i]を前回実行した際の x[i]の値、R_TRIG_instance[i]を今回実行した際の x[i]の値、Count2[i]へのインクリメント処理の関係は、下表のようになります。

R_TRIG_instance[i]を前回実行した際の x[i]の値	R_TRIG_instance[i]を今回実行した際の x[i]の値	Count2[i]へのインクリメント処理
TRUE	TRUE	実行されない。
TRUE	FALSE	実行されない。
FALSE	TRUE	実行される。
FALSE	FALSE	実行されない。


FOR
EN
ENO
DINT#0
InitVal
Index
i
DINT#10
EndVal
DINT#1
StepVal
R_TRIG_instance[i]の、前回実行時の
x[i]の値と、今回実行時のx[i]の値を比較
して、立ち上がりの有無を判断。
R_TRIG_instance[i]
x[i]
R_TRIG
Clk
Q
R_TRIG_instance[i].Q
Inc
EN
ENO
InOut
Count2[i]
NEXT
EN
ENO

BREAK

最も内側の FOR 命令から NEXT 命令までの繰り返し処理を中断します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BREAK	ループブレイク	FUN	BREAK EN ENO	なし

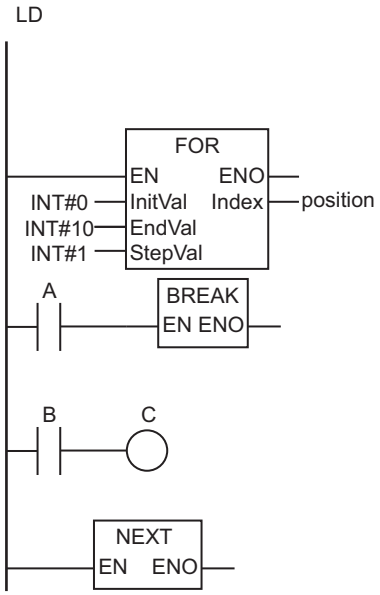
変数

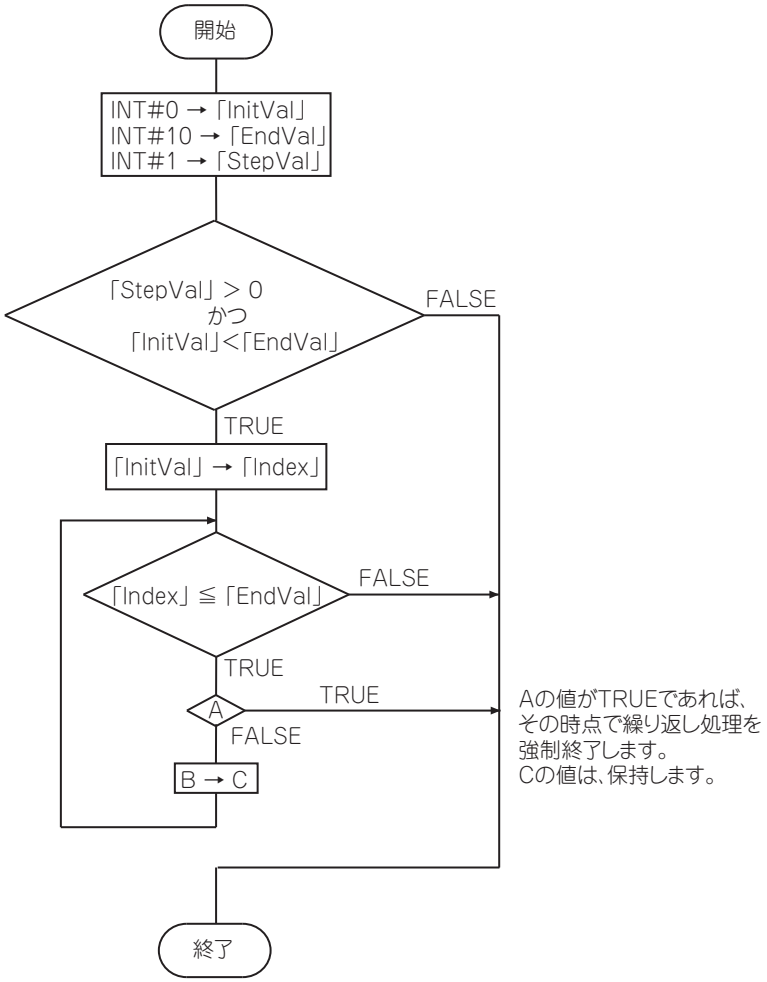
なし

機能

入力条件が TRUE の場合、最も内側の FOR 命令から NEXT 命令までの繰り返し処理を強制終了し、NEXT 命令のつぎの処理に移ります。
BREAK 命令から NEXT 命令までの処理は、実行されません。

以下に記述例を示します。FOR ループを実行中、毎回、変数「A」の値を確認します。変数「A」の値が TRUE であれば、その時点で繰り返し処理を強制終了します。
その場合、BREAK 命令の後ろにある Out 命令は実行されず、変数「C」は値を保持します。





使用上の注意

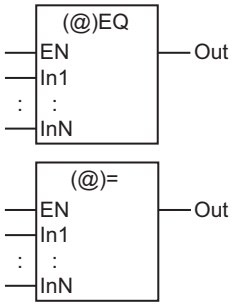
- 本命令は、必ず FOR 命令と NEXT 命令の間で使用してください。
- FOR 命令と NEXT 命令を階層的に使用している（ネスティング）場合、繰り返し処理を完全に中断するには、階層の数だけ BREAK 命令が必要です。
- 繰り返し処理を中断するために、ジャンプ系命令（JMP 命令など）を使用しないでください。繰り返し処理の中断には、必ず BREAK 命令を使用してください。

比較命令

命令	名称	ページ
EQ, =	比較 EQ	P.2-96
NE, <>	比較 NE	P.2-98
LT,< / LE,<= / GT,> / GE,>=	比較 LT／比較 LE／比較 GT／比較 GE	P.2-100
EQascii	文字列比較 EQ	P.2-103
NEascii	文字列比較 NE	P.2-105
LTascii / LEascii / GTascii / GEascii	文字列比較 LT／文字列比較 LE／文字列比較 GT／文字列比較 GE	P.2-107
Cmp	比較	P.2-110
ZoneCmp	領域比較	P.2-112
TableCmp	テーブル比較	P.2-114
AryCmpEQ / AryCmpNE	配列一括比較 EQ／配列一括比較 NE	P.2-117
AryCmpLT / AryCmpLE / AryCmpGT / AryCmpGE	配列一括比較 LT／配列一括比較 LE／配列一括比較 GT／配列一括比較 GE	P.2-119
AryCmpEQV / AryCmpNEV	配列要素比較 EQ／配列要素比較 NE	P.2-121
AryCmpLTV / AryCmpLEV / AryCmpGTV / AryCmpGEV	配列要素比較 LT／配列要素比較 LE／配列要素比較 GT／配列要素比較 GE	P.2-123

EQ, =

複数のデータが、すべて等しいかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EQ,=	比較 EQ	FUN		Out:=(In1=In2) & (In2=In3) & ··· & (InN-1=InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	比較対象	入力	比較する数値 N は 2～5	データ型に従う	−	0 *1
Out	比較結果	出力	比較結果	データ型に従う	−	−

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

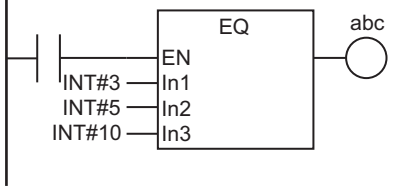
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			
Out	○																			

機能

2 個以上 5 個以下のデータ「In1」～「InN」が、すべて等しいかどうかを判定します。
すべて等しい場合だけ、比較結果「Out」の値が TRUE になります。それ以外は、「Out」の値が FALSE になります。
STRING 型の比較の場合、“等しい”とは、文字列の長さも内容も同じことを指します。

「In1」=INT#3、「In2」=INT#5、「In3」= INT#10 の場合の例を、以下に示します。変数「abc」の値は、FALSE になります。

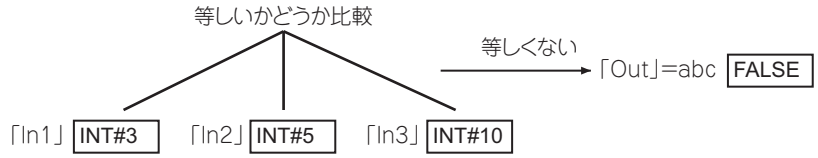
LD



ST

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

「In1」～「In3」がすべて等しいかどうかを判定します。
等しくないので、abcの値はFALSEになります。



参考

- EQ と=は、まったく同じ機能の命令です。使いやすい方を使用してください。
- TIME 型、DT 型、TOD 型の比較を行う場合、比較する値どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- 「In1」～「InN」のデータ型が異なる場合、すべてのデータ型の有効範囲を包含するデータ型に型拡張した後、比較を行います。
- ビット列型(BYTE, WORD, DWORD, LWORD)と整数型(SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT)の比較はできません。また、ビット列型と実数型(REAL, LREAL)の比較もできません。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- TIME 型、DATE 型、TOD 型、DT 型、STRING 型は、同じデータ型どうしの比較しかできません。異なるデータ型を指定すると、ビルド時に異常になります。
- 列挙型と、列挙型以外の型の比較はできません。また、列挙型どうしでも、型が一致していないと比較できません。
- +∞どうし、または-∞どうしは、等しいと判断します。
- 「In1」～「InN」のいずれかの値が非数の場合、「Out」の値は FALSE になります。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- 「In1」～「InN」が実数の場合、丸め誤差のため、処理結果が意図どおりにならない可能性があります。等しいことを検証する値のいずれかが実数の場合は、本命令を使用せず、大小比較の命令を使用して、両者の差の絶対値が許容範囲より小さいかどうかを検証してください。たとえば、REAL 型変数「real_a」と「real_b」の和と 0.1 が等しいことを検証する場合は、次のように記述してください。BOOL 型変数「boolv」の値が TRUE になれば、両者は等しいと判断します。

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

  threshold: 許容範囲の値

NE, <>

2 つのデータが、異なっているかどうかを判断します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NE, <>	比較 NE	FUN	(@)NE EN In1 In2 Out (@)<> EN In1 In2 Out	Out:=(In1 <> In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1, In2	比較対象	入力	比較する数値	データ型に従う	−	*1
Out	比較結果	出力	比較結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

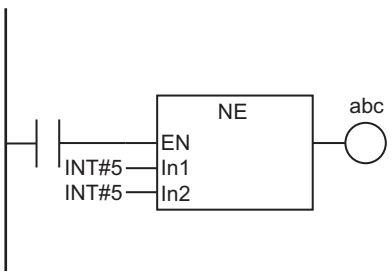
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1, In2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			
Out	○																			

機能

2 つのデータ「In1」と「In2」が異なっているかどうかを判定します。
異なっていれば、比較結果「Out」は TRUE、同一であれば FALSE になります。
STRING 型の比較の場合、“等しい”とは、文字列の長さも内容も同じことを指します。

「In1」=「In2」=INT#5 の場合の例を、以下に示します。変数「abc」の値は、FALSE になります。

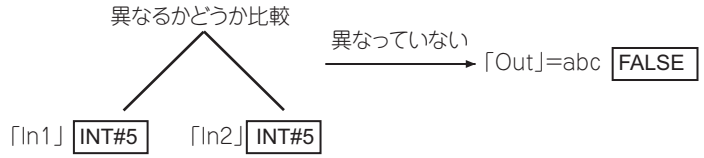
LD



ST

```
abc:=(INT#5<>INT#5);
```

「In1」と「In2」が異なるかどうかを判定します。
異なっていないので、abcの値はFALSEになります。



参考

- NE 命令と<>命令の機能はまったく同じです。使いやすい方を使用してください。
- TIME 型、DT 型、TOD 型の比較を行う場合、比較する値どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

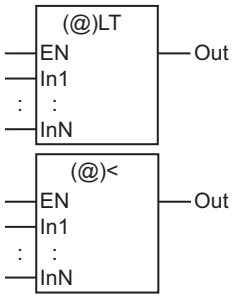
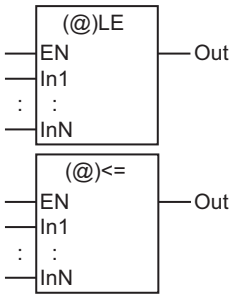
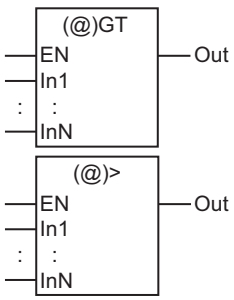
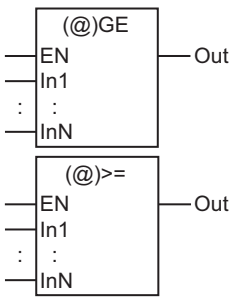
- 「In1」と「In2」のデータ型が異なる場合、両者のデータ型の有効範囲を包含するデータ型に型拡張した後、比較を行います。
- ビット列型(BYTE, WORD, DWORD, LWORD)と整数型(SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT)の比較はできません。また、ビット列型と実数型(REAL, LREAL)の比較もできません。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- TIME 型、DATE 型、TOD 型、DT 型、STRING 型は、同じデータ型どうしの比較しかできません。異なるデータ型を指定すると、ビルド時に異常になります。
- 列挙型と、列挙型以外の型の比較はできません。また、列挙型どうしでも、型が一致していないと比較できません。
- +∞どうし、または-∞どうしは、等しいと判断します。
- 「In1」、「In2」のいずれかの値が非数の場合、「Out」の値は TRUE になります。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- 「In1」、「In2」が実数の場合、丸め誤差のため、処理結果が意図どおりにならない可能性があります。異なっていることを検証する値のいずれかが実数の場合は、本命令を使用せず、大小比較の命令を使用して、両者の差の絶対値が許容範囲より大きいかどうかを検証してください。たとえば、REAL 型変数「real_a」と「real_b」の和と、0.1 が異なっていることを検証する場合は、次のように記述してください。BOOL 型変数「boolv」の値が TRUE になれば、両者は異なっていると判断します。

  boolv:= (ABS((real_a + real_b) - 0.1) > threshold);
  threshold: 許容範囲の値

LT,< / LE,<= / GT,> / GE,>=

複数の数値の、大小比較を行います。

LT, < ： 大小比較(<)を行います。
LE, <= ： 大小比較(<=)を行います。
GT, > ： 大小比較(>)を行います。
GE, >= ： 大小比較(>=)を行います。

命令	名称	FB/ FUN	グラフィック表現	ST 表現
LT, <	比較 LT	FUN		Out:=(In1<In2) & (In2<In3) & ··· & (InN-1<InN);
LE, <=	比較 LE	FUN		Out:=(In1<=In2) & (In2<=In3) & ··· & (InN-1<=InN);
GT, >	比較 GT	FUN		Out:=(In1>In2) & (In2>In3) & ··· & (InN-1>InN);
GE, >=	比較 GE	FUN		Out:=(In1>=In2) & (In2>=In3) & ··· & (InN-1>=InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	比較対象	入力	比較する数値 N は 2～5	データ型に従う	－	*1
Out	比較結果	出力	比較結果	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Out	○																			

機能

2 個以上 5 個以下のデータ「In1」～「InN」の大小を比較します。
各命令の出力「Out」の値は、つぎのようになります。

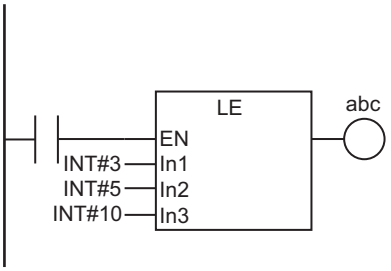
命令	「Out」の値
LT, <	「In1」<「In2」<・・<「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
LE, <=	「In1」<=「In2」<=・・<=「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
GT, >	「In1」>「In2」>・・>「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
GE, >=	「In1」>=「In2」>=・・>=「InN」のとき、「Out」は TRUE、それ以外のとき FALSE

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

データ型	大小関係
BYTE, WORD, DWORD, LWORD	符号なし整数として判断します。
TIME	値が大きい方が、大きいと判断します。
DATE, TOD, DT	日付や時刻が後の方が大きいと判断します。
STRING	「LTascii / LEascii / GTascii / GEascii (P.2-107)」命令と同じ仕様です。そちらを参照してください。

LE 命令で、「In1」=INT#3、「In2」=INT#5、「In3」= INT#10 の場合の例を、以下に示します。変数「abc」の値は、TRUE になります。

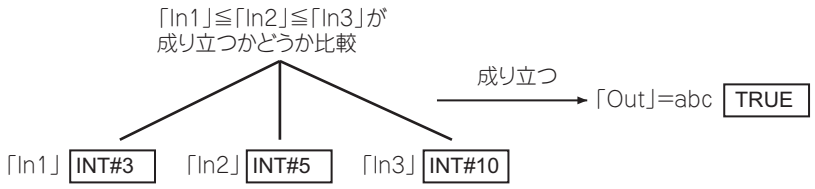
LD



ST

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

「In1」≦「In2」≦「In3」が成り立つかどうか比較します。
成り立つので、abcの値はTRUEになります。



参考

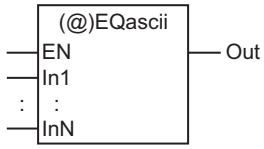
- LT と<、LE と<=、GT と>、GE と>=は、それぞれまったく同じ機能の命令です。使いやすい方を使用してください。
- TIME 型、DT 型、TOD 型の大小比較を行う場合、比較する値どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」 命令があります。

使用上の注意

- 「In1」～「InN」のデータ型が異なる場合、すべてのデータ型の有効範囲を包含するデータ型に型拡張した後、比較を行います。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- ビット列型(BYTE, WORD, DWORD, LWORD)と整数型(SINT, INT, DINT, LINT, USINT, UINT, UDINT,ULINT)の比較はできません。また、ビット列型と実数型(REAL, LREAL)の比較もできません。
- TIME 型、DATE 型、TOD 型、DT 型、STRING 型は、同じデータ型どうしの比較しかできません。異なるデータ型を指定すると、ビルド時に異常になります。
- 「In1」～「InN」が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- +∞どうし、または-∞どうしは、等しいと判断します。
- 「In1」～「InN」のいずれかの値が非数の場合、「Out」の値は FALSE になります。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。

EQascii

複数の文字列が、すべて等しいかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EQascii	文字列比較 EQ	FUN		Out:=EQascii(In1, ‥, InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	比較対象文字列	入力	比較する文字列 N は 2～5	データ型に従う	−	" *1
Out	比較結果	出力	比較結果	データ型に従う	−	−

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

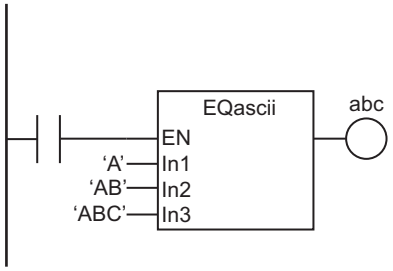
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN																				○
Out	○																			

機能

2 個以上 5 個以下の文字列「In1」～「InN」が、すべて等しいかどうかを判定します。
すべて等しい場合だけ、比較結果「Out」の値が TRUE になります。それ以外は、「Out」の値が FALSE になります。
“等しい”とは、長さも内容も同じことを指します。

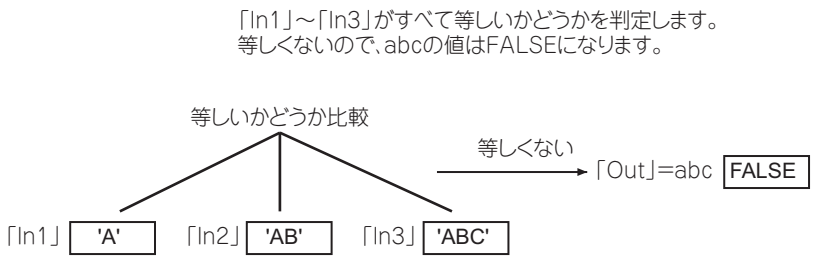
「In1」='A'、「In2」='AB'、「In3」='ABC'の場合の例を、以下に示します。変数「abc」の値は、FALSE になります。

LD



ST

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



参考

文字列比較の命令は、複数の文字列を、文字コード順に並び替える場合などに便利です。たとえば、英文字は、文字コードの順番がアルファベットの順番と同じなので、アルファベット順での並び替えが可能です。

参考

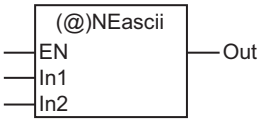
「EQ, = (P.2-96)」命令の文字列比較を使う方法もあります。EQ, =命令の文字列比較と、本命令とは、まったく同じ仕様です。

使用上の注意

- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- 「In1」～「InN」には、ASCII コードだけで構成された文字列を指定してください。

NEascii

2 つの文字列が、異なっているかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NEascii	文字列比較 NE	FUN		Out:=NEascii(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1,In2	比較対象文字列	入力	比較する文字列	データ型に従う	−	*1
Out	比較結果	出力	比較結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

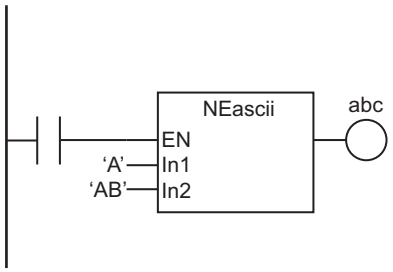
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1, In2																				○
Out	○																			

機能

2 つの文字列「In1」、「In2」が、異なっているかどうかを判定します。
異なっていれば、比較結果「Out」は TRUE、等しければ FALSE になります。
“等しい”とは、長さも内容も同じことを指します。

「In1」='A'、「In2」='AB'の場合の例を、以下に示します。変数「abc」の値は、TRUE になります。

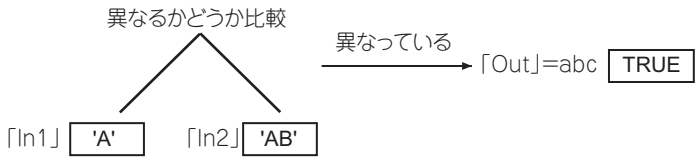
LD



ST

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

「In1」と「In2」が異なるかどうかを判定します。
異なっているので、abcの値はTRUEになります。



参考

文字列比較の命令は、複数の文字列を、文字コード順に並び替える場合などに便利です。たとえば、英文字は、文字コードの順番がアルファベットの順番と同じなので、アルファベット順での並び替えが可能です。

参考

「NE, <> (P.2-98)」命令の文字列比較を使う方法もあります。NE,<>命令の文字列比較と、本命令とは、まったく同じ仕様です。

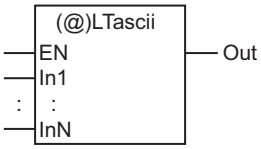
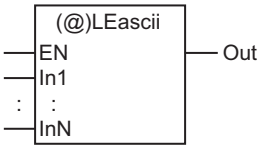
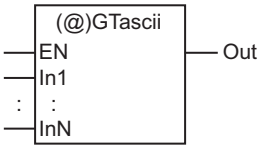
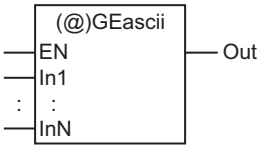
使用上の注意

- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- 「In1」、「In2」には、ASCII コードだけで構成された文字列を指定してください。

LTascii / LEascii / GTascii / GEascii

複数の文字列の、大小比較を行います。

LTascii ： 大小比較(<)を行います。
LEascii ： 大小比較(<=)を行います。
GTascii ： 大小比較(>)を行います。
GEascii ： 大小比較(>=)を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LTascii	文字列比較 LT	FUN		Out:=LTascii(In1, ･･･, InN);
LEascii	文字列比較 LE	FUN		Out:=LEascii(In1, ･･･, InN);
GTascii	文字列比較 GT	FUN		Out:=GTascii(In1, ･･･, InN);
GEascii	文字列比較 GE	FUN		Out:=GEascii(In1, ･･･, InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	比較対象文字列	入力	比較する文字列 N は 2～5	データ型に従う	－	" *1
Out	比較結果	出力	比較結果	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN																				○
Out	○																			

機能

2 個以上 5 個以下の文字列「In1」～「InN」の大小を比較します。
各命令の出力「Out」の値は、つぎのようになります。

命令	「Out」の値
LTascii	「In1」＜「In2」＜・・＜「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
LEascii	「In1」<=「In2」<=・・<=「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
GTascii	「In1」＞「In2」＞・・＞「InN」のとき、「Out」は TRUE、それ以外のとき FALSE
GEascii	「In1」>=「In2」>=・・>=「InN」のとき、「Out」は TRUE、それ以外のとき FALSE

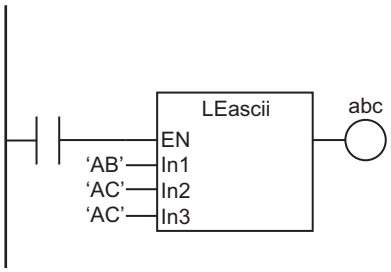
大小比較は、文字コードを用いて行います。比較手順は以下のとおりです。
まず、各文字列の 1 文字めの文字コードを比較します。文字コードが異なっていれば、その文字コードの大小が、それらの文字列の大小関係になります。
もし 1 文字めの文字コードが同じだった場合、2 文字め、3 文字めと、異なる文字コードが見つかるまで順に比較していきます。
各文字列の長さが異なっている場合、長い方の文字列に合わせて、短い方の文字列の末尾に NULL 文字(16#00)を埋めて、比較を行います。

いくつかの文字列の大小関係を、示します。
'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)

文字列の中に多バイト文字が使われている場合は、その文字を 1 バイトずつに分割して解釈します。たとえば、2 バイト文字 16#C281 は、16#C2 と 16#81 という 2 つの 1 バイト文字があると解釈します。

LEascii 命令で、「In1」='AB'、「In2」='AC'、「In3」='AC'の場合の例を、以下に示します。変数「abc」の値は、TRUE になります。

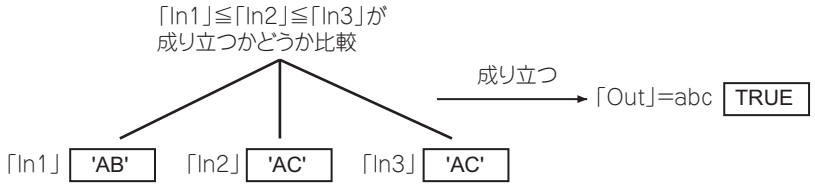
LD



ST

```
abc:=LEascii('AB', 'AC', 'AC');
```

「In1」≦「In2」≦「In3」が成り立つかどうか比較します。
成り立つので、abcの値はTRUEになります。



参考

文字列比較の命令は、複数の文字列を、文字コード順に並び替える場合などに便利です。たとえば、英文字は、文字コードの順番がアルファベットの順番と同じなので、アルファベット順での並び替えが可能です。



参考

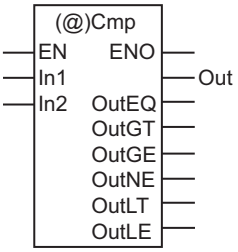
「LT,< / LE,<= / GT,> / GE,>= (P.2-100)」命令の文字列比較を使う方法もあります。
LT,< / LE,<= / GT,> / GE,>=命令の文字列比較と、本命令とは、まったく同じ仕様です。

使用上の注意

- 本命令は、回路の最終段には使用できません。使用した場合は、Sysmac Studio で異常が発生し、コントローラに転送できません。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Out」の値は FALSE になります。
- 「In1」～「InN」には、ASCII コードだけで構成された文字列を指定してください。

Cmp

2 つの数値を比較します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Cmp	比較	FUN		Out:=Cmp(In1, In2, OutEQ, OutGT, OutGE, OutNE, OutLT, OutLE); Out は省略可能

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1, In2	比較対象	入力	比較する数値	データ型に従う	−	*1
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−
OutEQ	=フラグ		=フラグ	データ型に従う		
OutGT	>フラグ		>フラグ			
OutGE	>=フラグ		>=フラグ			
OutNE	<>フラグ		<>フラグ			
OutLT	<フラグ		<フラグ			
OutLE	<=フラグ		<=フラグ			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1, In2						○	○	○	○	○	○	○	○	○	○					
Out	○																			
OutEQ	○																			
OutGT	○																			
OutGE	○																			
OutNE	○																			
OutLT	○																			
OutLE	○																			

機能

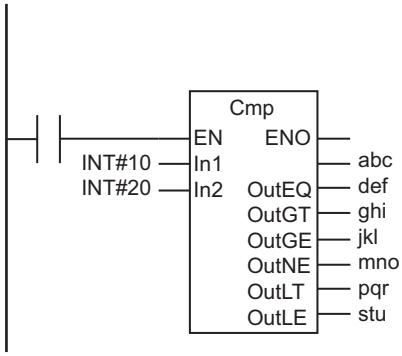
2 つの数値「In1」と「In2」を比較し、各フラグの値を出力します。
各フラグの値は、つぎのようになります。

フラグ	値
OutEQ	「In1」=「In2」のとき TRUE、それ以外のとき FALSE
OutGT	「In1」>「In2」のとき TRUE、それ以外のとき FALSE

フラグ	値
OutGE	「In1」>=「In2」のとき TRUE、それ以外のとき FALSE
OutNE	「In1」<>「In2」のとき TRUE、それ以外のとき FALSE
OutLT	「In1」<「In2」のとき TRUE、それ以外のとき FALSE
OutLE	「In1」<=「In2」のとき TRUE、それ以外のとき FALSE

「In1」=INT#10、「In2」=INT#20 の場合の例を、以下に示します。変数「def」、「ghi」、「jkl」の値は FALSE、「abc」、「mno」、「pqr」、「stu」の値は TRUE になります。

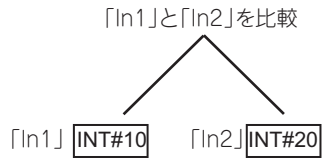
LD



ST

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

「In1」と「In2」を比較します。
各判定条件に従って、結果は以下のようになります。



「Out」	常にTRUE
「OutEQ」	「In1」=「In2」ではないのでFALSE
「OutGT」	「In1」>「In2」ではないのでFALSE
「OutGE」	「In1」≧「In2」ではないのでFALSE
「OutNE」	「In1」≠「In2」なのでTRUE
「OutLT」	「In1」<「In2」なのでTRUE
「OutLE」	「In1」≦「In2」なのでTRUE

→

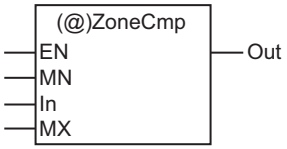
「Out」 =abc	TRUE
「OutEQ」=def	FALSE
「OutGT」=ghi	FALSE
「OutGE」=jkl	FALSE
「OutNE」=mno	TRUE
「OutLT」=pqr	TRUE
「OutLE」=stu	TRUE

使用上の注意

- 「In1」と「In2」のデータ型が異なる場合、両者のデータ型の有効範囲を包含するデータ型に型拡張した後、比較を行います。
- 「In1」、「In2」が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- +∞どうし、または-∞どうしは、等しいと判断します。
- 「In1」、「In2」のいずれかの値が非数の場合、「OutEQ」、「OutGT」、「OutGE」、「OutNE」、「OutLT」、「OutLE」の値は FALSE になります。

ZoneCmp

比較データが、指定した上限値と下限値の間にあるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ZoneCmp	領域比較	FUN		Out:=ZoneCmp(MN, In, MX);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
MN	下限値	入力	下限値	データ型に従う	−	0
In	比較データ		比較する数値			*1
MX	上限値		上限値			0
Out	比較結果	出力	比較結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
In						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
MX						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Out	○																			

機能

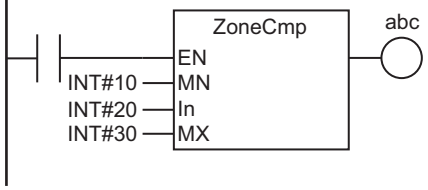
比較データ「In」が、上限値「MX」と下限値「MN」の間にあるかどうかを判定します。
「MX」≧「In」≧「MN」であるとき、「Out」の値は TRUE、そうでなければ FALSE となります。

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

データ型	大小関係
TIME	値が大きい方が、大きいと判断します。
DATE, TOD, DT	日付や時刻が後の方が大きいと判断します。

「MN」=INT#10、「In」=INT#20、「MX」=INT#30 の場合の例を、以下に示します。変数「abc」の値は、TRUE になります。

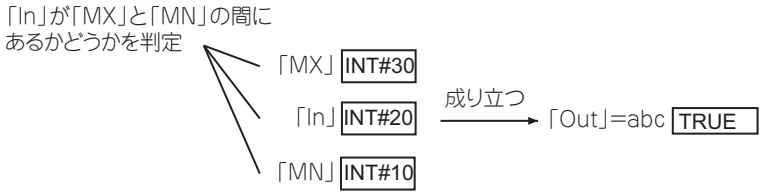
LD



ST

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

「MX」≧「In」≧「MN」が成り立つかどうかを判定します。
成り立つので、abcの値はTRUEになります。



参考

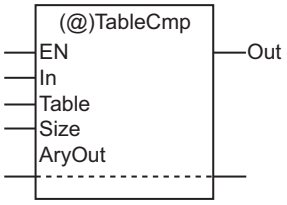
TIME 型、DT 型、TOD 型の大小比較を行う場合、比較する値どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- 「In」、「MX」、「MN」のデータ型が異なる場合、すべてのデータ型の有効範囲を包含するデータ型に型拡張した後、比較を行います。
- 「In」、「MX」、「MN」が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- TIME 型、DATE 型、TOD 型、DT 型は、同じデータ型どうしの比較しかできません。異なるデータ型を指定すると、ビルド時に異常になります。
- +∞どうし、または-∞どうしは、等しいと判断します。
- 「In」の値が非数の場合、「Out」の値は FALSE になります。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- つぎの場合、異常となり、「Out」の値は FALSE になります。
  a)「MN」の値が、「MX」の値より大きい場合。
  b)「MX」、「MN」のいずれかが非数の場合。

TableCmp

比較データと、比較テーブルで指定された複数の定義区間とを比較します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TableCmp	テーブル比較	FUN		Out:=TableCmp(In, Table, Size, AryOut);

変数

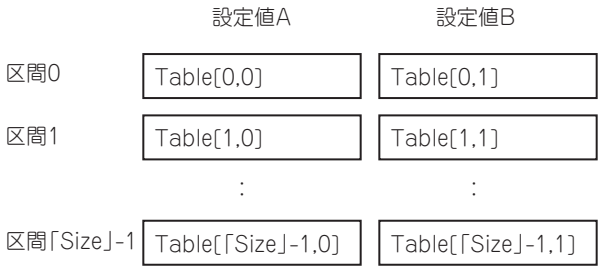
	名称	入／出力	内容	有効範囲	単位	初期値
In	比較データ	入力	比較する数値	データ型に従う	−	*1
Table[] 2 次元配列	比較テーブル		各定義区間を要素とする 2 次元配列			
Size	比較サイズ		「In」と比較する Table[]の要素数			1
AryOut[] 配列	個別比較結果配列	入出力	Table[]の各要素の比較結果 TRUE: 一致 FALSE: 不一致	データ型に従う	−	−
Out	比較結果	出力	TRUE: Table[]のすべての要素が「In」と一致 FALSE: 一致しない要素が 1 個以上あり	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

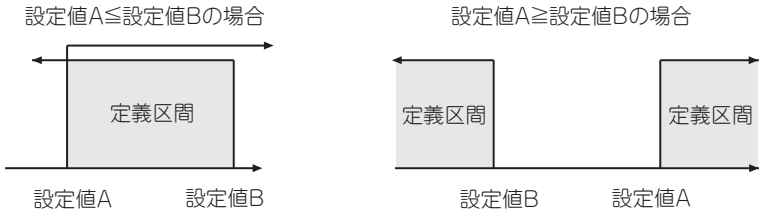
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Table[] 2 次元配列	「In」と同じデータ型の要素を持つ 2 次元配列																			
Size							○													
AryOut[] 配列	○																			
Out	○																			

機能

比較データ「In」と、比較テーブル Table[]で指定された「Size」組の定義区間とを比較します。
Table[]は 2 次元配列で、第 1 次元が定義区間の番号、第 2 次元の 0 番めの要素が定義区間の設定値 A、1 番めの要素が定義区間の設定値 B を示します。

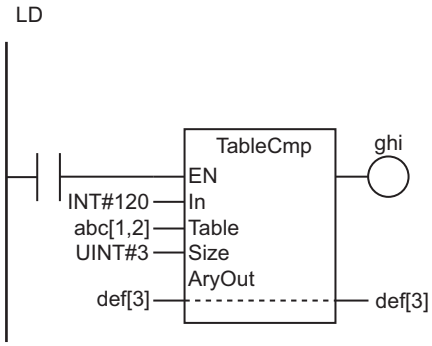


設定値 A と設定値 B による定義区間の指定方法を、以下の図に示します。設定値 A および設定値 B の値そのものは、定義区間内に含まれます。



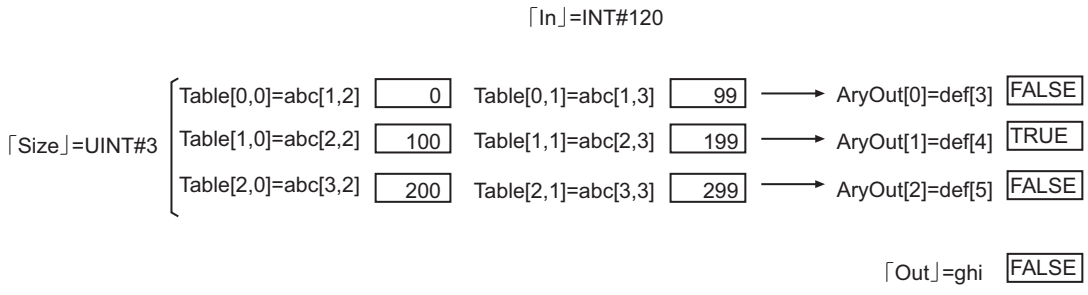
「In」と Table[]との比較結果は、個別比較結果配列 AryOut[]に格納されます。「In」が、i 番めの定義区間に収まっていれば AryOut[i]の値が TRUE に、収まっていなければ AryOut[i]の値が FALSE になります。そして、AryOut[]の「Size」個の要素がすべて TRUE なら、比較結果「Out」の値が TRUE に、そうでなければ FALSE になります。

「In」=INT#120、「Size」=UINT#3 の場合の例を、以下に示します。



ST

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



使用上の注意

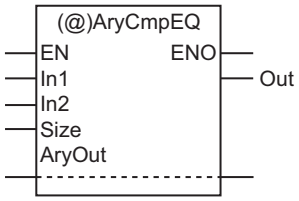
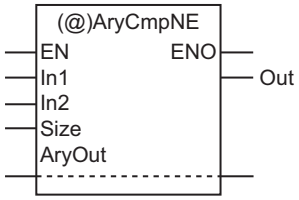
- 「In」と、Table[]の要素のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- Table[]は、必ず 2 次元配列にしてください。
- Table[]の第 2 次元の配列の大きさが 3 以上の場合、第 2 次元の 2 番め以降の要素は無視されます。

- AryOut[]の配列の大きさが「Size」以上の場合、AryOut[0]～AryOut[「Size」-1]に比較結果が格納されます。その他の配列要素は、変化しません。
- 符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)の比較はできません。
- 比較対象が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 「Size」の値が 0 の場合、「Out」の値は FALSE になり、AryOut[]は変化しません。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- つぎの場合、異常となります。「Out」は FALSE となります。
  a)「Size」の値が、AryOut[]の配列の大きさを超える場合。
  b)「Size」の値が、Table[]の第 1 次元の配列の大きさを超える場合。
  c) Table[]の第 2 次元の配列の大きさが、1 の場合。

AryCmpEQ / AryCmpNE

2 つの配列の各要素の、比較を行います。

AryCmpEQ ： 等しいかどうかを判定します。
AryCmpNE ： 異なるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCmpEQ	配列一括比較 EQ	FUN		AryCmpEQ(In1, In2, Size, AryOut);
AryCmpNE	配列一括比較 NE	FUN		AryCmpNE(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[], In2[] 配列	比較配列	入力	比較する数値を要素とする配列	データ型に従う	−	*1
Size	比較要素数		比較する要素数			1
AryOut[] 配列	比較結果配列	入出力	比較結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○					
In2[] 配列	In1[]と同じデータ型の配列																			
Size							○													
AryOut[] 配列	○																			
Out	○																			

機能

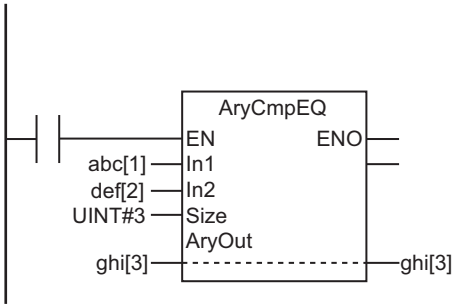
2 つの配列 In1[0]～In1[「Size」-1]、In2[0]～In2[「Size」-1]の、同じ要素番号どうしを比較します。
比較結果は、比較結果配列 AryOut[0]～AryOut[「Size」-1]の、対応する要素番号に格納されます。

各命令の AryOut[i]の値は、つぎのようになります。

命令	AryOut[i]の値
AryCmpEQ	In1[i]=In2[i]のとき TRUE、それ以外のとき FALSE
AryCmpNE	In1[i]≠In2[i]のとき TRUE、それ以外のとき FALSE

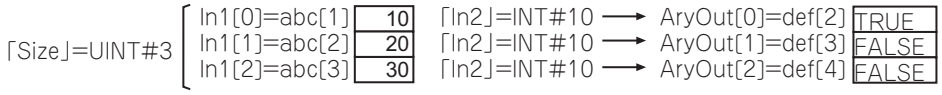
AryCmpEQ 命令で、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



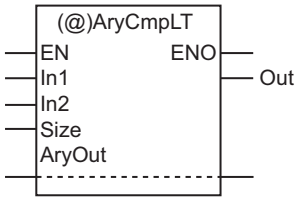
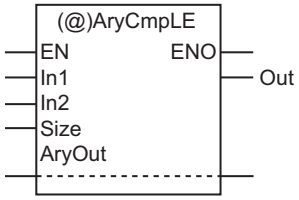
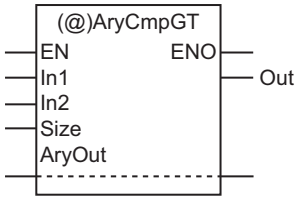
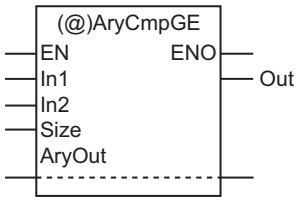
使用上の注意

- In1[]と In2[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- AryOut[]の配列の大きさは、「Size」以上にしてください。
- In1[]、In2[]が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）In1[]、In2[]、AryOut[]のいずれかの配列の大きさが、「Size」より小さい場合。

AryCmpLT / AryCmpLE / AryCmpGT / AryCmpGE

2 つの配列の各要素の、大小比較を行います。

AryCmpLT ： 大小比較(<)を行います。
AryCmpLE ： 大小比較(<=)を行います。
AryCmpGT ： 大小比較(>)を行います。
AryCmpGE ： 大小比較(>=)を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCmpLT	配列一括比較 LT	FUN		AryCmpLT(In1, In2, Size, AryOut);
AryCmpLE	配列一括比較 LE	FUN		AryCmpLE(In1, In2, Size, AryOut);
AryCmpGT	配列一括比較 GT	FUN		AryCmpGT(In1, In2, Size, AryOut);
AryCmpGE	配列一括比較 GE	FUN		AryCmpGE(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[], In2[] 配列	比較配列	入力	比較する数値を要素とする配列	データ型に従う	－	*1
Size	比較要素数		比較する要素数			1
AryOut[] 配列	比較結果配列	入出力	比較結果配列	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2[] 配列	In1[]と同じデータ型の配列																			
Size							○													
AryOut[] 配列	○																			
Out	○																			

機能

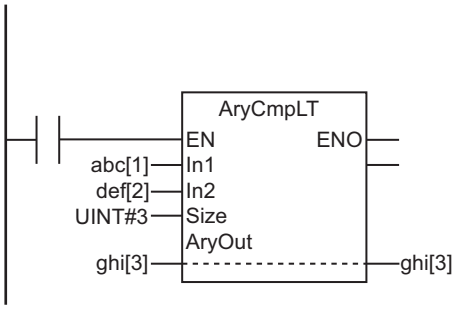
2 つの配列 In1[0]～In1[「Size」-1]、In2[0]～In2[「Size」-1]の、同じ要素番号どうしの大小を比較します。比較結果は、比較結果配列 AryOut[0]～AryOut[「Size」-1]の、対応する要素番号に格納されます。

各命令の AryOut[i]の値は、つぎのようになります。

命令	AryOut[i]の値
AryCmpLT	In1[i]<In2[i] のとき TRUE、それ以外のとき FALSE
AryCmpLE	In1[i]<=In2[i] のとき TRUE、それ以外のとき FALSE
AryCmpGT	In1[i]>In2[i] のとき TRUE、それ以外のとき FALSE
AryCmpGE	In1[i]>=In2[i] のとき TRUE、それ以外のとき FALSE

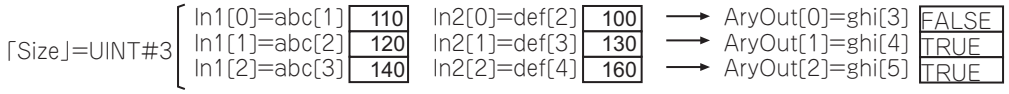
AryCmpLT 命令で、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



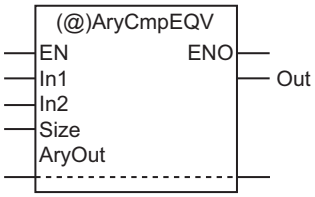
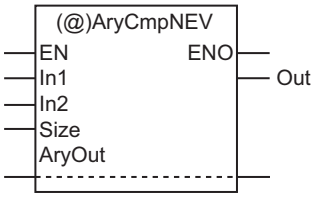
使用上の注意

- In1[]と In2[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- AryOut[]の配列の大きさは、「Size」以上にしてください。
- In1[]、In2[]が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）In1[]、In2[]、AryOut[]のいずれかの配列の大きさが、「Size」より小さい場合。

AryCmpEQV / AryCmpNEV

配列の各要素と数値との、比較を行います。

AryCmpEQV ： 等しいかどうかを判定します。
AryCmpNEV ： 異なるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCmpEQV	配列要素比較 EQ	FUN		AryCmpEQV(In1, In2, Size, AryOut);
AryCmpNEV	配列要素比較 NE	FUN		AryCmpNEV(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列	比較配列	入力	比較する数値を要素とする配列	データ型に従う	－	*1
In2	比較数値		比較する数値			
Size	比較要素数		比較する要素数			1
AryOut[] 配列	比較結果配列	入出力	比較結果配列	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○					
In2	In1[]の要素と同じデータ型																			
Size							○													
AryOut[] 配列	○																			
Out	○																			

機能

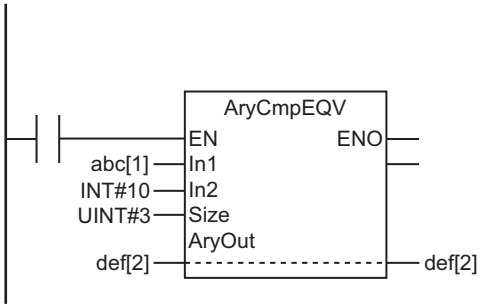
比較配列 In1[0]～In1[「Size」-1]と、比較数値「In2」とを比較します。
比較結果は、比較結果配列 AryOut[0]～AryOut[「Size」-1]の、対応する要素番号に格納されます。

各命令の AryOut[i]の値は、つぎのようになります。

命令	AryOut[i]の値
AryCmpEQV	In1[i] = In2 のとき TRUE、それ以外のとき FALSE
AryCmpNEV	In1[i] ≠ In2 のとき TRUE、それ以外のとき FALSE

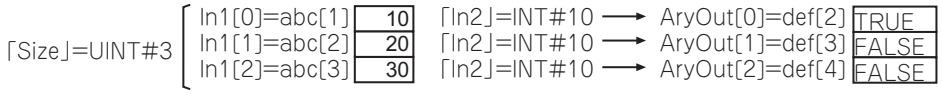
AryCmpEQV 命令で、「In2」=INT#10、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



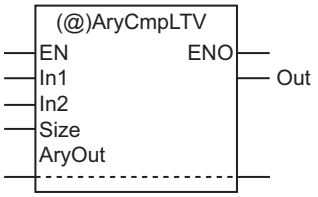
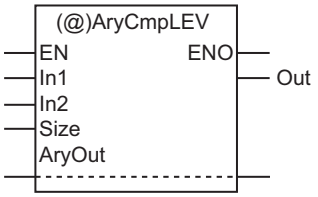
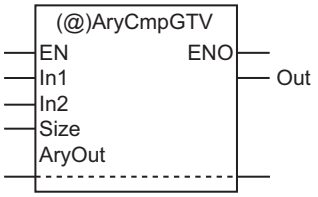
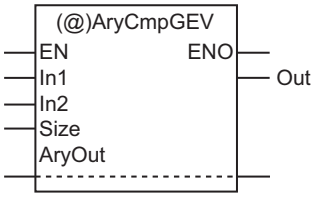
使用上の注意

- In1[]と「In2」のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- AryOut[]の配列の大きさは、「Size」以上にしてください。
- In1[]、「In2」が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）In1[]、AryOut[]のいずれかの配列の大きさが、「Size」より小さい場合。

AryCmpLTV / AryCmpLEV / AryCmpGTV / AryCmpGEV

配列の各要素と数値との、大小比較を行います。

AryCmpLTV ： 大小比較(<)を行います。
AryCmpLEV ： 大小比較(<=)を行います。
AryCmpGTV ： 大小比較(>)を行います。
AryCmpGEV ： 大小比較(>=)を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCmpLTV	配列要素比較 LT	FUN		AryCmpLTV(In1, In2, Size, AryOut);
AryCmpLEV	配列要素比較 LE	FUN		AryCmpLEV(In1, In2, Size, AryOut);
AryCmpGTV	配列要素比較 GT	FUN		AryCmpGTV(In1, In2, Size, AryOut);
AryCmpGEV	配列要素比較 GE	FUN		AryCmpGEV(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列	比較配列	入力	比較する数値を要素とする配列	データ型に従う	−	*1
In2	比較数値		比較する数値			
Size	比較要素数		比較する要素数			1
AryOut[] 配列	比較結果配列	入出力	比較結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2	In1[]の要素と同じデータ型																			
Size							○													
AryOut[] 配列	○																			
Out	○																			

機能

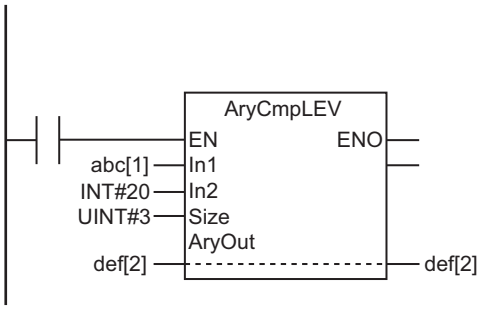
比較配列 In1[0]～In1[「Size」-1]と、比較数値「In2」との大小を比較します。
比較結果は、比較結果配列 AryOut[0]～AryOut[「Size」-1]の、対応する要素番号に格納されます。

各命令の AryOut[i]の値は、つぎのようになります。

命令	AryOut[i]の値
AryCmpLTV	In1[i] < In2 のとき TRUE、それ以外のとき FALSE
AryCmpLEV	In1[i] <= In2 のとき TRUE、それ以外のとき FALSE
AryCmpGTV	In1[i] > In2 のとき TRUE、それ以外のとき FALSE
AryCmpGEV	In1[i] >= In2 のとき TRUE、それ以外のとき FALSE

AryCmpLEV 命令で、「In2」=INT#20、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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

「Size」=UINT#3

In1[0]=abc[1]	10	「In2」=INT#20 ⟶	AryOut[0]=def[2]	TRUE
In1[1]=abc[2]	20	「In2」=INT#20 ⟶	AryOut[1]=def[3]	TRUE
In1[2]=abc[3]	30	「In2」=INT#20 ⟶	AryOut[2]=def[4]	FALSE

使用上の注意

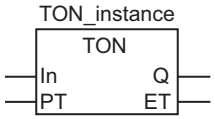
- In1[]と「In2」のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- AryOut[]の配列の大きさは、「Size」以上にしてください。
- In1[]、「In2」が実数で、割り切れない除算結果などを含んでいると、誤差のため、処理結果が期待どおりにならないことがあります。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）In1[]、AryOut[]のいずれかの配列の大きさが、「Size」より小さい場合。

タイマ命令

命令	名称	ページ
TON	オンディレータイマ	P.2-126
TOF	オフディレータイマ	P.2-131
TP	パルス出力	P.2-134
AccumulationTimer	積算タイマ	P.2-137
Timer	100ms タイマ	P.2-141

TON

起動から設定時間経過後に TRUE を出力するタイマです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TON	オンディレータイマ	FB		TON_instance(In, PT, Q, ET);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	タイマ入力	入力	TRUE: タイマ起動指示 FALSE: タイマリセット指示	データ型に従う	−	FALSE
PT	設定時間		タイマ起動から、「Q」が TRUE に変わるまでの時間	*1	ms	0
Q	タイマ出力	出力	TRUE: タイマ出力 ON FALSE: タイマ出力 OFF	データ型に従う	−	−
ET	経過時間		タイマ起動からの経過時間	*1	ms	

*1. T#0ms～T#106751d_23h_47m_16s_854.775807ms

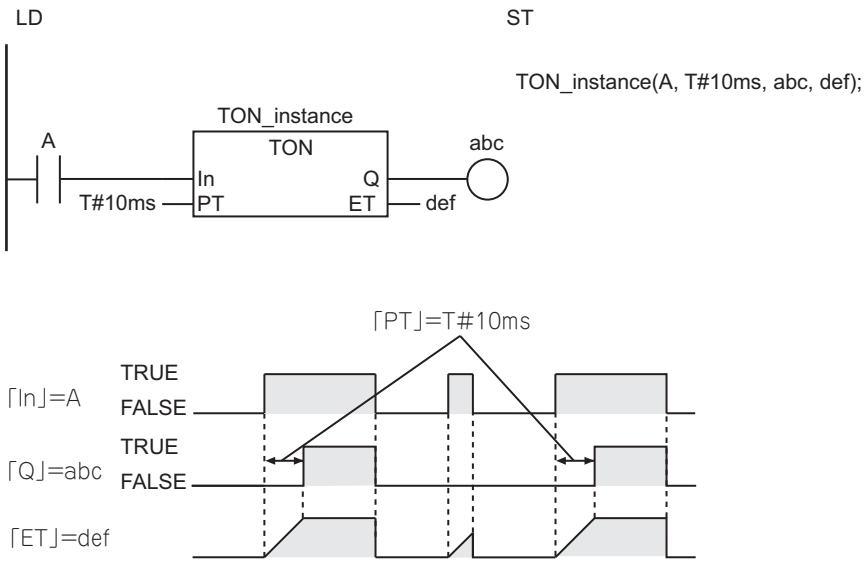
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Q	○																			
ET																○				

機能

起動から設定時間経過後に TRUE を出力するタイマです。設定時間の最小単位は ns です。
タイマ入力「In」が FALSE から TRUE に変化するとタイマが起動し、経過時間「ET」が時間とともに増加していきます。
「ET」が設定時間「PT」に達すると、タイマ出力「Q」が TRUE になります。「ET」の増加は、その時点で止まります。
「In」が FALSE になると、タイマがリセットします。「ET」は 0 になり、「Q」が FALSE になります。

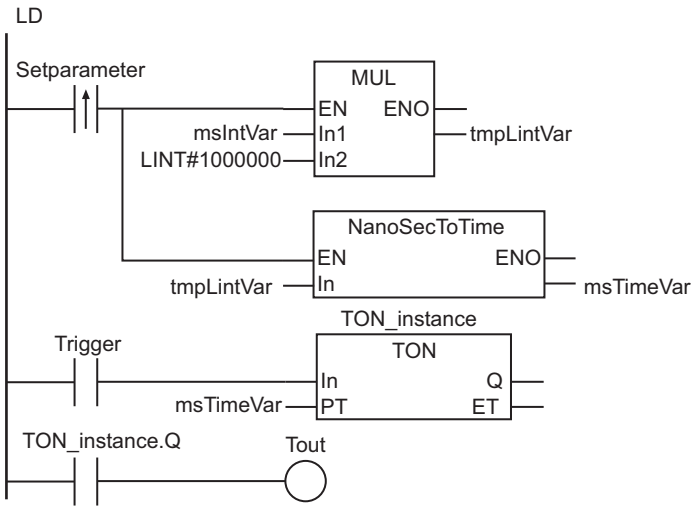
タイマ起動後、「ET」が「PT」に達する前でも「In」が FALSE になると、タイマはリセットされます。

「PT」が T#10ms の場合の、記述例とタイミングチャートを、以下に示します。変数「A」が TRUE に変化してから 10ms 後に、変数「abc」の値が TRUE になります。



参考

- タイマ起動後すぐにタイマ出力が TRUE になり、設定時間経過後に FALSE になるタイマが必要な場合は、「TP (P.2-134)」命令を使用してください。
- 「In」が FALSE になったときにタイマが起動し、経過時間が設定時間に達するとタイマ出力が FALSE になるタイマが必要な場合は、「TOF (P.2-131)」命令を使用してください。
- 命令実行時間を短縮したい場合は、100ms 単位で計時を行う「Timer (P.2-141)」命令を使用してください。
- TIME 型に対応していないタッチパネルなどと接続する場合は、整数型で表された設定時間を TIME 型にいったん変換してから本命令に入力する必要があります。整数型から TIME 型への変換には、「NanoSecToTime (P.2-642)」命令を使用してください。また、TIME 型から整数型への変換には、「TimeToNanoSec (P.2-639)」命令を使用してください。どちらの命令も、時間の単位はナノ秒です。INT 型変数「msIntVar」が ms 単位の設定時間である場合のユーザプログラムを、以下に示します。



ST

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

使用上の注意

- 「PT」に対して「Q」が TRUE になる計時誤差は、-100ns ～ （100ns+1 タスク周期）です。
  上記の範囲の内訳は、以下のとおりです。
  a）±100ns は、「ET」の計時誤差。
  b）タスク周期ごとに「ET」の計時が「PT」に達したか判断しており、判断直後に「ET」の計時が「PT」に達した場合に+1 タスク周期遅れます。
- Sysmac Studio では、時間値が 0.001ms 単位で表示されますが、計時精度は 1ns です。
- 運転開始時に、すでに「In」の値が TRUE の場合、その時点から計時を開始します。
- 「PT」に T#0ms または負数が設定された場合、「In」の値が TRUE になった直後に「Q」が TRUE になります。
- 「In」の値が TRUE である間に、「PT」の値を変更することは、可能です。その場合の動作は、以下のようになります。

タイマの状態	「Q」の値	変更後の「PT」の値	動作
計時終了後	TRUE	−	「Q」の値は、TRUE のままです。 「ET」の値も、変化しません。変更前の「PT」の値のままです。
計時中	FALSE	PT ≧ ET	計時を続行します。「ET」の値が、変更後の「PT」の値に達した時点で、「Q」の値が TRUE になり、「ET」の増加が止まります。
		PT ＜ ET	直ちに「Q」の値が TRUE になります。 「ET」の増加も、直ちに止まります。

- 本命令がマスタコントロール領域に存在し、マスタコントロールによるリセットが実行された場合、タイマがリセットされます。「ET」の値が 0 になり、「Q」の値は FALSE になります。
- JMP 系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「ET」の値は更新されません。しかし、その間も計時は継続しています。したがって、つぎに本命令が実行されたときに、「ET」は正しい値に更新されます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Q」の値は FALSE になります。

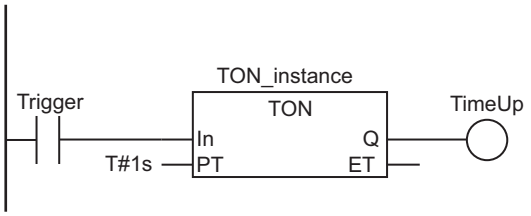
サンプルプログラム

1 個のオンディレータイマで時間を測る例

Trigger の値が TRUE になってから 1s 後に、TimeUp の値が TRUE になります。

● **LD**

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
TimeUp	BOOL	FALSE	タイマ出力
TON_instance	TON		



● ST（例 1）

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
TimeUp	BOOL	FALSE	タイマ出力
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;
```

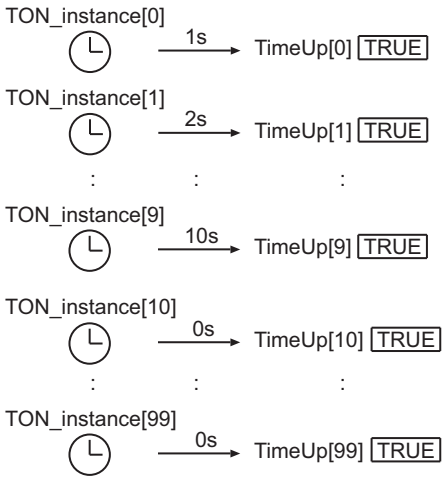
● ST（例 2）

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
TimeUp	BOOL	FALSE	タイマ出力
TON_instance	TON		

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

複数のオンディレータイマで時間を測る例

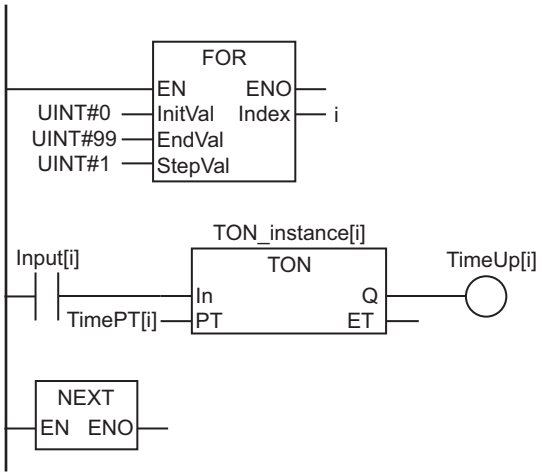
TON_instance[0] ～ TON_instance[99]の 100 個のオンディレータイマを用意します。それぞれのタイマを、タイマ入力 Input[0] ～ Input[99]の値を TRUE にすることで、起動させます。
最初の 10 個のタイマ TON_instance[0] ～ TON_instance[9]は、それぞれが起動してから、i+1 秒後(i=0 ～ 9)に TimeUp[i](i=0 ～ 9)の値を TRUE にします。
残りの 90 個のタイマ TON_instance[10] ～ TON_instance[99]は、起動後直ちに TimeUp[i](i=10 ～ 99)の値を TRUE にします。



● LD

名称	データ型	初期値	コメント
Input	ARRAY[0..99] OF BOOL	[100(FALSE)]	タイマ入力
TimeUp	ARRAY[0..99] OF BOOL	[100(FALSE)]	タイマ出力
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)]	設定時間

名称	データ型	初期値	コメント
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	インデックス



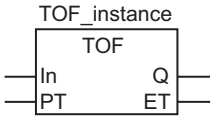
● ST

名称	データ型	初期値	コメント
Input	ARRAY[0..99] OF BOOL	[100(FALSE)]	タイマ入力
TimeUp	ARRAY[0..99] OF BOOL	[100(FALSE)]	タイマ出力
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)]	設定時間
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	インデックス

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

TOF

起動から設定時間経過後に FALSE を出力するタイマです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TOF TOF	オフディレータイマ	FB		TOF_instance(In, PT, Q, ET);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	タイマ入力	入力	TRUE: タイマリセット指示 FALSE: タイマ起動指示	データ型に従う	−	FALSE
PT	設定時間		タイマ起動から、「Q」が FALSE に変わるまでの時間	*1	ms	0
Q	タイマ出力	出力	TRUE: タイマ出力 ON FALSE: タイマ出力 OFF	データ型に従う	−	−
ET	経過時間		タイマ起動からの経過時間	*1	ms	

*1. T#0ms～T#106751d_23h_47m_16s_854.775807ms

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Q	○																			
ET																○				

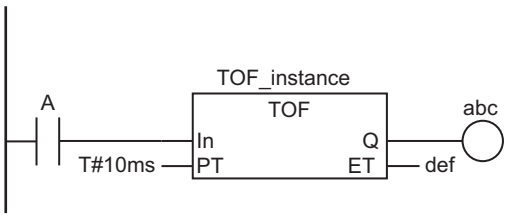
機能

起動から設定時間経過後に FALSE を出力するタイマです。設定時間の最小単位は ns です。
タイマ入力「In」が TRUE から FALSE に変化するとタイマが起動し、経過時間「ET」が時間とともに増加していきます。
「ET」が設定時間「PT」に達すると、タイマ出力「Q」が FALSE になります。「ET」の増加は、その時点で止まります。
「In」が TRUE になると、タイマがリセットします。「ET」は 0 になり、「Q」が TRUE になります。

タイマ起動後、「ET」が「PT」に達する前でも「In」が FALSE になると、タイマはリセットされます。

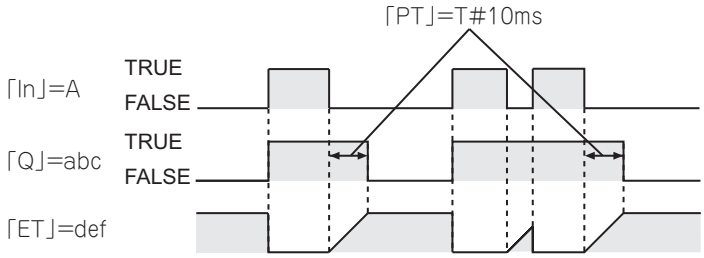
「PT」が T#10ms の場合の、記述例とタイミングチャートを、以下に示します。変数「A」が FALSE に変化してから 10ms 後に、変数「abc」の値が FALSE になります。

LD



ST

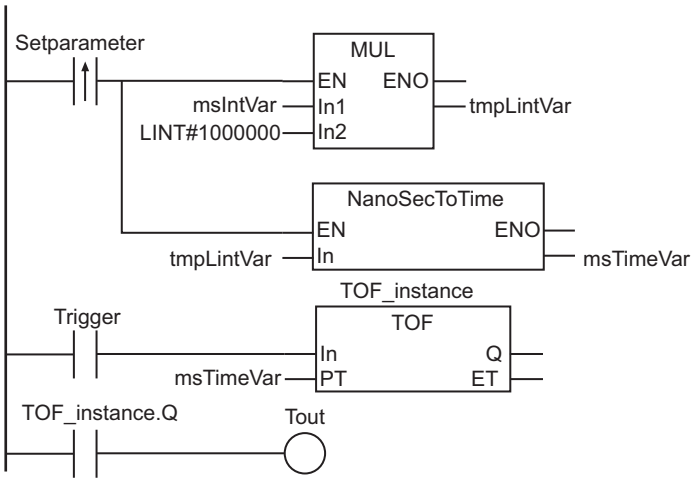
```
TOF_instance(A, T#10ms, abc, def);
```



参考

- タイマ起動後すぐにタイマ出力が TRUE になり、設定時間経過後に FALSE になるタイマが必要な場合は、「TP (P.2-134)」命令を使用してください。
- 「In」が TRUE になったときにタイマが起動し、経過時間が設定時間に達するとタイマ出力が TRUE になるタイマが必要な場合は、「TON (P.2-126)」命令を使用してください。
- TIME 型に対応していないタッチパネルなどと接続する場合は、整数型で表された設定時間を TIME 型にいったん変換してから本命令に入力する必要があります。整数型から TIME 型への変換には、「NanoSecToTime (P.2-642)」命令を使用してください。また、TIME 型から整数型への変換には、「TimeToNanoSec (P.2-639)」命令を使用してください。どちらの命令も、時間の単位はナノ秒です。INT 型変数 msIntVar が ms 単位の設定時間である場合のユーザプログラムを、以下に示します。

LD



ST

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

使用上の注意

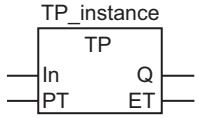
- 「PT」に対して「Q」が TRUE になる計時誤差は、-100ns ～ （100ns+1 タスク周期）です。
  上記の範囲の内訳は、以下のとおりです。
  a）±100ns は、「ET」の計時誤差。
  b）タスク周期ごとに「ET」の計時が「PT」に達したか判断しており、判断直後に「ET」の計時が「PT」に達した場合に+1 タスク周期遅れます。
- Sysmac Studio では、時間値が 0.001ms 単位で表示されますが、計時精度は 1ns です。
- 「PT」に T#0ms または負数が設定された場合、「In」の値が FALSE になった直後に「Q」が FALSE になります。
- 本命令の実行直後、「In」の値が TRUE ならば、「Q」の値は TRUE になります。「Q」の値が FALSE になるのは、タイマ起動後、「PT」だけ時間が経過したときです。
- 「In」の値が FALSE である間に、「PT」の値を変更することは、可能です。その場合の動作は、以下のようになります。

タイマの状態	「Q」の値	変更後の「PT」の値	動作
計時終了後	FALSE	－	「Q」の値は、FALSE のままです。 「ET」の値も、変化しません。変更前の「PT」の値のままです。
計時中	TRUE	PT ≧ ET	計時を続行します。「ET」の値が、変更後の「PT」の値に達した時点で、「Q」の値が FALSE になり、「ET」の増加が止まります。
		PT ＜ ET	直ちに「Q」の値が FALSE になります。 「ET」の増加も、直ちに止まります。

- 本命令がマスタコントロール領域に存在し、マスタコントロールによるリセットが実行された場合、以下のように動作します。
  a)「ET」の値が 0 になり、「Q」の値は TRUE になります。
  b)「Q」を後段の Out 命令に接続している場合、その Out 命令には FALSE が入力されます。
  c）リセット解除後に計時を開始します。
- JMP 系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「ET」の値は更新されません。しかし、その間も計時は継続しています。したがって、つぎに本命令が実行されたときに、「ET」は正しい値に更新されます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

TP

起動から設定時間だけ TRUE を出力するタイマです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TP	パルス出力	FB		TP_instance(In, PT, Q, ET);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	タイマ入力	入力	TRUE: タイマ起動指示 FALSE: タイマリセット指示	データ型に従う	−	FALSE
PT	設定時間		「Q」が TRUE を保持する時間	*1	ms	0
Q	タイマ出力	出力	TRUE: タイマ出力 ON FALSE: タイマ出力 OFF	データ型に従う	−	−
ET	経過時間		タイマ起動からの経過時間	*1	ms	

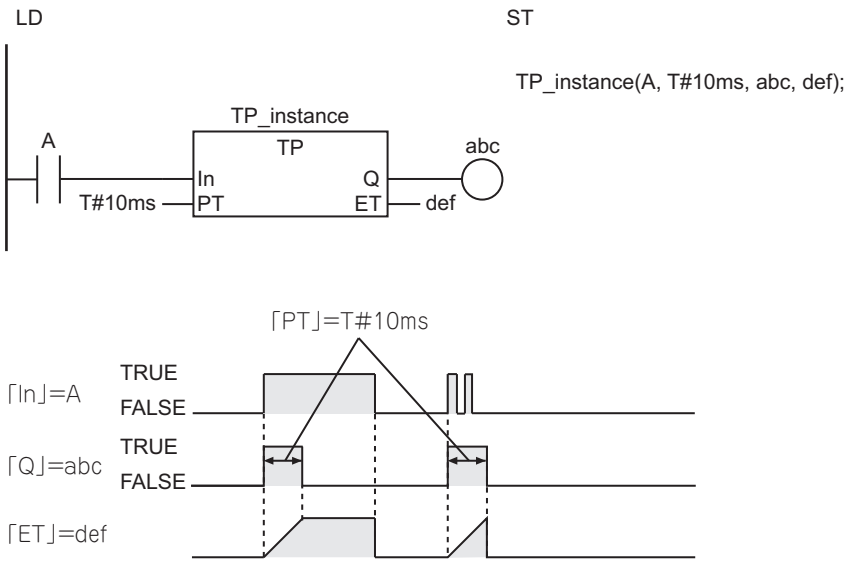
*1. T#0ms～T#106751d_23h_47m_16s_854.775807ms

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Q	○																			
ET																○				

機能

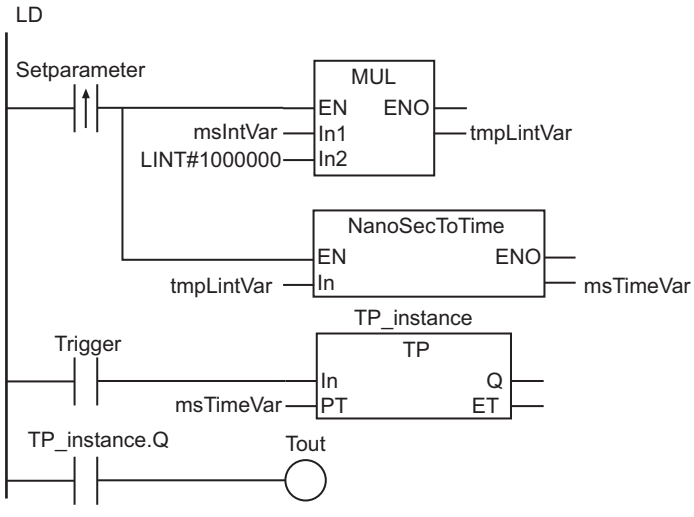
起動から設定時間だけ TRUE を出力するタイマです。設定時間の最小単位は ns です。
タイマ入力「In」が FALSE から TRUE に変化するとタイマが起動し、タイマ出力「Q」が TRUE になります。そして、経過時間「ET」が時間とともに増加していきます。
「ET」が設定時間「PT」に達すると、タイマ出力「Q」が FALSE になります。「ET」の増加は、その時点で止まります。
「In」が FALSE になると、タイマがリセットします。「ET」は 0 になります。
ただし、起動後、「ET」が「PT」に達する以前に「In」が FALSE になった場合は、タイマはリセットしません。

「PT」が T#10ms の場合の、記述例とタイミングチャートを、以下に示します。変数「A」が TRUE に変化すると変数「abc」の値が TRUE になり、10ms 後に FALSE になります。



参考

- 「In」が TRUE になったときにタイマが起動し、経過時間が設定時間に達するとタイマ出力が TRUE になるタイマが必要な場合は、「TON (P.2-126)」命令を使用してください。
- 「In」が FALSE になったときにタイマが起動し、経過時間が設定時間に達するとタイマ出力が FALSE になるタイマが必要な場合は、「TOF (P.2-131)」命令を使用してください。
- TIME 型に対応していないタッチパネルなどと接続する場合は、整数型で表された設定時間を TIME 型にいったん変換してから本命令に入力する必要があります。整数型から TIME 型への変換には、「NanoSecToTime (P.2-642)」命令を使用してください。また、TIME 型から整数型への変換には、「TimeToNanoSec (P.2-639)」命令を使用してください。どちらの命令も、時間の単位はナノ秒です。INT 型変数「msIntVar」が ms 単位の設定時間である場合のユーザプログラムを、以下に示します。



ST

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

使用上の注意

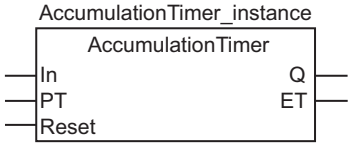
- 「PT」に対して「Q」が TRUE になる計時誤差は、-100ns ～ （100ns+1 タスク周期）です。
  上記の範囲の内訳は、以下のとおりです。
  a）±100ns は、「ET」の計時誤差。
  b）タスク周期ごとに「ET」の計時が「PT」に達したか判断しており、判断直後に「ET」の計時が「PT」に達した場合に+1 タスク周期遅れます。
- Sysmac Studio では、時間値が 0.001ms 単位で表示されますが、計時精度は 1ns です。
- 運転開始時に、すでに「In」の値が TRUE の場合、その時点から計時を開始します。
- 「PT」に T#0ms または負数が設定された場合、「In」の値が TRUE になっても、「Q」は TRUE になりません。
- 「In」の値が TRUE である間に、「PT」の値を変更することは、可能です。その場合の動作は、以下のようになります。

タイマの状態	「Q」の値	変更後の「PT」の値	動作
計時終了後	FALSE	−	「Q」の値は、FALSE のままです。 「ET」の値も、変化しません。変更前の「PT」の値のままです。
計時中	TRUE	PT ≧ ET	計時を続行します。「ET」の値が、変更後の「PT」の値に達した時点で、「Q」の値が FALSE になり、「ET」の増加が止まります。
		PT ＜ ET	直ちに「Q」の値が FALSE になります。 「ET」の増加も、直ちに止まります。

- 本命令がマスタコントロール領域に存在し、マスタコントロールによるリセットが実行された場合、計時中であれば、完了まで計時を継続します。その後、「ET」の値が 0 になり、「Q」の値は FALSE になります。ただし、「Q」を後段の Out 命令に接続している場合、「Q」の値が TRUE であっても、Out 命令には FALSE が入力されます。
- JMP 系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「ET」の値は更新されず、計時を行いません。つぎに本命令が実行されたときに、計時を再開します。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

AccumulationTimer

タイマ入力が TRUE である時間を、積算するタイマです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AccumulationTimer	積算タイマ	FB		AccumulationTimer_instance(In, PT, Reset, Q, ET);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	タイマ入力	入力	TRUE: タイマ作動 FALSE: タイマ停止	データ型に従う	−	FALSE
PT	設定時間		計時の最大値	*1	ms	0
Reset	リセット		TRUE: タイマリセット FALSE: タイマ非リセット	データ型に従う	−	FALSE
Q	タイマ出力	出力	TRUE:「ET」が、「PT」に達した FALSE:「ET」が、「PT」に達していない	データ型に従う	−	−
ET	積算時間		積算時間	*1	ms	

*1. T#0ms～T#106751d_23h_47m_16s_854.775807ms

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Reset	○																			
Q	○																			
ET																○				

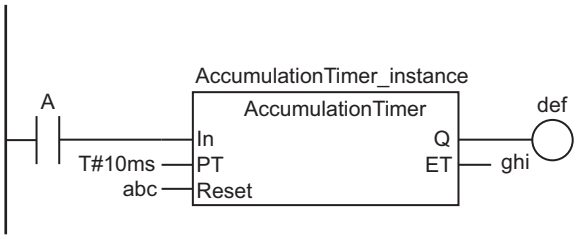
機能

タイマ入力「In」が TRUE である時間を、積算するタイマです。設定時間の最小単位は ns です。
リセット「Reset」が FALSE であるとき、「In」が FALSE から TRUE に変化するとタイマが起動し、積算時間「ET」が時間とともに増加していきます。
「In」が FALSE になると、タイマが停止します。しかし、その時点の「ET」の値は保持されます。
再び「In」が TRUE になると、タイマは再起動します。そして、「ET」は、保持されていた値から増加していきます。
「ET」が設定時間「PT」に達すると、タイマ出力「Q」が TRUE になります。「ET」の増加は、その時点で止まります。

「Reset」が TRUE になると、タイマがリセットします。「ET」の値は 0 になり、「Q」は FALSE になります。

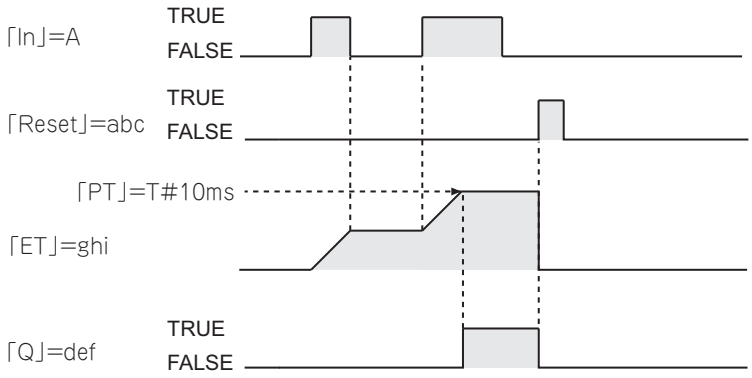
「PT」=T#10ms の場合の、記述例とタイミングチャートを、以下に示します。変数「A」が TRUE である積算時間が 10ms になると、変数「abc」の値が TRUE になります。

LD



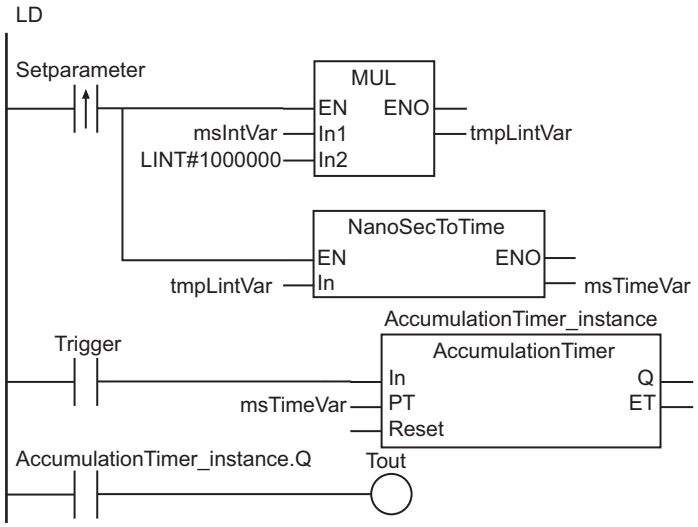
ST

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



参考

- 「In」が FALSE になったときに、タイマ出力および経過時間をリセットしたい場合は、「TON (P.2-126)」命令を使用してください。
- TIME 型に対応していないタッチパネルなどと接続する場合は、整数型で表された設定時間を TIME 型にいったん変換してから本命令に入力する必要があります。整数型から TIME 型への変換には、「NanoSecToTime (P.2-642)」命令を使用してください。また、TIME 型から整数型への変換には、「TimeToNanoSec (P.2-639)」命令を使用してください。どちらの命令も、時間の単位はナノ秒です。INT 型変数「msIntVar」が ms 単位の設定時間である場合のユーザプログラムを、以下に示します。



ST

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

使用上の注意

- 「PT」に対して「Q」が TRUE になる計時誤差は、-100ns ～ （100ns+1 タスク周期）です。
  上記の範囲の内訳は、以下のとおりです。
  a）±100ns は、「ET」の計時誤差。
  b）タスク周期ごとに「ET」の計時が「PT」に達したか判断しており、判断直後に「ET」の計時が「PT」に達した場合に+1 タスク周期遅れます。
- Sysmac Studio では、時間値が 0.001ms 単位で表示されますが、計時精度は 1ns です。
- 「In」と「Reset」がともに TRUE の場合、「Reset」が優先されます。つまり、「ET」の値は 0 になり、「Q」の値は FALSE になります。
- 運転開始時に、すでに「In」の値が TRUE の場合、その時点から計時を開始します。
- 「PT」に T#0ms または負数が設定された場合、「Q」が TRUE になります。
- 「ET」の値が「PT」の値に達する前に、「PT」の値を変更することは、可能です。その場合の動作は、以下のようになります。

タイマの状態	「Q」の値	変更後の「PT」の値	動作
計時終了後	TRUE	−	「Q」の値は、TRUE のままです。 「ET」の値も、変化しません。変更前の「PT」の値のままです。
計時中	FALSE	PT ≧ ET	計時を続行します。「ET」の値が、変更後の「PT」の値に達した時点で、「Q」の値が TRUE になり、「ET」の増加が止まります。
		PT ＜ ET	直ちに「Q」の値が TRUE になります。 「ET」の増加も、直ちに止まります。

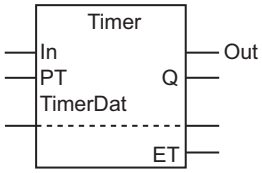
- 本命令がマスタコントロール領域に存在し、マスタコントロールによるリセットが実行された場合、以下のように動作します。
  a）タイマが停止します。「ET」および「Q」は、その時点の値を保持します。
  b）マスタコントロールによるリセットが解除されると、「ET」は、保持していた値から増加を再開します。
  c）「Q」を後段の Out 命令に接続している場合、「Q」の値が TRUE であっても、Out 命令には FALSE が入力されます。

d)「Reset」は、有効です。

- JMP 系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「ET」の値は更新されません。しかし、その間も計時は継続しています。したがって、つぎに本命令が実行されたときに、「ET」は正しい値に更新されます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

Timer

起動から設定時間経過後に TRUE を出力するタイマです。設定時間の最小単位は 100ms です。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Timer	100ms タイマ	FUN		Out:=Timer(In, PT, TimerDat, Q, ET);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	タイマ入力	入力	TRUE: タイマ起動指示 FALSE: タイマリセット指示	データ型に従う	−	FALSE
PT	設定時間		タイマ起動から、「Q」が TRUE に変わるまでの時間		ms	*1
TimerDat	タイマ状態	入出力	タイマの現在の状態	−	−	−
Out	戻り値	出力	TRUE: タイマ出力 ON FALSE: タイマ出力 OFF	データ型に従う	−	−
Q	タイマ出力		「Out」と同じ意味			
ET	残り時間		残り時間		ms	

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT							○													
TimerDat	構造体_sTimer																			
Out	○																			
Q	○																			
ET							○													

機能

起動から設定時間経過後に TRUE を出力するタイマです。ただし、設定時間の最小単位は 100ms です。

タイマ入力「In」の値が FALSE になると、タイマがリセットします。残り時間「ET」に設定時間「PT」がセットされ、タイマ出力「Q」の値が FALSE になります。
「In」が FALSE から TRUE に変化するとタイマが起動します。「ET」の値が、時間とともに減少していきます。

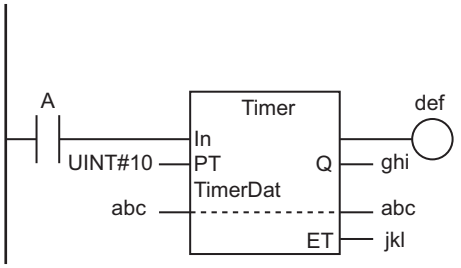
「ET」の値が 0 に達すると、タイマ出力「Q」が TRUE になります。「ET」の値の減少は、その時点で止まります。

タイマ起動後、「ET」が 0 に達する前でも「In」が FALSE になると、タイマはリセットされます。

タイマ状態「TimerDat」のデータ型は、構造体_sTimer です。

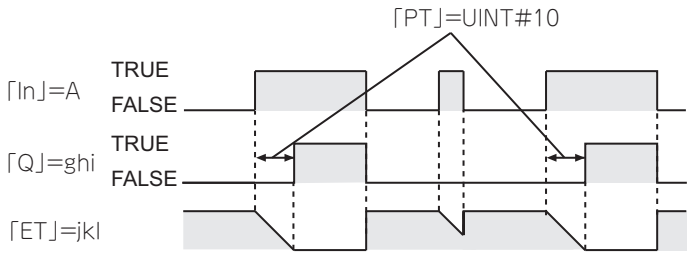
「PT」=UINT#10 の場合の、記述例とタイミングチャートを、以下に示します。変数「A」が TRUE に変化してから 1000ms（1s）後に、変数「ghi」の値が TRUE になります。

LD



ST

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



参考

より正確な計時を行う場合は、1ns 単位の計時を行う「TON (P.2-126)」命令を使用してください。TON 命令は、命令実行時に 1ns 単位の計時を行うため、Timer 命令より正確な計時が可能です。ただし、命令実行時間は、Timer 命令の方が短いです。

使用上の注意

- 計時は、本命令を記述した POU の先頭で行います。そのため、同一 POU 内のどこで本命令を実行しても、「ET」の値は同じです。
- 「PT」に対して「Q」が TRUE になる計時誤差は、+1 タスク周期です。
  上記の範囲の内訳は、以下のとおりです。
  a）タスク周期ごとに「ET」の計時が「PT」に達したか判断しており、判断直後に「ET」の計時が「PT」に達した場合に+1 タスク周期遅れます。
- 「TimerDat」は入出力変数ですが、値を渡す必要はありません。構造体_sTimer 分のメモリ領域を確保して、本命令に渡してください。
- 「TimerDat」の内容は、変更しないでください。
- 運転開始時に、すでに「In」の値が TRUE の場合、その時点から計時を開始します。
- 「PT」の値を変更した場合、それが反映されるのは、つぎにタイマをリセットしたときです。計時中には反映されません。
- 本命令がマスタコントロール領域に存在し、マスタコントロールによるリセットが実行された場合、タイマがリセットされます。「ET」には、「PT」の値がセットされ、「Q」の値は FALSE になります。

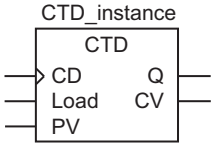
- JMP 系命令（JMP 命令など）の実行により、本命令が実行されなかった場合、「ET」の値は更新されません。しかし、その間も計時は継続しています。したがって、つぎに本命令が実行されたときに、「ET」は正しい値に更新されます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」および「Out」の値は FALSE になります。

カウンタ命令

命令	名称	ページ
CTD	減算カウンタ	P.2-146
CTD_**	減算カウンタグループ	P.2-148
CTU	加算カウンタ	P.2-151
CTU_**	加算カウンタグループ	P.2-153
CTUD	可逆カウンタ	P.2-156
CTUD_**	可逆カウンタグループ	P.2-161

CTD

カウンタ入力信号が入力されるごとに、減算していくカウンタです。プリセット値、カウンタ値のデータ型は、INT です。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTD	減算カウンタ	FB		CTD_instance(CD, Load, PV, Q, CV);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CD	カウンタ入力	入力	カウンタ入力	データ型に従う	–	FALSE
Load *1	ロード信号		TRUE:「CV」に「PV」をロード			
PV	プリセット値		カウンタのプリセット値	0～32767		0
Q	カウンタ出力	出力	TRUE: カウンタ出力 ON FALSE: カウンタ出力 OFF	データ型に従う	–	–
CV	カウンタ値		カウンタの現在値	0～32767		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Load」を「LD」と記述することができます。
たとえば、CTD_instance(CD:=A, LD:=abc, PV:=INT#5, Q=>def, CV=>ghi);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	○																			
Load	○																			
PV											○									
Q	○																			
CV											○									

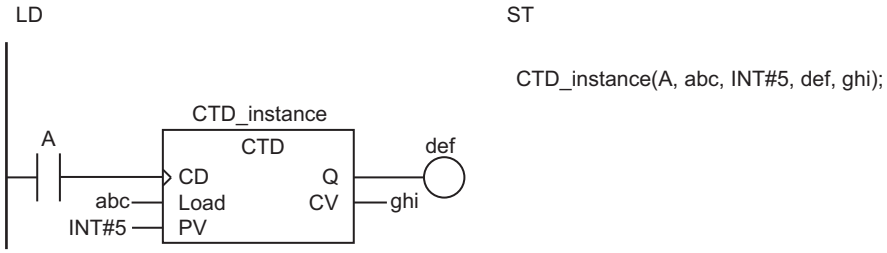
機能

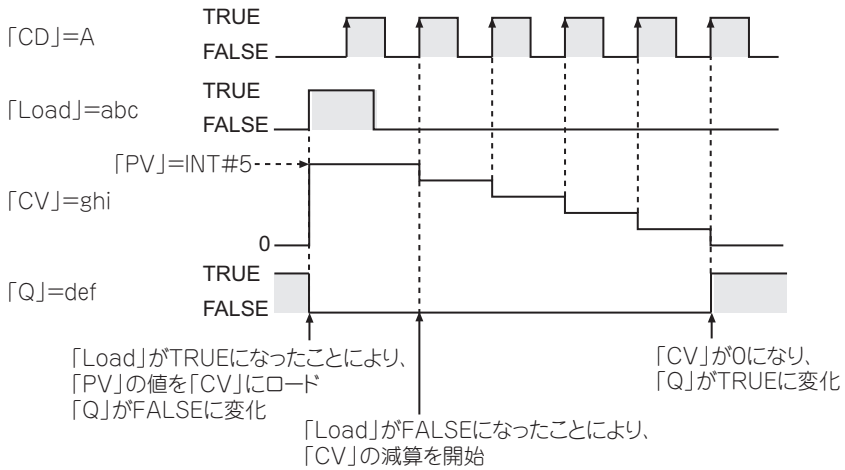
減算カウンタです。プリセット値、カウンタ値のデータ型は、INT です。

ロード信号「Load」が TRUE にセットされると、プリセット値「PV」の値をカウンタ値「CV」にロードし、カウンタ出力「Q」が FALSE になります。
カウンタ入力信号「CD」の立ち上がり時に、「CV」を減算していきます。「CV」の値が 0 以下に達したら、「Q」の値が TRUE に変わります。

「CV」の値が 0 以下になったら、それ以上「CD」が立ち上がっても「CV」は変化しません。
「Load」が TRUE の間は、「CD」を無視します。「CV」の減算は行いません。

「PV」=INT#5 の場合の、記述例とタイミングチャートを、以下に示します。





参考

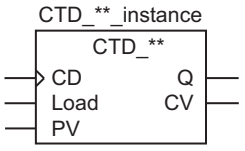
- カウンタ入力信号が入力されるごとに加算していくカウンタが必要な場合は、「CTU (P.2-151)」命令を使用してください。
- 加算と減算の両方ができるカウンタが必要な場合は、「CTUD (P.2-156)」命令を使用してください。

使用上の注意

- カウントダウン終了後にカウンタを再起動するには、「Load」の値をいったん TRUE にしてから FALSE にしてください。
- 「PV」に負数が設定された場合も、「Load」の値が TRUE になった時点で、「PV」の値を「CV」にロードします。「CV」の値が 0 以下なので、直ちに「Q」の値が TRUE になります。その後、「CD」が変化しても「CV」の減算は行いません。
- 「CD」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CD」の値が TRUE であれば、「CV」の減算が 1 回行われます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

CTD_**

カウンタ入力信号が入力されるごとに、減算していくカウンタです。プリセット値、カウンタ値のデータ型は、DINT、LINT、UDINT、ULINT のいずれかです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTD_**	減算カウンタグループ	FB	 ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか	CTD_**_instance(CD, Load, PV, Q, CV); ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CD	カウンタ入力	入力	カウンタ入力	データ型に従う	–	FALSE
Load *1	ロード信号		TRUE:「CV」に「PV」をロード			
PV	プリセット値		カウンタのプリセット値	データ型に従う *2		0
Q	カウンタ出力	出力	TRUE: カウンタ出力 ON FALSE: カウンタ出力 OFF	データ型に従う	–	–
CV	カウンタ値		カウンタの現在値	データ型に従う *2		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Load」を「LD」と記述することができます。
たとえば、CTD_LINT_instance(CD:=A, LD:=abc, PV:=LINT#5, Q=>def, CV=>ghi);と記述できます。
*2. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	○																			
Load	○																			
PV								○	○			○	○							
Q	○																			
CV	「PV」と同じデータ型																			

機能

減算カウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。
命令名称は、「PV」と「CV」のデータ型によって、異なります。たとえば、両者が LINT 型の場合、命令名称は CTD_LINT となります。

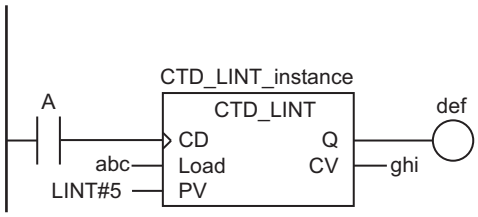
ロード信号「Load」が TRUE にセットされると、プリセット値「PV」の値をカウンタ値「CV」にロードし、カウンタ出力「Q」が FALSE になります。
カウンタ入力信号「CD」の立ち上がり時に、「CV」を減算していきます。「CV」の値が 0 以下に達したら、「Q」の値が TRUE に変わります。

「CV」の値が 0 以下になったら、それ以上「CD」が立ち上がっても「CV」は変化しません。

「Load」が TRUE の間は、「CD」を無視します。「CV」の減算は行いません。

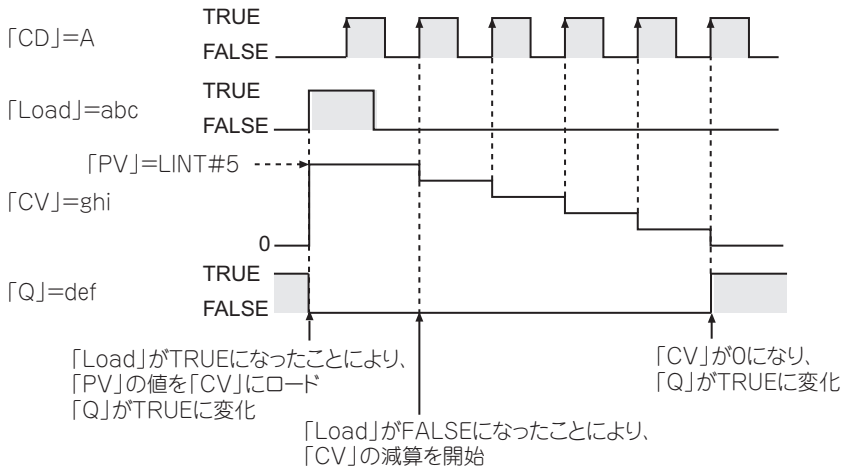
CTD_LINT 命令で、「PV」=LINT#5 の場合の、記述例とタイミングチャートを、以下に示します。

LD



ST

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



参考

- カウンタ入力信号が入力されるごとに加算していくカウンタが必要な場合は、「CTU (P.2-151)」命令を使用してください。
- 加算と減算の両方ができるカウンタが必要な場合は、「CTUD (P.2-156)」命令を使用してください。

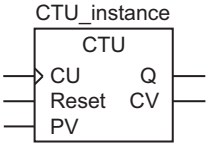
使用上の注意

- カウントダウン終了後にカウンタを再起動するには、「Load」の値をいったん TRUE にしてから FALSE にしてください。
- 「PV」と「CV」のデータ型は、同じにしてください。
- 「PV」に負数が設定された場合も、「Load」の値が TRUE になった時点で、「PV」の値を「CV」にロードします。「CV」の値が 0 以下なので、直ちに「Q」の値が TRUE になります。その後、「CD」が変化しても「CV」の減算は行いません。
- 「CD」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CD」の値が TRUE であれば、「CV」の減算が 1 回行われます。

- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

CTU

カウンタ入力信号が入力されるごとに、加算していくカウンタです。プリセット値、カウンタ値のデータ型は、INT です。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTU	加算カウンタ	FB		CTU_instance(CU, Reset, PV, Q, CV);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CU	カウンタ入力	入力	カウンタ入力	データ型に従う	–	FALSE
Reset *1	リセット信号		TRUE: 「CV」を 0 にリセット			
PV	プリセット値		カウンタのプリセット値	0～32767		0
Q	カウンタ出力	出力	TRUE: カウンタ出力 ON FALSE: カウンタ出力 OFF	データ型に従う	–	–
CV	カウンタ値		カウンタの現在値	0～32767		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Reset」を「R」と記述することができます。
たとえば、CTU_instance(CU:=A, R:=abc, PV:=INT#5, Q=>def, CV=>ghi);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
Reset	○																			
PV											○									
Q	○																			
CV											○									

機能

加算カウンタです。プリセット値、カウンタ値のデータ型は、INT です。

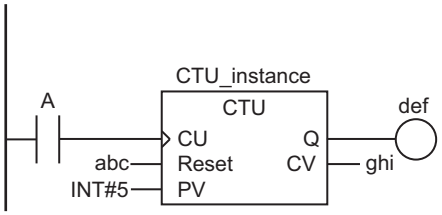
リセット信号「Reset」が TRUE にセットされると、カウンタ値「CV」の値が 0 になり、カウンタ出力「Q」が FALSE になります。
カウンタ入力信号「CU」の立ち上がり時に、「CV」を加算していきます。「CV」の値がプリセット値「PV」の値以上になったら、「Q」の値が TRUE に変わります。

「CV」の値が「PV」の値以上になったら、それ以上「CU」が入力されても「CV」は変化しません。

「Reset」が TRUE の間は、「CU」を無視します。「CV」の加算は行いません。

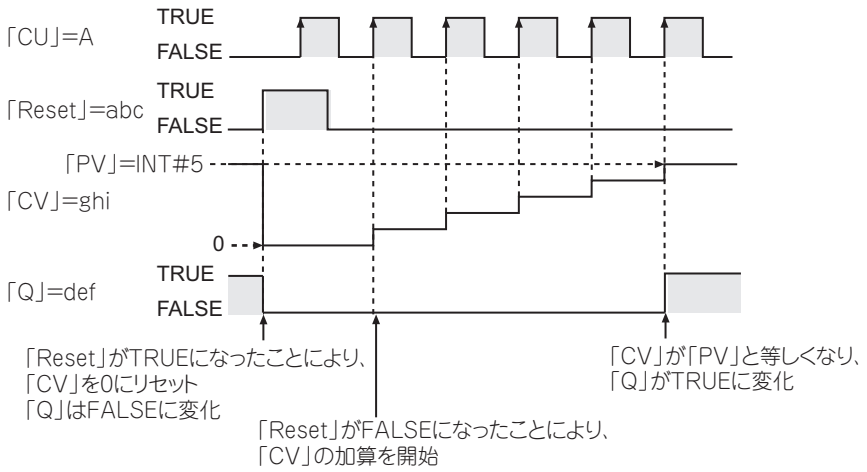
「PV」=INT#5 の場合の、記述例とタイミングチャートを、以下に示します。

LD



ST

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



参考

- カウンタ入力信号が入力されるごとに減算していくカウンタが必要な場合は、「CTD (P.2-146)」命令を使用してください。
- 加算と減算の両方ができるカウンタが必要な場合は、「CTUD (P.2-156)」命令を使用してください。

使用上の注意

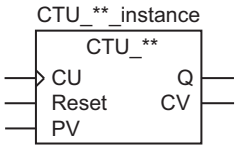
- カウントアップ終了後にカウンタを再起動するには、「Reset」の値をいったん TRUE にしてから FALSE にしてください。
- 「PV」に負数が設定された場合も、「Reset」の値が TRUE になった時点で、「CV」の値は 0 になります。「CV」の値が「PV」の値以上なので、直ちに「Q」の値が TRUE になります。その後、「CU」が変化しても「CV」の加算は行いません。
- 「Reset」の値が FALSE の状態で、「PV」の値が変更された場合、以下のように動作します。

「PV」の値	意味
その時点での「CV」より大	カウント動作を継続します。
その時点での「CV」以下	カウント動作を終了します。「Q」の値が TRUE になります。「CV」の値は、その時点の値のまま変化しません。

- 「CU」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CU」の値が TRUE であれば、「CV」の加算が 1 回行われます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

CTU_**

カウンタ入力信号が入力されるごとに、加算していくカウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTU_**	加算カウンタグループ	FB	 ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか	CTU_**__instance(CU, Reset, PV, Q, CV); ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CU	カウンタ入力	入力	カウンタ入力	データ型に従う	–	FALSE
Reset *1	リセット信号		TRUE:「CV」を 0 にリセット			
PV	プリセット値		カウンタのプリセット値	データ型に従う *2		0
Q	カウンタ出力	出力	TRUE: カウンタ出力 ON FALSE: カウンタ出力 OFF	データ型に従う	–	–
CV	カウンタ値		カウンタの現在値	データ型に従う *2		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Reset」を「R」と記述することができます。
たとえば、CTU_LINT_instance(CU:=A, R:=abc, PV:=LINT#5, Q=>def, CV=>ghi);と記述できます。
*2. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
Reset	○																			
PV								○	○			○	○							
Q	○																			
CV	「PV」と同じデータ型																			

機能

加算カウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。
命令名称は、「PV」と「CV」のデータ型によって、異なります。たとえば、両者が LINT 型の場合、命令名称は CTU_LINT となります。

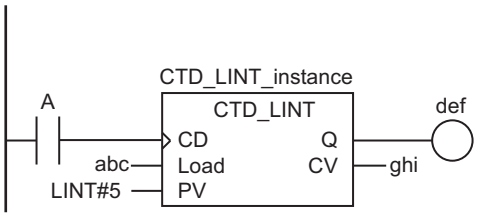
リセット信号「Reset」が TRUE にセットされると、カウンタ値「CV」の値が 0 になり、カウンタ出力「Q」が FALSE になります。
カウンタ入力信号「CU」の立ち上がり時に、「CV」を加算していきます。「CV」の値がプリセット値「PV」の値以上になったら、「Q」の値が TRUE に変わります。

「CV」の値が「PV」の値以上になったら、それ以上「CU」が入力されても「CV」は変化しません。

「Reset」が TRUE の間は、「CU」を無視します。「CV」の加算は行いません。

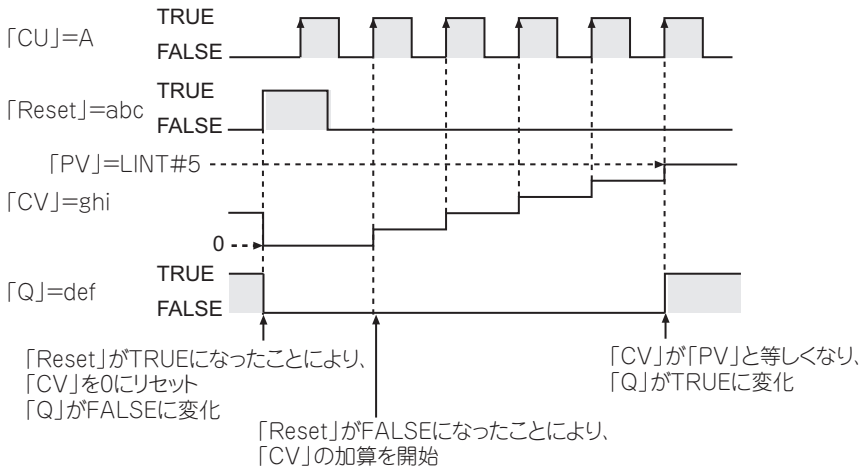
CTU_LINT 命令で、「PV」=LINT#5 の場合の、記述例とタイミングチャートを、以下に示します。

LD



ST

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



参考

- カウンタ入力信号が入力されるごとに減算していくカウンタが必要な場合は、「CTD (P.2-146)」命令を使用してください。
- 加算と減算の両方ができるカウンタが必要な場合は、「CTUD (P.2-156)」命令を使用してください。

使用上の注意

- カウントアップ終了後にカウンタを再起動するには、「Reset」の値をいったん TRUE にしてから FALSE にしてください。
- 「PV」に負数が設定された場合も、「Reset」の値が TRUE になった時点で、「CV」の値は 0 になります。「CV」の値が「PV」の値以上なので、直ちに「Q」の値が TRUE になります。その後、「CU」が変化しても「CV」の加算は行いません。
- 「PV」と「CV」のデータ型は、同じにしてください。
- 「Reset」の値が FALSE の状態で、「PV」の値が変更された場合、以下のように動作します。

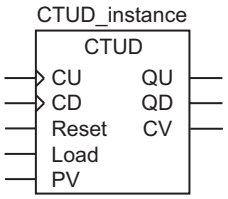
「PV」の値	意味
その時点での「CV」より大	カウント動作を継続します。

「PV」の値	意味
その時点での「CV」以下	カウント動作を終了します。「Q」の値が TRUE になります。「CV」の値は、その時点の値のまま変化しません。

- 「CU」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CU」の値が TRUE であれば、「CV」の加算が 1 回行われます。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Q」の値は FALSE になります。

CTUD

加算カウンタ入力および減算カウンタ入力に従い、加減算するカウンタです。プリセット値、カウンタ値のデータ型は、INT です。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTUD	可逆カウンタ	FB		CTUD_instance(CU, CD, Reset, Load, PV, QU, QD, CV);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CU	加算カウンタ入力	入力	加算カウンタ入力	データ型に従う	–	FALSE
CD	減算カウンタ入力		減算カウンタ入力			
Reset *1	リセット信号		TRUE:「CV」が 0 にリセット			
Load *1	ロード信号		TRUE:「CV」に「PV」がロードされる。			
PV	プリセット値		加算カウンタにとってのカウントアップ終了値 減算カウンタにとっての初期値	0～32767		0
QU	加算カウンタ出力	出力	TRUE: 加算カウンタ出力 ON FALSE: 加算カウンタ出力 OFF	データ型に従う	–	–
QD	減算カウンタ出力		TRUE: 減算カウンタ出力 ON FALSE: 減算カウンタ出力 OFF			
CV	カウンタ値		カウンタの現在値	0～32767		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Reset」を「R」、「Load」を「LD」と記述することができます。
たとえば、CTUD_instance(CU:=A, CD:=B, R:=abc, LD:=def, PV:=INT#3, QU=>ghi, QD=>jkl, CV=>mno);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
CD	○																			
Reset	○																			
Load	○																			
PV											○									

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
QU	○																			
QD	○																			
CV											○									

機能

加算カウンタ入力信号および減算カウンタ入力信号の入力に従い、加減算するカウンタです。
加算カウンタと減算カウンタの両方の機能を持っています。
プリセット値、カウンタ値のデータ型は、INT です。

加算カウンタとしての機能

リセット信号「Reset」が TRUE にセットされると、カウンタ値「CV」の値が 0 になり、加算カウンタ出力「QU」が FALSE になります。
加算カウンタ入力信号「CU」の立ち上がり時に、「CV」を加算していきます。「CV」の値がプリセット値「PV」の値以上になったら、「QU」の値が TRUE に変わります。
「CV」の値が「PV」の値以上になったら、それ以上「CU」が入力されても「CV」は変化しません。

減算カウンタとしての機能

ロード信号「Load」が TRUE にセットされると、カウンタ値「CV」にプリセット値「PV」の値がロードされ、減算カウンタ出力「QD」が FALSE になります。
減算カウンタ入力信号号「CD」の立ち上がり時に、「CV」を減算していきます。「CV」の値が 0 以下になったら、「QD」の値が TRUE に変わります。
「CV」の値が 0 以下になったら、それ以上「CD」が入力されても「CV」は変化しません。

加算カウンタ、減算カウンタ共通の機能

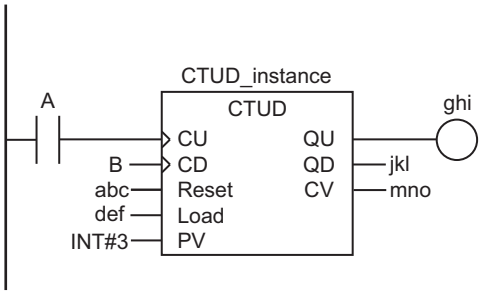
「Load」または「Reset」が TRUE の間は、「CU」および「CD」を無視します。「CV」の加減算は行いません。
「CU」と「CD」が同時に立ち上がった場合、「CV」は変化しません。
「Reset」と「Load」がともに TRUE の場合、「Reset」が優先され「CV」の値は 0 になります。
「Reset」が TRUE にセットされると、「CV」の値が 0 になるため、「QD」の値は TRUE になります。
「Load」が TRUE にセットされると、「CV」の値が「PV」と等しくなるため、「QU」の値は TRUE になります。

「Reset」,「Load」,「CV」,「QU」,「QD」の関係を以下に示します。ただし、「PV」の値は 0 より大きいとします。

Reset	Load	CV	QU	QD	動作
FALSE	FALSE	0 以下	FALSE	TRUE	加算カウンタとしてだけ、動作します。 • 「CV」は「CU」立ち上がりで加算されます。「CD」立ち上がりでは減算されません。
		0 と「PV」の間	FALSE	FALSE	加算カウンタ、減算カウンタとして動作します。 • 「CV」は「CU」立ち上がりで加算、「CD」立ち上がりで減算されます。
		「PV」以上	TRUE	FALSE	減算カウンタとしてだけ、動作します。 • 「CV」は「CD」立ち上がりで減算されます。「CU」立ち上がりでは加算されません。
TRUE	FALSE	0	FALSE	TRUE	加算カウンタのリセット状態です。 • 「CV」の値は、0 にセットされます。
FALSE	TRUE	「PV」	TRUE	FALSE	減算カウンタのリセット状態です。 • 「CV」の値は、「PV」にセットされます。
TRUE	TRUE	0	FALSE	TRUE	加算カウンタのリセット状態です。「Load」より「Reset」が優先されます。 • 「CV」の値は、0 にセットされます。

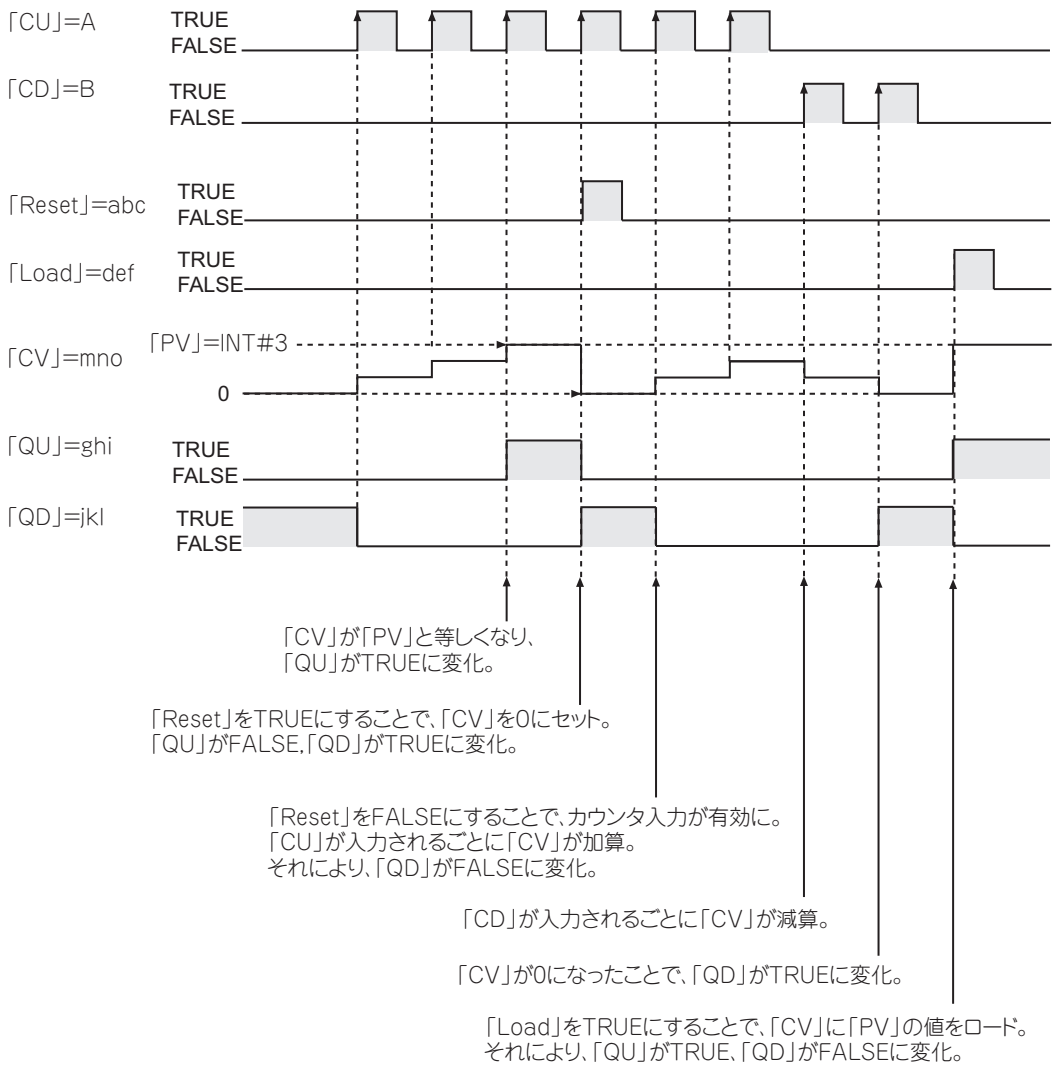
「PV」=INT#3 の場合の、記述例とタイミングチャートを、以下に示します。

LD



ST

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



参考

減算カウンタだけまたは加算カウンタだけで十分な場合は、「CTD (P.2-146)」命令、「CTU (P.2-151)」命令もあります。

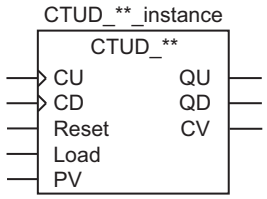
使用上の注意

- 加算カウンタをリセットするために「Reset」の値を TRUE にすると、「QU」の値が FALSE になると同時に、「QD」の値が TRUE になることに注意してください。
- 減算カウンタをリセットするために「Load」の値を TRUE にすると、「QD」の値が FALSE になると同時に、「QU」の値が TRUE になることに注意してください。
- 「PV」に負数が設定された場合も、「Load」の値が TRUE になった時点で、「PV」の値を「CV」にロードします。「CV」の値が 0 以下なので、直ちに「QD」の値が TRUE になります。その後、「CD」が変化しても「CV」の減算は行いません。また、「Reset」の値が TRUE になった時点で、「CV」の値は 0 になります。「CV」の値が「PV」の値以上なので、直ちに「QU」の値が TRUE になります。その後、「CU」が変化しても「CV」の加算は行いません。
- 本命令の実行中に、「PV」の値を変更することが可能です。その場合、変更後の「PV」の値が、その時点の「CV」未満ならば、直ちに「QU」の値が TRUE になります。

・「CU」または「CD」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CU」または「CD」の値が TRUE であれば、「CV」の加算または減算が 1 回行われます。

CTUD_**

加算カウンタ入力および減算カウンタ入力に従い、加減算するカウンタです。プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CTUD_**	可逆カウンタグループ	FB	 ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか	CTUD_**_instance(CU, CD, Reset, Load, PV, QU, QD, CV); ただし、“**”は DINT, LINT, UDINT, ULINT のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
CU	加算カウンタ入力	入力	加算カウンタ入力	データ型に従う	–	FALSE
CD	減算カウンタ入力		減算カウンタ入力			
Reset *1	リセット信号		TRUE:「CV」が 0 にリセット			
Load *1	ロード信号		TRUE:「CV」に「PV」がロードされる。			
PV	プリセット値		加算カウンタにとってのカウントアップ終了値 減算カウンタにとっての初期値	データ型に従う *2		0
QU	加算カウンタ出力	出力	TRUE: 加算カウンタ出力 ON FALSE: 加算カウンタ出力 OFF	データ型に従う	–	–
QD	減算カウンタ出力		TRUE: 減算カウンタ出力 ON FALSE: 減算カウンタ出力 OFF			
CV	カウンタ値		カウンタの現在値	データ型に従う *2		

*1. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Reset」を「R」、「Load」を「LD」と記述することができます。
たとえば、CTUD_LINT_instance(CU:=A, CD:=B, R:=abc, LD:=def, PV:=LINT#3, QU=>ghi, QD=>jkl,CV=>mno);と記述できます。
*2. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	○																			
Reset	○																			
Load	○																			
PV								○	○			○	○							
QU	○																			
QD	○																			
CV	「PV」と同じデータ型																			

機能

加算カウンタ入力信号および減算カウンタ入力信号の入力に従い、加減算するカウンタです。
加算カウンタと減算カウンタの両方の機能を持っています。
プリセット値、カウンタ値のデータ型は、DINT, LINT, UDINT, ULINT のいずれかです。
命令名称は、「PV」と「CV」のデータ型によって、異なります。たとえば、両者が LINT 型の場合、命令名称は CTUD_LINT となります。

加算カウンタとしての機能

リセット信号「Reset」が TRUE にセットされると、カウンタ値「CV」の値が 0 になり、加算カウンタ出力「QU」が FALSE になります。
加算カウンタ入力信号「CU」の立ち上がり時に、「CV」を加算していきます。「CV」の値がプリセット値「PV」の値以上になったら、「QU」の値が TRUE に変わります。
「CV」の値が「PV」の値以上になったら、それ以上「CU」が入力されても「CV」は変化しません。

減算カウンタとしての機能

ロード信号「Load」が TRUE にセットされると、カウンタ値「CV」にプリセット値「PV」の値がロードされ、減算カウンタ出力「QD」が FALSE になります。
減算カウンタ入力信号「CD」の立ち上がり時に、「CV」を減算していきます。「CV」の値が 0 以下になったら、「QD」の値が TRUE に変わります。
「CV」の値が 0 以下になったら、それ以上「CD」が入力されても「CV」は変化しません。

加算カウンタ、減算カウンタ共通の機能

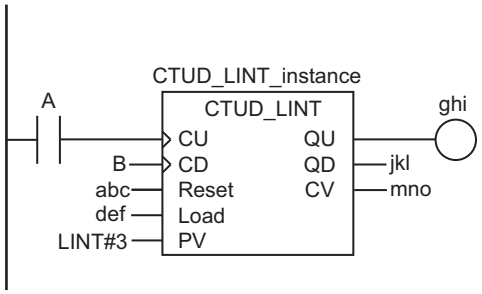
「Load」または「Reset」が TRUE の間は、「CU」および「CD」を無視します。「CV」の加減算は行いません。
「CU」と「CD」が同時に立ち上がった場合、「CV」は変化しません。
「Reset」と「Load」がともに TRUE の場合、「Reset」が優先され「CV」の値は 0 になります。
「Reset」が TRUE にセットされると、「CV」の値が 0 になるため、「QD」の値は TRUE になります。
「Load」が TRUE にセットされると、「CV」の値が「PV」と等しくなるため、「QU」の値は TRUE になります。

「Reset」,「Load」,「CV」,「QU」,「QD」の関係を以下に示します。ただし、「PV」の値は 0 より大きいとします。

Reset	Load	CV	QU	QD	動作
FALSE	FALSE	0 以下	FALSE	TRUE	加算カウンタとしてだけ、動作します。 •「CV」は「CU」立ち上がりで加算されます。「CD」立ち上がりでは減算されません。
		0 と「PV」の間	FALSE	FALSE	加算カウンタ、減算カウンタとして動作します。 •「CV」は「CU」立ち上がりで加算、「CD」立ち上がりで減算されます。
		「PV」以上	TRUE	FALSE	減算カウンタとしてだけ、動作します。 •「CV」は「CD」立ち上がりで減算されます。「CU」立ち上がりでは加算されません。
TRUE	FALSE	0	FALSE	TRUE	加算カウンタのリセット状態です。 •「CV」の値は、0 にセットされます。
FALSE	TRUE	「PV」	TRUE	FALSE	減算カウンタのリセット状態です。 •「CV」の値は、「PV」にセットされます。
TRUE	TRUE	0	FALSE	TRUE	加算カウンタのリセット状態です。「Load」より「Reset」が優先されます。 •「CV」の値は、0 にセットされます。

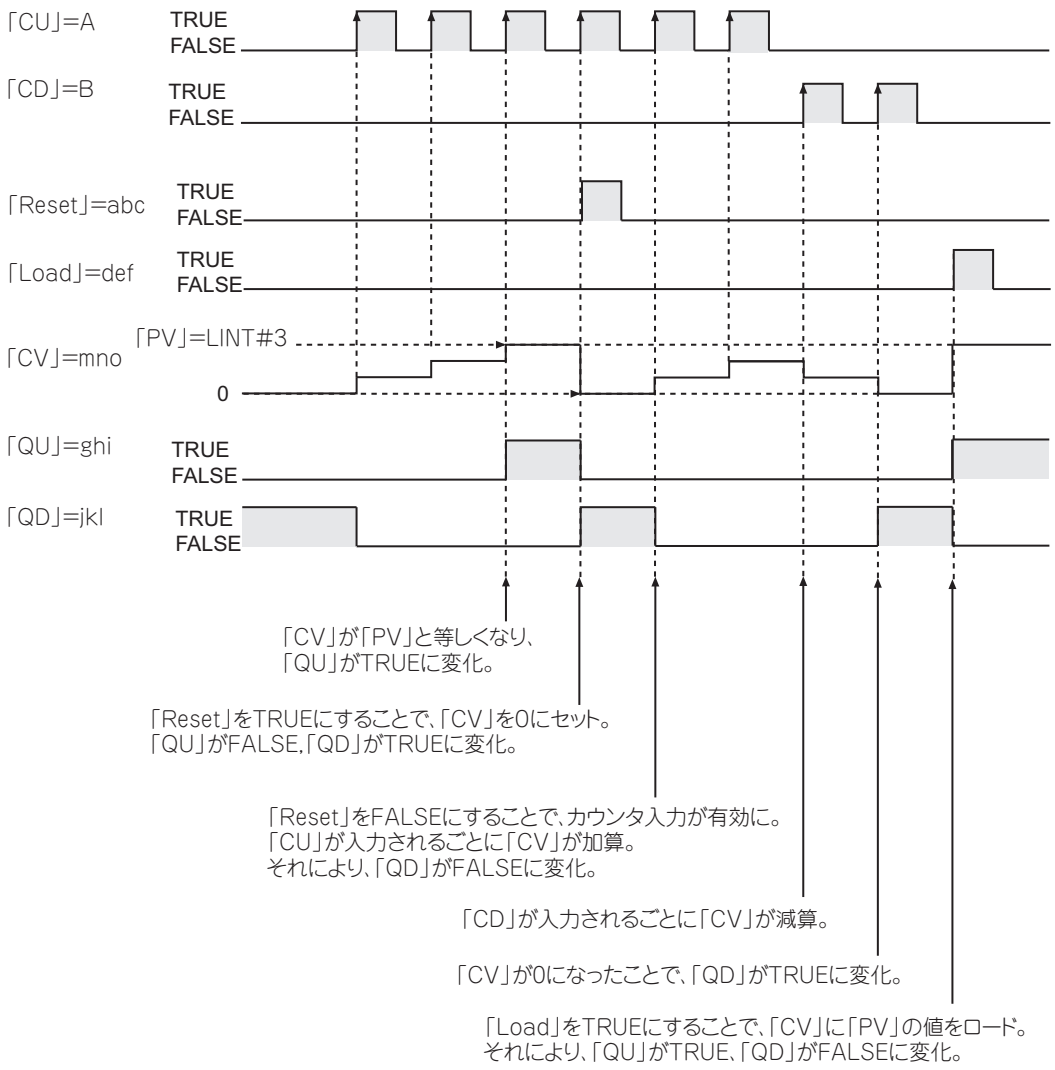
CTUD_LINT 命令で、「PV」=LINT#3 の場合の、記述例とタイミングチャートを、以下に示します。

LD



ST

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



参考

減算カウンタだけまたは加算カウンタだけで十分な場合は、「CTD_** (P.2-148)」命令、「CTU_** (P.2-153)」命令もあります。

使用上の注意

- 加算カウンタをリセットするために「Reset」の値を TRUE にすると、「QU」の値が FALSE になると同時に、「QD」の値が TRUE になることに注意してください。
- 減算カウンタをリセットするために「Load」の値を TRUE にすると、「QD」の値が FALSE になると同時に、「QU」の値が TRUE になることに注意してください。
- 「PV」に負数が設定された場合も、「Load」の値が TRUE になった時点で、「PV」の値を「CV」にロードします。「CV」の値が 0 以下なので、直ちに「QD」の値が TRUE になります。その後、「CD」が変化しても「CV」の減算は行いません。また、「Reset」の値が TRUE になった時点で、「CV」の値は 0 になります。「CV」の値が「PV」の値以上なので、直ちに「QU」の値が TRUE になります。その後、「CU」が変化しても「CV」の加算は行いません。
- 本命令の実行中に、「PV」の値を変更することが可能です。その場合、変更後の「PV」の値が、その時点の「CV」未満ならば、直ちに「QU」の値が TRUE になります。
- 「PV」と「CV」のデータ型は、同じにしてください。

・「CU」または「CD」の値が FALSE の状態で電源断発生または動作モードをプログラムにした場合、本命令実行再開時に「CU」または「CD」の値が TRUE であれば、「CV」の加算または減算が 1 回行われます。

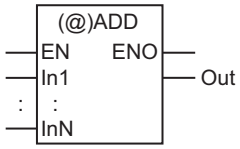
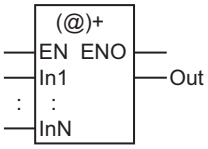
算術命令

命令	名称	ページ
ADD, +	加算	P.2-169
AddOU, +OU	加算（オーバーフローチェック付き）	P.2-173
SUB, -	減算	P.2-176
SubOU, -OU	減算（オーバーフローチェック付き）	P.2-179
MUL, *	乗算	P.2-183
MulOU, *OU	乗算（オーバーフローチェック付き）	P.2-186
DIV, /	除算	P.2-190
MOD	剰余	P.2-193
ABS	絶対値	P.2-195
RadToDeg / DegToRad	ラジアン→角度変換／角度→ラジアン変換	P.2-197
SIN / COS / TAN	SIN 演算／COS 演算／TAN 演算	P.2-199
ASIN / ACOS / ATAN	SIN-1 演算／COS-1 演算／TAN-1 演算	P.2-202
SQRT	平方根演算	P.2-205
LN / LOG	自然対数演算／常用対数演算	P.2-207
EXP	自然指数演算	P.2-211
EXPT, **	べき乗演算	P.2-213
Inc / Dec	インクリメント／デクリメント	P.2-217
Rand	乱数発生	P.2-219
AryAdd	配列一括加算	P.2-221
AryAddV	配列要素加算	P.2-223
ArySub	配列一括減算	P.2-225
ArySubV	配列要素減算	P.2-227
AryMean	配列要素の平均値算出	P.2-229

命令	名称	ページ
ArySD	配列要素の標準偏差	P.2-231
ModReal	実数剰余	P.2-233
Fraction	実数小数部抽出	P.2-235
CheckReal	実数チェック	P.2-237

ADD, +

整数、実数を加算します。または、文字列を結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD, +	加算	FUN	 	Out:=In1 + In2;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	加算値	入力	加算する数 ラダー図: N は 2～5 ST: N は 2 *1	データ型に従う	−	0 *2
Out	出力値	出力	出力値	データ型に従う	−	−

*1. result := val1 + val2 + val3; のように、1 つの式に本命令を演算子として複数記述することが可能です。
1 つの式に記述できる演算子の数は、最大 64 個です。

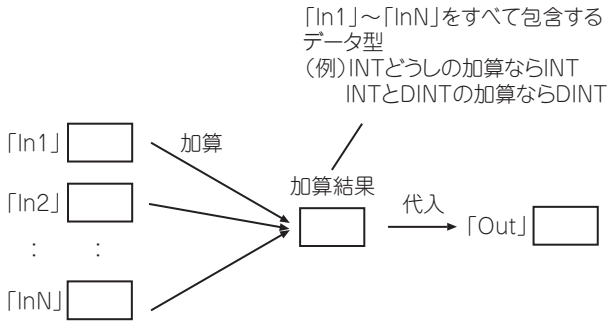
*2. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN						○	○	○	○	○	○	○	○	○	○					○
Out						○	○	○	○	○	○	○	○	○	○					○

機能

ラダー図の場合、2 個以上 5 個以下の整数または実数の加算結果を、出力値「Out」に出力します。
ST の場合、2 個の整数または実数の加算結果を、出力値「Out」に出力します。

加算値「In1」～「InN」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、加算結果は DINT 型になります。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。



オーバーフロー発生時の処理

「In1」～「InN」の和が、加算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。オーバーフローが発生した場合の、「In1」～「InN」のデータ型、加算結果のデータ型、加算結果の値の関係は、以下のとおりです。

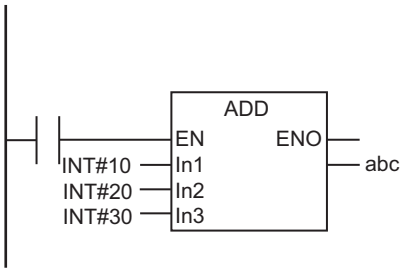
「In1」～「InN」のデータ型	加算結果のデータ型	加算結果の値
すべて整数型	整数型	「In1」～「InN」の和のうち、加算結果のデータ型のビット数で表現できる値 *1
いずれかが実数型	実数型	±∞ *2

*1. たとえば、「In1」の値が INT#32767、「In2」の値が INT#3 の場合、加算結果のデータ型は INT 型になります。加算結果の値は、両者の和 32770 の下位 16 ビットで INT#-32766 になります。
*2. 「In1」～「InN」の和が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=INT#10、「In2」=INT#20、「In3」=INT#30 の場合の例を、以下に示します。INT 型変数「abc」の値は、INT#60 になります。

LD



ST

```
abc:=INT#10+INT#20+INT#30;
```

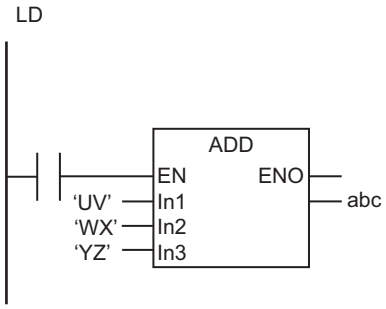
「In1」～「InN」を加算します。
10+20+30=60なので、abcの値はINT#60になります。



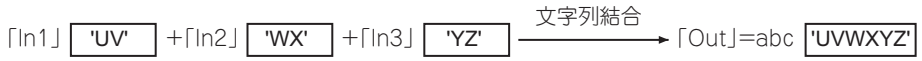
文字列の結合

「In1」～「InN」が STRING 型の場合は、それらの文字列を結合します。
ただし、「In1」～「InN」に STRING 型を指定できるのは、ラダー図プログラムの場合だけです。

「In1」='UV'、「In2」='WX'、「In3」='YZ'の場合の例を、以下に示します。STRING 型変数「abc」の値は、'UVWXYZ'になります。



「In1」～「InN」を文字列結合します。
'UV'+'WX'+'YZ'='UVWXYZ'なので、abcの値は'UVWXYZ'になります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とでは、仕様に違いがあります。仕様の違いを以下に示します。なお、ラダー図プログラムの ADD 命令と、ラダー図プログラムの+命令は、まったく同じ仕様です。

仕様項目	ラダー図	ST
加算値の最大個数	5	2 *1
加算値の入力パラメータの省略可否	「InN」に接続する入力パラメータ以外は省略可能	すべての入力パラメータが省略不可
変数「EN」、「ENO」の存在	あり	なし
加算値のデータ型がすべて整数の場合の、演算処理のビット数	8, 16, 32, 64 *2	32, 64 *3
文字列の結合の可否	可能	不可
異常の発生の有無	文字列の結合で、結合結果のサイズが1986 バイトを超えた場合に異常発生	なし

*1. result := val1 + val2 + val3; のように、1 つの式に本命令を演算子として複数記述することが可能です。1 つの式に記述できる演算子は、最大 64 個です。
*2. 演算ビット数は、加算値のうち、データ型のサイズが最大のものに合わせられます。たとえば、SINT 型、INT 型、DINT 型の加算をする場合は、DINT 型のサイズに合わせて、32 ビットで演算されます。
*3. 加算値の中に LINT 型および ULINT 型がなければ、32 ビットで演算されます。たとえば、SINT 型どうしの加算でも、32 ビットで演算されます。加算値の中に LINT 型または ULINT 型がある場合は、64 ビットで演算されます。

参考

- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。
- ST プログラムで文字列結合を行う場合は、「CONCAT (P.2-554)」命令を使用してください。

使用上の注意

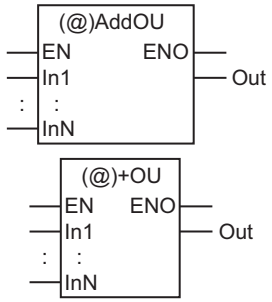
- 「Out」は、加算結果と異なるデータ型でもかまいません。ただし、加算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 文字列の結合の場合、「In1」～「InN」も「Out」も、すべて STRING 型にしてください。
- 加算によってオーバーフローやアンダーフローが発生しても、異常にはなりません。
- 加算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する加算結果の値は、つぎのようになります。

加算内容	加算結果の値
+∞と数値の加算	+∞
-∞と数値の加算	-∞
+∞と+∞の加算	+∞
-∞と-∞の加算	-∞
+∞と-∞の加算	非数

- 「In1」～「InN」のいずれかの値が非数の場合、加算結果の値は非数になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）文字列の結合で、結合した文字列のサイズが 1986 バイトを超える場合。

AddOU, +OU

整数、実数を加算します。さらに、整数の加算結果に対して、オーバーフローのチェックを行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AddOU, +OU	加算（オーバーフローチェック付き）	FUN		Out:=AddOU(In1, …, InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	加算値	入力	加算する数 N は 2～5	データ型に従う	−	0 *1
Out	出力値	出力	出力値	データ型に従う	−	−

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN						○	○	○	○	○	○	○	○	○ *1	○ *1					
Out						○	○	○	○	○	○	○	○	○	○					

*1. 「In1」～「InN」のいずれかのデータ型が実数型の場合、オーバーフローのチェックは行いません。

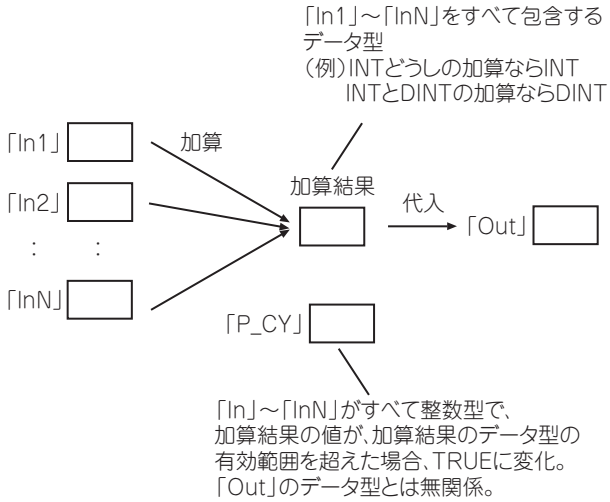
機能

2 個以上 5 個以下の整数または実数の加算結果を、出力値「Out」に出力します。

加算値「In1」～「InN」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、加算結果は DINT 型になります。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

オーバーフロー発生時の処理

「In1」～「InN」の和が、加算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。
「In1」～「InN」がすべて整数型の場合、オーバーフローが発生するとシステム定義変数「P_CY」（キャリーフラグ）の値が TRUE になります。
「In1」～「InN」のいずれかが実数型の場合、オーバーフローのチェックは行いません。したがって、「P_CY」の値は変化しません。



オーバーフローが発生した場合の、「In1」～「InN」のデータ型、加算結果のデータ型、加算結果の値、「P_CY」の値の関係は、以下のとおりです。

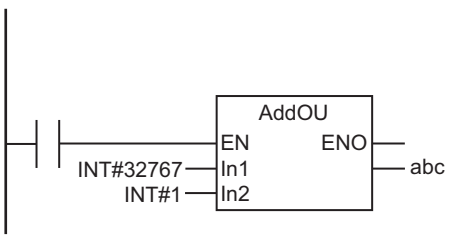
「In1」～「InN」のデータ型	加算結果のデータ型	加算結果の値	「P_CY」の値
すべて整数型	整数型	「In1」～「InN」の和のうち、加算結果のデータ型のビット数で表現できる値 *1	TRUE
いずれかが実数型	実数型	±∞ *2	変化しない

*1. たとえば、「In1」の値が INT#32767、「In2」の値が INT#3 の場合、加算結果のデータ型は INT 型になります。加算結果の値は、両者の和 32770 の下位 16 ビットで INT#-32766 になります。
*2. 「In1」～「InN」の和が正数の場合は+∞、負数の場合は-∞になります。

記述例

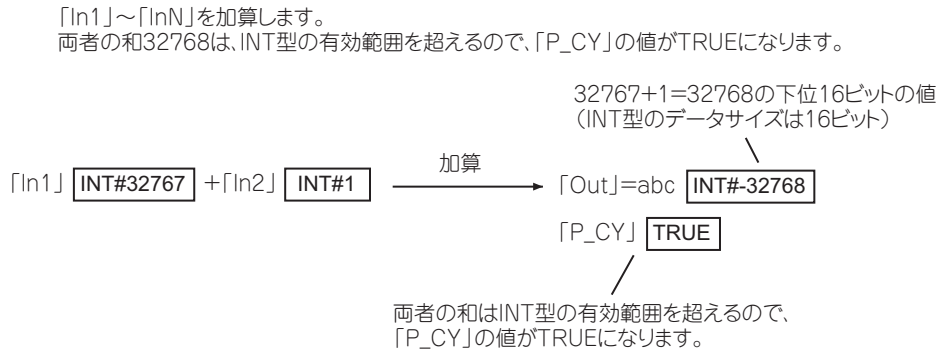
「In1」=INT#32767、「In2」=INT#1 で、変数「abc」が INT 型の場合の例を、以下に示します。
「In1」と「In2」はいずれも INT 型なので、加算結果のデータ型は INT 型になります。両者の和 32768 は INT 型の有効範囲を超えるので、「P_CY」の値が TRUE になります。また、INT 型変数「abc」の値は INT#-32768 になります（32768 の下位 16 ビットの値）。

LD



ST

```
abc:=AddOU(INT#32767, INT#1);
```



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とで、仕様に違いはありません。また、ラダー図プログラムの AddOU 命令と、ラダー図プログラムの+OU 命令にも、仕様に違いはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	TRUE: 整数演算でオーバーフローあり FALSE: 整数演算でオーバーフローなし

参考

- 「Out」が実数型の場合、値が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。
- オーバーフローのチェックが必要ない場合は、「ADD, + (P.2-169)」命令を使用してください。その方が処理時間が短くなります。

使用上の注意

- 「Out」は、加算結果と異なるデータ型でもかまいません。ただし、加算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 加算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する加算結果の値は、つぎのようになります。

加算内容	加算結果の値
+∞と数値の加算	+∞
-∞と数値の加算	-∞
+∞と+∞の加算	+∞
-∞と-∞の加算	-∞
+∞と-∞の加算	非数

- 「In1」～「InN」のいずれかの値が非数の場合、加算結果の値は非数になります。

SUB, -

整数、実数を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB,-	減算	FUN	(@)SUB EN ENO In1 Out In2 (@)- EN ENO In1 Out In2	Out:=In1 - In2;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算値	入力	被減算値	データ型に従う	−	0 *1
In2	減算値		減算値			
Out	出力値	出力	出力値	データ型に従う	−	−

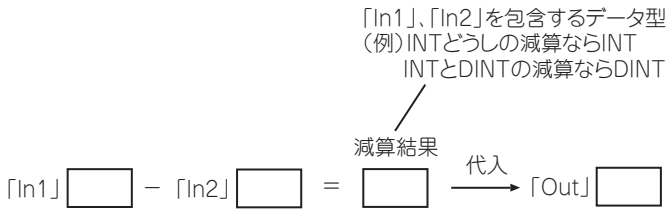
*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						○	○	○	○	○	○	○	○	○	○					
In2						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

被減算値「In1」から減算値「In2」を減算した結果を、出力値「Out」に出力します。

「In1」と「In2」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、減算結果は DINT 型になります。



データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

オーバーフロー発生時の処理

「In1」と「In2」の差が、減算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。オーバーフローが発生した場合の、「In1」、「In2」のデータ型、減算結果のデータ型、減算結果の値の関係は、以下のとおりです。

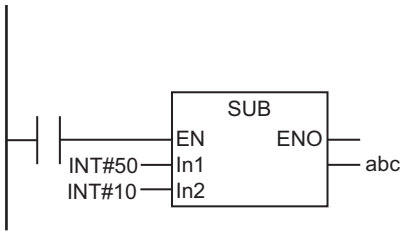
「In1」、「In2」のデータ型	減算結果のデータ型	減算結果の値
すべて整数型	整数型	「In1」と「In2」の差のうち、減算結果のデータ型のビット数で表現できる値 *1
いずれかが実数型	実数型	±∞ *2

*1. たとえば、「In1」の値が INT#32767、「In2」の値が INT#-3 の場合、減算結果のデータ型は INT 型になります。減算結果の値は、両者の差 32770 の下位 16 ビットで INT#-32766 になります。
*2. 「In1」と「In2」の差が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=INT#50、「In2」=INT#10 の場合の例を、以下に示します。INT 型変数「abc」の値は、INT#40 になります。

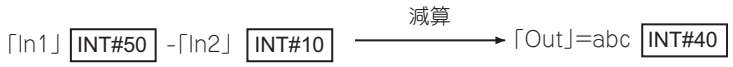
LD



ST

```
abc:=INT#50-INT#10;
```

「In1」から「In2」を減算します。
50-10=40なので、abcの値はINT#40になります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とでは、仕様に違いがあります。仕様の違いを以下に示します。なお、ラダー図プログラムの SUB 命令と、ラダー図プログラムのー命令は、まったく同じ仕様です。

仕様項目	ラダー図	ST
変数「EN」、「ENO」の存在	あり	なし
被減算値と減算値のデータ型が整数の場合の、演算処理のビット数	8, 16, 32, 64 *1	32, 64 *2

*1. 演算ビット数は、被減算値と減算値のうち、データ型のサイズが最大のものに合わせられます。たとえば、SINT 型と DINT 型の減算をする場合は、DINT 型のサイズに合わせて、32 ビットで演算されます。
*2. 被減算値と減算値の中に LINT 型および ULINT 型がなければ、32 ビットで演算されます。たとえば、SINT 型どうしの減算でも、32 ビットで演算されます。被減算値と減算値の中に LINT 型または ULINT 型がある場合は、64 ビットで演算されます。

参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

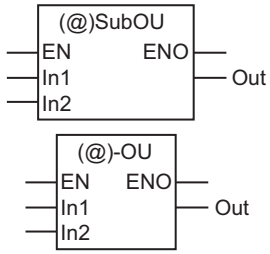
- 「Out」は、減算結果と異なるデータ型でもかまいません。ただし、減算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 減算によってオーバーフローやアンダーフローが発生しても、異常にはなりません。
- 減算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する減算結果は、つぎのようになります。

減算内容	減算結果
（+∞） -（数値）	+∞
（数値） -（+∞）	-∞
（-∞） -（数値）	-∞
（数値） -（-∞）	+∞
（+∞） -（+∞）	非数
（+∞） -（-∞）	+∞
（-∞） -（+∞）	-∞
（-∞） -（-∞）	非数

- 「In1」、「In2」のいずれかの値が非数の場合、減算結果の値は非数になります。

SubOU, -OU

整数、実数を減算します。さらに、整数の減算結果に対して、オーバーフローのチェックを行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SubOU, -OU	減算（オーバーフローチェック付き）	FUN		Out:=SubOU(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算値	入力	被減算値	データ型に従う	−	0 *1
In2	減算値		減算値			
Out	出力値	出力	出力値	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						○	○	○	○	○	○	○	○	○ *1	○ *1					
In2						○	○	○	○	○	○	○	○	○ *1	○ *1					
Out						○	○	○	○	○	○	○	○	○	○					

*1. 「In1」、「In2」のいずれかのデータ型が実数型の場合、オーバーフローのチェックは行いません。

機能

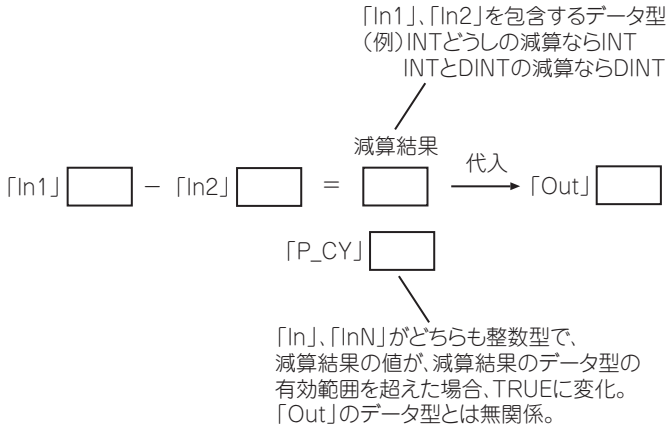
被減算値「In1」から減算値「In2」を減算した結果を、出力値「Out」に出力します。

「In1」と「In2」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、減算結果は DINT 型になります。

データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

オーバーフロー発生時の処理

「In1」と「In2」の差が、減算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。「In1」と「In2」がどちらも整数型の場合、オーバーフローが発生するとシステム定義変数「P_CY」（キャリーフラグ）の値が TRUE になります。
「In1」、「In2」のいずれかが実数型の場合、オーバーフローのチェックは行いません。したがって、「P_CY」の値は変化しません。



オーバーフローが発生した場合の、「In1」、「In2」のデータ型、減算結果のデータ型、減算結果の値、「P_CY」の値の関係は、以下のとおりです。

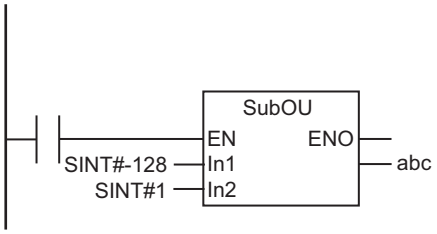
「In1」、「In2」のデータ型	減算結果のデータ型	減算結果の値	「P_CY」の値
すべて整数型	整数型	「In1」と「In2」の差のうち、減算結果のデータ型のビット数で表現できる値 *1	TRUE
いずれかが実数型	実数型	±∞ *2	変化しない

*1. たとえば、「In1」の値が INT#32767、「In2」の値が INT#-3 の場合、減算結果のデータ型は INT 型になります。減算結果の値は、両者の差 32770 の下位 16 ビットで INT#-32766 になります。
*2. 「In1」と「In2」の差が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=SINT#-128、「In2」=SINT#1 で、変数「abc」が SINT 型の場合の例を、以下に示します。「In1」と「In2」はいずれも SINT 型なので、減算結果のデータ型は SINT 型になります。両者の差-129 は、SINT 型の有効範囲を超えるので、「P_CY」の値が TRUE になります。また、SINT 型変数「abc」の値は SINT#127 になります（-129 の下位 8 ビットの値）。

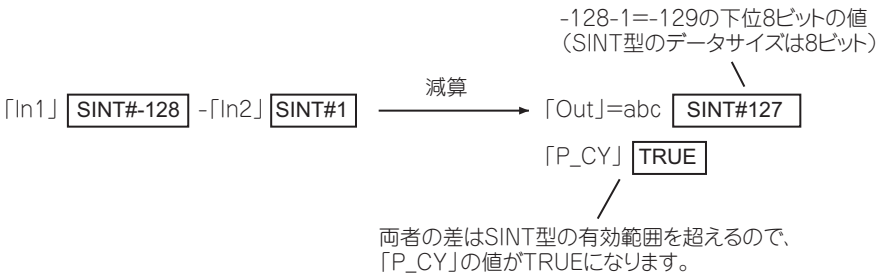
LD



ST

```
abc:=SubOU(SINT#-128, SINT#1);
```

「In1」から「In2」を減算します。
両者の差-129は、SINT型の有効範囲を超えるので、「P_CY」の値がTRUEになります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とで、仕様に違いはありません。また、ラダー図プログラムの SubOU 命令と、ラダー図プログラムの-OU 命令にも、仕様に違いはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	TRUE: 整数演算でオーバーフローあり FALSE: 整数演算でオーバーフローなし

参考

- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。
- オーバーフローのチェックが必要ない場合は、「SUB, - (P.2-176)」命令を使用してください。その方が処理時間が短くなります。

使用上の注意

- 「Out」は、減算結果と異なるデータ型でもかまいません。ただし、減算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 減算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する減算結果は、つぎのようになります。

減算内容	減算結果
（+∞） - （数値）	+∞
（数値） -（+∞）	-∞
（-∞） - （数値）	-∞
（数値） -（-∞）	+∞
（+∞） -（+∞）	非数
（+∞） -（-∞）	+∞
（-∞） -（+∞）	-∞

減算内容	減算結果
(-∞) - (-∞)	非数

・「In1」、「In2」のいずれかの値が非数の場合、減算結果の値は非数になります。

MUL, *

整数、実数を乗算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MUL,*	乗算	FUN	(@)MUL EN ENO In1 Out : : InN (@)* EN ENO In1 Out : : InN	Out:=In1 * In2;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	乗算値	入力	乗算する数 ラダー図: N は 2～5 ST: N は 2 *1	データ型に従う	−	1 *2
Out	出力値	出力	出力値	データ型に従う	−	−

*1. result := val1 * val2 * val3; のように、1 つの式に本命令を演算子として複数記述することが可能です。
1 つの式に記述できる演算子の数は、最大 64 個です。

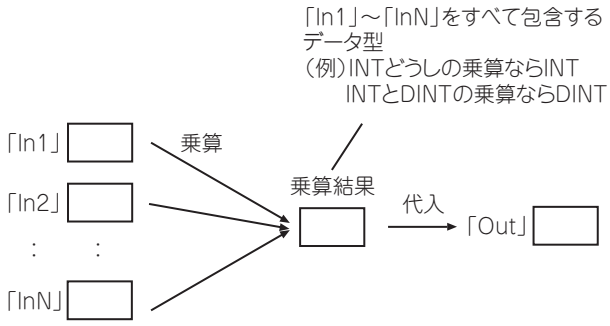
*2. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

ラダー図の場合、2 個以上 5 個以下の整数または実数の乗算結果を、出力値「Out」に出力します。
ST の場合、2 個の整数または実数の乗算結果を、出力値「Out」に出力します。

乗算値「In1」～「InN」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、乗算結果は DINT 型になります。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。



オーバーフロー発生時の処理

「In1」～「InN」の積が、乗算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。オーバーフローが発生した場合の、「In1」～「InN」のデータ型、乗算結果のデータ型、乗算結果の値の関係は、以下のとおりです。

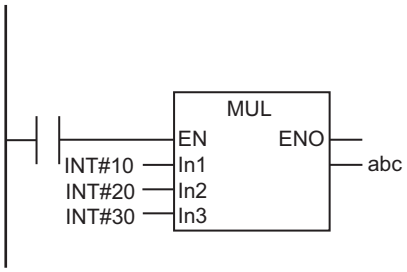
「In1」～「InN」のデータ型	乗算結果のデータ型	乗算結果の値
すべて整数型	整数型	「In1」～「InN」の積のうち、乗算結果のデータ型のビット数で表現できる値 *1
いずれかが実数型	実数型	±∞ *2

*1. たとえば、「In1」の値が INT#16384、「In2」の値が INT#2 の場合、乗算結果のデータ型は INT 型になります。乗算結果の値は、両者の積 32768 の下位 16 ビットで INT#-32768 になります。
*2. 「In1」～「InN」の積が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=INT#10、「In2」=INT#20、「In3」=INT#30 の場合の例を、以下に示します。INT 型変数「abc」の値は、INT#6000 になります。

LD



ST

```
abc:=INT#10*INT#20*INT#30;
```

「In1」～「InN」を乗算します。
10*20*30=6000なので、abcの値はINT#6000になります。

「In1」 INT#10 *「In2」 INT#20 *「In3」 INT#30 ——乗算——→ 「Out」=abc INT#6000

ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とでは、仕様に違いがあります。仕様の違いを以下に示します。なお、ラダー図プログラムの MUL 命令と、ラダー図プログラムの＊命令は、まったく同じ仕様です。

仕様項目	ラダー図	ST
乗算値の最大個数	5	2 *1
乗算値の入力パラメータの省略可否	InN に接続する入力パラメータ以外は省略可能	すべての入力パラメータが省略不可
変数「EN」、「ENO」の存在	あり	なし
乗算値のデータ型がすべて整数の場合の、演算処理のビット数	8, 16, 32, 64 *2	32, 64 *3

*1. result := val1 * val2 * val3; のように、1 つの式に本命令を演算子として複数記述することが可能です。1 つの式に記述できる演算子は、最大 64 個です。
*2. 演算ビット数は、乗算値のうち、データ型のサイズが最大のものに合わせられます。たとえば、SINT 型、INT 型、DINT 型の乗算をする場合は、DINT 型のサイズに合わせて、32 ビットで演算されます。
*3. 乗算値の中に LINT 型および ULINT 型がなければ、32 ビットで演算されます。たとえば、SINT 型どうしの乗算でも、32 ビットで演算されます。乗算値の中に LINT 型または ULINT 型がある場合は、64 ビットで演算されます。

参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

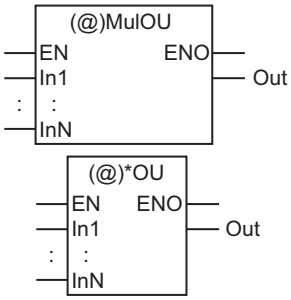
- 「Out」は、乗算結果と異なるデータ型でもかまいません。ただし、乗算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 乗算によってオーバーフローやアンダーフローが発生しても、異常にはなりません。
- 乗算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する乗算結果は、つぎのようになります。

乗算内容	乗算結果
+∞と正数の乗算	+∞
+∞と負数の乗算	-∞
-∞と正数の乗算	-∞
-∞と負数の乗算	+∞
+∞と+∞の乗算	+∞
-∞と-∞の乗算	+∞
+∞と-∞の乗算	-∞
+∞と 0 の乗算	非数
-∞と 0 の乗算	非数

- 「In1」～「InN」のいずれかの値が非数の場合、乗算結果の値は非数になります。

MulOU, *OU

整数、実数を乗算した結果を出力します。さらに、整数の乗算結果に対して、オーバーフローのチェックを行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MulOU,*OU	乗算（オーバーフローチェック付き）	FUN		Out:=MulOU(In1, …, InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	乗算値	入力	乗算する数 N は 2～5	データ型に従う	－	1 *1
Out	出力値	出力	出力値	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN						○	○	○	○	○	○	○	○	○ *1	○ *1					
Out						○	○	○	○	○	○	○	○	○	○					

*1. 「In1」～「InN」のいずれかのデータ型が実数型の場合、オーバーフローのチェックは行いません。

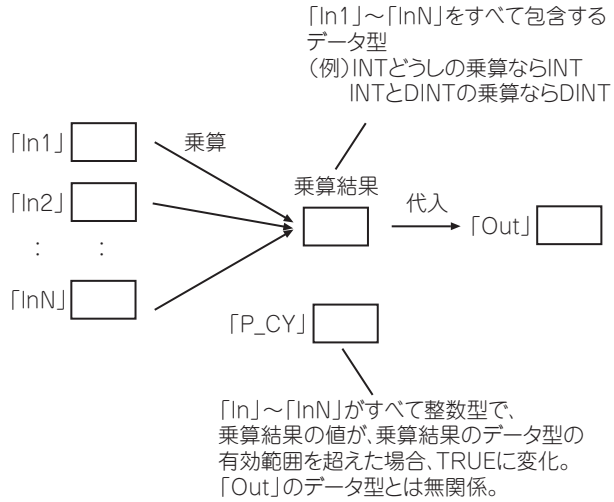
機能

2 個以上 5 個以下の整数または実数の乗算結果を、出力値「Out」に出力します。

乗算値「In1」～「InN」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、乗算結果は DINT 型になります。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

オーバーフロー発生時の処理

「In1」～「InN」の積が、乗算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。
「In1」～「InN」がすべて整数型の場合、オーバーフローが発生するとシステム定義変数「P_CY」（キャリーフラグ）の値が TRUE になります。
「In1」～「InN」のいずれかが実数型の場合、オーバーフローのチェックは行いません。したがって、「P_CY」の値は変化しません。



オーバーフローが発生した場合の、「In1」～「InN」のデータ型、乗算結果のデータ型、乗算結果の値、「P_CY」の値の関係は、以下のとおりです。

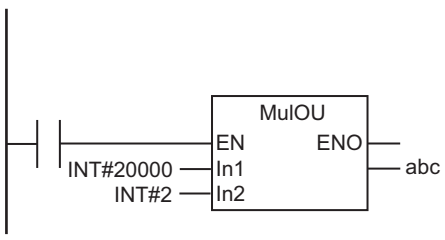
「In1」～「InN」のデータ型	乗算結果のデータ型	乗算結果の値	「P_CY」の値
すべて整数型	整数型	「In1」～「InN」の積のうち、乗算結果のデータ型のビット数で表現できる値 *1	TRUE
いずれかが実数型	実数型	±∞ *2	変化しない

*1. たとえば、「In1」の値が INT#16384、「In2」の値が INT#2 の場合、乗算結果のデータ型は INT 型になります。乗算結果の値は、両者の積 32768 の下位 16 ビットで INT#-32768 になります。
*2. 「In1」～「InN」の積が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=INT#20000、「In2」=INT#2 で、変数「abc」が INT 型の場合の例を、以下に示します。
「In1」と「In2」はいずれも INT 型なので、乗算結果のデータ型は INT 型になります。両者の積 40000 は、INT 型の有効範囲を超えるので、「P_CY」の値が TRUE になります。また、INT 型変数「abc」の値は INT#-25536 になります（40000 の下位 16 ビットの値）。

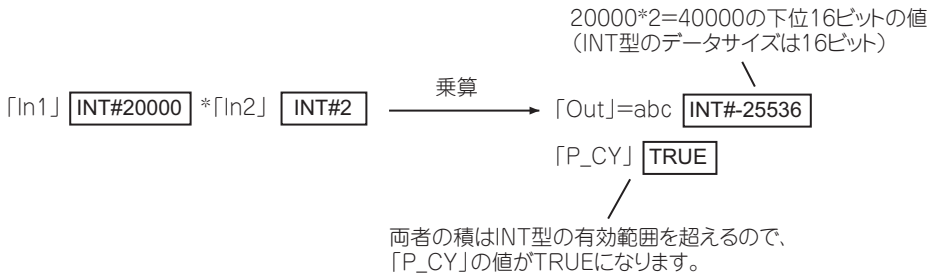
LD



ST

```
abc:=MulOU(INT#20000, INT#2);
```

「In1」～「InN」を乗算します。
両者の積40000は、INT型の有効範囲を超えるので、「P_CY」の値がTRUEになります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とで、仕様に違いはありません。また、ラダー図プログラムの MulOU 命令と、ラダー図プログラムの*OU 命令にも、仕様に違いはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	TRUE: 整数演算でオーバーフローあり FALSE: 整数演算でオーバーフローなし

参考

- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。
- オーバーフローのチェックが必要ない場合は、「MUL, * (P.2-183)」命令を使用してください。その方が処理時間が短くなります。

使用上の注意

- 「Out」は、乗算結果と異なるデータ型でもかまいません。ただし、乗算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 乗算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞に対する乗算結果は、つぎのようになります。

乗算内容	乗算結果
+∞と正数の乗算	+∞
+∞と負数の乗算	-∞
-∞と正数の乗算	-∞
-∞と負数の乗算	+∞
+∞と+∞の乗算	+∞
-∞と-∞の乗算	+∞
+∞と-∞の乗算	-∞

乗算内容	乗算結果
+∞と 0 の乗算	非数
-∞と 0 の乗算	非数

・「In1」～「InN」のいずれかの値が非数の場合、乗算結果の値は非数になります。

DIV, /

整数、実数を除算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DIV,/	除算	FUN	(@)DIV EN ENO In1 Out In2 (@)/ EN ENO In1 Out In2	Out:=In1/ In2;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被除算値	入力	被除算値	データ型に従う	−	*1
In2	除算値		除算値			
Out	出力値	出力	出力値	データ型に従う	−	−

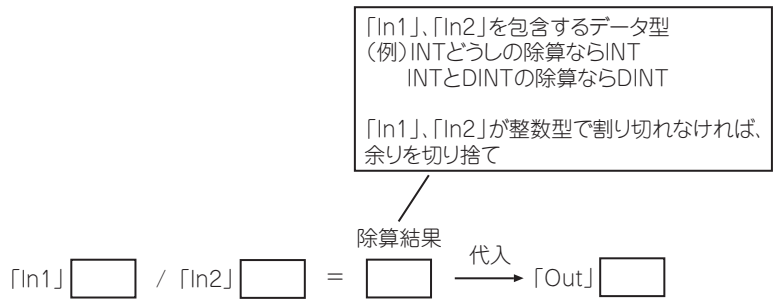
*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						○	○	○	○	○	○	○	○	○	○					
In2						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

被除算値「In1」を除算値「In2」で除算した結果を、出力値「Out」に出力します。

「In1」と「In2」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。その場合、それらすべてを包含するデータ型で、演算処理を行います。たとえば、「In1」が INT 型、「In2」が DINT 型の場合は、演算処理を DINT 型で行います。したがって、除算結果は DINT 型になります。
「In1」、「In2」がいずれも整数で、かつ割り切れない場合は、余りを切り捨てます。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。



オーバーフロー発生時の処理

「In1」と「In2」の商が、除算結果のデータ型の有効範囲を超えることをオーバーフローと呼びます。オーバーフローが発生した場合の、「In1」、「In2」のデータ型、除算結果のデータ型、除算結果の値の関係は、以下のとおりです。

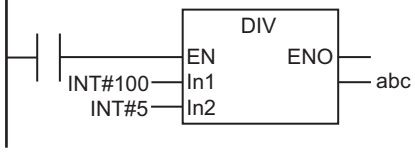
「In1」、「In2」のデータ型	除算結果のデータ型	除算結果の値
いずれかが実数型	実数型	±∞ *1

*1. 「In1」、「In2」の商が正数の場合は+∞、負数の場合は-∞になります。

記述例

「In1」=INT#100、「In2」=INT#5 の場合の例を、以下に示します。INT 型変数「abc」の値は、INT#20 になります。

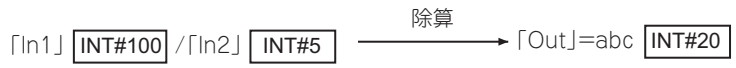
LD



ST

```
abc:=INT#100/INT#5;
```

「In1」を「In2」で除算します。
100/5=20なので、abcの値はINT#20になります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで使用した場合とでは、仕様に違いがあります。仕様の違いを以下に示します。なお、ラダー図プログラムの DIV 命令と、ラダー図プログラムの／命令は、まったく同じ仕様です。

仕様項目	ラダー図	ST
変数「EN」、「ENO」の存在	あり	なし
被除算値と除算値のデータ型が整数の場合の、演算処理のビット数	8, 16, 32, 64 *1	32, 64 *2

*1. 演算ビット数は、被除算値と除算値のうち、データ型のサイズが最大のものに合わせられます。たとえば、SINT 型と DINT 型の除算をする場合は、DINT 型のサイズに合わせて、32 ビットで演算されます。

*2. 被除算値と除算値の中に LINT 型および ULINT 型がなければ、32 ビットで演算されます。たとえば、SINT 型どうしの除算でも、32 ビットで演算されます。被除算値と除算値の中に LINT 型または ULINT 型がある場合は、64 ビットで演算されます。

参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

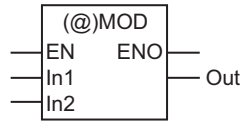
- 「Out」は、除算結果と異なるデータ型でもかまいません。ただし、除算結果のデータ型を包含するデータ型にしてください。包含していないと、ビルド時に異常となります。
  データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。
- 除算によってオーバーフローやアンダーフローが発生しても、異常にはなりません。
- 除算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- 実数の+∞/-∞および 0 に対する除算結果は、つぎのようになります。

		In1				
		+∞	正数	0	負数	-∞
In2	+∞	非数	0	0	0	非数
	正数	+∞	正数	0	負数	-∞
	0	+∞	+∞	非数	-∞	-∞
	負数	-∞	負数	0	正数	+∞
	-∞	非数	0	0	0	非数

- 「In1」、「In2」のいずれかの値が非数の場合、除算結果の値は非数になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In1」、「In2」が整数型で、「In2」の値が 0 の場合。

MOD

整数を除算したときの剰余を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MOD	剰余	FUN		Out:=In1 MOD In2;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被除算値	入力	被除算値	データ型に従う	−	*1
In2	除算値		除算値			
Out	剰余	出力	剰余	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						○	○	○	○	○	○	○	○							
In2						○	○	○	○	○	○	○	○							
Out						○	○	○	○	○	○	○	○							

機能

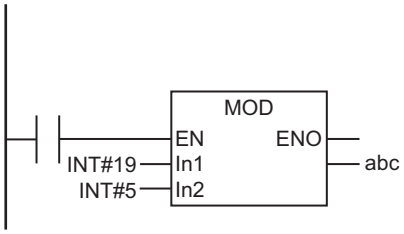
被除算値「In1」を除算値「In2」で除算したときの、剰余を求めます。
「In1」、「In2」および剰余「Out」は、異なるデータ型でもかまいませんが、包含する型が存在する組み合わせにしてください。
データ型の包含関係については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

本命令の演算は、つぎの式で行います。
「Out」＝「In1」-（「In1」/「In2」）*「In2」
ただし、かっこ内の除算は、小数点以下切り捨てです。
したがって、「In1」、「In2」と「Out」の値の例を示すと、以下のようになります。

「In1」の値	「In2」の値	「Out」の値
5	3	2
5	-3	2
-5	3	-2
-5	-3	-2

「In1」=INT#19、「In2」=INT#5 の場合の例を、以下に示します。変数「abc」の値は、INT#4 になります。

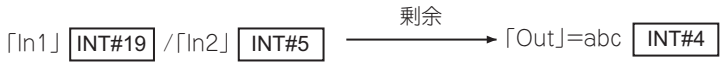
LD



ST

```
abc:=INT#19 MOD INT#5;
```

「In1」を「In2」で除算したときの剰余を求めます。
19/5の剰余は4なので、abcの値はINT#4になります。



使用上の注意

「Out」のデータ型は、「In1」、「In2」の有効範囲を包含するように設定してください。
データ型の包含関係については、 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』の「データ型のランク表」および「キャストルール」を参照してください。

ABS

整数、実数の絶対値を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ABS	絶対値	FUN	(@)ABS EN ENO In Out	Out:=ABS(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う	－	*1
Out	絶対値	出力	絶対値	データ型に従う *2	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 負数は含みません。

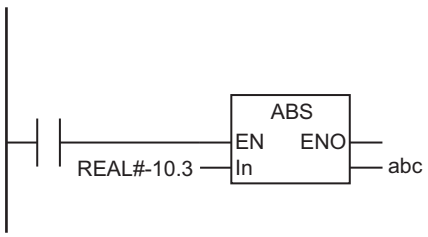
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

演算対象「In」の、絶対値を出力します。
「In」と絶対値「Out」は、異なるデータ型でもかまいません。

「In」=REAL# -10.3 の場合の例を、以下に示します。変数「abc」の値は、REAL#10.3 になります。

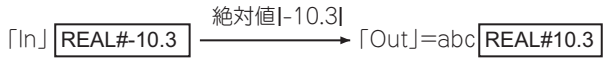
LD



ST

```
abc:=ABS(REAL#-10.3);
```

「In」の絶対値を出力します。
REAL#-10.3の絶対値なので、abcの値はREAL#10.3になります。



参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

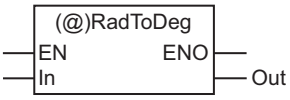
使用上の注意

- 「Out」のデータ型は、「In」の絶対値が収まるように設定してください。
- 「In」の値が+∞、-∞、非数の場合、「Out」の値は以下のようになります。

「In」の値	「Out」の値
+∞	+∞
-∞	+∞
非数	非数

RadToDeg / DegToRad

RadToDeg ： 実数をラジアン(rad)単位から度(°)単位に変換します。
DegToRad ： 実数を度(°)単位からラジアン(rad)単位に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RadToDeg	ラジアン→角度変換	FUN		Out:=RadToDeg(In);
DegToRad	角度→ラジアン変換	FUN		Out:=DegToRad(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換するデータ	データ型に従う	RadToDeg: rad DegToRad: °	*1
Out	変換結果	出力	変換結果	データ型に従う	RadToDeg: ° DegToRad: rad	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

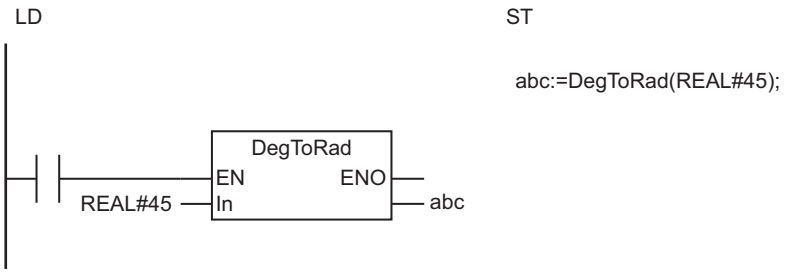
RadToDeg

変換対象「In」をラジアン(rad)単位から度(°)単位に変換します。
変換式は、以下のとおりです。
「Out」＝「In」*180/π

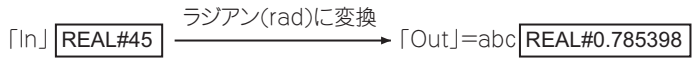
DegToRad

変換対象「In」を度(°)単位からラジアン(rad)単位に変換します。
変換式は、以下のとおりです。
「Out」＝「In」*π/180

DegToRad 命令で、「In」=REAL#45 の場合の例を、以下に示します。REAL 型変数「abc」の値は、REAL#0.785398 になります。



「In」の値を、度(°)からラジアン(rad)に変換します。
45°は、ラジアン(rad)では0.785398なので、abcの値はREAL#0.785398になります。



参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

- 変換結果の絶対値が、「Out」のデータ型で表現できる最大値よりも大きくなる場合、「Out」の値は+∞/-∞になります。
- 変換結果の絶対値が、「Out」のデータ型で表現できる最小値よりも小さくなる場合、「Out」の値は0 になります。
- 「Out」のデータ型は、「In」のデータ型以上のサイズにしてください。
- 「In」の値が+∞、-∞、非数の場合、「Out」の値は以下のようになります。

「In」の値	「Out」の値
+∞	+∞
-∞	-∞
非数	非数

- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

SIN / COS / TAN

実数の三角関数演算を行います。

SIN : sin（正弦値）
COS : cos（余弦値）
TAN : tan（正接値）

命令	名称	FB/FUN	グラフィック表現	ST 表現
SIN	SIN 演算	FUN	(@)SIN EN ENO In Out	Out:=SIN(In);
COS	COS 演算	FUN	(@)COS EN ENO In Out	Out:=COS(In);
TAN	TAN 演算	FUN	(@)TAN EN ENO In Out	Out:=TAN(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う	rad	*1
Out	演算結果	出力	演算結果	SIN: *2 COS: *2 TAN: データ型に従う	–	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. REAL の場合、-1.000000e+0 ～ 1.000000e+0 です。LREAL の場合、-1.00000000000000e+0 ～ 1.00000000000000e+0 です。

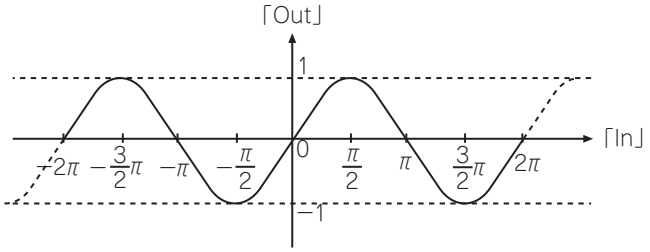
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

実数の三角関数演算を行います。
演算対象「In」はラジアン(rad)単位の角度です。

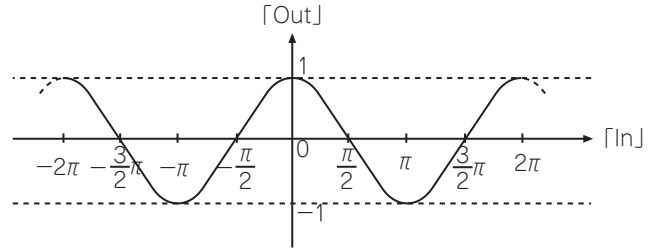
SIN

「In」の sin（正弦値）を、求めます。



COS

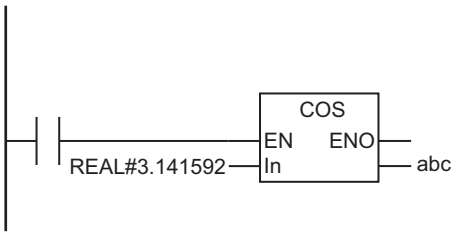
「In」の cos（余弦値）を、求めます。



● COS 命令の記述例

COS 命令で、「In」=REAL#3.141592 の場合の例を、以下に示します。変数「abc」の値は、REAL#-1.0 になります。

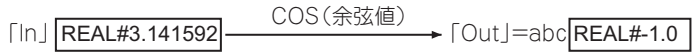
LD



ST

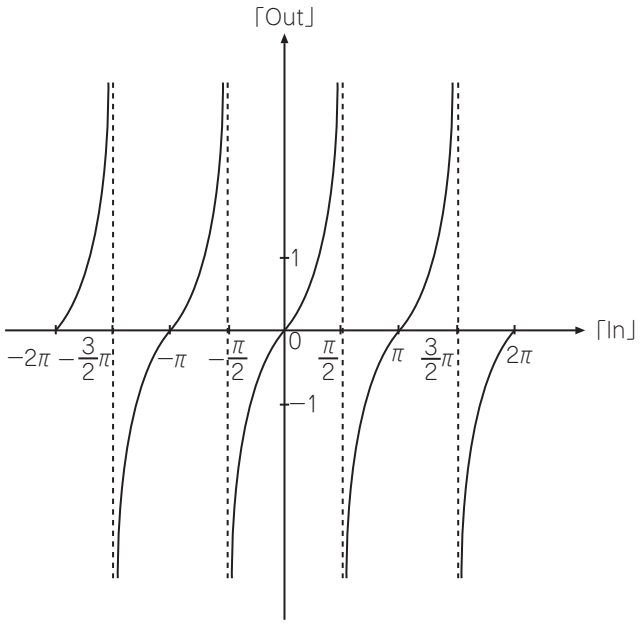
```
abc:=COS(REAL#3.141592);
```

「In」のCOS（余弦値）を求めます。
3.141592のCOS（余弦値）は-1.0なので、abcの値はREAL#-1.0になります。



TAN

「In」の tan（正接値）を、求めます。



参考

- 度(°)とラジアン(rad)の変換は、「RadToDeg / DegToRad (P.2-197)」命令を使用してください。
- TAN 命令では、「In」が nπ/2（n は整数）の場合、「Out」の値が+∞/-∞になります。
  「Out」の値が+∞/-∞であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

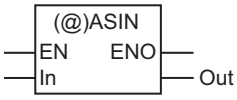
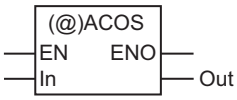
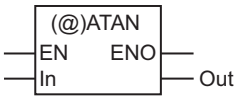
- 「In」の値が、+∞、-∞、非数の場合、「Out」の値は非数になります。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

ASIN / ACOS / ATAN

実数の逆三角関数演算を行います。

ASIN ： $\sin^{-1}$（逆正弦値）
ACOS ： $\cos^{-1}$（逆余弦値）
ATAN ： $\tan^{-1}$（逆正接値）

命令	名称	FB/FUN	グラフィック表現	ST 表現
ASIN	SIN^{-1} 演算	FUN		Out:=ASIN(In);
ACOS	COS^{-1} 演算	FUN		Out:=ACOS(In);
ATAN	TAN^{-1} 演算	FUN		Out:=ATAN(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う	−	*1
Out	演算結果	出力	演算結果	ASIN: -π/2 ～ π/2 ACOS: 0 ～ π ATAN: -π/2 ～ π/2	rad	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

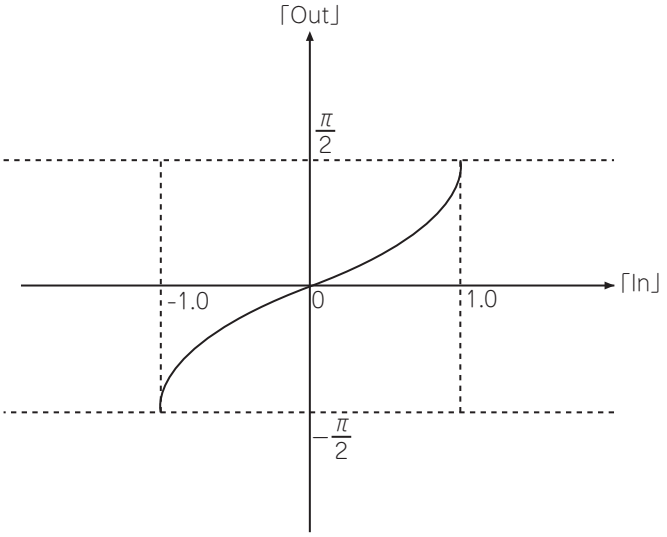
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

実数の逆三角関数演算を行います。
演算結果「Out」はラジアン単位の角度です。

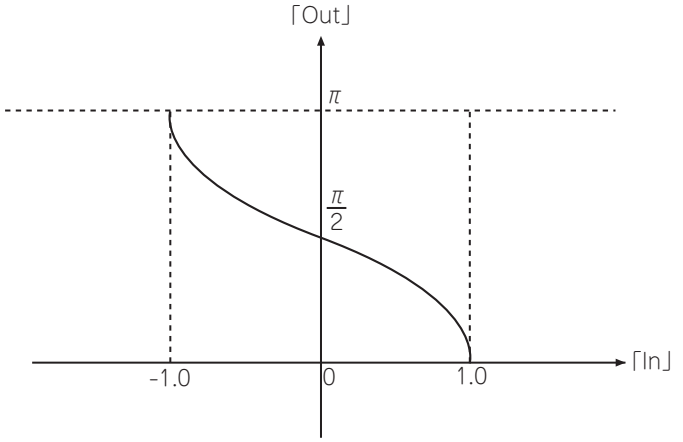
ASIN

「In」の $\sin^{-1}$（逆正弦値）を、出力します。「Out」は-π/2 ～ π/2 の範囲になります。



ACOS

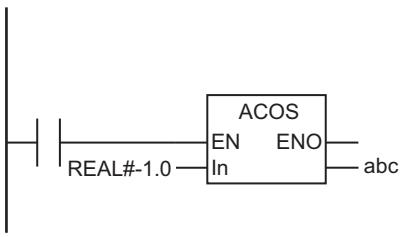
「In」の $\cos^{-1}$（逆余弦値）を、出力します。「Out」は、0 ～ π の範囲になります。



● ACOS 命令の記述例

ACOS 命令で、「In」=REAL#-1.0 の例を、以下に示します。変数「abc」の値は、REAL#3.141592 になります。

LD



ST

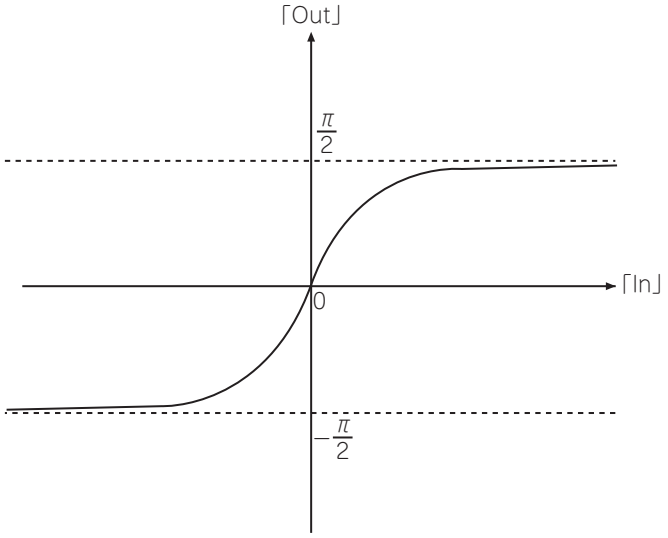
```
abc:=ACOS(REAL#-1.0);
```

「In」のCOS^{-1}（逆余弦値）を求めます。
-1.0のCOS^{-1}（逆余弦値）は3.141592なので、abcの値はREAL#3.141592になります。

「In」 REAL#-1.0 → COS^{-1}（逆余弦値） → 「Out」=abc REAL#3.141592

ATAN

「In」の tan^{-1}（逆正接値）を、出力します。「Out」は、-π/2 ～ π/2 の範囲になります。
「In」の値が+∞の場合、「Out」の値は π/2 となります。「In」の値が-∞の場合、「Out」の値は-π/2 となります。



参考

度(°)とラジアン(rad)の変換は、「RadToDeg / DegToRad (P.2-197)」命令を使用してください。

使用上の注意

- ASIN 命令、ACOS 命令で、「In」の値が-1.0～1.0 以外の場合、「Out」の値は非数になります。「In」の値が、+∞、-∞、非数の場合も同様です。
- ATAN 命令で、「In」の値が非数の場合、「Out」の値は非数になります。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

SQRT

実数の平方根を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SQRT	平方根演算	FUN	(@)SQRT EN ENO In Out	Out:=SQRT(In);

変数

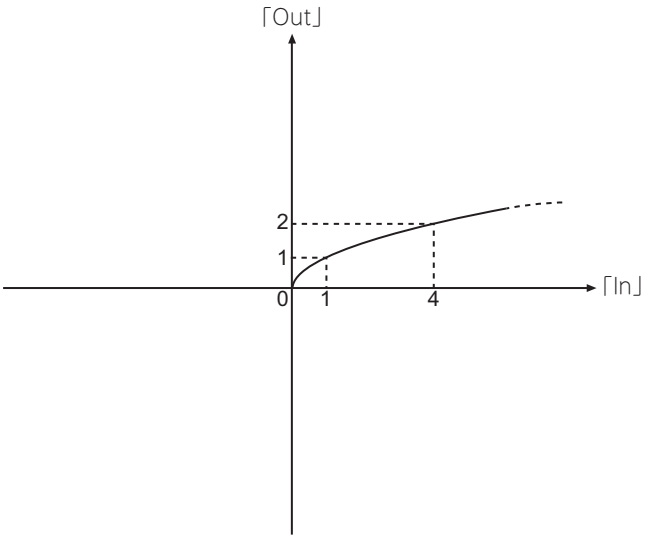
	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う *1	−	*2
Out	平方根	出力	平方根	*3	−	−

*1. 負数は含みません。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*3. REAL の場合、0.000000e+00 ～ 1.844674e+19 または+∞です。LREAL の場合、0.00000000000000e+000 ～ 1.34078079299425e+154 または+∞です。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

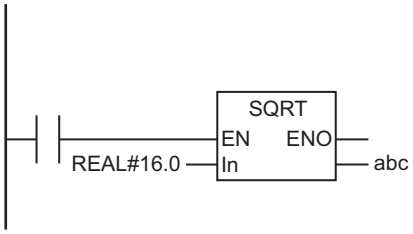
機能

実数の演算対象「In」の、平方根を求めます。
「In」と平方根「Out」は、異なるデータ型でもかまいません。



「In」=REAL#16.0 の場合の例を、以下に示します。変数「abc」の値は、REAL#4.0 になります。

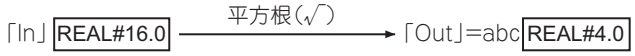
LD



ST

```
abc:=SQRT(REAL#16.0);
```

「In」の平方根(√)を求めます。
16.0の平方根(√)は4.0なので、abcの値はREAL#4.0になります。



参考

「Out」の値が+∞であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

- 「In」の値が正の数値ではない場合、「Out」の値は以下のようになります。

「In」の値	「Out」の値
負の数値	非数
0	0
+∞	+∞
-∞	非数
非数	非数

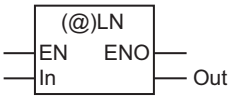
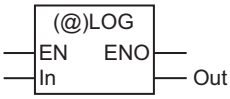
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

LN / LOG

実数の対数を求めます。

LN　：自然対数
LOG　：常用対数

命令	名称	FB/FUN	グラフィック表現	ST 表現
LN	自然対数演算	FUN		Out:=LN(In);
LOG	常用対数演算	FUN		Out:=LOG(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う *1	－	*2
Out	対数	出力	対数	*3	－	－

*1. 負数は含みません。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*3. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-209)」を参照してください。

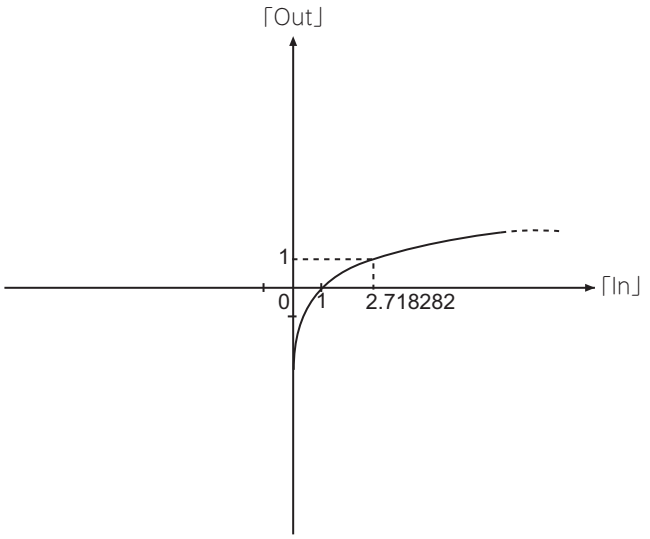
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

実数の対数を求めます。

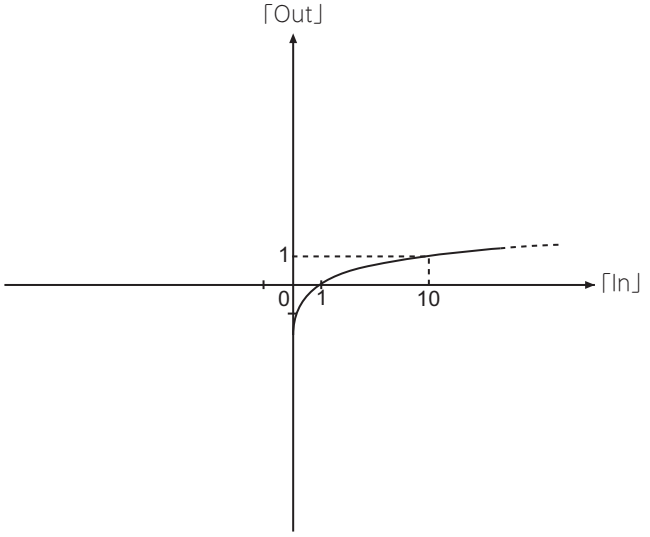
LN

自然対数（底が e 2.718282 の対数）を求めます。



LOG

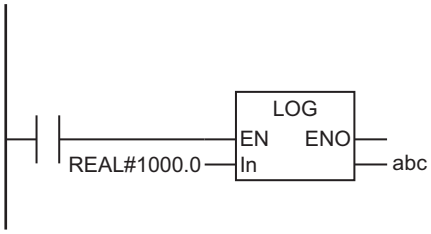
常用対数（底が 10 の対数）を求めます。



● LOG 命令での使用例

LOG 命令で、「In」=REAL#1000.0 の場合の例を、以下に示します。変数「abc」の値は、REAL#3.0 になります。

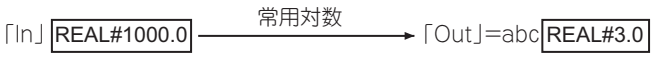
LD



ST

```
abc:=LOG(REAL#1000.0);
```

「In」の常用対数を求めます。
1000.0の常用対数は3.0なので、abcの値はREAL#3.0になります。



有効範囲

LN、および LOG の有効範囲は、次のようになります。

● LN の有効範囲

「In」のデータ型	「Out」のデータ型	有効範囲
REAL	REAL	-8.73365448e+1 ～ 8.87228390e+1 または-∞/+∞
REAL	LREAL	-8.7336544750000000e+1 ～ 8.8722839050000000e+1 または-∞/+∞
LREAL	REAL	-7.08384950e+2 ～ 7.09782712e+2 または-∞/+∞
LREAL	LREAL	-7.0838495021978327e+1 ～ 7.0978271289338399e+2 または-∞/+∞

● LOG の有効範囲

「In」のデータ型	「Out」のデータ型	有効範囲
REAL	REAL	-3.79297795e+1 ～ 3.85318394e+1 または-∞/+∞
REAL	LREAL	-3.7929779453965430e+1 ～ 3.8531839419564961e+1 または-∞/+∞
LREAL	REAL	-3.07652656e+2 ～ 3.08254716e+2 または-∞/+∞
LREAL	LREAL	-3.0765265556858878e+2 ～ 3.0825471555991674e+2 または-∞/+∞

参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

・「In」の値が正の数値ではない場合、「Out」の値は以下のようになります。

「In」の値	「Out」の値
負の数値	非数
0	-∞
+∞	+∞
-∞	非数
非数	非数

・「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

EXP

自然対数の底 e の指数関数を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EXP	自然指数演算	FUN	(@)EXP EN ENO In Out	Out:=EXP(In);

変数

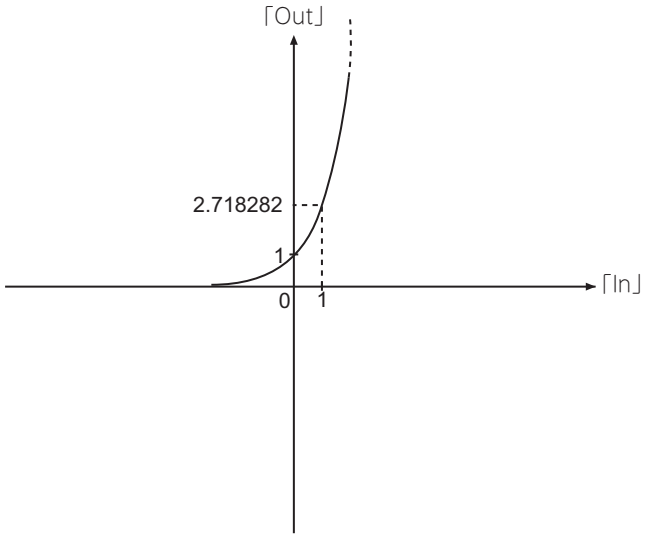
	名称	入／出力	内容	有効範囲	単位	初期値
In	べき乗数	入力	べき乗数	データ型に従う	−	*1
Out	演算結果	出力	演算結果	データ型に従う *2	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

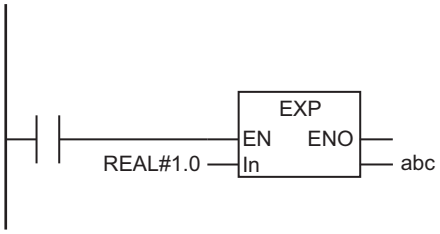
機能

自然対数の底 e の「In」乗を求めます。



「In」=REAL#1.0、の場合の例を、以下に示します。変数「abc」の値は、REAL#2.718282 になります。

LD



ST

```
abc:=EXP(REAL#1.0);
```

自然対数の底eの「In」乗を求めます。
e^1は2.718282なので、abcの値はREAL#2.718282になります。



参考

- e 以外を底とする指数を求める場合は、「EXPT, ** (P.2-213)」命令を使用してください。
- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

- 「In」の値が 0.0、+∞、-∞、非数の場合、「Out」の値は以下のようになります。

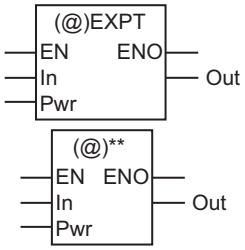
「In」の値	「Out」の値
0.0	1.0
+∞	非数
-∞	非数
非数	非数

- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

EXPT, **

2 つの実数のべき乗を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EXPT,**	べき乗演算	FUN		Out:=EXPT(In, Pwr); Out:=In ** Pwr;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	被べき乗数	入力	被べき乗数 5^2 の場合の 5	データ型に従う	−	*1
Pwr	べき乗数		べき乗数 5^2 の場合の 2			
Out	演算結果	出力	演算結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

ラダーの命令および ST での EXPT 命令の場合

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Pwr														○	○					
Out														○	○					

ST での**演算子の場合

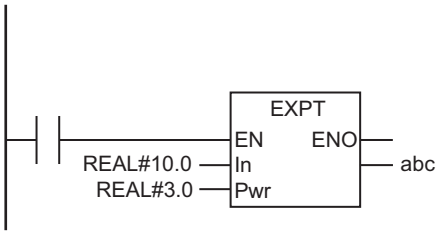
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Pwr						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

被べき乗数「In」とべき乗数「Pwr」に対して、In^{Pwr} の値を求めます。

「In」=REAL#10.0、「Pwr」=REAL#3.0 の場合の例を、以下に示します。変数「abc」の値は、REAL#1000.0 になります。

LD



ST

```
abc:=EXPT(REAL#10.0, REAL#3.0);
```

「In」の「Pwr」乗を求めます。
$10.0^{3.0}$は1000.0なので、abcの値はREAL#1000.0になります。



ラダー図と ST の仕様の違い

本命令をラダー図プログラムで使用した場合と、ST プログラムで**演算子を使用した場合とでは、仕様に違いがあります。仕様の違いを以下に示します。なお、ラダー図プログラムの EXPT 命令と、ラダー図プログラムの**命令、ST プログラムの EXPT ファンクションは、まったく同じ仕様です。

仕様項目	ラダー図および ST での EXPT ファンクション	ST での**演算子
変数「EN」、「ENO」の存在	あり	なし
「In」、「Pwr」が整数型の場合の演算処理のビット数	32, 64 *1	64 *2

*1. REAL 型または LREAL 型のうち、包含する小さい方の実数型で演算されます。例えば、SINT 型と DINT 型との演算をする場合は、LREAL 型にあわせて、64 ビットで演算されます。
*2. 64 ビットで演算されます。たとえば、SINT 型同士のべき乗でも 64 ビットで演算されます。

参考

- e を底とする指数を求める場合は、「EXP (P.2-211)」命令を使用してください。
- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

- 演算結果の絶対値が実数で表現できる最小値より小さくなるとき、「Out」の値は 0 になります。
  （例）$(1.175494e\text{-}38)^2 \rightarrow 0$
- **演算子の場合、演算によってオーバーフローやアンダーフローが発生しても、異常にはなりません。

- **演算子の場合、演算によってオーバーフローやアンダーフローが発生する場合、演算結果が意図どおりにならない可能性があります。入力パラメータ、出力パラメータのデータ型のサイズには、オーバーフローやアンダーフローが発生しないように、十分な余裕をもたせてください。
- ラダー図の EXPT 命令、**命令、および ST の EXPT ファンクションの場合、「In」に整数型のパラメータを渡すと、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

- **演算子の場合、NY シリーズコントローラでは、Sysmac Studio のプロジェクトのデバイス選択画面にある［バージョン］に［Ver.1.15］以前を選択した場合、整数型の変数をオペランドに設定しても実数型で演算されます。
  演算結果に丸め誤差が含まれる場合は、小数点以下が切り捨てられるため、意図しない値になることがあります。
  小数点以下を丸める場合には、EXPT 命令と TO_**（整数変換グループ）命令を組み合わせて使用してください。
  例）TO_INT(EXPT(X,Y))

「In」と「Pwr」の値の組み合わせ

「In」と「Pwr」の値の組み合わせにより、「Out」の値はつぎのようになります。

●「EXPT」FUN を使用する場合

		In									
		+∞	1 ～ +∞	1	0 ～ 1	0	-1 ～ 0	-1	-1 ～ -∞	-∞	非数
Pwr	+∞	+∞	+∞	1	0	1	0	1	+∞	1	非数
	正の偶数	+∞	数値 *1 *2	1	数値 *1 *2	0	数値 *1 *2	1	数値 *1 *2	+∞	非数
	正の奇数						数値 *2 *3	-1	数値 *2 *3		
	正の小数						非数				
	0	1	1			1	1			1	1
	負の偶数	0	数値 *1 *2	1	数値 *1 *2	+∞	数値 *1 *2	1	数値 *1 *2	0	非数
	負の奇数						数値 *2 *3	-1	数値 *2 *3		
	負の小数						非数				
	-∞	0	0	1	+∞	+∞	+∞	1	0	0	非数
	非数	1	非数	1	非数	1	非数			1	非数 1

*1. 演算結果が、「Out」のデータ型の有効範囲を超えた場合、「Out」の値は、「+∞」になります。
*2. 演算結果が、「Out」のデータ型では表現できないほど「0」に近い場合、または非正規化数の場合、「Out」の値は、「0」になります。
*3. 演算結果が、「Out」のデータ型の有効範囲を超えた場合、「Out」の値は、「-∞」になります。

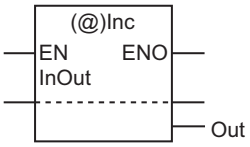
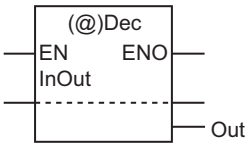
●「**」演算子を使用する場合

		In								
		+∞	1 ～ +∞	1	0 ～ 1	0	-1 ～ 0	-1	-1 ～ -∞	-∞	非数
Pwr	+∞	+∞	+∞	1	0	1	0	1	+∞	1	非数
	正の偶数	+∞	数値 *1 *2 *3	1	数値 *1 *2	0	数値 *1 *2	1	数値 *1 *2 *3	+∞	非数
	正の奇数						数値 *2 *4	-1	数値 *2 *3 *4	-∞	
	正の小数						非数			+∞	
	0	1	1			1	1			1	1
	負の偶数	0	数値 *1 *2	1	数値 *1 *2	+∞ *5	数値 *1 *2	1	数値 *1 *2	0	非数
	負の奇数						数値 *2 *4	-1	数値 *2 *4	-0	
	負の小数						非数			0	
	-∞	0	0	1	+∞	+∞	+∞	1	0	0	非数
	非数	1	非数			1	非数			1	非数 1

*1. 演算結果が、「Out」のデータ型の有効範囲を超えた場合、「Out」の値は、「+∞」になります。
*2. 演算結果が、「Out」のデータ型では表現できないほど「0」に近い場合、または非正規化数の場合、「Out」の値は、「0」になります。
*3. 「In」、「Pwr」の両方が整数型の場合、演算結果が「Out」のデータ型の有効範囲を超えると、「Out」は不定値となります。
*4. 演算結果が、「Out」のデータ型の有効範囲を超えた場合、「Out」の値は、「-∞」になります。
*5. 「In」、「Pwr」の両方が整数型の場合、「Out」は不定値となります。

Inc / Dec

Inc ： 整数値をインクリメントします。
Dec ： 整数値をデクリメントします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Inc	インクリメント	FUN		Inc(InOut);
Dec	デクリメント	FUN		Dec(InOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	対象データ	入出力	対象データ	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut						○	○	○	○	○	○	○	○							
Out	○																			

機能

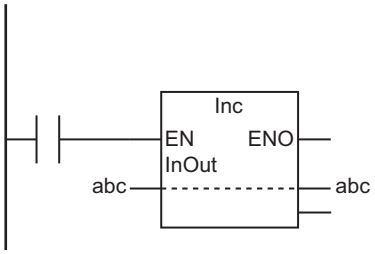
Inc

対象データ「InOut」をインクリメントします。その結果、「InOut」の最大値を超えた場合は、最小値に戻します。

● Inc 命令の記述例

Inc 命令で、「InOut」に変数「abc」を渡す場合の記述例を、以下に示します。
「abc」の値が INT#4 の場合、命令実行後の「abc」の値は INT#5 になります。

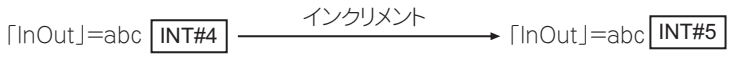
LD



ST

```
Inc(abc);
```

「InOut」をインクリメントします。
abcの値がINT#4の場合、命令実行後のabcの値はINT#5になります。



Dec

対象データ「InOut」をデクリメントします。その結果、「InOut」の最小値を超えた場合は、最大値に戻します。

使用上の注意

本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。

Rand

疑似乱数を発生します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Rand	乱数発生	FB	Rand_instance Rand Execute ENO Seed Rnd	Rand_instance(Execute, Seed, Rnd);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Seed	乱数パターン	入力	乱数系列の指定 0: 指定なし	データ型に従う	−	*1
Rnd	乱数	出力	乱数	*2	−	−

*1. 入力パラメータが省略された場合、値は 0 になります。初期値属性で指定された値にはなりません。
*2. 0.00000000000000e+0 ～ 1.00000000000000e+0

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Seed							○													
Rnd															○					

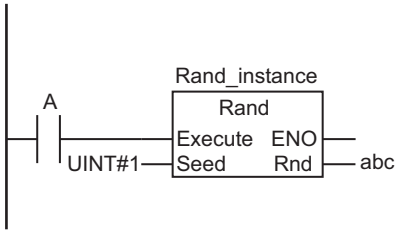
機能

乱数「Rnd」を発生します。「Rnd」の値は、命令実行のたびに異なります。
乱数パターン「Seed」は、乱数系列の指定です。「Seed」の値が同じであれば、電源を ON するたびに、同じ乱数系列が発生します。したがって、再現性のある乱数を発生させることが可能です。

「Seed」の値が 0 の場合は、再現性のない乱数が発生します。電源 ON のたびに同じ乱数系列を発生させたくない場合は、「Seed」の値を 0 にしてください。

「Seed」=UINT#1 の場合の記述例を、以下に示します。「Seed」の値が 0 ではないので、再現性のある乱数を発生します。

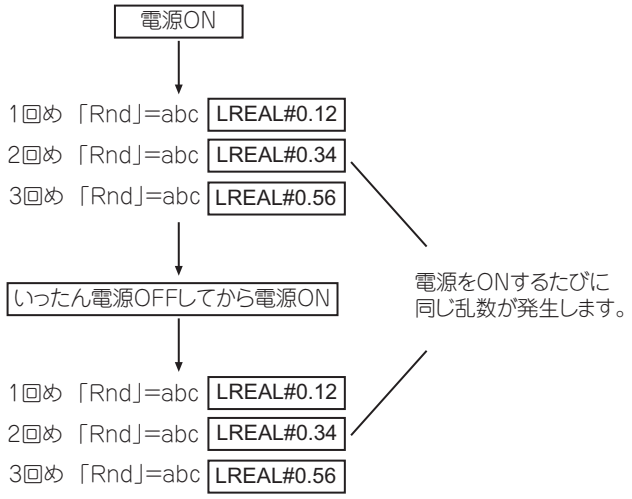
LD



ST

```
Rand_instance(A, UINT#1, abc);
```

再現性のある乱数を発生します。



（注）上記の乱数の値は例です。実際の値とは異なります。

参考

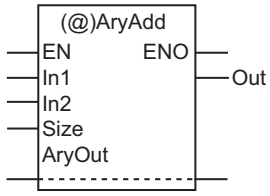
「Rnd」の値は、0～1 の範囲の実数です。したがって、以下の処理をすることで、特定の範囲の乱数を発生させることができます。

（例）100～200 の範囲の乱数を発生させる場合

```
Rand_instance(A, UINT#1, abc);
乱数:=LREAL_TO_INT((201-100)*abc-0.5)+100;
```

AryAdd

2 つの配列の、要素ごとの加算を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryAdd	配列一括加算	FUN		AryAdd(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列 In2[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	演算対象の要素数		演算対象の要素数			1
AryOut[] 配列	演算結果配列	入出力	演算結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

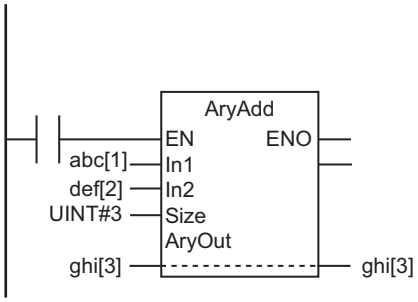
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2[] 配列	In1[]と同じデータ型の配列																			
Size							○													
AryOut[] 配列	In1[]と同じデータ型の配列																			
Out	○																			

機能

演算対象配列 In1[]、In2[]の、In1[0]、In2[0]以降の「Size」個の要素のそれぞれの加算値を、演算結果配列 AryOut[]に代入します。

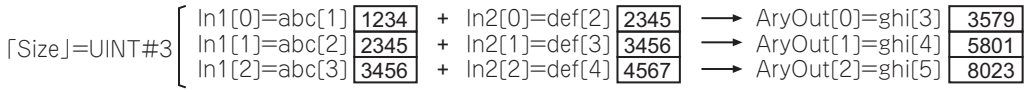
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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

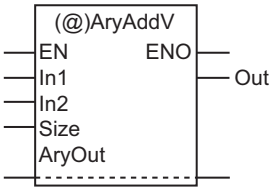


使用上の注意

- In1[]、In2[]、AryOut[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- 演算結果が、AryOut[]の有効範囲外になると、結果は不正値になります。この場合、異常にはなりません。また、その要素の隣のメモリ領域を破壊することもありません。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、In1[]、In2[]、AryOut[]のいずれかの配列領域を超える場合。

AryAddV

配列の各要素に、同一数値を加算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryAddV	配列要素加算	FUN		AryAddV(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列	加算配列	入力	加算配列	データ型に従う	−	*1
In2	加算値		加算値			
Size	要素数		加算する In1[]の要素数			1
AryOut[] 配列	加算結果配列	入出力	加算結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

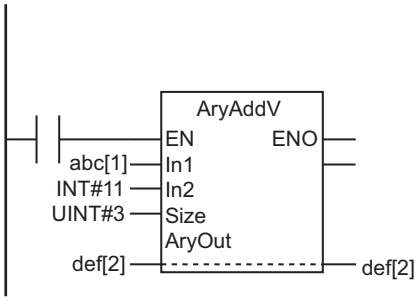
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2	In1[]の要素と同じデータ型																			
Size							○													
AryOut[] 配列	In1[]と同じデータ型																			
Out	○																			

機能

加算配列 In1[]の、In1[0]からの「Size」個の各要素に、加算値「In2」の値を加算し、加算結果配列 AryOut[]に出力します。

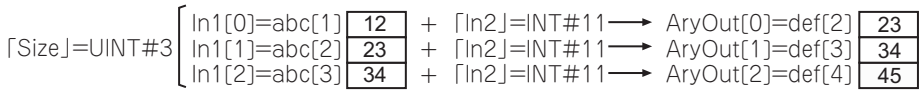
「In2」=INT#11、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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

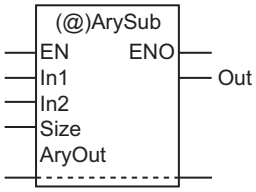


使用上の注意

- In1[]、「In2」、AryOut[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- 加算結果が、AryOut[]の有効範囲外になった場合、AryOut[]の要素は不正値になります。この場合、異常にはなりません。また、その要素の隣のメモリ領域を破壊することもありません。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、In1[]または AryOut[]の配列領域を超える場合。

ArySub

2 つの配列の、要素ごとの減算を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySub	配列一括減算	FUN		ArySub(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列	被減算配列	入力	被減算配列	データ型に従う	−	*1
In2[] 配列	減算配列		減算配列			
Size	要素数		減算する要素数			1
AryOut[] 配列	減算結果配列	入出力	減算結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

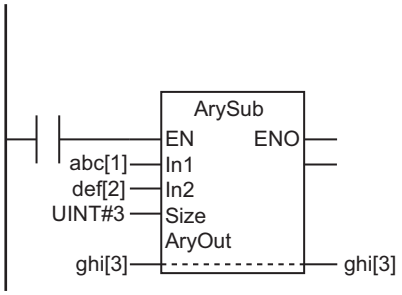
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2[] 配列	In1[]と同じデータ型																			
Size							○													
AryOut[] 配列	In1[]と同じデータ型																			
Out	○																			

機能

被減算配列 In1[]と減算配列 In2[]に対して、先頭から「Size」個の各要素の減算を行います。減算結果は、減算結果配列 AryOut[]に出力します。

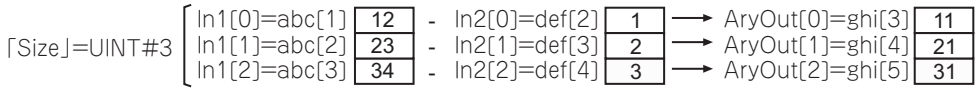
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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

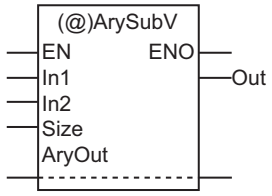


使用上の注意

- In1[]、In2[]、AryOut[]のデータ型は同じにしてください。異なると、ビルド時に異常になります。
- 減算結果が、AryOut[]の有効範囲外になった場合、AryOut[]の要素は不正値になります。この場合、異常にはなりません。また、その要素の隣のメモリ領域を破壊することもありません。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、In1[]、In2[]、AryOut[]のいずれかの配列領域を超える場合。

ArySubV

配列の各要素から、同一数値を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySubV	配列要素減算	FUN		ArySubV(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[] 配列	被減算配列	入力	被減算配列	データ型に従う	−	*1
In2	減算値		減算値			
Size	要素数		減算する In1[]の要素数			1
AryOut[] 配列	減算結果配列	入出力	減算結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

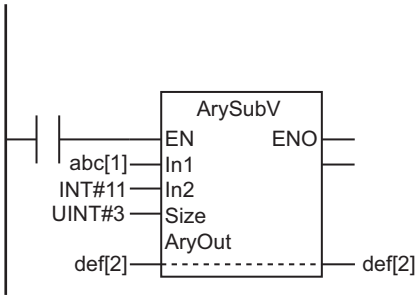
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列						○	○	○	○	○	○	○	○	○	○					
In2	In1[]の要素と同じデータ型																			
Size							○													
AryOut[] 配列	In1[]と同じデータ型																			
Out	○																			

機能

被減算配列 In1[]の、In1[0]からの「Size」個の各要素から、減算値「In2」の値を減算し、減算結果配列 AryOut[]に出力します。

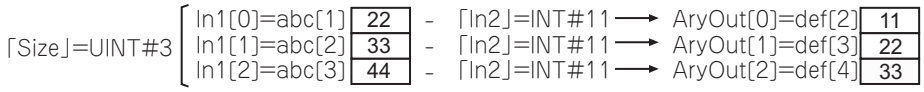
「In2」=INT#11、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



使用上の注意

- In1[]、「In2」、AryOut[]のデータ型は同じにしてください。異なると、ビルド時に異常になります。
- 減算結果が、AryOut[]の有効範囲外になった場合、AryOut[]の要素は不正値になります。この場合、異常にはなりません。また、その要素の隣のメモリ領域を破壊することもありません。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、In1[]または AryOut[]の配列領域を超える場合。

AryMean

配列要素の平均値を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryMean	配列要素の平均値算出	FUN	(@)AryMean EN ENO In Out Size	Out := AryMean(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	演算対象の要素数		In[]の要素数			1
Out	演算結果	出力	演算結果	データ型に従う	−	−

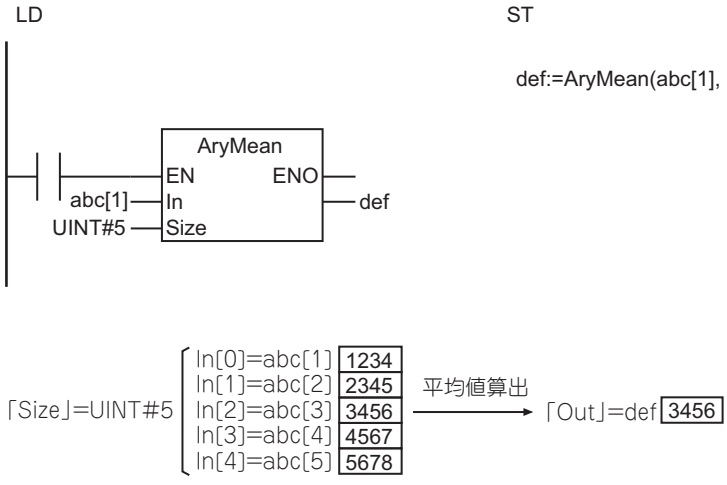
*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列						○	○	○	○	○	○	○	○	○	○	○				
Size							○													
Out						○	○	○	○	○	○	○	○	○	○	○				

機能

演算対象配列 In[]の、In[0]以降の「Size」個の要素の平均値を算出します。

「Size」=UINT#5 の場合の例を、以下に示します。



使用上の注意

- In[]の値が+∞/-∞および非数の場合の四則演算の結果は、「ADD, + (P.2-169)」命令、「SUB, - (P.2-176)」命令、「MUL, * (P.2-183)」命令、「DIV, / (P.2-190)」命令の機能説明を参照してください。
- In[]、「Out」が整数型の場合、平均値の小数点以下を切り捨てます。
- In[]と「Out」のデータ型が異なる場合は、「Out」の有効範囲が In[]の有効範囲を包含するようにしてください。
- 演算結果が「Out」の有効範囲を超える場合、「Out」の値は不正値になります。その場合、異常にはなりません。
- 演算過程における中間値が、In[]の有効範囲を超えた場合、「Out」の値は不正値になります。その場合、異常にはなりません。
- 「Size」の値が 0 の場合、「Out」の値は 0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、In[]の配列領域を超える場合。

ArySD

配列要素の、標準偏差を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySD	配列要素の標準偏差	FUN	(@)ArySD EN ENO In Out Size	Out:=ArySD(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	要素数		変換する In[]の要素数			2
Out	標準偏差	出力	標準偏差	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列														○	○					
Size							○													
Out														○	○					

機能

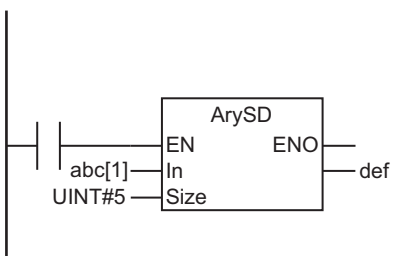
演算対象配列 In[]の、In[0]からの「Size」個の要素の標準偏差を算出します。
本命令の標準偏差の算出には不偏分散を用いています。

$$\text{標準偏差}=\sqrt{\frac{\sum_{i}(In[i]-InM)^2}{「Size」-1}}$$

i :In[]の添え字 0～「Size」-1
InM:In[0]～In[「Size」-1]の平均値

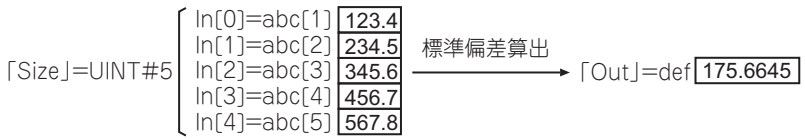
「Size」=UINT#5 の場合の例を、以下に示します。

LD



ST

```
def:=ArySD(abc[1], UINT#5);
```

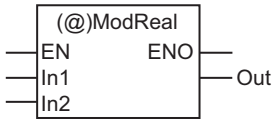


使用上の注意

- 「Size」の値が 0 または 1 の場合、「Out」の値は 0 になります。
- 演算過程における中間値が、In[]の有効範囲を超えた場合、「Out」の値は不正値になります。その場合、異常にはなりません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、In[]の配列領域を超える場合。

ModReal

実数を除算したときの剰余を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ModReal	実数剰余	FUN		Out:=ModReal(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被除算値	入力	被除算値	データ型に従う	–	*1
In2	除算値		除算値			
Out	剰余	出力	剰余	データ型に従う	–	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														○	○					
In2														○	○					
Out														○	○					

機能

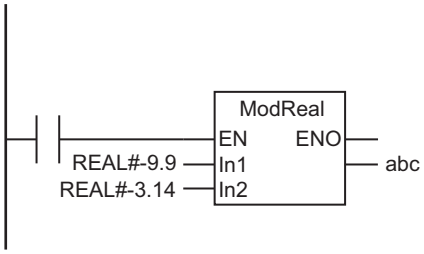
被除算値「In1」を、除算値「In2」で割ったときの剰余を算出します。

本命令の演算は、つぎの式で行います。
「Out」＝「In1」-（「In1」/「In2」）*「In2」
ただし、かっこ内の除算は、小数点以下切り捨てです。
したがって、「In1」、「In2」と「Out」の値の例を示すと、以下のようになります。

「In1」の値	「In2」の値	「Out」の値
9.9	3.14	0.48
9.9	-3.14	0.48
-9.9	3.14	-0.48
-9.9	-3.14	-0.48

「In1」=REAL#-9.9、「In2」=REAL#-3.14 の場合の例を以下に示します。
変数「abc」の値は REAL#-0.48 になります。

LD



ST

```
abc:=ModReal(REAL#-9.9, REAL#-3.14);
```

「In1」を「In2」で除算したときの剰余を求めます。
-9.9/(-3.14)の剰余は-0.48なので、abcの値はREAL#-0.48になります。

「In1」 REAL#-9.9 /「In2」 REAL#-3.14 ——剰余——→「Out」=abc REAL#-0.48

参考

実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。

使用上の注意

・「In1」と「In2」の値の組み合わせにより、「Out」の値は以下のようになります。

		In1				
		0	数値	+∞	-∞	非数
In2	0	非数	非数	非数	非数	非数
	数値	0	「In1」/「In2」の剰余	非数	非数	非数
	+∞	0	「In1」の値	非数	非数	非数
	-∞	0	「In1」の値	非数	非数	非数
	非数	非数	非数	非数	非数	非数

・「In1」または「In2」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In1」または「In2」に渡すパラメータのデータ型	「In1」または「In2」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

Fraction

実数の、小数部を抽出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Fraction	実数小数部抽出	FUN	(@)Fraction EN ENO In Out	Out:=Fraction(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	実数	入力	実数	データ型に従う	−	*1
Out	小数部	出力	小数部	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

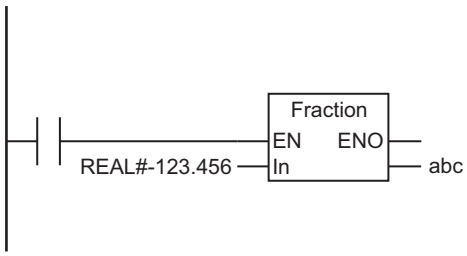
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

実数「In」の小数部を抽出します。

「In」=REAL#-123.456 の場合の例を、以下に示します。
変数「abc」の値は REAL#-0.456 になります。

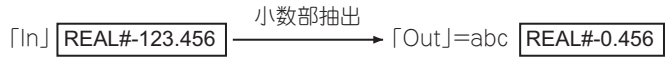
LD



ST

```
abc:=Fraction(REAL#-123.456);
```

「In」の小数部を抽出します。
-123.456の小数部は-0.456なので、abcの値はREAL#-0.456になります。



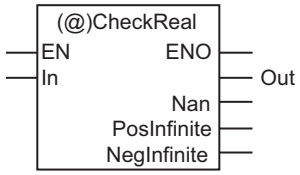
参考

- 実数を演算する場合、「Out」が+∞、-∞、非数であるかどうかの判定には、「CheckReal (P.2-237)」命令を使用してください。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

CheckReal

実数が、無限大または非数であるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CheckReal	実数チェック	FUN		CheckReal(In, Nan, PosInfinite, NegInfinite);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	実数	入力	実数	データ型に従う	−	*1
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−
Nan	非数判定結果		TRUE: 非数 FALSE: 非数ではない	データ型に従う		
PosInfinite	正の無限大判定結果		TRUE: 正の無限大 FALSE: 正の無限大ではない			
NegInfinite	負の無限大判定結果		TRUE: 負の無限大 FALSE: 負の無限大ではない			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

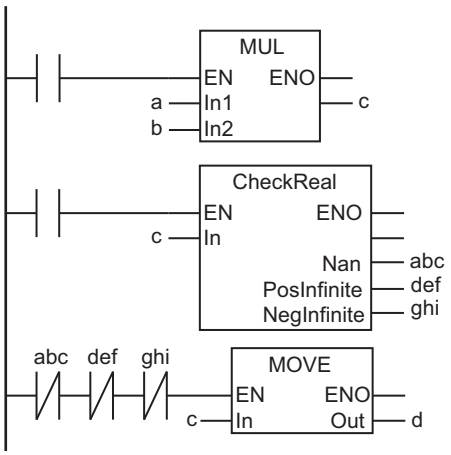
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	○																			
Nan	○																			
PosInfinite	○																			
NegInfinite	○																			

機能

実数「In」が、非数、正の無限大、負の無限大であるかを判定し、それぞれ「Nan」、「PosInfinite」、「NegInfinite」に結果を出力します。

記述例を、以下に示します。REAL 型の変数「a」、「b」を乗算した結果が、有効な実数であるかどうかを調べます。有効な実数であれば、乗算結果を変数「d」に代入します。

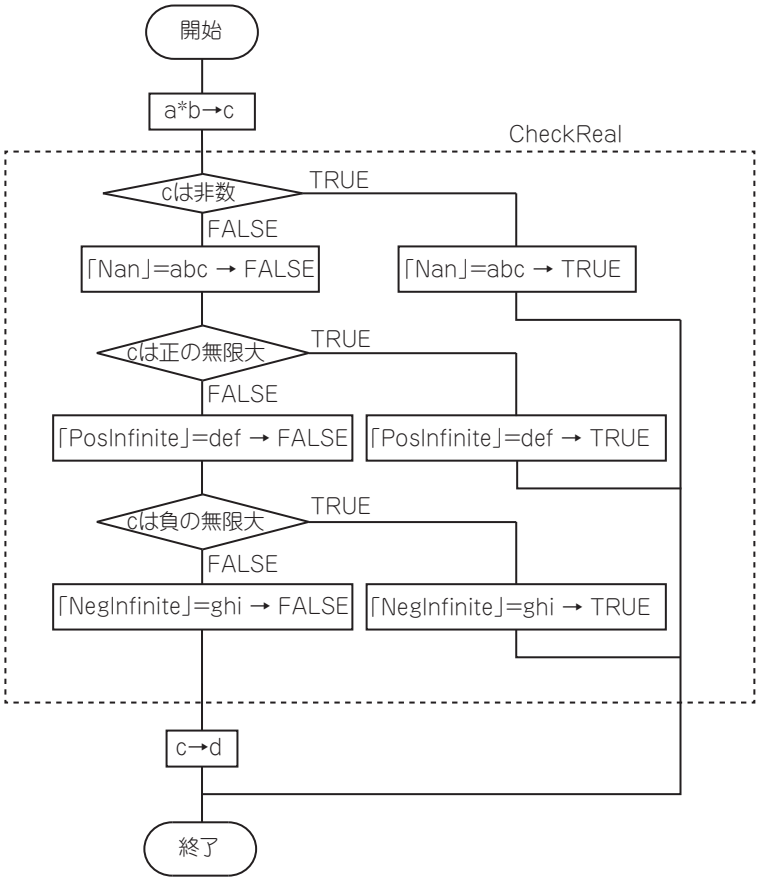
LD



ST

```
c:=a*b;
CheckReal(c, abc, def, ghi);
IF ( (abc=FALSE) AND (def=FALSE) AND (ghi=FALSE) ) THEN
    d:=c;
END_IF;
```

aとbの積cが、非数、正の無限大、負の無限大のいずれでもなければ
cをdに代入します。



参考

本命令は、実数を用いた演算命令の結果が、非数、正の無限大、負の無限大であるかどうかを判定する場合に使用してください。

使用上の注意

- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

BCD 変換命令

命令	名称	ページ
_BCD_TO_*	BCD→符号なし整数変換グループ	P.2-242
_TO_BCD_*	符号なし整数→BCD 変換グループ	P.2-245
BCD_TO_**	BCD 群→符号なし整数変換グループ	P.2-248
BCDsToBin	符号付き BCD→符号付き整数変換	P.2-251
BinToBCDs_**	符号付き整数→BCD 変換グループ	P.2-254
AryToBCD	配列一括 BCD 変換	P.2-257
AryToBin	配列一括符号なし整数変換	P.2-259

_BCD_TO_*

BCD のビット列を、符号なし整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_BCD_TO_*	BCD→符号なし整数変換グループ	FUN	(@)**_BCD_TO_*** EN　ENO In　Out ただし、“**”はビット列のデータ型名、“***”は整数のデータ型名	Out:=**_BCD_TO_***(In); ただし、“**”はビット列のデータ型名、“***”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-243)」を参照してください。

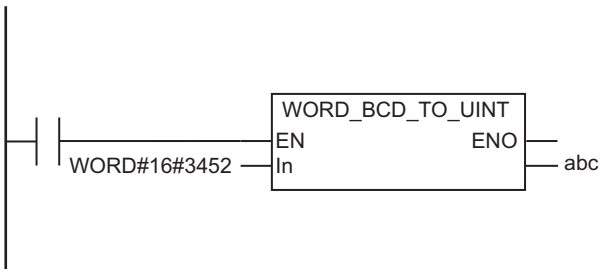
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○	○	○	○	○	○	○	○							

機能

BCD のビット列である変換対象「In」を、符号なし整数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が WORD 型、「Out」が UINT 型の場合、命令名称は WORD_BCD_TO_UINT となります。

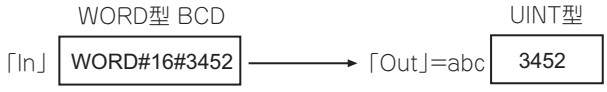
WORD_BCD_TO_UINT 命令で、「In」=WORD#16#3452 の場合の例を、以下に示します。

LD



ST

```
abc:=WORD_BCD_TO_UINT(WORD#16#3452);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
BYTE	USINT	16#00 ～ 16#99(BCD)	0 ～ 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 ～ 16#0255(BCD)	0 ～ 255
	UINT	16#0000 ～ 16#9999(BCD)	0 ～ 9999
	UDINT		
	ULINT		
	SINT	16#0000 ～ 16#0127(BCD)	0 ～ 127
	INT	16#0000 ～ 16#9999(BCD)	0 ～ 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 ～ 16#0000_0255(BCD)	0 ～ 255
	UINT	16#0000_0000 ～ 16#0006_5535(BCD)	0 ～ 65535
	UDINT	16#0000_0000 ～ 16#9999_9999(BCD)	0 ～ 99999999
	ULINT		
	SINT	16#0000_0000 ～ 16#0000_0127(BCD)	0 ～ 127
	INT	16#0000_0000 ～ 16#0003_2767(BCD)	0 ～ 32767
	DINT	16#0000_0000 ～ 16#9999_9999(BCD)	0 ～ 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0255(BCD)	0 ～ 255
	UINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0006_5535(BCD)	0 ～ 65535
	UDINT	16#0000_0000_0000_0000 ～ 16#0000_0042_9496_7295(BCD)	0 ～ 4294967295
	ULINT	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD	0 ～ 9999999999999999
	SINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0127(BCD)	0 ～ 127
	INT	16#0000_0000_0000_0000 ～ 16#0000_0000_0003_2767(BCD)	0 ～ 32767
	DINT	16#0000_0000_0000_0000 ～ 16#0000_0021_4748_3647(BCD)	0 ～ 2147483647
	LINT	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)	0 ～ 9999999999999999

参考

- 任意の BCD ビット列から整数への変換は、「BCD_TO_** (P.2-248)」命令を使用してください。
- 整数から BCD ビット列への変換は、「**_TO_BCD_*** (P.2-245)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「In」の値が、有効範囲外の場合。
  b）「In」の値が、BCD のビット列でない（16 進数表記で A,B,C,D,E,F のいずれかが含まれる）場合。

_TO_BCD_*

符号なし整数を、BCD のビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_BCD_*	符号なし整数→BCD 変換グループ	FUN	(@)**_TO_BCD_*** EN ENO In Out ただし、“**”は整数のデータ型名、“***”はビット列のデータ型名	Out:=**_TO_BCD_***(In); ただし、“**”は整数のデータ型名、“***”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-246)」を参照してください。

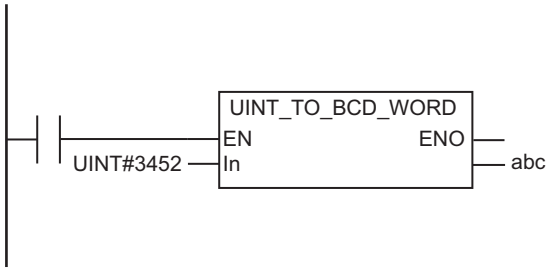
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Out		○	○	○	○															

機能

符号なし整数の変換対象「In」を、BCD のビット列に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が UINT 型、「Out」が WORD 型の場合、命令名称は UINT_TO_BCD_WORD となります。

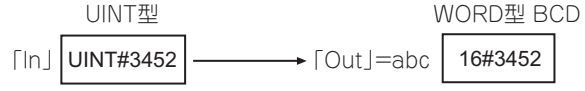
UINT_TO_BCD_WORD 命令で、「In」=UINT#3452 の場合の例を、以下に示します。

LD



ST

```
abc:=UINT_TO_BCD_WORD(UINT#3452);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
USINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 255	16#0000 ～ 16#0255(BCD)
	DWORD		16#0000_0000 ～ 16#0000_0255(BCD)
	LWORD		16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0255(BCD)
UINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 65535	16#0000_0000 ～ 16#0006_5535(BCD)
	LWORD		16#0000_0000_0000_0000 ～ 16#0000_0000_0006_5535(BCD)
UDINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 99999999	16#0000_0000 ～ 16#9999_9999(BCD)
	LWORD	0 ～ 4294967295	16#0000_0000_0000_0000 ～ 16#0000_0042_9496_7295(BCD)
ULINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 99999999	16#0000_0000 ～ 16#9999_9999(BCD)
	LWORD	0 ～ 9999999999999999	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)
SINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 127	16#0000 ～ 16#0127(BCD)
	DWORD		16#0000_0000 ～ 16#0000_0127(BCD)
	LWORD		16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0127(BCD)
INT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 32767	16#0000_0000 ～ 16#0003_2767(BCD)
	LWORD		16#0000_0000_0000_0000 ～ 16#0000_0000_0003_2767(BCD)
DINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 99999999	16#0000_0000 ～ 16#9999_9999(BCD)
	LWORD	0 ～ 2147483647	16#0000_0000_0000_0000 ～ 16#0000_0021_4748_3647(BCD)
LINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
	DWORD	0 ～ 99999999	16#0000_0000 ～ 16#9999_9999(BCD)
	LWORD	0 ～ 9999999999999999	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)

参考

- 特定の BCD ビット列から整数への変換は、「**_BCD_TO_*** (P.2-242)」命令を使用してください。
- 任意の BCD ビット列から整数への変換は、「BCD_TO_** (P.2-248)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、有効範囲外の場合。

BCD_TO_**

BCD のビット列を、符号なし整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BCD_TO_**	BCD 群→符号なし整数変換グループ	FUN	(@)BCD_TO_** EN ENO In Out ただし、“**”は整数のデータ型名	Out:=BCD_TO_**(In); ただし、“**”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	–	*2
Out	変換結果	出力	変換結果	*1	–	–

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-249)」を参照してください。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

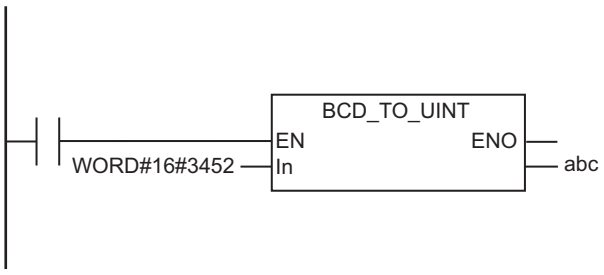
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○	○	○	○	○	○	○	○							

機能

BCD のビット列である変換対象「In」を、符号なし整数に変換します。
命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が UINT 型の場合、命令名称は BCD_TO_UINT となります。

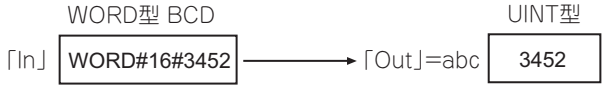
BCD_TO_UINT 命令で、「In」=WORD#16#3452 の場合の例を、以下に示します。

LD



ST

```
abc:=BCD_TO_UINT(WORD#16#3452);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
BYTE	USINT	16#00 ～ 16#99(BCD)	0 ～ 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 ～ 16#0255(BCD)	0 ～ 255
	UINT	16#0000 ～ 16#9999(BCD)	0 ～ 9999
	UDINT		
	ULINT		
	SINT	16#0000 ～ 16#0127(BCD)	0 ～ 127
	INT	16#0000 ～ 16#9999(BCD)	0 ～ 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 ～ 16#0000_0255(BCD)	0 ～ 255
	UINT	16#0000_0000 ～ 16#0006_5535(BCD)	0 ～ 65535
	UDINT	16#0000_0000 ～ 16#9999_9999(BCD)	0 ～ 99999999
	ULINT		
	SINT	16#0000_0000 ～ 16#0000_0127(BCD)	0 ～ 127
	INT	16#0000_0000 ～ 16#0003_2767(BCD)	0 ～ 32767
	DINT	16#0000_0000 ～ 16#9999_9999(BCD)	0 ～ 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0255(BCD)	0 ～ 255
	UINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0006_5535(BCD)	0 ～ 65535
	UDINT	16#0000_0000_0000_0000 ～ 16#0000_0042_9496_7295(BCD)	0 ～ 4294967295
	ULINT	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)	0 ～ 9999999999999999
	SINT	16#0000_0000_0000_0000 ～ 16#0000_0000_0000_0127(BCD)	0 ～ 127
	INT	16#0000_0000_0000_0000 ～ 16#0000_0000_0003_2767(BCD)	0 ～ 32767
	DINT	16#0000_0000_0000_0000 ～ 16#0000_0021_4748_3647(BCD)	0 ～ 2147483647
	LINT	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)	0 ～ 9999999999999999

参考

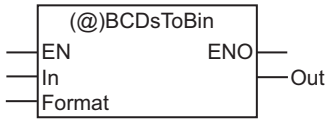
- 特定の BCD ビット列から整数への変換は、「**_BCD_TO_*** (P.2-242)」命令を使用してください。
- 整数から BCD ビット列への変換は、「**_TO_BCD_*** (P.2-245)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- 「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「In」の値が、有効範囲外の場合。
  b）「In」の値が、BCD のビット列でない（16 進数表記で A,B,C,D,E,F のいずれかが含まれる）場合。

BCDsToBin

符号付き BCD のビット列を、符号付き整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BCDsToBin	符号付き BCD →符号付き整数変換	FUN		Out:=BCDsToBin(In, Format);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	*2
Format	データ形式指定番号		BCD のビット列の形式	_BCD0～_BCD3		_BCD0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「Format」の値によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-252)」を参照してください。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Format	列挙型_eBCD_FORMAT　列挙子は「機能 (P.2-251)」参照																			
Out	「In」と同じサイズの、符号付き整数のデータ型																			

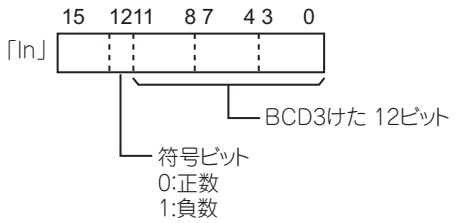
機能

符号付き BCD のビット列「In」を、符号付き整数に変換します。

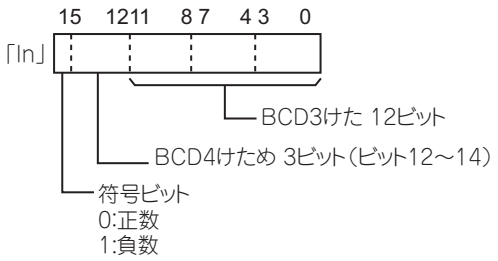
データ形式指定番号「Format」のデータ型は、列挙型_eBCD_FORMAT です。
列挙子は、_BCD0、_BCD1、_BCD2、_BCD3 の 4 つです。この値によって、「In」の最上位 4 ビットの符号表現方式が変わります。

「In」が WORD 型の場合を例にとって、データ形式を以下に示します。

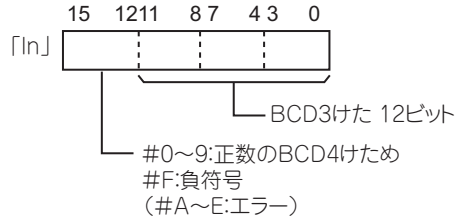
「Format」=_BCD0の場合
「In」の有効範囲:-999～999(BCD)



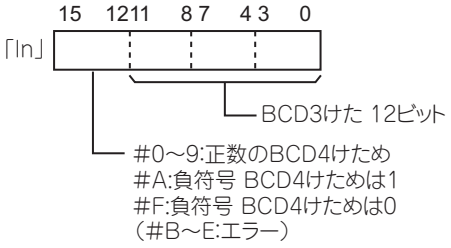
「Format」=_BCD1の場合
「In」の有効範囲:-7999～7999(BCD)



「Format」=_BCD2の場合
「In」の有効範囲:-999～9999(BCD)



「Format」=_BCD3の場合
「In」の有効範囲:-1999～9999(BCD)



有効範囲

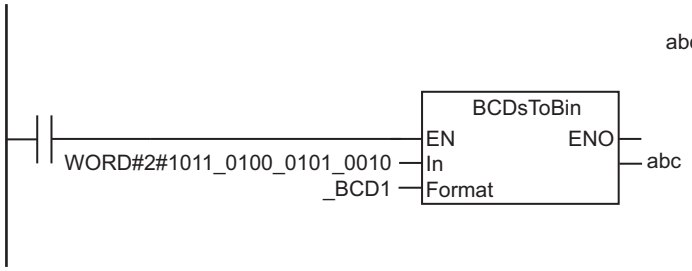
「In」と「Out」データ型は、同じサイズにします。それぞれの有効範囲は、「Format」の値によって、以下のように変わります。

		「Format」の値			
		_BCD0	_BCD1	_BCD2	_BCD3
「In」のデータ型 ↓ 「Out」のデータ型	BYTE ↓ SINT	-9 ～ 9	-79 ～ 79	-9 ～ 99	-19 ～ 99
	WORD ↓ INT	-999 ～ 999	-7999 ～ 7999	-999 ～ 9999	-1999 ～ 9999
	DWORD ↓ DINT	-9999999 ～ 9999999	-79999999 ～ 79999999	-9999999 ～ 99999999	-19999999 ～ 99999999
	LWORD ↓ LINT	-999999999999999 ～ 999999999999999	-7999999999999999 ～ 7999999999999999	-9999999999999999 ～ 9999999999999999	-19999999999999999 ～ 9999999999999999

記述例

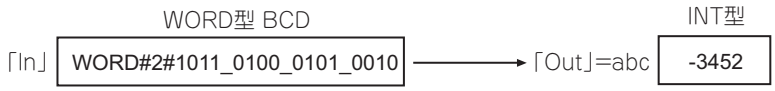
「In」=WORD#2#1011_0100_0101_0010、「Format」=_BCD1 の場合の例を、以下に示します。

LD



ST

```
abc:=BCDsToBin(WORD#2#1011_0100_0101_0010,
               _BCD1);
```

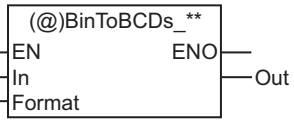


使用上の注意

- 「In」と「Out」のデータ型のサイズは、同じにしてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Format」の値が_BCD0 で、「In」の最上位けたが 2～F の場合。
  b)「Format」の値が_BCD2 で、「In」の最上位けたが A～E の場合。
  c)「Format」の値が_BCD3 で、「In」の最上位けたが B～E の場合。
  d) 上記条件を除いて、「In」のいずれかのけたが A～F の場合。
  e)「Format」の値が、有効範囲外の場合。

BinToBCDs_**

符号付き整数を、符号付き BCD のビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BinToBCDs_**	符号付き整数→BCD 変換グループ	FUN	 ただし、“**”はビット列のデータ型名	Out:=BinToBCDs_**(In, Format); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	0
Format	データ形式指定番号		BCD のビット列の形式	_BCD0～_BCD3		_BCD0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「Format」の値によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-255)」を参照してください。

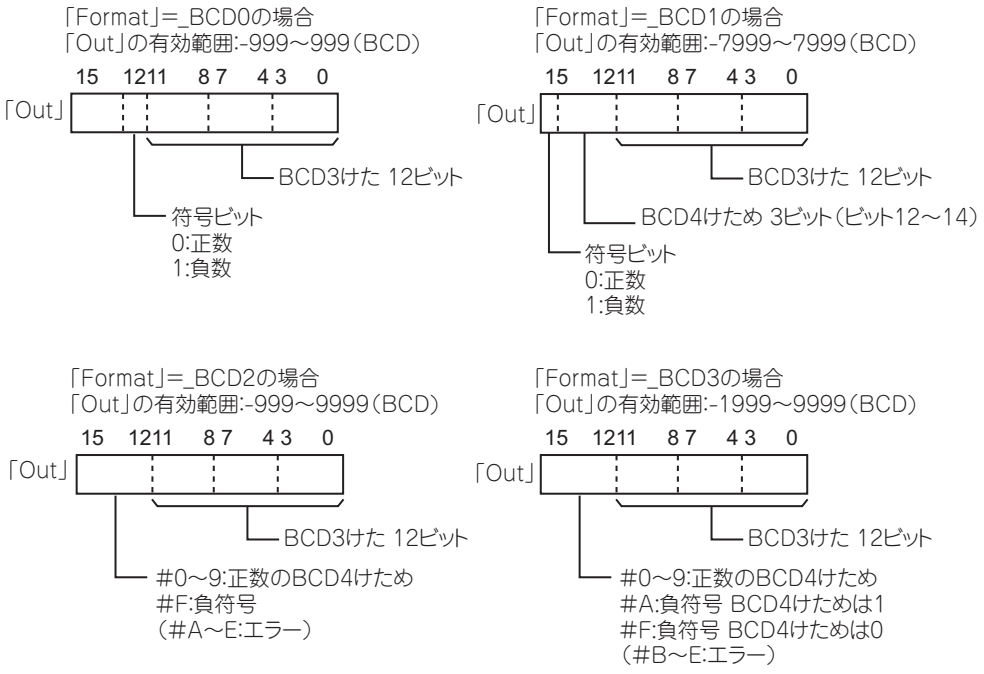
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										○	○	○	○							
Format	列挙型_eBCD_FORMAT　列挙子は「機能 (P.2-254)」参照																			
Out	「In」と同じサイズの、ビット列のデータ型																			

機能

符号付き整数「In」を、符号付き BCD のビット列に変換します。
命令名称は、「Out」のデータ型によって、異なります。たとえば、「Out」が WORD 型の場合、命令名称は BinToBCDs_WORD となります。

データ形式指定番号「Format」のデータ型は、列挙型_eBCD_FORMAT です。
列挙子は、_BCD0、_BCD1、_BCD2、_BCD3 の 4 つです。この値によって、「Out」の最上位 4 ビットの符号表現方式が変わります。

「Out」が WORD 型の場合を例にとって、データ形式を以下に示します。



有効範囲

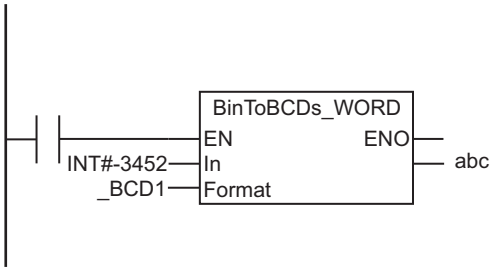
「In」と「Out」のデータ型は、同じサイズにします。それぞれの有効範囲は、「Format」の値によって、以下のように変わります。

		「Format」の値			
		_BCD0	_BCD1	_BCD2	_BCD3
「In」のデータ型 ↓ 「Out」のデータ型	SINT ↓ BYTE	-9 ～ 9	-79 ～ 79	-9 ～ 99	-19 ～ 99
	INT ↓ WORD	-999 ～ 999	-7999 ～ 7999	-999 ～ 9999	-1999 ～ 9999
	DINT ↓ DWORD	-9999999 ～ 9999999	-79999999 ～ 79999999	-9999999 ～ 99999999	-19999999 ～ 99999999
	LINT ↓ LWORD	-999999999999999 ～ 999999999999999	-7999999999999999 ～ 7999999999999999	-9999999999999999 ～ 9999999999999999	-19999999999999999 ～ 9999999999999999

記述例

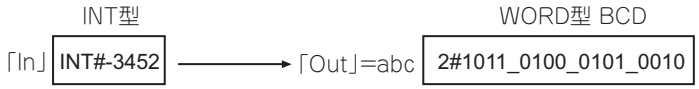
BinToBCDs_WORD 命令で、「In」=INT#-3452、「Format」=_BCD1 の場合の例を、以下に示します。

LD



ST

```
abc:=BinToBCDs_WORD(INT#-3452, _BCD1);
```

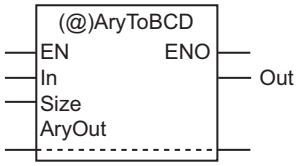


使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、有効範囲外の場合。
  b)「Format」の値が、有効範囲外の場合。

AryToBCD

符号なし整数配列の各要素を、BCD のビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryToBCD	配列一括 BCD 変換	FUN		AryToBCD(In, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	符号なし整数配列	入力	符号なし整数配列	*1	－	*2
Size	要素数		変換する In[]の要素数	データ型に従う		1
AryOut[] 配列	BCD 配列	入出力	BCD 配列	*1	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. In[]、AryOut[]の要素のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-258)」を参照してください。

*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

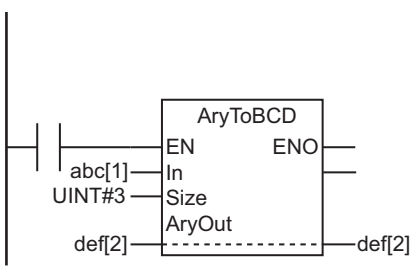
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列						○	○	○	○											
Size							○													
AryOut[] 配列	ビット列の配列。ただし、In[]の要素と同じサイズのデータ型。																			
Out	○																			

機能

符号なし整数配列 In[]の、In[0]からの「Size」個の各要素を BCD のビット列に変換し、BCD 配列 AryOut[]に出力します。

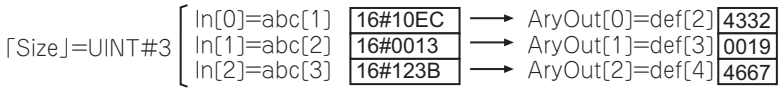
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



有効範囲

In[]と AryOut[]の有効範囲は、要素のデータ型の組み合わせで以下のように異なります。

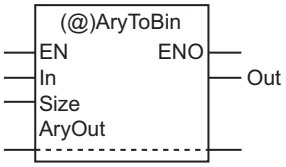
In[]の要素のデータ型	AryOut[]の要素のデータ型	In[]の有効範囲	AryOut[]の有効範囲
USINT	BYTE	0 ～ 99	16#00 ～ 16#99(BCD)
UINT	WORD	0 ～ 9999	16#0000 ～ 16#9999(BCD)
UDINT	DWORD	0 ～ 99999999	16#0000_0000 ～ 16#9999_9999(BCD)
ULINT	LWORD	0 ～ 9999999999999999	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)

使用上の注意

- In[]の要素と AryOut[]の要素のデータ型は、同じサイズのものにしてください。たとえば、In[]の要素が UINT 型の場合は、AryOut[]の要素は WORD 型にしてください。同じサイズのものでないと、ビルド時に異常となります。
- 符号付き BCD 変換はできません。したがって、In[]のデータ型は、符号なし整数(USINT, UINT, UDINT, ULINT)のいずれかにしてください。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）In[]の値が、有効範囲外の場合。
  b）「Size」の値が、In[]または AryOut[]の配列領域を超える場合。

AryToBin

BCD のビット列配列の各要素を、符号なし整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryToBin	配列一括符号なし整数変換	FUN		AryToBin(In, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	BCD のビット列配列	入力	BCD のビット列配列	*1	−	*2
Size	要素数		変換する In[]の要素数	データ型に従う		1
AryOut[] 配列	符号なし整数配列	入出力	符号なし整数配列	*1	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. In[]、AryOut[]の要素のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-260)」を参照してください。

*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

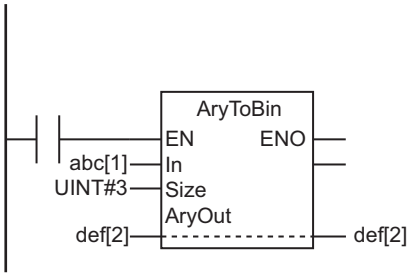
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○	○	○	○															
Size							○													
AryOut[] 配列	符号なし整数の配列。ただし、In[]の要素と同じサイズのデータ型。																			
Out	○																			

機能

BCD のビット列配列 In[]の、In[0]からの「Size」個の各要素を符号なし整数に変換し、符号なし整数配列 AryOut[]に出力します。

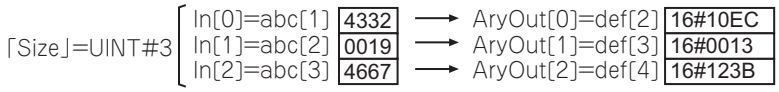
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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



有効範囲

In[]と AryOut[]の有効範囲は、要素のデータ型の組み合わせで以下のように異なります。

In[]の要素のデータ型	AryOut[]の要素のデータ型	In[]の有効範囲	AryOut[]の有効範囲
BYTE	USINT	16#00 ～ 16#99(BCD)	0 ～ 99
WORD	UINT	16#0000 ～ 16#9999(BCD)	0 ～ 9999
DWORD	UDINT	16#0000_0000 ～ 16#9999_9999(BCD)	0 ～ 99999999
LWORD	ULINT	16#0000_0000_0000_0000 ～ 16#9999_9999_9999_9999(BCD)	0 ～ 9999999999999999

使用上の注意

- In[]と AryOut[]のデータ型は、同じサイズのものにしてください。たとえば、In[]の要素が WORD 型の場合は、AryOut[]の要素は UINT 型にしてください。同じサイズのものでないと、ビルド時に異常となります。
- 符号付き BCD 変換はできません。したがって、AryOut[]のデータ型は、符号なし整数(USINT, UINT, UDINT, ULINT)のいずれかにしてください。
- 「Size」の値が 0 の場合、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）「Size」の値が、In[]または AryOut[]の配列領域を超える場合。
  b）In[]の値が、BCD のビット列ではない（16 進数表記で、A,B,C,D,E,F のいずれかが含まれる）場合。

データ型変換命令

命令	名称	ページ
_TO_*（整数→整数変換グループ）	整数→整数変換グループ	P.2-262
_TO_*（整数→ビット列変換グループ）	整数→ビット列変換グループ	P.2-265
_TO_*（整数→実数変換グループ）	整数→実数変換グループ	P.2-268
_TO_*（ビット列→整数変換グループ）	ビット列→整数変換グループ	P.2-270
_TO_*（ビット列→ビット列変換グループ）	ビット列→ビット列変換グループ	P.2-273
_TO_*（ビット列→実数変換グループ）	ビット列→実数変換グループ	P.2-275
_TO_*（実数→整数変換グループ）	実数→整数変換グループ	P.2-277
_TO_*（実数→ビット列変換グループ）	実数→ビット列変換グループ	P.2-280
_TO_*（実数→実数変換グループ）	実数→実数変換グループ	P.2-282
**_TO_STRING（整数→文字列変換グループ）	整数→文字列変換グループ	P.2-284
**_TO_STRING（ビット列→文字列変換グループ）	ビット列→文字列変換グループ	P.2-286
**_TO_STRING（実数→文字列変換グループ）	実数→文字列変換グループ	P.2-288
RealToFormatString	実数（REAL）→文字列変換書式付き	P.2-290
LrealToFormatString	実数（LREAL）→文字列変換書式付き	P.2-296
STRING_TO_**（文字列→整数変換グループ）	文字列→整数変換グループ	P.2-302
STRING_TO_**（文字列→ビット列変換グループ）	文字列→ビット列変換グループ	P.2-304
STRING_TO_**（文字列→実数変換グループ）	文字列→実数変換グループ	P.2-306
TO_**（整数変換グループ）	整数変換グループ	P.2-309
TO_**（ビット列変換グループ）	ビット列変換グループ	P.2-311
TO_**（実数変換グループ）	実数変換グループ	P.2-313
EnumToNum	列挙型→整数変換	P.2-315
NumToEnum	整数→列挙型変換	P.2-317
TRUNC / Round / RoundUp	実数切り捨て／実数丸め／実数切り上げ	P.2-319

_TO_*（整数→整数変換グループ）

整数を、異なるデータ型の整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	整数→整数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”、“***”は異なる整数のデータ型名	Out:=**_TO_***(In); ただし、“**”、“***”は異なる整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-263)」を参照してください。

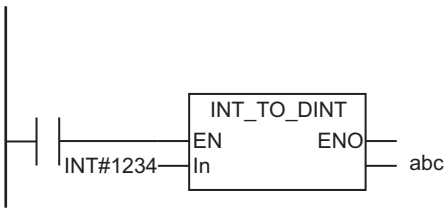
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Out						○	○	○	○	○	○	○	○							

機能

整数「In」を、異なるデータ型の整数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が INT 型で「Out」が DINT 型の場合、命令名称は INT_TO_DINT となります。

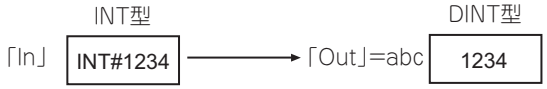
INT_TO_DINT 命令の場合で、「In」=INT#1234 の例を、以下に示します。

LD



ST

```
abc:=INT_TO_DINT(INT#1234);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」、「Out」の有効範囲
USINT	UINT	0 ～ 255
	UDINT	
	ULINT	
	SINT	0 ～ 127
	INT	0 ～ 255
	DINT	
	LINT	
UINT	USINT	0 ～ 255
	UDINT	0 ～ 65535
	ULINT	
	SINT	0 ～ 127
	INT	0 ～ 32767
	DINT	0 ～ 65535
	LINT	
UDINT	USINT	0 ～ 255
	UINT	0 ～ 65535
	ULINT	0 ～ 4294967295
	SINT	0 ～ 127
	INT	0 ～ 32767
	DINT	0 ～ 2147483647
	LINT	0 ～ 4294967295
ULINT	USINT	0 ～ 255
	UINT	0 ～ 65535
	UDINT	0 ～ 4294967295
	SINT	0 ～ 127
	INT	0 ～ 32767
	DINT	0 ～ 2147483647
	LINT	0 ～ 9223372036854775807
SINT	USINT	0 ～ 127
	UINT	
	UDINT	
	ULINT	
	INT	-128 ～ 127
	DINT	
	LINT	
INT	USINT	0 ～ 255
	UINT	0 ～ 32767
	UDINT	
	ULINT	
	SINT	-128 ～ 127
	DINT	-32768 ～ 32767
	LINT	

「In」のデータ型	「Out」のデータ型	「In」、「Out」の有効範囲
DINT	USINT	0 ～ 255
	UINT	0 ～ 65535
	UDINT	0 ～ 2147483647
	ULINT	
	SINT	-128 ～ 127
	INT	-32768 ～ 32767
	LINT	-2147483648 ～ 2147483647
LINT	USINT	0 ～ 255
	UINT	0 ～ 65535
	UDINT	0 ～ 4294967295
	ULINT	0 ～ 9223372036854775807
	SINT	-128 ～ 127
	INT	-32768 ～ 32767
	DINT	-2147483648 ～ 2147483647

参考

任意のデータ型から整数型への変換は、「TO_**（整数変換グループ）（P.2-309）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」が符号付きの整数で、「In」のデータサイズより「Out」のデータサイズの方が大きい場合、符号拡張を行います。
- 「In」が符号なしの整数で、「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- 「In」のデータサイズより「Out」のデータサイズの方が小さい場合、上位けたが切り捨てられます。

_TO_*（整数→ビット列変換グループ）

整数を、ビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	整数→ビット列変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”は整数のデータ型名、“***”はビット列のデータ型名	Out:=**_TO_***(In); ただし、“**”は整数のデータ型名、“***”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

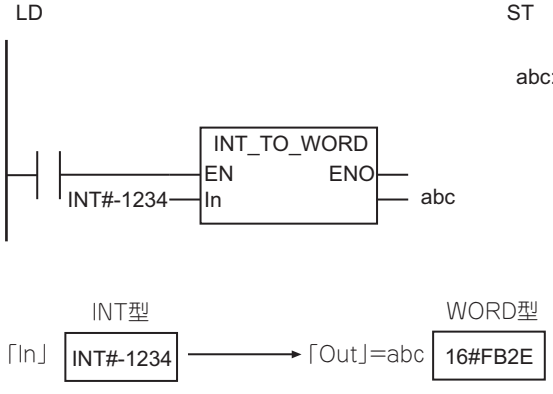
*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-266)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Out		○	○	○	○															

機能

整数「In」を、ビット列に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が INT 型、「Out」が WORD 型の場合、命令名称は INT_TO_WORD となります。

INT_TO_WORD 命令で、「In」=INT#-1234 の場合の例を、以下に示します。



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
USINT	BYTE	0 ～ 255	16#00 ～ 16#FF
	WORD		
	DWORD		
	LWORD		
UINT	BYTE	0 ～ 255	16#00 ～ 16#FF
	WORD	0 ～ 65535	16#0000 ～ 16#FFFF
	DWORD		
	LWORD		
UDINT	BYTE	0 ～ 255	16#00 ～ 16#FF
	WORD	0 ～ 65535	16#0000 ～ 16#FFFF
	DWORD	0 ～ 4294967295	16#0000_0000 ～ 16#FFFF_FFFF
	LWORD		
ULINT	BYTE	0 ～ 255	16#00 ～ 16#FF
	WORD	0 ～ 65535	16#0000 ～ 16#FFFF
	DWORD	0 ～ 4294967295	16#0000_0000 ～ 16#FFFF_FFFF
	LWORD	0 ～ 18446744073709551645	16#0000_0000_0000_0000 ～ 16#FFFF_FFFF_FFFF_FFFF
SINT	BYTE	-128 ～ 127	16#00 ～ 16#FF
	WORD		
	DWORD		
	LWORD		
INT	BYTE	-128 ～ 127	16#00 ～ 16#FF
	WORD	-32768 ～ 32767	16#0000 ～ 16#FFFF
	DWORD		
	LWORD		
DINT	BYTE	-128 ～ 127	16#00 ～16#FF
	WORD	-32768 ～ 32767	16#0000 ～ 16#FFFF
	DWORD	-2147483648 ～ 2147483647	16#0000_0000 ～ 16#FFFF_FFFF
	LWORD		
LINT	BYTE	-128 ～ 127	16#00 ～ 16#FF
	WORD	-32768 ～ 32767	16#0000 ～ 16#FFFF
	DWORD	-2147483648 ～ 2147483647	16#0000_0000 ～ 16#FFFF_FFFF
	LWORD	-9223372036854775808 ～ 9223372036854775807	16#0000_0000_0000_0000 ～ 16#FFFF_FFFF_FFFF_FFFF

参考

- ビット列から整数への変換は、「**_TO_***（ビット列→整数変換グループ）（P.2-270)」命令を使用してください。
- 任意のデータ型からビット列への変換は、「TO_**（ビット列変換グループ）（P.2-311)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。

- 「In」が符号付きの整数で、「In」のデータサイズより「Out」のデータサイズの方が大きい場合、符号拡張を行います。
- 「In」が符号なしの整数で、「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- 「In」のデータサイズより「Out」のデータサイズの方が小さい場合、上位けたが切り捨てられます。

_TO_*（整数→実数変換グループ）

整数を、実数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	整数→実数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”は整数のデータ型名、“***”は実数のデータ型名	Out:=**_TO_***(In); ただし、“**”は整数のデータ型名、“***”は実数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-269)」を参照してください。

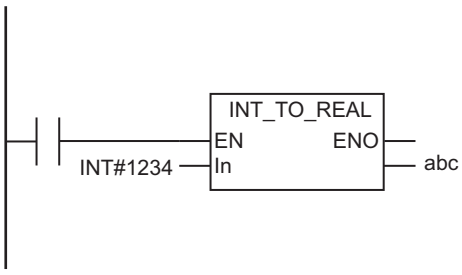
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Out														○	○					

機能

整数「In」を、実数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が INT 型、「Out」が REAL 型の場合、命令名称は INT_TO_REAL となります。

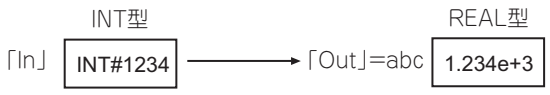
INT_TO_REAL 命令で、「In」=INT#1234 の場合の例を、以下に示します。

LD



ST

```
abc:=INT_TO_REAL(INT#1234);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
USINT	REAL	0 ～ 255	0 ～ 2.55e+2
	LREAL		
UINT	REAL	0 ～ 65535	0 ～ 6.5535e+4
	LREAL		
UDINT	REAL	0 ～ 4294967295	0 ～ 4.294967e+9
	LREAL		0 ～ 4.294967295e+9
ULINT	REAL	0 ～ 18446744073709551615	0 ～ 1.844674e+19
	LREAL		0 ～ 1.84467440737095e+19
SINT	REAL	-128 ～ 127	-1.28e+2 ～ 1.27e+2
	LREAL		
INT	REAL	-32768 ～ 32767	-3.2768e+4 ～ 3.2767e+4
	LREAL		
DINT	REAL	-2147483648 ～ 2147483647	-2.147483e+9 ～ 2.147483e+9
	LREAL		-2.147483648e+9 ～ 2.147483647e+9
LINT	REAL	-9223372036854775808 ～ 9223372036854775807	-9.223372e+18 ～ 9.223372e+18
	LREAL		-9.22337203685477e+18 ～ 9.22337203685477e+18

参考

- 実数から整数への変換は、「**_TO_***（実数→整数変換グループ）（P.2-277）」命令を使用してください。
- 任意のデータ型から実数への変換は、「TO_**（実数変換グループ）（P.2-313）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」と「Out」のデータ型の組み合わせによって、実数の有効けたに応じた丸めが発生し、変換前後の値に差異が生じることがあります。差異の発生するデータ型の組み合わせは、つぎのとおりです。

「In」のデータ型	「Out」のデータ型	差異の発生する値
DINT	REAL	-16777216 以下または 16777216 以上
LINT		
UDINT	REAL	16777216 以上
ULINT		
LINT	LREAL	-9007199254740992 以下または 9007199254740992 以上
ULINT	LREAL	9007199254740992 以上

_TO_*（ビット列→整数変換グループ）

ビット列を、整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	ビット列→整数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”はビット列のデータ型名、“***”は整数のデータ型名	Out:=**_TO_***(In); ただし、“**”はビット列のデータ型名、“***”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-271)」を参照してください。

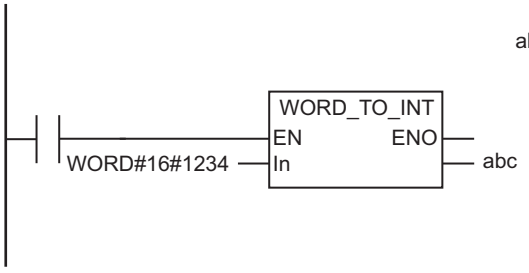
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○	○	○	○	○	○	○	○							

機能

ビット列「In」を、整数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が WORD 型、「Out」が INT 型の場合、命令名称は WORD_TO_INT となります。

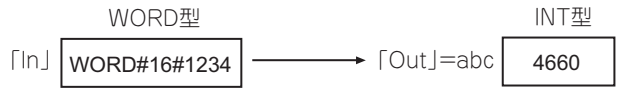
WORD_TO_INT 命令で、「In」=WORD#16#1234 の場合の例を、以下に示します。

LD



ST

```
abc:=WORD_TO_INT(WORD#16#1234);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
BYTE	USINT	16#00 ～ 16#FF	0 ～ 255
	UINT		
	UDINT		
	ULINT		
	SINT		-128 ～ 127
	INT		
	DINT		
	LINT		
WORD	USINT	16#00 ～ 16#FF	0 ～ 255
	UINT	16#0000 ～ 16#FFFF	0 ～ 65535
	UDINT		
	ULINT		
	SINT	16#00 ～ 16#FF	-128 ～ 127
	INT	16#0000 ～ 16#FFFF	-32768 ～ 32767
	DINT		
	LINT		
DWORD	USINT	16#00 ～ 16#FF	0 ～ 255
	UINT	16#0000 ～ 16#FFFF	0 ～ 65535
	UDINT	16#0000_0000 ～ 16#FFFF_FFFF	0 ～ 4294967295
	ULINT		
	SINT	16#00 ～ 16#FF	-128 ～ 127
	INT	16#0000 ～ 16#FFFF	-32768 ～ 32767
	DINT	16#0000_0000 ～ 16#FFFF_FFFF	-2147483648 ～ 2147483647
	LINT		
LWORD	USINT	16#00 ～ 16#FF	0 ～ 255
	UINT	16#0000 ～ 16#FFFF	0 ～ 65535
	UDINT	16#0000_0000 ～ 16#FFFF_FFFF	0 ～ 4294967295
	ULINT	16#0000_0000_0000_0000 ～ 16#FFFF_FFFF_FFFF_FFFF	0 ～ 18446744073709551645
	SINT	16#00 ～ 16#FF	-128 ～ 127
	INT	16#0000 ～ 16#FFFF	-32768 ～ 32767
	DINT	16#0000_0000 ～ 16#FFFF_FFFF	-2147483648 ～ 2147483647
	LINT	16#0000_0000_0000_0000 ～ 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808 ～ 9223372036854775807

参考

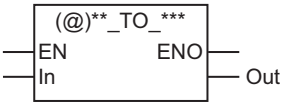
- 整数からビット列への変換は、「**_TO_***（整数→ビット列変換グループ）（P.2-265）」命令を使用してください。
- 任意のデータ型からビット列への変換は、「TO_**（ビット列変換グループ）（P.2-311）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- 「In」のデータサイズより「Out」のデータサイズの方が小さい場合、上位けたが切り捨てられます。

_TO_*（ビット列→ビット列変換グループ）

ビット列を、異なるデータ型のビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	ビット列→ビット列変換グループ	FUN	 ただし、“**”、“***”は異なるビット列のデータ型名	Out:=**_TO_***(In); ただし、“**”、“***”は異なるビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-274)」を参照してください。

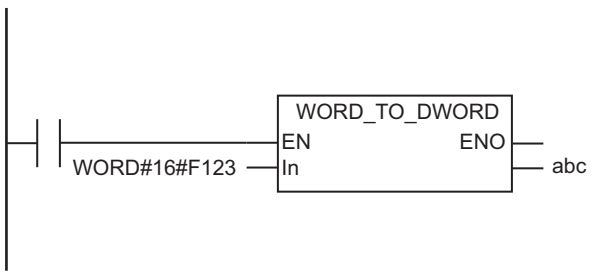
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out		○	○	○	○															

機能

ビット列「In」を、異なるデータ型のビット列に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が WORD 型で「Out」が DWORD 型の場合、命令名称は WORD_TO_DWORD となります。

WORD_TO_DWORD 命令で、「In」=WORD#16#F123 の場合の例を、以下に示します。

LD



ST

```
abc:=WORD_TO_DWORD(WORD#16#F123);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」、「Out」の有効範囲
BYTE	WORD	16#00 ～ 16#FF
	DWORD	
	LWORD	
WORD	BYTE	16#00 ～ 16#FF
	DWORD	16#0000 ～ 16#FFFF
	LWORD	
DWORD	BYTE	16#00 ～ 16#FF
	WORD	16#0000 ～ 16#FFFF
	LWORD	16#0000_0000 ～ 16#FFFF_FFFF
LWORD	BYTE	16#00 ～ 16#FF
	WORD	16#0000 ～ 16#FFFF
	DWORD	16#0000_0000 ～ 16#FFFF_FFFF

参考

任意のデータ型からビット列への変換は、「TO_**（ビット列変換グループ）（P.2-311）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
- 「In」のデータサイズより「Out」のデータサイズの方が小さい場合、上位けたが切り捨てられます。

_TO_*（ビット列→実数変換グループ）

ビット列を、実数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	ビット列→実数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”はビット列のデータ型名、“***”は実数のデータ型名	Out:=**_TO_***(In); ただし、“**”はビット列のデータ型名、“***”は実数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-276)」を参照してください。

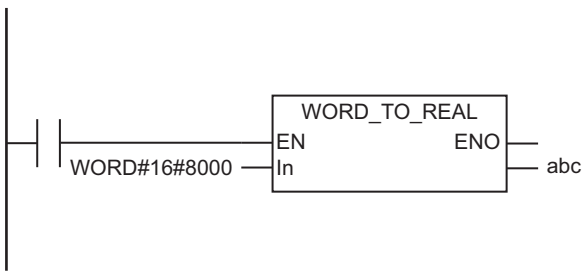
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out														○	○					

機能

ビット列「In」を、同じサイズの符号なし整数とみなして、実数型に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が WORD 型、「Out」が REAL 型の場合、命令名称は WORD_TO_REAL となります。

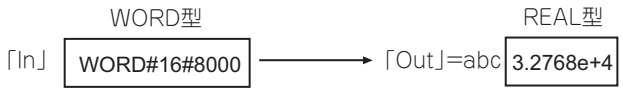
WORD_TO_REAL 命令で、「In」=WORD#16#8000 の場合の例を、以下に示します。

LD



ST

```
abc:=WORD_TO_REAL(WORD#16#8000);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
BYTE	REAL	16#00 ～ 16#FF	0 ～ 2.55e+2
	LREAL		
WORD	REAL	16#0000 ～ 16#FFFF	0 ～ 6.5535e+4
	LREAL		
DWORD	REAL	16#0000_0000 ～ 16#FFFF_FFFF	0 ～ 4.294967e+9
	LREAL		0 ～ 4.294967295e+9
LWORD	REAL	16#0000_0000_0000_0000 ～ 16#FFFF_FFFF_FFFF_FFFF	0 ～ 1.844674e+19
	LREAL		0 ～ 1.84467440737095e+19

参考

- 実数からビット列への変換は、「**_TO_***（実数→ビット列変換グループ）（P.2-280）」命令を使用してください。
- 任意のデータ型から実数への変換は、「TO_**（実数変換グループ）（P.2-313）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」と「Out」のデータ型の組み合わせによって、実数の有効けたに応じた丸めが発生し、変換前後の値に差異が生じることがあります。差異の発生するデータ型の組み合わせは、つぎのとおりです。

「In」のデータ型	「Out」のデータ型	差異の発生する値
DWORD	REAL	16#0100_0000 以上
LWORD	LREAL	16#0002_0000_0000_0000 以上

_TO_*（実数→整数変換グループ）

実数を、整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	実数→整数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”は実数のデータ型名、“***”は整数のデータ型名	Out:=**_TO_***(In); ただし、“**”は実数のデータ型名、“***”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-278)」を参照してください。

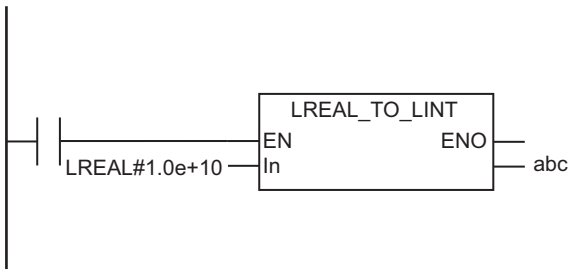
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out						○	○	○	○	○	○	○	○							

機能

実数「In」を、整数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が LREAL 型、「Out」が LINT 型の場合、命令名称は LREAL_TO_LINT となります。

LREAL_TO_LINT 命令で、「In」＝ LREAL#1.0e+10 の場合の例を、以下に示します。

LD



ST

```
abc:=LREAL_TO_LINT(LREAL#1.0e+10);
```



「In」の値の小数点以下

「In」の値の小数点以下は、五捨五入して整数に丸めます。五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1 -1.49→-1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2 -1.50→-2 -2.50→-2
0.5 より大	切り上げ	1.51→2 -1.51→-2

有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで以下のように異なります。

「In」のデータ型	「Out」のデータ型	「In」の有効範囲	「Out」の有効範囲
REAL	USINT	0 ～ 2.55e+2	0 ～ 255
	UINT	0 ～ 6.5535e+4	0 ～ 65535
	UDINT	0 ～ 4.294967e+9	0 ～ 4294967295
	ULINT	0 ～ 1.844674e+19	0 ～ 18446744073709551615
	SINT	-1.28e+2 ～ 1.27e+2	-128 ～ 127
	INT	-3.2768e+4 ～ 3.2767e+4	-32768 ～ 32767
	DINT	-2.147483e+9 ～ 2.147483e+9	-2147483648 ～ 2147483647
	LINT	-9.223372e+18 ～ 9.223372e+18	-9223372036854775808 ～ -9223372036854775807
LREAL	USINT	0 ～ 2.55e+2	0 ～ 255
	UINT	0 ～ 6.5535e+4	0 ～ 65535
	UDINT	0 ～ 4.294967295e+9	0 ～ 4294967295
	ULINT	0 ～ 1.84467440737095e+19	0 ～ 18446744073709551615
	SINT	-1.28e+2 ～ 1.27e+2	-128 ～ 127
	INT	-3.2768e+4 ～ 3.2767e+4	-32768 ～ 32767
	DINT	-2.147483648e+9 ～ 2.147483647e+9	-2147483648 ～ 2147483647
	LINT	-9.22337203685477e+18 ～ 9.22337203685477e+18	-9223372036854775808 ～ -9223372036854775807

参考

- 整数から実数への変換は、「**_TO_***（整数→実数変換グループ）（P.2-268）」命令を使用してください。
- 任意のデータ型から整数への変換は、「TO_**（整数変換グループ）（P.2-309）」命令を使用してください。
- 本命令以外に、実数→整数変換する命令には、TRUNC（小数点以下切り捨て）、Round（小数点以下五捨五入）、RoundUp（小数点以下切り上げ）があります。
  いずれも、入力が REAL 型なら出力は DINT 型、入力が LREAL 型なら出力は LINT 型です。
  各命令の処理の違いを、以下に示します。

入力値	出力値			
	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#1.4	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

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 変換結果が「Out」の有効範囲を超えると、「Out」の値は不定値になります。変換結果が「Out」の有効範囲を超えないように、「In」の値が有効範囲内に入っているか、必ずチェックしてください。

_TO_*（実数→ビット列変換グループ）

実数を、ビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	実数→ビット列変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”は実数のデータ型名、“***”はビット列のデータ型名	Out:=**_TO_***(In); ただし、“**”は実数のデータ型名、“***”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0
Out	変換結果	出力	変換結果	データ型に従う	−	−

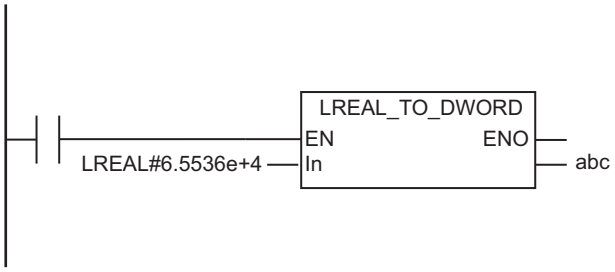
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out		○	○	○	○															

機能

実数「In」を、ビット列に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が LREAL 型、「Out」が DWORD 型の場合、命令名称は LREAL_TO_DWORD となります。

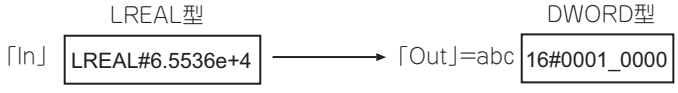
LREAL_TO_DWORD 命令で、「In」=LREAL#6.5536e+4 の場合の例を、以下に示します。

LD



ST

```
abc:=LREAL_TO_DWORD(LREAL#6.5536e+4);
```



変換例を、いくつか示します。

「In」の値	整数化した値	「Out」の値
1.6	2	16#0002
3.5	4	16#0004

変換の手順

変換の手順を、以下に示します。

1 **「In」の値の小数点以下を五捨五入（後述）して、整数に丸めます。**

2 **その整数を、符号なし整数としてビット列で表現します。**

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

参考

ビット列から実数への変換は、「**_TO_***（ビット列→実数変換グループ）（P.2-275)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 変換結果が「Out」の有効範囲を超えると、「Out」の値は不定値になります。変換結果が「Out」の有効範囲を超えないように、「In」の値が有効範囲内に入っているか、必ずチェックしてください。
- 負の値を入力すると、CPU ユニットの形式により変換結果が異なります。負の値を入力する場合は、十分にデバッグをしてから使用してください。

_TO_*（実数→実数変換グループ）

実数を、異なるデータ型の実数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
_TO_*	実数→実数変換グループ	FUN	(@)**_TO_*** EN ENO In Out ただし、“**”、“***”は異なる実数のデータ型名	Out:=**_TO_***(In); ただし、“**”、“***”は異なる実数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	−	0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-283)」を参照してください。

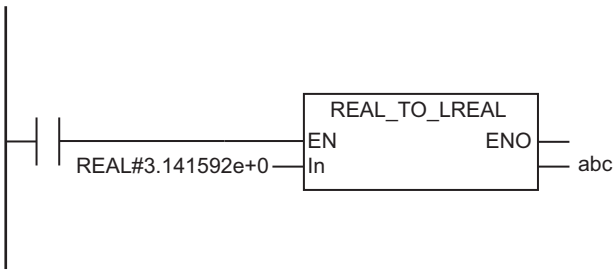
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out														○	○					

機能

実数「In」を、異なるデータ型の実数に変換します。
命令名称は、「In」と変換結果「Out」のデータ型の組み合わせによって、異なります。たとえば、「In」が REAL 型、「Out」が LREAL 型の場合、命令名称は REAL_TO_LREAL となります。

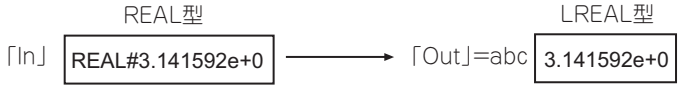
REAL_TO_LREAL 命令で、「In」=REAL#3.141592e+0 の場合の例を、以下に示します。

LD



ST

```
abc:=REAL_TO_LREAL(REAL#3.141592e+0);
```



有効範囲

「In」と「Out」の有効範囲は、以下のようになります。

「In」のデータ型	「Out」のデータ型	「In」、「Out」の有効範囲
REAL	LREAL	-3.402823e+38 ～ 3.402823e+38 または+∞/-∞
LREAL	REAL	

参考

任意のデータ型から実数への変換は、「TO_**（実数変換グループ）（P.2-313）」命令を使用してください。

使用上の注意

- 命令名称を、必ず「In」と「Out」のデータ型に合わせてください。
- 「In」の値が+∞/-∞の場合、「Out」の値は+∞/-∞となります。
- 「In」の値が非数の場合、「Out」の値は非数になります。
- 変換結果が「Out」の有効範囲外の場合、「Out」の値は、「In」の値と同じ符号の無限大になります。
- LREAL_TO_REAL 命令の場合、「In」の値が±1.175494e-38 より 0 に近いと、「Out」の値は 0 になります。

**_TO_STRING（整数→文字列変換グループ）

整数を、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
**_TO_STRING	整数→文字列変換グループ	FUN	(@)**_TO_STRING EN ENO In Out ただし、“**”は整数のデータ型名	Out:=**_TO_STRING(In); ただし、“**”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0
Out	変換結果	出力	変換結果	*1	−	−

*1. 「In」のデータ型によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-285)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Out																				○

機能

整数「In」を、文字列に変換します。
「In」を数字で表現したものが、文字列として、変換結果「Out」になります。「Out」の終端には、NULL 文字(16#00)を付けます。

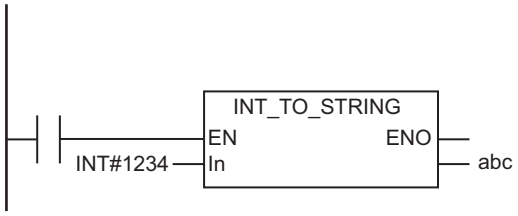
「Out」は、左詰めで行われます。

「In」の値が、「In」のデータ型の最大けた数に満たない場合は、上位けたの 0 は「Out」の文字列として出力しません。つまり、ゼロサプレスします。

「In」が負数の場合は、文字列の先頭にマイナス符号'-'を付けます。
命令名称は、「In」のデータ型によって、異なります。たとえば、「In」が INT 型の場合、命令名称は INT_TO_STRING となります。

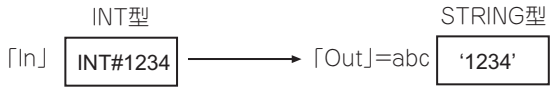
INT_TO_STRING 命令で、「In」=INT#1234 の場合の例を、以下に示します。

LD



ST

```
abc:=INT_TO_STRING(INT#1234);
```



有効範囲

「Out」の有効範囲は、「In」のデータ型によって、以下のように異なります。

「In」のデータ型	「Out」の有効範囲（最大バイト数）
USINT	4 バイト（半角英数 3 文字+終端 NULL 文字）
UINT	6 バイト（半角英数 5 文字+終端 NULL 文字）
UDINT	11 バイト（半角英数 10 文字+終端 NULL 文字）
ULINT	21 バイト（半角英数 20 文字+終端 NULL 文字）
SINT	5 バイト（半角英数 4 文字+終端 NULL 文字）
INT	7 バイト（半角英数 6 文字+終端 NULL 文字）
DINT	12 バイト（半角英数 11 文字+終端 NULL 文字）
LINT	21 バイト（半角英数 20 文字+終端 NULL 文字）

参考

文字列から整数への変換は、「STRING_TO_**（文字列→整数変換グループ）（P.2-302）」命令を使用してください。

使用上の注意

命令名称を、必ず「In」のデータ型に合わせてください。

**_TO_STRING（ビット列→文字列変換グループ）

ビット列を、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
**_TO_STRING	ビット列→文字列変換グループ	FUN	(@)**_TO_STRING EN ENO In Out ただし、“**”はビット列のデータ型名	Out:=**_TO_STRING(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	－	0
Out	変換結果	出力	変換結果	*1	－	－

*1. 「In」のデータ型によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-287)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out																				○

機能

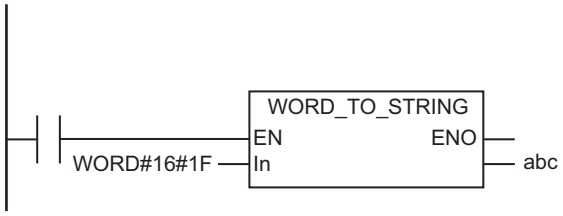
ビット列「In」を、文字列に変換します。
「In」を 16 進数として英数字で表現したものが、文字列として、変換結果「Out」になります。ただし、16 進数であることを示す接頭語“#16”は、「Out」に含まれません。また、「Out」の終端には、NULL 文字(16#00)を付けます。

「Out」は、左詰めで行われます。

「In」の値が、「In」のデータ型の最大けた数に満たない場合は、「Out」の上位けたを“0”で埋めます。つまり、ゼロパディングします。したがって、「Out」のバイト数は、終端の NULL 文字も含めて、常に（「In」のバイト数×2+1）に一致します。
命令名称は、「In」のデータ型によって、異なります。たとえば、「In」が WORD 型の場合、命令名称は WORD_TO_STRING となります。

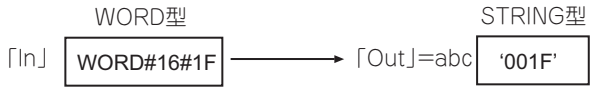
WORD_TO_STRING 命令で、「In」=WORD#16#1F の場合の例を、以下に示します。

LD



ST

```
abc:=WORD_TO_STRING(WORD#16#1F);
```



有効範囲

「Out」の有効範囲は、「In」のデータ型によって、以下のように異なります。

「In」のデータ型	「Out」の有効範囲（最大バイト数）
BYTE	3 バイト（半角英数 2 文字+終端 NULL 文字）
WORD	5 バイト（半角英数 4 文字+終端 NULL 文字）
DWORD	9 バイト（半角英数 8 文字+終端 NULL 文字）
LWORD	17 バイト（半角英数 16 文字+終端 NULL 文字）

参考

「In」を符号付きで文字列に変換したい場合は、まず「**_TO_***（ビット列→整数変換グループ）(P.2-270)」命令で符号付き整数に変換した後に、「**_TO_STRING（整数→文字列変換グループ）(P.2-284)」命令を使用してください。

使用上の注意

命令名称を、必ず「In」のデータ型に合わせてください。

**_TO_STRING（実数→文字列変換グループ）

実数を、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
**_TO_STRING	実数→文字列変換グループ	FUN	(@)**_TO_STRING EN　ENO In　Out ただし、“**”は実数のデータ型名	Out:=**_TO_STRING(In); ただし、“**”は実数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	－	0.0
Out	変換結果	出力	変換結果	*1	－	－

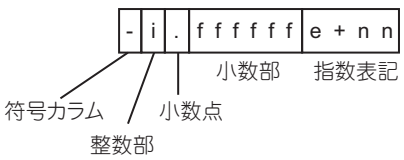
*1. 「In」のデータ型によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-289)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out																				○

機能

実数「In」を、文字列に変換します。
「In」を英数字の文字列で表現したものが、変換結果「Out」になります。

「Out」の構成は、以下のようになります。

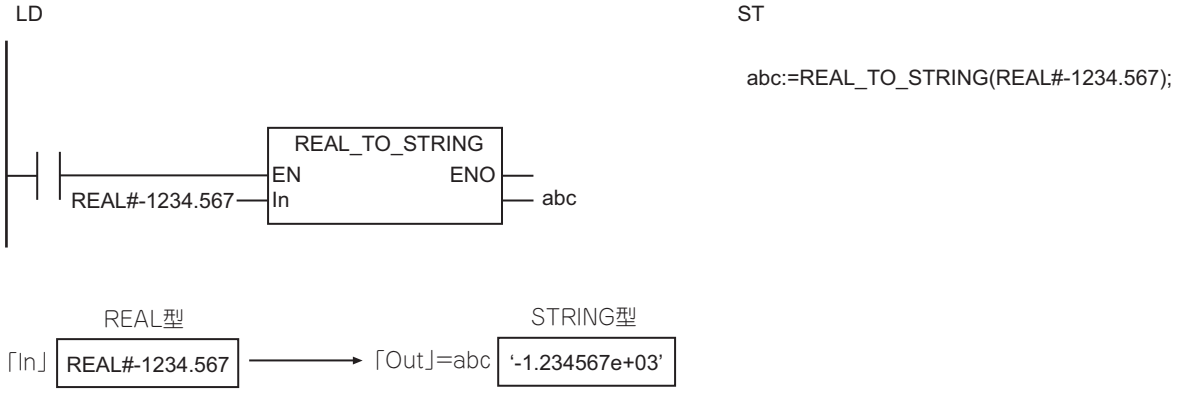


項目	内容
符号カラム	「In」が負数の場合だけ、'-'（マイナス）が付加されます。 「In」が正数の場合、'+'（プラス）は付加されません。
整数部	常に 1 けたで表記されます。
小数点	「In」が小数でなくても、常に表記されます。
小数部	「In」が REAL 型の場合 6 けた、LREAL 型の場合 14 けたで表記されます。
指数表記	常に表記されます。 “nn”のけた数は、2 または 3 です。 “nn”の符号は、「In」の絶対値が 1.0 以上の場合'+'（プラス）、1.0 未満の場合'-'（マイナス）です。

「Out」の終端には、NULL 文字（16#00）を付けます。

命令名称は、「In」のデータ型によって、異なります。たとえば、「In」が REAL 型の場合、命令名称は REAL_TO_STRING となります。

REAL_TO_STRING 命令で、「In」=REAL#-1234.567 の場合の例を、以下に示します。



有効範囲

「In」の値が、0、無限大、非数の場合、「Out」の値は、以下のようになります。

「In」の値	「Out」の値
0	'0'
+∞	'inf'
-∞	'-inf'
非数	'nan' または '-nan'

参考

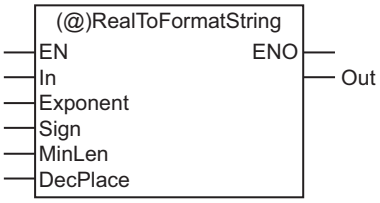
- 文字列から実数への変換は、「STRING_TO_**（文字列→実数変換グループ）（P.2-306)」命令を使用してください。
- 書式を指定して、実数から文字列に変換する場合は、「RealToFormatString (P.2-290)」命令、「LrealToFormatString (P.2-296)」命令を使用してください。

使用上の注意

命令名称を、必ず「In」のデータ型に合わせてください。

RealToFormatString

REAL 型変数を、書式指定付きで、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RealToFormatString	実数(REAL)→文字列変換書式付き	FUN		Out:=RealToFormatString(In, Exponent, Sign, MinLen, DecPlace);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0.0
Exponent	指数表記		TRUE: 指数表記あり FALSE: 指数表記なし			FALSE
Sign	符号カラム		TRUE: 符号カラムあり FALSE: 符号カラムなし			
MinLen	最小けた数		「Out」の最小けた数			6
DecPlace	精度		「Out」の小数点以下のけた数	0〜15		
Out	変換結果	出力	変換結果	最大 327 バイト（半角英数字 326 文字+終端 NULL 文字）	−	−

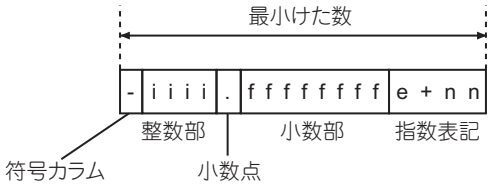
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○						
Exponent	○																			
Sign	○																			
MinLen						○														
DecPlace						○														
Out																				○

機能

REAL 型変数「In」を、文字列に変換します。
「In」を英数字の文字列で表現したものが、変換結果「Out」になります。「Out」の終端には、NULL 文字(16#00)を付けます。

「In」の値が負数の場合は先頭に'-'（マイナス）を入れますが、「In」の値が正数の場合は先頭に'+'（プラス）を入れません。

「Out」の形式は、指数表記「Exponent」、符号カラム「Sign」、最小けた数「MinLen」、精度「DecPlace」によって変わります。



入力変数	意味
Exponent	「Exponent」は、指数表記があるかどうかの指示です。 TRUE: 指数表記あり FALSE: 指数表記なし
Sign	「Sign」は、符号カラムがあるかどうかの指示です。 TRUE: 符号カラムあり FALSE: 符号カラムなし 符号カラムに入るのは、'-'（マイナス）だけです。正数で符号カラムありの場合、符号カラムには' '（空白文字）が入ります。 負数で符号カラムなしの場合は、整数部の先頭に'-'（マイナス）が付きます。 ただし、変換結果のけた数が、「MinLen」の値を超え、かつ、正数の場合、符号カラムに最上位けたの数字が入ります。
MinLen	「MinLen」は、符号カラム、整数部、小数点、小数部、指数表記を合わせた全体のけた数の最小値です。 変換結果のけた数が、「MinLen」の値未満の場合、符号カラムを除く文字列を右詰めに配置し、空いたけたには' '（空白文字）を埋めます。 変換結果のけた数が、「MinLen」の値を超える場合、文字列を左詰めにして、「MinLen」の値を超えるけた数の文字列を「Out」に代入します。
DecPlace	「DecPlace」は、小数部のけた数です。 「DecPlace」の値を超えるけたは五捨五入（後述）により丸められます。 また、「DecPlace」の値が「0」の場合は、小数部と共に小数点の表記もなくなります。

「In」の値が、無限大、非数の場合、「Out」の値は、以下のようになります。

「In」の値	「Out」の値
+∞	'inf'
-∞	'-inf'
非数	'nan' または '-nan'

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

入力変数と出力変数の関係例

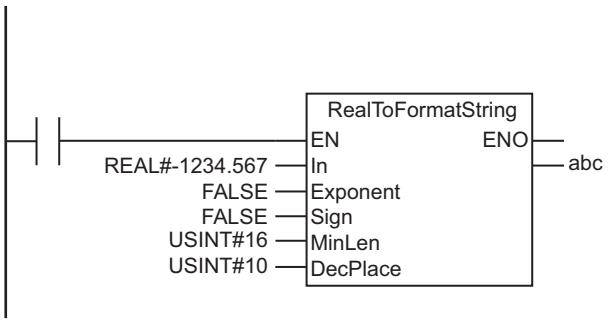
各入力変数の値と「Out」の値との関係例を示します。

● **例 1**

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	FALSE
MinLen	USINT#16
DecPlace	USINT#10

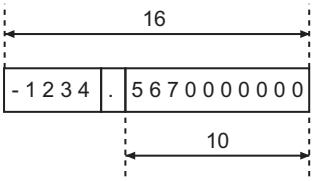
符号カラムなしの負数なので、整数部の先頭が、'-'（マイナス）になります。

LD



ST

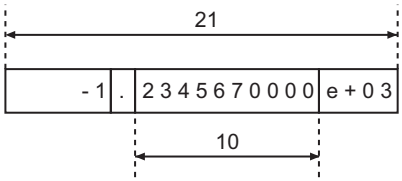
```
abc:=RealToFormatString(REAL#-1234.567, FALSE,
                        FALSE, USINT#16,
                        USINT#10);
```



● **例 2**

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	FALSE
MinLen	USINT#21
DecPlace	USINT#10

文字列のけた数より「MinLen」の方が大きいので、文字列を右詰めにし、先頭に' '（空白文字）を埋めます。

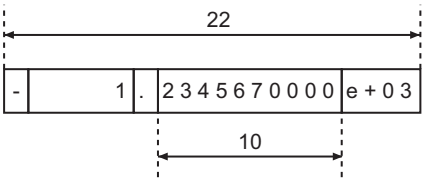


● **例 3**

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE

変数	設定内容
MinLen	USINT#22
DecPlace	USINT#10

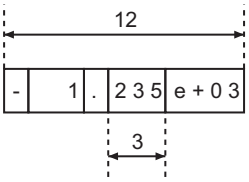
符号カラムは必ず左端です。整数部の先頭に、' '（空白文字）を埋めます。



● **例 4**

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#3

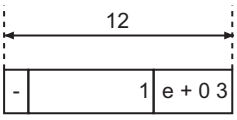
「DecPlace」=USINT#3 なので、小数第 4 位が五捨五入されます。



● **例 5**

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#0

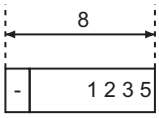
「DecPlace」=USINT#0 なので、小数第 1 位が五捨五入されます。小数点の表記もなくなります。



● **例 6**

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	TRUE
MinLen	USINT#8
DecPlace	USINT#0

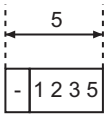
指数表記がなくなり、整数部が 4 けたになります。そのうえで、小数第 1 位が五捨五入されます。



● **例 7**

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	TRUE
MinLen	USINT#2
DecPlace	USINT#0

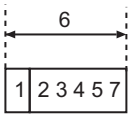
「MinLen」=USINT#2 より、「In」の整数部けた数 4 の方が大きいので、整数部は 4 けたで表記されます。



● **例 8**

変数	設定内容
In	REAL#123456.7
Exponent	FALSE
Sign	TRUE
MinLen	USINT#4
DecPlace	USINT#0

「MinLen」=USINT#4 より、「In」の整数部けた数 6 の方が大きいので、整数部は 6 けたで表記さ れます。また、「In」の値が正数なので、符号カラムに最上位けたの数字が入ります。



参考

- 「Exponent」、「Sign」、「MinLen」、「DecPlace」は、それぞれ省略可能です。省略した場合は、それぞれの初期値が適用されます。
- LREAL 型変数から文字列への変換は、「LrealToFormatString (P.2-296)」命令を使用してください。
- 文字列から実数への変換は、「STRING_TO_**（文字列→実数変換グループ） (P.2-306)」命令を使用してください。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

・「DecPlace」の値が、有効範囲外の場合。
・「DecPlace」の値が、「MinLen」の値より大きい場合。

LrealToFormatString

LREAL 型変数を、書式指定付きで、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LrealToFormatString	実数(LREAL)→文字列変換書式付き	FUN	(@)LrealToFormatString EN In Exponent Sign MinLen DecPlace ENO Out	Out:=LrealToFormatString(In, Exponent, Sign, MinLen, DecPlace);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	–	0.0
Exponent	指数表記		TRUE: 指数表記あり FALSE: 指数表記なし			FALSE
Sign	符号カラム		TRUE: 符号カラムあり FALSE: 符号カラムなし			
MinLen	最小けた数		「Out」の最小けた数			6
DecPlace	精度		「Out」の小数点以下のけた数	0～15		
Out	変換結果	出力	変換結果	最大 327 バイト（半角英数字 326 文字+終端 NULL 文字）	–	–

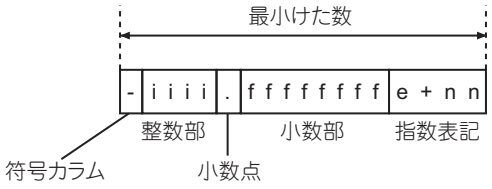
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In															○					
Exponent	○																			
Sign	○																			
MinLen						○														
DecPlace						○														
Out																				○

機能

LREAL 型変数「In」を、文字列に変換します。「In」を英数字の文字列で表現したものが、変換結果「Out」になります。「Out」の終端には、NULL 文字(16#00)を付けます。

「In」の値が負数の場合は先頭に'-'（マイナス）を入れますが、「In」の値が正数の場合は先頭に'+'（プラス）を入れません。

「Out」の形式は、指数表記「Exponent」、符号カラム「Sign」、最小けた数「MinLen」、精度「DecPlace」によって変わります。



入力変数	意味
Exponent	「Exponent」は、指数表記があるかどうかの指示です。 TRUE: 指数表記あり FALSE: 指数表記なし
Sign	「Sign」は、符号カラムがあるかどうかの指示です。 TRUE: 符号カラムあり FALSE: 符号カラムなし 符号カラムに入るのは、'-'（マイナス）だけです。正数で符号カラムありの場合、符号カラムには' '（空白文字）が入ります。 負数で符号カラムなしの場合は、整数部の先頭に'-'（マイナス）が付きます。 ただし、変換結果のけた数が、「MinLen」の値を超え、かつ、正数の場合、符号カラムに最上位けたの数字が入ります。
MinLen	「MinLen」は、符号カラム、整数部、小数点、小数部、指数表記を合わせた全体のけた数の最小値です。 変換結果のけた数が、「MinLen」の値未満の場合、符号カラムを除く文字列を右詰めに配置し、空いたけたには' '（空白文字）を埋めます。 変換結果のけた数が、「MinLen」の値を超える場合、文字列を左詰めにして、「MinLen」の値を超えるけた数の文字列を「Out」に代入します。
DecPlace	「DecPlace」は、小数部のけた数です。 「DecPlace」の値を超えるけたは五捨五入（後述）により丸められます。 また、「DecPlace」の値が 0 場合は、小数部と共に小数点の表記もなくなります。

「In」の値が、無限大、非数の場合、「Out」の値は、以下のようになります。

「In」の値	「Out」の値
+∞	'inf'
-∞	'-inf'
非数	'nan' または '-nan'

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

入力変数と出力変数の関係例

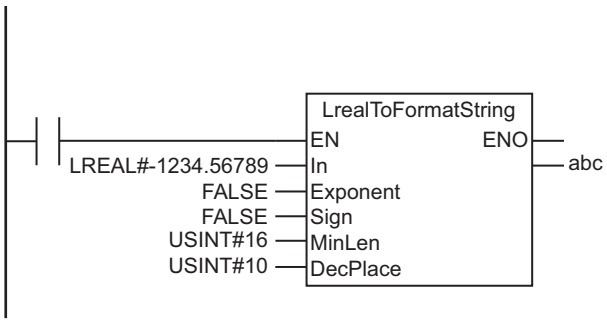
各入力変数の値と「Out」の値との関係例を示します。

● **例 1**

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	FALSE
MinLen	USINT#16
DecPlace	USINT#10

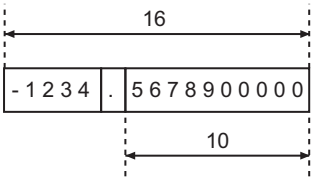
符号カラムなしの負数なので、整数部の先頭が、'-'（マイナス）になります。

LD



ST

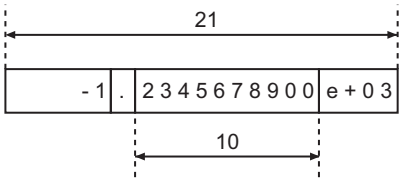
```
abc:=LrealToFormatString(LREAL#-1234.56789, FALSE,
                         FALSE, USINT#16,
                         USINT#10);
```



● **例 2**

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	FALSE
MinLen	USINT#21
DecPlace	USINT#10

文字列のけた数より「MinLen」の方が大きいので、文字列を右詰めにし、先頭に' '（空白文字）を埋めます。

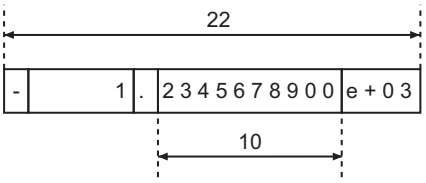


● **例 3**

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE

変数	設定内容
MinLen	USINT#22
DecPlace	USINT#10

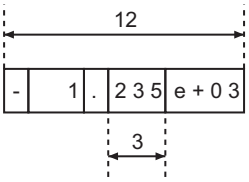
符号カラムは必ず左端です。整数部の先頭に、' '（空白文字）を埋めます。



● **例 4**

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#3

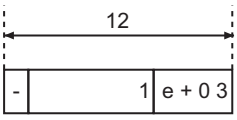
「DecPlace」=USINT#3 なので、小数第 4 位が五捨五入されます。



● **例 5**

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#0

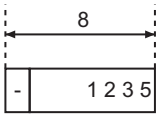
「DecPlace」=USINT#0 なので、小数第 1 位が五捨五入されます。小数点の表記もなくなります。



● **例 6**

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	TRUE
MinLen	USINT#8
DecPlace	USINT#0

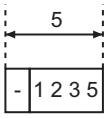
指数表記がなくなり、整数部が 4 けたになります。そのうえで、小数第 1 位が五捨五入されます。



● **例 7**

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	TRUE
MinLen	USINT#2
DecPlace	USINT#0

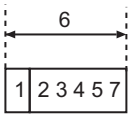
「MinLen」=USINT#2 より、「In」の整数部けた数 4 の方が大きいので、整数部は 4 けたで表記されます。



● **例 8**

変数	設定内容
In	LREAL#123456.789
Exponent	FALSE
Sign	TRUE
MinLen	USINT#4
DecPlace	USINT#0

「MinLen」=USINT#4 より、「In」の整数部けた数 6 の方が大きいので、整数部は 6 けたで表記されます。また、「In」の値が正数なので、符号カラムに最上位けたの数字が入ります。



参考

- 「Exponent」、「Sign」、「MinLen」、「DecPlace」は、それぞれ省略可能です。省略した場合は、それぞれの初期値が適用されます。
- REAL 型変数から文字列への変換は、「RealToFormatString (P.2-290)」命令を使用してください。
- 文字列から実数への変換は、「STRING_TO_**（文字列→実数変換グループ） (P.2-306)」命令を使用してください。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

・「DecPlace」の値が、有効範囲外の場合。
・「DecPlace」の値が、「MinLen」の値より大きい場合。

STRING_TO_**（文字列→整数変換グループ）

文字列を、整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
STRING_TO_**	文字列→整数変換グループ	FUN	(@)STRING_TO_** EN　ENO In　Out ただし、“**”は整数のデータ型名	Out:=STRING_TO_**(In); ただし、“**”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	''
Out	変換結果	出力	変換結果	データ型に従う	－	－

*1. 「Out」のデータ型によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-303)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out						○	○	○	○	○	○	○	○							

機能

文字列「In」を、整数型に変換します。

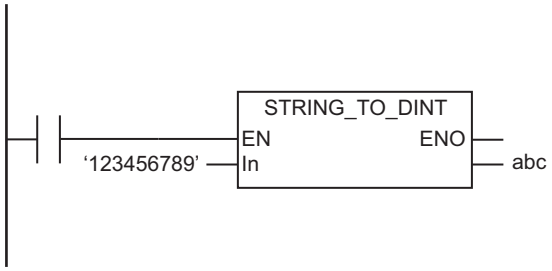
「In」の文字列は、'0'～'9'の数字だけで構成されていなければなりません。ただし、つぎのような例外があります。

- 「In」の先頭に'-'（マイナス）または'+'（プラス）がひとつだけある場合、それを符号として処理します。
- 「In」の先頭にいくつかの' '（空白文字）がある場合、それを無視します。
- 先頭の'-'（マイナス）または'+'（プラス）と数字との間に、いくつかの' '（空白文字）がある場合、その' '（空白文字）を無視します。
- 任意の位置に、単独の'_'（アンダーバー）を含んでいる場合、それを無視します。
- 任意の位置に、'_'（アンダーバー）が 2 個以上連続している場合、異常になります。
- 先頭や最後尾に'_'（アンダーバー）がある場合、異常になります。
- 先頭の'-'（マイナス）または'+'（プラス）と数字との間に、'_'（アンダーバー）がある場合、異常になります。

命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が DINT 型の場合、命令名称は STRING_TO_DINT となります。

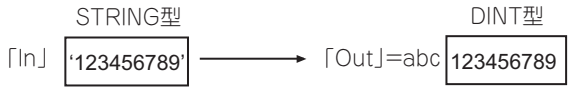
STRING_TO_DINT 命令で、「In」='123456789'の場合の例を、以下に示します。

LD



ST

```
abc:=STRING_TO_DINT('123456789');
```



有効範囲

「In」の有効範囲は、「Out」のデータ型によって、以下のように異なります。

「Out」のデータ型	「In」の有効範囲（最大バイト数）*1
USINT	4 バイト（半角英数 3 文字+終端 NULL 文字）
UINT	6 バイト（半角英数 5 文字+終端 NULL 文字）
UDINT	11 バイト（半角英数 10 文字+終端 NULL 文字）
ULINT	21 バイト（半角英数 20 文字+終端 NULL 文字）
SINT	5 バイト（半角英数 4 文字+終端 NULL 文字）
INT	7 バイト（半角英数 6 文字+終端 NULL 文字）
DINT	12 バイト（半角英数 11 文字+終端 NULL 文字）
LINT	21 バイト（半角英数 20 文字+終端 NULL 文字）

*1. 文字列先頭の複数の' '（空白文字）、文字列先頭の複数の'0'、文字列中の単独の'_'（アンダーバー）は、バイト数に含めません。

参考

- 文字列を、16 進数に変換したい場合は、「STRING_TO_**（文字列→ビット列変換グループ）(P.2-304)」命令を使用してください。
- 整数型から文字列型への変換は、「**_TO_STRING（整数→文字列変換グループ）（P.2-284)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- 「In」の値が'-0'の場合は、「Out」の値は「0」になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、数値を表す文字列ではない場合。
  b）変換結果が、「Out」のデータ型の有効範囲外の場合。

STRING_TO_**（文字列→ビット列変換グループ）

文字列を、ビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
STRING_TO_**	文字列→ビット列変換グループ	FUN	(@)STRING_TO_** EN　ENO In　Out ただし、“**”はビット列のデータ型名	Out:=STRING_TO_**(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	－	''
Out	変換結果	出力	変換結果	データ型に従う	－	－

*1. 「Out」のデータ型によって、有効範囲が変わります。詳細は、「有効範囲 (P.2-305)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out		○	○	○	○															

機能

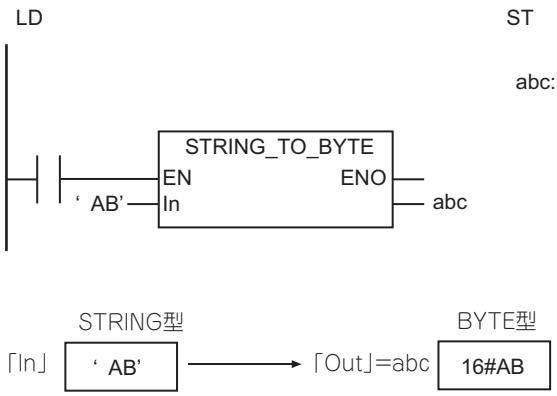
文字列「In」を 16 進数とみなし、ビット列に変換します。

「In」の文字列は、“0”～“9”の数字、または“a”～“f”および“A”～“F”だけで構成されていなければなりません。ただし、つぎのような例外があります。

- 「In」の先頭に' '（空白文字）や'0'が連続してある場合、それを無視します。
- 任意の位置に、単独の'_'（アンダーバー）を含んでいる場合、それを無視します。
- 任意の位置に、'_'（アンダーバー）が 2 個以上連続している場合、異常になります。
- 先頭や最後尾に'_'（アンダーバー）がある場合、異常になります。
- 先頭の'-'（マイナス）または'+'（プラス）と数字との間に、'_'（アンダーバー）がある場合、異常になります。

命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が BYTE 型の場合、命令名称は STRING_TO_BYTE となります。

STRING_TO_BYTE 命令で、「In」=' AB'の場合の例を、以下に示します。先頭の' '（ 空白文字）は、無視されます。



有効範囲

「In」の有効範囲は、「Out」のデータ型によって、以下のように異なります。

「Out」のデータ型	「In」の有効範囲（最大バイト数）*1
BYTE	3 バイト（半角英数 2 文字+終端 NULL 文字）
WORD	5 バイト（半角英数 4 文字+終端 NULL 文字）
DWORD	9 バイト（半角英数 8 文字+終端 NULL 文字）
LWORD	17 バイト（半角英数 16 文字+終端 NULL 文字）

*1. 文字列先頭の複数の' '（空白文字）、文字列先頭の複数の'0'、文字列中の単独の'_'（アンダーバー）は、バイト数に含めません。

参考

- 符号付きの数値を表す文字列を扱う場合は、「STRING_TO_**（文字列→整数変換グループ）(P.2-302)」命令を使用してください。
- ビット列から文字列への変換は、「**_TO_STRING（ビット列→文字列変換グループ）(P.2-286)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、16 進の数値を表す文字列ではない場合。
  b) 変換結果が、「Out」のデータ型の有効範囲外の場合。

STRING_TO_**（文字列→実数変換グループ）

文字列を、実数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
STRING_TO_**	文字列→実数変換グループ	FUN	(@)STRING_TO_** EN　ENO In　Out ただし、“**”は実数のデータ型名	Out:=STRING_TO_**(In); ただし、“**”は実数のデータ型名

変数

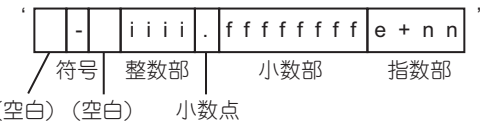
	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	最大 311 バイト（半角英数字 310 文字+終端 NULL 文字）	−	"
Out	変換結果	出力	変換結果	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out														○	○					

機能

文字列「In」を、実数に変換します。
命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が LREAL 型の場合、命令名称は STRING_TO_LREAL となります。

「In」の文字列形式は、つぎのとおりです。



名称	書式
符号	• 文字列の先頭に現れる連続する' '（空白文字）を無視した後に現れる単一の '+' または '-' を符号とします。 • '+'の符号は省略可能です。 • 符号の後の連続する空白文字も無視されます。

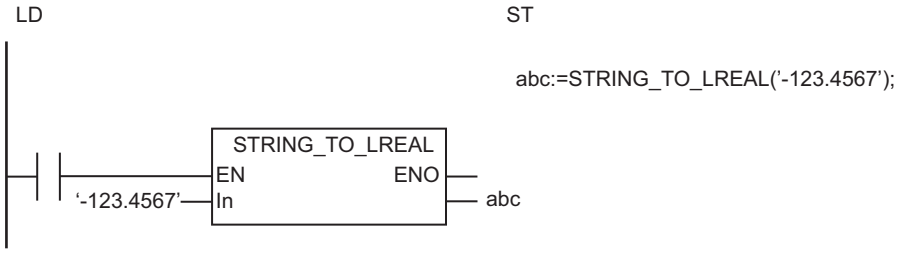
名称	書式
整数部	• 符号の後から小数点の前までを整数部とします。ただし、符号の後の連続する空白文字は、整数部に含めません。また、符号は省略される場合があります。 • 小数点および小数部がない場合は、指数部の前までを整数部とします。 • 小数点、小数部、指数部がない場合は、文字列終端までを整数部とします。 • 整数部は、'0'～'9'の数字で構成されます。 • 整数部は省略できません。 • 整数部の最大けた数は、最大文字列長 1985 から、符号、小数点、小数部、指数部、および符号の前後の空白文字のバイト数の合計を引いた数です。
小数点	• 整数部に続く単一の '.'（ピリオド） を小数点とします。 • 小数部が存在しない場合、小数点は省略してください。
小数部	• 小数点の後から指数部の前までを小数部とします。 • 指数部がない場合は、文字列終端までを小数部とします。 • 小数部は、'0'～'9'の数字で構成されます。 • 小数部は省略可能です。 • 小数部の最大けた数は 15 です。 • 小数点が存在しない場合、小数部も存在しません。
指数部	• 小数部に続く単一の'e'または'E'、それに続く単一の '+' または '-' から文字列終端までを、指数部とします。 • 小数部がない場合は、小数点に続く上記文字列を指数部とします。 • 小数点および小数部がない場合は、整数部に続く上記文字列を指数部とします。 • 指数部の数字部分は、'0'～'9'の数字で構成されます。 • 指数部は省略可能です。 • 指数部の数字部分の最大けた数は 3 です。

「In」の値が'+inf'の場合、「Out」の値は+∞となります。「In」の値が'-inf'の場合、「Out」の値は、-∞となります。どちらの場合も、大文字小文字の区別はありません。

記述例

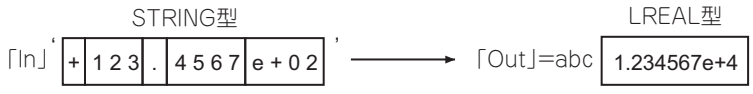
● 例 1

符号あり、小数点あり、小数部あり、指数部なしの場合の例を以下に示します。



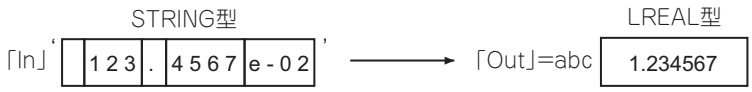
● 例 2

符号あり、小数点あり、小数部あり、指数部ありの場合の例を以下に示します。



● **例 3**

符号なし、小数点あり、小数部あり、指数部ありの場合の例を以下に示します。



● **例 4**

符号なし、小数点なし、小数部なし、指数部なしの場合の例を以下に示します。



参考

実数から文字列への変換は、「**_TO_STRING（実数→文字列変換グループ）（P.2-288)」命令を使用してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- 「In」の任意の位置に、単独の'_'（アンダーバー）を含む場合、それを無視します。
- 「In」の先頭や最後尾に'_'（アンダーバー）がある場合、異常になります。
- 「In」の任意の位置に、'_'（アンダーバー）が 2 個以上連続する場合、異常になります。
- 「In」の先頭の'-'（マイナス）または'+'（プラス）と数字との間に、'_'（アンダーバー）がある場合、異常になります。
- 「In」の内容が、「Out」のデータ型で表現できる精度を超えている場合、丸めが行われます。
- 「In」の内容が、「Out」のデータ型で表現できる値よりも 0 に近い場合、「Out」の値は 0 になります。
- 「In」の内容が、「Out」の有効範囲を超える場合、正数なら「Out」の値は+∞、負数なら-∞になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、数値を表す文字列ではない場合。
  b)「In」の文字列が、小数点ありで小数部なしの場合。

TO_**（整数変換グループ）

整数、ビット列、実数、文字列を、整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TO_**	整数変換グループ	FUN	(@)TO_** EN ENO In Out ただし、“**”は整数のデータ型名	Out:=TO_**(In); ただし、“**”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	–	*2
Out	変換結果	出力	変換結果	*1	–	–

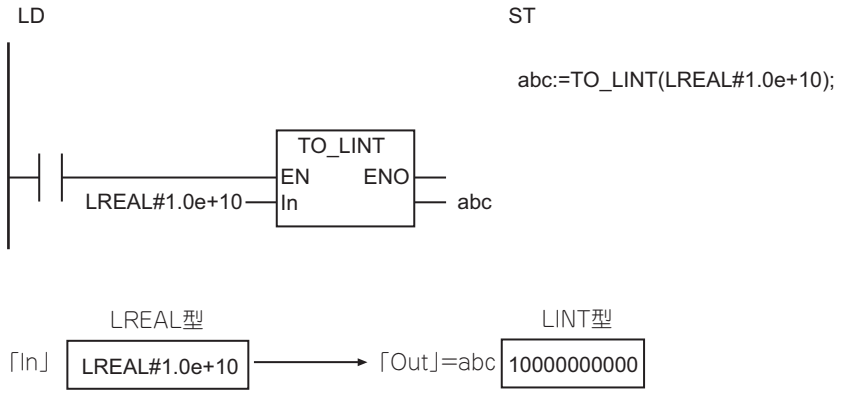
*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-310)」を参照してください。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○	○	○	○	○	○	○	○	○	○	○					○
Out						○	○	○	○	○	○	○	○							

機能

整数、ビット列、実数、文字列のいずれかである「In」を、整数に変換します。
命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が LINT 型の場合、命令名称は TO_LINT となります。

TO_LINT 命令で、「In」=LREAL#1.0e+10 の場合の例を、以下に示します。



・ 変換は、「In」のデータ型の有効けたの範囲で実行します。「In」が浮動小数点型の場合、小数点以下は、五捨五入して整数に丸めます。

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで異なります。それぞれの有効範囲は、**_TO_***（整数→整数変換グループ）命令の「有効範囲 (P.2-263)」、**_TO_***（ビット列→整数変換グループ）命令の「有効範囲 (P.2-271)」、**_TO_***（実数→整数変換グループ）命令の「有効範囲 (P.2-278)」を、参照してください。

「In」が STRING 型の場合の仕様の詳細は、STRING_TO_**（文字列→整数変換グループ）命令の「有効範囲 (P.2-303)」を、参照してください。

使用上の注意

・ 命令名称を、必ず「Out」のデータ型に合わせてください。
・「In」のデータ型がビット列で、「In」と「Out」のデータ型のサイズが異なる場合は、つぎの処理を行います。
  a)「In」のデータサイズより「Out」のデータサイズの方が大きい場合、「Out」の上位けたに 0 が埋められます。
  b)「In」のデータサイズより「Out」のデータサイズの方が小さい場合、上位けたが切り捨てられます。
・「In」が STRING 型の場合、つぎのことに注意してください。
  a)「In」の先頭に'-'（マイナス）または'+'（プラス）がひとつだけ存在する場合、符号として処理されます。
  b)「In」の先頭の'-'（マイナス）または'+'（プラス）を除いて、連続する'0'～'9'の数字で構成されていなければなりません。ただし、文字列の中の'_'（アンダーバー）と、'-'や'+'前後の' '（空白文字）は、あってもかまいません。
・ 変換結果が「Out」の有効範囲を超えると、「Out」の値は不定値になります。変換結果が「Out」の有効範囲を超えないように、「In」の値が有効範囲内に入っているか、必ずチェックしてください。
・ つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が STRING 型で、その値が数値を表す文字列ではない場合。

TO_**（ビット列変換グループ）

整数、ビット列、実数、文字列を、ビット列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TO_**	ビット列変換グループ	FUN	(@)TO_** EN ENO In Out ただし、“**”はビット列のデータ型名	Out:=TO_**(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1	–	*2
Out	変換結果	出力	変換結果	*1	–	–

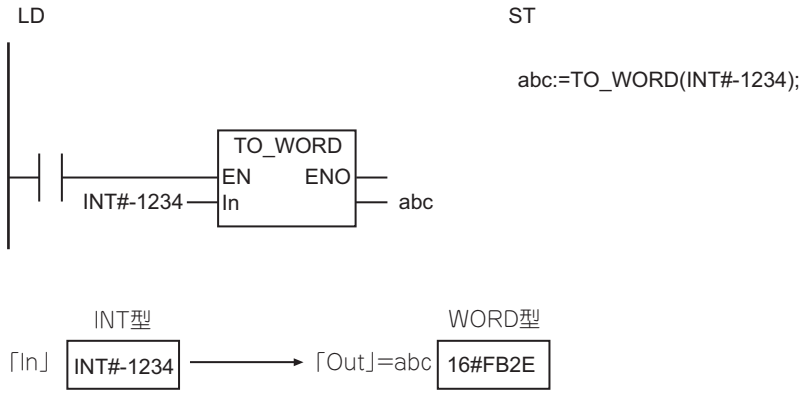
*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-312)」を参照してください。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○	○	○	○	○	○	○	○	○	○	○					○
Out		○	○	○	○															

機能

整数、実数、ビット列、文字列のいずれかである「In」を、ビット列に変換します。
命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が WORD 型の場合、命令名称は TO_WORD となります。

TO_WORD 命令で、「In」=INT#-1234 の場合の例を、以下に示します。



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで異なります。それぞれの有効範囲は、**_TO_***（整数→ビット列変換グループ）命令の「有効範囲 (P.2-266)」、**_TO_***（ビット列→ビット列変換グループ）命令の「有効範囲 (P.2-274)」を、参照してください。

「In」が STRING 型の場合の仕様の詳細は、STRING_TO_**（文字列→ビット列変換グループ）命令の「有効範囲 (P.2-305)」を、参照してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- 変換結果が「Out」の有効範囲を超えると、「Out」の値は不定値になります。変換結果が「Out」の有効範囲を超えないように、「In」の値が有効範囲内に入っているか、必ずチェックしてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が STRING 型で、その値が数値を表す文字列ではない場合。

TO_**（実数変換グループ）

整数、ビット列、実数、文字列を、実数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TO_**	実数変換グループ	FUN	(@)TO_** EN ENO In Out ただし、“**”は実数のデータ型名	Out:=TO_**(In); ただし、“**”は実数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	*1 *2	–	*3
Out	変換結果	出力	変換結果	*1	–	–

*1. 「In」、「Out」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「有効範囲 (P.2-314)」を参照してください。
*2. STRING の場合、最大 311 バイト（半角英数字 310 文字+終端 NULL 文字）です。
*3. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

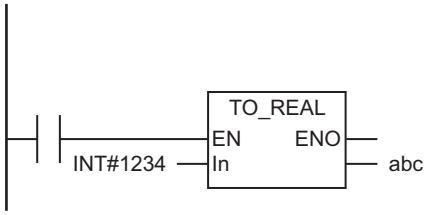
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○	○	○	○	○	○	○	○	○	○	○					○
Out														○	○					

機能

整数、ビット列、実数、文字列である変換対象「In」を、実数型に変換します。
命令名称は、変換結果「Out」のデータ型によって、異なります。たとえば、「Out」が REAL 型の場合、命令名称は TO_REAL となります。
「In」の値が+∞/-∞の場合、「Out」の値は+∞/-∞となります。

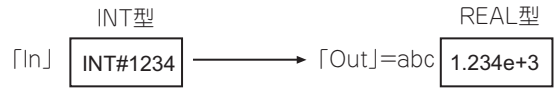
TO_REAL 命令で、「In」=INT#1234 の場合の例を、以下に示します。

LD



ST

```
abc:=TO_REAL(INT#1234);
```



有効範囲

「In」と「Out」の有効範囲は、データ型の組み合わせで異なります。それぞれの有効範囲は、**_TO_***（整数→実数変換グループ）命令の「有効範囲 (P.2-269)」、**_TO_***（ビット列→実数変換グループ）命令の「有効範囲 (P.2-276)」、**_TO_***（実数→実数変換グループ）命令の「有効範囲 (P.2-283)」を、参照してください。

「In」が STRING 型の場合の仕様の詳細は、STRING_TO_** （文字列→実数変換グループ）命令の「機能 (P.2-306)」を、参照してください。

使用上の注意

- 命令名称を、必ず「Out」のデータ型に合わせてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が STRING 型で、その値が数値を表す文字列ではない場合。

EnumToNum

列挙型を DINT 型に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EnumToNum	列挙型→整数変換	FUN	(@)EnumToNum EN ENO In Out	Out:=EnumToNum(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	−	−	0
Out	変換結果	出力	変換結果	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	列挙型																			
Out												○								

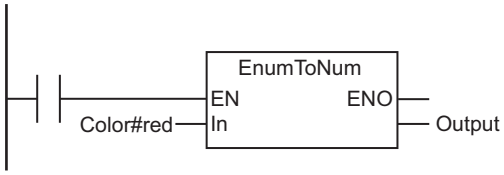
機能

列挙型の値である変換対象「In」を DINT 型の値に変換し、変換結果「Out」に出力します。

表示器などの、列挙型の変数を扱えない機器で、列挙型の変数の値をモニタしたい場合などに本命令を使用します。

列挙型「Color」の列挙子「red」を、DINT 型の変数「Output」に変換する場合の記述例を以下に示します。
「red」の列挙値が「0」場合、「Output」は、DINT#0 となります。

LD



ST

```
Output:=EnumToNum(Color#red);
```

サンプルプログラム

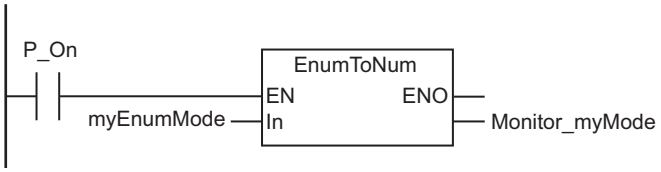
ユーザプログラムの動作モードを列挙型 EnumMode で定義しているとします。
動作モードを表示器でモニタするため、列挙型 EnumMode の変数「myEnumMode」の値を DINT 型の変数「Monitor_myMode」に変換します。たとえば、「myEnumMode」の値が mode2 の場合、「Monitor_myMode」の値は 2 となります。

データ型の定義

名称	列挙値	コメント
EnumMode	−	列挙型
mode0	0	メンバ
mode1	1	メンバ
mode2	2	メンバ

LD

名称	データ型	初期値	コメント
myEnumMode	EnumMode	mode0	列挙型のモードの値
Monitor_myMode	DINT	0	モードのモニタ値



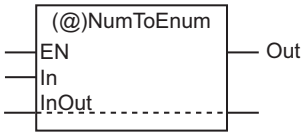
ST

名称	データ型	初期値	コメント
myEnumMode	EnumMode	mode0	列挙型のモードの値
Monitor_myMode	DINT	0	モードのモニタ値

```
Monitor_myMode:=EnumToNum(myEnumMode);
```

NumToEnum

DINT 型を列挙型に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NumToEnum	整数→列挙型変換	FUN		NumToEnum(In, InOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0
InOut	変換結果	入出力	変換結果	−	−	−
Out	戻り値	出力	TRUE: 命令を実行、かつ異常でない。 FALSE: 命令を実行しない、または異常である。	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In												○								
InOut	列挙型																			
Out	○																			

機能

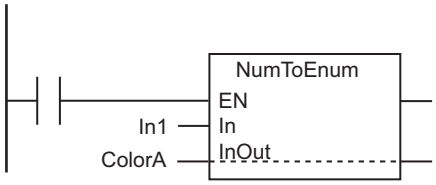
DINT 型の値である変換対象「In」を列挙型の値に変換し、変換結果「InOut」に出力します。

表示器などの、列挙型の変数を扱えない機器から、列挙型の変数の値を変更したい場合などに、本命令を使用します。

DINT 型の変数「In1」を、列挙型「Color」の変数「ColorA」に変換する場合の記述例を以下に示します。

「In1」の値が「1」、「Color」の列挙値 1 に対応する列挙子が「green」である場合、「ColorA」の値は「green」となります。

LD



ST

```
NumToEnum(In1,ColorA);
```

参考

本命令をラダー図プログラムで使用する場合、「In」の値が「InOut」の値の範囲内にあるかどうかを、「Out」によって調べることができます。

使用上の注意

「In」の値が「InOut」の値の範囲外である場合は、異常となります。「Out」は FALSE となり、「InOut」の値は変化しません。

サンプルプログラム

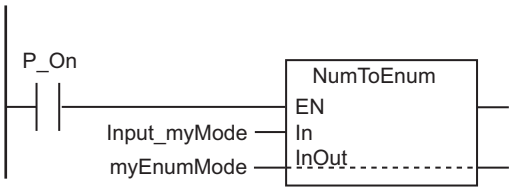
ユーザプログラムの動作モードを列挙型 EnumMode で定義しているとします。
表示器から動作モードを変更するため、DINT 型の値「Input_myMode」を書き込みます。
ユーザプログラムで、「Input_myMode」を列挙型 EnumMode の変数「myEnumMode」に変換します。たとえば、「Input_myMode」の値が「1」の場合、「myEnumMode」の値は mode1 となります。

データ型の定義

名称	列挙値	コメント
EnumMode	−	列挙型
mode0	0	メンバ
mode1	1	メンバ
mode2	2	メンバ

LD

名称	データ型	初期値	コメント
myEnumMode	EnumMode	mode0	列挙型のモードの値
Input_myMode	DINT	0	変更するモードの値



ST

名称	データ型	初期値	コメント
myEnumMode	EnumMode	mode0	列挙型のモードの値
Input_myMode	DINT	0	変更するモードの値

```
NumToEnum(Input_myMode, myEnumMode);
```

TRUNC / Round / RoundUp

実数を、整数にします。

TRUNC　：小数第 1 位を切り捨てます。
Round　：小数第 1 位を五捨五入します。
RoundUp ：小数第 1 位を切り上げます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TRUNC	実数切り捨て	FUN	(@)TRUNC EN　ENO In　Out	Out:=TRUNC(In);
Round	実数丸め	FUN	(@)Round EN　ENO In　Out	Out:=Round(In);
RoundUp	実数切り上げ	FUN	(@)RoundUp EN　ENO In　Out	Out:=RoundUp(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	－	*1
Out	変換結果	出力	変換結果	データ型に従う	－	－

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out												○	○							

機能

実数「In」の小数第 1 位を処理して、整数にします。

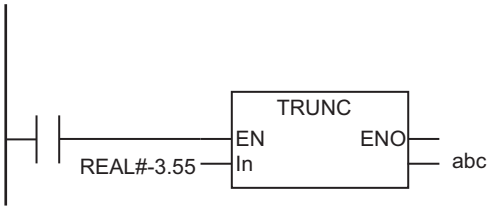
TRUNC

小数第 1 位を切り捨てます。

TRUNC 命令で、「In」= REAL#-3.55 の場合の記述例を、以下に示します。

変数「abc」の値は、DINT#-3 になります。

LD



ST

```
abc:=TRUNC(REAL#-3.55);
```

「In」の小数第1位を切り捨てます。
「In」の値がREAL#-3.55なので、abcの値はDINT#-3になります。

「In」 REAL#-3.55 → 小数第1位切り捨て → 「Out」=abc DINT#-3

Round

小数第 1 位を、五捨五入して整数にします。

● 五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1 -1.49→-1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2 -1.50→-2 -2.50→-2
0.5 より大	切り上げ	1.51→2 -1.51→-2

RoundUp

小数第 1 位を、切り上げます。

動作の違い

3 つの命令の動作の違いを、いくつかの例で示します。

入力値	出力値		
	TRUNC	Round	RoundUp
REAL#1.6	DINT#1	DINT#2	DINT#2
REAL#1.5	DINT#1	DINT#2	DINT#2
REAL#1.4	DINT#1	DINT#1	DINT#2
REAL#2.5	DINT#2	DINT#2	DINT#3
REAL#-1.6	DINT#-1	DINT#-2	DINT#-2
REAL#-1.5	DINT#-1	DINT#-2	DINT#-2
REAL#-1.4	DINT#-1	DINT#-1	DINT#-2
REAL#-2.5	DINT#-2	DINT#-2	DINT#-3

参考

「In」のデータ型が REAL なら、「Out」のデータ型は DINT となります。
「In」のデータ型が LREAL なら「Out」のデータ型は LINT となります。

使用上の注意

- 変換結果が「Out」の有効範囲を超えると、「Out」の値は不定値になります。
  変換結果が「Out」の有効範囲を超えないように、「In」の値が有効範囲内に入っているか、必ずチェックしてください。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

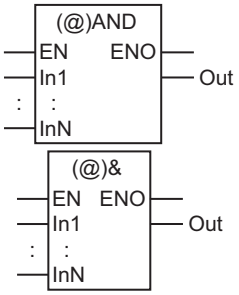
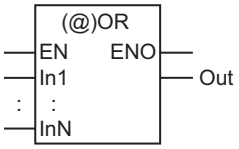
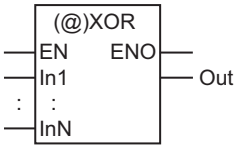
ビット列演算命令

命令	名称	ページ
AND, & / OR / XOR	論理積／論理和／排他的論理和	P.2-324
XORN	排他的論理和否定	P.2-327
NOT	ビット反転	P.2-329
AryAnd / AryOr / AryXor / AryXorN	配列論理積／配列論理和／配列排他的論理和／配列排他的論理和否定	P.2-331

AND, & / OR / XOR

複数のブール、ビット列の 1 ビットごとの演算を行います。

AND, & ： 論理積
OR ： 論理和
XOR ： 排他的論理和

命令	名称	FB/FUN	グラフィック表現	ST 表現
AND, &	論理積	FUN		Out:=In1 AND ··AND InN; Out:=In1 & ··& InN;
OR	論理和	FUN		Out:=In1 OR ··OR InN;
XOR	排他的論理和	FUN		Out:=In1 XOR ··XOR InN;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	演算対象	入力	演算対象 N は 2～5	データ型に従う	－	0 *1
Out	演算結果	出力	演算結果	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN	○	○	○	○	○															
Out	「In1」～「InN」と同じデータ型																			

機能

複数のブールまたはビット列の演算対象「In1」～「InN」に対して、対応するビットごとの演算を行います。
したがって、「In1」～「InN」および「Out」は、同じデータ型でなければなりません。

演算対象が 3 個以上ある場合、処理の手順はつぎのようになります。

1 **「In1」と「In2」とで演算**

2 **1 の結果と「In3」とで演算**

3 **2 の結果と「In4」とで演算**

以下、同じように演算します。

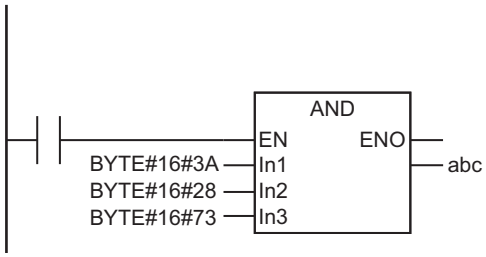
AND, &

両者とも TRUE のとき、演算結果が TRUE。それ以外は、演算結果が FALSE。

「In1」のビット	「In2」のビット	「Out」のビット
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

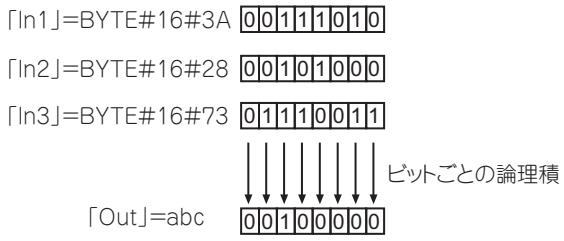
AND 命令で、「In1」=BYTE#16#3A、「In2」= BYTE#16#28、「In3」=BYTE#16#73 の場合の例を、以下に示します。

LD



ST

```
abc:=BYTE#16#3A AND BYTE#16#28 AND BYTE#16#73;
```



AND 命令と&命令の機能はまったく同じです。使いやすい方を使用してください。

OR

両者とも FALSE のとき、演算結果が FALSE。それ以外は、演算結果が TRUE。

「In1」のビット	「In2」のビット	「Out」のビット
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

XOR

両者の値が同じとき、演算結果が FALSE。異なるとき、演算結果が TRUE。

「In1」のビット	「In2」のビット	「Out」のビット
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

参考

ST 言語で以下のような記述をする場合は、入力変数の数に制限はありません。

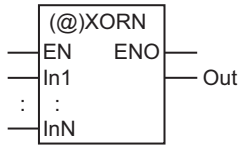
```
Out:=In1 AND In2 AND In3 AND In4 AND In5 AND In6 ···
Out:=In1 & In2 & In3 & In4 & In5 & In6 ···
Out:=In1 OR In2 OR In3 OR In4 OR In5 OR In6 ···
Out:=In1 XOR In2 XOR In3 XOR In4 XOR In5 XOR In6 ··
```

使用上の注意

「In1」～「InN」および「Out」のデータ型はすべて同じにしてください。
すべて同じでないと、ビルド時に異常となります。

XORN

複数のブール、ビット列の 1 ビットごとの排他的論理和否定演算を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
XORN	排他的論理和否定	FUN		Out:=In1 XOR NOT ‥ XOR NOT InN;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	演算対象	入力	演算対象 N は 2～5	データ型に従う	−	0 *1
Out	演算結果	出力	演算結果	データ型に従う	−	−

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN	○	○	○	○	○															
Out	「In1」～「InN」と同じデータ型																			

機能

複数のブールまたはビット列の演算対象「In1」～「InN」に対して、対応するビットごとの演算を行います。
したがって、「In1」～「InN」および「Out」は、同じデータ型でなければなりません。

演算対象が 3 個以上ある場合、処理の手順はつぎのようになります。

1 **「In1」と「In2」とで演算**

2 **1 の結果と「In3」とで演算**

3 **2 の結果と「In4」とで演算**

以下、同じように演算します。

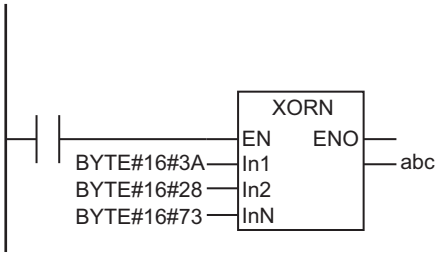
XORN

演算の入出力関係は、つぎのようになります。
両者の値が同じとき演算結果が TRUE、それ以外は FALSE になります。

「In1」のビット	「In2」のビット	「Out」のビット
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

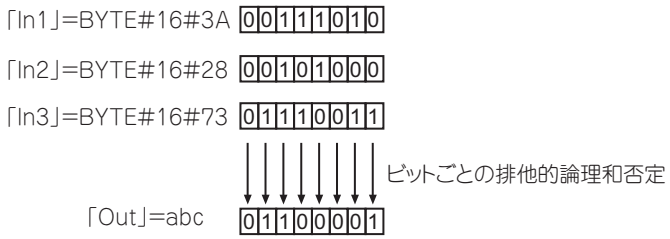
「In1」=BYTE#16#3A、「In2」= BYTE#16#28、「In3」=BYTE#16#73 の場合の例を、以下に示します。

LD



ST

```
abc:=BYTE#16#3A XOR NOT BYTE#16#28
      XOR NOT BYTE#16#73;
```



使用上の注意

「In1」～「InN」および「Out」のデータ型はすべて同じにしてください。
すべて同じでないと、ビルド時に異常となります。

NOT

ブール、ビット列の各ビットを反転します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NOT	ビット反転	FUN	(@)NOT EN ENO In Out	Out:=NOT(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	演算対象	入力	演算対象	データ型に従う	−	*1
Out	演算結果	出力	演算結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

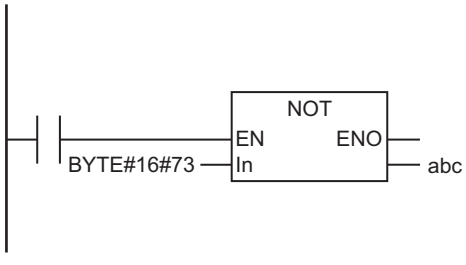
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○															
Out	「In1」と同じデータ型																			

機能

ブールまたはビット列の演算対象「In」の、各ビットを反転します。
したがって、「In」と演算結果「Out」は、ビット数が同じ、すなわち同じデータ型でなければなりません。

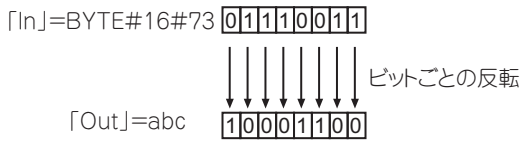
「In」=BYTE#16#73 の場合の例を、以下に示します。

LD



ST

```
abc:=NOT(BYTE#16#73);
```



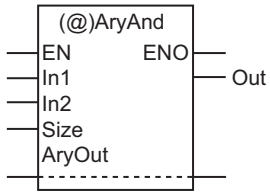
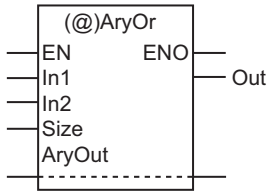
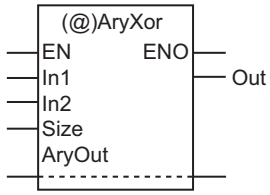
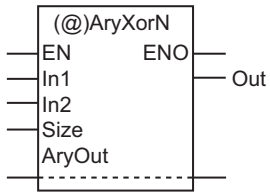
使用上の注意

「In」と「Out」のデータ型は、同じにしてください。
同じでないと、ビルド時に異常となります。

AryAnd / AryOr / AryXor / AryXorN

配列間で、各要素のブール、ビット列の 1 ビットごとの演算を行います。

AryAnd　：論理積
AryOr　：論理和
AryXor　：排他的論理和
AryXorN　：排他的論理和否

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryAnd	配列論理積	FUN		AryAnd(In1, In2, Size, AryOut);
AryOr	配列論理和	FUN		AryOr(In1, In2, Size, AryOut);
AryXor	配列排他的論理和	FUN		AryXor(In1, In2, Size, AryOut);
AryXorN	配列排他的論理和否定	FUN		AryXorN(In1, In2, Size, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1[], In2[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	要素数		演算対象要素数			1
AryOut[] 配列	演算結果配列	入出力	演算結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 配列	○	○	○	○	○															
In2[] 配列	In1[]と同じデータ型																			
Size							○													
AryOut[] 配列	In1[]と同じデータ型																			
Out	○																			

機能

演算対象配列 In1[]、In2[]の先頭の「Size」個の要素に対して、対応する要素の対応するビットごとに論理演算を行います。演算結果は、AryOut[]の対応する要素に格納されます。
したがって、In1[]、In2[]および AryOut[]は、同じデータ型でなければなりません。

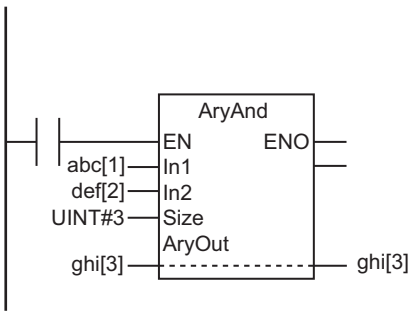
AryAnd

両者とも TRUE のとき、演算結果が TRUE。それ以外は、演算結果が FALSE。

In1[]の要素のビット	In2[]の要素のビット	AryOut[]のビット
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

AryAnd 命令で、「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

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

「Size」=UINT#3
- In1[0]=abc[1] TRUE　AND　In2[0]=def[2] TRUE　→　AryOut[0]=ghi[3] TRUE
- In1[1]=abc[2] FALSE　AND　In2[1]=def[3] TRUE　→　AryOut[1]=ghi[4] FALSE
- In1[2]=abc[3] FALSE　AND　In2[2]=def[4] FALSE　→　AryOut[2]=ghi[5] FALSE

AryOr

両者とも FALSE のとき、演算結果が FALSE。それ以外は、演算結果が TRUE。

In1[]の要素のビット	In2[]の要素のビット	AryOut[]のビット
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE

In1[]の要素のビット	In2[]の要素のビット	AryOut[]のビット
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

AryXor

両者の値が同じとき、演算結果が FALSE。異なるとき、演算結果が TRUE。

In1[]の要素のビット	In2[]の要素のビット	AryOut[]のビット
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

AryXorN

両者の値が同じとき、演算結果が TRUE。異なるとき、演算結果が FALSE。

In1[]の要素のビット	In2[]の要素のビット	AryOut[]のビット
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

使用上の注意

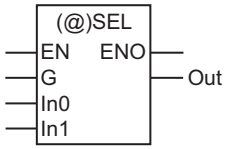
- In1[]、In2[]、AryOut[]のデータ型は、すべて同じにしてください。
  すべて同じでないと、ビルド時に異常となります。
- AryOut[]の配列要素数は、「Size」個以上にしてください。
- 「Size」の値が 0 の場合は、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、In1[]、In2[]、AryOut[]のいずれかの要素数を超える場合。

選択命令

命令	名称	ページ
SEL	ビット選択	P.2-336
MUX	マルチプレクサ	P.2-338
LIMIT	上下限リミット制限	P.2-341
Band	デッドバンド制御	P.2-343
Zone	デッドゾーン制御	P.2-345
MAX / MIN	最大値検索／最小値検索	P.2-347
AryMax / AryMin	配列変数の最大値検索／配列変数の最小値検索	P.2-349
ArySearch	配列検索	P.2-352

SEL

2 個の選択肢のうちの 1 つを選択します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SEL	ビット選択	FUN		Out:=SEL(G, In0, In1);

変数

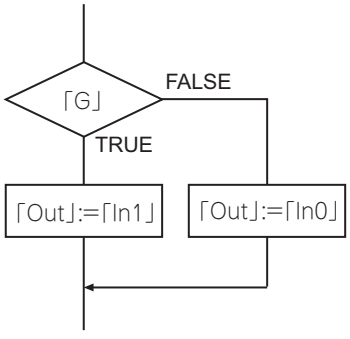
	名称	入／出力	内容	有効範囲	単位	初期値
G	ゲート	入力	FALSE:「In0」を選択 TRUE:「In1」を選択	データ型に従う	−	FALSE
In0, In1	選択肢		選択肢			*1
Out	選択結果	出力	選択結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
G	○																			
In0, In1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			
Out	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			

機能

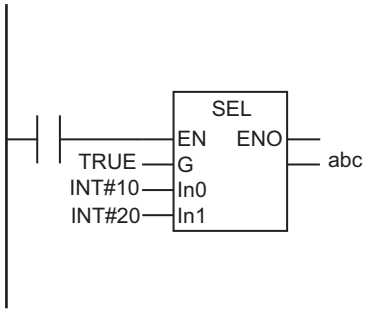
2 個の選択肢「In0」、「In1」のうちの 1 つを選択します。
「In0」と「In1」のどちらを選択するかは、ゲート「G」で指定します。
「G」の値が FALSE なら「In0」、TRUE なら「In1」が、「Out」に代入されます。



「In0」=INT#10、「In1」=INT#20、「G」=TRUE の場合の例を、以下に示します。

変数「abc」の値は、INT#20 になります。

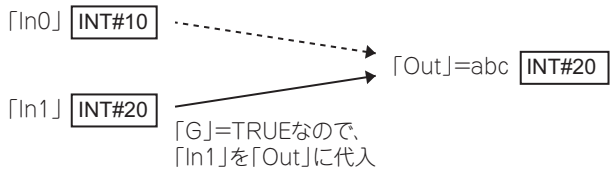
LD



ST

```
abc:=SEL(TRUE, INT#10, INT#20);
```

「In0」、「In1」のいずれかを選択します。
「G」=TRUEなので、「In1」=INT#20を選択し、abcに代入します。



参考

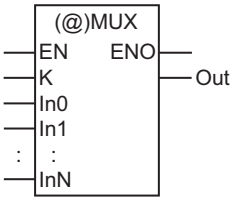
「MUX (P.2-338)」命令を使う方法もあります。MUX 命令は、2～5 個の選択肢のうちの 1 つを選択する命令です。

使用上の注意

- 「In0」、「In1」、「Out」は、異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a）「Out」の有効範囲が、「In0」、「In1」の有効範囲を包含するようにしてください。
  b）「In0」、「In1」、「Out」は、種類の異なるデータ型（ビット列と整数、整数と文字列など）にしないでください。

MUX

2～5 個の選択肢のうちの 1 つを選択します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MUX	マルチプレクサ	FUN		Out:=MUX(K, In0, In1, ... , InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
K	セレクタ	入力	0: 「In0」を選択 1: 「In1」を選択 2: 「In2」を選択 3: 「In3」を選択 4: 「In4」を選択	0～N	－	*1
In0 ～ InN	選択肢		選択肢 N は 1～4	データ型に従う		0 *2
Out	選択結果	出力	選択結果	データ型に従う	－	－

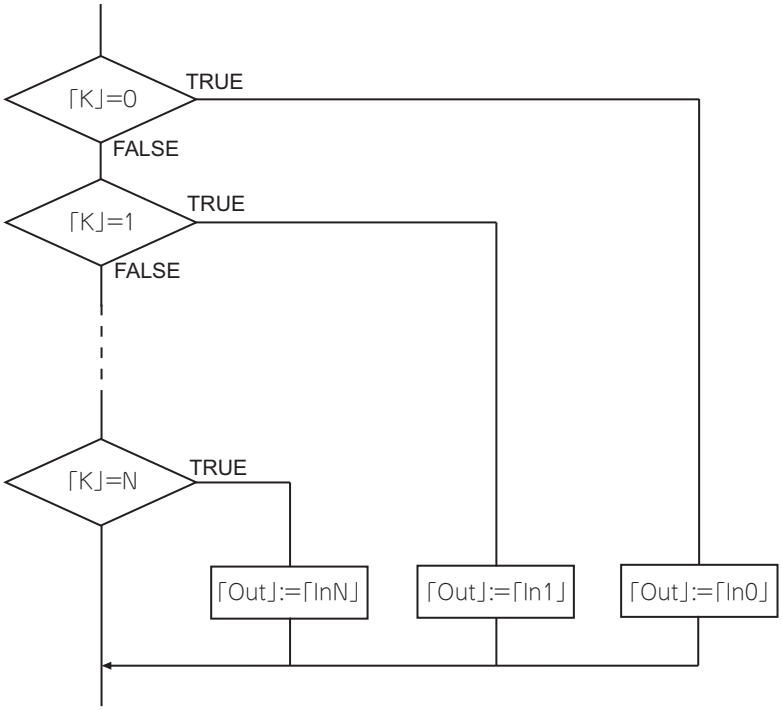
*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=2 の場合、「In0」と「In1」に接続する入力パラメータを省略すると初期値が適用されますが、「In2」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
K						○			○											
In0 ～ InN	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			
Out	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			

機能

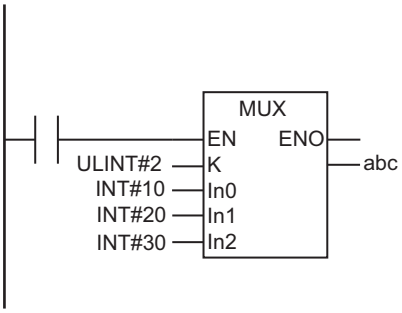
2～5 個の選択肢「In0」～「InN」のうちの 1 つを選択します。

「In0」～「InN」のどれを選択するかは、セレクタ「K」で指定します。
「K」の値が 0 なら「In0」、1 なら「In1」、以下「K」の値に応じた選択肢の値が、「Out」に代入されます。



「In0」=INT#10、「In1」=INT#20、「In2」=INT#30、「K」=ULINT#2 の場合の例を、以下に示します。変数「abc」の値は、INT#30 になります。

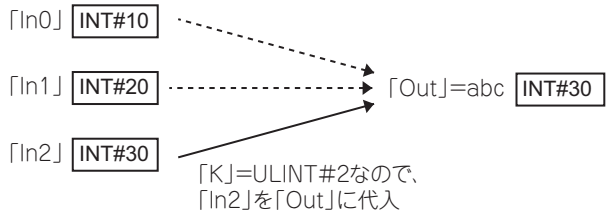
LD



ST

```
abc:=MUX(ULINT#2, INT#10, INT#20, INT#30);
```

「In0」～「InN」のいずれかを選択します。
「K」=ULINT#2なので、「In2」=INT#30を選択し、abcに代入します。



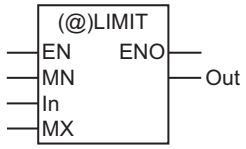
使用上の注意

- 「In0」～「InN」、および「Out」は、それぞれ異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a）「Out」の有効範囲が、「In0」～「InN」のすべての有効範囲を包含するようにしてください。
  b）「In0」～「InN」、「Out」は、種類の異なるデータ型（ビット列と整数、整数と文字列など）にしないでください。

・つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
a)「K」の値が、有効範囲外（0 未満または N より大きい）の場合。

LIMIT

設定された上下限値によって、入力変数の値を制限します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LIMIT	上下限リミット制限	FUN		Out:=LIMIT(MN, In, MX);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
MN	下限値	入力	制限の下限値	データ型に従う	−	*1
In	制限対象		制限対象			
MX	上限値		制限の上限値			
Out	処理結果	出力	処理結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						○	○	○	○	○	○	○	○	○	○					
In						○	○	○	○	○	○	○	○	○	○					
MX						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

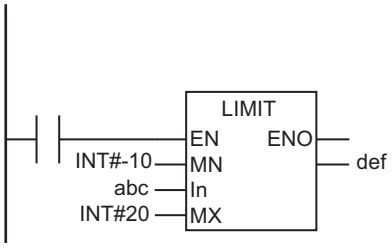
機能

上限値「MX」、下限値「MN」によって、制限対象「In」の値を制限します。
処理結果「Out」の値は、つぎのようになります。

「In」の値	「Out」の値
In < MN	MN
MN ≦ In ≦ MX	In
MX < In	MX

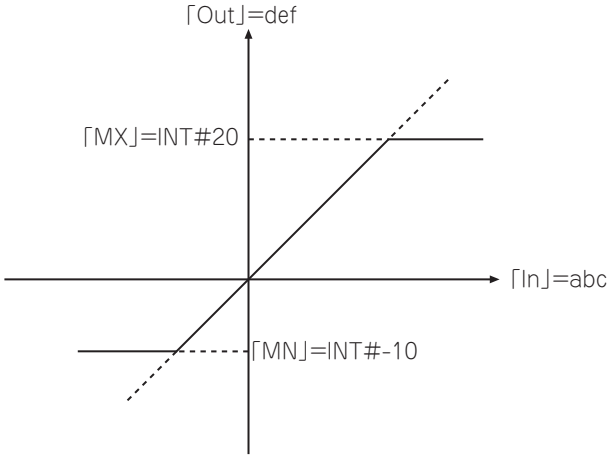
「MN」=INT#-10、「MX」=INT#20 の場合の例を、以下に示します。

LD



ST

```
def:=LIMIT(INT#-10, abc, INT#20);
```

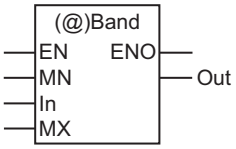


使用上の注意

- 「In」、「MN」、「MX」および「Out」は、それぞれ異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a)「Out」の有効範囲が、「In」、「MN」、「MX」のすべての有効範囲を包含するようにしてください。
  b)「In」、「MN」、「MX」は、符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)が混在しないようにしてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「MX」の値が「MN」の値より小さい場合。

Band

デッドバンド（不感帯）制御します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Band	デッドバンド制御	FUN		Out:=Band(MN, In, MX);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
MN	下限値	入力	デッドバンドの下限値	データ型に従う	−	*1
In	制御対象		制御対象			
MX	上限値		デッドバンドの上限値			
Out	処理結果	出力	処理結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN										○	○	○	○	○	○					
In										○	○	○	○	○	○					
MX										○	○	○	○	○	○					
Out										○	○	○	○	○	○					

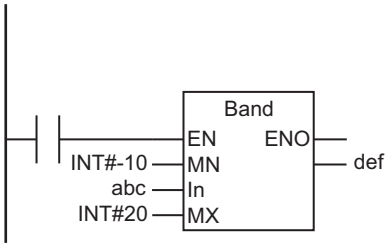
機能

上限値「MX」、下限値「MN」によって、制御対象「In」の値を制御します。
処理結果「Out」の値は、つぎのようになります。

「In」の値	「Out」の値
In < MN	「In」-「MN」
MN ≦ In ≦ MX	0
MX < In	「In」-「MX」

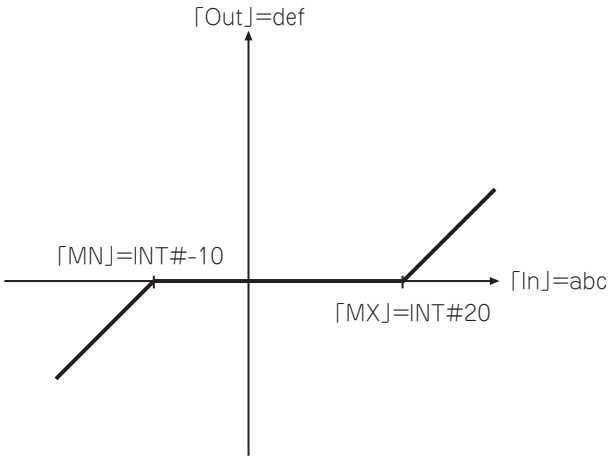
「MN」=INT#-10、「MX」=INT#20 の場合の例を、以下に示します。

LD



ST

```
def:=Band(INT#-10, abc, INT#20);
```



使用上の注意

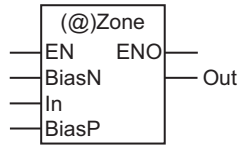
- 「In」、「MN」、「MX」および「Out」は、それぞれ異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a）「Out」の有効範囲が、「In」、「MN」、「MX」のすべての有効範囲を包含するようにしてください。
- 「In」の値が非数の場合、「Out」の値は非数になります。
- 「In」、「MN」、「MX」の値が+∞、-∞の場合、「Out」の値は、以下のようになります。

「In」の値	「MN」の値	「MX」の値	「Out」の値
+∞	+∞	+∞	0
		-∞	異常
	-∞	+∞	0
		-∞	+∞
-∞	+∞	+∞	-∞
		-∞	異常
	-∞	+∞	0
		-∞	0

- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「MX」の値が「MN」の値より小さい場合。
  b）「MX」、「MN」のいずれかが非数の場合。
  c）処理結果が、「Out」の有効範囲外の場合。

Zone

入力値にバイアス値を付加します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Zone	デッドゾーン制御	FUN		Out:=Zone(BiasN, In, BiasP);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
BiasN	負のバイアス値	入力	負のバイアス値	データ型に従う	−	*1
In	制御対象		制御対象			
BiasP	正のバイアス値		正のバイアス値			
Out	処理結果	出力	処理結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
BiasN										○	○	○	○	○	○					
In										○	○	○	○	○	○					
BiasP										○	○	○	○	○	○					
Out										○	○	○	○	○	○					

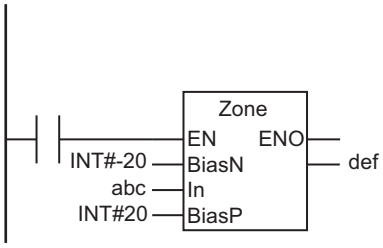
機能

正のバイアス値「BiasP」、負のバイアス値「BiasN」によって、制御対象「In」の値を制御します。
処理結果「Out」の値は、つぎのようになります。

「In」の値	「Out」の値
In ＜ 0	「In」＋「BiasN」
In ＝ 0	0
0 ＜ In	「In」＋「BiasP」

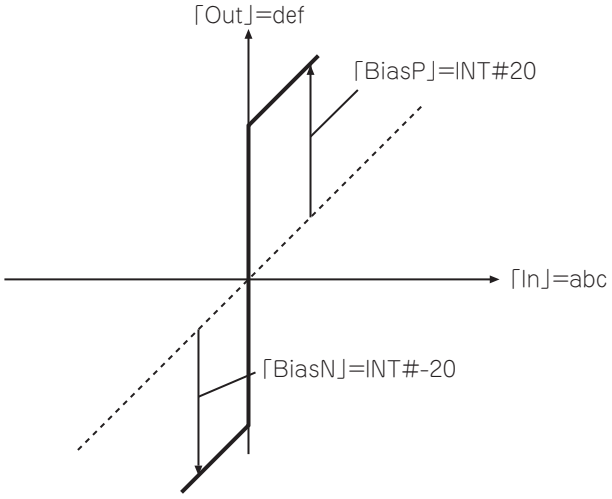
「BiasP」=INT#20、「BiasN」=INT#-20 の場合の例を以下に示します。

LD



ST

```
def:=Zone(INT#-20, abc, INT#20);
```



使用上の注意

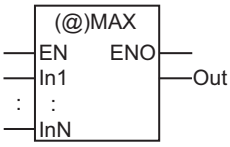
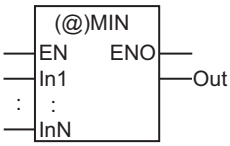
- 「In」、「BiasP」、「BiasN」および「Out」は、それぞれ異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a）「Out」の有効範囲が、「In」、「BiasP」、「BiasN」のすべての有効範囲を包含するようにしてください。
- 「In」の値が非数の場合、「Out」の値は非数になります。
- 「In」、「BiasP」、「BiasN」の値が+∞、-∞の場合、「Out」の値は、以下のようになります。

「In」の値	「BiasP」の値	「BiasN」の値	「Out」の値
+∞	+∞	+∞	+∞
		-∞	+∞
	-∞	+∞	異常
		-∞	0
-∞	+∞	+∞	0
		-∞	-∞
	-∞	+∞	異常
		-∞	-∞

- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「BiasP」＜「BiasN」の場合。
  b）「BiasP」、「BiasN」のいずれかが非数の場合。
  c）処理結果が、「Out」の有効範囲外の場合。

MAX / MIN

MAX ： 2～5 個の数値のうちの、最大値を検索します。
MIN ： 2～5 個の数値のうちの、最小値を検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MAX	最大値検索	FUN		Out:=MAX(In1, In2, ... , InN);
MIN	最小値検索	FUN		Out:=MIN(In1, In2, ... , InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1 ～ InN	演算対象	入力	演算対象 N は 2～5	データ型に従う	－	0 *1
Out	検索結果	出力	検索結果	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 ～ InN						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

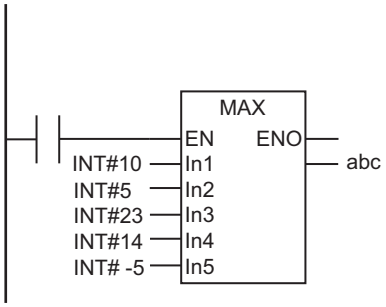
機能

MAX

2～5 個の演算対象「In1」～「InN」のうちの、最大値を検索します。

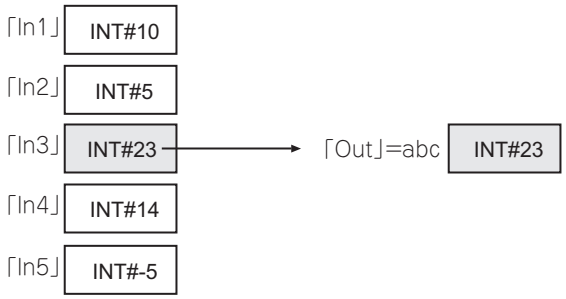
MAX 命令で、「In1」=INT#10、「In2」=INT#5、「In3」=INT#23、「In4」=INT#14、「In5」=INT#-5 の例を、以下に示します。

LD



ST

```
abc:=MAX(INT#10, INT#5, INT#23, INT#14, INT#-5);
```



MIN

2～5 個の演算対象「In1」～「InN」のうちの、最小値を検索します。

参考

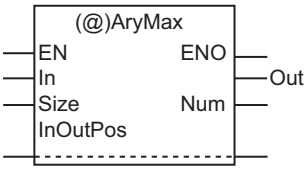
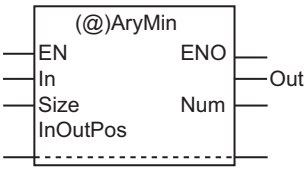
6 個以上の演算対象の最大値や最小値を検索したい場合は、「AryMax / AryMin (P.2-349)」命令を使用してください。

使用上の注意

- 「In1」～「InN」,および「Out」は、それぞれ異なるデータ型でもかまいませんが、つぎの点に注意してください。
  a)「Out」の有効範囲が、「In1」～「InN」のすべての有効範囲を包含するようにしてください。
  b)「In1」～「InN」には、符号付き整数型(SINT, INT, DINT, LINT)と符号なし整数型(USINT, UINT, UDINT, ULINT)が混在しないようにしてください。
- 「In1」～「InN」が実数の場合、誤差の影響で、期待どおりの結果が得られないことがあります。

AryMax / AryMin

AryMax ： 1 次元配列要素の最大値を検索します。
AryMin ： 1 次元配列要素の最小値を検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryMax	配列変数の最大値検索	FUN		Out:=AryMax(In, Size, InOutPos, Num);
AryMin	配列変数の最小値検索	FUN		Out:=AryMin(In, Size, InOutPos, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	検索対象配列	入力	検索対象配列	データ型に従う	－	*1
Size	検索対象の要素数		In[]の要素のうち、検索対象となるものの数			1
InOutPos	検索要素番号	入出力	検索値の配列要素番号	データ型に従う	－	－
Out	検索結果	出力	検索結果	データ型に従う	－	－
Num	検索値の数		検索値の数			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Size							○													
InOutPos							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Num							○													

機能

検索対象配列 In[]の、In[0]から始まる「Size」個の配列要素に対して、検索を行います。

検索値を「Out」に、検索要素番号を「InOutPos」に、検索値の数を「Num」にそれぞれ代入します。
「Num」が 1 より大きい場合、「InOutPos」の値は、検索値の中で最も下位の要素番号になります。

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

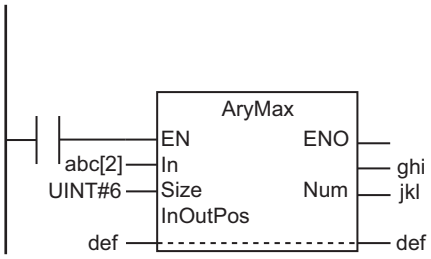
データ型	大小関係
TIME	値が大きい方が、大きいと判断します。
DATE, TOD, DT	日付や時刻が後の方が大きいと判断します。
STRING	「LTascii / LEascii / GTascii / GEascii (P.2-107)」命令と同じ仕様です。そちらを参照してください。

AryMax

最大値を検索します。

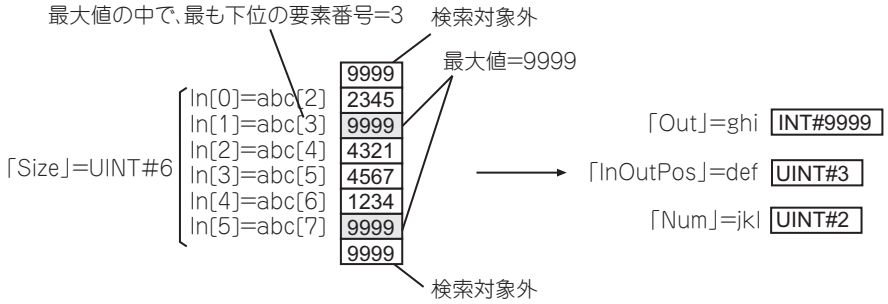
AryMax 命令で、「Size」=UINT#6 の場合の例を、以下に示します。
In[]に渡している入力パラメータは abc[2]なので、abc[2]以降が検索対象になります。

LD



ST

```
ghi:=AryMax(abc[2], UINT#6, def, jkl);
```



AryMin

最小値を検索します。

参考

TIME 型、DT 型、TOD 型の大小比較を行う場合、比較する値どうしの精度を合わせてください。
値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

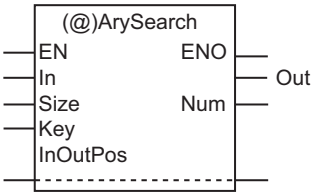
使用上の注意

- In[]と「Out」のデータ型が異なる場合は、「Out」の有効範囲が In[]の有効範囲を包含するようにしてください。
- In[]が実数の場合、誤差の影響で、期待どおりの結果が得られないことがあります。
- In[]は、必ず 1 次元の配列にしてください。
- 「Size」の値が 0 の場合、「Out」、「Num」の値は 0 になります。また、「InOutPos」の値は、変化しません。

- In[]が STRING 型で、「Size」の値が 0 の場合、「Out」は NULL 文字だけとなります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、有効範囲外の場合。
  b)「Size」が、In[]の配列の領域を超える場合。
  c) In[]が、1 次元配列ではない場合。
  d) In[]が、STRING 型でかつ NULL 文字で終端されていない場合。

ArySearch

1 次元配列から、指定した値を検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySearch	配列検索	FUN		Out:=ArySearch(In, Size, Key, InOutPos, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	検索対象配列	入力	検索対象配列	データ型に従う	−	*1
Size	検索対象の要素数		In[]の要素のうち、検索対象となるものの数			1
Key	検索キー		検索する値	データ型に従う		*1
InOutPos	検索要素番号	入出力	検索値の配列要素番号	データ型に従う	−	−
Out	検索結果	出力	検索結果	データ型に従う	−	−
Num	検索値の数		検索値の数			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型の配列も指定可能																			
Size							○													
Key	In[]の要素と同じデータ型																			
InOutPos							○													
Out	○																			
Num							○													

機能

1 次元の検索対象配列 In[]の、In[0]から始まる「Size」個の配列要素に対して、検索キー「Key」と同じ値の要素を検索します。

検索結果「Out」、検索要素番号「InOutPos」、検索値の数「Num」の値は、以下のようになります。

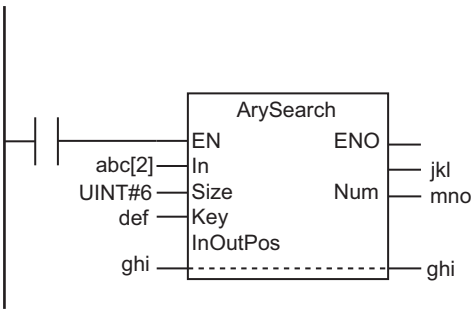
「Key」と同じ値の要素の存在	Out	InOutPos	Num
存在する	TRUE	「Key」と同じ値の要素のうち、最も下位にあるものの要素番号	「Key」と同じ値の要素の数
存在しない	FALSE	変化しない	0

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

データ型	大小関係
TIME	値が大きい方が、大きいと判断します。
DATE, TOD, DT	日付や時刻が後の方が大きいと判断します。

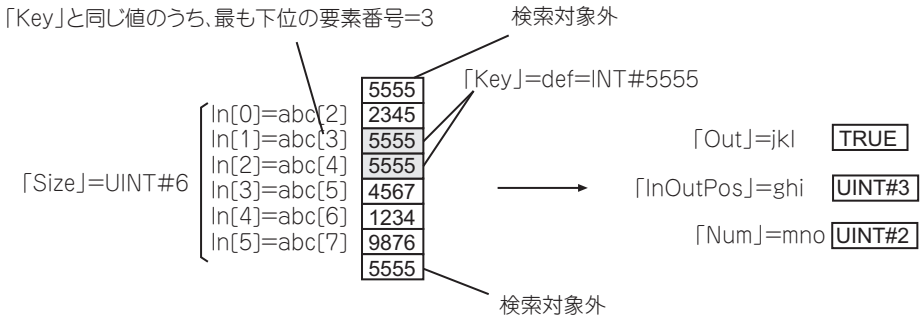
「Size」=UINT#6 の場合の例を、以下に示します。
In[]に渡している入力パラメータは abc[2]なので、abc[2]以降が検索対象になります。

LD



ST

```
jkl:=ArySearch(abc[2], UINT#6, def, ghi, mno);
```



参考

TIME 型、DT 型、TOD 型の大小比較を行う場合、比較する値どうしの精度を合わせてください。
値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- In[]は、必ず 1 次元の配列にしてください。
- 「Key」は、In[]の要素と同じデータ型にしてください。
- 「Size」の値が 0 の場合、「Out」、「Num」の値は 0 になります。また、「InOutPos」の値は、変化しません。
- 「Key」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「Key」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、「Num」、「InOutPos」は変化しません。
  a）「Size」が、In[]の配列の領域を超える場合。
  b）In[]が、STRING 型でかつ NULL 文字で終端されていない場合。
  c）In[]が、1 次元配列ではない場合。

データ転送命令

命令	名称	ページ
MOVE	転送	P.2-356
MoveBit	ビット転送	P.2-359
MoveDigit	ディジット転送	P.2-361
TransBits	多ビット転送	P.2-363
MemCopy	メモリコピー	P.2-365
SetBlock	ブロック設定	P.2-367
Exchange	データ交換	P.2-369
AryExchange	配列データ交換	P.2-371
AryMove	配列の転送	P.2-373
Clear	初期化	P.2-375
Copy**ToNum（ビット列→符号付き整数）	ビットパターンコピー（ビット列→符号付き整数）グループ	P.2-377
Copy**To***（ビット列→実数）	ビットパターンコピー（ビット列→実数）グループ	P.2-379
CopyNumTo**（符号付き整数→ビット列）	ビットパターンコピー（符号付き整数→ビット列）グループ	P.2-381
CopyNumTo**（符号付き整数→実数）	ビットパターンコピー（符号付き整数→実数）グループ	P.2-383
Copy**To***（実数→ビット列）	ビットパターンコピー（実数→ビット列）グループ	P.2-385
Copy**ToNum（実数→符号付き整数）	ビットパターンコピー（実数→符号付き整数）グループ	P.2-386

MOVE

変数や定数の値を、別の変数に転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MOVE	転送	FUN	(@)MOVE EN ENO In Out	Out:=In;

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	−	*1
Out	転送先	出力	転送先	データ型に従う	−	*1

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

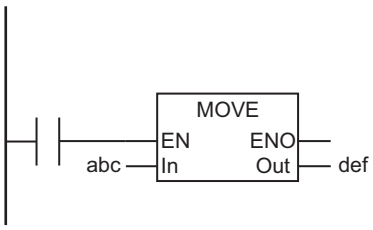
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
Out	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	「In」が列挙型、配列の 1 要素、構造体全体、構造体の 1 メンバの場合、「In」と同じデータ型を指定 「In」が配列全体の場合、「In」と同じデータ型、大きさ、添え字の配列を指定																			

機能

転送元「In」の値を、転送先「Out」に転送します。
「In」に渡す入力パラメータは、定数でも変数でもかまいません。また、「In」に列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバを指定することも可能です。

記述例を、以下に示します。変数「abc」の内容が、変数「def」に転送されます。

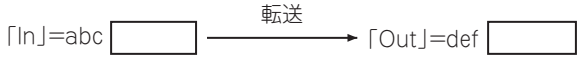
LD



ST

```
def:=abc;
```

「In」の値を「Out」に転送します。

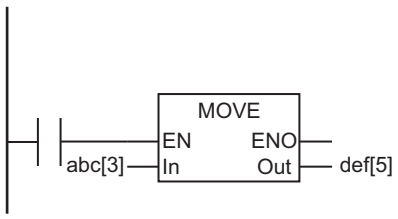


参考

- 配列を転送する場合、要素を 1 つだけ転送することも、配列全体を転送することも可能です。
  要素を 1 つだけ転送する場合は、配列変数名に添え字を付けます。
  配列全体を転送する場合は、配列変数名に添え字を付けません。

配列要素を1つだけ転送する場合

LD

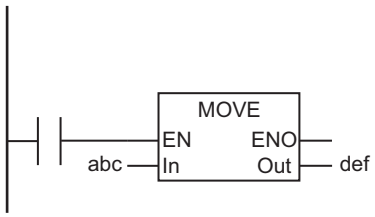


ST

```
def[5]:=abc[3];
```

配列全体を転送する場合

LD



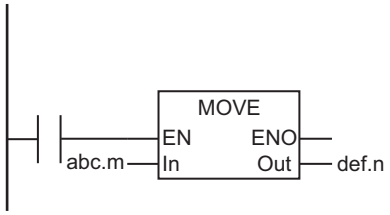
ST

```
def:=abc;
```

- 構造体を転送する場合、メンバを 1 つだけ転送することも、構造体全体を転送することも可能です。
  メンバを 1 つだけ転送する場合は、メンバを明示します。
  構造体全体を転送する場合は、構造体名だけを記述します。

構造体のメンバを1つだけ転送する場合

LD

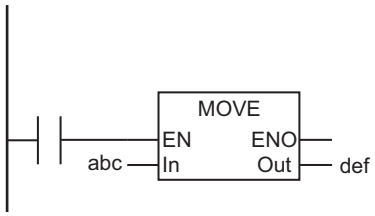


ST

```
def.n:=abc.m;
```

構造体全体を転送する場合

LD



ST

```
def:=abc;
```

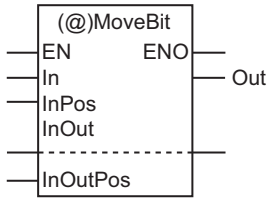
- 配列全体を転送する場合は、本命令より MemCopy 命令の方が高速に処理できます。

使用上の注意

- 「In」と「Out」のデータ型が異なる場合は、以下のいずれかのデータ型群の中から、「Out」の有効範囲が「In」の有効範囲を包含する組み合わせにしてください。
  a）BYTE, WORD, DWORD, LWORD
  b）USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL
- 「In」が列挙型、配列の 1 要素、構造体全体、構造体の 1 メンバの場合、「Out」は「In」と同じデータ型にしてください。
- 「In」が配列全体の場合、「Out」は「In」と同じデータ型、大きさ、添え字の配列にしてください。

MoveBit

ビット列内の 1 ビットを転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MoveBit	ビット転送	FUN		MoveBit(In, InPos, InOut, InOutPos);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	−	*1
InPos	転送元のビット位置		「In」の中の、転送されるビット位置	0～「In」のビット数-1		0
InOutPos	転送先のビット位置		「InOut」の中の、転送先ビット位置	0～「InOut」のビット数-1		
InOut	転送先	入出力	転送先	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

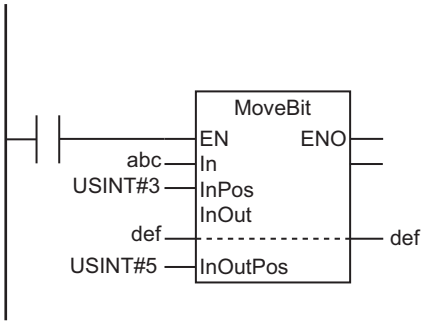
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
InPos						○														
InOutPos						○														
InOut		○	○	○	○															
Out	○																			

機能

転送元「In」のビット位置「InPos」にある 1 ビットデータを、転送先「InOut」のビット位置「InOutPos」に転送します。

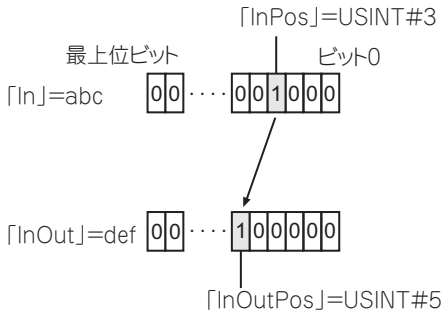
「InPos」=USINT#3、「InOutPos」=USINT#5 の場合の例を、以下に示します。

LD



ST

```
MoveBit(abc, USINT#3, def, USINT#5);
```

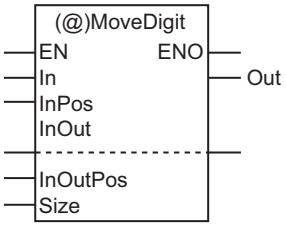


使用上の注意

- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「InOut」は変化しません。
  a)「InPos」の値が、有効範囲外の場合。
  b)「InOutPos」の値が、有効範囲外の場合。

MoveDigit

ビット列内の複数けた（1 けたは 4 ビット）を転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MoveDigit	ディジット転送	FUN		MoveDigit(In, InPos, InOut, InOutPos, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	−	*1
InPos	転送元のけた位置		「In」の中の、転送されるけた位置	0～「In」のビット数/4 - 1		0
InOutPos	転送先のけた位置		「InOut」の中の、転送先けた位置	0～「InOut」のビット数/4 - 1		
Size	転送けた数		転送するけた数	0～「In」のビット数/4		1
InOut	転送先	入出力	転送先	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

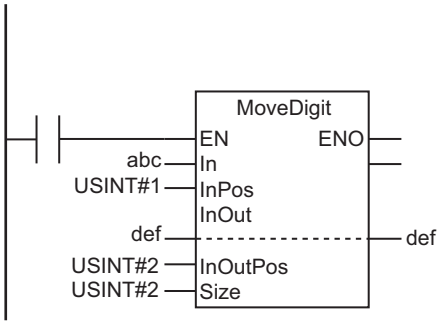
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
InPos						○														
InOutPos						○														
Size						○														
InOut		○	○	○	○															
Out	○																			

機能

転送元「In」のけた位置「InPos」から「Size」けた分を、転送先「InOut」のけた位置「InOutPos」に転送します。1 けたは 4 ビットです。

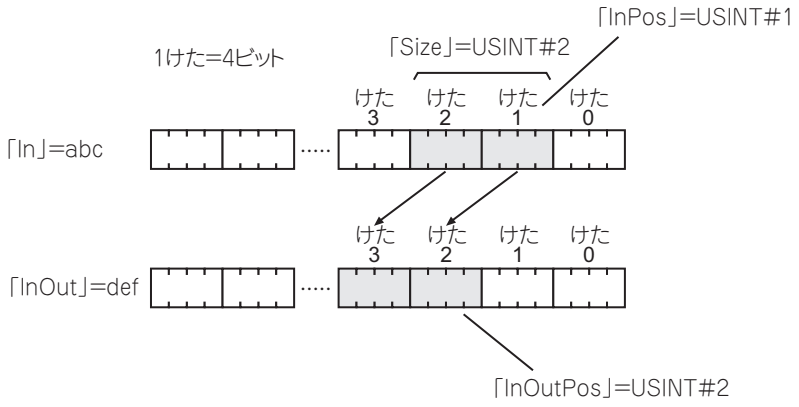
「InPos」=USINT#1、「InOutPos」=USINT#2、「Size」=USINT#2 の場合の例を、以下に示します。

LD



ST

```
MoveDigit(abc, USINT#1, def, USINT#2, USINT#2);
```

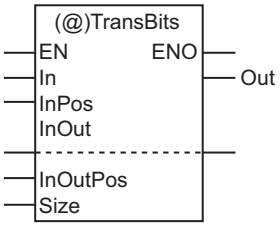


使用上の注意

- 転送先のけた位置が、「InOut」の最上位けたからはみ出すような指定をした場合、はみ出したデータは、「InOut」の最下位けたに回り込んで格納されます。
- 転送元のけた位置が、「In」の最上位けたからはみ出すような指定をした場合、はみ出したデータは、「In」の最下位けたに回り込んで転送されます。
- 「Size」の値が 0 の場合、「Out」の値は TRUE になり、「InOut」は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「InOut」は変化しません。
  a)「InPos」の値が、有効範囲外の場合。
  b)「InOutPos」の値が、有効範囲外の場合。
  c)「Size」の値が、有効範囲外の場合。

TransBits

ビット列内の複数ビットを転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TransBits	多ビット転送	FUN		TransBits(In, InPos, InOut, InOutPos, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	−	*1
InPos	転送元のビット位置		「In」の中の、転送されるビット位置	0～「In」のビット数 - 1		0
InOutPos	転送先のビット位置		「InOut」の中の、転送先ビット位置	0～「InOut」のビット数 - 1		
Size	転送ビット数		転送するビット数	0～「In」のビット数		1
InOut	転送先	入出力	転送先	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

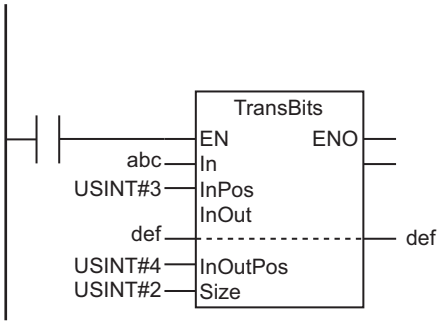
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
InPos						○														
InOutPos						○														
Size						○														
InOut		○	○	○	○															
Out	○																			

機能

転送元「In」のビット位置「InPos」から「Size」ビット分を、転送先「InOut」のビット位置「InOutPos」に転送します。

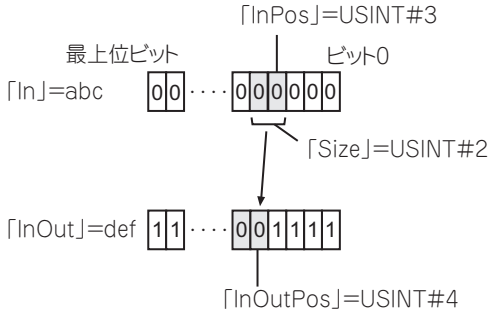
「InPos」=USINT#3、「InOutPos」=USINT#4、「Size」=USINT#2 の場合の例を、以下に示します。

LD



ST

```
TransBits(abc, USINT#3, def, USINT#4, USINT#2);
```



参考

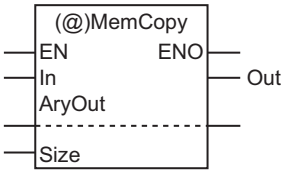
転送元と転送先のデータエリアが重なっていてもかまいません。

使用上の注意

- 転送元および転送先のビット位置が、「In」および「InOut」の最上位ビットからはみ出すような指定をしないでください。その場合、異常となり、動作しません。
- 「Size」の値が 0 の場合、転送は行われません。
- 「InOut」内の、転送に関係ないビットは変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「InOut」は変化しません。
  a)「InPos」の値が、有効範囲外の場合。
  b)「InOutPos」の値が、有効範囲外の場合。
  c)「Size」の値が、有効範囲外の場合。
  d)「InPos」と「Size」の指定が、「In」のビット数を超える場合。
  e)「InOutPos」と「Size」の指定が、「InOut」のビット数を超える場合。

MemCopy

複数の配列要素を転送します。転送元と転送先は、同じデータ型に限られます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MemCopy	メモリコピー	FUN		MemCopy(In, AryOut, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	転送元配列	入力	転送元配列	データ型に従う	−	*1
Size	転送要素数		転送する配列要素数			1
AryOut[] 配列	転送先配列	入出力	転送先配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

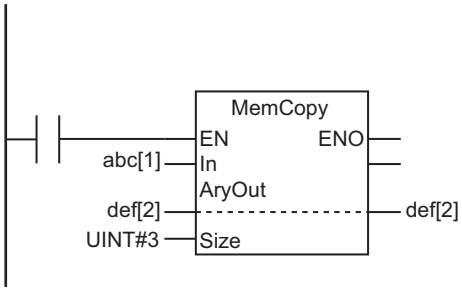
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Size							○													
AryOut[] 配列	In[]と同じデータ型の配列																			
Out	○																			

機能

転送元配列の In[]の In[0]から始まる「Size」個の要素を、転送先配列の AryOut[]の AryOut[0]以降に転送します。

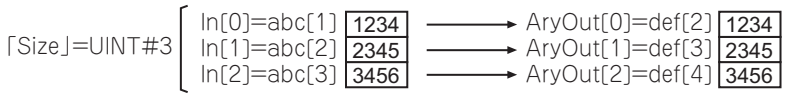
「Size」=UINT#3 の例を、以下に示します。

LD



ST

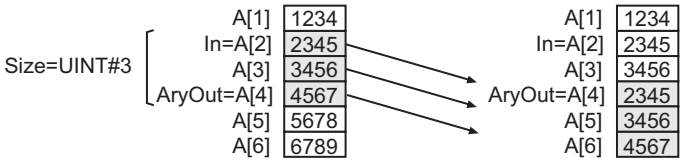
```
MemCopy(abc[1], def[2], UINT#3);
```



参考

- In[]と AryOut[]に、同じ配列の異なる位置を指定することも可能です。また、それらの位置が重複してもかまいません。

「In」=A[2]、「AryOut」=A[4]、「Size」=UINT#3 の場合の例を、以下に示します。



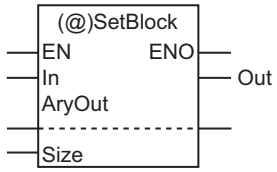
- 転送元と転送先のデータ型が異なる場合は、「AryMove (P.2-373)」命令を使用してください。
- 転送元と転送先のデータ型が同じ場合は、AryMove 命令より本命令の方が高速に処理できます。
- 配列ではない変数を転送する場合は、「MOVE (P.2-356)」命令を使用してください。

使用上の注意

- In[]と AryOut[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- In[]と AryOut[]が STRING 型の配列の場合、それぞれの領域サイズを一致させてください。
- 「Size」の値が 0 の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」が、In[]の配列領域を超える場合。
  b)「Size」が、AryOut[]の配列領域を超える場合。

SetBlock

複数の配列要素に、変数や定数の値を転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SetBlock	ブロック設定	FUN		SetBlock(In, AryOut, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	－	*1
Size	転送要素数		転送する配列要素数			1
AryOut[] 配列	転送先配列	入出力	転送先配列	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

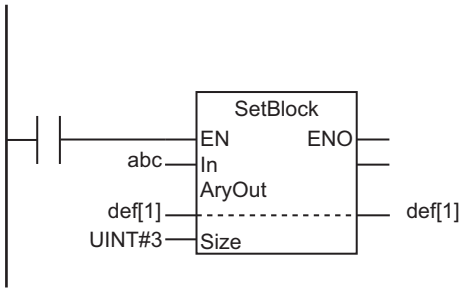
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、構造体全体、構造体の 1 メンバも指定可能																			
Size							○													
AryOut[] 配列	「In」と同じデータ型を要素とする配列																			
Out	○																			

機能

転送元「In」の値を、転送先配列 AryOut[]の AryOut[0]から始まる「Size」個の位置に転送します。

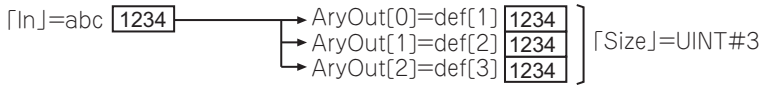
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

```
SetBlock(abc, def[1], UINT#3);
```

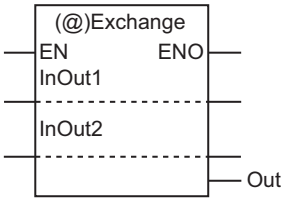


使用上の注意

- 「In」と AryOut[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- 「In」と AryOut[]が STRING 型の場合、それぞれの領域サイズを一致させてください。
- 「Size」の値が 0 の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Size」の値が、AryOut[]の配列領域を超える場合。

Exchange

2 つの変数の値を交換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Exchange	データ交換	FUN		Exchange(InOut1, InOut2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut1, InOut2	交換対象	入出力	交換対象	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

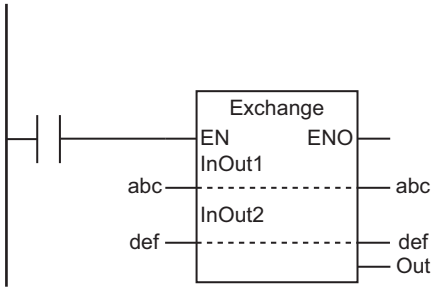
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、構造体全体、構造体の 1 メンバも指定可能																			
InOut2	「InOut1」と同じデータ型																			
Out	○																			

機能

交換対象「InOut1」と「InOut2」の値を交換します。
「InOut1」、「InOut2」には、列挙型、構造体全体、構造体の 1 メンバを指定することも可能です。

記述例を、以下に示します。変数「abc」の値と、変数「def」の値を交換します。

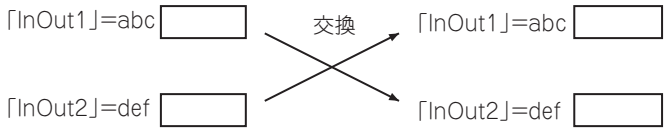
LD



ST

```
Exchange(abc, def);
```

「InOut1」と「InOut2」の値を交換します。

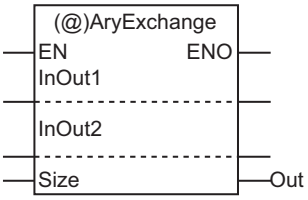


使用上の注意

- 「InOut1」と「InOut2」のデータ型は同じにしてください。異なると、ビルド時に異常になります。
- 「InOut1」と「InOut2」で指定する変数の領域が重複する場合、命令の実行結果は不定となります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「InOut1」、「InOut2」は変化しません。
  a)「InOut1」、「InOut2」が STRING 型で、どちらかの文字列の長さが、もう一方のサイズに収まらない場合。

AryExchange

2 つの配列間の要素を交換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryExchange	配列データ交換	FUN		AryExchange(InOut1, InOut2, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	交換要素数	入力	交換する要素数	データ型に従う	−	1
InOut1[], InOut2[] 配列	交換対象配列	入出力	交換対象配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

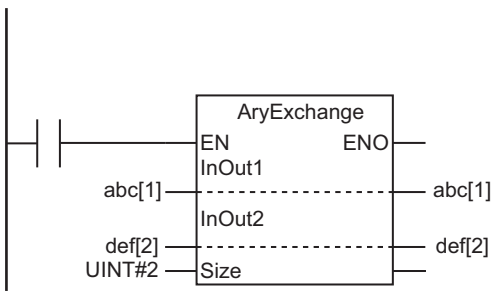
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
InOut1[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
InOut2[] 配列	InOut1[]と同じデータ型の配列																			
Out	○																			

機能

交換対象配列 InOut1[]の、InOut1[0]から始まる「Size」個の要素と、InOut2[]の、InOut2[0]から始まる「Size」個の要素を交換します。

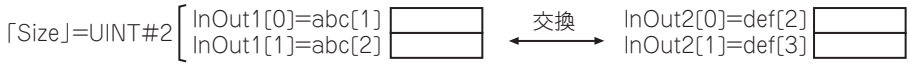
「Size」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
AryExchange(abc[1], def[2], UINT#2);
```



参考

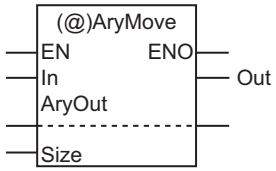
- 定数を変数に代入する場合は、「MOVE (P.2-356)」命令を使用してください。
- 変数の値を、別の変数にコピーする場合は「MemCopy (P.2-365)」命令を使用してください。

使用上の注意

- InOut1[]と InOut2[]の要素のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、InOut1[]も InOut2[]も変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut1[]も InOut2[]も変化しません。
  a）「Size」の値が、InOut1[]または InOut2[]の配列領域を超える場合。
  b）InOut1[]、InOut2[]が STRING 型の配列で、文字列の長さが、もう一方のサイズを超える要素が存在する場合。
  c）InOut1[]、InOut2[]が STRING 型の配列で、NULL 文字で終端されていない要素が存在する場合。

AryMove

複数の配列要素を転送します。転送元と転送先のデータ型が、異なっていてもかまいません。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryMove	配列の転送	FUN		AryMove(In, AryOut, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	転送対象配列	入力	転送する配列	データ型に従う	－	*1
Size	転送要素数		転送する要素数			1
AryOut[] 配列	転送結果配列	入出力	転送結果配列	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

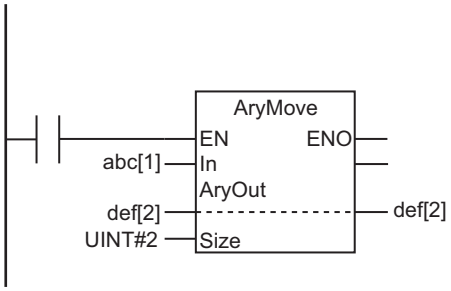
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Size							○													
AryOut[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Out	○																			

機能

転送対象配列 In[]の、In[0]から始まる「Size」個の要素を、転送結果配列 AryOut[]の、AryOut[0]から始まる要素に転送します。
In[]と AryOut[]のデータ型が、異なっていてもかまいません。

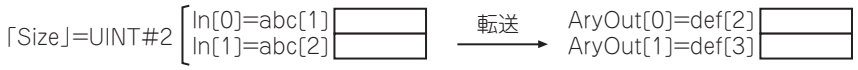
「Size」=UINT#2 の場合の例を、以下に示します。

LD



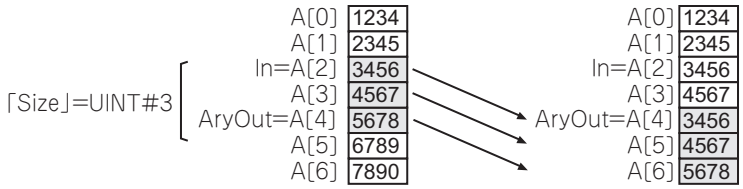
ST

```
AryMove(abc[1], def[2], UINT#2);
```



参考

- In[]と AryOut[]のデータ型が同じ場合は、MemCopy 命令の方が高速に処理できます。
- In[]と AryOut[]に、同じ配列を指定することも可能です。また、その場合、転送元要素と転送先要素が重なっていてもかまいません。In[0]=A[2]、AryOut[0]=A[4]、「Size」=UINT#3 の場合の例を以下に示します。

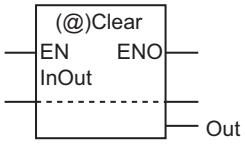


使用上の注意

- In[]と AryOut[]のデータ型が異なる場合は、以下のいずれかのデータ型群の中から、AryOut[]の有効範囲が In[]の有効範囲を包含する組み合わせにしてください。
  a）BYTE, WORD, DWORD, LWORD
  b）USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL
- In[]が構造体を要素とする配列の場合は、In[]と AryOut[]のデータ型を同じにしてください。
- 「Size」の値が「0」の場合、「Out」の値は TRUE になり、AryOut[]は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a）「Size」の値が、In[]または AryOut[]の大きさを超える場合。
  b）In[]、AryOut[]が STRING 型の配列で、かつ、転送される要素の文字列の長さが、AryOut[]の要素のサイズを超える場合。

Clear

変数を初期化します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Clear	初期化	FUN		Clear(InOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	初期化対象	入出力	初期化対象	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
Out	○																			

機能

初期化対象「InOut」の値を、初期化します。
変数の初期値属性が設定されている場合は、設定されている値に初期化されます。初期値属性が設定されていない場合は、「InOut」のデータ型のデフォルトの初期値になります。
「InOut」が外部変数の場合は、それに対応するグローバル変数の初期値属性とは関係なく、「InOut」のデータ型のデフォルトの初期値になります。

各データ型のデフォルトの初期値は、以下のとおりです。

データ型	デフォルトの初期値
BOOL	FALSE
BYTE, WORD, DWORD, LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

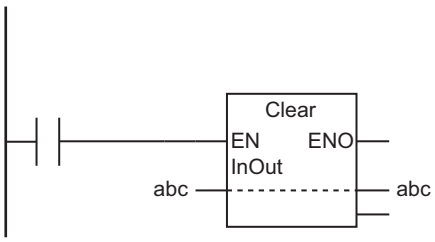
「InOut」が、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバの場合は、以下のように処理します。

InOut	処理内容
配列全体	配列のすべての要素が初期化されます。

InOut	処理内容
配列の 1 要素	その要素だけが初期化されます。
構造体全体	構造体のすべてのメンバが初期化されます。
構造体の 1 メンバ	そのメンバだけが初期化されます。

記述例を、以下に示します。変数「abc」の値を初期化します。たとえば「abc」=INT#100 の場合、「abc」の値は INT#0 になります。

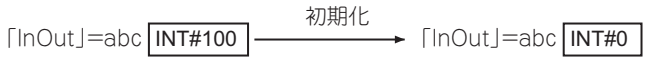
LD



ST

```
Clear(abc);
```

「InOut」の値を初期化します。
abcのデータ型がINTなので、abcの値はINT#0になります。



参考

- 「InOut」が、スタックとして使用している配列の場合は、本命令を実行するとともに、スタックの格納数を管理する変数の値も 0 にしてください。
- カムデータ変数を本命令で初期化した場合、値は MC_SaveCamTable 命令で保存されたものではなく、0 になります。

使用上の注意

- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- 列挙型の変数を初期化する場合は、初期値属性を設定してください。初期値属性が設定されていない場合、列挙型の変数の値は 0 になります。
- 以下のすべての条件に該当する処理は、行わないでください。正しく動作しない場合があります。
  a）FUN または FB に、入出力変数として BOOL 型配列の 1 要素を渡す。
  b）上記 FUN または FB 内で、Clear 命令を実行する。
  c）Clear 命令に渡すパラメータは、上記の BOOL 型配列の 1 要素を受けた入出力変数である。

Copy**ToNum（ビット列→符号付き整数）

ビット列の内容を、そのまま符号付き整数にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Copy**ToNum	ビットパターンコピー（ビット列→符号付き整数）グループ	FUN	(@)Copy**ToNum EN ENO In Out ただし、“**”はビット列のデータ型名	Out:=Copy**ToNum(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out	「In」のデータ型と同じサイズの、符号付き整数のデータ型																			

機能

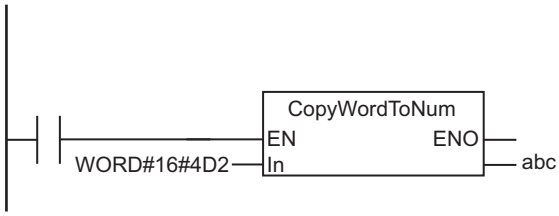
コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、4 種類あります。

In	Out	命令名称
BYTE	SINT	CopyByteToNum
WORD	INT	CopyWordToNum
DWORD	DINT	CopyDwordToNum
LWORD	LINT	CopyLwordToNum

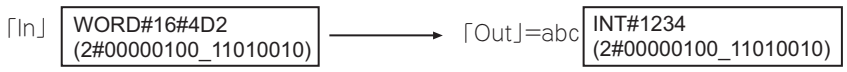
CopyWordToNum 命令で、「In」=WORD#16#4D2 の場合の例を、以下に示します。

LD



ST

```
abc:=CopyWordToNum(WORD#16#4D2);
```


「In」
WORD#16#4D2
(2#00000100_11010010)
「Out」=abc
INT#1234
(2#00000100_11010010)

Copy**To***（ビット列→実数）

ビット列の内容を、そのまま実数にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Copy**To***	ビットパターンコピー（ビット列→実数）グループ	FUN	(@)CopyDwordToReal EN ENO In Out (@)CopyLwordToLreal EN ENO In Out	Out:=CopyDwordToReal(In); または Out:=CopyLwordToLreal(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				○	○															
Out	「In」のデータ型が DWORD なら REAL、LWORD なら LREAL																			

機能

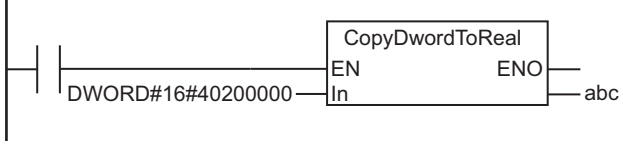
コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、2 種類あります。

In	Out	命令名称
DWORD	REAL	CopyDwordToReal
LWORD	LREAL	CopyLwordToLreal

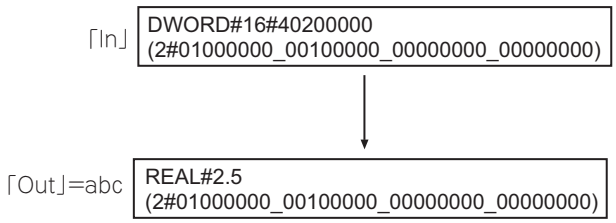
CopyDwordToReal 命令で、「In」=DWORD#16#40200000 の場合の例を、以下に示します。

LD



ST

```
abc:=CopyDwordToReal(DWORD#16#40200000);
```


「In」
DWORD#16#40200000
(2#01000000_00100000_00000000_00000000)
「Out」=abc
REAL#2.5
(2#01000000_00100000_00000000_00000000)

CopyNumTo**（符号付き整数→ビット列）

符号付き整数の内容を、そのままビット列にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CopyNumTo**	ビットパターンコピー（符号付き整数→ビット列）グループ	FUN	(@)CopyNumTo** EN ENO In Out ただし、“**”はビット列のデータ型名	Out:=CopyNumTo**(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										○	○	○	○							
Out	「In」のデータ型と同じサイズの、ビット列のデータ型																			

機能

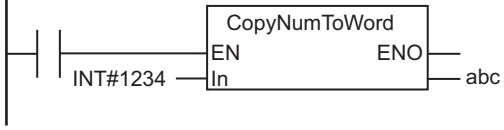
コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、4 種類あります。

In	Out	命令名称
SINT	BYTE	CopyNumToByte
INT	WORD	CopyNumToWord
DINT	DWORD	CopyNumToDword
LINT	LWORD	CopyNumToLword

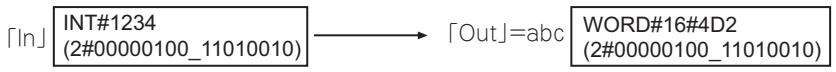
CopyNumToWord 命令で、「In」=INT#1234 の場合の例を、以下に示します。

LD



ST

```
abc:=CopyNumToWord(INT#1234);
```


「In」
INT#1234
(2#00000100_11010010)
「Out」=abc
WORD#16#4D2
(2#00000100_11010010)

CopyNumTo**（符号付き整数→実数）

符号付き整数の内容を、そのまま実数にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CopyNumTo**	ビットパターンコピー（符号付き整数→実数）グループ	FUN	(@)CopyNumToReal EN ENO In Out (@)CopyNumToLreal EN ENO In Out	Out:=CopyNumToReal(In); または Out:=CopyNumToLreal(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In												○	○							
Out	「In」のデータ型が DINT なら REAL、LINT なら LREAL																			

機能

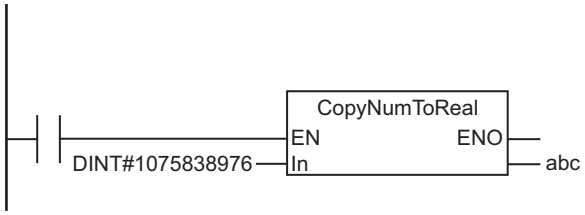
コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、2 種類あります。

In	Out	命令名称
DINT	REAL	CopyNumToReal
LINT	LREAL	CopyNumToLreal

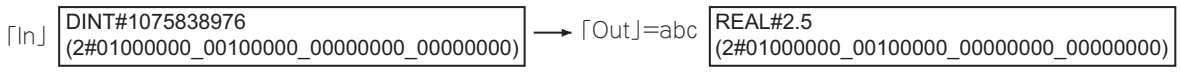
CopyNumToReal 命令で、「In」=DINT#1075838976 の場合の例を、以下に示します。

LD



ST

```
abc:=CopyNumToReal(DINT#1075838976);
```


「In」
DINT#1075838976
(2#01000000_00100000_00000000_00000000)
「Out」=abc
REAL#2.5
(2#01000000_00100000_00000000_00000000)

Copy**To***（実数→ビット列）

実数の内容を、そのままビット列にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Copy**To***	ビットパターンコピー（実数→ビット列）グループ	FUN	(@)CopyRealToDword EN / ENO In / Out (@)CopyLrealToLword EN / ENO In / Out	Out:=CopyRealToDword(In); または Out:=CopyLrealToLword(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0.0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	「In」のデータ型が REAL なら DWORD、LREAL なら LWORD																			

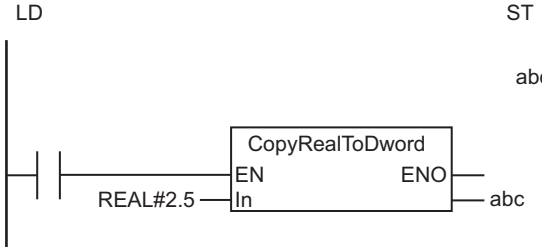
機能

コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、2 種類あります。

In	Out	命令名称
REAL	DWORD	CopyRealToDword
LREAL	LWORD	CopyLrealToLword

CopyRealToDword 命令で、「In」=REAL#2.5 の場合の例を、以下に示します。



「In」 REAL#2.5 (2#01000000_00100000_00000000_00000000) → 「Out」=abc DWORD#16#40200000 (2#01000000_00100000_00000000_00000000)

Copy**ToNum（実数→符号付き整数）

実数の内容を、そのまま符号付き整数にコピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Copy**ToNum	ビットパターンコピー（実数→符号付き整数）グループ	FUN	(@)CopyRealToNum EN ENO In Out (@)CopyLrealToNum EN ENO In Out	Out:=CopyRealToNum(In); または Out:=CopyLrealToNum(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	コピー元	入力	コピー元	データ型に従う	−	0.0
Out	コピー先	出力	コピー先	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	「In」のデータ型が REAL なら DINT、LREAL なら LINT																			

機能

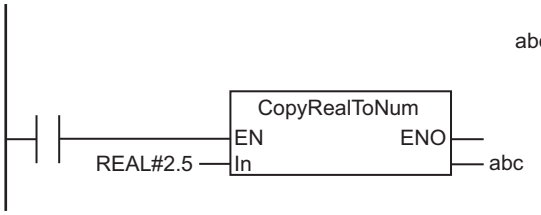
コピー元「In」の内容を、加工せずにそのままコピー先「Out」にコピーします。

命令名称は、「In」と「Out」のデータ型の組み合わせによって、2 種類あります。

In	Out	命令名称
REAL	DINT	CopyRealToNum
LREAL	LINT	CopyLrealToNum

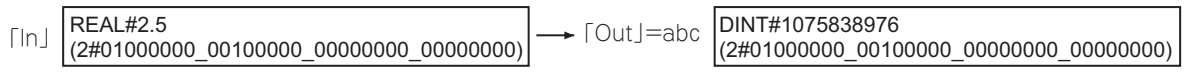
CopyRealToNum 命令で、「In」=REAL#2.5 の場合の例を、以下に示します。

LD



ST

```
abc:=CopyRealToNum(REAL#2.5);
```

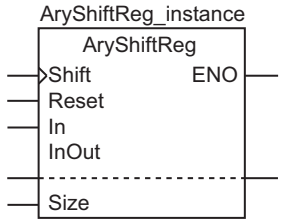

「In」
REAL#2.5
(2#01000000_00100000_00000000_00000000)
「Out」=abc
DINT#1075838976
(2#01000000_00100000_00000000_00000000)

シフト命令

命令	名称	ページ
AryShiftReg	シフトレジスタ	P.2-390
AryShiftRegLR	左右シフトレジスタ	P.2-392
ArySHL / ArySHR	配列の N 要素左シフト／配列の N 要素右シフト	P.2-395
SHL / SHR	N ビット左シフト／N ビット右シフト	P.2-398
NSHLC / NSHRC	N ビットデータ N ビット左シフト CY 付き／N ビットデータ N ビット右シフト CY 付き	P.2-400
ROL / ROR	N ビット左回転グループ／N ビット右回転グループ	P.2-402

AryShiftReg

配列要素からなるビット列全体を 1 ビット左シフトし、最下位ビットに入力値を挿入します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryShiftReg	シフトレジスタ	FB		AryShiftReg_instance(Shift, Reset, In, InOut, Size);

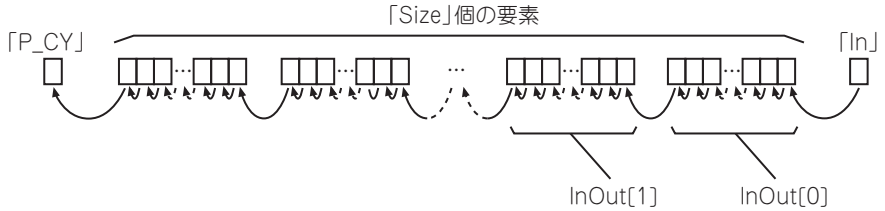
変数

	名称	入／出力	内容	有効範囲	単位	初期値
Shift	シフト	入力	FALSE→TRUE 時にシフト実行	データ型に従う	–	FALSE
Reset	リセット		TRUE: リセット実行			
In	入力値		InOut[]の最下位ビットに挿入する値			
Size	ビット列配列の要素数		InOut[]内の、シフトレジスタに使用する要素数			1
InOut[] 配列	ビット列配列	入出力	ビット列配列	データ型に従う	–	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift	○																			
Reset	○																			
In	○																			
Size							○													
InOut[] 配列	○	○	○	○	○															

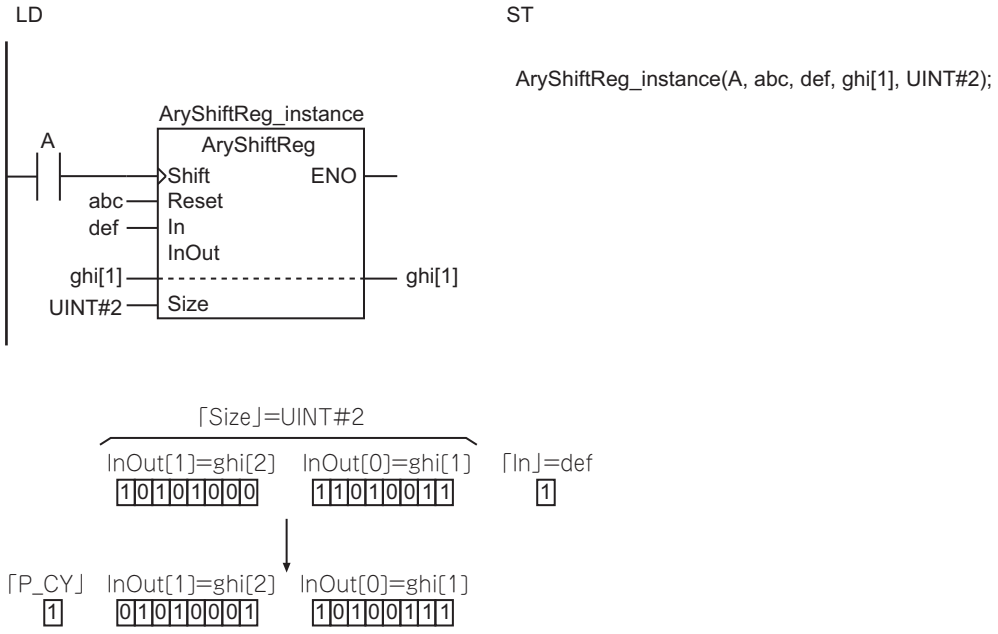
機能

「Shift」が FALSE→TRUE 時、ビット列配列 InOut[]の InOut[0]から始まる「Size」個の配列要素を、左方向（上位方向）へ 1 ビットシフトします。
最下位ビットには、入力値「In」を挿入します。ビット列配列からあふれた最上位ビットは、キャリー(CY)フラグ「P_CY」に出力します。



「Reset」が TRUE のとき、InOut[0]から始まる「Size」個の要素のすべてのビットとキャリー(CY)フラグに、FALSE をセットします。

InOut[]が BYTE 型配列、「Size」=UINT#2 の場合の例を、以下に示します。



関連するシステム定義変数

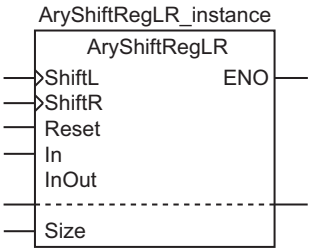
変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	キャリーフラグに格納されている値

使用上の注意

- 「Reset」が TRUE のときは、「Shift」が FALSE→TRUE となっても、シフトは実行されません。
- 「ENO」が TRUE になるのは、「Shift」が FALSE→TRUE となり、シフト動作が正常に行われたとき、または「Reset」が TRUE になり、リセット動作が正常に行われているときです。
- 「Size」の値が「0」の場合、InOut[]は変化しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が、InOut[]の配列領域を超える場合。

AryShiftRegLR

配列要素からなるビット列全体を左（または右）方向に 1 ビットシフトし、最下位（または最上位）ビットに入力値を挿入します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryShiftRegLR	左右シフトレジスタ	FB		AryShiftRegLR_instance(ShiftL, ShiftR, Reset, In, InOut, Size);

変数

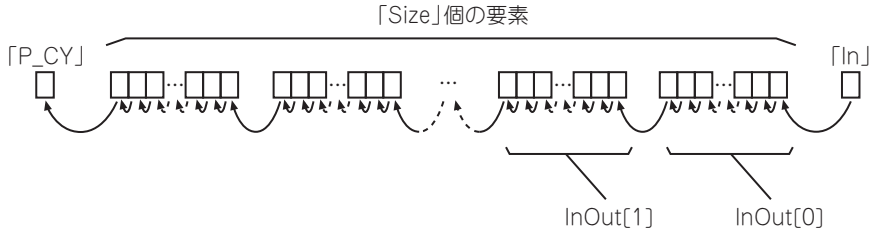
	名称	入／出力	内容	有効範囲	単位	初期値
ShiftL	左シフト	入力	FALSE→TRUE 時に左シフト実行	データ型に従う	−	FALSE
ShiftR	右シフト		FALSE→TRUE 時に右シフト実行			
Reset	リセット		TRUE: リセット実行			
In	入力値		InOut[]の最下位ビットまたは最上位ビットに挿入する値			
Size	ビット列配列の要素数		InOut[]内の、シフトレジスタに使用する要素数			1
InOut[] 配列	ビット列配列	入出力	ビット列配列	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ShiftL	○																			
ShiftR	○																			
Reset	○																			
In	○																			
Size							○													
InOut[] 配列	○	○	○	○	○															

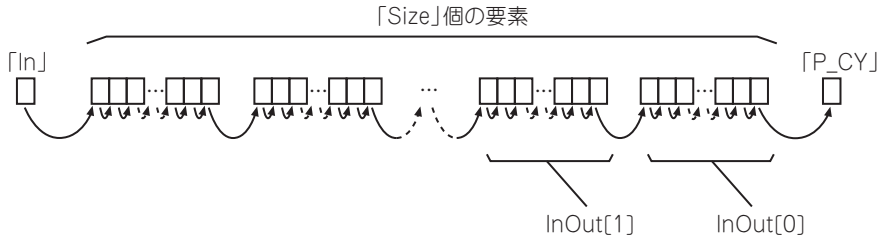
機能

「ShiftL」が FALSE→TRUE 時、ビット列配列 InOut[]の InOut[0]からの「Size」個の配列要素を、左方向へ 1 ビットシフトします。
最下位ビットには、入力値「In」を挿入します。

ビット列配列からあふれた最上位ビットは、キャリー(CY)フラグ「P_CY」に出力します。

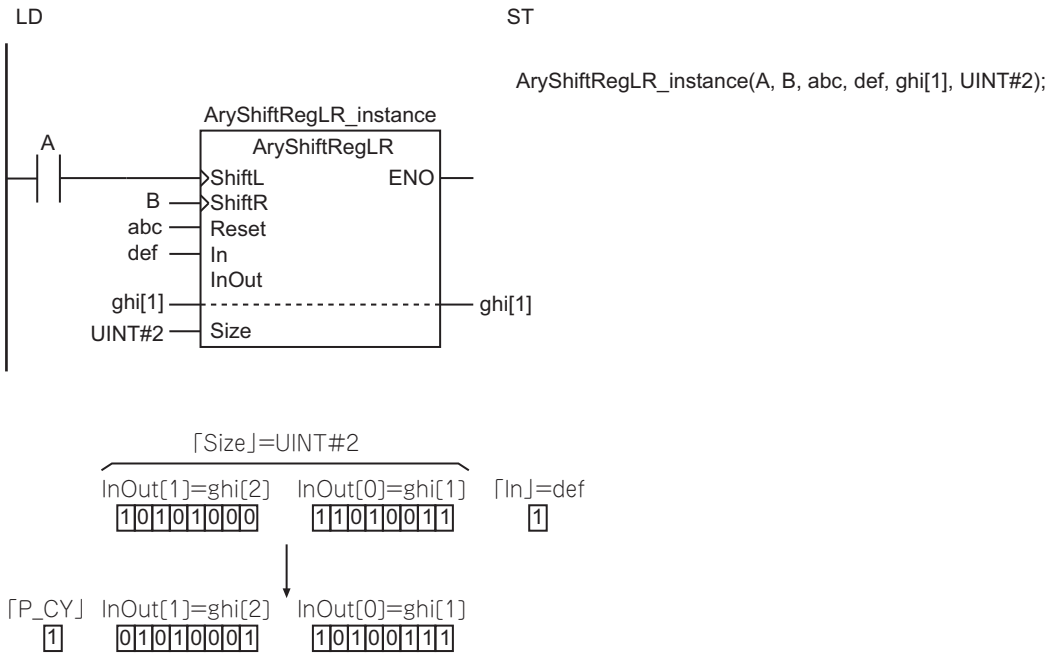


「ShiftR」が FALSE→TRUE 時には、右方向へ 1 ビットシフトし、最上位ビットに「In」を挿入します。
ビット列配列からあふれた最下位ビットを、キャリー(CY)フラグ「P_CY」に出力します。



「Reset」が TRUE のとき、InOut[0]以降の「Size」個の要素のすべてのビットと「P_CY」に、FALSE をセットします。

「InOut」が BYTE 型配列、「Size」=UINT#2 で、「ShiftL」が FALSE→TRUE の場合の例を、以下に示します。



関連するシステム定義変数

変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	キャリーフラグに格納されている値

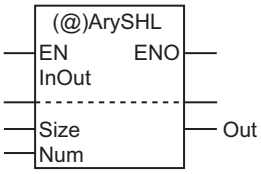
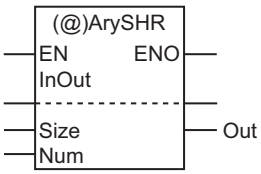
使用上の注意

- 「Reset」が TRUE のときは、「ShiftL」や「ShiftR」が FALSE→TRUE となっても、シフトは実行されません。
- 「ShiftL」と「ShiftR」が同時に FALSE→TRUE となった場合、シフトは実行されません。
- 「ENO」が TRUE になるのは、「ShiftL」または「ShiftR」が FALSE→TRUE となり、シフト動作が正常に行われたとき、または「Reset」が TRUE になり、リセット動作が正常に行われているときです。
- 「Size」の値が「0」の場合、InOut[]は変化しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が、InOut[]の配列領域を超える場合。

ArySHL / ArySHR

配列要素を、複数要素分シフトします。

ArySHL ： 左方向（上位方向）にシフトします。
ArySHR ： 右方向（下位方向）にシフトします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySHL	配列の N 要素左シフト	FUN		ArySHL(InOut, Size, Num);
ArySHR	配列の N 要素右シフト	FUN		ArySHR(InOut, Size, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	シフト対象の要素数	入力	シフト対象の要素数	データ型に従う	−	1
Num	シフト要素数		シフトさせる要素数			
InOut[] 配列	シフト対象配列	入出力	シフト対象配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Num							○													
InOut[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	構造体を要素とする配列も指定可能																			
Out	○																			

機能

シフト対象配列 InOut[]の、上位「Size」個の要素に対して、シフト要素数「Num」個分のシフトを行います。
シフトによりあふれた値は破棄します。
また空き要素には、InOut[]のデータ型のデフォルトの初期値を格納します。
InOut[]が、構造体を要素とする配列の場合は、各要素の構造体のすべてのメンバが初期化されます。

各データ型のデフォルトの初期値は、以下のとおりです。

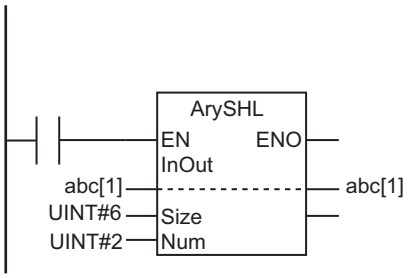
データ型	初期値
BOOL	FALSE
BYTE, WORD, DWORD, LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

ArySHL

左方向（配列の上位方向）にシフトします。

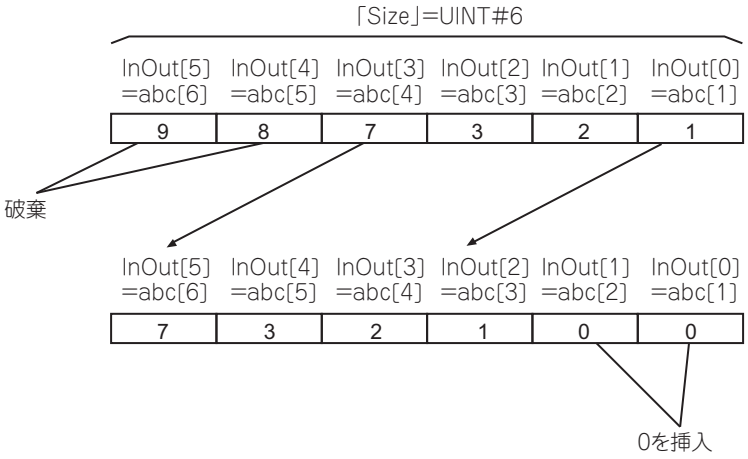
ArySHL 命令で、「Size」=UINT#6、「Num」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
ArySHL(abc[1], UINT#6, UINT#2);
```



ArySHR

右方向（配列の下位方向）にシフトします。

参考

InOut[]が BOOL 型の場合、「Size」ビットのビット列を「Num」ビット分シフトするのと同じことになります。

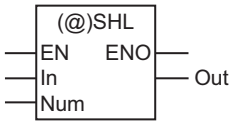
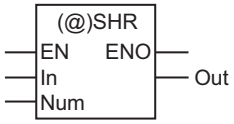
使用上の注意

- 「Num」の値が「0」の場合、シフト動作は行われません。
- 「Num」の値が「Size」より大きい場合、InOut[0]から InOut[「Size」-1]のすべての値が、初期化されます。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が、InOut[]の配列領域を超える場合。

SHL / SHR

ビット列を複数ビットシフトします。

SHL ： 左方向（上位方向）にシフトします。
SHR ： 右方向（下位方向）にシフトします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SHL	N ビット左シフト	FUN		Out:=SHL(In, Num);
SHR	N ビット右シフト	FUN		Out:=SHR(In, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	シフト対象	入力	シフト対象	データ型に従う	－	*1
Num *2	シフト数		シフトさせるビット数	0～「In」のビット数	ビット	1
Out	処理結果	出力	処理結果	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Num」を「N」と記述することができます。
たとえば、Out:=SHL(In:=BYTE#16#89, N:=ULINT#2);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Num									○											
Out	「In」と同じデータ型																			

機能

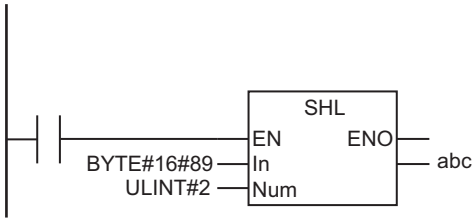
ビット列型のシフト対象「In」を、シフト数「Num」のビット数だけシフトします。
あふれたビットは破棄され、反対側から挿入されるビットには 0 が入ります。

SHL

右から左（下位ビットから上位ビット）方向にシフトします。

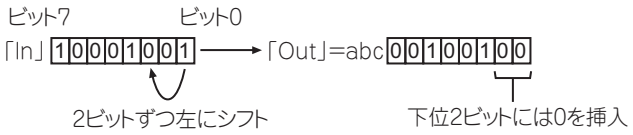
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=SHL(BYTE#16#89, ULINT#2);
```

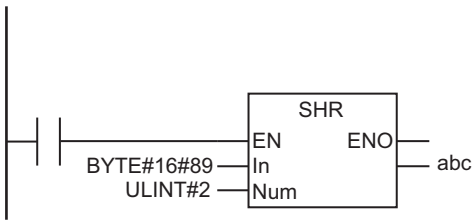


SHR

左から右（上位ビットから下位ビット）方向にシフトします。

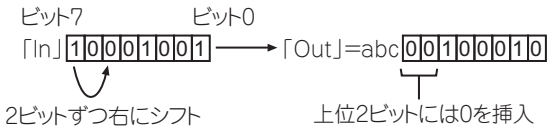
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=SHR(BYTE#16#89, ULINT#2);
```



参考

ROL/ROR 命令は、あふれたビットの値が、反対側から挿入されます。

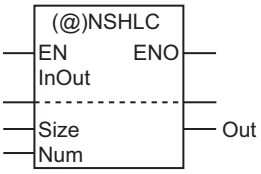
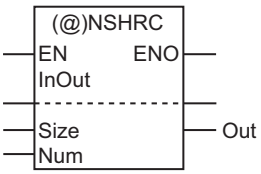
使用上の注意

- 「In」と「Out」は、同じデータ型にしてください。
- 「Num」の値が「0」の場合、異常にはならず、「In」の値がそのまま「Out」に代入されます。
- 「Num」の値が「In」のビット数より大きい場合、異常にはならず、「Out」の値は 16#0 になります。

NSHLC / NSHRC

ビット列配列を、キャリー(CY)フラグ付きで複数ビット、シフトします。

NSHLC ： 左方向（上位方向）にシフトします。
NSHRC ： 右方向（下位方向）にシフトします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NSHLC	N ビットデータ N ビット左シフト CY 付き	FUN		NSHLC(InOut, Size, Num);
NSHRC	N ビットデータ N ビット右シフト CY 付き	FUN		NSHRC(InOut, Size, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	シフト対象のビット数	入力	シフト対象のビット数	データ型に従う	ビット	1
Num	シフトビット数		シフトさせるビット数			
InOut[] 配列	シフト対象配列	入出力	シフト対象のビット列配列	データ型に従う	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size						○														
Num						○														
InOut[] 配列	○	○	○	○	○															
Out	○																			

機能

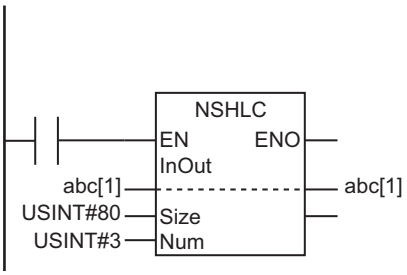
シフト対象配列 InOut[]の InOut[0]からの「Size」個の配列要素に対して、「Num」ビット分シフトします。
あふれたビットの最後の値は、キャリー(CY)フラグに出力します。反対側から挿入されるビットには 0 が入ります。

NSHLC

左方向（配列の下位から上位、および下位ビットから上位ビット）にシフトします。

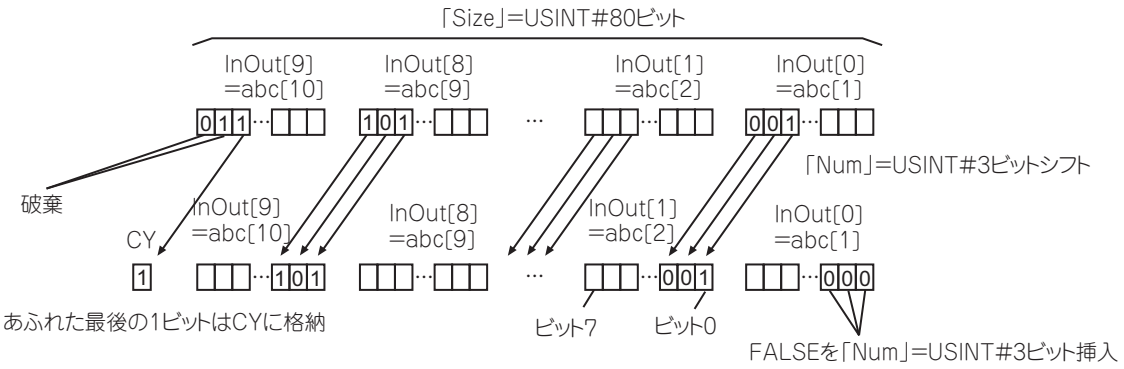
NSHLC 命令で、InOut[]が BYTE 型配列、「Size」= USINT#80、「Num」=USINT#3 の場合の例を、以下に示します。

LD



ST

```
NSHLC(abc[1], USINT#80, USINT#3);
```



NSHRC

右方向（配列の上位から下位、および上位ビットから下位ビット）にシフトします。

関連するシステム定義変数

変数名	名称	データ型	内容
P_CY	キャリー(CY)フラグ	BOOL	キャリーフラグに格納されている値

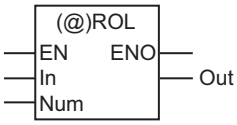
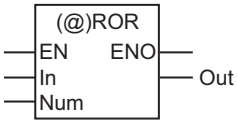
使用上の注意

- 「Num」の値が「0」の場合、シフト動作は行われません。
- 「Num」の値が「Size」より大きい場合、InOut[0]のビット 0 から「Size」ビット分の値が FALSE になります。また、キャリー(CY)フラグの値も FALSE になります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が、InOut[]の配列領域を超える場合。

ROL / ROR

ビット列を複数ビット回転します。

ROL ： 左方向（上位方向）に回転します。
ROR ： 右方向（下位方向）に回転します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ROL	N ビット左回転	FUN		Out:=ROL(In, Num);
ROR	N ビット右回転	FUN		Out:=ROR(In, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	回転対象	入力	回転対象	データ型に従う	－	*1
Num *2	回転数		回転させるビット数	0～「In」のビット数	ビット	1
Out	処理結果	出力	処理結果	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 変数名とパラメータ名の対応を明示する ST 表現をする場合、「Num」を「N」と記述することができます。
たとえば、Out:=ROL(In:=BYTE#16#89, N:=ULINT#2);と記述できます。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Num									○											
Out	「In」と同じデータ型																			

機能

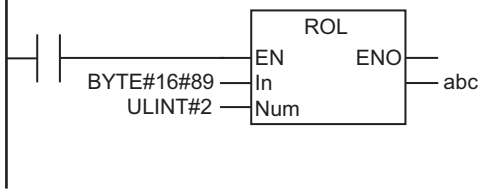
ビット列型の回転対象「In」を、回転数「Num」のビット数だけ回転します。あふれたビットは、反対側から挿入します。

ROL

右から左（下位ビットから上位ビット）方向に回転します。

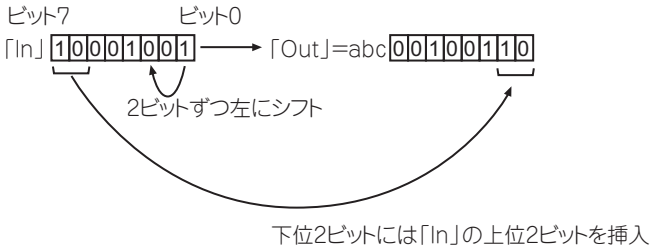
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=ROL(BYTE#16#89, ULINT#2);
```

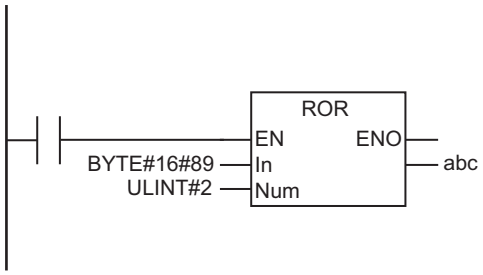


ROR

左から右（上位ビットから下位ビット）方向に回転します。

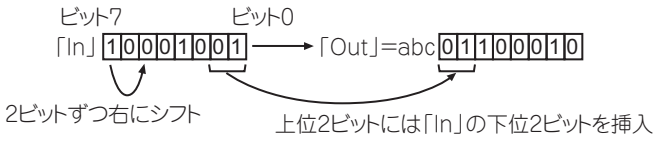
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=ROR(BYTE#16#89, ULINT#2);
```



参考

SHL/SHR 命令は、あふれたビットを破棄し、反対側から挿入されるビットに 0 を入れます。

使用上の注意

- 「In」と「Out」は、同じデータ型にしてください。
- 「Num」の値が「0」の場合、異常にはならず、「In」の値がそのまま「Out」に代入されます。
- 「Num」の値が「In」のビット数を超える場合、異常にはならず、「Num」の値だけ回転します。たとえば、「In」が WORD 型の場合、「Num」の値が USINT#1 の場合と USINT#17 の場合とでは、「Out」の値は同じになります。

データ変換命令

命令	名称	ページ
Swap	バイトスワップ	P.2-407
Neg	符号反転	P.2-408
Decoder	ビットデコーダ	P.2-410
Encoder	ビットエンコーダ	P.2-413
BitCnt	ビットカウンタ	P.2-415
ColmToLine_**	ビット列→ビット行変換グループ	P.2-416
LineToColm	ビット行→ビット列変換	P.2-418
Gray	グレイコード変換	P.2-420
UTF8ToSJIS	文字コード変換（UTF-8→SJIS）	P.2-425
SJISToUTF8	文字コード変換（SJIS→UTF-8）	P.2-427
PWLApprox / PWLApproxNoLineChk	折れ線近似変換（折れ線データチェックあり）／折れ線近似変換（折れ線データチェックなし）	P.2-429
PWLLineChk	折れ線データチェック	P.2-434
MovingAverage	移動平均	P.2-437
DispartReal	実数の仮数・指数分離	P.2-443
UniteReal	仮数・指数の実数への結合	P.2-446
NumToDecString / NumToHexString	固定長 10 進文字列変換／固定長 16 進文字列変換	P.2-448
HexStringToNum_**	16 進文字列→数値変換グループ	P.2-451
FixNumToString	固定小数点数→文字列変換	P.2-453
StringToFixNum	文字列→固定小数点数変換	P.2-455
DtToString	日時→文字列変換	P.2-458
DateToString	日付→文字列変換	P.2-460

命令	名称	ページ
TodToString	時刻→文字列変換	P.2-461
GrayToBin_** / BinToGray_**	グレイコード→BIN コード変換グループ／BIN コード→グレイコード変換グループ	P.2-463
StringToAry	文字列→配列変換	P.2-466
AryToString	配列→文字列変換	P.2-468
DispartDigit	4 ビット分離	P.2-470
UniteDigit_**	4 ビット結合グループ	P.2-472
Dispart8Bit	バイト単位データ分離	P.2-474
Unite8Bit_**	バイト単位データ結合グループ	P.2-476
ToAryByte	バイト配列への変換	P.2-478
AryByteTo	バイト配列からの変換	P.2-484
SizeOfAry	配列要素数取得	P.2-489
PackWord	2 バイト結合	P.2-491
PackDword	4 バイト結合	P.2-493
LOWER_BOUND / UPPER_BOUND	配列開始番号取得／配列終了番号取得	P.2-495

Swap

16 ビットの値の上位バイトと下位バイトを交換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Swap	バイトスワップ	FUN	(@)Swap EN ENO In Out	Out:=Swap(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0
Out	変換結果	出力	変換結果	データ型に従う	−	−

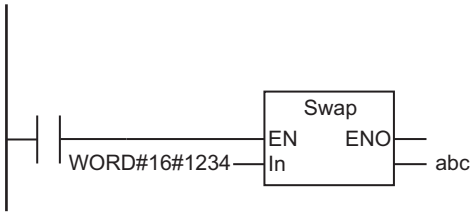
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			○																	
Out			○																	

機能

変換対象「In」の上位バイトと下位バイトを交換して、変換結果「Out」に代入します。

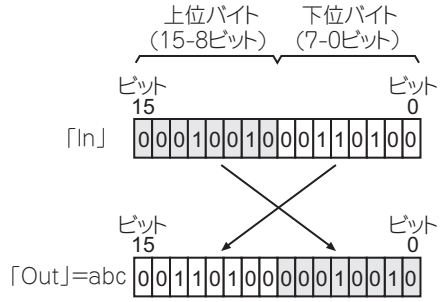
「In」=WORD#16#1234 の場合の例を、以下に示します。

LD



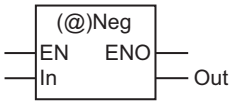
ST

```
abc:=Swap(WORD#16#1234);
```



Neg

数値の符号を反転します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Neg	符号反転	FUN		Out:=Neg(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	－	*1
Out	変換結果	出力	変換結果	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Out						○	○	○	○	○	○	○	○	○	○					

機能

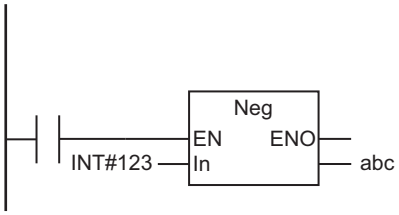
変換対象「In」の符号を反転します。

処理内容は、「In」のデータ型によって、以下のように異なります。

「In」のデータ型	「Out」の値
符号付き整数 SINT, INT, DINT, LINT	「In」の全ビットを反転して、1 を加算 （「In」×（-1）と同じ意味）
符号なし整数 USINT, UINT, UDINT, ULINT	「In」の全ビットを反転して、1 を加算
実数 REAL, LREAL	「In」×（-1）

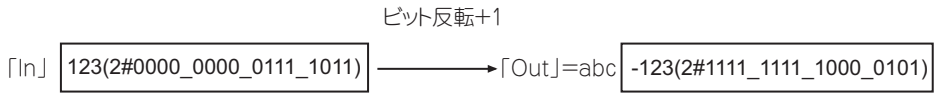
「In」=INT#123 の場合の例を、以下に示します。

LD

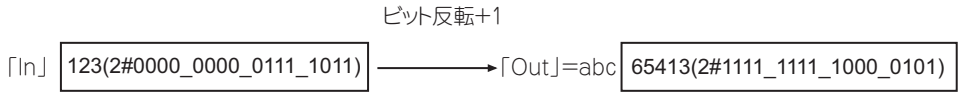


ST

```
abc:=Neg(INT#123);
```

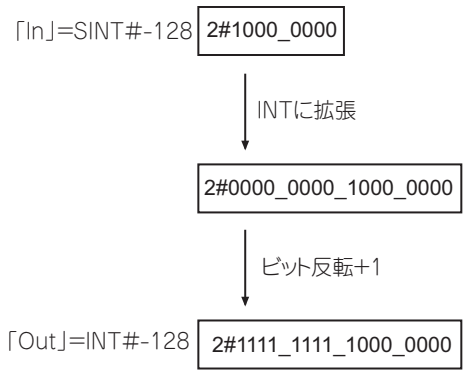


「In」=UINT#123 の場合の例を、以下に示します。



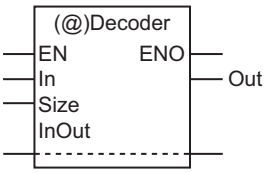
使用上の注意

「In」と「Out」のデータ型が異なる場合は、「Out」の有効範囲が「In」の有効範囲を包含するようにしてください。包含していない場合、異常にはならず、「Out」の値は不正値になります。
たとえば、「In」の値が SINT#-128、「Out」のデータ型が INT の場合、「Out」の値は INT#128 にはならず、INT#-128 になります。



Decoder

最大 256 ビットからなる配列要素の、指定した 1 ビットを TRUE、その他のビットを FALSE にセットします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Decoder	ビットデコーダ	FUN		Decoder(In, Size, InOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換ビット位置	入力	変換するビット位置	データ型に従う	−	0
Size	変換ビット数		変換するビット数	0～8	ビット	1
InOut[] 配列	変換対象配列	入出力	変換対象配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

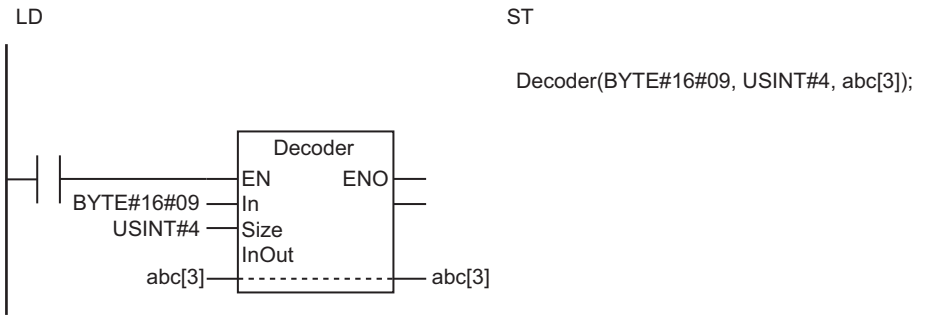
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○																		
Size						○														
InOut[] 配列	○	○	○	○	○															
Out	○																			

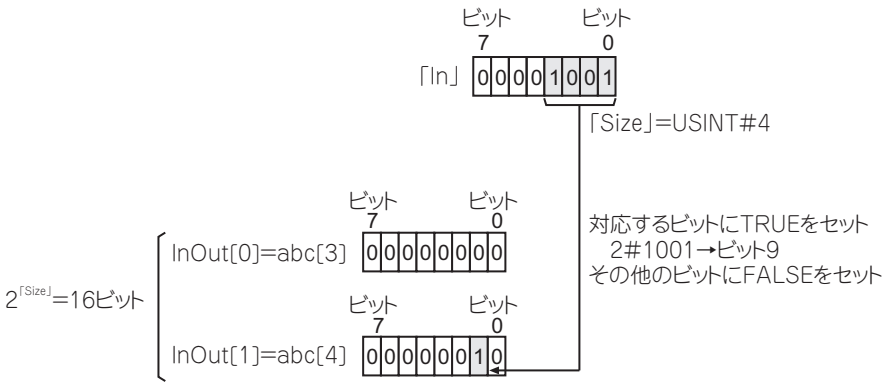
機能

変換対象配列 InOut[]の、InOut[0]から始まる、2「Size」ビットのサイズの配列要素の指定ビットを TRUE にします。それ以外のビットは、FALSE にします。
値を TRUE にするビット位置は、変換ビット位置「In」の、下位「Size」ビットで指定します。
InOut[]に渡す入出力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

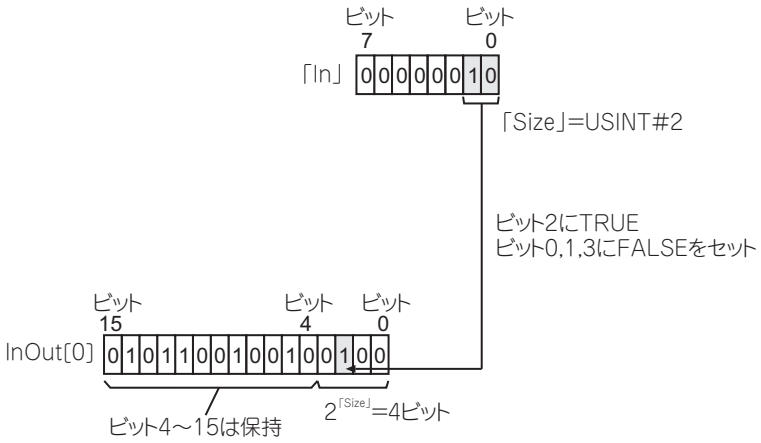
「In」=BYTE#16#09、「Size」=USINT#4、InOut[]が BYTE 型配列の場合を例にとって、説明します。
変換ビット位置「In」の値 16#09 は、10 進数に直すと 9 です。したがって、InOut[]の最下位から 9 番めのビットが TRUE になり、それ以外のビットが FALSE になります。

InOut[]は BYTE 型配列なので、最下位から 9 番めのビットは InOut[1]のビット 1 です。したがって、InOut[1]のビット 1 が TRUE になり、InOut[0]のすべてのビットと InOut[1]のビット 1 以外のビットが FALSE になります。





InOut[]の要素のビット数が、「Size」で示されるビット数より大きい場合、残りのビットの値は保持されます。「In」=BYTE#16#02、「Size」=USINT#2、InOut[]が WORD 型配列の場合を例にとって、説明します。
「Size」=USINT#2 なので、InOut[0]の下位 4 ビットに値がセットされます。そして、InOut[0]の残りのビットであるビット 4～15 の値は保持されます。



参考

最大 256 ビットからなる配列要素のうち、TRUE になっているビットの位置を求める場合は、「Encoder (P.2-413)」命令を使用してください。

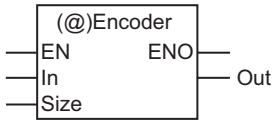
使用上の注意

- 「Size」の値が「0」の場合、InOut[]のすべてのビットが、FALSE になります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。

- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」が有効範囲外の場合。
  b) 2「Size」の値が、InOut[]の配列要素のビット数を超える場合。

Encoder

最大 256 ビットからなる配列要素のうちの、値が TRUE であるビットの位置を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Encoder	ビットエンコーダ	FUN		Out:=Encoder(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	変換対象配列	入力	変換対象配列	データ型に従う	−	*1
Size	変換ビット数		変換するビット数	0〜8	ビット	1
Out	変換結果	出力	変換結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	○	○	○	○	○															
Size						○														
Out		○																		

機能

変換対象配列 In[]の、任意の範囲にある、値が TRUE であるビットの位置を求めます。

ビット位置を求めるのは、In[0]から始まる 2「Size」ビットの範囲です。この範囲内にある、値が TRUE であるビットの位置を 2 進数で表し、変換結果「Out」の下位「Size」ビットに格納します。そして、「Out」の残りのビットには FALSE を格納します。

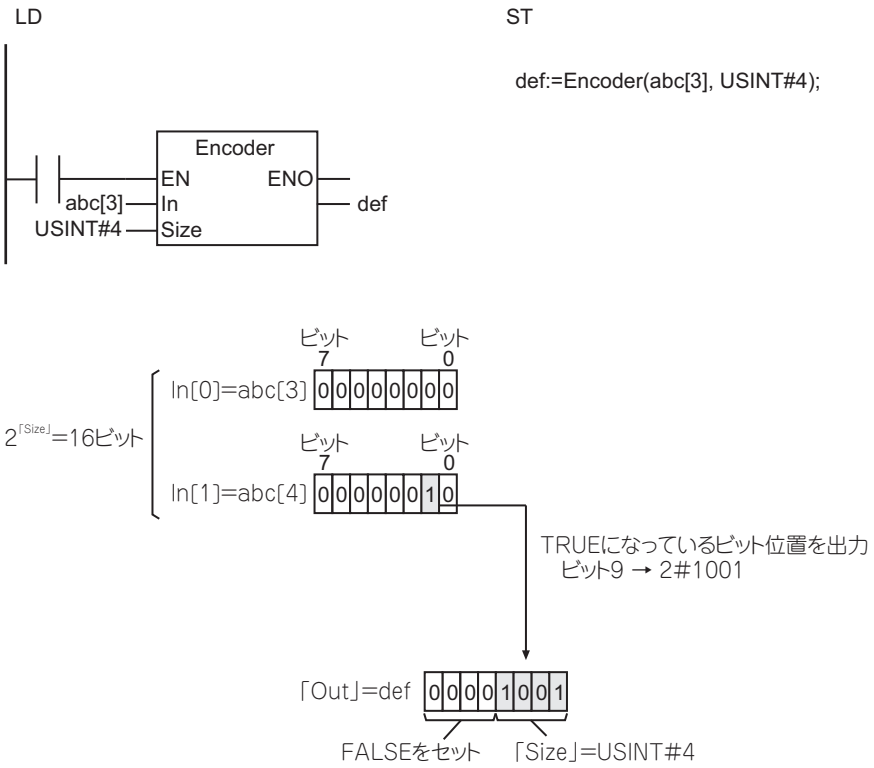
もし、指定範囲内に、値が TRUE のビットが複数ある場合は、それらの中から最上位にあるビットの位置を求めます。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

「Size」=USINT#4、In[]が BYTE 型配列の場合の例を示します。

「Size」=USINT#4 なので、In[0]から始まる 2^4=16 ビットが、TRUE のビット位置を求める対象範囲になります。下図では、この範囲の 9 ビットめに、値が TRUE のビットがあります。

「Size」=USINT#4 なので、「Out」の下位 4 ビットに、求めた 9=2#1001 を格納します。「Out」の上位 4 ビットには FALSE を格納します。



参考

最大 256 ビットからなる配列要素のうち、1 ビットを TRUE、その他のビットを FALSE にセットする場合は、「Decoder (P.2-410)」命令を使用してください。

使用上の注意

- 「Size」の値が「0」の場合、「Out」のすべてのビットが、FALSE になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「Size」が有効範囲外の場合。
  b）2「Size」の値が、In[]の配列要素のビット数を超える場合。
  c）In[]のビットのうち、「Size」で指定されたすべての値が FALSE の場合。

BitCnt

ビット列内の、値が TRUE のビットの総数をカウントします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BitCnt	ビットカウンタ	FUN	(@)BitCnt EN ENO In Out	Out:=BitCnt(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	カウント対象	入力	値が TRUE のビットをカウントする対象	データ型に従う	−	*1
Out	カウント結果	出力	値が TRUE のビットの数	0～「In」のビット数	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

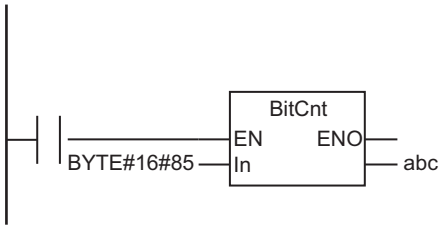
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○														

機能

カウント対象「In」内の、値が TRUE であるビットの総数をカウントします。

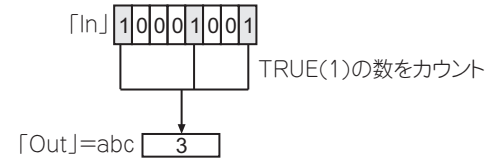
「In」が BYTE 型で、値が BYTE#16#85 の場合の例を、以下に示します。

LD



ST

```
abc:=BitCnt(BYTE#16#85);
```



ColmToLine_**

各配列要素の、指定した位置にあるビットの値を抽出し、ビット列として出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ColmToLine_**	ビット列→ビット行変換グループ	FUN	(@)ColmToLine_** EN ENO In Out Size Pos ただし、“**”はビット列のデータ型名	Out:=ColmToLine_**(In, Size, Pos); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	変換対象配列	入力	変換対象配列	データ型に従う	−	*1
Size	変換対象の要素数		In[]の中で、変換対象となる要素の数	0～「Out」のビット数		1
Pos	変換ビット位置		変換するビット位置	0～In[]のビット数-1		0
Out	変換結果	出力	変換結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○	○	○	○															
Size						○														
Pos						○														
Out		○	○	○	○															

機能

配列の各要素の、指定した位置のビットの値を抽出し、それを並べてビット列として出力します。

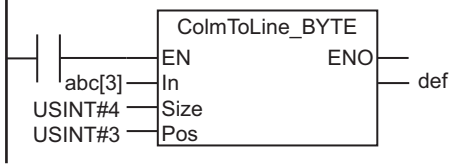
まず、変換対象配列 In[]の In[0]から始まる「Size」個の要素を抽出します。
つぎに、それらの「Pos」ビットめの値だけを抽出します。
それらを並べて「Size」ビットのビット列にし、変換結果「Out」の最下位ビット位置に格納します。
そして、「Out」の残りのビットには、FALSE を格納します。

命令名称は、「Out」のデータ型によって、異なります。たとえば、「Out」が BYTE 型の場合、命令名称は ColmToLine_BYTE となります。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

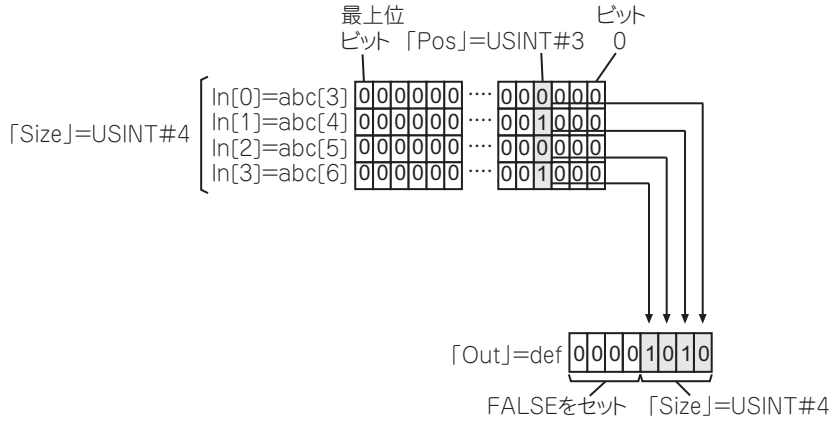
ColmToLine_BYTE 命令で、「Pos」=USINT#3、「Size」=USINT#4 の場合の例を、以下に示します。

LD



ST

```
def:=ColmToLine_BYTE(abc[3], USINT#4, USINT#3);
```



参考

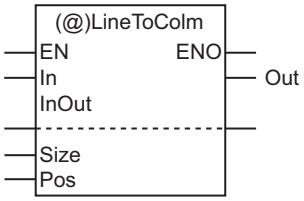
各配列要素の、指定したビット位置にビット列を出力する場合は、「LineToColm (P.2-418)」命令を使用してください。

使用上の注意

- 「Size」の値が「0」の場合、「Out」のすべてのビットが、FALSE になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が有効範囲外の場合。
  b)「Pos」の値が有効範囲外の場合。
  c)「Size」の値が、In[]の配列領域を超える場合。

LineToColm

ビット列を分解して、配列要素の指定したビット位置に出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LineToColm	ビット行→ビット列変換	FUN		LineToColm(In, InOut, Size, Pos);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	*1
Size	変換結果の要素数		変換結果の要素数	0～「In」のビット数		1
Pos	変換ビット位置		変換するビット位置	0～InOut[]のビット数-1		0
InOut[] 配列	変換結果配列	入出力	変換結果	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Size						○														
Pos						○														
InOut[] 配列		○	○	○	○															
Out	○																			

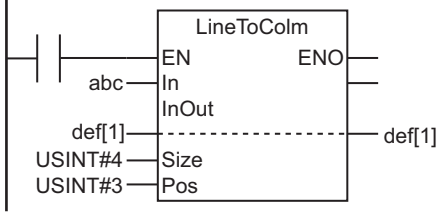
機能

ビット列を分解して、配列要素の、指定したビット位置に出力します。

まず、変換対象「In」の最下位ビットから「Size」ビット分を抽出し、1 ビットずつに分解します。つぎに、それらを、変換結果配列 InOut[]の InOut[0]から始まる各要素の「Pos」ビットめに格納します。格納される配列要素数は、「Size」個になります。
値が格納されるビット以外のビットは、値を保持します。

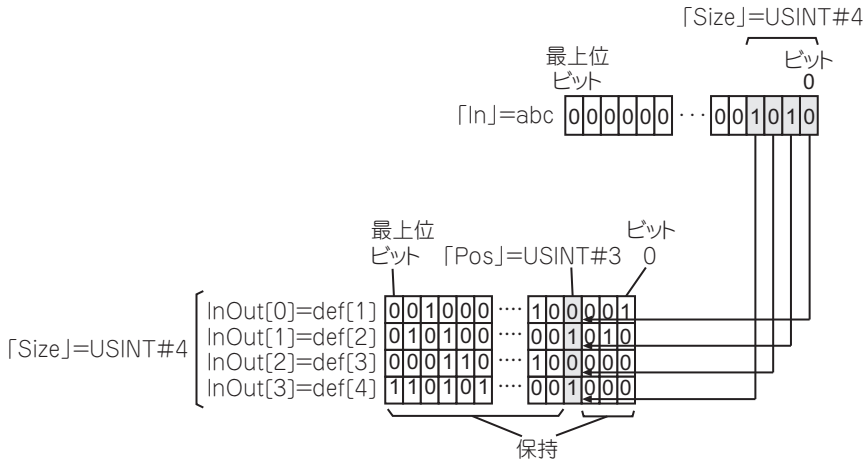
「Pos」=USINT#3、「Size」=USINT#4 の場合の例を、以下に示します。

LD



ST

```
LineToColm(abc, def[1], USINT#4, USINT#3);
```



参考

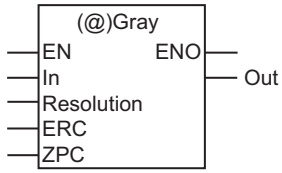
各配列要素の、指定した位置のビットの値を抽出して、ビット列に出力する場合は、「ColmToLine_** (P.2-416)」命令を使用してください。

使用上の注意

- 「Size」の値が「0」の場合、InOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が有効範囲外の場合。
  b)「Pos」の値が有効範囲外の場合。
  c)「Size」の値が、InOut[]の配列領域を超える場合。

Gray

グレイコードを角度に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Gray	グレイコード変換	FUN		Out:=Gray(In, Resolution, ERC, ZPC);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象のグレイコード	データ型に従う	－	0
Resolution	分解能		分解能	_R256, _R1B ～ _R15B, _R360, _R720, _R1024		_R256
ERC	エンコーダ余り補正値		エンコーダ余り補正値	0～「Resolution」の分解能		0
ZPC	原点補正値		原点補正値			
Out	変換結果	出力	変換結果	*1	°	－

*1. 0 ～ 3.59999999999999e+2

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			○																	
Resolution	列挙型_eGRY_RESOLUTION　列挙子は「機能 (P.2-420)」参照																			
ERC							○													
ZPC							○													
Out															○					

機能

グレイコードで表現されたロータリエンコーダの出力値「In」を角度値に変換します。変換結果「Out」の単位は、° です。

「Resolution」のデータ型は、列挙型_eGRY_RESOLUTION です。列挙子の意味は、以下のとおりです。

列挙子	意味
_R256	256
_R1B	1 ビット(2)
_R2B	2 ビット(4)
_R3B	3 ビット(8)

列挙子	意味
_R4B	4 ビット(16)
_R5B	5 ビット(32)
_R6B	6 ビット(64)
_R7B	7 ビット(128)
_R8B	8 ビット(256)
_R9B	9 ビット(512)
_R10B	10 ビット(1024)
_R11B	11 ビット(2048)
_R12B	12 ビット(4096)
_R13B	13 ビット(8192)
_R14B	14 ビット(16384)
_R15B	15 ビット(32768)
_R360	360
_R720	720
_R1024	1024

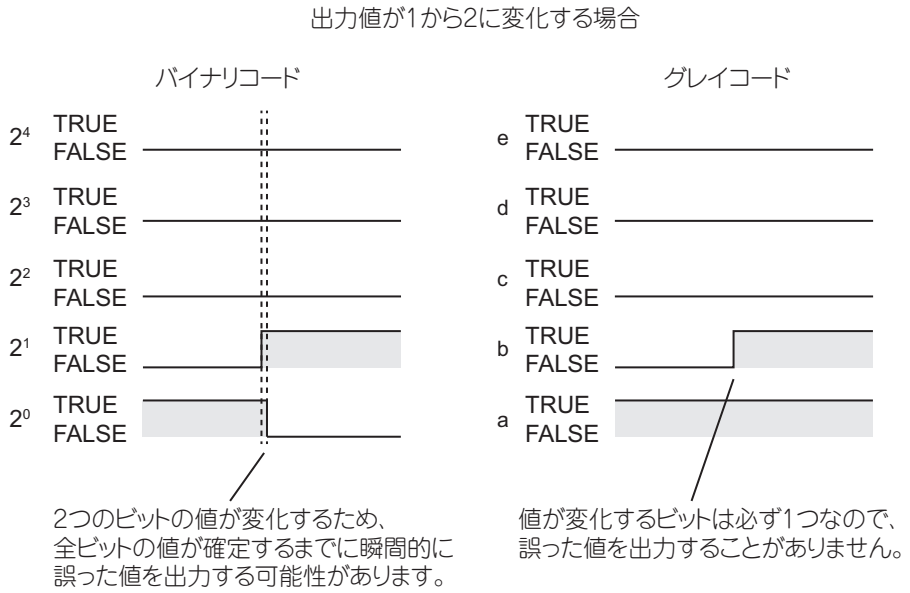
グレイコード

グレイコードとは、交番二進符号と呼ばれる 2 進数の一種です。
0 と 1、1 と 2 のように、差が 1 の数値のコードは、必ず 1 つのビットの値だけが異なるという特徴を持っています。
グレイコードは、絶対値エンコーダの出力などに使用されます。

以下に、4 ビットのバイナリコードとグレイコードを示します。

10 進数	バイナリコード				グレイコード			
	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

グレイコードを使用すると、エンコーダの出力値が 1 増加または減少するときに 1 つのビットの値だけしか変化しないので、瞬間的に誤った値を出力することが防げます。
グレイコードとバイナリコードの違いによる、エンコーダの出力値の変化の違いを、以下に示します。

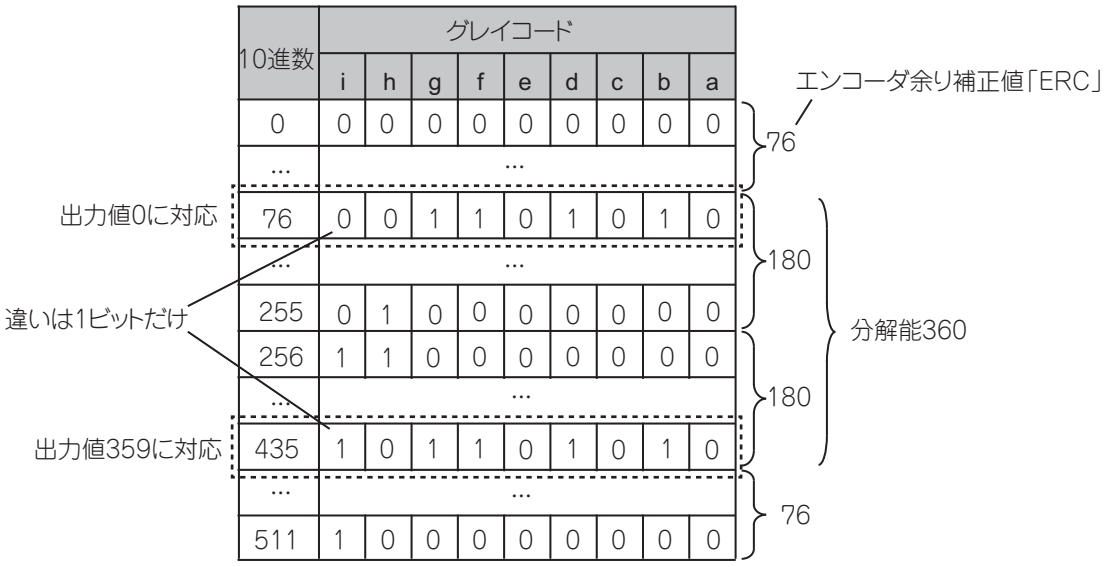


エンコーダ余り補正値「ERC」

「ERC」は、エンコーダの分解能が 2 のべき乗でない場合に、エンコーダの出力の最大値と最小値に対するグレイコードの違いが 1 ビットだけになるように、グレイコードの範囲を指定する補正値です。

たとえば、分解能が 360 の絶対値エンコーダを使用するとします。グレイコードには 9 ビットが使用されます。9 ビットで表現できる値の範囲は 0～511 です。この場合、0～511 の中心から前後 180 ずつの範囲、つまり 76～435 に対するグレイコードを使用します。これにより、出力値 0 を示すグレイコードは、001101010（10 進数の 76）、出力値 359 を示すグレイコードは、101101010（10 進数の 435）となり、両者の違いは 1 ビットだけになります。

この場合のエンコーダ余り補正値「ERC」の値は、76 になります。



原点補正値「ZPC」

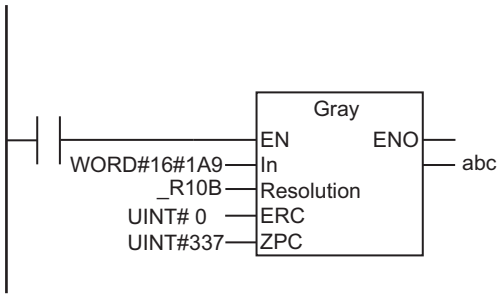
「ZPC」は、ロータリエンコーダの原点の角度をずらす場合に設定します。たとえば、分解能 256 のロータリエンコーダの原点を 90° ずらす場合、「ZPC」の値は、256×（90/360）=64 とします。

記述例

「In」=WORD#16#1A9、「Resolution」=_R10B、「ERC」=UINT#0、「ZPC」=UINT#337 の場合の例を、以下に示します。

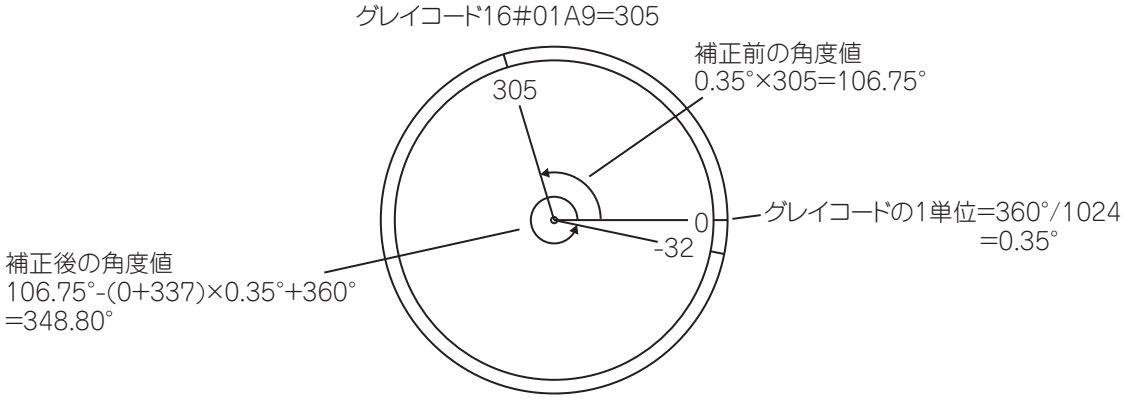
まず、分解能が 10 ビットなので、グレイコードの 1 単位は、360° /1024=0.35° となります。
グレイコード 16#01A9 に対応する 10 進数の値は、305 です。したがって、補正前の角度値は 0.35° ×305=106.75° となります。
「ERC」の値が 0、「ZPC」の値が 337 です。したがって、これらを補正した角度値は 106.75° -（0+337）×0.35° =-11.20° となります。
「Out」の値の範囲は 0 以上なので、-11.20° +360° =348.80° となります。「Out」の値は LREAL#348.8 となります。

LD



ST

```
abc:=Gray(WORD#16#1A9, _R10B, UINT#0, UINT#337);
```



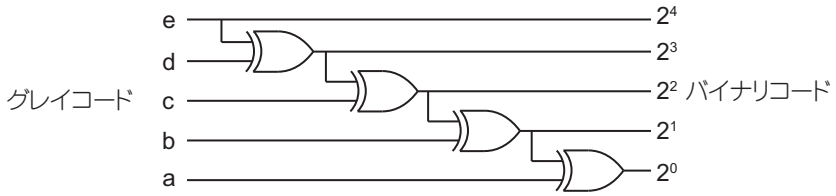
参考

「Resolution」や「ERC」に指定する値は、使用するロータリエンコーダの取扱説明書などを参照して決めてください。

グレイコードからバイナリコードへの変換

グレイコードからバイナリコードへは、つぎの処理で変換できます。図中の論理記号は、排他的論理和を示します。

5ビットの場合の変換回路



使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 「Resolution」の値が有効範囲外の場合。
- 「ERC」の値が、「Resolution」で指定された分解能以上の場合。
- 「ZPC」の値が、「Resolution」で指定された分解能以上の場合。
- 「In」をビット列に変換した値が、「ERC」の値より小さい場合。
- ビット列を「ERC」で補正した値が、「Resolution」で指定された分解能以上の場合。

UTF8ToSJIS

文字コード UTF-8 の文字列を、文字コード SJIS の BYTE 型配列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
UTF8ToSJIS	文字コード変換（UTF-8→SJIS）	FUN	(@)UTF8ToSJIS EN ENO In Out SJISCode	Out:=UTF8ToSJIS(In, SJISCode);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象文字列	入力	変換する文字列	データ型に従う	−	''
SJIS Code[] 配列	SJIS 配列	入出力	文字コード SJIS の配列	データ型に従う	−	−
Out	変換要素数	出力	SJISCode[]に格納した要素数	0～1985	−	−

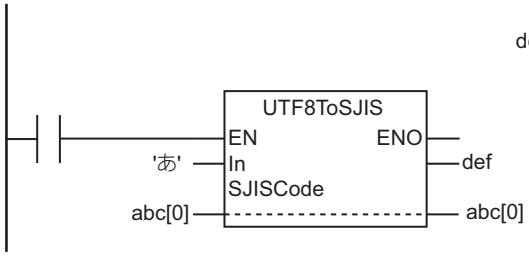
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
SJIS Code[] 配列		○																		
Out							○													

機能

文字コード UTF-8 の変換対象文字列「In」を、文字コード SJIS の BYTE 型配列 SJISCode[]に変換します。変換後データは 1 バイトずつに分割し、SJISCode[0]から順に格納します。
変換要素数「Out」には、SJISCode[]に格納した変換後データの要素数を格納します。

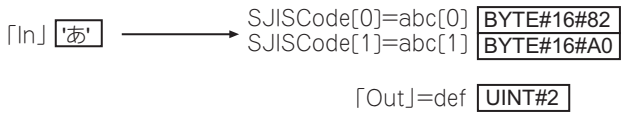
「In」='あ'の場合の例を、以下に示します。

LD



ST

```
def:=UTF8ToSJIS('あ', abc[0]);
```

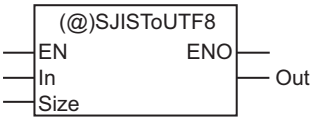


使用上の注意

- 「In」の終端の NULL 文字は、変換されません。また、変換要素数にもカウントされません。
- 「In」が NULL 文字だけの文字列の場合、「Out」の値は 0 となり、SJISCode[]は変化しません。
- SJISCode[]の、「Out」番め以降の要素は、変化しません。たとえば、変換要素数が 5 の場合、SJISCode[5]以降の要素は変化しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、SJISCode[]は変化しません。
  a）変換結果の要素数が、SJISCode[]に接続する出力パラメータの領域サイズを超える場合。
  b）「In」の内容が、変換不可能な文字の場合。

SJISToUTF8

文字コード SJIS の BYTE 型配列を、文字コード UTF-8 の文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SJISToUTF8	文字コード変換(SJIS→UTF-8)	FUN		Out:=SJISToUTF8(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	変換対象 SJIS 配列	入力	変換する文字コード SJIS の配列 *1	データ型に従う	−	*2
Size	SJIS 配列要素数		変換する In[]の要素数			−
Out	変換文字列	出力	文字コード UTF-8 に変換した文字列	データ型に従う	−	−

*1. NULL 文字(BYTE#16#00)を含む場合の最大要素数は、1986 です。NULL 文字を含まない場合の最大要素数は、1985 です。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

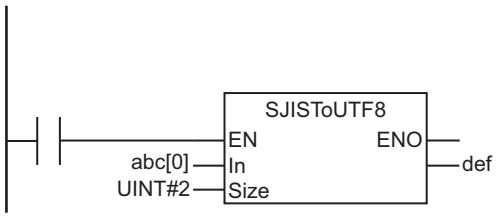
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○																		
Size							○													
Out																				○

機能

文字コード SJIS の BYTE 型配列 In[]の要素を、文字コード UTF-8 の文字列に変換します。
変換対象は、In[0]以降の「Size」個の要素です。ただし、途中に NULL 文字(BYTE#16#00)があった場合は、そこまでで変換を打ち切ります。
変換後の文字列は、変換文字列「Out」に格納します。「Out」の終端には、NULL 文字を付加します。

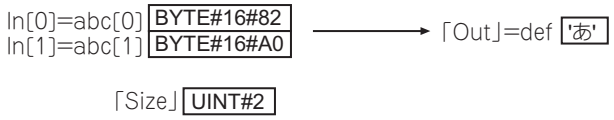
In[0]=BYTE#16#82、In[1]=BYTE#16#A0、「Size」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
def:=SJISToUTF8(abc[0], UINT#2);
```



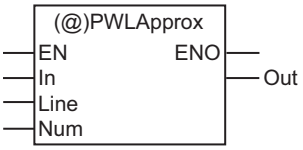
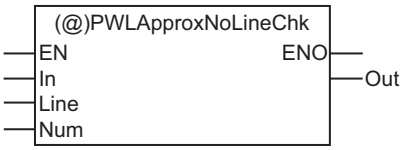
使用上の注意

- 「Size」の値が 0 の場合、「Out」の値は NULL 文字だけになります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「Size」の値が、In[]の配列の大きさを超える場合。
  b）In[]の内容が、変換不可能な文字の場合。

PWLApprox / PWLApproxNoLineChk

整数、実数の折れ線近似計算を行います。

PWLApprox　　：折れ線データが有効かどうか、チェックします。
PWLApproxNoLineChk　：折れ線データが有効かどうか、チェックしません。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PWLApprox	折れ線近似変換（折れ線チェックあり）	FUN		Out:=PWLApprox(In, Line, Num);
PWLApprox NoLineChk	折れ線近似変換（折れ線チェックなし）	FUN		Out:=PWLApproxNoLineChk(In, Line, Num);

変数

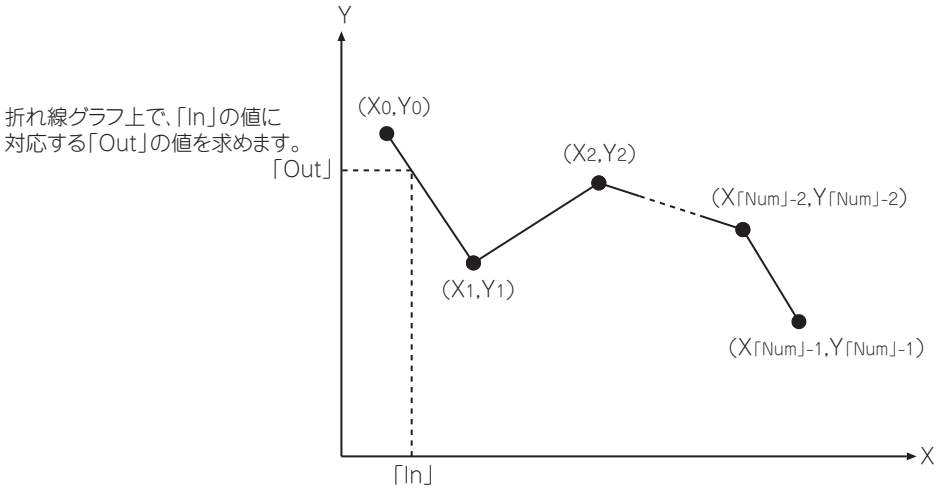
	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	－	*1
Line[] 配列	折れ線データ配列		折れ線データ配列			
Num	折れ線データの数		折れ線データの数			1
Out	変換結果	出力	変換結果	データ型に従う	－	－

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Line[] 配列	「In」と同じデータ型を要素とする配列																			
Num							○													
Out						○	○	○	○	○	○	○	○	○	○					

機能

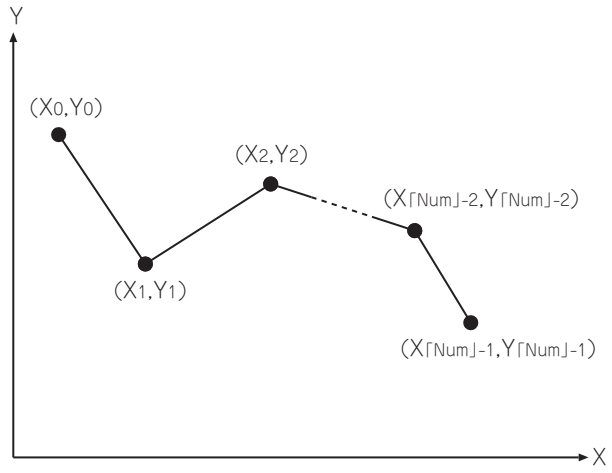
折れ線データ配列 Line[]の、Line[0,0]以降の（「Num」×2）個の要素からなる折れ線データにもとづいて、変換対象「In」を近似変換します。
図のように、折れ線データ上の X 座標「In」に対応する Y 座標が、変換結果「Out」に代入されます。



折れ線データ Line[]の要素と折れ線データの数「Num」

Line[]は 2 次元または 3 次元配列で、第 1 次元の要素数を 2 としてください。そして、図のように、折れ線データの各点の座標値(X_0, Y_0), (X_1, Y_1)・・・を、Line[]の各要素としてください。

折れ線データの数「Num」は、折れ線近似計算に用いる Line[]の要素数の 1/2 になります。



Line[]が2次元配列の場合

Line[0,0]	X_0
Line[0,1]	Y_0
Line[1,0]	X_1
Line[1,1]	Y_1
Line[2,0]	X_2
Line[2,1]	Y_2
:	:
Line[「Num」-1,0]	X「Num」-1
Line[「Num」-1,1]	Y「Num」-1

Line[]が3次元の場合

Line[0,0,0]	X_0
Line[0,0,1]	Y_0
Line[0,1,0]	X_1
Line[0,1,1]	Y_1
Line[0,2,0]	X_2
Line[0,2,1]	Y_2
:	:
Line[0,「Num」-1,0]	X「Num」-1
Line[0,「Num」-1,1]	Y「Num」-1

記述例

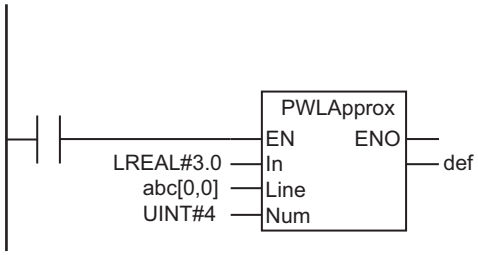
要素数 4 の折れ線データ配列 abc[]に基づいて、「In」=LREAL#3.0 を近似変換する場合の例を、以下に示します。「Num」= UINT#4 で、abc[]の要素の値は以下のとおりとします。

- abc[0.0]=X_0=LREAL#1.0, abc[0,1]=Y_0=LREAL#5.0
- abc[1.0]=X_1=LREAL#2.0, abc[1,1]=Y_1=LREAL#6.0

- abc[2.0]=X_2=LREAL#4.0, abc[2,1]=Y_2=LREAL#2.0
- abc[3.0]=X_3=LREAL#5.0, abc[3,1]=Y_3=LREAL#3.0

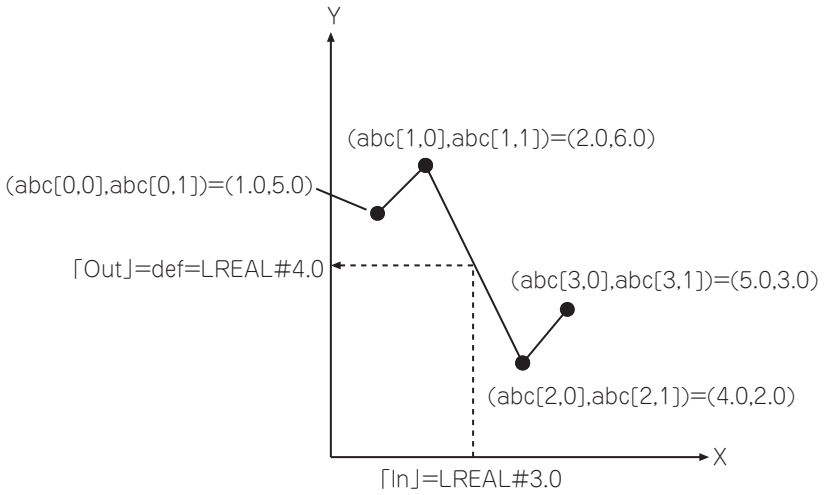
変換の結果、「Out」の値は LREAL#4.0 になります。

LD



ST

```
def:=PWLApprox(LREAL#3.0, abc[0,0], UINT#4);
```



PWLApprox 命令と PWLApproxNoLineChk 命令の違い

PWLApprox 命令と PWLApproxNoLineChk 命令の違いは、「In」や Line[]の有効性をチェックするかどうかと処理時間です。それぞれの仕様は、以下のとおりです。

命令	チェック内容	データが有効でない場合の処理	処理時間
PWLApprox	• Line[]の中身が、X 座標の昇順に並んでいること • 「In」、Line[]が実数の場合、「In」と Line[]の要素が非数、正の無限大、負の無限大でないこと	• 異常になります。 • 「ENO」の値が FALSE になります。 • 「Out」の値が変化しません。	長い
PWLApproxNoLineChk	チェックしない	• 異常になりません。 • 「ENO」の値が TRUE になります。 • 「Out」に有効な値が出力されない場合があります。	短い

PWLApproxNoLineChk 命令と PWLLineChk 命令の使用方法

PWLApproxNoLineChk は、「In」や Line[]の有効性をチェックしないかわりに、処理時間が短いという特長があります。したがって、それらの入力変数の有効性が保証されている場合は、PWLApprox 命令より、PWLApproxNoLineChk 命令を使う方が有利です。

一方、「PWLLineChk (P.2-434)」命令は、Line[]の中身が X 座標の昇順に並んでいるかどうかをチェックする命令です。したがって、通常は PWLApproxNoLineChk 命令を使用し、Line[]の中身が X 座標の昇順に並んでいるかどうか保証されない場合だけ、PWLLineChk 命令を併用して処理時間を短縮する方法もあります。

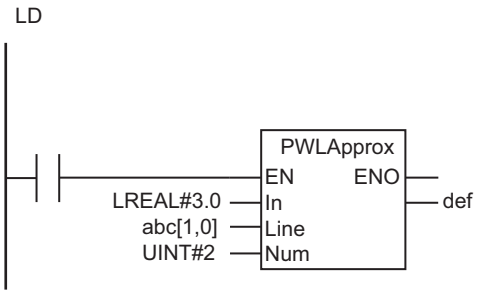
参考

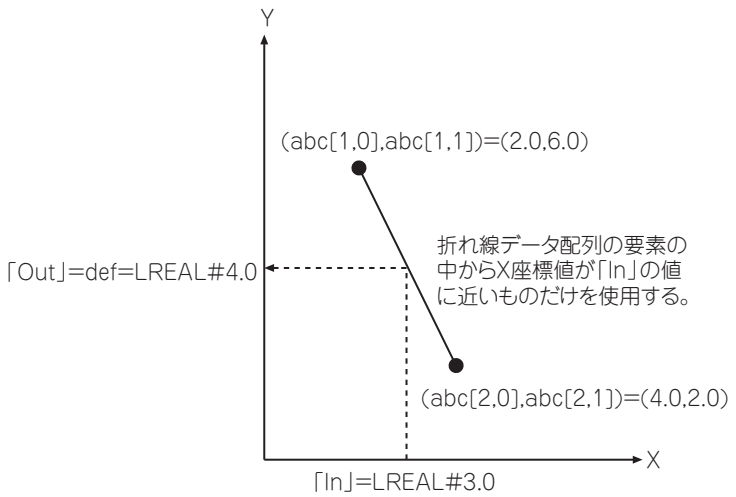
近似変換に使用する折れ線データ配列の要素範囲を絞ることでも、処理時間を短くすることができます。

先の記述例の場合、「In」の値は LREAL#3.0 なので、折れ線データ配列の要素の中から X 座標値が 3.0 に近い以下の 4 要素だけで近似変換をすると、処理時間が短くなります。

(abc[1,0], abc[1,1]) = (2.0, 6.0)
(abc[2,0], abc[2,1]) = (4.0, 2.0)

この場合、「Num」= UINT#2 とし、Line[]に渡す abc[]の要素は、abc[1,0]とします。
変換結果は、「Out」= LREAL#4.0 のままです。





使用上の注意

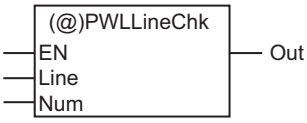
- 「In」の値が Line[0,0]の値（つまり X_1 の値）より小さい場合、「Out」の値は Line[0,1]（つまり Y_1 の値）となります。
- 「In」の値が Line[「Num」-1,0]（つまり $X_{「Num」}$ の値）より大きい場合、「Out」の値は、以下となります。

  Line[「Num」-1,1]（つまり $Y_{「Num」}$ の値）

- Line[]は 2 次元または 3 次元配列で、第 1 次元の要素数を 2 としてください。
- 「Num」の値が 0 の場合、「Out」の値は 0 になります。
- つぎの場合、PWLApprox 命令では異常となります。「ENO」は FALSE となり、「Out」は変化しません。PWLApproxNoLineChk 命令では異常となりません。
  a）折れ線データの X 座標が昇順、つまり $X_1<X_2<\cdots<X_{「Num」}$ となっていない場合。
  b）「In」と Line[]が実数型で、かつ、それらの値が非数、正の無限大、負の無限大の場合。
- つぎの場合、PWLApprox 命令でも PWLApproxNoLineChk 命令でも異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「Num」の値が、Line[]の配列領域を超える場合。
  b）「In」の値が、Line[]で指定された折れ線データの、X 座標の範囲外の場合。

PWLLineChk

折れ線近似変換（折れ線データチェックなし）命令で使用する折れ線データが、X 座標の昇順に並んでいるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PWLLineChk	折れ線データチェック	FUN		Out:=PWLLineChk(Line, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Line[] 配列	折れ線データ配列	入力	折れ線データ配列	データ型に従う	−	*1
Num	折れ線データの数		折れ線データの数			1
Out	判定結果	出力	判定結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Line[] 配列						○	○	○	○	○	○	○	○	○	○					
Num							○													
Out	○																			

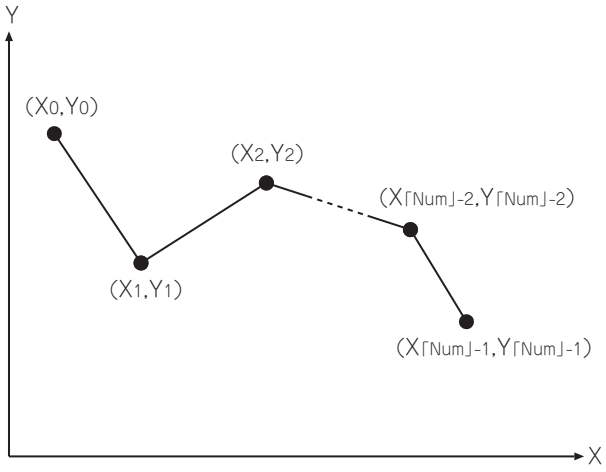
機能

折れ線近似変換（折れ線データチェックなし）PWLApproxNoLineChk 命令で使用する、折れ線データ配列 Line[]の中身が、X 座標の昇順に並んでいるかどうかを判定します。
昇順に並んでいれば、判定結果「Out」の値が TRUE、そうでなければ「Out」の値が FALSE になります。

折れ線データ Line[]の要素と折れ線データの数「Num」

Line[]は 2 次元または 3 次元配列で、第 1 次元の要素数を 2 としてください。そして、図のように、折れ線データの各点の座標値(X_0, Y_0), (X_1, Y_1)・・・を、Line[]の各要素としてください。

折れ線データの数「Num」は、折れ線近似計算に用いる Line[]の要素数の 1/2 になります。



Line[]が2次元配列の場合

Line[0,0]	X0
Line[0,1]	Y0
Line[1,0]	X1
Line[1,1]	Y1
Line[2,0]	X2
Line[2,1]	Y2
:	:
Line[「Num」-1,0]	X「Num」-1
Line[「Num」-1,1]	Y「Num」-1

Line[]が3次元の場合

Line[0,0,0]	X0
Line[0,0,1]	Y0
Line[0,1,0]	X1
Line[0,1,1]	Y1
Line[0,2,0]	X2
Line[0,2,1]	Y2
:	:
Line[0,「Num」-1,0]	X「Num」-1
Line[0,「Num」-1,1]	Y「Num」-1

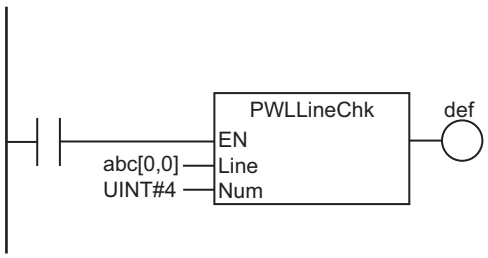
記述例

要素数 4 の折れ線データ配列 abc[]が、X 座標の昇順に並んでいるかどうかを判定する場合の例を、以下に示します。「Num」= UINT#4 で、abc[]の要素の値は以下のとおりとします。

- abc[0.0]=X_0=LREAL#1.0, abc[0,1]=Y_0=LREAL#5.0
- abc[1.0]=X_1=LREAL#6.0, abc[1,1]=Y_1=LREAL#6.0
- abc[2.0]=X_2=LREAL#4.0, abc[2,1]=Y_2=LREAL#2.0
- abc[3.0]=X_3=LREAL#5.0, abc[3,1]=Y_3=LREAL#3.0

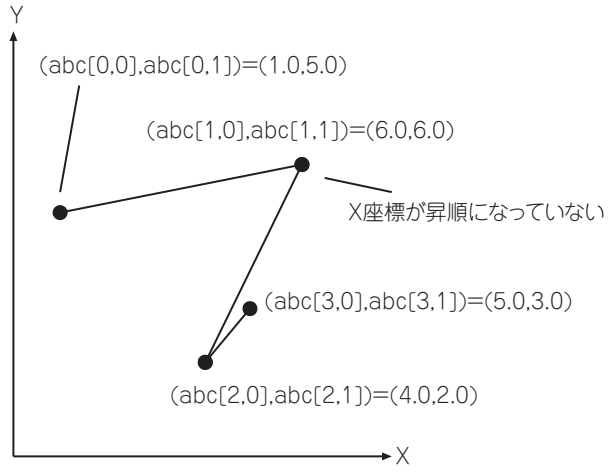
X 座標が昇順になっていないので、「Out」の値は FALSE になります。

LD



ST

```
def:=PWLLineChk(abc[0,0], UINT#4);
```



参考

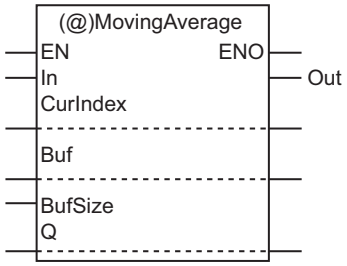
- 本命令は、PWLApproxNoLineChk 命令とあわせて使用してください。PWLApproxNoLineChk 命令の詳細は、「PWLApprox / PWLApproxNoLineChk (P.2-429)」命令を参照してください。
- 折れ線近似変換のたびに折れ線データのチェックを行う場合は、PWLApprox 命令を使用してください。PWLApprox 命令の詳細も、「PWLApprox / PWLApproxNoLineChk (P.2-429)」命令を参照してください。ただし、PWLApprox 命令より、PWLApproxNoLineChk 命令の方が処理時間が短いです。

使用上の注意

- Line[]は 2 次元または 3 次元配列で、第 1 次元の要素数を 2 としてください。
- つぎの場合、異常となります。「Out」は、FALSE となります。
  a)「Num」の値が、Line[]の配列領域を超える場合。
  b) Line[]が実数型で、かつ、要素の値が非数、正の無限大、負の無限大の場合。

MovingAverage

移動平均値を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MovingAverage	移動平均	FUN		Out:=MovingAverage(In, CurIndex, Buf, BufSize, Q);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	入力値	入力	平均値演算に用いる数値	データ型に従う	－	*1
BufSize	最大要素数		平均値演算に用いる最大要素数			1
CurIndex	入力値格納位置	入出力	「In」を格納する、Buf[]の位置	データ型に従う	－	－
Buf[] 配列	入力値格納配列		「In」を格納する配列			
Q	計算完了フラグ		TRUE: Buf[]への数値格納数が、「BufSize」以上 FALSE: Buf[]への数値格納数が、「BufSize」未満			
Out	演算結果	出力	演算結果	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

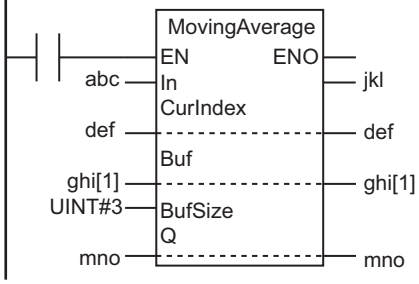
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
BufSize							○													
CurIndex							○													
Buf[] 配列	「In」と同じデータ型を要素とする配列																			
Q	○																			
Out						○	○	○	○	○	○	○	○	○	○					

機能

本命令が実行されるごとに、入力値「In」を入力値格納配列 Buf[]に格納していきます。そして、格納した値の平均値を、演算結果「Out」に格納します。平均値計算に用いる最大要素数は、「BufSize」で指定します。

「BufSize」=UINT#3 の場合を例にとって、処理手順を説明します。記述は、以下のようになります。

LD

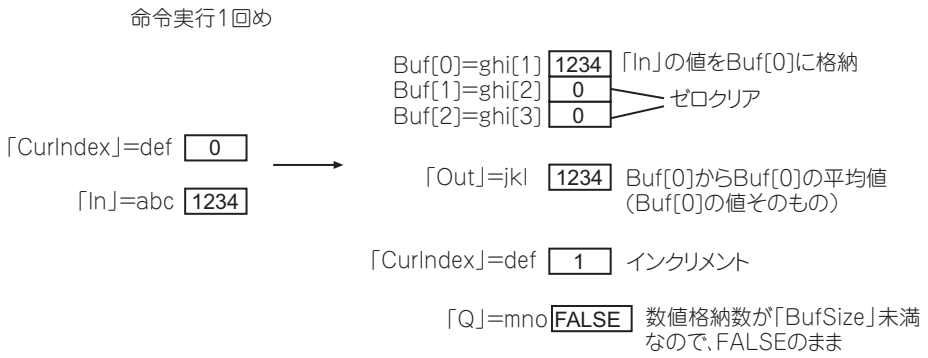


ST

```
jkl:=MovingAverage(abc, def, ghi[1], UINT#3, mno);
```

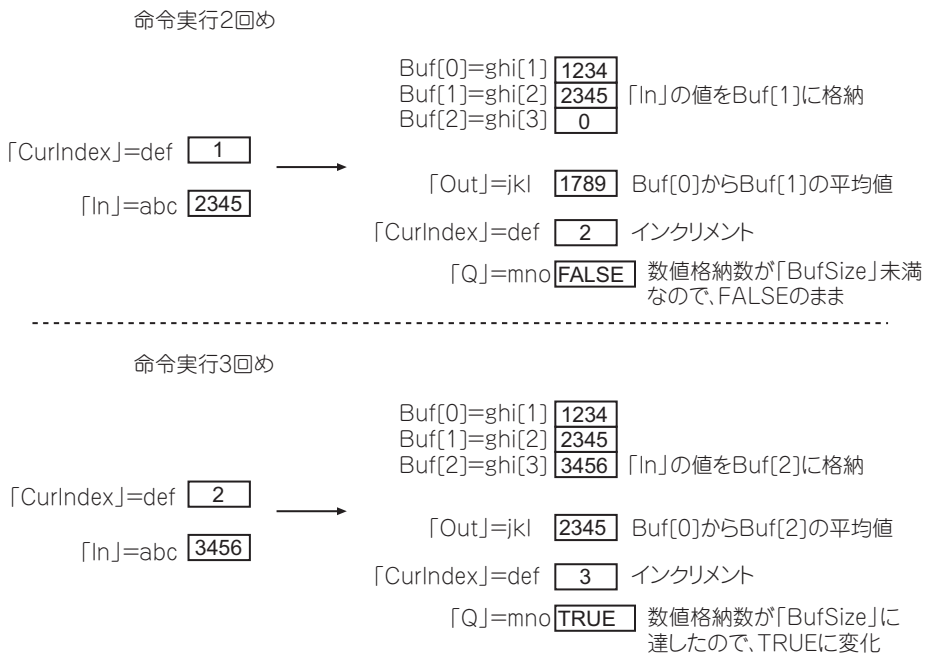
初回の数値入力

入力値格納位置「CurIndex」に 0 を設定して、本命令を実行します。
入力値格納配列 Buf[]の、Buf[0]から Buf[「BufSize」-1]がゼロクリアされた後、最初の入力値「In」の値が、Buf[0]に格納されます。
計算完了フラグ「Q」の値が、FALSE になります。これは、Buf[]への数値格納数が、まだ「BufSize」に達していないことを意味します。
「Q」の値が FALSE である間、平均値計算は Buf[0]から始まる「CurIndex」+1 個の数値で行います。そして、演算結果が「Out」に格納されます。
その後、「CurIndex」の値をインクリメントします。



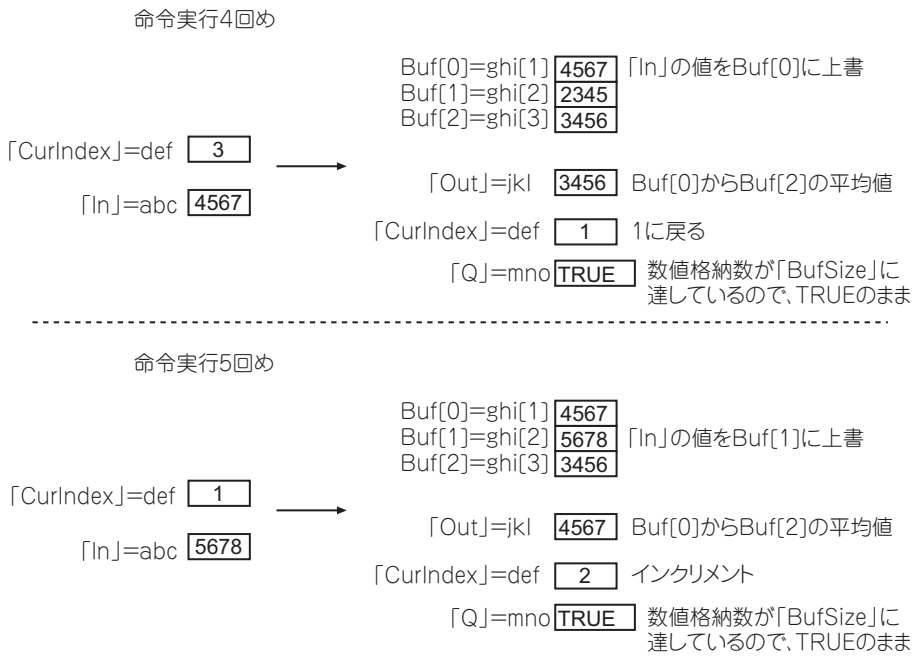
「BufSize」回めまでの数値入力

本命令を実行するごとに、「In」の値が、Buf[「CurIndex」]に格納されます。平均値計算は、Buf[0]から始まる「CurIndex」+1 個の数値で行われ、演算結果が「Out」に格納されます。
実行回数が「BufSize」に達したら、「Q」の値が TRUE に変わります。



「BufSize」回めより後の数値入力

本命令を実行するごとに、「In」の値が Buf[0]から Buf[「BufSize」-1]までの間に、周期的に上書されます。平均値計算は、その時点での Buf[0]から Buf[「BufSize」-1]で行われ、演算結果が「Out」に格納されます。
「CurIndex」の値は、「BufSize」のつぎは 1 に戻り、そこからまたインクリメントされます。「Q」の値は、TRUE のままです。

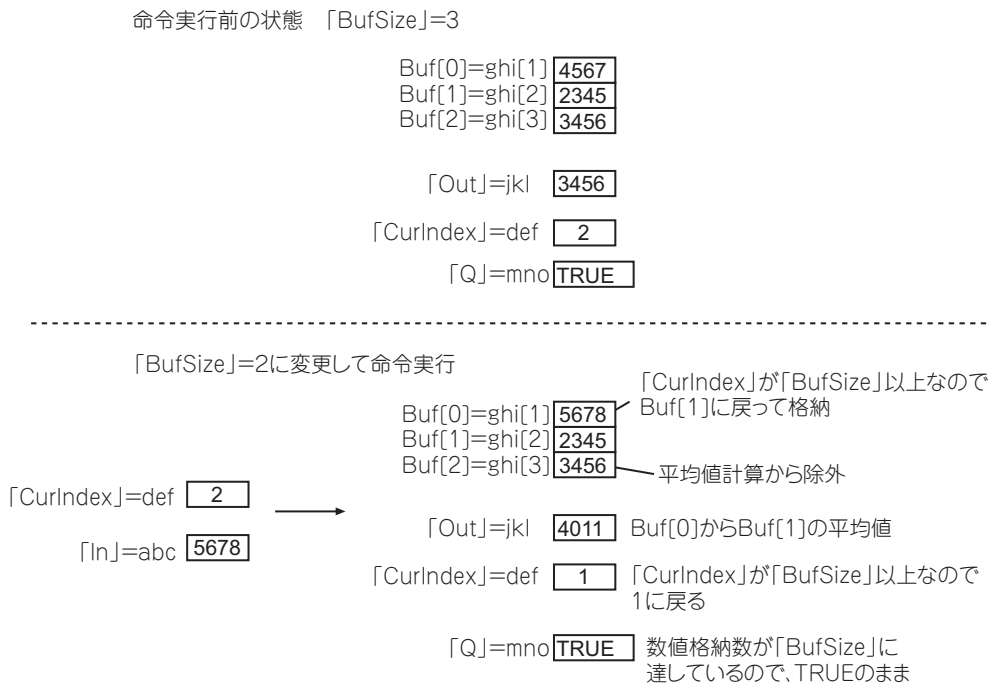


格納値の初期化

「CurIndex」の値を 0 にして本命令を実行すると、いったん Buf[0]から Buf[「BufSize」-1]の値が 0 になり、あらためて、その時点の「In」の値が Buf[0]に格納されます。
「CurIndex」の値は 1 になり、「Q」の値は FALSE になります。

「BufSize」の値の変更

「BufSize」の値を変更して本命令を実行した場合、変更後の「BufSize」と、その時点の「CurIndex」の値に従って動作します。



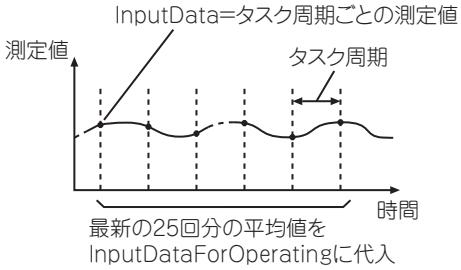
使用上の注意

- 「In」と Buf[]の要素のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- Buf[]の大きさは、必ず「BufSize」以上にしてください。
- 演算結果が「Out」の有効範囲外であっても、異常にはならず、「Out」には不正値が格納されます。
- 「BufSize」の値が 0 の場合、「Out」、「CurIndex」の値は 0 となります。「Q」の値は TRUE になります。
- 「BufSize」の値を変更したら、必ず「CurIndex」の値を 0 にして、格納値の初期化を行ってください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「BufSize」の値が、Buf[]の大きさを超える場合。

サンプルプログラム

センサなどのアナログ入力データに含まれる、ノイズなどの外乱の影響を除去する例です。
入力データ「InputData」の、最新の 25 回分の平均値「DataAve」を、次工程の入力「InputDataForOperating」に代入します。
「InputData」は、実行条件「Trigger」の値が TRUE である間、タスク周期ごとに入力します。

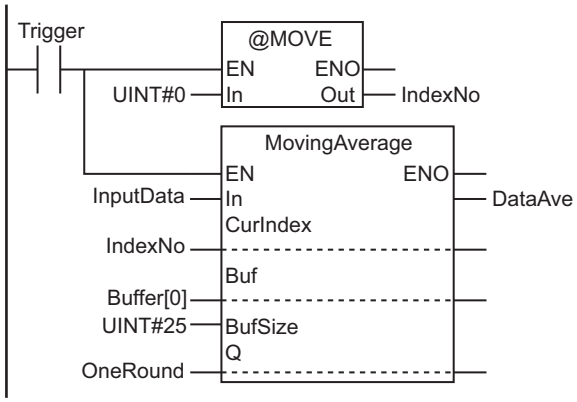
「InputData」が 25 回以上入力されるまでは、データ量が不十分なので、平均値ではなく、最新の 1 回の「InputData」の値を、「InputDataForOperating」に代入します。
「Trigger」の値が FALSE→TRUE になると、平均値をクリアし、「InputData」の入力を初めからやり直します。



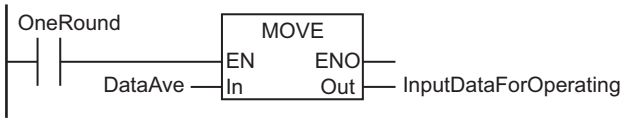
LD

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
InputData	INT	10	入力値
Buffer	ARRAY[0..24] OF INT	[25(0)]	入力値格納配列
DataAve	INT	0	平均値
OneRound	BOOL	FALSE	25 回入力フラグ
IndexNo	UINT	0	入力値格納位置
InputDataForOperating	INT	0	次工程入力

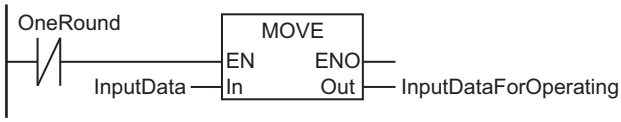
Triggerが立ち上がったら、IndexNoに、0を代入
TriggerがTRUEの間、タスク周期ごとにInputDataを入力して平均値計算



InputDataが25回以上入力されたら、DataAveをInputDataForOperatingに代入



InputDataが25回以上入力されるまでは、InputDataをInputDataForOperatingに代入



ST

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
LastTrigger	BOOL	FALSE	前タスク周期の「Trigger」の値
Operating	BOOL	FALSE	処理中
OperatingStart	BOOL	FALSE	処理開始
Buffer	ARRAY[0..24] OF INT	[25(0)]	入力値格納配列
InputData	INT	10	入力値
DataAve	INT	0	平均値
OneRound	BOOL	FALSE	25 回入力フラグ
IndexNo	UINT	0	入力値格納位置
InputDataForOperating	INT	0	次工程入力

```
// Trigger の立ち上がり検出
IF ((Trigger=TRUE) AND (LastTrigger=FALSE)) THEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

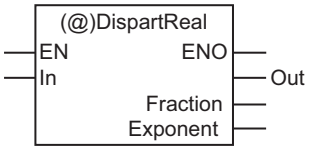
// 平均値クリア
IF (OperatingStart=TRUE) THEN
    IndexNo:=UINT#0;
    OperatingStart:=FALSE;
END_IF;

// 移動平均処理
IF (Operating=TRUE) THEN
    DataAve:=MovingAverage(
        In :=InputData,
        CurIndex:=IndexNo,
        Buf :=Buffer[0],
        BufSize :=UINT#25,
        Q :=OneRound);
    IF (OneRound=TRUE) THEN
        // 最新の 25 回分の入力値の平均値を代入
        InputDataForOperating:=DataAve;
    ELSE
        // 最新の 1 回分の入力値を代入
        InputDataForOperating:=InputData;
    END_IF;
END_IF;

// 移動平均処理終了
IF (Trigger=FALSE) THEN
    Operating:=FALSE;
END_IF;
```

DispartReal

実数を、符号付き仮数部と、指数部に分離します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DispartReal	実数の仮数・指数分離	FUN		Out:=DispartReal(In, Fraction, Exponent);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	実数	入力	分離する実数	データ型に従う	－	*1
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－
Fraction	符号付き仮数部		符号付き仮数部	*2		
Exponent	指数部		指数部	*3		

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. 「In」、「Fraction」のデータ型の組み合わせで、有効範囲が変わります。詳細は、「「Fraction」の有効範囲 (P.2-444)」を参照してください。
*3. 「In」が REAL 型の場合、-44～32、LREAL 型の場合、-322～294。

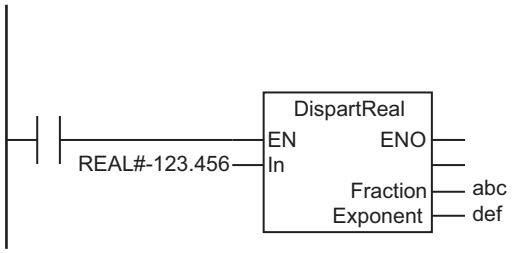
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	○																			
Fraction	「In」のデータ型が REAL なら DINT、LREAL なら LINT																			
Exponent											○									

機能

実数「In」を、符号付き仮数部と、指数部「Exponent」に分離します。

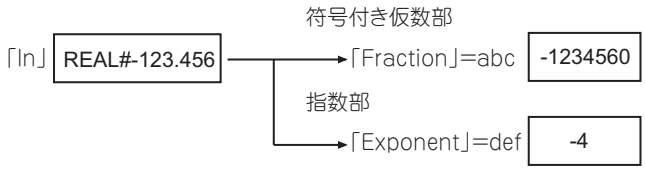
「In」が REAL 型の場合、「Fraction」は 7 けたの整数になります。「In」が LREAL 型の場合、「Fraction」は 15 けたの整数になります。
「In」が REAL 型で、値が REAL#-123.456 の場合の例を、以下に示します。

LD

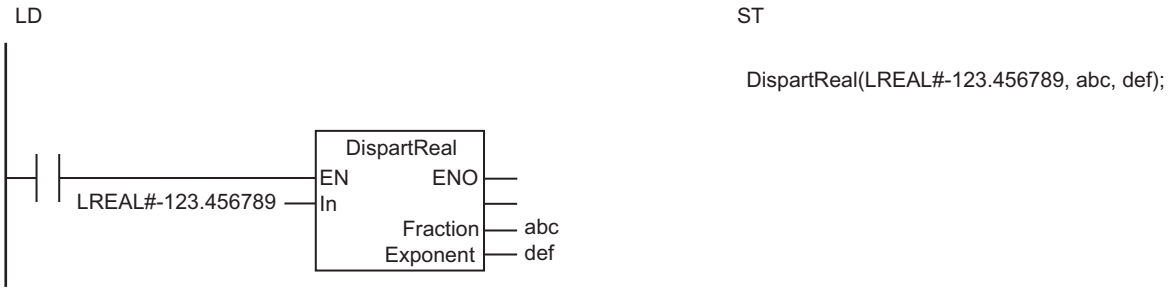


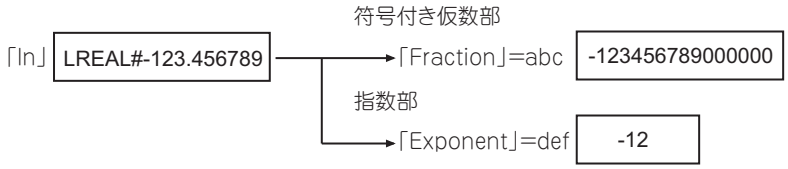
ST

```
DispartReal(REAL#-123.456, abc, def);
```



「In」が LREAL 型で、値が LREAL#-123.456789 の場合の例を、以下に示します。





「Fraction」の有効範囲

「Fraction」の有効範囲は、「In」と「Fraction」のデータ型の組み合わせで以下のように異なります。

「In」のデータ型	「Fraction」のデータ型	「Fraction」の有効範囲
REAL	DINT	-9999999 ～ 9999999
LREAL	LINT	-999999999999999 ～ 999999999999999

参考

符号付き仮数部と指数部を結合して、実数を取得する場合は、「UniteReal (P.2-446)」命令を使用してください。

使用上の注意

- 「In」の値によって、整数に変換する際に、誤差が生じる場合があります。
- 「In」の有効けた数が「Fraction」の有効けた数より大きい場合、「Fraction」の有効範囲内に収まるように、五捨五入します。
- 「In」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「In」に渡すパラメータのデータ型	「In」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

- つぎの場合、異常となります。「ENO」は FALSE となり、「Fraction」、「Exponent」は変化しません。
  a)「In」の値が、非数または無限大の場合。

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1 -1.49→-1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2 -1.50→-2 -2.50→-2
0.5 より大	切り上げ	1.51→2 -1.51→-2

UniteReal

符号付き仮数部と、指数部を結合して実数にします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
UniteReal	仮数・指数の実数への結合	FUN	(@)UniteReal EN ENO Fraction Out Exponent	Out:=UniteReal(Fraction, Exponent);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Fraction	符号付き仮数部	入力	符号付き仮数部	データ型に従う	−	*1
Exponent	指数部		指数部			0
Out	実数	出力	実数	データ型に従う	−	−

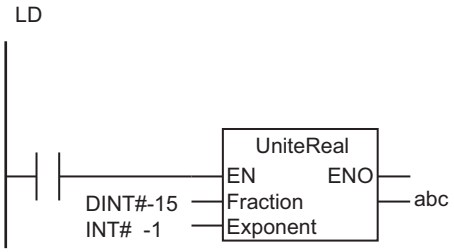
*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

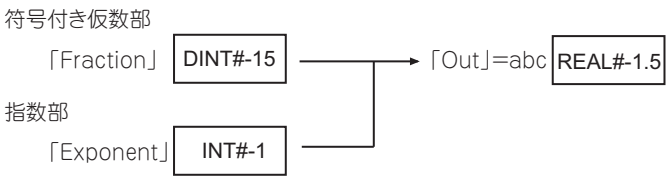
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Fraction												○	○							
Exponent											○									
Out	「Fraction」のデータ型が DINT なら REAL、LINT なら LREAL																			

機能

符号付き仮数部「Fraction」と、指数部「Exponent」を結合して、実数「Out」を取得します。

「Fraction」=DINT#-15、「Exponent」=INT#-1 の場合の例を、以下に示します。





参考

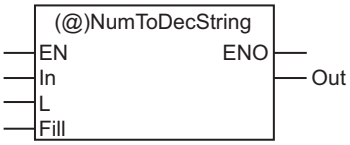
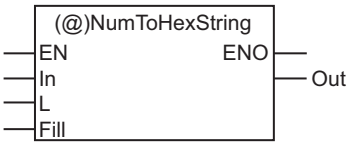
実数を、符号付き仮数部と指数部に分離する場合は、「DispartReal (P.2-443)」命令を使用してください。

使用上の注意

- 「Fraction」と「Exponent」の値によって、整数から実数に変換する際に、誤差が生じる場合があります。
- 結合結果が「Out」の有効範囲外の場合、「Exponent」が正数なら「Out」の値は「Fraction」と同じ符号を持つ無限大になります。「Exponent」が負数なら「Out」の値は 0 になります。

NumToDecString / NumToHexString

NumToDecString ： 整数を、固定長 10 進文字列表現に変換します。
NumToHexString ： 整数を、固定長 16 進文字列表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NumToDecString	固定長 10 進文字列変換	FUN		Out:=NumToDecString(In, L, Fill);
NumToHexString	固定長 16 進文字列変換	FUN		Out:=NumToHexString(In, L, Fill);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	整数	入力	整数	データ型に従う	－	*1
L	文字数		「Out」の文字数	0～1985		1
Fill	付加文字		付加文字	_BLANK, _ZERO		_BLANK
Out	文字列	出力	文字列	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
L							○													
Fill	列挙型_eFILL_CHR　列挙子は「機能 (P.2-448)」参照																			
Out																				○

機能

どちらの命令も、文字列「Out」の文字数が、文字数「L」になるようにします。
文字数が足りない場合は、付加文字「Fill」で指定された文字を上位けたに付加します。
変換結果の文字数が「L」より多い場合は、変換結果の最下位けたから「L」文字分だけを、「Out」に代入します。
「Out」の終端には、NULL 文字を付加します。NULL 文字は、文字数には含めません。

「Fill」のデータ型は、列挙型_eFILL_CHR です。列挙子の意味は、以下のとおりです。

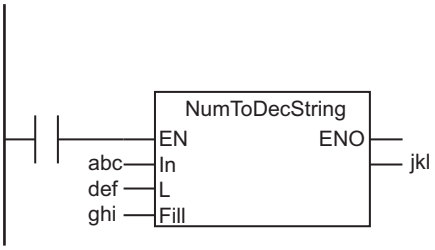
列挙子	意味
_BLANK	' '（半角空白文字）
_ZERO	'0'

NumToDecString

整数「In」を、UTF-8 半角英数の 10 進数表記の文字列に変換します。「In」が負数の場合は、先頭に '-'（マイナス符号）を付けます。

NumToDecString 命令のいくつかの例を、以下に示します。

LD



ST

```
jkl:=NumToDecString(abc, def, ghi);
```

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl [ ][ ][ ][ ][ ][1][2][8]

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl [ ][ ][ ][ ][-][1][2][8]

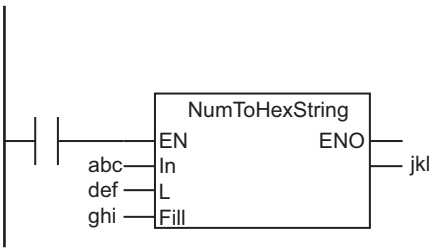
「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl [-][0][0][0][0][1][2][8]

NumToHexString

整数「In」を、UTF-8 半角英数の 16 進数表記の文字列に変換します。「In」が負数の場合は、2 の補数で表現（ビット反転+1）します。

NumToHexString 命令のいくつかの例を、以下に示します。

LD



ST

```
jkl:=NumToHexString(abc, def, ghi);
```

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl [ ][ ][ ][ ][ ][ ][8][0]

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl [0][0][0][0][0][0][8][0]

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl [F][F][F][F][F][F][8][0]

使用上の注意

- 「L」の値が 0 の場合、「Out」は NULL 文字だけの文字列となります。
- 変換結果の文字数が「L」の値より多い場合は、変換結果の下位から「L」けた分を「Out」に格納します。以下に例を示します。

命令	「In」の値	「L」の値	「Out」の値
NumToDecString	128	2	28
NumToHexString			80

- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「L」の値が、有効範囲外の場合。
  b)「Fill」の値が、有効範囲外の場合。

HexStringToNum_**

16 進文字列表現を、整数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
HexStringToNum_**	16 進文字列→数値変換グループ	FUN	(@)HexStringToNum_** EN ENO In Out ただし、“**”は整数のデータ型名	Out:=HexStringToNum_**(In); ただし、“**”は整数のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	16 進文字列	入力	16 進文字列	データ型に従う	−	"
Out	整数	出力	整数	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out						○	○	○	○	○	○	○	○							

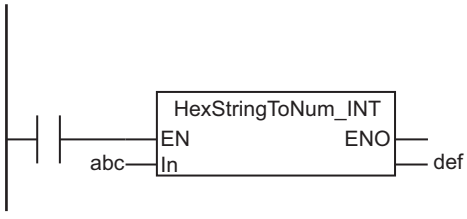
機能

16 進表記の文字列「In」を、整数に変換します。上位けたの半角スペース(16#20)や'0'(16#30)は、無視します。また、文字列の中のアンダーバー(16#5F)も無視します。

命令名称は、「Out」のデータ型によって異なります。たとえば、「Out」が INT 型の場合、命令名称は HexStringToNum_INT となります。

いくつかの例を、以下に示します。

LD



ST

```
def:=HexStringToNum_INT(abc);
```

「In」=abc 8 0 ⟶ 「Out」=def=128

「In」=abc - 8 0 ⟶ 「Out」=def=-128

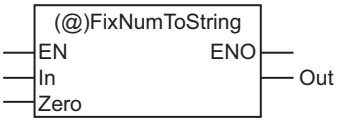
「In」=abc - 0 0 0 0 0 F ⟶ 「Out」=def=-15

使用上の注意

- 変換結果が、「Out」の有効範囲外であっても、異常にはならず、「Out」には不正値が格納されます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の内容が、数値変換不可能な文字の場合。

FixNumToString

符号付き固定小数点数を、10 進文字列表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FixNumToString	固定小数点数→文字列変換	FUN		Out:=FixNumToString(In, Zero);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	固定小数点数	入力	符号付き固定小数点数	データ型に従う	−	0
Zero	ゼロ表示		小数点以下のけた数が、3 未満の場合の表示 TRUE: '0'を補う FALSE: '0'を補わない			TRUE
Out	10 進文字列	出力	10 進文字列	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In					○															
Zero	○																			
Out																				○

機能

符号付き固定小数点数「In」を、10 進文字列に変換します。変換の手順は、以下のとおりです。

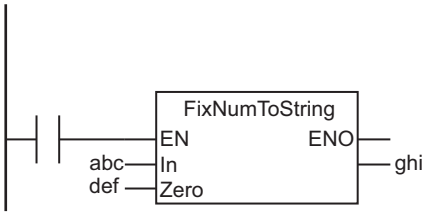
1 **16 進表現である「In」を、10 進表現に直します。**

2 **その値を、1,000 で割ります。**

ゼロ表示「Zero」は、「In」の小数点以下のけた数が 3 未満の場合に、「Out」の小数第 3 位までに'0'を補うかどうかを指定します。「Zero」の値が TRUE なら、'0'を補います。
「Out」の終端には、NULL 文字を付加します。

いくつかの例を、以下に示します。

LD



ST

```
ghi:=FixNumToString(abc, def);
```

In = abc	Out = ghi	
	Zero = def = TRUE	Zero = def = FALSE
16#0001462C (10#83500)	'83.500'	'83.5'
16#00051AA4 (10#334500)	'334.500'	'334.5'
16#0003BEFC (10#245500)	'245.500'	'245.5'

参考

固定小数点数のフォーマットは、オムロンの視覚センサ FZ シリーズの固定小数点出力のフォーマットと共通です。

StringToFixNum

10 進文字列を、符号付き固定小数点数表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringToFixNum	文字列→固定小数点数変換	FUN	(@)StringToFixNum EN ENO In Out	Out:=StringToFixNum(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	10 進文字列	入力	10 進文字列	データ型に従う	−	''
Out	固定小数点数	出力	固定小数点数	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out				○																

機能

10 進文字列「In」を、固定小数点数に変換します。変換の手順は、以下のとおりです。

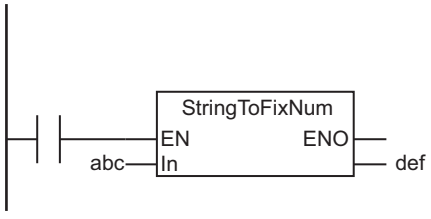
1 **「In」が表現している数値を、1,000 倍します。**

2 **その値の小数点以下を、切り捨てます。**

3 **その値を、32 ビットの 16 進表現（DWORD 型）にします。**

いくつかの値の例を、以下に示します。

LD



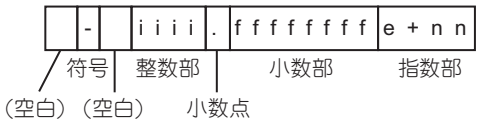
ST

```
def:=StringToFixNum(abc);
```

In = abc	Out = def
'83.5'	16#0001462C (10#83500)
'334.5'	16#00051AA4 (10#334500)

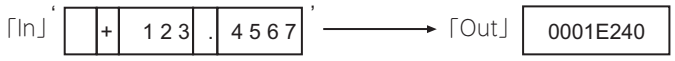
In = abc	Out = def
'245.5'	16#0003BEFC (10#245500)

「In」の文字列形式は、つぎのとおりです。

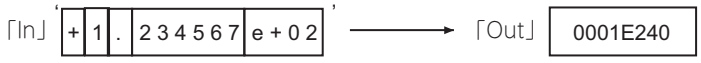


名称	書式
符号	• 文字列の先頭に現れる連続する' '（空白文字 16#20）を無視した後に現れる単一の '+' または '-' を符号とします。 • 符号は省略可能です。 • 符号の後の連続する空白文字も無視されます。
整数部	• 符号の後から小数点の前までの、連続する数値（'0'～'9'）を整数部とします。ただし、符号は省略される場合もあります。また、符号と整数部の間に連続する空白文字が入る場合もあります。 • 小数点および小数部がない場合は、指数部の前までを整数部とします。 • 小数点、小数部、指数部がない場合は、文字列終端までを整数部とします。 • 整数部は省略できません。 • 整数部の最大けた数は、最大文字列長 1986 から、符号、小数点、小数部、指数部、および符号の前後の空白文字のバイト数の合計を引いたけた数です。
小数点	• 整数部に続く単一の '.'（ドット）を小数点とします。 • 小数部が存在しない場合は、小数点は省略してください。
小数部	• 小数点の後から指数部の前までの、連続する数値（'0'～'9'）を小数部とします。 • 指数部がない場合は、文字列終端までを小数部とします。 • 小数部は省略可能です。小数点がない場合は、小数部も存在しません。 • 小数部の最大けた数は 15 です。
指数部	• 小数部に続く単一の 'e' もしくは 'E'、それに続く単一の '+' もしくは '-' から文字列終端までの、連続する数値（'0'～'9'）を、指数部とします。 • 小数部がない場合は、小数点に続く上記文字列を指数部とします。 • 小数点および小数部がない場合は、整数部に続く上記文字列を指数部とします。 • 指数部は省略可能です。 • 指数部の数字部分の最大けた数は 3 です。

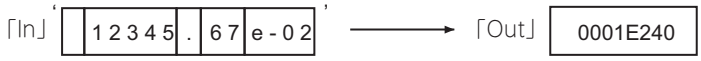
（例 1） 符号あり、小数点あり、小数部あり、指数部なし



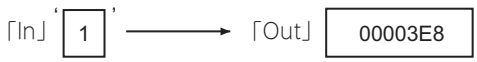
（例 2） 符号あり、小数点あり、小数部あり、指数部あり



（例 3） 符号なし、小数点あり、小数部あり、指数部あり



（例 4） 符号なし、小数点なし、小数部なし、指数部なし



参考

固定小数点数のフォーマットは、オムロンの視覚センサ FZ シリーズの固定小数点出力のフォーマットと共通です。

使用上の注意

- 「In」の小数点以下 4 けため以降は、切り捨てられます。
- 「In」の文字列の中のアンダーバー(16#5F)は、無視します。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の内容が、数値変換不可能な文字の場合。
  b)「In」の内容が、小数点ありで小数部なしの場合。

DtToString

日時を、文字列表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DtToString	日時→文字列変換	FUN	(@)DtToString EN ENO In Out	Out:=DtToString(In);

変数

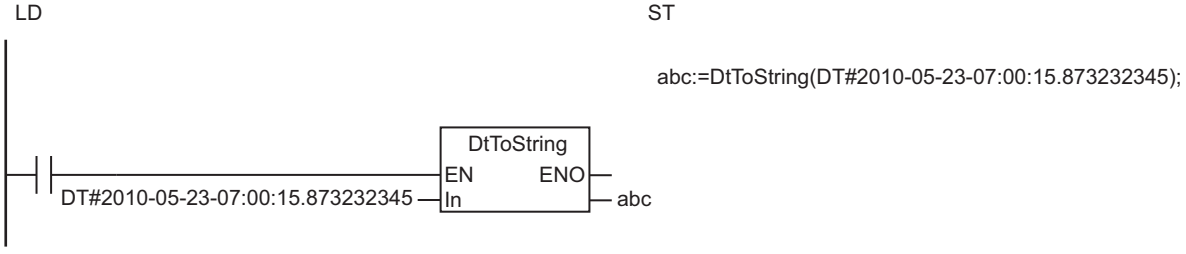
	名称	入／出力	内容	有効範囲	単位	初期値
In	日時	入力	日時	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Out	文字列	出力	文字列	30 バイト（半角英数字 29 文字+終端 NULL 文字）	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																				○

機能

日時「In」を、文字列に変換します。文字列「Out」の終端には、NULL 文字を付加します。

「In」の値が 2010 年 5 月 23 日 7 時 0 分 15.873232345 秒の場合の例を、以下に示します。
変数「abc」の値は'2010-05-23-07:00:15.873232345'となります。



日時「In」を文字列に変換します。
「In」の値が2010年5月23日7時0分15.873232345秒なので、abcの値は'2010-05-23-07:00:15.873232345'になります。

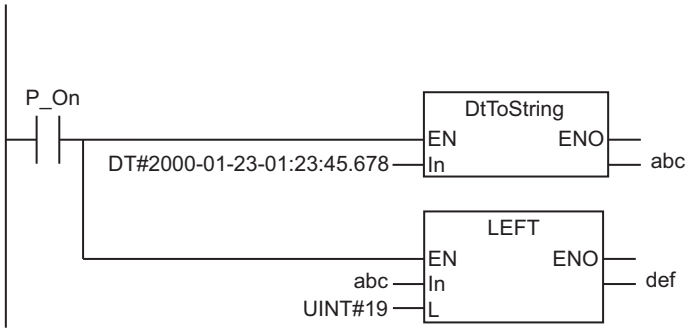
「In」 DT#2010-05-23-07:00:15.873232345 → 文字列変換 → 「Out」=abc '2010-05-23-07:00:15.873232345'

参考

「Out」は、ナノ秒単位で表現されます。もし、秒単位やミリ秒単位の文字列を取得したい場合は、本命令と「LEFT / RIGHT (P.2-556)」命令を組み合わせて使用してください。

秒単位の文字列を取得する場合の例を、以下に示します。

● LD



● ST

```
def:=LEFT(DtToString(DT#2000-01-23-01:23:45.678), UINT#19);
```

DateToString

日付を、文字列表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DateToString	日付→文字列変換	FUN	(@)DateToString EN ENO In Out	Out:=DateToString(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付	入力	日付	データ型に従う	年月日	D#1970-1-1
Out	文字列	出力	文字列	11 バイト（半角英数字 10 文字＋終端 NULL 文字）	－	－

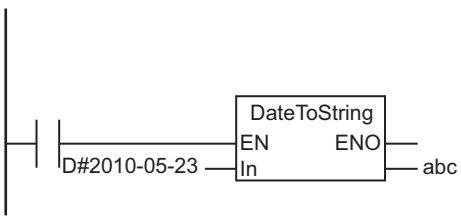
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○			
Out																				○

機能

日付「In」を、文字列に変換します。「Out」の終端には、NULL 文字を付加します。

「In」の値が 2010 年 5 月 23 日の場合の例を、以下に示します。
変数「abc」の値は'2010-05-23'となります。

LD



ST

```
abc:=DateToString(D#2010-05-23);
```

日付「In」を文字列に変換します。
「In」の値が2010年5月23日なので、abcの値は'2010-05-23'になります。

「In」D#2010-05-23 → 文字列変換 → 「Out」=abc '2010-05-23'

TodToString

時刻を、文字列表現に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TodToString	時刻→文字列変換	FUN	(@)TodToString EN ENO In Out	Out:=TodToString(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	時刻	入力	時刻	データ型に従う	時分秒	TOD#0:0:0
Out	文字列	出力	文字列	19 バイト（半角英数字 18 文字＋終端 NULL 文字）	–	–

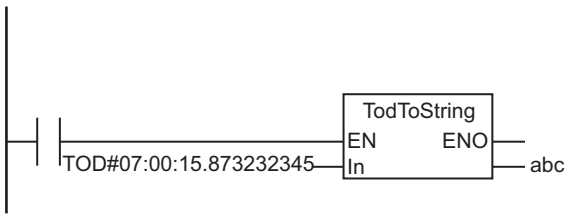
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out																				○

機能

時刻「In」を、文字列に変換します。「Out」の終端には、NULL 文字を付加します。

「In」の値が 7 時 0 分 15.873232345 秒の場合の例を、以下に示します。
変数「abc」の値は'07:00:15.873232345'となります。

LD



ST

```
abc:=TodToString(TOD#07:00:15.873232345);
```

時刻「In」を文字列に変換します。
「In」の値が"7時0分15.873232345秒なので、abcの値は'07:00:15.873232345'になります。

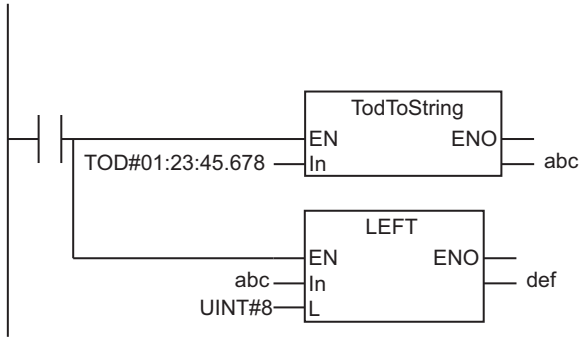
「In」 TOD#07:00:15.873232345 → 文字列変換 → 「Out」=abc '07:00:15.873232345'

参考

「Out」は、ナノ秒単位で表現されます。
秒単位やミリ秒単位の文字列を取得する場合は、本命令と「LEFT / RIGHT (P.2-556)」命令を組み合わせて使用してください。

秒単位の文字列を取得する場合の例を、以下に示します。

● LD

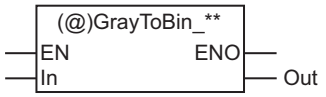
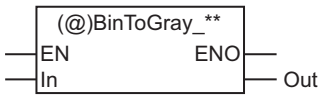


● ST

```
def:=LEFT(TodToString(TOD#01:23:45.678), UINT#8);
```

GrayToBin_** / BinToGray_**

GrayToBin_** ： グレイコードをビット列に変換します。
BinToGray_** ： ビット列をグレイコードに変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GrayToBin_**	グレイコード→BIN コード変換グループ	FUN	 ただし、“**”はビット列のデータ型名	Out:=GrayToBin_**(In); ただし、“**”はビット列のデータ型名
BinToGray_**	BIN コード→グレイコード変換グループ	FUN	 ただし、“**”はビット列のデータ型名	Out:=BinToGray_**(In); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	0
Out	変換結果	出力	変換結果	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out	「In」と同じデータ型																			

機能

命令名称は、「In」、「Out」のデータ型によって、異なります。たとえば、「In」、「Out」が WORD 型の場合、命令名称は GrayToBin_WORD や BinToGray_WORD となります。

GrayToBin_**

グレイコードの変換対象「In」をビット列に変換します。

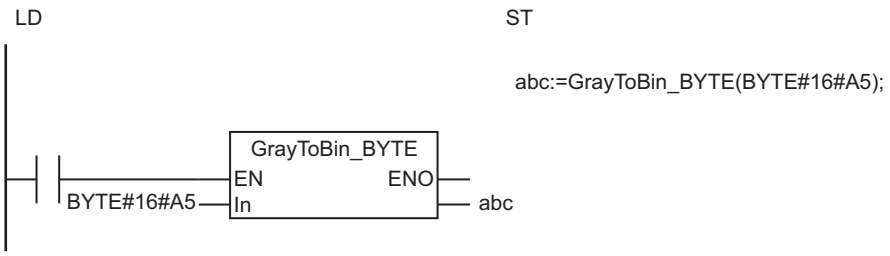
変換の手順を、「In」、「Out」が BYTE 型の場合で説明します。

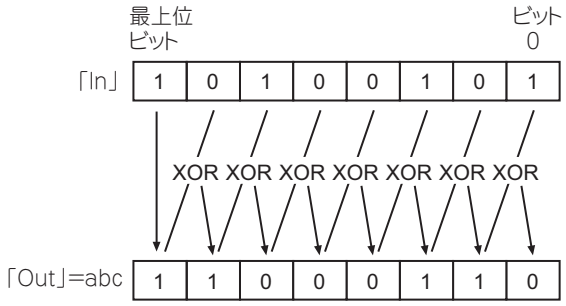
1 **「In」の最上位ビット（ビット 7）を、そのまま「Out」の最上位ビット（ビット 7）とします。**

2 **「In」のビット 6 の値と、「Out」のビット 7 の値の排他的論理和を、「Out」のビット 6 とします。**

3 **以下、同様にして、「Out」の最下位ビット（ビット 0）までの値を決めます。**

GrayToBin_BYTE 命令で、「In」=BYTE#16#A5 の場合の例を以下に示します。





BinToGray_**

ビット列の変換対象「In」をグレイコードに変換します。

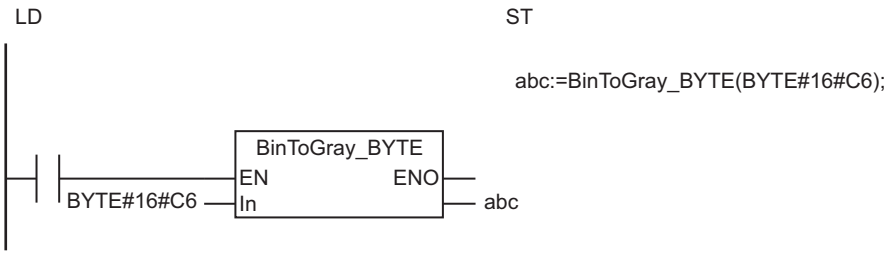
変換の手順を、「In」、「Out」が BYTE 型の場合で説明します。

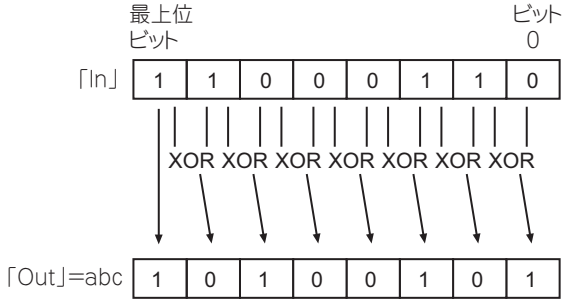
1 **「In」の最上位ビット（ビット 7）を、そのまま「Out」の最上位ビット（ビット 7）とします。**

2 **「In」のビット 7 の値と、「In」のビット 6 の値の排他的論理和を、「Out」のビット 6 とします。**

3 **以下、同様にして、「Out」の最下位ビット（ビット 0）までの値を決めます。**

BinToGray_BYTE 命令で、「In」=BYTE#16#C6 の場合の例を以下に示します。



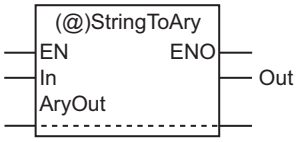


使用上の注意

「In」と「Out」のデータ型は、同じにしてください。

StringToAry

文字列を、BYTE 型の配列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringToAry	文字列→配列変換	FUN		Out:=StringToAry(In, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	文字列	入力	文字列	データ型に従う	−	''
AryOut[] 配列	BYTE 型配列	入出力	BYTE 型配列	データ型に従う	−	−
Out	変換したバイト数	出力	変換したバイト数	0～1985	バイト	−

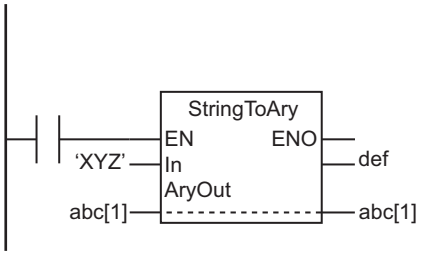
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
AryOut[] 配列		○																		
Out							○													

機能

文字列「In」の文字コードを数値と見なして、1 文字ずつ、BYTE 型配列 AryOut[]に格納します。
「Out」には、変換したバイト数を格納します。

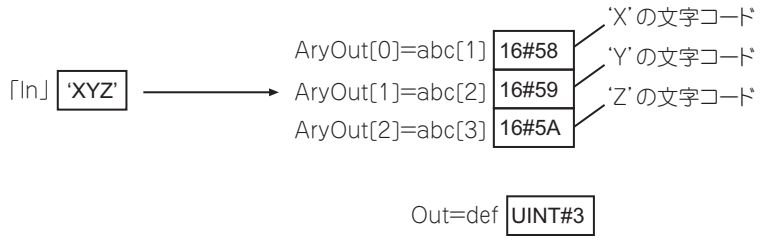
「In」='XYZ'の場合の例を、以下に示します。

LD



ST

```
def:=StringToAry('XYZ', abc[1]);
```



使用上の注意

- 「In」の終端にある NULL 文字は、AryOut[]に格納しません。
- 「In」が NULL 文字だけの文字列の場合、「Out」の値は 0 になり、AryOut[]は変化しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」や AryOut[]は変化しません。
  a)「In」のバイト数が、AryOut[]の要素数より大きい場合。

AryToString

BYTE 型配列を、文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryToString	配列→文字列変換	FUN	(@)AryToString EN ENO In Out Size	Out:=AryToString(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	BYTE 型配列	入力	BYTE 型配列 要素数は、最大 1985	データ型に従う	−	*1
Size	変換要素数		変換する In[]の要素数	0～1985		1
Out	文字列	出力	文字列	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

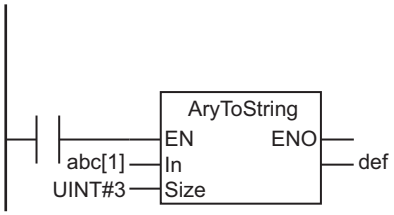
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○																		
Size							○													
Out																				○

機能

BYTE 型の配列 In[0]以降の各要素の値を文字コードと見なして、文字列「Out」に格納します。
「Out」の終端には、NULL 文字を付加します。
「Size」には、変換する In[]の要素数を指定します。ただし、In[0]から In[「Size」-1]までに NULL 文字がある場合は、そこまでしか「Out」に格納しません。

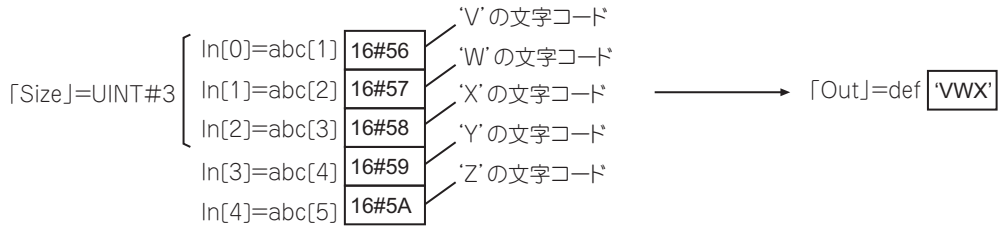
「Size」=UINT#3 の場合の例を、以下に示します。

LD



ST

```
def:=AryToString(abc[1], UINT#3);
```

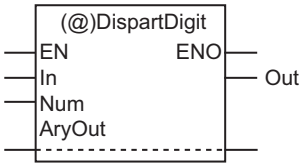


使用上の注意

- 「Size」の値が 0 の場合、「Out」は NULL 文字だけの文字列になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、In[]の配列領域を超える場合。

DispartDigit

ビット列を、4 ビット単位に分離します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DispartDigit	4 ビット分離	FUN		DispartDigit(In, Num, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	分離対象	入力	分離対象のビット列	データ型に従う	−	*1
Num	分離けた数		分離するけた数	0～「In」のビット数		1
AryOut[] 配列	分離結果配列	入出力	分離結果配列	16#00 ～ 16#0F	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Num						○														
AryOut[] 配列		○																		
Out	○																			

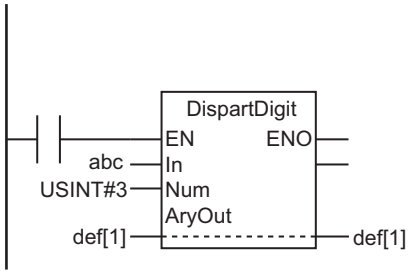
機能

分離対象「In」を 4 ビット単位（けた単位）に分離して、分離結果配列 AryOut[]に格納します。

まず、「In」を 4 ビット単位に分離します。そして、最下位けたの 4 ビットを AryOut[0]に格納します。その際、BYTE 型である AryOut[0]のビット 4 からビット 7 には 16#0 を付加します。
同じことを、分離けた数「Num」の数だけ行います。

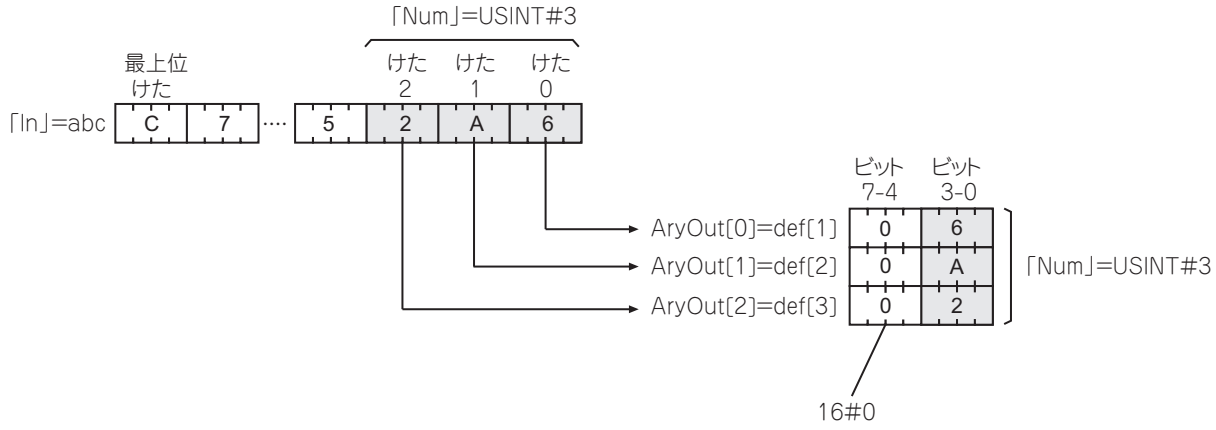
「Num」=USINT#3 の場合の例を、以下に示します。

LD



ST

```
DispartDigit(abc, USINT#3, def[1]);
```



参考

配列の各要素を 4 ビットずつ結合する場合は、「UniteDigit_** (P.2-472)」命令を使用してください。

使用上の注意

- 「Num」の値が 0 の場合は、AryOut[]の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Num」の値が、有効範囲外の場合。
  b)「Num」の値が、AryOut[]の配列領域を超える場合。

UniteDigit_**

4 ビット単位のデータを、ビット列に結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
UniteDigit_**	4 ビット結合グループ	FUN	(@)UniteDigit_** EN ENO In Out Num ただし、“**”はビット列のデータ型名	Out:=UniteDigit_**(In, Num); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	結合対象配列	入力	結合対象の配列	データ型に従う	−	*1
Num	結合けた数		結合するけた数	0 ～「Out」のビット数		1
Out	結合結果	出力	結合結果のビット列	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○																		
Num						○														
Out		○	○	○	○															

機能

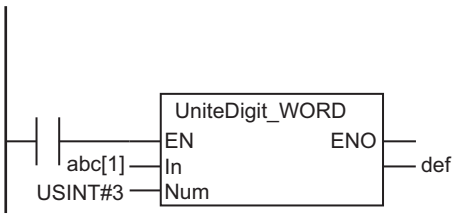
結合対象配列 In[]の、各要素の下位 4 ビットずつ（1 けた=4 ビット）を結合して、結合結果のビット列「Out」を作成します。

結合する配列要素数は、結合けた数「Num」で指定します。
まず、In[0]～In[「Num」-1]の下位 4 ビットずつを結合し、「Num」けたのビット列を作成します。
その上位に、{(「Out」のけた数）－「Num」個} の 16#0 を結合したものを、「Out」とします。

命令名称は、「Out」のデータ型によって、異なります。たとえば、「Out」が WORD 型の場合、命令名称は UniteDigit_WORD となります。

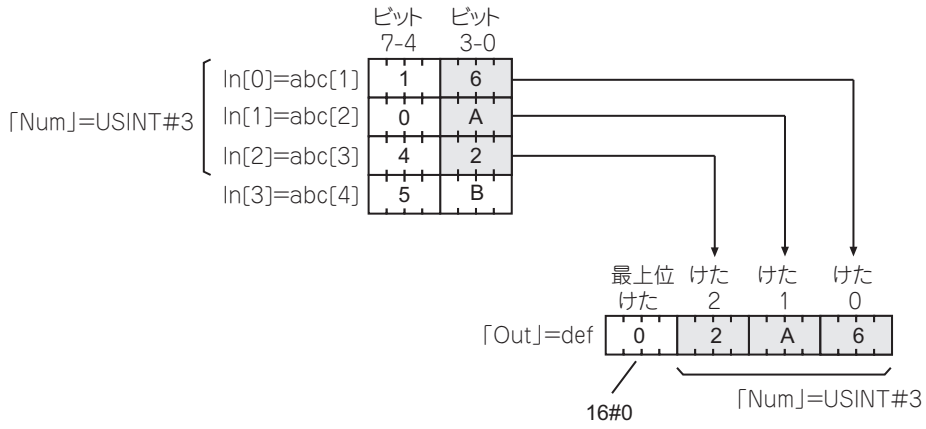
UniteDigit_WORD 命令で、「Num」=USINT#3 の場合の例を、以下に示します。

LD



ST

```
def:=UniteDigit_WORD(abc[1], USINT#3);
```



参考

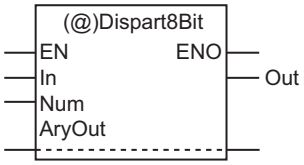
ビット列を、4 ビット単位に分離する場合は、「DispartDigit (P.2-470)」命令を使用してください。

使用上の注意

- 「Num」の値が 0 の場合、「Out」の値は 0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Num」の値が、有効範囲外の場合。
  b)「Num」の値が、In[]の配列領域を超える場合。

Dispart8Bit

ビット列を、1 バイト単位に分離します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Dispart8Bit	バイト単位データ分離	FUN		Dispart8Bit(In, Num, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	分離対象	入力	分離対象のビット列	データ型に従う	−	*1
Num	分離バイト数		分離するバイト数	0 ～「In」のバイト数		1
AryOut[] 配列	分離結果配列	入出力	分離結果配列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

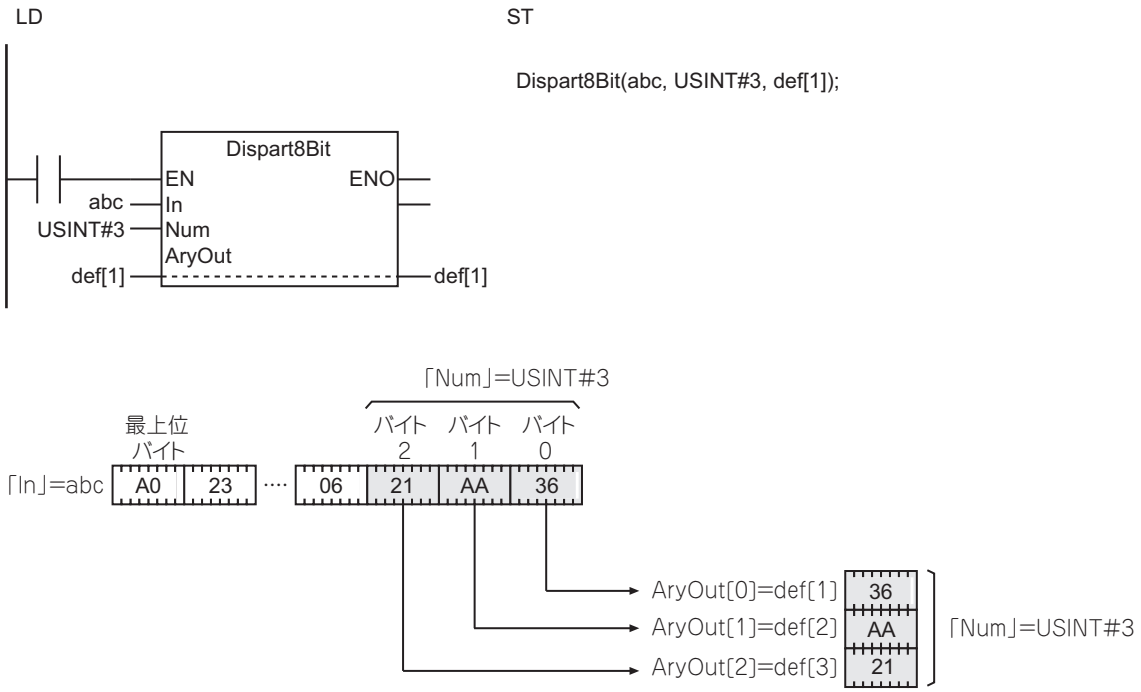
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Num						○														
AryOut[] 配列		○																		
Out	○																			

機能

分離対象「In」を 1 バイト単位で分離して、分離結果配列 AryOut[]に格納します。

まず、「In」を 1 バイト単位で分離します。そして、最下位バイトを AryOut[0]に格納します。
そして、つぎの 1 バイトを、同様に AryOut[1]に格納します。同じことを、分離バイト数「Num」の数だけ行います。

「Num」=USINT#3 の場合の例を、以下に示します。



参考

配列の各要素を 1 バイトずつ結合する場合は、「Unite8Bit_** (P.2-476)」命令を使用してください。

使用上の注意

- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、AryOut[]は変化しません。
  a)「Num」の値が、有効範囲外の場合。
  b)「Num」の値が、AryOut[]の配列領域を超える場合。

Unite8Bit_**

1 バイト単位のデータを、ビット列に結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Unite8Bit_**	バイト単位データ結合グループ	FUN	(@)Unite8Bit_** EN ENO In Out Num ただし、“**”はビット列のデータ型名	Out:=Unite8Bit_**(In, Num); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	結合対象配列	入力	結合対象の配列	データ型に従う	−	*1
Num	結合バイト数		結合するバイト数	0 ～「Out」のバイト数		1
Out	結合結果	出力	結合結果のビット列	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○																		
Num						○														
Out		○	○	○	○															

機能

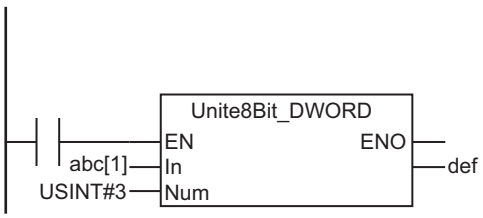
結合対象配列 In[]の、各要素を結合して、結合結果のビット列「Out」を作成します。

結合する配列要素数は、結合バイト数「Num」で指定します。まず、In[0]～In[「Num」-1]を結合し、「Num」バイトのビット列を作成します。
その上位に、{(「Out」のバイト数）－「Num」個} の 16#00 を結合したものを、「Out」とします。

命令名称は、「Out」のデータ型によって、異なります。たとえば、「Out」が DWORD 型の場合、命令名称は Unite8Bit_DWORD となります。

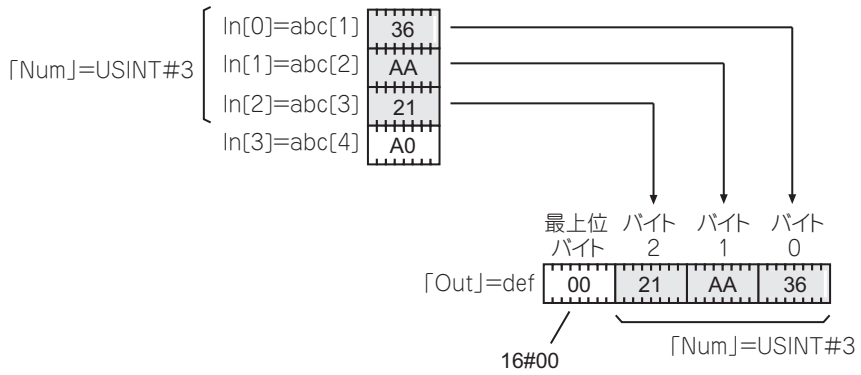
Unite8Bit_DWORD 命令で、「Num」=USINT#3 の場合の例を、以下に示します。

LD



ST

```
def:=Unite8Bit_DWORD(abc[1], USINT#3);
```



参考

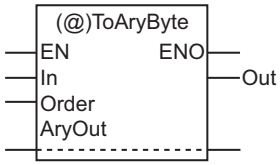
ビット列を、1 バイト単位に分離する場合は、「Dispart8Bit (P.2-474)」命令を使用してください。

使用上の注意

- 「Num」の値が 0 の場合、「Out」の値は 0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Num」の値が、有効範囲外の場合。
  b)「Num」の値が、In[]の配列領域を超える場合。

ToAryByte

変数を 1 バイト単位に分割して、BYTE 型配列に格納します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ToAryByte	バイト配列への変換	FUN		Out:=ToAryByte(In, Order, AryOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換対象	データ型に従う	−	*1
Order	変換順序		変換する順序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
AryOut[] 配列	変換結果配列	入出力	変換結果配列	データ型に従う	−	−
Out	変換結果要素数	出力	変換結果の要素数	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
Order	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-478)」参照																			
AryOut[] 配列		○																		
Out							○													

機能

変換対象「In」を 1 バイト単位に分割し、変換結果配列 AryOut[]の AryOut[0]から順に格納します。そして、AryOut[]に格納した要素数を、変換結果要素数「Out」に格納します。

「In」の値を、1 バイト単位に変換する際の順序は、変換順序「Order」で指定します。「Order」のデータ型は、列挙型_eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

「In」のデータ型が、2バイト以上のサイズの場合

「In」のデータ型が、2バイト以上のサイズの場合、「In」を1バイト単位に分割して、AryOut[]に格納します。
2バイト以上のサイズのデータ型とは、以下に示すものです。

種類	データ型
ビット列	WORD, DWORD, LWORD
整数	UINT, UDINT, ULINT, INT, DINT, LINT
実数	REAL, LREAL
時刻、持続時間、日付、文字列	TIME, DATE, TOD, DT, 2バイト以上のSTRING
その他	• 列挙型 • 全要素の合計サイズが2バイト以上の配列全体 • 2バイト以上の、配列の1要素 • 全メンバの合計サイズが2バイト以上の構造体全体 • 2バイト以上の、構造体の1メンバ

処理手順は、以下のとおりです。

1 **「In」の値を、まず1ワード（2バイト）単位に分割します。**

2 **そのうちの最下位ワード（2バイト）を、さらに1バイト単位に分割します。**

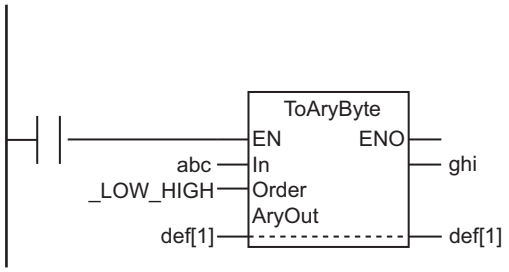
3 **「Order」の値が_LOW_HIGHなら、そのうちの下位バイトをAryOut[0]に、上位バイトをAryOut[1]に格納します。「Order」の値が_HIGH_LOWなら、上位バイトをAryOut[0]に、下位バイトをAryOut[1]に格納します。**

4 **同様に、つぎの1ワードを1バイト単位に分割して、AryOut[2]とAryOut[3]に格納します。**

5 **同じことを、「In」の値の最後まで行います。「In」が配列の場合は、同じことを「In」の最後の要素まで行います。**

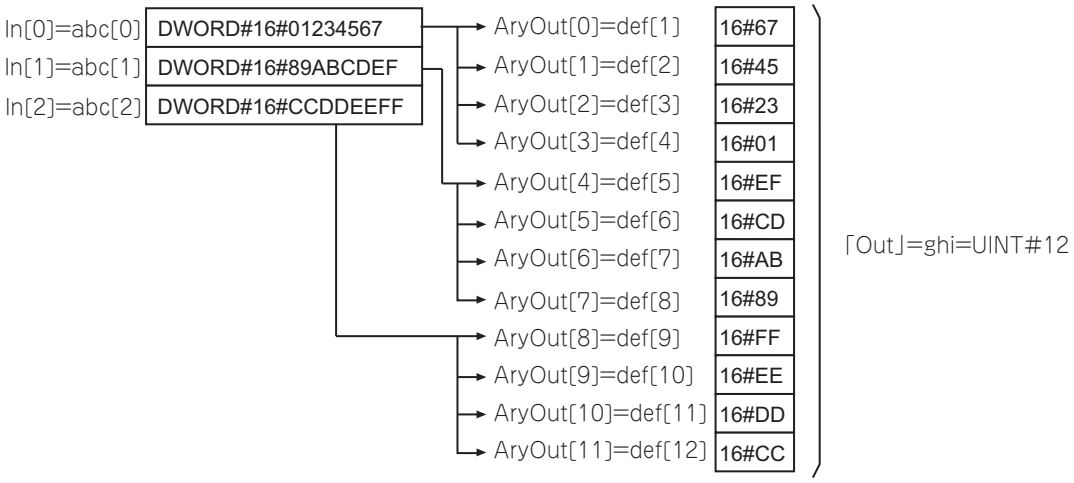
「In」がDWORD型で要素数3の配列全体、「Order」＝_LOW_HIGHの場合の例を、以下に示します。

LD

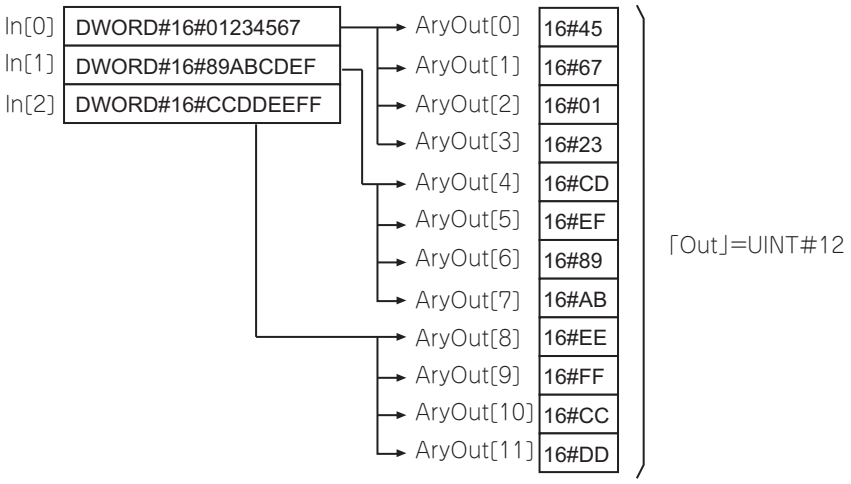


ST

```
ghi:=ToAryByte(abc, _LOW_HIGH, def[1]);
```



「In」が上記と同じで、「Order」＝_HIGH_LOW の場合の例を、以下に示します。



「In」のデータ型が、1 バイトのサイズの場合

「In」のデータ型が、1 バイトのサイズの場合、「In」を 1 バイト単位のまま、AryOut[]に格納します。
1 バイトのサイズのデータ型とは、以下に示すものです。

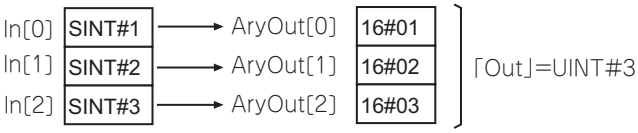
種類	データ型
ビット列	BYTE
整数	USINT, SINT
実数	なし
時刻、持続時間、日付、文字列	1 バイトの STRING
その他	• 全要素の合計サイズが 1 バイトの配列全体 • 1 バイトの、配列の 1 要素 • 全メンバの合計サイズが 1 バイトの構造体全体 • 1 バイトの、構造体の 1 メンバ

格納の方法は、以下のとおりです。

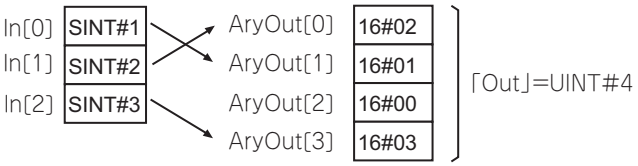
「Order」の値	「In」が配列かどうか	AryOut[]への格納のしかた
_LOW_HIGH	配列ではない	「In」の値を、AryOut[0]に格納します。
	配列である	In[i]の値を、AryOut[i]に格納します。

「Order」の値	「In」が配列かどうか	AryOut[]への格納のしかた
_HIGH_LOW	配列ではない	「In」の値を、AryOut[1]に格納します。 16#00 を、AryOut[0]に格納します。
	配列である	In[i]（「i」は偶数）を AryOut[i+1]に格納します。 In[i]（「i」は奇数）を AryOut[i-1]に格納します。 In[]の要素数 n が奇数なら、最後に AryOut[n-1]に 16#00 を格納します。

「In」が SINT 型で要素数 3 の配列、「Order」＝_LOW_HIGH の場合の例を、以下に示します。



「In」が上記と同じで、「Order」＝_HIGH_LOW の場合の例を、以下に示します。

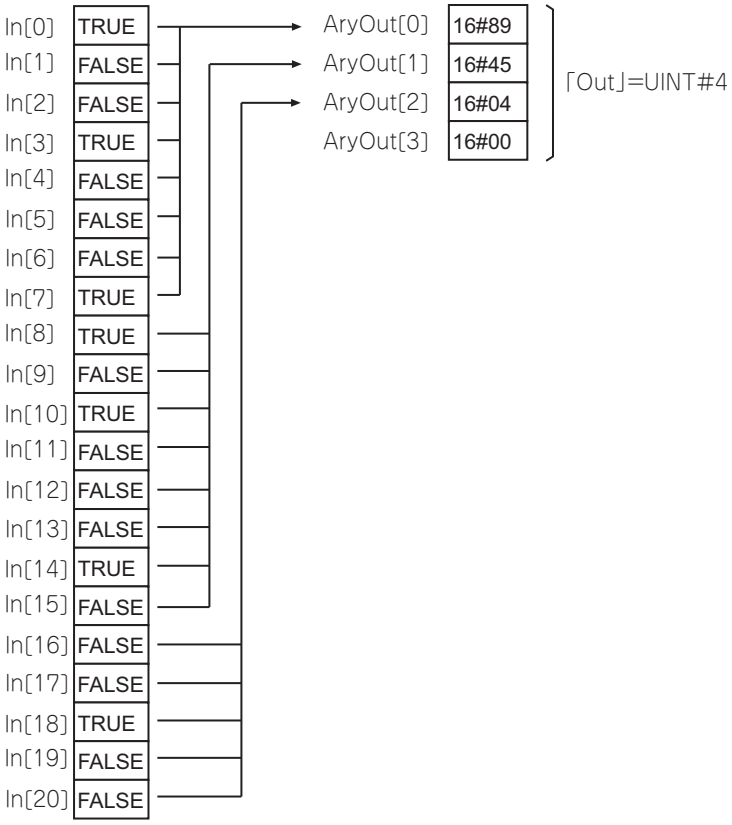


「In」のデータ型が、BOOL の場合

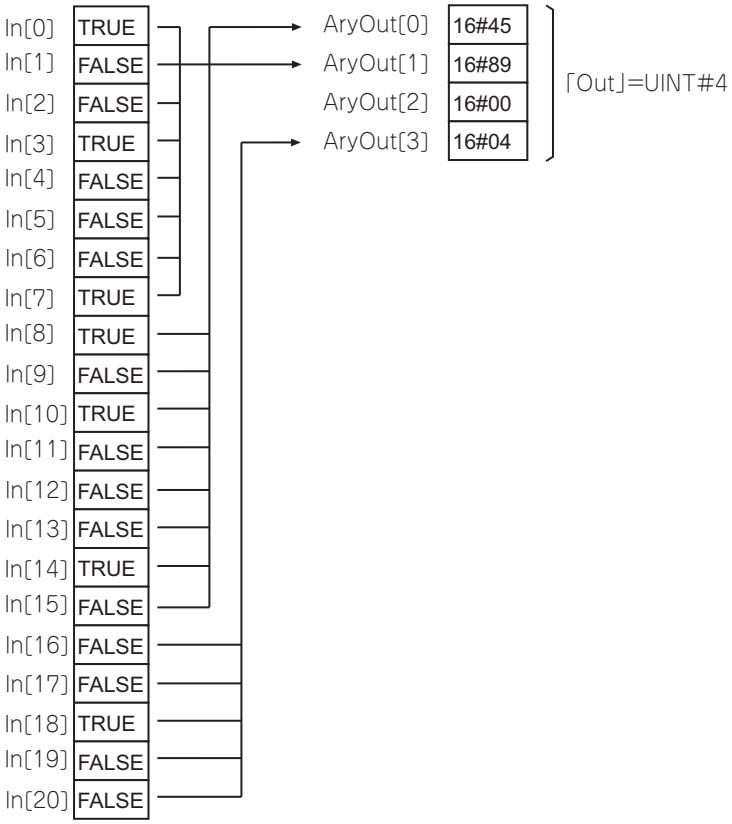
「In」のデータ型が、BOOL（1 ビット）の場合、AryOut[]への格納のしかたは、以下のようになります。

「Order」の値	「In」が配列かどうか	AryOut[]への格納のしかた
_LOW_HIGH	配列ではない	「In」の値と 16#00 との論理和を AryOut[0]に格納します。
	配列である	In[0]～In[7]の値を連結して、AryOut[0]に格納します。 In[8]～In[15]の値を連結して、AryOut[1]に格納します。以下、同様に格納していきます。 In[]が 8 個に足りない場合は、上位ビットに FALSE を付加します。 「Out」の値は、必ず偶数にします。足りないビットの値は、すべて FALSE になります。
_HIGH_LOW	配列ではない	「In」の値と 16#00 との論理和を、AryOut[1]に格納します。 16#00 を、AryOut[0]に格納します。
	配列である	In[0]～In[7]の値を連結して AryOut[1]に格納します。 In[8]～In[15]の値を連結して AryOut[0]に格納します。 以下、同様に格納していきます。 「Out」の値は、必ず偶数になります。足りないビットの値は、すべて FALSE になります。

「In」が BOOL 型で要素数 21 の配列、「Order」＝_LOW_HIGH の場合の例を、以下に示します。



「In」が上記と同じで、「Order」＝_HIGH_LOW の場合の例を、以下に示します。

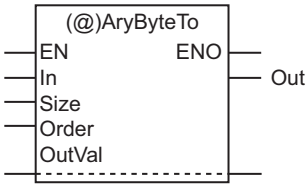


使用上の注意

- 「In」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「In」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- 「In」が STRING 型の場合、文字列を数値に変換することはしません。変数の内容をビット列と見なして、バイト配列に変換します。
- 「In」が構造体の場合、構成に応じて、各メンバ間にある調整用の領域が AryOut[]に挿入されることがあります。
- 「Order」の値が_HIGH_LOW で「In」の合計サイズが奇数バイトの場合、「In」の最後に 16#00 を付け加えて偶数バイトにしてから、変換を開始します。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」や AryOut[]は変化しません。
  a)「Order」の値が、有効範囲外の場合。
  b) 変換結果が、AryOut[]の配列領域を超える場合。

AryByteTo

BYTE 型配列要素を結合して、変数に格納します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryByteTo	バイト配列からの変換	FUN		AryByteTo(In, Size, Order, OutVal);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	変換対象配列	入力	変換対象の配列	データ型に従う	−	*1
Size	変換要素数		In[]の変換する要素数			1
Order	変換順序		変換する順序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
OutVal	変換結果	入出力	変換結果	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○																		
Size							○													
Order	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-484)」参照																			
OutVal	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
Out	○																			

機能

変換対象配列 In[]の、先頭の「Size」個の要素を、変換結果「OutVal」のデータ型のサイズに合わせて結合し、「OutVal」に格納します。

In[]の要素を結合する際の順序は、変換順序「Order」で指定します。「Order」のデータ型は、列挙型 _eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

「OutVal」のデータ型が、2 バイト以上のサイズの場合

「OutVal」のデータ型が、2 バイト以上のサイズの場合、In[]を「OutVal」のデータ型のサイズ分だけ結合して、「OutVal」に格納します。
2 バイト以上のサイズのデータ型とは、以下に示すものです。

種類	データ型
ビット列	WORD, DWORD, LWORD
整数	UINT, UDINT, ULINT, INT, DINT, LINT
実数	REAL, LREAL
時刻、持続時間、日付、文字列	TIME, DATE, TOD, DT, 2 バイト以上の STRING
その他	• 列挙型 • 全要素の合計サイズが 2 バイト以上の配列全体 • 2 バイト以上の、配列の 1 要素 • 全メンバの合計サイズが 2 バイト以上の構造体全体 • 2 バイト以上の、構造体の 1 メンバ

処理手順は、以下のとおりです。

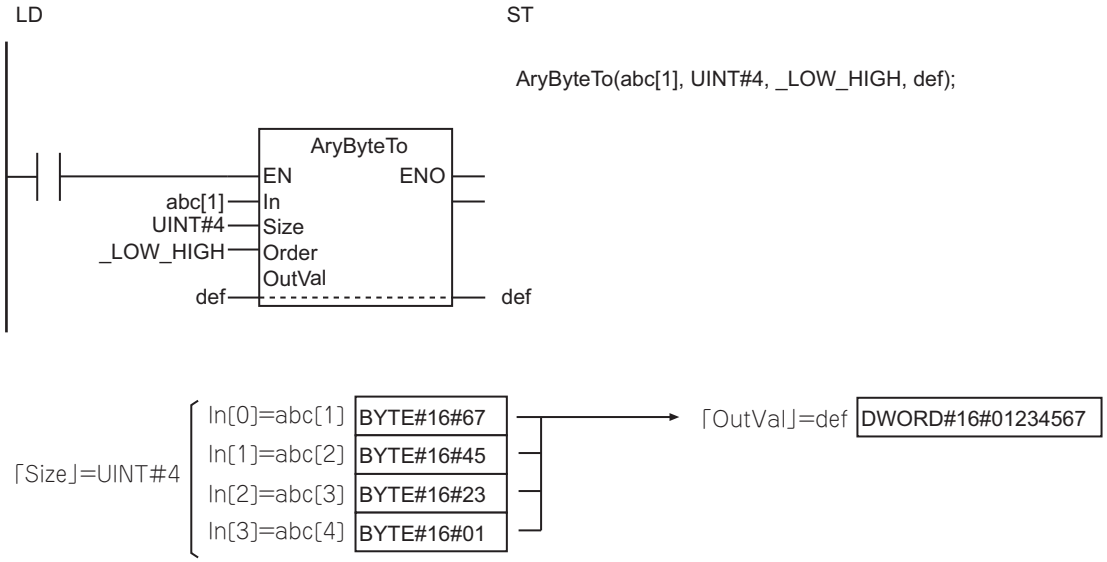
1 **In[0]と In[1]を、「Order」の値にしたがって結合し、1 ワード（2 バイト）のデータを作ります。**
「Order」の値が_LOW_HIGH なら、上位バイトが In[1]、下位バイトが In[0]になります。
「Order」の値が_HIGH_LOW なら、上位バイトを In[0]、下位バイト In[1]になります。

2 **同様に、In[2]と In[3]以降も結合し、1 ワードのデータを複数作ります。**

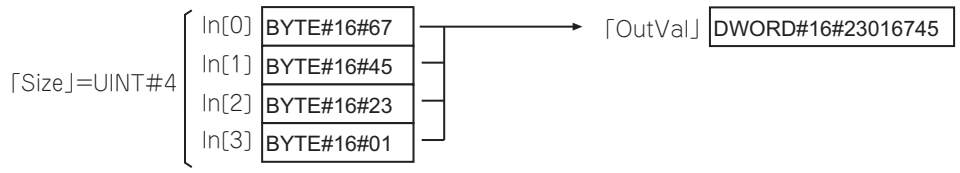
3 **できあがった複数の 1 ワードのデータを、「OutVal」のデータ型のサイズに合うようにさらに結合します。たとえば、「OutVal」が DWORD 型の場合、1 ワードのデータを 4 個結合します。**

4 **できあがったデータを、「OutVal」に格納します。**

「OutVal」が DWORD 型、「Size」=UINT#4、「Order」=_LOW_HIGH の場合の例を、以下に示します。



「OutVal」が上記と同じで、「Size」=UINT#4、「Order」=_HIGH_LOW の場合の例を、以下に示します。



「OutVal」のデータ型が、1 バイトのサイズの場合

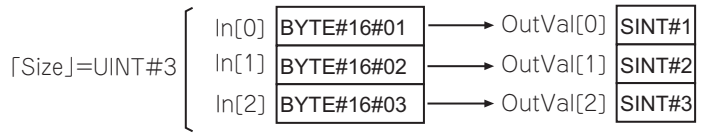
「OutVal」のデータ型が、1 バイトのサイズの場合、In[]を 1 バイト単位のまま、「OutVal」に格納します。
1 バイトのサイズのデータ型とは、以下に示すものです。

種類	データ型
ビット列	BYTE
整数	USINT, SINT
実数	なし
時刻、持続時間、日付、文字列	1 バイトの STRING
その他	• 全要素の合計サイズが 1 バイトの配列全体 • 1 バイトの、配列の 1 要素 • 全メンバの合計サイズが 1 バイトの構造体全体 • 1 バイトの、構造体の 1 メンバ

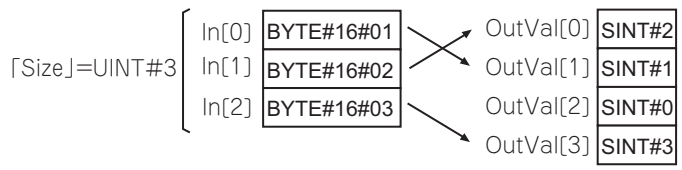
格納の方法は、以下のとおりです。

「Order」の値	「OutVal」が配列かどうか	「OutVal」への格納のしかた
_LOW_HIGH	配列ではない	In[0]の値を、「OutVal」に格納します。
	配列である	In[i]の値を、OutVal[i]に格納します。
_HIGH_LOW	配列ではない	In[1]の値を、「OutVal」に格納します。
	配列である	In[i]（「i」は偶数）を OutVal[i+1]に格納します。 In[i]（「i」は奇数）を OutVal[i-1]に格納します。 「Size」の値が奇数なら、OutVal[「Size」]まで格納され、 OutVal[「Size」-1]には 16#00 を格納します。

「OutVal」が SINT 型で要素数 3 の配列、「Size」＝ UINT#3、「Order」＝_LOW_HIGH の場合の例を、以下に示します。



「OutVal」と「Size」が上記と同じで、「Order」＝_HIGH_LOW の場合の例を、以下に示します。

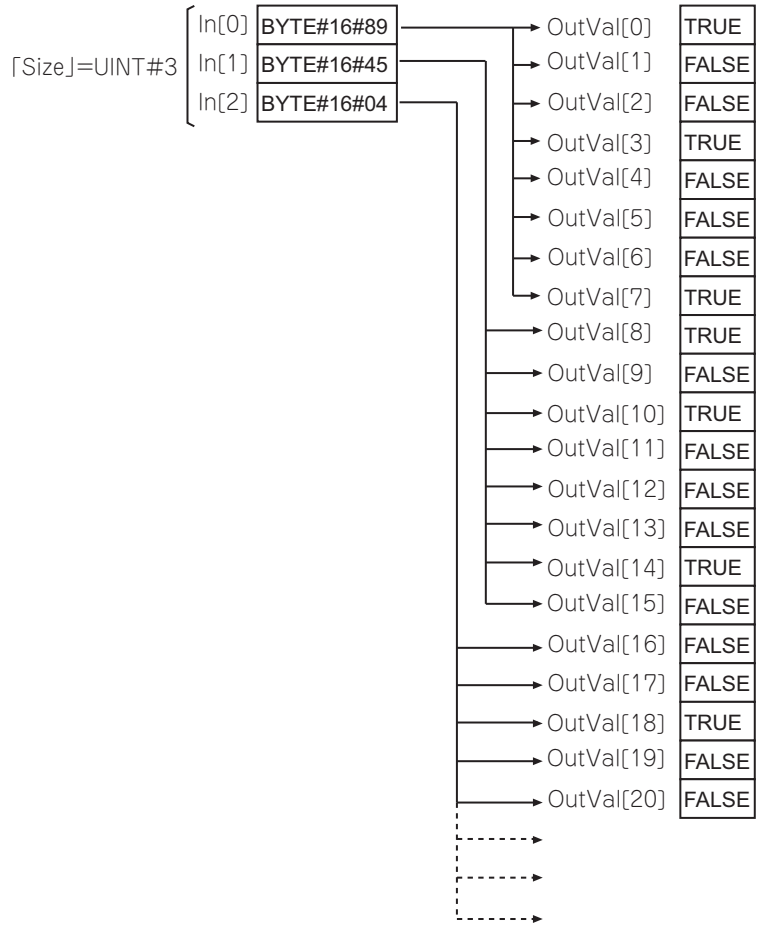


「OutVal」のデータ型が、BOOL の場合

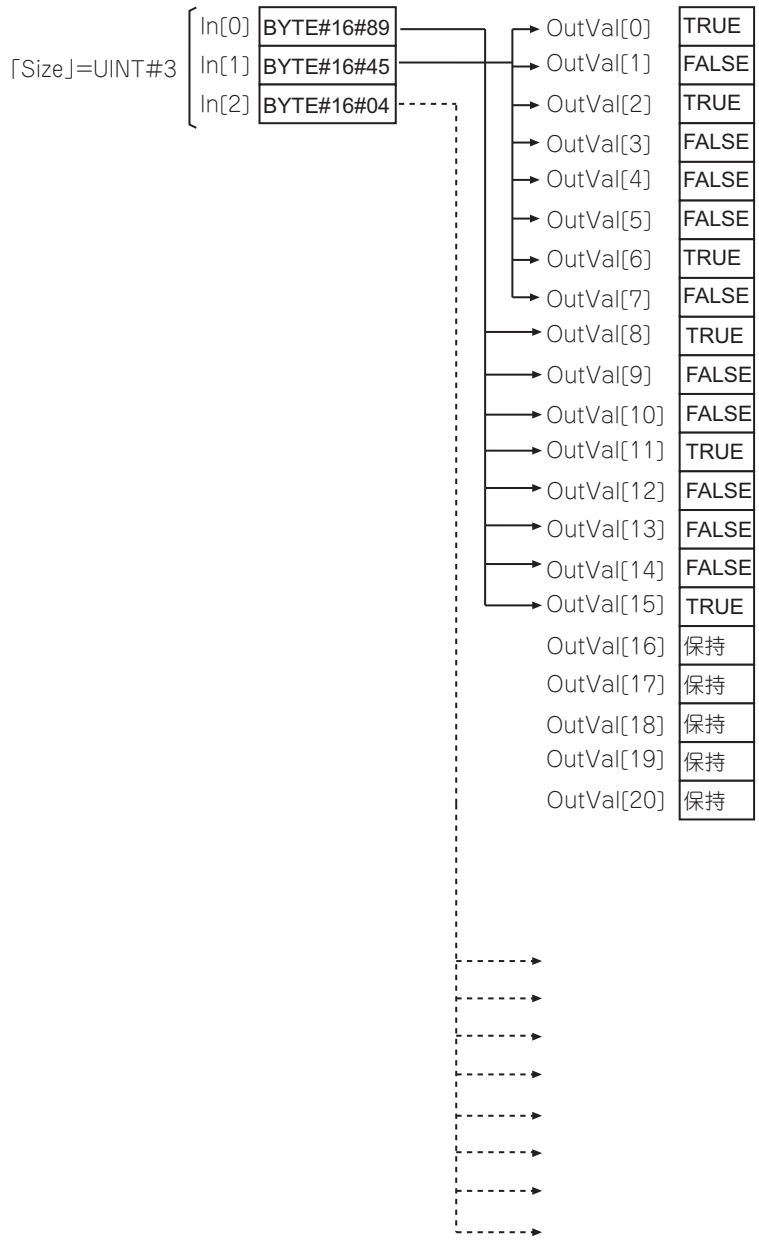
「OutVal」のデータ型が、BOOL（1 ビット）の場合、「OutVal」への格納のしかたは、以下のようになります。

「Order」の値	「OutVal」が配列かどうか	「OutVal」への格納のしかた
_LOW_HIGH	配列ではない	In[0]のビット 0 の値を、「OutVal」に格納します。
	配列である	In[0]の値を分解して、OutVal[0]～OutVal[7]に格納します。 In[1]の値を分解して、OutVal[8]～OutVal[15]に格納します。 以下、同様に格納していきます。 余ったビットは破棄します。
_HIGH_LOW	配列ではない	In[1]のビット 0 の値を、「OutVal」に格納します。
	配列である	In[0]の値を分解して OutVal[8]～OutVal[15]に格納します。 In[1]の値を分解して OutVal[0]～OutVal[7]に格納します。 以下、同様に格納していきます。 余ったビットは破棄します。

OutVal[]が BOOL 型で要素数 21 の配列、「Size」=UINT#3、「Order」=_LOW_HIGH の場合の例を、以下に示します。



OutVal[]と「Size」が上記と同じで、「Order」=_HIGH_LOW の場合の例を、以下に示します。



使用上の注意

- 「OutVal」が構造体全体の場合、構成に応じて In[]の値の一部が、各メンバ間にある調整用の領域に挿入されることがあります。
- 「Size」が「OutVal」のサイズに満たない場合はエラーとはならず、指定バイトデータが「OutVal」に格納されます。不足部分については、命令実行前の値が保持されます。
  前回実行時よりサイズを小さくする場合は、事前に Clear（初期化）命令などで変数をクリアしてから使用してください。
- 「Size」の値が 0 の場合、「Out」の値は TRUE になり、「OutVal」は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「OutVal」は変化しません。
  a）「Order」の値が、有効範囲外の場合。
  b）「Size」の値が、In[]の要素数を超える場合。

SizeOfAry

配列要素数を、取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SizeOfAry	配列要素数取得	FUN	(@)SizeOfAry EN ENO In Out	Out:=SizeOfAry(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	配列	入力	配列	データ型に従う	−	*1
Out	要素数	出力	要素数	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Out							○													

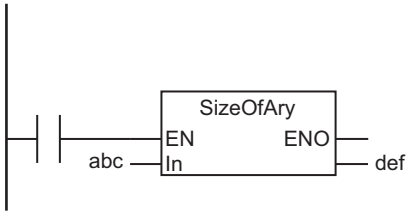
機能

配列 In[]の、要素数を取得します。
入力パラメータには、array[0]のような配列要素名ではなく、「array」のような配列名を指定してください。

動作例を、以下に示します。

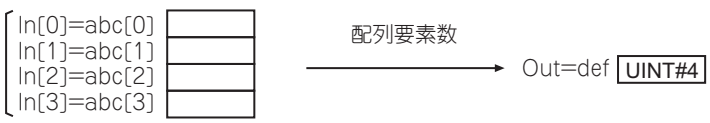
名称	データ型
abc	ARRAY[0..3] OF INT

LD



ST

```
def:=SizeOfAry(abc);
```

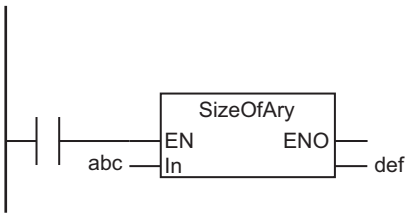


参考

In[]は、2 次元以上の配列でもかまいません。その場合、「Out」には、In[]の全要素数が格納されます。
たとえば、In[]に渡す入力パラメータの配列が ARRAY[0..1,0..2]の場合、「Out」の値は、UINT#6 になります。

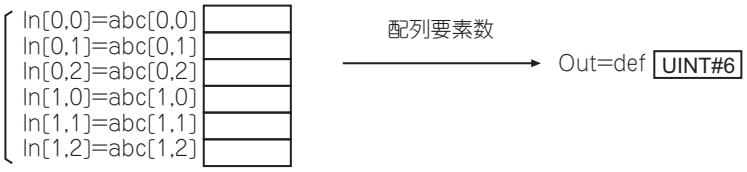
名称	データ型
abc	ARRAY[0..1,0..2] OF BOOL

LD



ST

```
def:=SizeOfAry(abc);
```



PackWord

2 つの 1 バイトデータを 1 つの 2 バイトデータに結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PackWord	2 バイト結合	FUN	(@)PackWord EN ENO High Out Low	Out:=PackWord(High, Low);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
High	バイトデータ H	入力	ビット 15-8 に格納するバイトデータ	データ型に従う	−	0
Low	バイトデータ L		ビット 7-0 に格納するバイトデータ	データ型に従う	−	0
Out	結合データ	出力	2 バイトデータ	データ型に従う	−	−

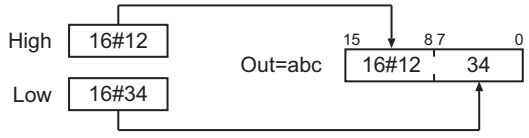
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
High		○																		
Low		○																		
Out			○																	

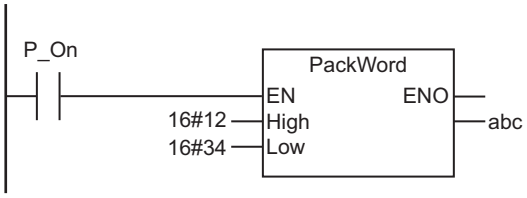
機能

2 つの 1 バイトデータを 1 つの 2 バイトデータに結合します。
「High」で指定したデータをビット 15-8 に、「Low」で指定したデータをビット 7-0 に格納します。

LD

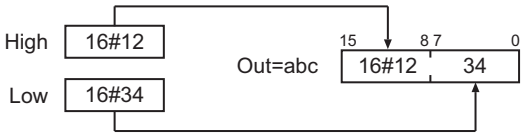
「High」=16#12、「Low」=16#34 の場合の例を、以下に示します。
変数「abc」は 16#1234 になります。





ST

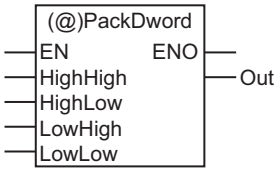
「High」=16#12、「Low」=16#34 の場合の例を、以下に示します。
変数「abc」は 16#1234 になります。




```
abc:=PackWord(16#12, 16#34);
```

PackDword

4 つの 1 バイトデータを 1 つの 4 バイトデータに結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PackDword	4 バイト結合	FUN		Out:=PackDword(HighHigh, HighLow, LowHigh, LowLow);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
HighHigh	バイトデータ HH	入力	ビット 31-24 に格納するバイトデータ	データ型に従う	−	0
HighLow	バイトデータ HL		ビット 23-16 に格納するバイトデータ	データ型に従う	−	0
LowHigh	バイトデータ LH		ビット 15-8 に格納するバイトデータ	データ型に従う	−	0
LowLow	バイトデータ LL		ビット 7-0 に格納するバイトデータ	データ型に従う	−	0
Out	結合データ	出力	4 バイトデータ	データ型に従う	−	−

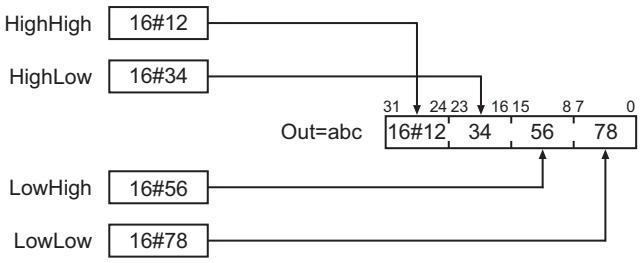
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
HighHigh		○																		
HighLow		○																		
LowHigh		○																		
LowLow		○																		
Out				○																

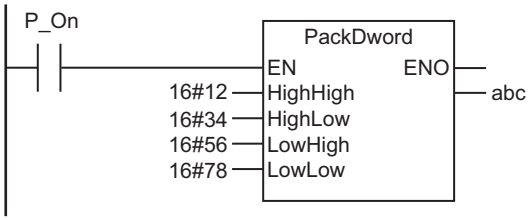
機能

4 つの 1 バイトデータを 1 つの 4 バイトデータに結合します。
「HighHigh」で指定したデータをビット 31-24 に、「HighLow」で指定したデータをビット 23-16 に、「LowHigh」で指定したデータをビット 15-8 に、「LowLow」で指定したデータをビット 7-0 に格納します。

LD

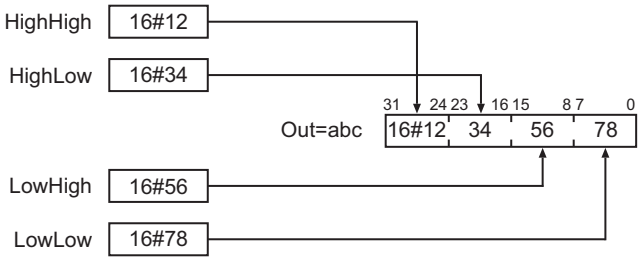
「HighHigh」=16#12、「HighLow」=16#34、「LowHigh」=16#56、「LowLow」=16#78 の場合の例を、以下に示します。
変数「abc」は 16#12345678 になります。





ST

「HighHigh」=16#12、「HighLow」=16#34、「LowHigh」=16#56、「LowLow」=16#78 の場合の例を、以下に示します。
変数「abc」は 16#12345678 になります。




```
abc:=PackDword(16#12, 16#34, 16#56, 16#78);
```

LOWER_BOUND / UPPER_BOUND

LOWER_BOUND ： 配列次元の開始番号を取得します。
UPPER_BOUND ： 配列次元の終了番号を取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LOWER_BOUND	配列開始番号取得	FUN	(@)LOWER_BOUND EN ENO ARR Out DIM	Out:=LOWER_BOUND(ARR, DIM);
UPPER_BOUND	配列終了番号取得	FUN	(@)UPPER_BOUND EN ENO ARR Out DIM	Out:=UPPER_BOUND(ARR, DIM);

バージョン関連情報

本命令は、CPU ユニット Ver.1.18 以降、かつ Sysmac Studio Ver.1.22 以降で、使用可能です。

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ARR	対象配列	入力	配列の開始番号、または終了番号を取得する配列を指定してください。*1	−	−	−
DIM	次元		次元を指定します。*2	−	−	1
Out	戻り値	出力	LOWER_BOUND: 開始番号 UPPER_BOUND: 終了番号	データ型に従う	−	−

*1. array[0]のような配列要素名ではなく、「array」のような配列名を指定してください。
*2. 1 次元めの場合、「1」を指定します。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ARR	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
DIM												○								
Out												○								

機能

LOWER_BOUND 命令は、「ARR」で指定する配列変数の「DIM」で指定する次元の開始番号を取得します。
同様に、UPPER_BOUND 命令は、「ARR」で指定する配列変数の「DIM」で指定する次元の終了番号を取得します。

関連するシステム定義変数

変数名	名称	データ型	内容
P_PRGER	命令エラー発生フラグ	BOOL	TRUE: 異常発生時。FALSE にされるまで TRUE となります。 FALSE: ユーザプログラムによって FALSE にします。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
- 「ARR」が配列ではない場合。
- 「DIM」で指定する値が「0」以下、または「ARR」の次元の範囲を超える場合。

サンプルプログラム

配列の総和の計算

ここでは、1 次元の可変長配列変数の定義方法と、可変長配列変数の次元の開始番号と終了番号を取得する方法についてのサンプルプログラムを示します。

● ユーザ定義の FUN プログラム(Sum)

内部変数	名称	データ型	初期値	コメント
	i	DINT		

入力／出力変数	名称	入／出力	データ型	コメント
	EN	入力	BOOL	
	ENO	出力	BOOL	
	a	入出力	ARRAY[*] OF INT	

戻り値	名称	データ型	初期値	コメント
	Sum	INT		

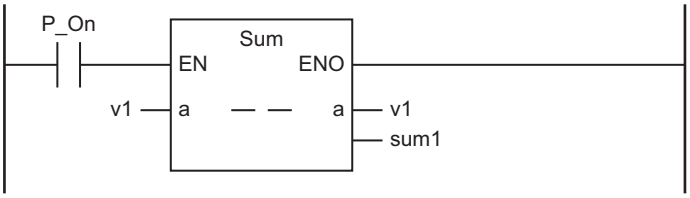
```
Sum := 0;
FOR i := LOWER_BOUND(a,1) TO UPPER_BOUND(a,1) DO
    Sum := Sum + a[i];
END_FOR;
```

● 呼び出し側プログラム

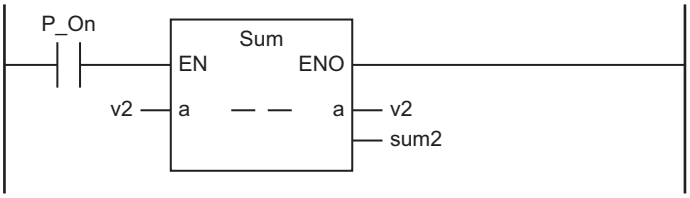
内部変数	名称	データ型	初期値	コメント
	v1	ARRAY[0..4] OF INT	[1,2,3,4,5]	

内部変数	名称	データ型	初期値	コメント
	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



2×2 行列の加算

ここでは、多次元の可変長配列変数の定義方法と、可変長配列変数の多次元の LOWER_BOUND 命令および UPPER_BOUND 命令の使い方についてのサンプルプログラムを示します。

● ユーザ定義の FUN プログラム(Matrix_Add)

内部変数	名称	データ型	初期値	コメント
	i	DINT		
	j	DINT		
	m1	DINT		
	m2	DINT		
	n1	DINT		
	n2	DINT		

入力／出力変数	名称	入／出力	データ型	コメント
	EN	入力	BOOL	
	ENO	出力	BOOL	
	A	入出力	ARRAY[*,*] OF DINT	
	B	入出力	ARRAY[*,*] OF DINT	
	C	入出力	ARRAY[*,*] OF DINT	

戻り値	名称	データ型	初期値	コメント
	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;
```

● **呼び出し側プログラム**

内部変数	名称	データ型	初期値	コメント
	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		

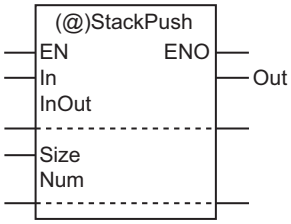
```
// Z = X + Y = |0 1| + |1 2| = |1 3|
//             |2 3|   |3 4|   |5 7|
Matrix_Add(X, Y, Z);
```

スタック／テーブル命令

命令	名称	ページ
StackPush	スタックデータ格納	P.2-500
StackFIFO / StackLIFO	先入れ先出し／後入れ先出し	P.2-509
StackIns	スタックデータ挿入	P.2-512
StackDel	スタックデータ削除	P.2-514
RecSearch	レコード検索	P.2-516
RecRangeSearch	範囲指定レコード検索	P.2-520
RecSort	レコードソート	P.2-525
RecNum	レコード数取得	P.2-531
RecMax / RecMin	レコード最大値検索／レコード最小値検索	P.2-533

StackPush

スタックに、値を格納します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StackPush	スタックデータ格納	FUN		StackPush(In, InOut, Size, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	入力値	入力	スタックに入力する値、構造体、構造体の 1 メンバ	データ型に従う	−	−
Size	スタックの要素数		スタックの配列要素数			1
InOut[] 配列	スタック配列	入出力	スタックを構成する配列	データ型に従う	−	−
Num	スタックの格納要素数		スタックに格納されている要素数			
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

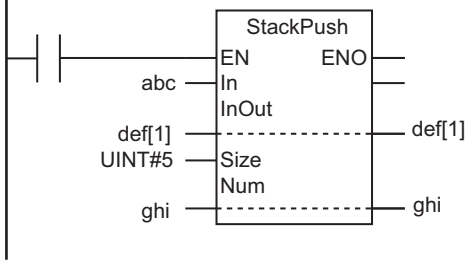
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、構造体全体、構造体の 1 メンバも指定可能																			
Size							○													
InOut[] 配列	「In」と同じデータ型を要素とする配列																			
Num							○													
Out	○																			

機能

スタック配列 InOut[]に、現在、格納要素数「Num」個の要素が格納されていると見なします。そして、入力値「In」を、そのつぎの要素 InOut[「Num」]に上書します。
その後、「Num」をインクリメントします。
スタックの要素数「Size」には、InOut[]のうち、スタックとして使用する要素数を指定します。

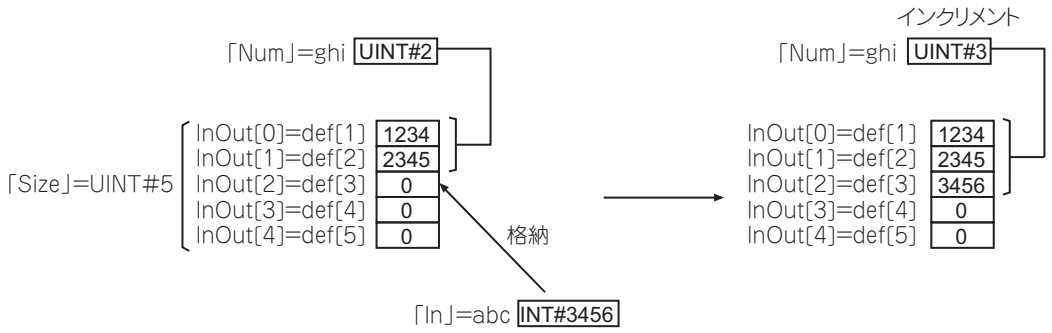
「Size」=UINT#5、「Num」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
StackPush(abc, def[1], UINT#5, ghi);
```



参考

スタックの最下位または最上位の値を取り出す場合は、「StackFIFO / StackLIFO (P.2-509)」命令を使用してください。

使用上の注意

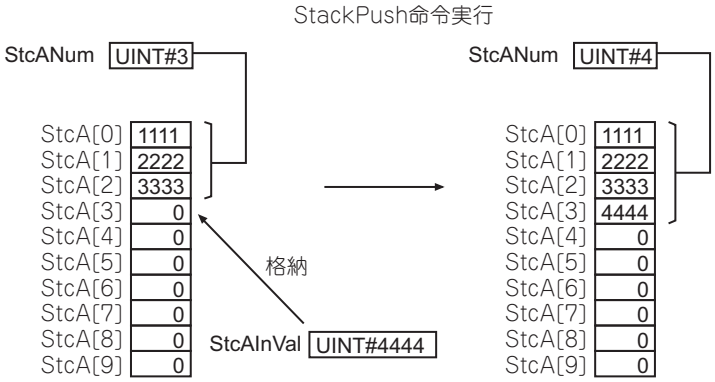
- 「In」と、InOut[]の要素のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- InOut[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Size」の値が 0 の場合、InOut[]、「Num」の値は変化しません。
- 「In」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「In」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が 0 以外で、「Num」≧「Size」の場合。
  b)「Size」の値が、InOut[]の配列領域を超える場合。
  c)「In」、InOut[]が STRING 型で、「In」のバイト数が InOut[]のサイズを超える場合。

サンプルプログラム

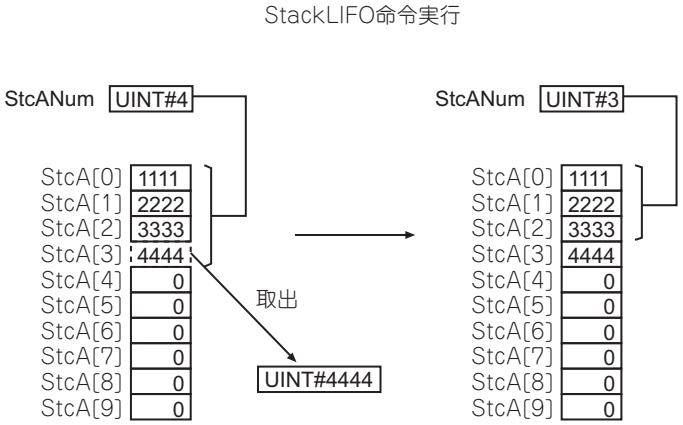
配列変数 StcA[0..9]をスタックとします。まず準備として、スタックに 3 個のデータ(UINT#1111、UINT#2222、UINT#3333)を格納しておきます。

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

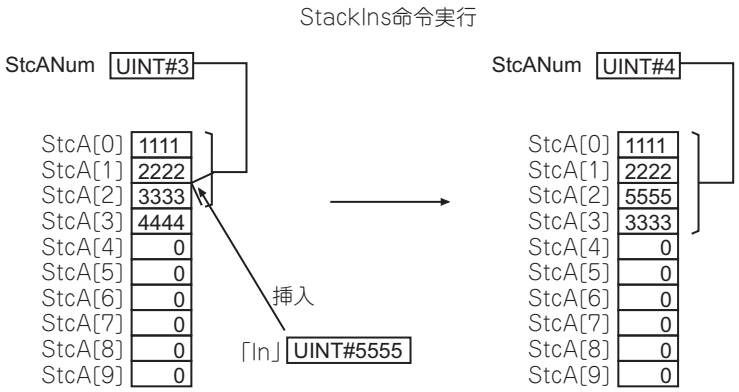
StackPush 命令を使って、スタックの最上位 StcA[3]に、新しいデータ(UINT#4444)を 1 個格納します。スタック内のデータ数は、4 個になります。



つぎに、StackLIFO 命令を使って、スタックの最上位 StcA[3]のデータを 1 個取り出します。スタックのデータ数は、3 個になります。



最後に、StackIns 命令を使って、StcA[1]と StcA[2]の間にデータ(UINT#5555)を 1 個挿入します。スタックのデータ数は、4 個になります。



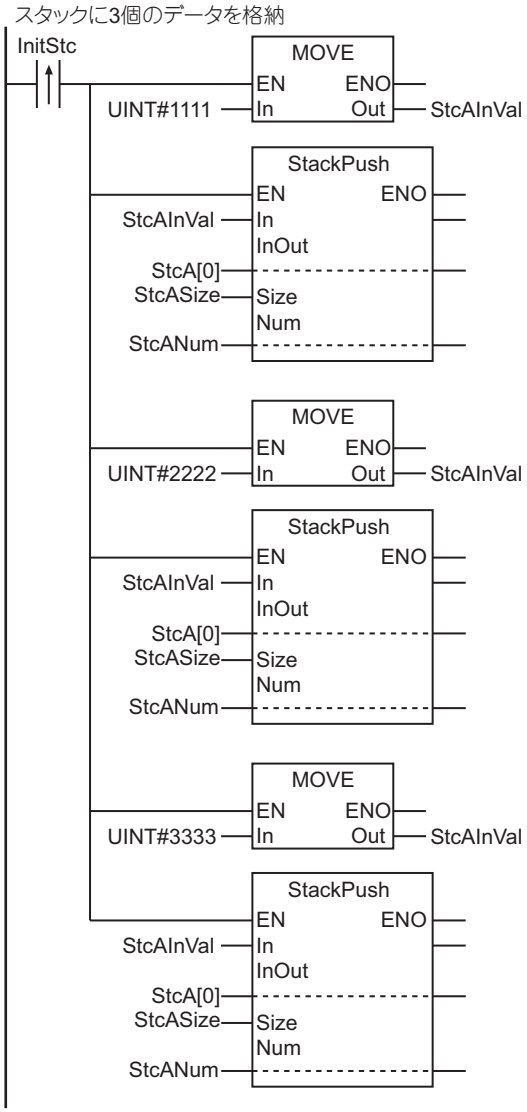
LD

名称	データ型	初期値	コメント
InitStc	BOOL	FALSE	スタックの初期化条件
StcANum	UINT	0	スタックの格納要素数
StcA	ARRAY[0..9] OF UINT	[10(0)]	スタック配列
StcASize	UINT	0	スタックの要素数
SetParaPush	BOOL	FALSE	StcAInVal をセットする実行条件
StcAInVal	UINT	0	StackPush で追加する値
StcAPushStat	BOOL	FALSE	StackPush の実行条件
StackPush_err	BOOL	FALSE	StackPush の異常フラグ
StcALIFOStat	BOOL	FALSE	StackLIFO の実行条件
StcAOutVal	UINT	0	StackLIFO で取り出す値
StackLIFO_err	BOOL	FALSE	StackLIFO の異常フラグ
SetParaIns	BOOL	FALSE	StcAInsVal と StcAOffset をセットする実行条件
StcAInsVal	UINT	0	StackIns で挿入する値
StcAOffset	UINT	0	StackIns のオフセット値
StcAInsStat	BOOL	FALSE	StackIns の実行条件
StackIns_err	BOOL	FALSE	StackIns の異常フラグ

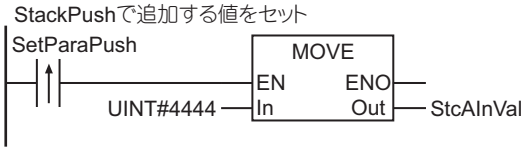
スタックの初期化



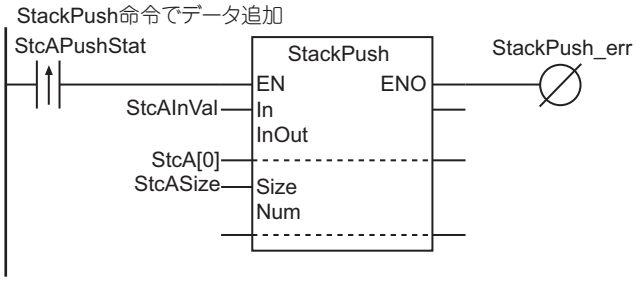
スタックに3個のデータを格納



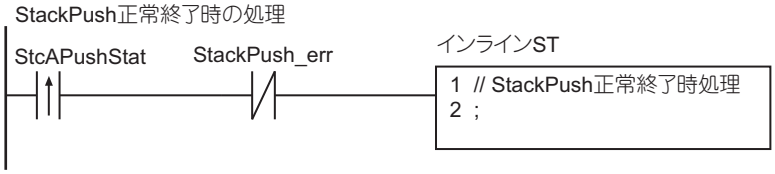
StackPushで追加する値をセット



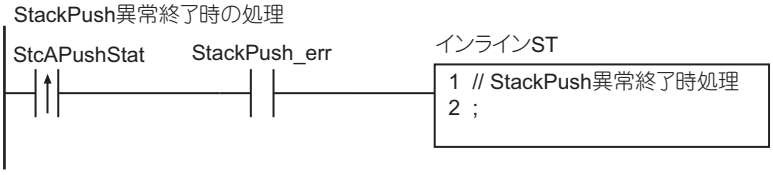
StackPush命令でデータ追加



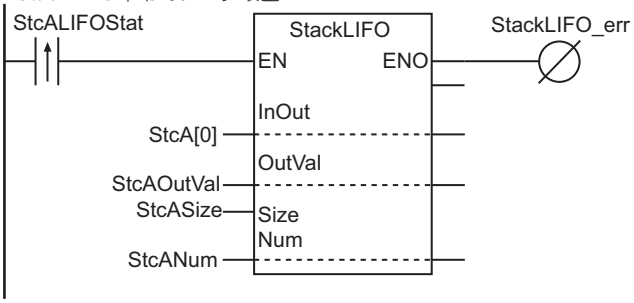
StackPush正常終了時の処理



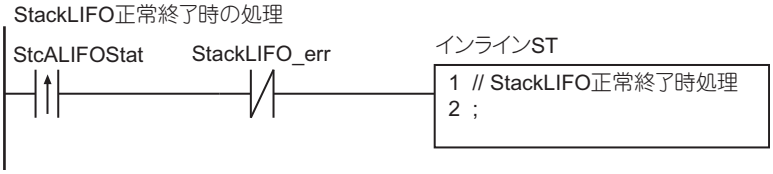
StackPush異常終了時の処理



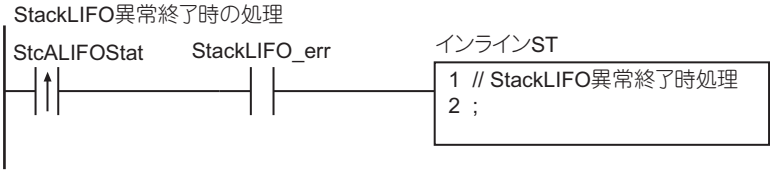
StackLIFO命令でデータ取出



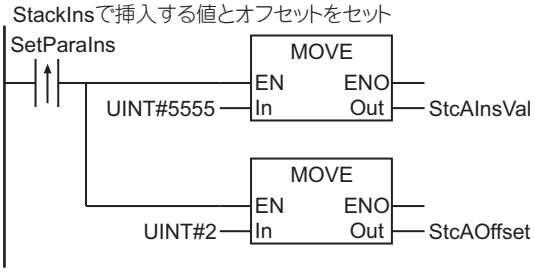
StackLIFO正常終了時の処理



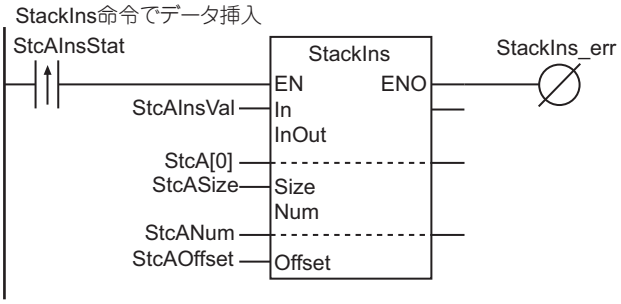
StackLIFO異常終了時の処理

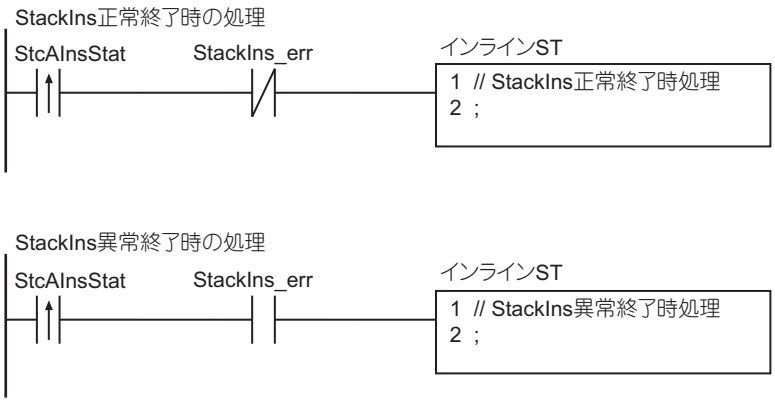


StackInsで挿入する値とオフセットをセット



StackIns命令でデータ挿入





● **インライン ST の内容**

```
StcANum:=0;
Clear(StcA);
StcASize:=SizeOfAry(StcA);
```

ST

名称	データ型	初期値	コメント
InitStc	BOOL	FALSE	スタックの初期化条件
preInitStc	BOOL	FALSE	前タスク周期の InitStc の値
StcANum	UINT	0	スタックの格納要素数
StcA	ARRAY[0..9] OF UINT	[10(0)]	スタック配列
StcASize	UINT	0	スタックの要素数
StcAPushStat	BOOL	FALSE	StackPush の実行条件
preStcAPushStat	BOOL	FALSE	前タスク周期の StcAPushStat の値
StcAInVal	UINT	0	StackPush で追加する値
StcAPush_OK	BOOL	FALSE	StackPush の正常終了フラグ
StcAPushNormalEnd	BOOL	FALSE	StackPush の正常終了時処理
StcAPushErrorEnd	BOOL	FALSE	StackPush の異常終了時処理
StcALIFOStat	BOOL	FALSE	StackLIFO の実行条件
preStcALIFOStat	BOOL	FALSE	前タスク周期の StcALIFOStat の値
StcAOutVal	UINT	0	StackLIFO で取り出す値
StcALIFO_OK	BOOL	FALSE	StackLIFO の正常終了フラグ
StcALIFONormalEnd	BOOL	FALSE	StackLIFO の正常終了時処理
StcALIFOErrorEnd	BOOL	FALSE	StackLIFO の異常終了時処理
StcAInsStat	BOOL	FALSE	StackIns の実行条件
preStcAInsStat	BOOL	FALSE	前タスク周期の StcAInsStat の値
StcAInsVal	UINT	0	StackIns で挿入する値
StcAOffset	UINT	0	StackIns のオフセット値
StcAIns_OK	BOOL	FALSE	StackIns の正常終了フラグ
StcAInsNormalEnd	BOOL	FALSE	StackIns の正常終了時処理
StcAInsErrorEnd	BOOL	FALSE	StackIns の異常終了時処理

```
// スタックの初期化
IF ( (InitStc=TRUE) AND (preInitStc=FALSE) ) THEN
  StcANum:=0;
  Clear(StcA);
```

```
  StcASize:=SizeOfAry(StcA);
END_IF;

// スタックに 3 個のデータを格納
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;

// StackPush 命令でデータ追加
IF ( (StcAPushStat=TRUE) AND (preStcAPushStat=FALSE) ) THEN
  StcAInVal:=UINT#4444;
  StackPush(
    In :=StcAInVal, // 追加する値
    InOut:=StcA[0], // スタック配列の先頭要素
    Size :=StcASize, // スタックの要素数
    Num :=StcANum, // スタックの格納要素数
    ENO =>StcAPush_OK); // 正常終了フラグ
  IF (StcAPush_OK=TRUE) THEN
    StcAPushNormalEnd:=TRUE;// 正常終了時処理
  ELSE
    StcAPushErrorEnd:=TRUE; // 異常終了時処理
  END_IF;
END_IF;

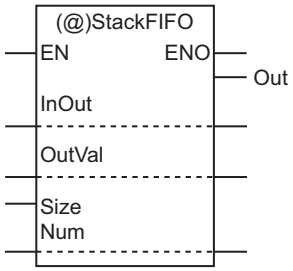
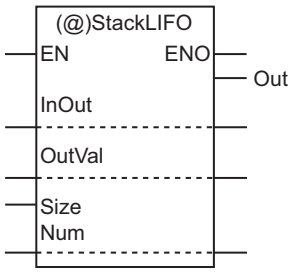
preStcAPushStat:=StcAPushStat;
// StackLIFO 命令でデータ取出
IF ( (StcALIFOStat=TRUE) AND (preStcALIFOStat=FALSE) ) THEN
  StackLIFO(
    InOut :=StcA[0], // スタック配列の先頭要素
    OutVal :=StcAOutVal, // スタックから取り出す値
    Size :=StcASize, // スタックの要素数
    Num :=StcANum, // スタックの格納要素数
    ENO =>StcALIFO_OK); // 正常終了フラグ
  IF (StcALIFO_OK=TRUE) THEN
    StcALIFONormalEnd:=TRUE; // 正常終了時処理
  ELSE
    StcALIFOErrorEnd:=TRUE; // 異常終了時処理
  END_IF;
END_IF;
preStcALIFOStat:=StcALIFOStat;

// StackIns 命令でデータ挿入
IF ( (StcAInsStat=TRUE) AND (preStcAInsStat=FALSE) ) THEN
  StcAInsVal:=UINT#5555;
  StcAOffset:=UINT#2;
  StackIns(
    In :=StcAInsVal, // スタックに挿入する値
```

```
    InOut :=StcA[0], // スタック配列の先頭要素
    Size :=StcASize, // スタックの要素数
    Num :=StcANum, // スタックの格納要素数
    Offset:=StcAOffset, // 挿入するオフセット位置
    ENO =>StcAIns_OK); // 正常終了フラグ
  IF (StcAIns_OK=TRUE) THEN
    StcAInsNormalEnd:=TRUE; // 正常終了時処理
  ELSE
    StcAInsErrorEnd:=TRUE; // 異常終了時処理
  END_IF;
END_IF;
preStcAInsStat:=StcAInsStat;
```

StackFIFO / StackLIFO

StackFIFO ： スタックの最下位の値を取り出します。
StackLIFO ： スタックの最上位の値を取り出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StackFIFO	先入れ先出し	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	後入れ先出し	FUN		StackLIFO(InOut, OutVal, Size, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	スタックの要素数	入力	スタックの配列要素数	データ型に従う	－	1
InOut[] 配列	スタック配列	入出力	スタックを構成する配列	データ型に従う	－	－
OutVal	出力値		スタックから出力する値、構造体全体			
Num	スタックの格納要素数		スタックに格納されている要素数			
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
InOut[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
OutVal	InOut[]の要素と同じデータ型																			
Num							○													
Out	○																			

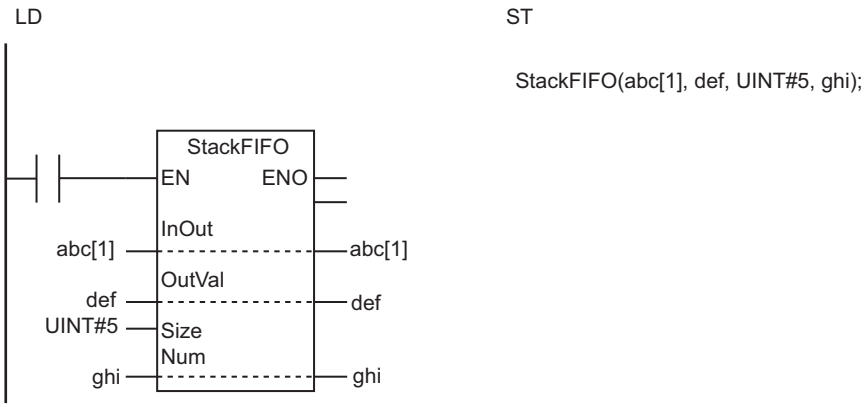
機能

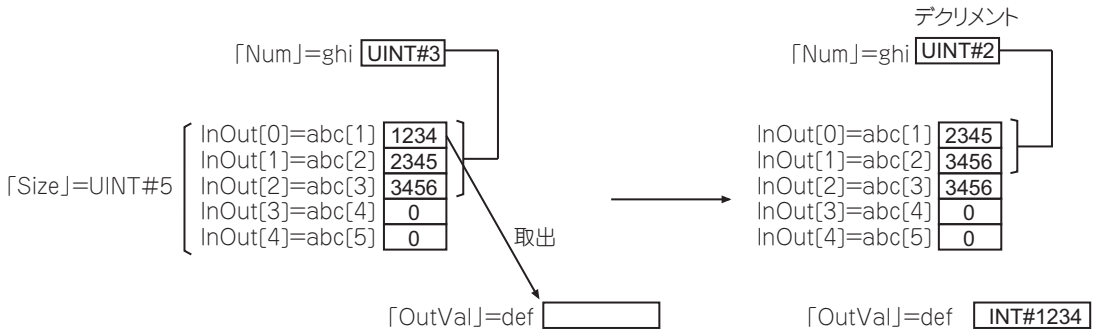
スタック配列 InOut[]に、現在、格納要素数「Num」個の要素が格納されていると見なします。そこから、値を取り出して、出力値「OutVal」に代入します。
スタックの要素数「Size」には、InOut[]のうち、スタックとして使用する要素数を指定します。

StackFIFO

スタックの最下位の値を取り出します。InOut[0]の値を、「OutVal」に代入します。
その後、InOut[1]以降の「Num」-1 個の要素を、それぞれスタック配列の下位方向に 1 つずつシフトします。
最後に、「Num」をデクリメントします。

「Size」=UINT#5、「Num」=UINT#3 の場合の例を、以下に示します。

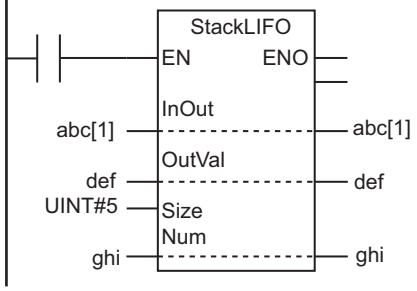




StackLIFO

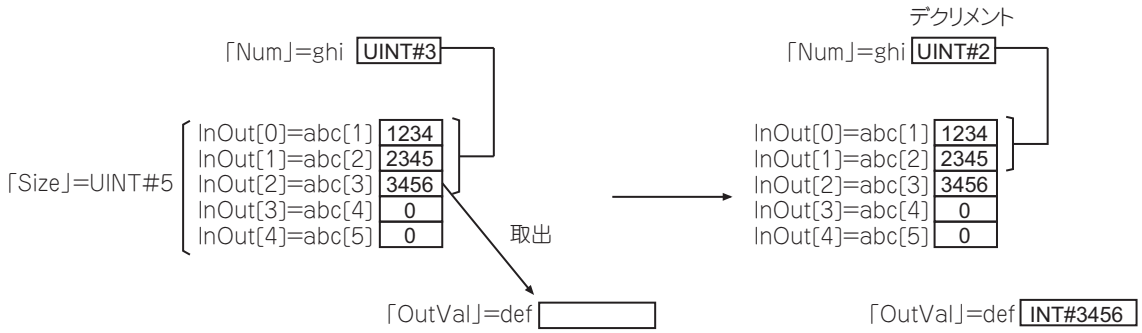
スタックの最上位の値を取り出します。InOut[「Num」-1]の値を、「OutVal」に代入します。
そして、「Num」をデクリメントします。
「Size」=UINT#5、「Num」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
StackLIFO(abc[1], def, UINT#5, ghi);
```



使用上の注意

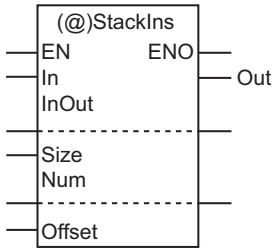
- InOut[]と「OutVal」のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- InOut[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Size」または「Num」が 0 の場合、InOut[]、「Num」、「OutVal」の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「OutVal」は変化しません。
  a)「Num」および「Size」の値が 0 以外で、「Num」＞「Size」の場合。
  b)「Size」の値が、InOut[]の配列領域を超える場合。
  c) InOut[]が STRING 型配列で、要素のいずれかが NULL 文字で終端されていない場合。
  d) InOut[]が STRING 型配列で、要素のバイト数が、「OutVal」のサイズを超える場合。

サンプルプログラム

StackPush 命令の「サンプルプログラム (P.2-501)」を、参照してください。

StackIns

スタックの任意の位置に、値を挿入します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StackIns	スタックデータ挿入	FUN		StackIns(In, InOut, Size, Num, Offset);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	挿入値	入力	スタックに挿入する値、構造体全体、構造体の 1 メンバ	データ型に従う	－	*1
Size	スタックの要素数		スタックの配列要素数			1
Offset	オフセット位置		「In」を挿入する、スタックのオフセット位置			0
InOut[] 配列	スタック配列	入出力	スタックを構成する配列	データ型に従う	－	－
Num	スタックの格納要素数		スタックに格納されている要素数			
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、構造体全体、構造体の 1 メンバも指定可能																			
Size							○													
Offset							○													
InOut[] 配列	「In」と同じデータ型を要素とする配列																			
Num							○													
Out	○																			

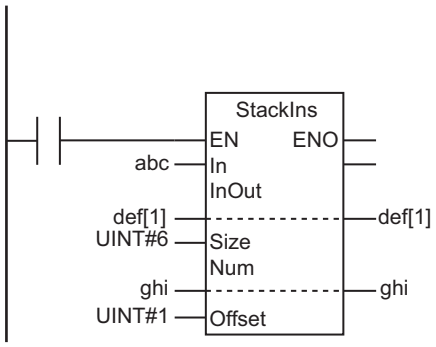
機能

スタック配列 InOut[]に、現在、格納要素数「Num」個の要素が格納されていると見なします。そして、オフセット位置「Offset」で決まる位置 InOut[「Offset」]に、挿入値「In」を挿入します。
そして、それよりも上位の要素、つまり InOut[「Offset」]から InOut[「Num」-1]は、それぞれスタック配列の上位方向に 1 つずつシフトします。

その後、「Num」をインクリメントします。
スタックの要素数「Size」には、InOut[]のうち、スタックとして使用する要素数を指定します。

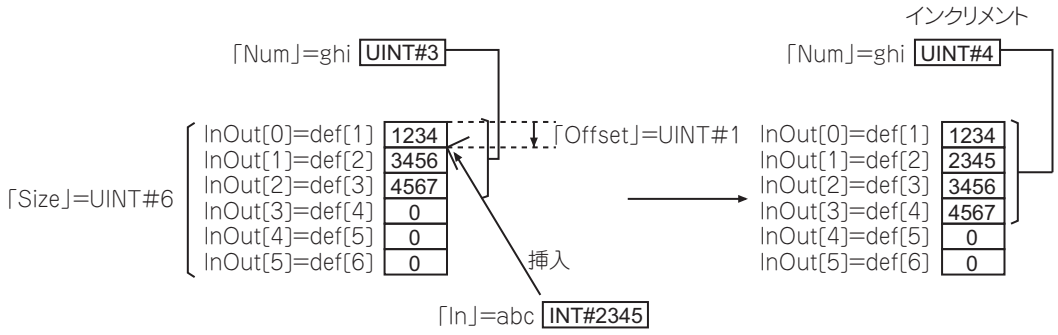
「Size」=UINT#6、「Num」=UINT#3、「Offset」=UINT#1 の場合の例を、以下に示します。

LD



ST

```
StackIns(abc, def[1], UINT#6, ghi, UINT#1);
```



使用上の注意

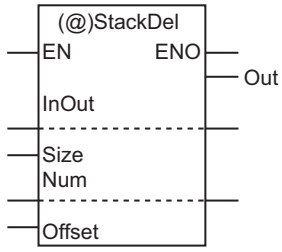
- 「In」と InOut[]のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- InOut[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Size」の値が 0 の場合、InOut[]、「Num」の値は変化しません。
- 「In」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「In」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Size」の値が 0 以外で、「Size」>「Num」≧「Offset」ではない場合。
  b)「Size」の値が、InOut[]の配列領域を超える場合。
  c)「In」、InOut[]が STRING 型で、「In」のバイト数が InOut[]のサイズを超える場合。

サンプルプログラム

StackPush 命令の「サンプルプログラム (P.2-501)」を、参照してください。

StackDel

スタックの任意の位置の、値を削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StackDel	スタックデータ削除	FUN		StackDel(InOut, Size, Num, Offset);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	スタックの要素数	入力	スタックの配列要素数	データ型に従う	−	1
Offset	オフセット位置		削除する、スタックのオフセット位置			0
InOut[] 配列	スタック配列	入出力	スタックを構成する配列	データ型に従う	−	−
Num	スタックの格納要素数		スタックに格納されている要素数			
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

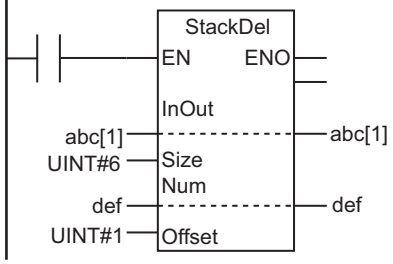
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Offset							○													
InOut[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Num							○													
Out	○																			

機能

スタック配列 InOut[]に、現在、格納要素数「Num」個の要素が格納されていると見なします。そして、オフセット位置「Offset」で決まる位置 InOut[「Offset」]の値を削除します。
そして、それよりも上位の要素、つまり InOut[「Offset」+1]から InOut[「Num」-1]は、それぞれスタック配列の下位方向に 1 つずつシフトします。
その後、「Num」をデクリメントします。
スタックの要素数「Size」には、InOut[]のうち、スタックとして使用する要素数を指定します。

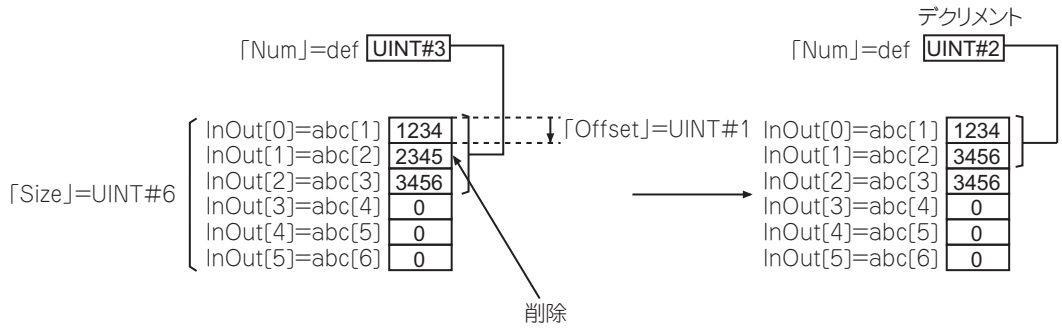
「Size」=UINT#6、「Num」=UINT#3、「Offset」=UINT#1 の場合の例を、以下に示します。

LD



ST

```
StackDel(abc[1], UINT#6, def, UINT#1);
```

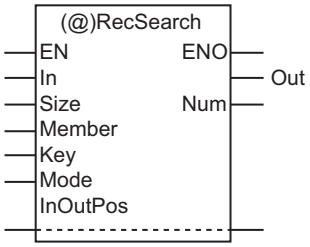


使用上の注意

- InOut[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Size」または「Num」の値が 0 の場合、InOut[]、「Num」の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、InOut[]は変化しません。
  a)「Num」および「Size」の値が 0 以外で、「Size」≧「Num」＞「Offset」ではない場合。
  b)「Size」の値が、InOut[]の配列領域を超える場合。

RecSearch

構造体を要素とする配列から、検索キーと一致する要素を、指定された方法で検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RecSearch	レコード検索	FUN		Out:=RecSearch(In, Size, Member, Key, Mode, InOutPos, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	検索対象配列	入力	検索対象となる、構造体を要素とする配列	－	—	*1
Size	検索対象の要素数		検索対象の配列要素数	データ型に従う		1
Member	検索対象メンバ		In[]の構造体の、検索対象メンバ			*1
Key	検索キー		検索値			
Mode	検索方法		検索方法	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[] 配列	一致要素の要素番号	入出力	一致要素の要素番号	データ型に従う	－	－
Out	検索結果	出力	TRUE: 一致した要素あり FALSE: 一致した要素なし	データ型に従う	—	－
Num	一致要素の個数		一致要素の個数			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	構造体を要素とする配列を指定																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	In[]の検索対象メンバと同じデータ型																			
Key	「Member」と同じデータ型																			
Mode	列挙型_eSEARCH_MODE 列挙子は「機能 (P.2-517)」参照																			
InOutPos[] 配列							○													
Out	○																			
Num							○													

機能

構造体を要素とする配列 In[]の「Size」個の要素、つまり In[0]から In[「Size」-1]の中から、構造体の検索対象メンバ「Member」の値が検索キー「Key」と一致するものを検索します。
「Member」には、In[]のいずれかの要素の検索対象メンバを、引数として渡します。
一致した要素があれば、検索結果「Out」の値が TRUE になります。そして、一致要素の要素番号が InOutPos[0]に、一致要素の数が「Num」にそれぞれ代入されます。一致要素が 2 個以上ある場合は、In[]の中で、最も下位にある一致要素の要素番号が、InOutPos[0]に代入されます。
一致した要素がなければ、「Out」の値は FALSE になり、InOutPos[0]と「Num」は 0 になります。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

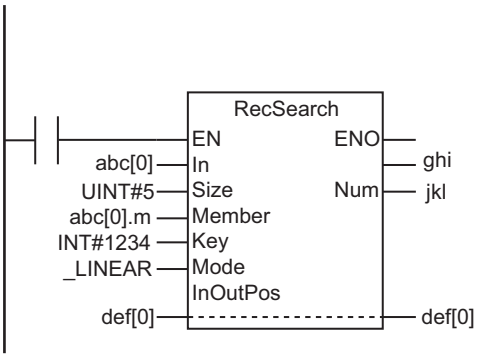
検索方法「Mode」のデータ型は、列挙型_eSEARCH_MODE です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LINEAR	線形探索
_BIN_ASC	昇順二分探索
_BIN_DESC	降順二分探索

線形探索の場合、In[]の先頭要素から順に検索を行います。

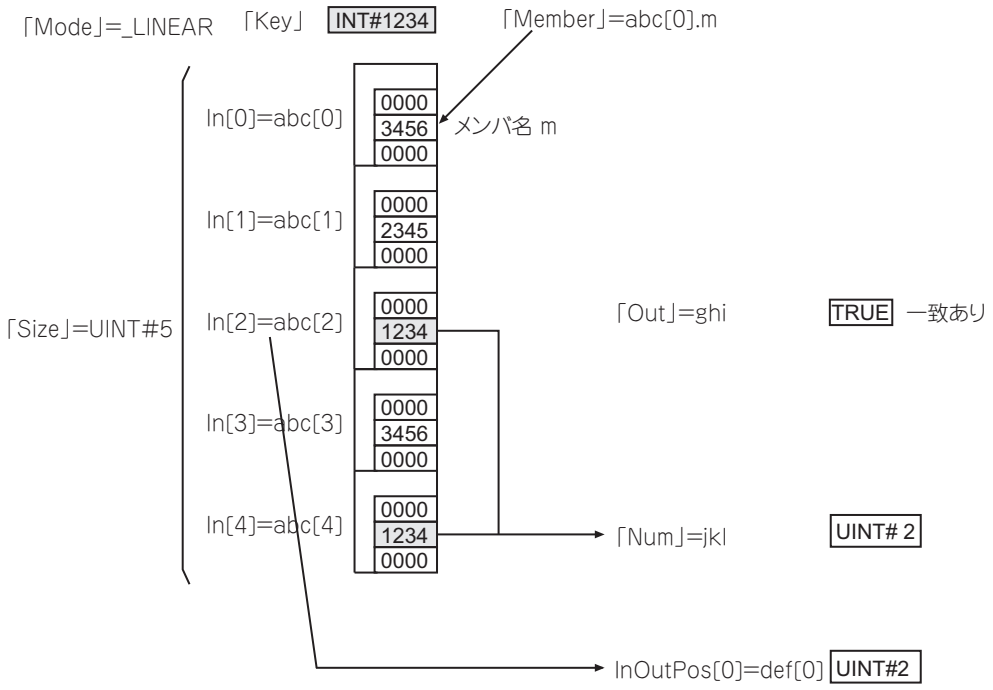
「Size」=UINT#5、「Key」=INT#1234、「Mode」=_LINEAR の例を、以下に示します。

LD



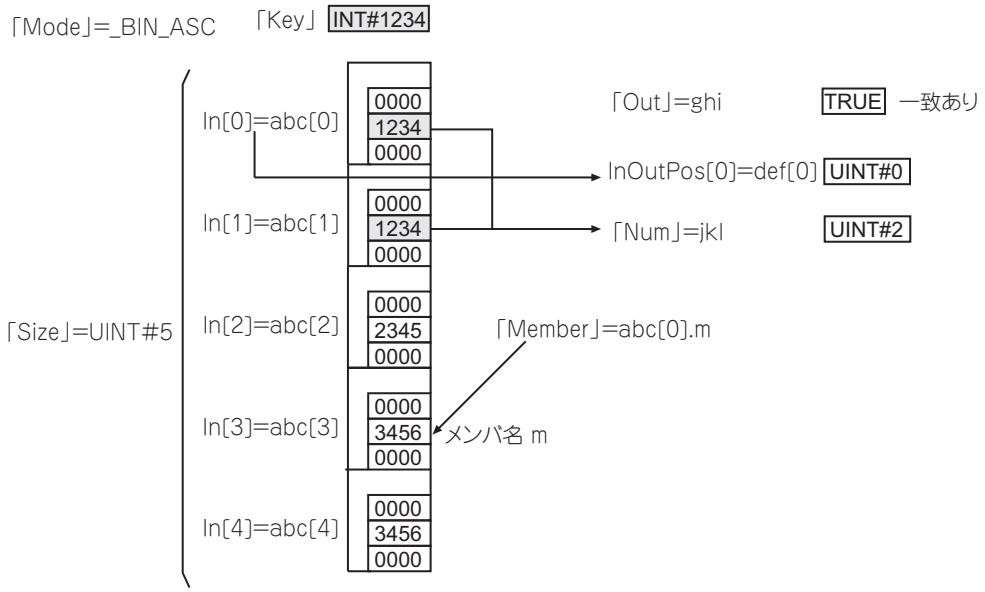
ST

```
ghi:=RecSearch(abc[0], UINT#5, abc[0].m,
                    INT#1234, _LINEAR, def[0], jkl);
```

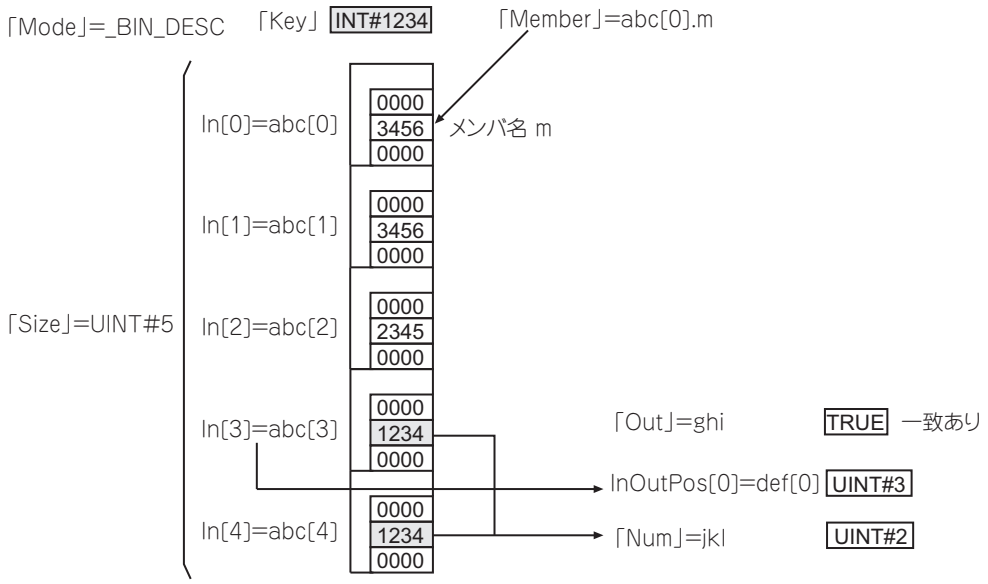


昇順二分探索の場合、本命令を実行する前に、In[]に渡す入力パラメータの配列要素を、昇順に並べておく必要があります。そして、本命令実行により、二分探索が行われます。

先と同じ例を、昇順二分探索した場合、配列要素の順序と処理結果は、以下のようになります。



降順二分探索の場合、本命令を実行する前に、In[]に渡す入力パラメータの配列要素を、降順に並べておく必要があります。そして、本命令実行により、二分探索が行われます。
先と同じ例を、降順順二分探索した場合、配列要素の順序と処理結果は、以下のようになります。



参考

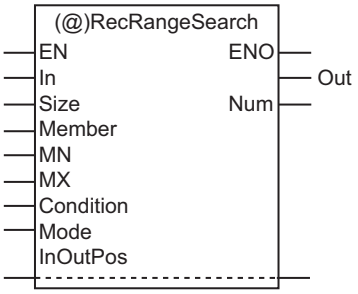
- In[]全体が、さらに上位の構造体のメンバであってもかまいません。
  （例）In[0]=str0.str1[0]
- In[]は、2 次元以上の配列でもかまいません。その場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、それぞれ代入されます。
- In[]が 3 次元の配列の場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、第 3 次元の要素番号が InOutPos[2]に、それぞれ代入されます。
- TIME 型、DT 型、TOD 型の検索を行う場合、「Member」と「Key」の精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- In[]は、構造体を要素とする配列にしてください。そうでないと、ビルド時に異常になります。
- 「Key」と「Member」のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- In[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Member」が実数の場合、値によって誤差が生ずるため、期待どおりの結果が得られないことがあります。
- 「Key」が実数の場合、「Key」に非数を指定しないでください。
- 「Size」の値が 0 の場合、「Out」の値は FALSE、「Num」の値は 0 になります。InOutPos[]は変化しません。
- 「Mode」の値が_BIN_ASC（または_BIN_DESC）の場合、In[]の要素が昇順（または降順）に並んでいないと、正しい結果が得られません。本命令実行前に、要素を昇順（または降順）に並べてください。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、InOutPos[]、「Num」は変化しません。
  a）「Mode」の値が、有効範囲外の場合。
  b）「Size」の値が、In[]の配列領域を超える場合。
  c）「Member」が、In[]のメンバでない場合。
  d）InOutPos[]の配列サイズが、In[]の次元数より小さい場合。
  e）「Member」が STRING 型で、かつ NULL 文字で終端されていない場合。

RecRangeSearch

構造体を要素とする配列から、検索条件の範囲に一致する要素を、指定された方法で検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RecRangeSearch	範囲指定レコード検索	FUN		Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	検索対象配列	入力	検索対象となる、構造体を要素とする配列	—	—	*1
Size	検索対象の要素数		検索対象の配列要素数	データ型に従う		1
Member	検索対象メンバ		In[]の構造体の検索対象メンバ			*1
MN	検索条件下限値		検索条件下限値			
MX	検索条件上限値		検索条件上限値			
Condition	検索条件		検索条件	_EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH		_EQ_BOTH
Mode	検索方法		検索方法	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[] 配列	一致要素の要素番号	入出力	一致要素の要素番号	データ型に従う	–	–
Out	検索結果	出力	TRUE: 一致した要素あり FALSE: 一致した要素なし	データ型に従う	—	–
Num	一致要素の個数		一致要素の個数			

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	構造体を要素とする配列を指定																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	In[]の検索対象メンバと同じデータ型																			
MN	「Member」と同じデータ型																			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MX	「Member」と同じデータ型																			
Condition	列挙型_eSEARCH_CONDITION 列挙子は「機能 (P.2-521)」参照																			
Mode	列挙型_eSEARCH_MODE 列挙子は「機能 (P.2-521)」参照																			
InOutPos[] 配列							○													
Out	○																			
Num							○													

機能

構造体を要素とする配列 In[]の「Size」個の要素、つまり In[0]から In[「Size」-1]の中から、構造体の検索対象メンバ「Member」の値が、検索条件と一致するものを検索します。
検索条件と検索方法は、「Condition」と「Mode」で指定します。詳細は、後述します。
「Member」には、In[]のいずれかの要素の検索対象メンバを、引数として渡します。
条件に一致した要素があれば、検索結果「Out」の値が TRUE になります。そして、一致要素の要素番号が InOutPos[0]に、一致要素の数が「Num」にそれぞれ代入されます。一致要素が 2 個以上ある場合は、In[]の中で、最も下位にある一致要素の要素番号が、InOutPos[0]に代入されます。
一致した要素がなければ、「Out」の値は FALSE になり、InOutPos[0]と「Num」は 0 になります。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

検索条件「Condition」のデータ型は、列挙型_eSEARCH_CONDITION です。列挙子の意味は、以下のとおりです。

列挙子	意味
_EQ_BOTH	「MN」≦「Member」≦「MX」
_EQ_MIN	「MN」≦「Member」＜「MX」
_EQ_MAX	「MN」＜「Member」≦「MX」
_NE_BOTH	「MN」＜「Member」＜「MX」

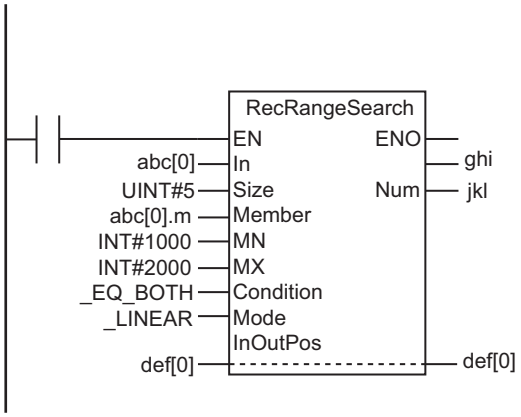
検索方法「Mode」のデータ型は、列挙型_eSEARCH_MODE です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LINEAR	線形探索
_BIN_ASC	昇順二分探索
_BIN_DESC	降順二分探索

線形探索の場合、In[]の先頭要素から順に検索を行います。

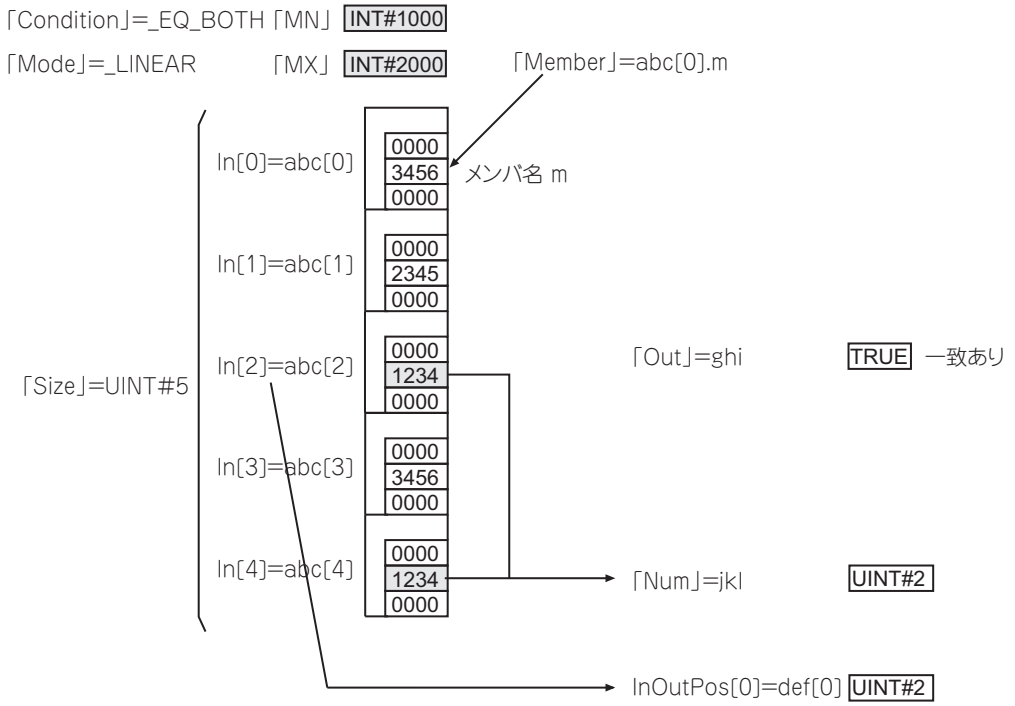
「Size」=UINT#5、「MN」=INT#1000、「MX」=INT#2000、「Condition」=_EQ_BOTH、「Mode」=_LINEAR の例を、以下に示します。

LD



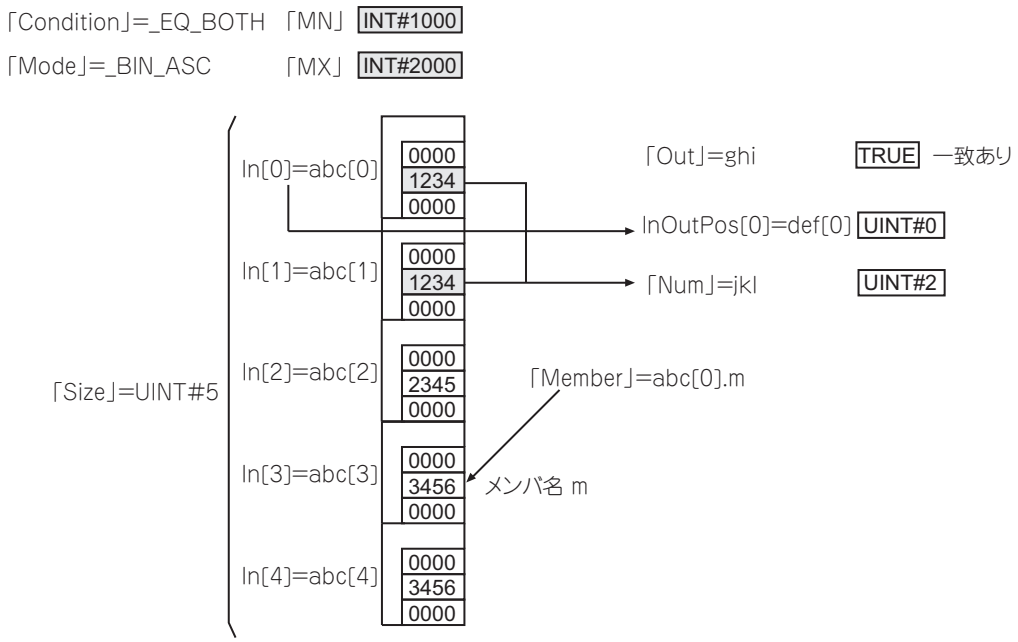
ST

```
ghi:=RecRangeSearch(abc[0], UINT#5, abc[0].m, INT#1000,
                    INT#2000, _EQ_BOTH, _LINEAR,
                    def[0], jkl);
```



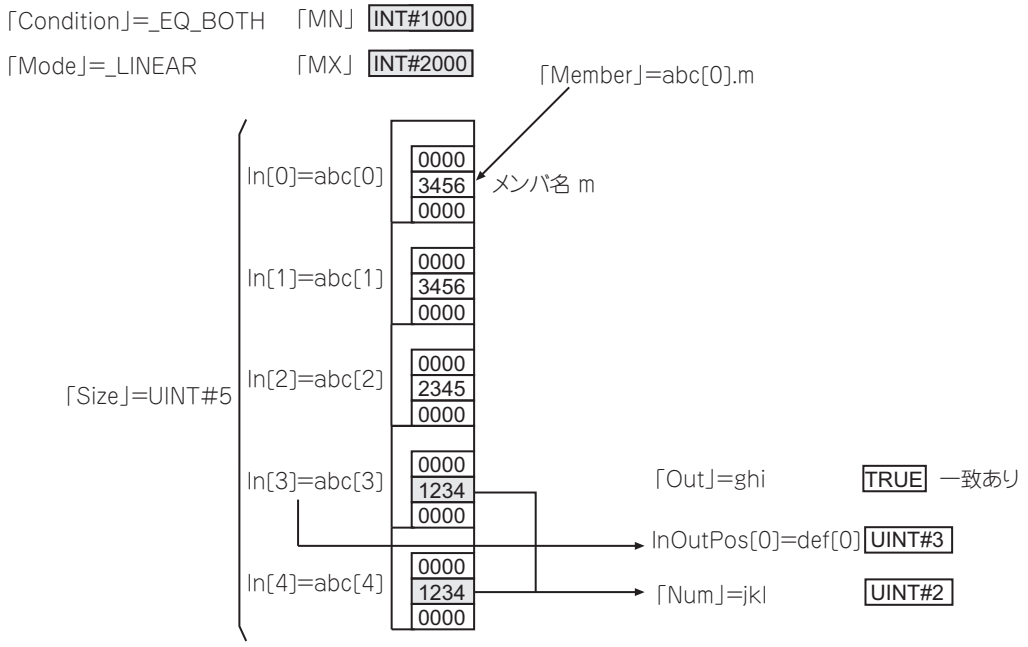
昇順二分探索の場合、本命令を実行する前に、In[]に渡す入力パラメータの配列要素を、昇順に並べておく必要があります。そして、本命令実行により、二分探索が行われます。

先と同じ例を、昇順二分探索した場合、配列要素の順序と処理結果は、以下のようになります。



降順二分探索の場合、本命令を実行する前に、In[]に渡す入力パラメータの配列要素を、降順に並べておく必要があります。そして、本命令実行により、二分探索が行われます。

先と同じ例を、降順二分探索した場合、配列要素の順序と処理結果は、以下のようになります。



参考

- In[]全体が、さらに上位の構造体のメンバであってもかまいません。
  （例）In[0]=str0.str1[0]
- In[]は、2 次元以上の配列でもかまいません。その場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、それぞれ代入されます。
- In[]が 3 次元の配列の場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、第 3 次元の要素番号が InOutPos[2]に、それぞれ代入されます。

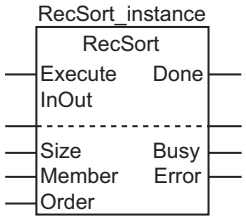
・TIME 型、DT 型、TOD 型の検索を行う場合、「Member」、「MN」、「MX」の精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

・「Member」、「MN」、「MX」のデータ型は、In[]の検索対象メンバのデータ型と同じにしてください。異なると、ビルド時に異常になります。
・In[]は、構造体を要素とする配列にしてください。そうでないと、ビルド時に異常になります。
・In[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
・「Member」が実数の場合、値によって誤差が生ずるため、期待どおりの結果が得られないことがあります。
・「MN」、「MX」が実数の場合、非数を指定しないでください。
・「Size」の値が 0 の場合、「Out」の値は FALSE、「Num」の値は 0 になります。InOutPos[]は変化しません。
・「Mode」の値が_BIN_ASC（または_BIN_DESC）の場合、In[]の要素が昇順（または降順）に並んでいないと、正しい結果が得られません。本命令実行前に、要素を昇順（または降順）に並べてください。
・つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、InOutPos[]、「Num」は変化しません。
  a)「MN」＞「MX」の場合。
  b)「Condition」の値が、有効範囲外の場合。
  c)「Mode」の値が、有効範囲外の場合。
  d)「Size」の値が、In[]の配列領域を超える場合。
  e)「Member」が、In[]のメンバでない場合。
  f) InOutPos[]の配列サイズが、In[]の次元数より小さい場合。

RecSort

構造体を要素とする配列の要素を、ソートします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RecSort	レコードソート	FB		RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Size	ソート対象の要素数	入力	ソート対象の配列要素数	データ型に従う	—	1
Member	ソート対象メンバ		In[]の構造体のソート対象メンバ			*1
Order	ソート順		ソート順	_ASC, _DESC		_ASC
InOut[] 配列	ソート対象配列	入出力	ソート対象となる、構造体を要素とする配列	—	—	—

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	InOut[]の検索対象メンバと同じデータ型																			
Order	列挙型_eSORT_ORDER 列挙子は「機能 (P.2-525)」参照																			
InOut[] 配列	構造体を要素とする配列を指定																			

機能

「Execute」の値が TRUE のとき、構造体を要素とする配列 InOut[]の「Size」個の要素、つまり InOut[0]から InOut[「Size」-1]を、構造体のソート対象メンバ「Member」の値でソートします。ソート順は、「Order」で指定します。
「Member」には、In[]のいずれかの要素のソート対象メンバを、引数として渡します。

InOut[]に渡す入出力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

ソート順「Order」のデータ型は、列挙型_eSORT_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_ASC	昇順

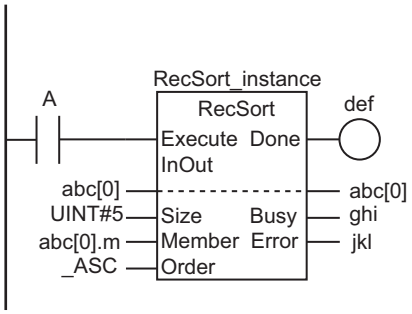
列挙子	意味
_DESC	降順

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

データ型	大小関係
TIME	値が大きい方が、大きいと判断します。
DATE、TOD、DT	日付や時刻が後の方が大きいと判断します。
STRING	「LTascii / LEascii / GTascii / GEascii (P.2-107)」命令と同じ仕様です。そちらを参照してください。

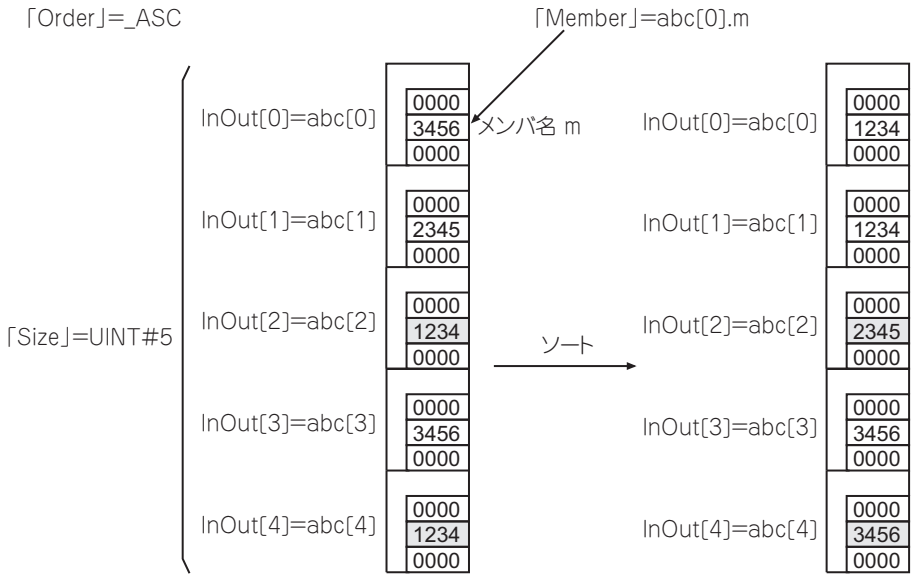
「Size」=UINT#5、「Order」=_ASC の場合の例を、以下に示します。

LD



ST

```
RecSort_instance(A, abc[0], UINT#5, abc[0].m, _ASC, def, ghi, jkl);
```



参考

- 本命令を実行中に電源断が発生した場合、InOut[]の内容が破壊される可能性があります。そこで、本命令を実行完了するごとに、InOut[]の内容をバックアップしておくと、破壊が起こっても修復することができます。
  「サンプルプログラム (P.2-527)」を参照してください。
- TIME 型、DT 型、TOD 型のソートを行う場合、「Member」どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- InOut[]は、構造体を要素とする配列にしてください。そうでないと、ビルド時に異常になります。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「Member」が実数の場合、値によって誤差が生ずるため、期待どおりの結果が得られないことがあります。
- InOut[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「Size」の値が 0 の場合、「Done」の値が TRUE になり、InOut[]は変化しません。
- つぎの場合、異常となります。「Error」が TRUE となります。
  a)「Order」の値が、有効範囲外の場合。
  b)「Size」の値が、InOut[]の配列領域を超える場合。
  c)「Member」が、InOut[]のメンバでない場合。
  d)「Member」が STRING 型で、かつ NULL 文字で終端されていない場合。

サンプルプログラム

構造体「MyStr」を要素とする配列 Abc[]を、昇順にソートします。ソート対象メンバは、Abc[].m です。
処理中に電断が発生してもデータが失われないように、ソート実行前に Abc[]を Abc_backup[]という変数名でバックアップしておきます。もし電断が発生したら、Abc_backup[]を Abc[]に戻して、ソートをやり直します。

グローバル変数の定義

● データ型

名称	データ型	コメント
MyStr	STRUCT	構造体
l	BOOL	メンバ
m	INT	メンバ
n	REAL	メンバ

● グローバル変数

名称	データ型	初期値	保持	コメント
Abc	ARRAY[0..4] OF MyStr	[5((l:=FALSE,m:=0,n:=0.0))]	☑	ソート対象配列
Abc_backup	ARRAY[0..4] OF MyStr	[5((l:=FALSE,m:=0,n:=0.0))]	☑	Abc[]のバックアップ

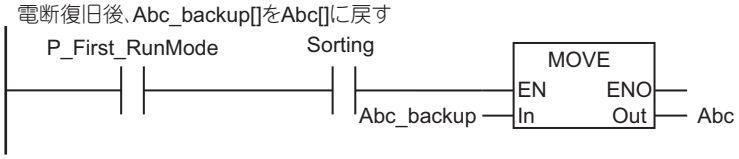
LD

内部変数	名称	データ型	初期値	保持	コメント
	Sorting	BOOL	FALSE	☑	処理中（保持あり）

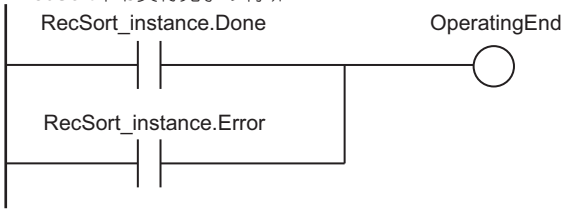
内部変数	名称	データ型	初期値	保持	コメント
	OperatingEnd	BOOL	FALSE	☐	処理終了
	Trigger	BOOL	FALSE	☐	実行条件
	Operating	BOOL	FALSE	☐	処理中
	RS_instance	RS		☐	
	RecSort_instance	RecSort		☐	

外部変数	名称	データ型	コメント
	Abc	ARRAY[0..4] OF MyStr	ソート対象配列
	Abc_backup	ARRAY[0..4] OF MyStr	Abc[]のバックアップ

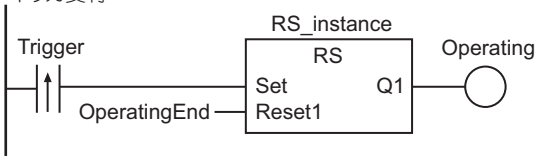
電断復旧後、Abc_backup[]をAbc[]に戻す



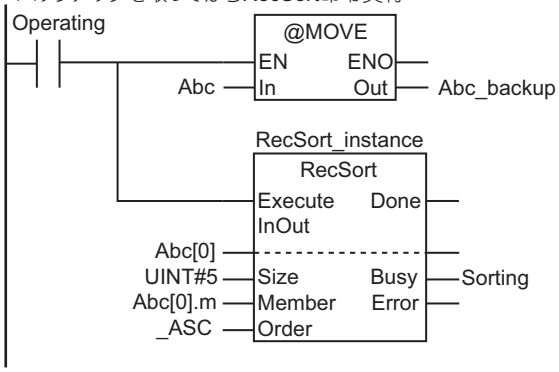
RecSort命令実行完了の判断



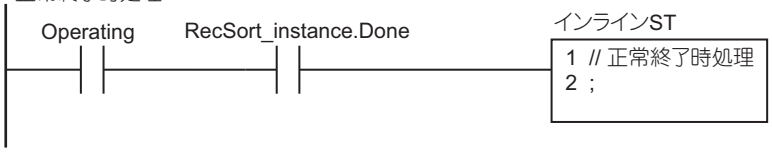
トリガ受付

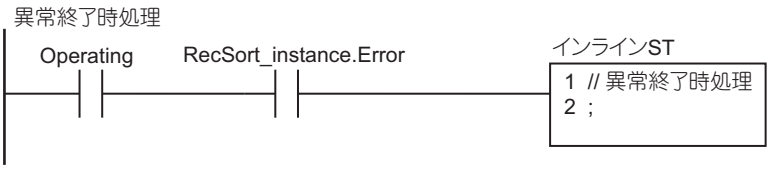


バックアップを取ってからRecSort命令実行



正常終了時処理





ST

内部変数	名称	データ型	初期値	保持	コメント
	Sorting	BOOL	FALSE	☑	処理中（保持あり）
	Trigger	BOOL	FALSE	☐	実行条件
	LastTrigger	BOOL	FALSE	☐	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	FALSE	☐	処理開始
	Operating	BOOL	FALSE	☐	処理中
	RS_instance	RS		☐	
	RecSort_instance	RecSort		☐	

外部変数	名称	データ型	コメント
	Abc	ARRAY[0..4] OF MyStr	ソート対象配列
	Abc_backup	ARRAY[0..4] OF MyStr	Abc[]のバックアップ

```
// 電断復旧後、Abc_backup[]を Abc[]に戻す
IF ( (P_First_RunMode = TRUE) AND (Sorting = TRUE) ) THEN
  Abc:=Abc_backup;
END_IF;

// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
  OperatingStart:=TRUE;
  Operating :=TRUE;
END_IF;
LastTrigger:=Trigger;

// RecSort 命令初期化
IF (OperatingStart=TRUE) THEN
  Abc_backup:=Abc;
  RecSort_instance(
    Execute:=FALSE, // 起動条件
    InOut :=Abc[0], // ソート対象配列
    Member :=Abc[0].m); // ソート対象メンバ
  OperatingStart:=FALSE;
END_IF;

// RecSort 命令実行
```

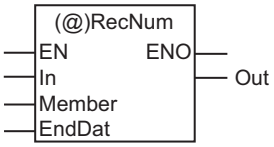
```
IF (Operating=TRUE) THEN
  RecSort_instance (
    Execute:=TRUE,
    InOut :=Abc[0],
    Size :=UINT#5,
    Member :=Abc[0].m,
    Order :=_ASC,
    Busy =>Sorting) ;

  IF (RecSort_instance.Done=TRUE) THEN
    // 正常終了時処理
    Operating:=FALSE;
  END_IF;

  IF (RecSort_instance.Error=TRUE) THEN
    // 異常終了時処理
    Operating:=FALSE;
  END_IF;
END_IF;
```

RecNum

構造体を要素とする配列の、終端データまでのレコード数を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RecNum	レコード数取得	FUN		Out:=RecNum(In, Member, EndDat);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	対象配列	入力	処理対象となる、構造体を要素とする配列	—	—	*1
Member	対象メンバ		In[]の構造体の処理対象メンバ	データ型に従う		
EndDat	終端データ		終端データ			
Out	レコード数	出力	レコード数	データ型に従う	—	—

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	構造体を要素とする配列を指定																			
Member	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型も指定可能																			
	In[]の処理対象メンバと同じデータ型																			
EndDat	「Member」と同じデータ型																			
Out							○													

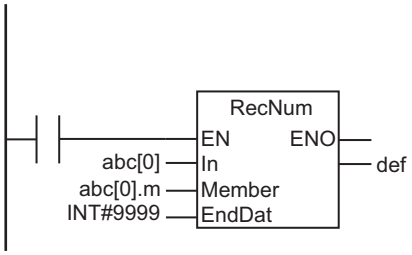
機能

構造体を要素とする配列 In[]の先頭から、対象メンバ「Member」の値が、終端データ「EndDat」と一致する要素を検索します。その結果、「EndDat」と一致した要素の直前までの要素数（レコード数）を「Out」に代入します。
「Member」には、In[]のいずれかの要素の対象メンバを、引数として渡します。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

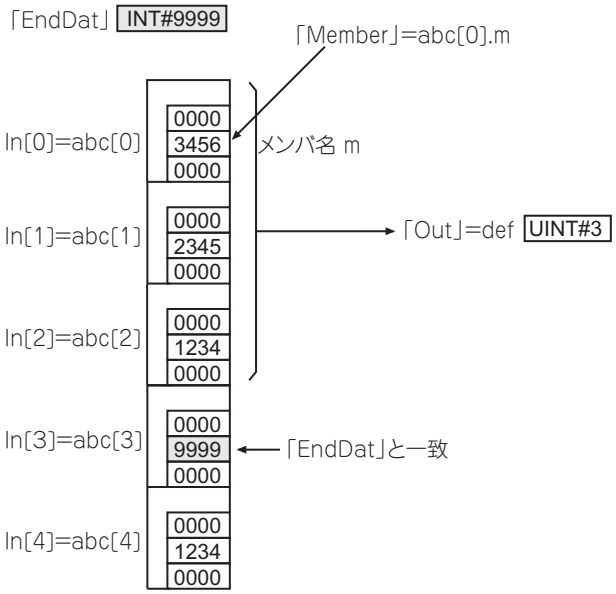
「EndDat」=INT#9999 の場合の例を、以下に示します。

LD



ST

```
def:=RecNum(abc[0], abc[0].m, INT#9999);
```



参考

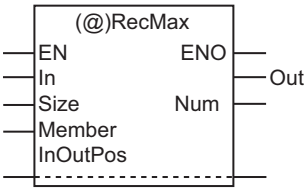
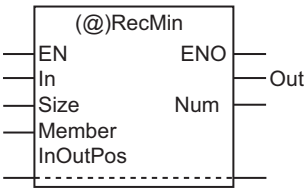
- In[]全体が、さらに上位の構造体のメンバであってもかまいません。
  （例）In[0]=str0.str1[0]
- TIME 型、DT 型、TOD 型の検索を行う場合、「Member」と「EndDat」の精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- In[]は、構造体を要素とする配列にしてください。そうでないと、ビルド時に異常になります。
- 「Member」と「EndDat」のデータ型は、同じにしてください。異なると、ビルド時に異常になります。
- In[]のメンバの中に、「EndDat」と一致するものがない場合は、In[]の全要素数が「Out」に代入されます。
- 「Member」が実数の場合、値によって誤差が生ずるため、期待どおりの結果が得られないことがあります。
- 「EndDat」が実数の場合、「EndDat」に非数を指定しないでください。
- In[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「Member」が、In[]のメンバでない場合。
  b）「Member」が STRING 型で、かつ NULL 文字で終端されていない場合。

RecMax / RecMin

RecMax ： 構造体を要素とする配列から、指定されたメンバの最大値を検索します。
RecMin ： 構造体を要素とする配列から、指定されたメンバの最小値を検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RecMax	レコード最大値検索	FUN		Out:=RecMax(In, Size,Member, InOutPos, Num);
RecMin	レコード最小値検索	FUN		Out:=RecMin(In, Size, Member, InOutPos, Num);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	検索対象配列	入力	検索対象となる、構造体を要素とする配列	—	—	*1
Size	検索対象の要素数		検索対象の配列要素数	データ型に従う		1
Member	検索対象メンバ		In[]の構造体の検索対象メンバ			*1
InOutPos[] 配列	検索結果の要素番号	入出力	検索結果の要素番号	データ型に従う	—	—
Out	検索結果	出力	検索結果	データ型に従う	—	—
Num	検索個数		検索個数			—

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列	構造体を要素とする配列を指定																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	In[]の検索対象メンバと同じデータ型																			
InOutPos[] 配列							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Num							○													

機能

構造体を要素とする配列 In[]の「Size」個の要素、つまり In[0]から In[「Size」-1]の中から、構造体の検索対象メンバ「Member」の値の検索を行います。
「Member」には、In[]のいずれかの要素の検索対象メンバを、引数として渡します。
検索結果の要素番号が InOutPos[0]に、検索要素数が「Num」にそれぞれ代入されます。検索結果が 2 個以上ある場合は、In[]の中で、最も下位にある検索結果の要素番号が、InOutPos[0]に代入されます。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

整数、実数以外のデータ型の値の大小関係は、以下のように判断します。

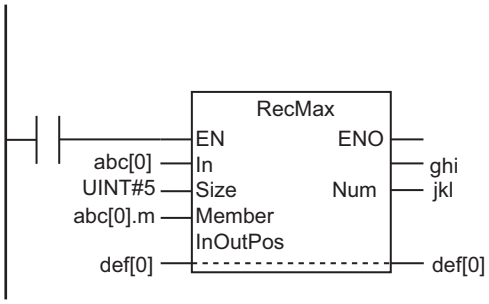
データ型	大小関係
TIME	値が大きい方が、大きいと判断します。
DATE、TOD、DT	日付や時刻が後の方が大きいと判断します。
STRING	「LTascii / LEascii / GTascii / GEascii (P.2-107)」命令と同じ仕様です。そちらを参照してください。

RecMax

最大値検索を行います。検索結果「Out」には、検索対象メンバの最大値が代入されます。

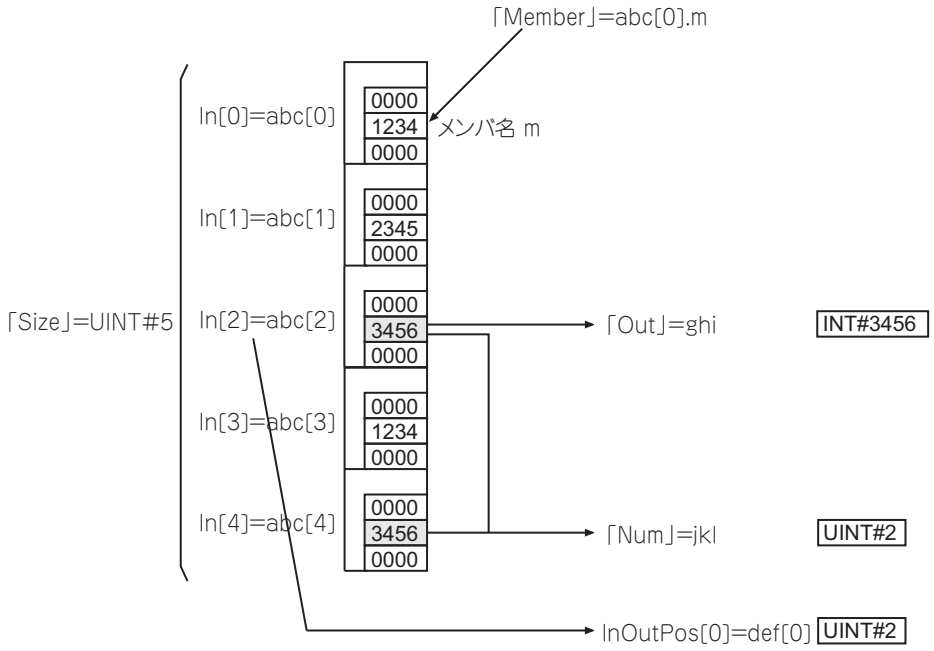
RecMax 命令で、「Size」=UINT#5 の場合の例を、以下に示します。

LD



ST

```
ghi:=RecMax(abc[0], UINT#5, abc[0].m, def[0], jkl);
```



RecMin

最小値検索を行います。検索結果「Out」には、検索対象メンバの最小値が代入されます。

参考

- In[]全体が、さらに上位の構造体のメンバであってもかまいません。
  （例）In[0]=str0.str1[0]
- In[]は、2 次元以上の配列でもかまいません。その場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、それぞれ代入されます。
- In[]が 3 次元の配列の場合、検索条件に一致した要素の第 1 次元の要素番号が InOutPos[0]に、第 2 次元の要素番号が InOutPos[1]に、第 3 次元の要素番号が InOutPos[2]に、それぞれ代入されます。
- TIME 型、DT 型、TOD 型の検索を行う場合、「Member」どうしの精度を合わせてください。値の精度を合わせるために、「TruncTime (P.2-658)」命令、「TruncDt (P.2-662)」命令、「TruncTod (P.2-666)」命令があります。

使用上の注意

- 「Member」と「Out」のデータ型が異なる場合は、つぎのデータ型の中から、「Out」の有効範囲が「Member」の有効範囲を包含するように選択してください。
  a）USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL
- 「Member」が実数の場合、値によって誤差が生ずるため、期待どおりの結果が得られないことがあります。
- In[]に、配列の途中の要素を渡した場合、その要素以降が処理対象になります。
- 「In」が列挙型の場合、「In」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド異常になります。
- 「Size」の値が 0 の場合、「Out」、「Num」の値は 0 になります。ただし、「Member」が STRING 型で「Size」の値が 0 の場合は、「Out」は NULL 文字だけの文字列になります。InOutPos[]の値は変化しません。

- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」、InOutPos[]、「Num」は変化しません。
  a)「Size」の値が、In[]の配列領域を超える場合。
  b)「Member」が、In[]のメンバでない場合。
  c) InOutPos[]の配列サイズが、In[]の次元数より小さい場合。
  d)「Member」が STRING 型で、かつ NULL 文字で終端されていない場合。

FCS 命令

命令	名称	ページ
StringSum	サム値算出	P.2-538
StringLRC	LRC 値算出（文字列）	P.2-540
StringCRCCCITT	CRC-CCITT 値算出（文字列）	P.2-542
StringCRC16	CRC-16 値算出（文字列）	P.2-544
AryLRC_**	LRC 値算出（配列）グループ	P.2-546
AryCRCCCITT	CRC-CCITT 値算出（配列）	P.2-548
AryCRC16	CRC-16 値算出（配列）	P.2-550

StringSum

文字列のサム値を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringSum	サム値算出	FUN	(@)StringSum EN ENO In Out Size	Out:=StringSum(In, Size);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象文字列	入力	対象文字列	データ型に従う	―	''
Size	バイトサイズ		サム値のバイトサイズ	1, 2	バイト	1
Out	サム値	出力	サム値	「Size」で示されるバイト数	バイト	―

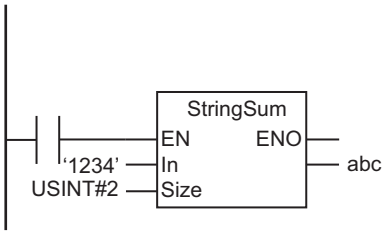
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Size						○														
Out																				○

機能

対象文字列「In」の、サム値（各文字の文字コードの総和）を算出します。サム値「Out」は、バイトサイズ「Size」で指定したバイト数になります。
「Out」は 16 進数の文字列で表現され、終端には NULL 文字が格納されます。

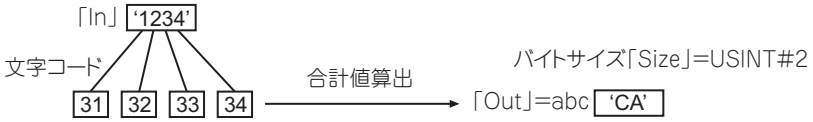
「In」='1234'、「Size」=USINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=StringSum('1234', USINT#2);
```



上の例で、「Size」=USINT#1 ならば、「Out」の値は'A'となります。

使用上の注意

- 「In」の文字コードの総和が、「Size」で表現できるけた数を超える場合は、上位けたを切り捨てます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、有効範囲外の場合。
  b)「In」のバイト数が 0（NULL 文字だけ）の場合。

StringLRC

文字列の LRC 値（水平パリティ）を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringLRC	LRC 値算出（文字列）	FUN	(@)StringLRC EN ENO In Out	Out:=StringLRC(In);

変数

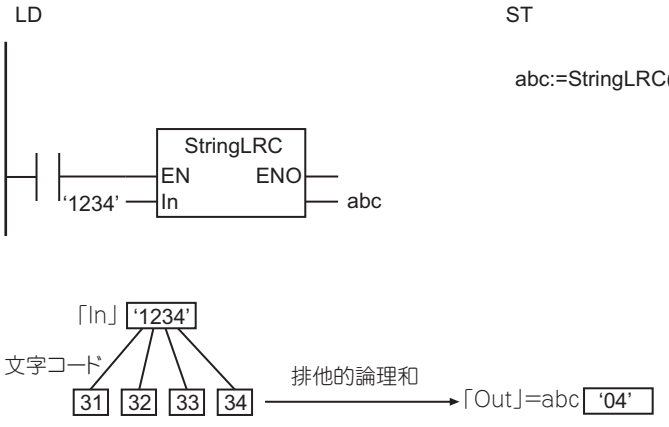
	名称	入／出力	内容	有効範囲	単位	初期値
In	対象文字列	入力	対象文字列	データ型に従う	—	"
Out	LRC 値	出力	LRC 値	最大 3 バイト（半角英数字 2 文字+終端 NULL 文字）	—	—

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

機能

対象文字列「In」の、LRC 値（水平パリティ）を算出します。LRC 値とは、「In」の各文字の文字コードの排他的論理和のことです。
LRC 値「Out」は 16 進数の文字列で表現され、終端には NULL 文字が格納されます。

「In」='1234'の場合の例を、以下に示します。

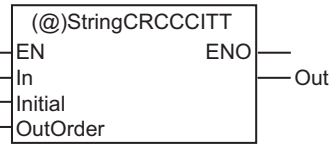


使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
・「In」のバイト数が 0（NULL 文字だけ）の場合。

StringCRCCCITT

文字列の CRC-CCITT 値を、XMODEM 方式で算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringCRCCCITT	CRC-CCITT 値算出（文字列）	FUN		Out:=StringCRCCCITT(In, Initial, OutOrder);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象文字列	入力	対象文字列	データ型に従う	—	"
Initial	初期値		CRC-CCITT 値算出の初期値			0
OutOrder	バイト順序		「In」を処理するバイト順序	_LOW_HIGH, _HIGH_LOW		_HIGH_LOW
Out	CRC-CCITT 値	出力	CRC-CCITT 値	5 バイト（半角英数字 4 文字＋終端 NULL 文字）	—	—

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Initial			○																	
OutOrder	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-542)」参照																			
Out																				○

機能

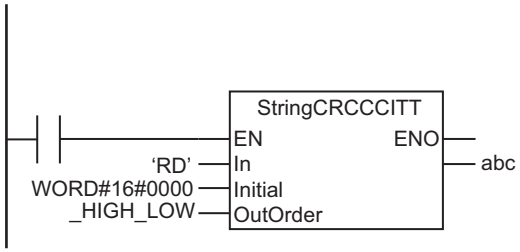
対象文字列「In」の、CRC-CCITT 値を、XMODEM 方式で算出します。
CRC-CCITT 値「Out」は 16 進数の文字列で表現され、終端には NULL 文字が格納されます。

CRC-CCITT 値を算出する際の初期値は、「Initial」で指定します。バイト順序は、「OutOrder」で指定します。
「OutOrder」のデータ型は、列挙型_eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

「In」='RD'、「Initial」=WORD#16#0000、「OutOrder」=_HIGH_LOW の場合の例を、以下に示します。

LD



ST

```
abc:=StringCRCCCITT('RD', WORD#16#0000,
                    _HIGH_LOW);
```



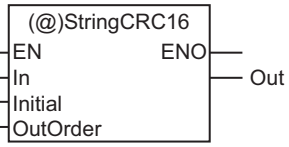
使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 「OutOrder」の値が、有効範囲外の場合。
- 「In」のバイト数が 0（NULL 文字だけ）の場合。

StringCRC16

文字列の CRC-16 値を、MODBUS 方式で算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
StringCRC16	CRC-16 値算出（文字列）	FUN		Out:=StringCRC16(In, Initial, OutOrder);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象文字列	入力	対象文字列	データ型に従う	－	''
Initial	初期値		CRC-16 値算出の初期値			16#FFFF
OutOrder	バイト順序		「In」を処理するバイト順序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
Out	CRC-16 値	出力	CRC-16 値	5 バイト（半角英数字 4 文字＋終端 NULL 文字）	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Initial			○																	
OutOrder	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-544)」参照																			
Out																				○

機能

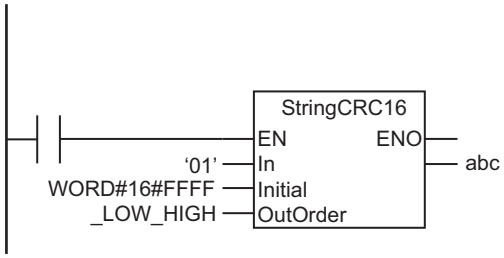
対象文字列「In」の、CRC-16 値を、MODBUS 方式で算出します。
CRC-16 値「Out」は 16 進数の文字列で表現され、終端には NULL 文字が格納されます。

CRC-16 値を算出する際の初期値は、「Initial」で指定します。バイト順序は、「OutOrder」で指定します。
「OutOrder」のデータ型は、列挙型_eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

「In」='01'、「Initial」=WORD#16#FFFF、「OutOrder」=_LOW_HIGH の場合の例を、以下に示します。

LD



ST

```
abc:=StringCRC16('01', WORD#16#FFFF,
                 _LOW_HIGH);
```



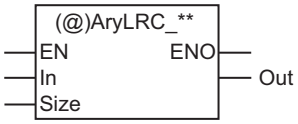
使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 「OutOrder」の値が、有効範囲外の場合。
- 「In」のバイト数が 0（NULL 文字だけ）の場合。

AryLRC_**

配列の LRC 値を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryLRC_**	LRC 値算出（配列）グループ	FUN	 ただし、“**”はビット列のデータ型名	Out:=AryLRC_**(In, Size); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	演算対象の要素数		In[]の要素数			1
Out	LRC 値	出力	LRC 値	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○	○	○	○															
Size							○													
Out	In[]と同じデータ型																			

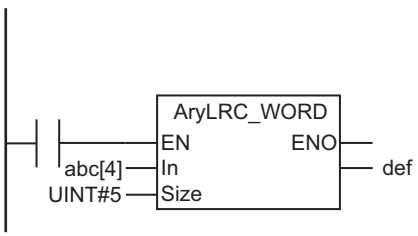
機能

演算対象配列 In[]の、In[0]以降の「Size」個の配列要素の LRC 値（排他的論理和）を算出します。命令名称は、In[]のデータ型によって、異なります。たとえば、In[]が WORD 型の場合、命令名称は AryLRC_WORD となります。

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

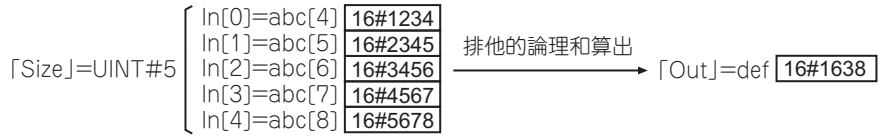
AryLRC_WORD 命令で、「Size」=UINT#5 の場合の例を、以下に示します。

LD



ST

```
def:=AryLRC_WORD(abc[4], UINT#5);
```



使用上の注意

- In[]と「Out」のデータ型は、同じにしてください。
- 「Size」の値が 0 の場合、「Out」の値は 16#00 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が、In[]の配列領域を超える場合。

AryCRCCCITT

配列の CRC-CCITT 値を、XMODEM 方式で算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCRCCCITT	CRC-CCITT 値算出（配列）	FUN	(@)AryCRCCCITT EN In Size Initial OutOrder ENO Out	Out:=AryCRCCCITT(In, Size, Initial, OutOrder);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	−	*1
Size	演算対象の要素数		In[]の要素数			1
Initial	初期値		CRC-CCITT 値算出の初期値			0
OutOrder	バイト順序		「In」を処理するバイト順序	_LOW_HIGH, _HIGH_LOW		_HIGH_LOW
Out	CRC-CCITT 値	出力	CRC-CCITT 値	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Size							○													
Initial			○																	
OutOrder	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-548)」参照																			
Out			○																	

機能

演算対象配列 In[]の、In[0]以降の「Size」個の配列要素の CRC-CCITT 値を、XMODEM 方式で算出します。

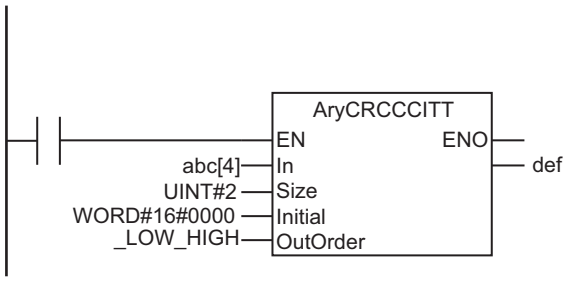
CRC-CCITT 値を算出する際の初期値は、「Initial」で指定します。バイト順序は、「OutOrder」で指定します。
「OutOrder」のデータ型は、列挙型_eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

「Size」=UINT#2、「Initial」=WORD#16#0000、「OutOrder」=_LOW_HIGH の場合の例を、以下に示します。

LD



ST

```
def:=AryCRCCCITT(abc[4], UINT#2,
                 WORD#16#0000,
                 _LOW_HIGH);
```

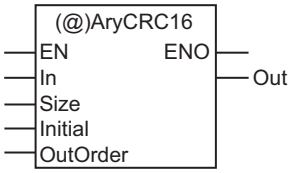


使用上の注意

- 「Size」の値が 0 の場合、「Out」の値は WORD#16#0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「OutOrder」の値が、有効範囲外の場合。
  b)「Size」の値が、In[]の配列領域を超える場合。

AryCRC16

配列の CRC-16 値を、MODBUS 方式で算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AryCRC16	CRC-16 値算出（配列）	FUN		Out:=AryCRC16(In, Size, Initial, OutOrder);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In[] 配列	演算対象配列	入力	演算対象配列	データ型に従う	–	*1
Size	演算対象の要素数		In[]の要素数			1
Initial	初期値		CRC-16 値算出の初期値			16#FFFF
OutOrder	バイト順序		「In」を処理するバイト順序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
Out	CRC-16 値	出力	CRC-16 値	データ型に従う	–	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] 配列		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Size							○													
Initial			○																	
OutOrder	列挙型_eBYTE_ORDER 列挙子は「機能 (P.2-550)」参照																			
Out			○																	

機能

演算対象配列 In[]の、In[0]以降の「Size」個の配列要素の CRC-16 値を、MODBUS 方式で算出します。

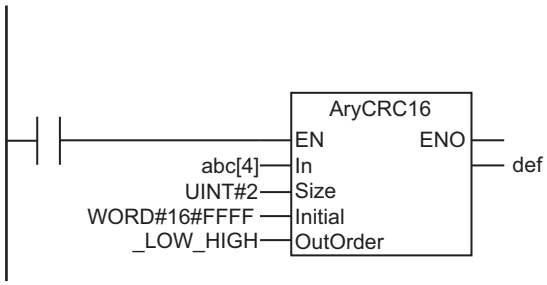
CRC-16 値を算出する際の初期値は、「Initial」で指定します。バイト順序は、「OutOrder」で指定します。
「OutOrder」のデータ型は、列挙型_eBYTE_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_LOW_HIGH	下位バイトが先、上位バイトが後
_HIGH_LOW	上位バイトが先、下位バイトが後

In[]に渡す入力パラメータは、array[3]のように、必ず要素番号も付けて指定します。

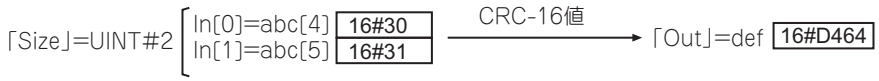
「Size」=UINT#2、「Initial」=WORD#16#FFFF、「OutOrder」=_LOW_HIGH の場合の例を、以下に示します。

LD



ST

```
def:=AryCRC16(abc[4], UINT#2,
                WORD#16#FFFF,
                _LOW_HIGH);
```



使用上の注意

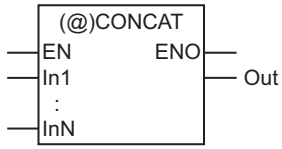
- 「Size」の値が 0 の場合、「Out」の値は WORD#16#0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「OutOrder」の値が、有効範囲外の場合。
  b)「Size」の値が、In[]の配列領域を超える場合。

文字列命令

命令	名称	ページ
CONCAT	文字列・連結	P.2-554
LEFT / RIGHT	文字列・左からの取出／文字列・右からの取出	P.2-556
MID	文字列・任意位置からの取出	P.2-558
FIND	文字列・検索	P.2-560
LEN	文字列・長さ検出	P.2-562
REPLACE	文字列・置換	P.2-563
DELETE	文字列・削除	P.2-565
INSERT	文字列・挿入	P.2-567
GetByteLen	文字列・バイト数取得	P.2-569
ClearString	文字列・クリア	P.2-570
ToUCase / ToLCase	文字列・大文字変換／文字列・小文字変換	P.2-571
TrimL / TrimR	文字列・左トリム／文字列・右トリム	P.2-573
AddDelimiter	デリミタ付加文字列書込	P.2-575
SubDelimiter	文字列読出デリミタ削除	P.2-586

CONCAT

2～5 個の文字列を連結します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CONCAT	文字列・連結	FUN		Out:=CONCAT(In1, ..., InN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1～InN	連結対象	入力	連結する文字列 N は 2～5	データ型に従う	－	" *1
Out	連結結果	出力	連結結果文字列	データ型に従う	－	－

*1. 「InN」に接続する入力パラメータを省略した場合、初期値は適用されず、ビルド時に異常になります。
たとえば、N=3 の場合、「In1」と「In2」に接続する入力パラメータを省略すると初期値が適用されますが、「In3」に接続する入力パラメータを省略すると、ビルド時に異常になります。

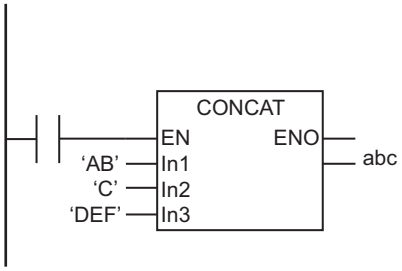
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1～InN																				○
Out																				○

機能

2～5 個の連結対象文字列「In1」～「InN」を、この順番で連結します。終端には NULL 文字が付きます。

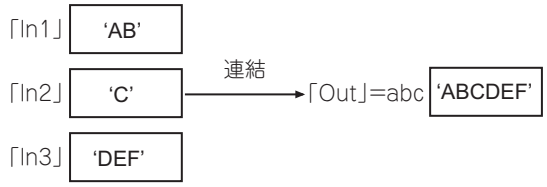
「In1」='AB'、「In2」='C'、「In3」='DEF'の場合の例を、以下に示します。変数「abc」の値は、'ABCDEF'になります。

LD



ST

```
abc:=CONCAT('AB', 'C', 'DEF');
```



使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 連結した文字列の長さが、1986 バイトを超える場合

LEFT / RIGHT

文字列から、指定数の文字列を取り出します。

LEFT　：文字列の左（先頭）から取り出します。
RIGHT　：文字列の右（末尾）から取り出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LEFT	文字列・左からの取出	FUN	(@)LEFT EN　ENO In　Out L	Out:=LEFT(In, L);
RIGHT	文字列・右からの取出	FUN	(@)RIGHT EN　ENO In　Out L	Out:=RIGHT(In, L);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	取出対象	入力	取出を行う対象文字列	データ型に従う	−	''
L	文字数		取り出す文字数	0〜1985		1
Out	取出結果	出力	取出結果文字列	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
L							○													
Out																				○

機能

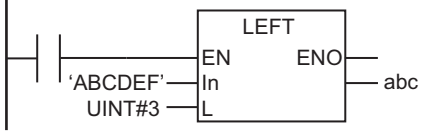
取出対象文字列「In」から、文字数「L」で指定した文字数の文字列を取り出します。取出結果「Out」の終端には、NULL 文字が付きます。

LEFT

「In」の左（先頭）から、取り出します。

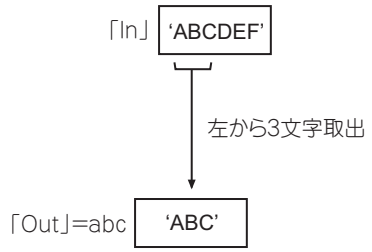
「In」='ABCDEF'、「L」=UINT#3 の場合の例を、以下に示します。変数「abc」の値は、'ABC'になります。

LD



ST

```
abc:=LEFT('ABCDEF', UINT#3);
```

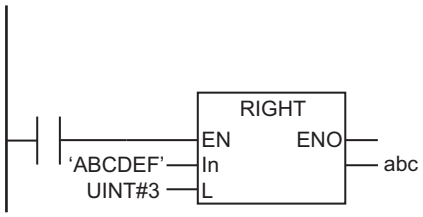


RIGHT

「In」の右（末尾）から、取り出します

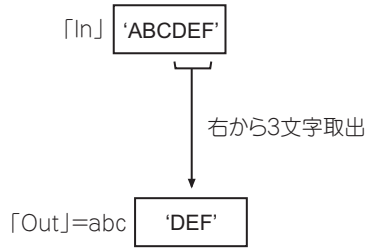
「In」='ABCDEF'、「L」=UINT#3 の場合の例を、以下に示します。変数「abc」の値は、'DEF'になります。

LD



ST

```
abc:=RIGHT('ABCDEF', UINT#3);
```

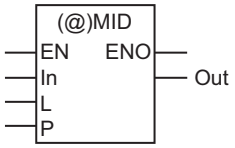


使用上の注意

- 「L」の値が「In」の文字数より大きい、または有効範囲内の場合、異常にはならず、「In」のすべての文字が「Out」にコピーされます。
- 「L」の値が 0 の場合、異常にはならず、NULL 文字だけを「Out」に代入します。
- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が、文字コード異常の場合。

MID

文字列の任意位置から、指定数の文字列を取り出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MID	文字列・任意位置からの取出	FUN		Out:=MID(In, L, P);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	取出対象	入力	取出を行う対象文字列	データ型に従う	−	''
L	文字数		取り出す文字数	0～1985		1
P	取出開始位置		取出を行う開始位置	1～1985		
Out	取出結果	出力	取出結果文字列	データ型に従う	−	−

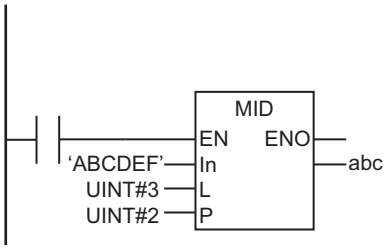
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
L							○													
P							○													
Out																				○

機能

対象文字列「In」から、文字数「L」で指定した数の文字列を取り出します。取出の開始位置は、開始位置「P」で指定します。取出結果「Out」の終端には、NULL 文字が付きます。

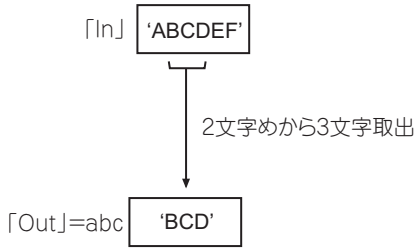
「In」='ABCDEF'、「L」=UINT#3、「P」=UINT#2 の場合の場合の例を、以下に示します。変数「abc」の値は、'BCD'になります。

LD



ST

```
abc:=MID('ABCDEF', UINT#3, UINT#2);
```



使用上の注意

- 「L」の値が 0 の場合、異常にはならず、NULL 文字だけを「Out」に代入します。
- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が、文字コード異常の場合。
  b)「In」が、「P」の値で示した位置から「L」の値で示した数だけの文字を持たない場合。
  c)「P」の値が 0 の場合。

FIND

文字列の中の指定文字列の位置を検索します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FIND	文字列・検索	FUN	(@)FIND EN ENO In1 Out In2	Out:=FIND(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	検索対象	入力	検索の対象文字列	データ型に従う	−	"
In2	検索文字列		検索する文字列			
Out	検索結果	出力	検索結果	0〜1985	−	−

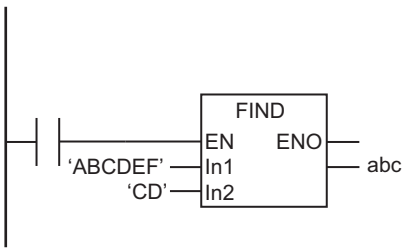
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				○
In2																				○
Out							○													

機能

検索対象「In1」の中の、検索文字列「In2」の位置を検索します。検索結果「Out」には、「In2」が「In1」の先頭から何文字めにあるかが代入されます。
「In1」の中に「In2」が存在しない場合は、「Out」の値は 0 になります。

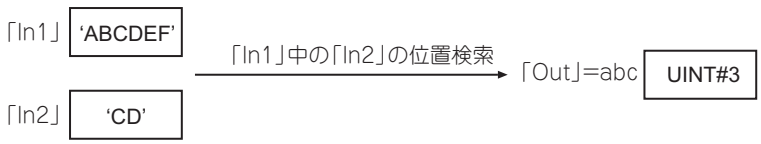
「In1」='ABCDEF'、「In2」='CD'の場合の例を、以下に示します。変数「abc」の値は、UINT#3 になります。

LD



ST

```
abc:=FIND('ABCDEF', 'CD');
```



使用上の注意

- 「In2」は「In1」よりも文字数を少なくしてください。多い場合は、「Out」の値は 0 になります。
- 「In1」の中に「In2」が複数含まれている場合、「Out」の値は、「In1」の先頭から検索して最初に現れる「In2」の位置になります。
- 「In1」、「In2」の値が、どちらも NULL 文字だけの場合、「Out」の値は 1 になります。
- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In1」または「In2」が、文字コード異常の場合。

LEN

文字列の文字数を求めます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LEN	文字列・長さ検出	FUN	(@)LEN EN ENO In Out	Out:=LEN(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	長さ検出対象	入力	長さ検出の対象文字列	データ型に従う	−	''
Out	検出結果	出力	長さ検出結果	0～1985	−	−

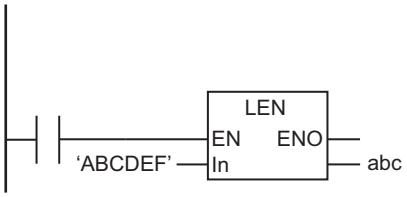
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out							○													

機能

長さ検出対象文字列「In」の、文字数を求めます。「In」の終端の NULL 文字は、文字数に含みません。

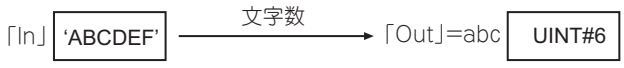
「In」='ABCDEF'の場合の例を、以下に示します。変数「abc」の値は UINT#6 になります。

LD



ST

```
abc:=LEN('ABCDEF');
```



使用上の注意

- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が、文字コード異常の場合。

REPLACE

文字列の一部を、別の文字列で置換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
REPLACE	文字列・置換	FUN	(@)REPLACE EN ENO In1 Out In2 L P	Out:=REPLACE(In1, In2, L, P);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	置換対象	入力	置換の対象文字列	データ型に従う	−	''
In2	置換文字列		置換する文字列			
L	文字数		置換する文字数	0～1985		1
P	置換開始位置		置換を開始する位置	1～1985		
Out	置換結果	出力	置換後の文字列	データ型に従う	−	−

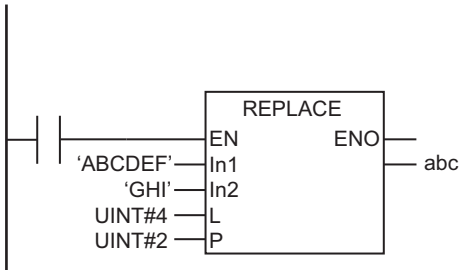
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				○
In2																				○
L							○													
P							○													
Out																				○

機能

置換対象文字列「In1」の一部を、置換文字列「In2」で置換します。
まず「In1」の置換開始位置「P」の値で示す位置から、文字数「L」の値分の文字を除去します。空いた位置に、「In2」をはめこみます。置換結果「Out」の終端には、NULL 文字が付きます。

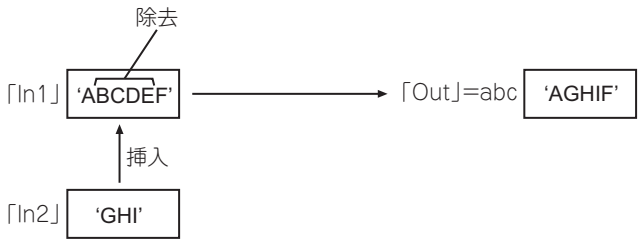
「In1」='ABCDEF'、「In2」='GHI'、「P」=UINT#2、「L」=UINT#4 の場合の例を、以下に示します。変数「abc」の値は、'AGHIF'になります。

LD



ST

```
abc:=REPLACE('ABCDEF', 'GHI', UINT#4, UINT#2);
```

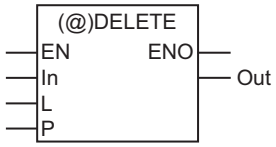


使用上の注意

- 「L」の値が 0 の場合、異常にはならず、「In1」のすべての文字が「Out」にコピーされます。
- 「In2」の値が 0 の場合、「In1」の先頭から「P」番めの文字から「L」文字分削除されます。
- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a）「In1」が、文字コード異常の場合。
  b）「In1」が、「P」の値で示した位置から「L」の値で示した数だけの文字を持たない場合。
  c）「P」の値が 0 の場合。
  d）置換後の文字列の長さが、1986 バイトを超える場合

DELETE

文字列の一部または全部を削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DELETE	文字列・削除	FUN		Out:=DELETE(In, L, P);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	削除対象	入力	削除の対象文字列	データ型に従う	−	"
L	文字数		削除する文字数	0～1985		1
P	削除開始位置		削除を開始する位置	1～1985		
Out	削除結果	出力	削除後の文字列	データ型に従う	−	−

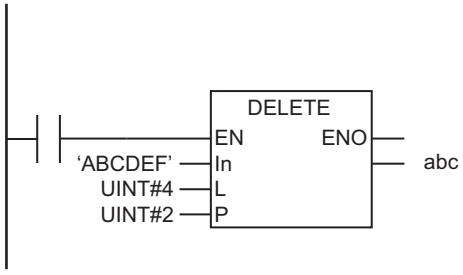
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
L							○													
P							○													
Out																				○

機能

削除対象文字列「In」の削除開始位置「P」の値で示す位置から、文字数「L」の値分の文字を削除します。削除結果「Out」の終端には、NULL 文字が付きます。

「In」='ABCDEF'、「L」=UINT#4、「P」=UINT#2 の場合の例を、以下に示します。変数「abc」の値は、'AF'になります。

LD



ST

```
abc:=DELETE('ABCDEF', UINT#4, UINT#2);
```

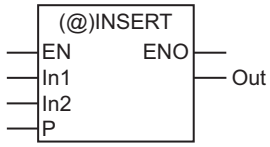


使用上の注意

- 「L」の値が 0 の場合、異常にはならず、「In」のすべての文字が「Out」にコピーされます。
- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が、文字コード異常の場合。
  b)「In」が、「P」の値で示した位置から「L」の値で示した数だけの文字を持たない場合。
  c)「P」の値が 0 の場合。

INSERT

文字列の中に、別の文字列を挿入します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
INSERT	文字列・挿入	FUN		Out:=INSERT(In1, In2, P);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	挿入対象	入力	挿入の対象文字列	データ型に従う	−	"
In2	挿入文字列		挿入する文字列			
P	挿入開始位置		挿入を開始する位置	0～1985		0
Out	挿入結果	出力	挿入後の文字列	データ型に従う	−	−

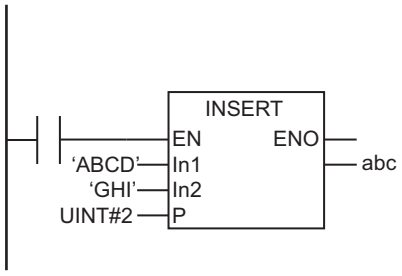
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				○
In2																				○
P							○													
Out																				○

機能

挿入対象文字列「In1」の挿入開始位置「P」の値で示す位置に、挿入文字列「In2」を挿入します。挿入結果「Out」の終端には、NULL 文字が付きます。

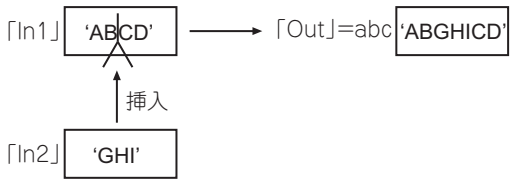
「In1」='ABCD'、「In2」='GHI'、「P」=UINT#2 の場合の例を、以下に示します。変数「abc」の値は、'ABGHICD'になります。

LD



ST

```
abc:=INSERT('ABCD', 'GHI', UINT#2);
```



参考

「P」の値が 0 の場合は、「In2」の後ろに「In1」が連結されます。

使用上の注意

- 多バイト文字も、1 文字と数えます。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In1」が、文字コード異常の場合。
  b)「P」の値が「In1」の文字数より大きい場合。
  c）挿入後の文字列の長さが、1986 バイトを超える場合

GetByteLen

文字列のバイト数をカウントします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetByteLen	文字列・バイト数取得	FUN	(@)GetByteLen EN ENO In Out	Out:=GetByteLen(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	カウント対象	入力	バイト数をカウントする対象文字列	データ型に従う	−	''
Out	バイト数	出力	バイト数	0〜1985	バイト	−

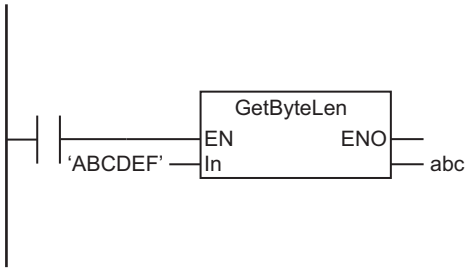
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out							○													

機能

カウント対象文字列「In」の、先頭から末尾までのバイト数をカウントします。文字列終端の NULL 文字は、バイト数に含みません。

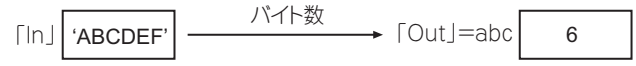
「In」='ABCDEF'の場合の例を、以下に示します。変数「abc」の値は、6 になります。

LD



ST

```
abc:=GetByteLen('ABCDEF');
```

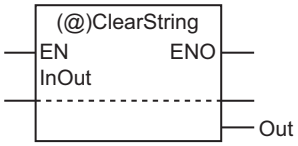


参考

「In」に ASCII 文字しか含まれない場合、本命令の結果は、LEN 命令の結果と一致します。

ClearString

文字列をクリアします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ClearString	文字列・クリア	FUN		ClearString(InOut);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
InOut	クリア文字列	入出力	クリアする文字列	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

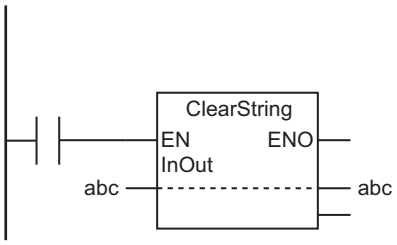
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut																				○
Out	○																			

機能

クリア文字列「InOut」を、クリアします。「InOut」のすべての領域に、NULL 文字を格納します。

記述例を、以下に示します。STRING 型変数「abc」の値が、すべて NULL 文字になります。

LD



ST

```
ClearString(abc);
```

「InOut」のすべての領域に NULL 文字を格納します。

「abc」が 5 文字の STRING 型変数の場合は、以下のようになります。

「InOut」=abc

NULL	NULL	NULL	NULL	NULL

使用上の注意

本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。

ToUCase / ToLCase

ToUCase ： 文字列中の半角アルファベットをすべて大文字に変換します。
ToLCase ： 文字列中の半角アルファベットをすべて小文字に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ToUCase	文字列・大文字変換	FUN	(@)ToUCase EN ENO In Out	Out:=ToUCase(In);
ToLCase	文字列・小文字変換	FUN	(@)ToLCase EN ENO In Out	Out:=ToLCase(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	変換対象	入力	変換の対象文字列	データ型に従う	－	"
Out	変換結果	出力	変換後の文字列	データ型に従う	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

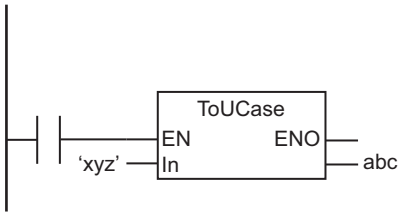
機能

ToUCase

変換対象文字列「In」の半角アルファベットを、すべて大文字に変換します。
文字列終端の NULL 文字を含めて出力します。また、半角アルファベット以外の文字に影響はありません。

ToUCase 命令で、「In」='xyz'の例を、以下に示します。変数「abc」の値は、'XYZ'になります。

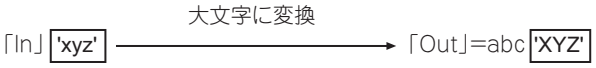
LD



ST

```
abc:=ToUCase('xyz');
```

「In」の半角アルファベットを、すべて大文字に変換します。



ToLCase

変換対象文字列「In」の半角アルファベットを、すべて小文字に変換します。
文字列終端の NULL 文字を含めて出力します。また、半角アルファベット以外の文字に影響はありません。

使用上の注意

- 全角アルファベットは、変換対象外です。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」が、文字コード異常の場合。

TrimL / TrimR

TrimL ： 文字列中の先頭の空白を削除します。
TrimR ： 文字列中の末尾の空白を削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TrimL	文字列・左トリム	FUN	(@)TrimL EN ENO In Out	Out:=TrimL(In);
TrimR	文字列・右トリム	FUN	(@)TrimR EN ENO In Out	Out:=TrimR(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	削除対象	入力	削除の対象文字列	データ型に従う	－	''
Out	削除結果	出力	削除後の文字列	データ型に従う	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

機能

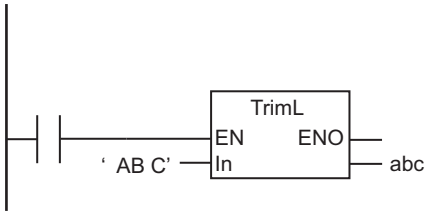
TrimL

削除対象文字列「In」の、先頭から最初に現れる文字までの空白を、削除します。
先頭に空白がない場合は、何も処理しません。

文字列終端の NULL 文字を含めて出力します。また、ASCII コードの空白(16#20)と、日本語の全角空白(16#E38080)の両方を、空白とみなします。

TrimL 命令で、「In」=' AB C'の場合の例を、以下に示します。変数「abc」の値は、'AB C'になります。

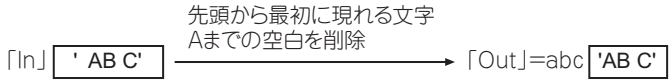
LD



ST

```
abc:=TrimL(’ AB C’);
```

「In」の先頭から最初に現れる文字までの空白を削除します。



TrimR

削除対象文字列「In」の、最後の文字から末尾までの空白を、削除します。
末尾に空白がない場合は、何も処理しません。

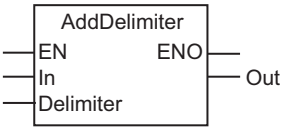
文字列終端の NULL 文字を含めて出力します。また、ASCII コードの空白(16#20)と、日本語の全角空白(16#E38080)の両方を、空白とみなします。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
・「In」が文字コード異常の場合。

AddDelimiter

構造体全体の値を、デリミタを付加した文字列に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AddDelimiter	デリミタ付加文字列書込	FUN		Out:=AddDelimiter(In, Delimiter);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	入力構造体	入力	文字列変換対象となる構造体全体	メンバのデータ型に従う	–	*1
Delimiter	デリミタ		デリミタ	_COMMA, _TAB, _SEMICOLON, _SPACE	–	_COMMA
Out	戻り値	出力	デリミタを付加した文字列	最大 1986 バイト（ 半角英数字 1985 文字＋終端 NULL 文字）	–	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	構造体全体																			
Delimiter	列挙型_eDELIMITER　列挙子は「機能 (P.2-575)」参照																			
Out																				○

機能

入力構造体「In」のメンバの値を、先頭のメンバから順に文字列に変換し、デリミタ「Delimiter」で区切って連結します。連結した文字列は、戻り値「Out」に出力します。「Out」の終端には NULL 文字が付加されます。

「Delimiter」のデータ型は、列挙型_eDELIMITER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_COMMA	','（カンマ）
_TAB	'$T'（タブ）
_SEMICOLON	';'（セミコロン）
_SPACE	' '（空白文字）

「In」のメンバの値は、データ型によって、変換します。

● ブール型

FALSE は'0'、TRUE は'1'に変換します。

● ビット列型

16 進数として英数字で表現したものを文字列に変換します。ただし、16 進数であることを示す接頭語'16#'は文字列に含みません。
メンバの値が、メンバのデータ型の最大けた数に満たない場合は、上位けたを'0'で埋めます。つまり、ゼロパディングします。
データ型によって、文字列の文字数は以下のように異なります。

メンバのデータ型	文字数
BYTE	半角英数 2 文字
WORD	半角英数 4 文字
DWORD	半角英数 8 文字
LWORD	半角英数 16 文字

いくつかの例を以下に示します。

メンバの値	変換後の文字列
BYTE#16#AB	'AB'
LWORD#16#0123	'0000000000000123'

● 整数型

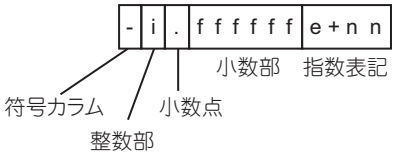
整数の値を文字列に変換します。上位けたの 0 は文字列として出力しません。メンバの値が負の場合は、文字列の先頭にマイナス符号'-'を付けます。

いくつかの例を以下に示します。

メンバの値	変換後の文字列
UINT#0012	'12'
LINT#-12	'-12'

● 実数型

メンバの値を変換した文字列の構成は、以下のようになります。



項目	内容
符号カラム	メンバの値が負数の場合だけ、'-'（マイナス）が付加されます。 メンバの値が正数の場合、'+'（プラス）は付加されません。
整数部	常に 1 けたで表記されます。
小数点	メンバの値が小数でなくても、常に表記されます。
小数部	メンバが REAL 型の場合 6 けた、LREAL 型の場合 14 けたで表記されます。
指数表記	常に表記されます。 'e'は指数表記の“e”を表します。 'nn'のけた数は、2 または 3 です。 'nn'の符号は、メンバの値の絶対値が 1.0 以上の場合'+'（プラス）、1.0 未満の場合'-'（マイナス）です。ただし、メンバの値が 0 の場合は'+'（プラス）です。

メンバの値が、無限大、非数の場合、文字列は以下のようになります。

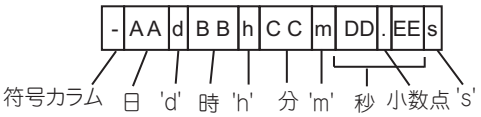
メンバの値	文字列
+∞	'inf'
-∞	'-inf'
非数	'nan'または'-nan'

いくつかの例を以下に示します。

メンバの値	変換後の文字列
REAL#3.14e1	'3.140000e+01'
REAL#-123.4567	'-1.234567e+02'
REAL#0	'0.000000e+00'
LREAL#0.00123456789	'1.23456789000000e-03'
LREAL#1.0e308	'1.00000000000000e+308'

● 持続時間型

メンバの値を変換した文字列の構成は、以下のようになります。



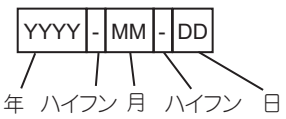
項目	内容
符号カラム	メンバの値が負数の場合だけ、'-'（マイナス）が付加されます。 メンバの値が正数の場合、'+'（プラス）は付加されません。
日	常に表記されます。 値の範囲は 0～106751 です。 上位けたには 0 を補いません。
時	常に 2 けたで表記されます。 値の範囲は 00～23 です。
分	常に 2 けたで表記されます。 値の範囲は 00～59 です。
秒	常に表記されます。 'DD'の値は常に 2 けたで表記され、範囲は 00～59 です。 'EE'の値は常に 9 けたで表記され、範囲は 000000000～999999999 です。
'd'、'h'、'm'、's'、小数点	常に表記されます。

いくつかの例を以下に示します。

メンバの値	変換後の文字列
T#-180122000ms	'-2d02h02m02.000000000s'
T#100d2h3m5.678s	'100d02h03m05.678000000s'
T#2h3m5.678s	'0d02h03m05.678000000s'

● 日付型

メンバの値を変換した文字列の構成は以下のようになります。



月、日はそれぞれ 2 けたで文字列に変換します。
例を以下に示します。

メンバの値	変換後の文字列
D#2010-1-2	'2010-01-02'

● 日付時刻型

メンバの値を変換した文字列の構成は以下のようになります。

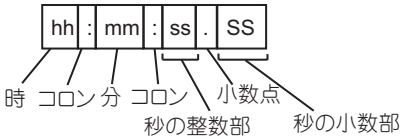


月 MM、日 DD、時 hh、分 mm、秒の整数部 ss はそれぞれ 2 けたで文字列に変換します。秒の小数部 SS は 9 けたで文字列に変換します。
例を以下に示します。

メンバの値	変換後の文字列
DT#2004-09-23-12:16:8.12	'2004-09-23-12:16:08.120000000'

● 時刻型

メンバの値を変換した文字列の構成は以下のようになります。



時 hh、分 mm、秒の整数部 ss はそれぞれ 2 けたで文字列に変換します。秒の小数部 SS は 9 けたで文字列に変換します。
例を以下に示します。

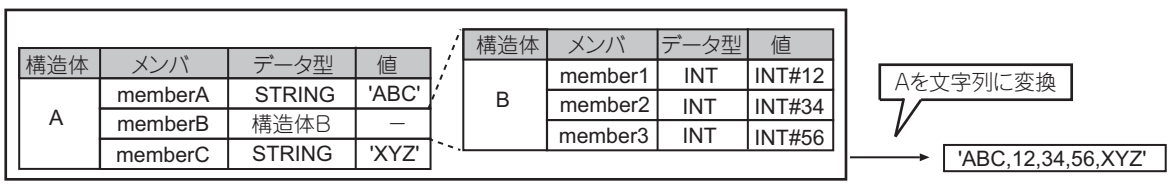
メンバの値	変換後の文字列
TOD#2:16:28.12	'02:16:28.120000000'

● 文字列型

そのままの文字列になります。ただし、終端の NULL 文字は含めません。
たとえば、メンバの値が終端に NULL 文字を含む'ABC'の場合は、終端に NULL 文字を含まない文字列'ABC'に変換します。

● 構造体

メンバのデータ型が構造体以外の型である階層まで、メンバの値を先頭から順に変換します。それぞれのメンバのデータ型の変換ルールに従って、メンバの値を文字列に変換します。
たとえば、構造体 A のメンバのデータ型に構造体 B がある場合、以下のように変換します。この例では、デリミタとしてカンマを使用します。



● **列挙型**

列挙型の値を DINT 型の値として変換します。
たとえば、列挙型「Color」の取り得る列挙子は「red」、「yellow」、「green」の 3 つとします。それぞれの列挙子に対応する値は、「red」が 1、「yellow」が 2、「green」が 3 とします。列挙型「Color」のメンバの値が「yellow」の場合は、文字列は'2'となります。

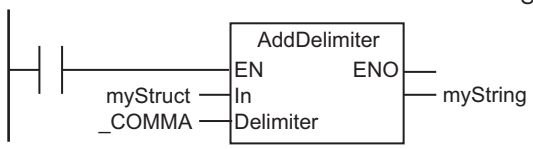
● **配列**

各要素の間をデリミタで区切ります。各要素の値は、配列のデータ型の変換ルールに従って変換されます。1 次元配列だけ変換できます。
たとえば、INT 型の配列 myArray[0..2]があるとします。myArray[0]の値が INT#225、myArray[1]の値が INT#-128、myArray[2]の値が INT#0、デリミタがカンマの場合、文字列は'225,-128,0'となります。

記述例

構造体「myStruct」を文字列「myString」に変換する場合の記述例を以下に示します。デリミタは','（カンマ）を使用します。

LD



ST

```
myString:=AddDelimiter(myStruct,_COMMA);
```

構造体	メンバ	データ型	値
myStruct	member1	STRING	'XYZ'
	member2	INT	INT#12
	member3	WORD	WORD#16#00AB
	member4	REAL	REAL#3.14e0

文字列に変換

「Out」=myString 'XYZ,12,00AB,3.140000e+00'

参考

- SD メモリカード内の指定した CSV ファイルに値を書き込む場合、「FilePuts (P.2-1275)」命令に本命令を併せて使用すると便利です。使用方法は、「サンプルプログラム (P.2-580)」を参照してください。
- 本命令で変換した文字列は、「SubDelimiter (P.2-586)」命令で構造体のメンバの値として読み出すことができます。

使用上の注意

- デリミタを「In」のメンバの値の一部には使用しないでください。デリミタを「In」のメンバの値の一部に使用すると、SubDelimiter 命令で文字列を構造体のメンバの値に変換するときに、正しく変換できません。
- つぎの場合、異常となります。「ENO」 は FALSE となり、「Out」は変化しません。
  a）変換結果の文字列が、終端の NULL 文字を含めて 1986 バイトを超える場合
  b）「In」に 2 次元以上の配列のメンバがある場合
  c）「In」に共用体のメンバがある場合

サンプルプログラム

10 個の SINT 型の変数をメンバに持つ構造体「myStruct」があるとします。
構造体 myStruct の配列 myArray[0..99]の内容を、100 行の CSV ファイル形式で、SD メモリカードのファイル'ABC.csv'に格納します。
1 行には、配列要素のメンバの数値が、文字列として 10 個並び、それらの間にはカンマを挿入します。行末には、改行コード CR + LF を付加します。

処理手順は以下のとおりです。

1 **FileOpen 命令を使って、ファイル'ABC.csv'をオープンします。**

2 **AddDelimiter 命令で、myArray[]の要素を 1 行分だけ、STRING 型の変数「Temp」に変換します。**

3 **CONCAT 命令で、「Temp」と CR+LF を結合し、STRING 型の変数「StrDat」に格納します。**

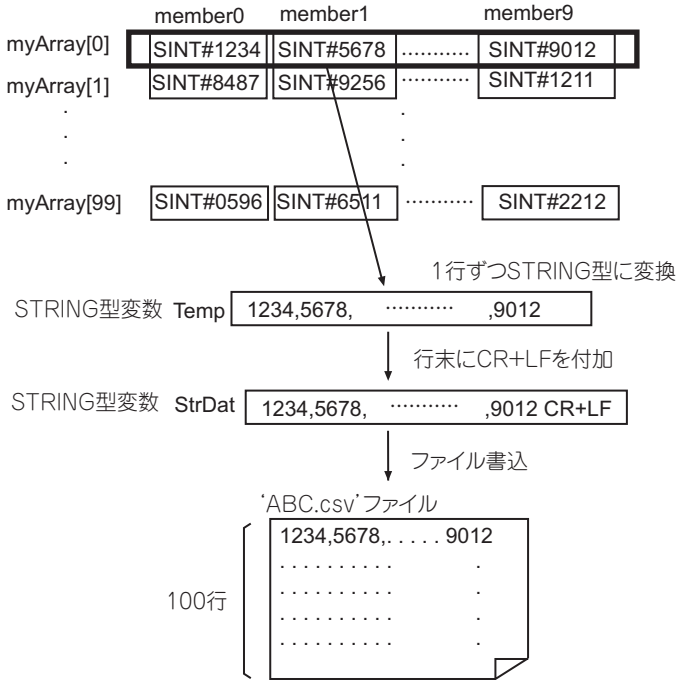
4 **「StrDat」をファイルに書き込みます。**

5 **上記 2～4 を、100 行分繰り返します。**

6 **FileClose 命令を使って、ファイルをクローズします。**

構造体	メンバ	データ型
myStruct	member0	SINT
	member1	SINT
	⋮	
	member9	SINT

構造体型myStructの配列myArray[0..99]



データ型の定義

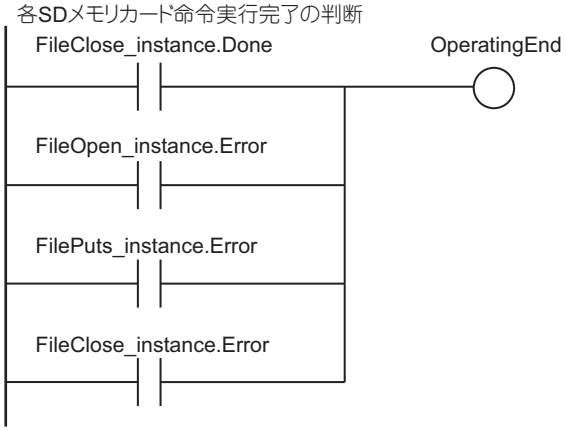
名称	データ型	コメント
myStruct	STRUCT	構造体
member0	SINT	メンバ
member1	SINT	メンバ
member2	SINT	メンバ
member3	SINT	メンバ
member4	SINT	メンバ
member5	SINT	メンバ
member6	SINT	メンバ
member7	SINT	メンバ
member8	SINT	メンバ
member9	SINT	メンバ

LD

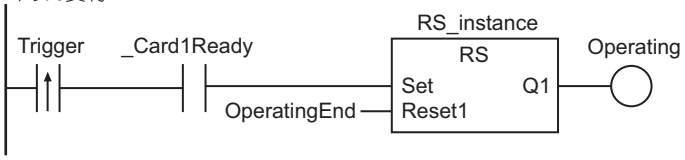
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	False	処理終了
	Trigger	BOOL	False	実行条件
	Operating	BOOL	False	処理中
	Index	INT	0	インデックス
	Fid	DWORD	16#0	ファイル ID
	StrDat	STRING[256]	''	文字列データ
	myArray	ARRAY[0..99] OF myStruct	[100((member0:=0,member1:=0,member2:=0,member3:=0,member4:=0,member5:=0,member6:=0,member7:=0,member8:=0,member9:=0))]	数値データ
	Temp	STRING[256]	''	テンポラリ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

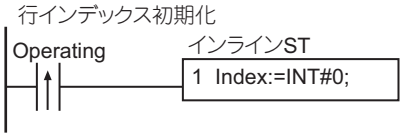
各SDメモリカード命令実行完了の判断



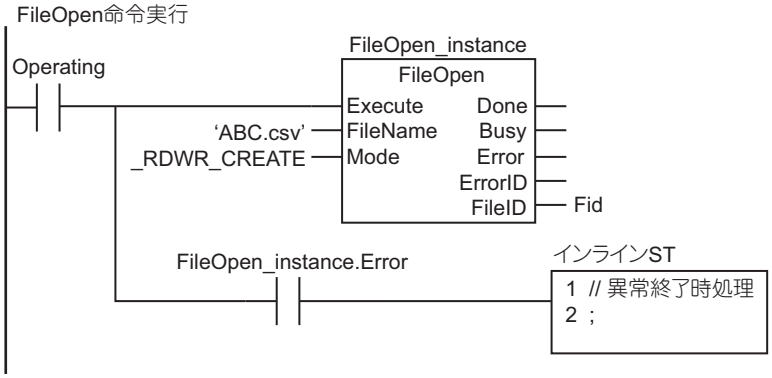
トリガ受付



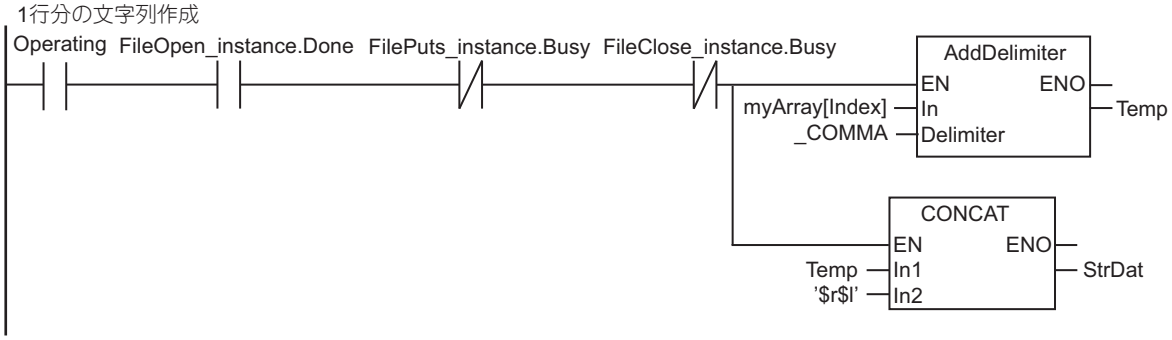
行インデックス初期化

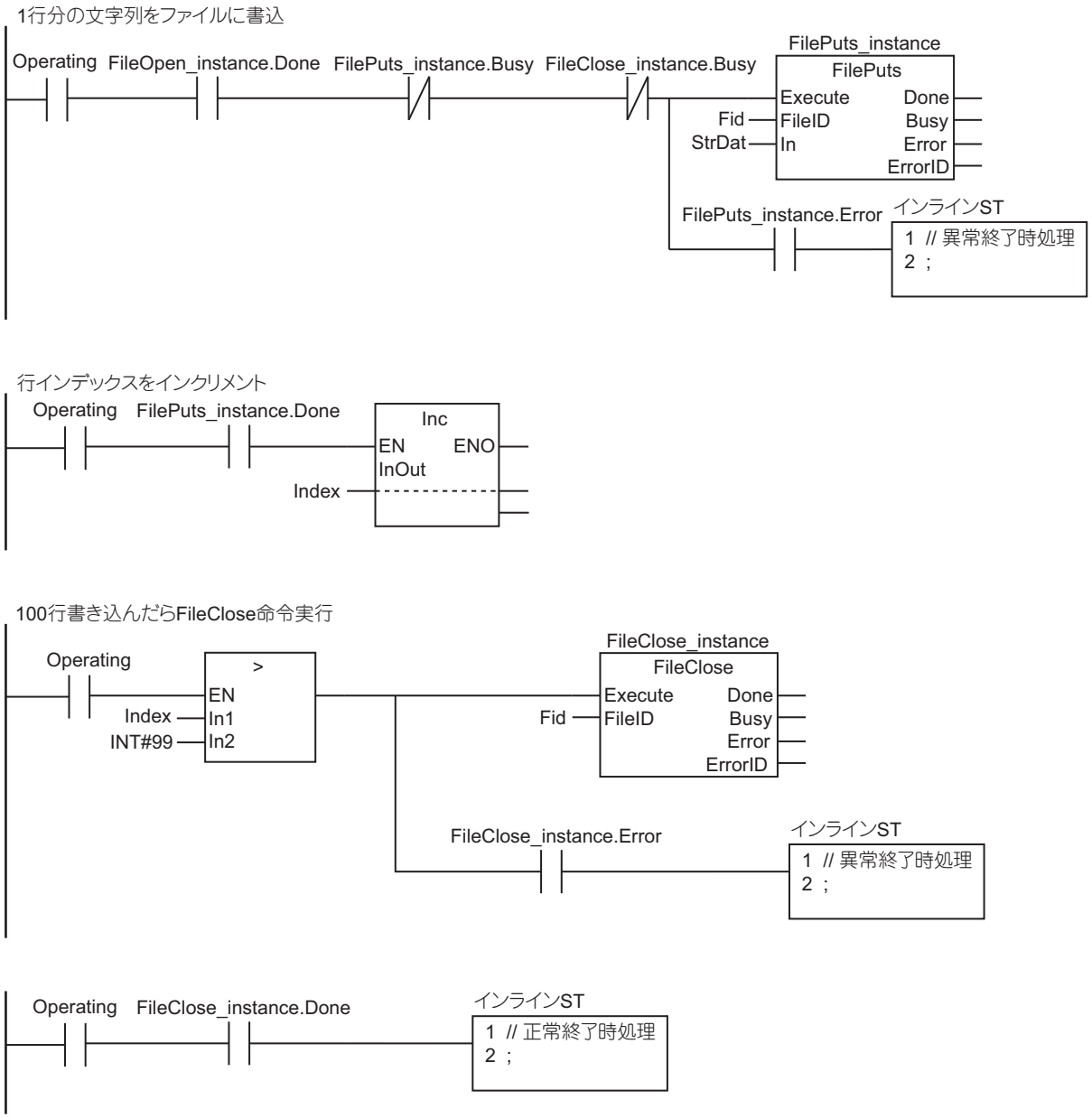


FileOpen命令実行



1行分の文字列作成





ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	False	実行条件
	LastTrigger	BOOL	False	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	False	処理開始
	Operating	BOOL	False	処理中
	Stage	INT	0	状態遷移
	Index	INT	0	インデックス
	Fid	DWORD	16#0	ファイル ID
	StrDat	STRING[256]	''	文字列データ

内部変数	名称	データ型	初期値	コメント
	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))]	数値データ
	Temp	STRING[256]	''	テンポラリ
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
  OperatingStart:=TRUE;
  Operating :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
  FileOpen_instance(Execute:=FALSE);
  FilePuts_instance(Execute:=FALSE);
  FileClose_instance(Execute:=FALSE);
  Stage :=INT#1;
  Index :=INT#0; // 行インデックス初期化
  OperatingStart:=FALSE;
END_IF;

// 各命令実行
IF (Operating=TRUE) THEN
  CASE Stage OF
  1 : // ファイルオープン
    FileOpen_instance(
      Execute :=TRUE,
      FileName:='ABC.csv', // ファイル名
      Mode :=_RDWR_CREATE, // ファイル読出
      FileID =>Fid); // ファイル ID

    IF (FileOpen_instance.Done=TRUE) THEN
      Stage:=INT#2; // 正常終了
    END_IF;

    IF (FileOpen_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  2 : // 1 行分の文字列作成
```

```
    StrDat:='';

    Temp :=AddDelimiter(myArray[Index],_COMMA);
    StrDat:=CONCAT(In1:=Temp, In2:='$r$l');

    Stage:=INT#3;

  3 : // 文字列書込
    FilePuts_instance(
      Execute:=TRUE,
      FileID :=Fid,
      In :=StrDat);

    IF (FilePuts_instance.Done=TRUE) THEN
      Index:=Index+INT#1;

      IF (Index>INT#99) THEN // 100 行書き込んだ
        Stage:=INT#4;
      ELSE
        FilePuts_instance(Execute:=FALSE);
        Stage:=INT#2;
      END_IF;
    END_IF;

    IF (FilePuts_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  4 : // ファイルクローズ
    FileClose_instance(
      Execute:=TRUE,
      FileID :=Fid); // ファイル ID

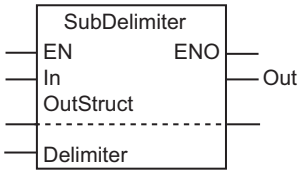
    IF (FileClose_instance.Done=TRUE) THEN
      Operating:=FALSE; // 正常終了
    END_IF;

    IF (FileClose_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  99 : // 異常終了処理
    Operating:=FALSE;
  END_CASE;
END_IF;
```

SubDelimiter

文字列からデリミタで区切られたデータを読み出し、構造体のメンバの値として格納します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SubDelimiter	文字列読出デリミタ削除	FUN		Out:=SubDelimiter(In, OutStruct, Delimiter);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	入力文字列	入力	構造体のメンバの値に変換するデータを、デリミタで区切った文字列	最大 1986 バイト（ 半角英数字 1985 文字＋終端 NULL 文字）	－	''
Delimiter	デリミタ		デリミタ	_COMMA, _TAB, _SEMICOLON, _SPACE	－	_COMMA
OutStruct	格納先構造体	入出力	変換したデータを格納する構造体全体	最大 8192 バイト	－	－
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Delimiter	列挙型_eDELIMITER　列挙子は「機能 (P.2-586)」参照																			
OutStruct	構造体全体																			
Out	○																			

機能

入力文字列「In」のデリミタ「Delimiter」で区切られた文字列データを、格納先構造体「OutStruct」のメンバの値に変換し、各メンバの値として先頭から順に格納します。

「Delimiter」のデータ型は、列挙型_eDELIMITER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_COMMA	','（カンマ）
_TAB	'$T'（タブ）
_SEMICOLON	';'（セミコロン）
_SPACE	' '（空白文字）

「In」のデリミタで区切られたデータの数が、「OutStruct」のメンバの数より多い場合、余ったデータを無視します。

「In」のデリミタで区切られたデータの数が、「OutStruct」のメンバの数より少ない場合、データが足りない分のメンバの値を変更しません。
「OutStruct」のメンバが構造体で、「In」のデータがその構造体のメンバ数に足りない場合でも、途中までのデータを格納します。
「OutStruct」のメンバが配列で、「In」のデータがその配列の要素数に足りない場合でも、途中までのデータを格納します。

「In」のデリミタで区切られたデータは、文字列型です。「OutStruct」のメンバのデータ型によって、文字列型のデータを変換します。

● ブール型

文字列型のデータが、'FALSE'または'0'の場合、FALSE に変換します。'TRUE'または'1'の場合 TRUE に変換します。
ただし、つぎのような例外があります。
- '0'または'1'の前に'0'が連続してある場合は、それを無視します。
- 'FALSE'、'TRUE'の大文字小文字は区別しません。

文字列型のデータが、'FALSE'、'TRUE'、'0'、'1'以外の値の場合は変換できません。

● ビット列型

変換ルールは、「STRING_TO_**（文字列→ビット列変換グループ）（P.2-304)」命令と同じです。
16 進数の数値を表すデータでない場合は変換できません。

● 整数型

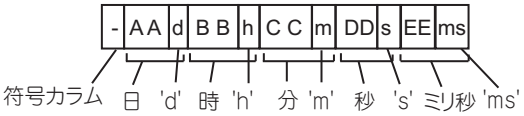
変換ルールは、「STRING_TO_**（文字列→整数変換グループ）（P.2-302)」命令と同じです。
整数型の数値を表すデータでない場合は変換できません。

● 実数型

変換ルールは、「STRING_TO_**（文字列→実数変換グループ）（P.2-306)」命令と同じです。
実数型の数値を表すデータでない場合は変換できません。

● 持続時間型

以下の構造のデータを、持続時間型の値に変換します。



項目	内容
符号カラム	'+'（プラス）または符号カラムがない場合は、メンバの値は正数になります。 '-'（マイナス）の場合は、メンバの値は負数になります。
日	AA の値の小数点以下 12 けため以降は切り捨てます。
時	BB の値の小数点以下 12 けため以降は切り捨てます。
分	CC の値の小数点以下 11 けため以降は切り捨てます。
秒	DD の値の小数点以下 10 けため以降は切り捨てます。
ミリ秒	EE の値の小数点以下 7 けため以降は切り捨てます。

（注）1. 符号カラム、日、時、分、秒、ミリ秒の値の前に' '（空白文字）がある場合、' '（空白文字）を無視します。
（注）2. AA、BB、CC、DD、EE の値の、文字と文字の間に、単独の'_'（アンダーバー）を含む場合、'_'（アンダーバー）を無視します。

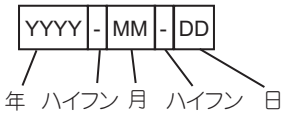
（注）3. 日、時、分、秒、ミリ秒の値が、'.'（ピリオド）を使った実数であっても変換できます。
（注）4. 日、時、分、秒、ミリ秒のうち、いずれかがデータに含まれていれば、他が省略されていても変換できます。
（注）5. 日、時、分、秒、ミリ秒の値の前に'0'が付加されていても変換できます。

ただし、つぎの場合は変換できません。
- 上記の構造でない場合
- 符号カラムと日の間に'_'（アンダーバー）がある場合
- '.'（ピリオド）と'_'（アンダーバー）が連続する場合

たとえば、文字列型のデータが"-0.5d48h0.123456789ms'の場合、メンバの値は
T#-2d12h0m0s0.123456ms(T#-216000000.123456ms)となります。

● 日付型

以下の構造のデータを、日付型の値に変換します。



ただし、つぎのような例外があります。
- 年、月、日の値の前に' '（空白文字）がある場合、' '（空白文字）を無視します。
- 年、月、日の値の、文字と文字の間に、単独の'_'（アンダーバー）を含む場合、'_'（アンダーバー）を無視します。
- 年、月、日の値の前に'0'が付加されていても変換できます。

つぎの場合は変換できません。
- 上記の構造でない場合
- 存在しない日付である場合

たとえば、文字列型のデータが'2000-1-01'の場合、メンバの値は D#2000-01-01 となります。

● 日付時刻型

以下の構造のデータを、日付時刻型の値に変換します。



項目	内容
年、月、日	日付を表す年月日です。
時	値の範囲は 0～23 です。
分	値の範囲は 0～59 です。
秒	値の範囲は、0～59.999999999 です。値が整数の場合、小数点は不要です。
ハイフン、コロン	常に必要です。

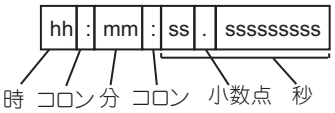
（注）1. 年、月、日、時、分、秒の値の前に' '（空白文字）がある場合、' '（空白文字）を無視します。
（注）2. 年、月、日、時、分、秒の値の、文字と文字の間に、単独の'_'（アンダーバー）を含む場合、'_'（アンダーバー）を無視します。
（注）3. 年、月、日、時、分、秒の値の前に'0'が付加されていても変換できます。

ただし、つぎの場合は変換できません。
- 上記の構造でない場合
- 存在しない日付である場合

たとえば、文字列型のデータが'2000-01-23-4:56:07.89'の場合、メンバの値は DT#2000-01-23-04:56:07.89 となります。

● 時刻型

以下の構造のデータを、時刻型の値に変換します。



項目	内容
時	値の範囲は 0～23 です。
分	値の範囲は 0～59 です。
秒	値の範囲は、0～59.999999999 です。値が整数の場合、小数点である'.'（ピリオド）は不要です。
コロン	常に必要です。

（注）1. 時、分、秒の値の前に' '（空白文字）がある場合、' '（空白文字）を無視します。
（注）2. 時、分、秒の値の、文字と文字の間に、単独の'_'（アンダーバー）を含む場合、'_'（アンダーバー）を無視します。
（注）3. 時、分、秒の値の前に'0'が付加されていても変換できます。

ただし、つぎの場合は変換できません。
- 上記の構造でない場合
- '.'（ピリオド）と'_'（アンダーバー）が連続する場合

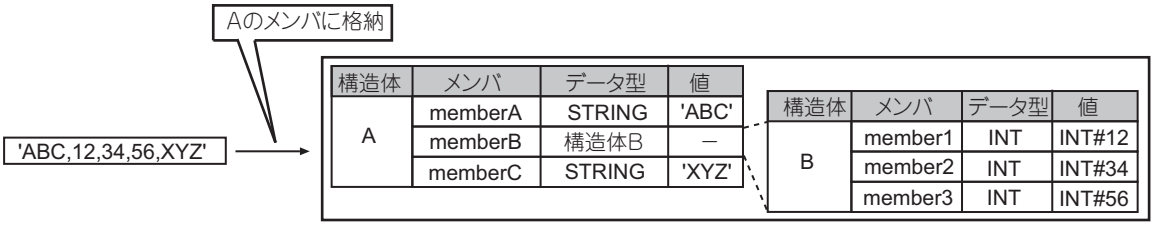
たとえば、文字列型のデータが'12:23:34.567'の場合、メンバの値は TOD#12:23:34.567 となります。

● 文字列型

データの終端に NULL 文字を付加した値がメンバの値となります。ただし、文字列がメンバのデータサイズを超える場合は変換できません。
たとえば、文字列型のデータが終端に NULL 文字のない'ABC'の場合、メンバの値は終端に NULL 文字を付加した'ABC'となります。

● 構造体

文字列型のデータを、メンバのデータ型の変換ルールに従って変換します。メンバが構造体以外のデータ型である階層まで、先頭から順に変換した値をメンバの値として格納します。
たとえば、構造体 A のメンバに構造体 B がある場合、以下のように変換します。



● 列挙型

DINT 型の変数を表す文字列型のデータを、列挙型の列挙子に変換します。
整数型と同じルールで DINT 型に変換し、DINT 型の値を列挙型の値として、列挙型の値に対応する列挙子に変換します。
ただし、文字列型のデータが DINT 型の数値を表す文字列でない場合、変換できません。

たとえば、列挙型「Color」の取り得る列挙子は「red」、「yellow」、「green」の 3 つとします。それぞれの列挙子に対応する値は、「red」が 1、「yellow」が 2、「green」が 3 とします。データが'3'の場合、メンバの値は「green」となります。

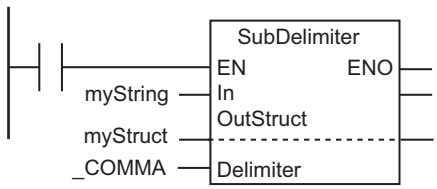
● **配列**

デリミタで区切られたデータを、各要素の値に変換します。変換ルールは、配列のデータ型に従います。メンバが 1 次元配列の場合だけ変換できます。
たとえば、メンバに BYTE 型の配列 myString[0..3]があるとします。文字列'AA,BB,CC,DD'のカンマで区切られたデータを各要素の値に変換する場合、myString[0]は BYTE#16#AA、mySring[1]は BYTE#16#BB、myString[2]は BYTE#16#CC、myString[3]は BYTE#16#DD となります。

記述例

文字列「myString」の、カンマで区切られたデータを読み出し、構造体「myStruct」のメンバの値として格納する場合の記述例を以下に示します。

LD



ST

```
SubDelimiter(myString,myStruct,_COMMA);
```

myString 'XYZ,1234,aB,3.14'

構造体	メンバ	データ型	値
myStruct	member1	STRING	''
	member2	INT	INT#0
	member3	WORD	WORD#16#0000
	member4	REAL	REAL#0

myStringのデータをmyStructのメンバの値として格納

構造体	メンバ	データ型	値
myStruct	member1	STRING	'XYZ'
	member2	INT	INT#1234
	member3	WORD	WORD#16#00AB
	member4	REAL	REAL#3.14e+0

参考

- SD メモリカード内の指定した CSV ファイルから値を読み出す場合、「FileGets (P.2-1268)」命令に本命令を併せて使用すると便利です。使用方法は「サンプルプログラム (P.2-591)」を参照してください。
- 「AddDelimiter (P.2-575)」命令で変換した文字列を、構造体のデータに戻す場合に本命令を使用します。

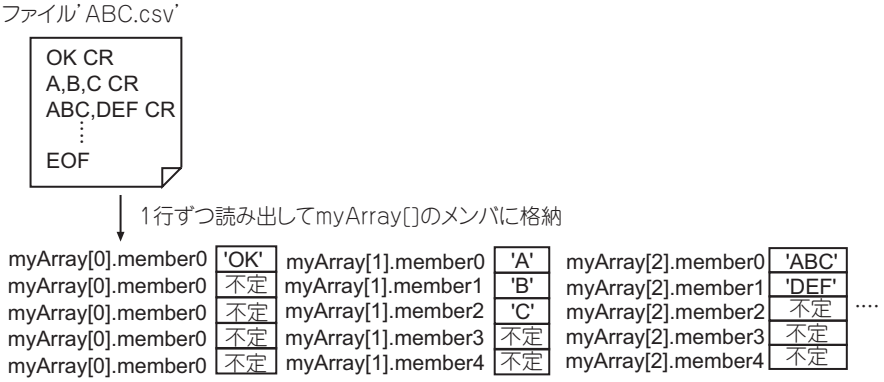
使用上の注意

- 「In」の中でデリミタが連続する場合、デリミタで区切られるデータは存在しません。データが存在しない場合は、「OutStruct」のメンバの値は不定になります。
- デリミタを「In」の中で、デリミタの用途以外に使用しないでください。デリミタの用途以外に使用した場合であっても、本命令はデリミタと認識して動作します。
- 「OutStruct」に文字列型のメンバがある場合、「In」の中の対応するデータには終端に NULL 文字を付加しないでください。「In」の終端以外で NULL 文字を使用すると、NULL 文字までの文字列しか変換できません。
- 「OutStruct」に列挙型のメンバがある場合、「In」の該当箇所は列挙子として定義した値を表すデータにしてください。列挙型変数の値が列挙子として定義された値でなくても、異常にはなりません。

・つぎの場合、異常となります。「ENO」 は FALSE となり、「OutStruct」の値は不定になります。
a)「OutStruct」のメンバのデータ型の値に変換できない場合
b）変換結果が「OutStruct」のメンバのデータ型の値の有効範囲外である場合
c)「OutStruct」に 2 次元以上の配列のメンバがある場合
d)「OutStruct」に共用体のメンバがある場合
e)「OutStruct」のサイズが 8192byte 以上の場合

サンプルプログラム

ファイル名'ABC.csv'のファイルには、改行コード CR により改行された複数行の文字列が格納されています。1 行の文字列は、複数のデータをカンマで区切って表しています。
このファイルから文字列を 1 行ずつ読み出し、カンマで区切られたデータを、構造体「myStruct」の配列変数 myArray[]のメンバの値として先頭から順に格納します。「myStruct」は、5 個の STRING 型の変数をメンバに持つ構造体とします。
ファイルを最後まで読み出したら（ファイル終端 EOF まで読み出したら）、処理を終了します。



処理手順は、以下のとおりです。

1 **FileOpen 命令を使って、ファイル'ABC.csv'をオープンします。**

2 **FileGets 命令を使って、ファイルの内容を 1 行読み出します。**

3 **SubDelimiter 命令を使って、カンマで区切られた文字列を myArray[]のメンバの値として格納します。**

4 **上記 2～3 を、ファイル終端 EOF を読み出すまで続けます。**

5 **FileClose 命令を使って、ファイルをクローズします。**

データ型の定義

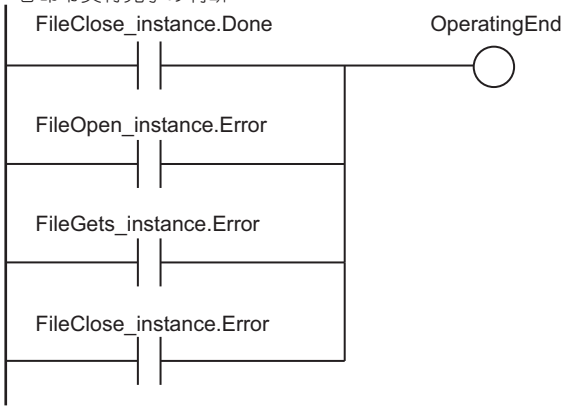
名称	データ型	コメント
myStruct	STRUCT	構造体
member0	STRING	メンバ
member1	STRING	メンバ
member2	STRING	メンバ
member3	STRING	メンバ
member4	STRING	メンバ

LD

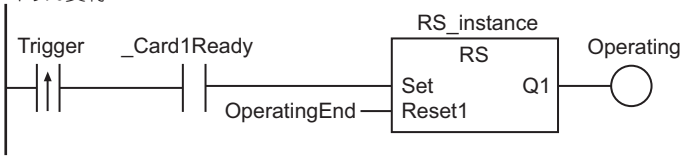
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	False	処理終了
	Trigger	BOOL	False	実行条件
	Operating	BOOL	False	処理中
	Index	INT	0	myArray[]の要素インデックス
	Fid	DWORD	16#0	ファイル ID
	myArray	ARRAY[0..999] OF myStruct	[1000((member0:='',member1:='',member2:='',member3:='',member4:=''))]	整数データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

各命令実行完了の判断

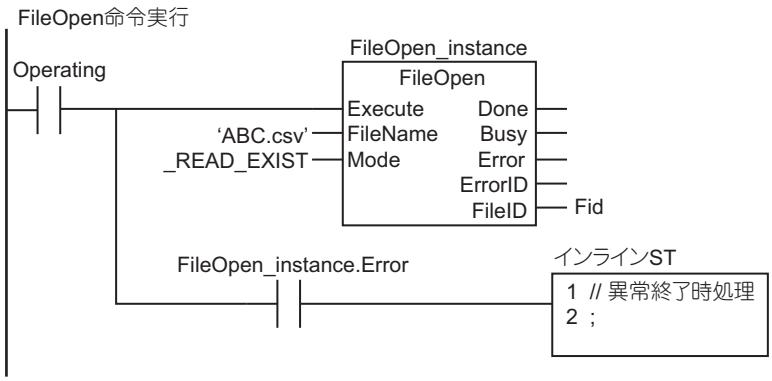


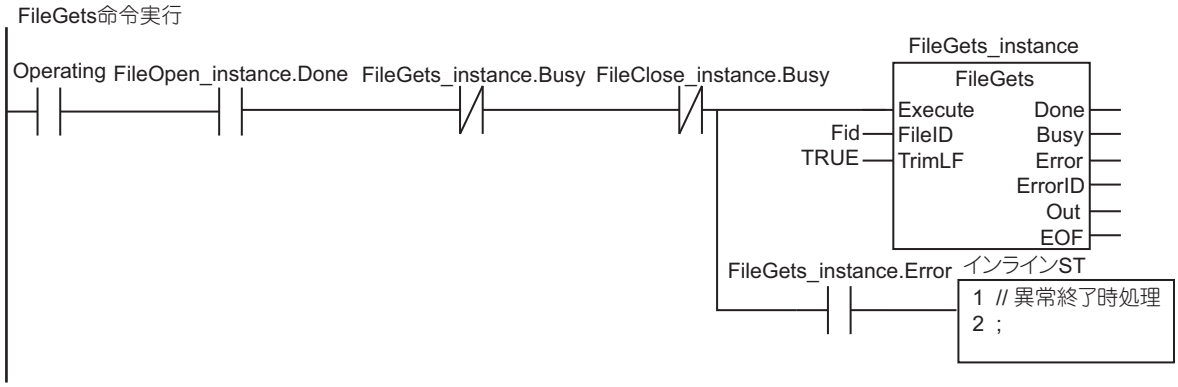
トリガ受付

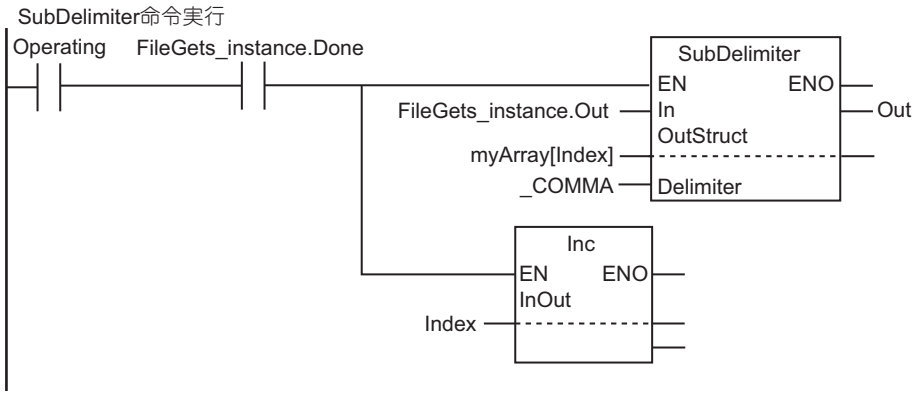


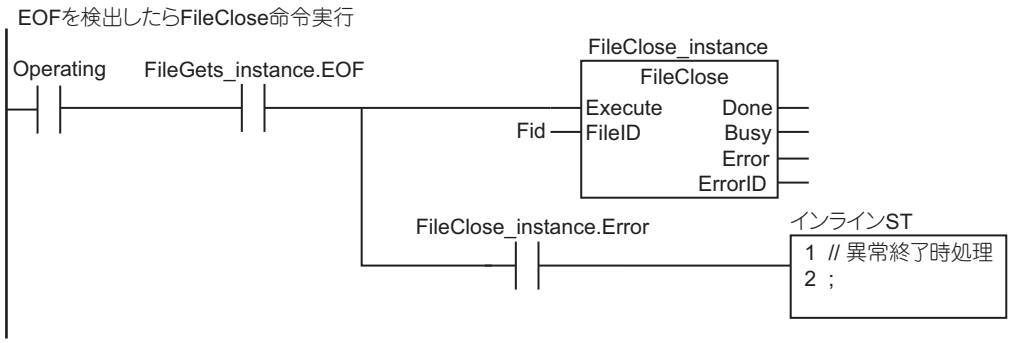
InDat[]の要素インデックス初期化




FileOpen命令実行
Operating
FileOpen_instance
FileOpen
Execute
FileName
Mode
Done
Busy
Error
ErrorID
FileID
'ABC.csv'
_READ_EXIST
Fid
FileOpen_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileGets命令実行
Operating
FileOpen_instance.Done
FileGets_instance.Busy
FileClose_instance.Busy
FileGets_instance
FileGets
Execute
FileID
TrimLF
Done
Busy
Error
ErrorID
Out
EOF
Fid
TRUE
FileGets_instance.Error
インラインST
1 // 異常終了時処理
2 ;


SubDelimiter命令実行
Operating
FileGets_instance.Done
SubDelimiter
EN
In
OutStruct
Delimiter
ENO
Out
FileGets_instance.Out
myArray[Index]
_COMMA
Inc
EN
InOut
ENO
Index


EOFを検出したらFileClose命令実行
Operating
FileGets_instance.EOF
FileClose_instance
FileClose
Execute
FileID
Done
Busy
Error
ErrorID
Fid
FileClose_instance.Error
インラインST
1 // 異常終了時処理
2 ;


正常終了時処理
Operating
FileClose_instance.Done
インラインST
1 // 正常終了時処理
2 ;

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	False	実行条件
	LastTrigger	BOOL	False	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	False	処理開始
	Operating	BOOL	False	処理中
	myArray	ARRAY[0..999] OF myStruct	[1000((member0:='',member1:='',member2:='',member3:='',member4:=''))]	整数データ
	Stage	INT	0	状態遷移
	Index	INT	0	myArray[]の要素インデックス
	Fid	DWORD	16#0	ファイル ID
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
  OperatingStart:=TRUE;
  Operating :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
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;

// 各命令実行
IF (Operating=TRUE) THEN
  CASE Stage OF
  1 : // ファイルオープン
    FileOpen_instance(
      Execute :=TRUE,
      FileName:='ABC.csv', // ファイル名
      Mode :=_READ_EXIST, // ファイル読出
      FileID =>Fid); // ファイル ID
```

```
    IF (FileOpen_instance.Done=TRUE) THEN
      Stage:=INT#2; // 正常終了
    END_IF;

    IF (FileOpen_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  2 : // 文字列読出
    FileGets_instance(
      Execute:=TRUE,
      FileID :=Fid,
      TrimLF :=TRUE);

    IF (FileGets_instance.Done=TRUE) THEN
      // 読み出した文字列を myArray[]のメンバの値として格納
      SubDelimiter(FileGets_instance.Out,myArray[Index],_COMMA);
      Index:=Index+INT#1;

      // ファイル終端に達した
      IF (FileGets_instance.EOF=TRUE) THEN
        Stage:=INT#3; // 正常終了
      ELSE
        FileGets_instance(Execute:=FALSE);
      END_IF;
    END_IF;

    IF (FileGets_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  3 : // ファイルクローズ
    FileClose_instance(
      Execute:=TRUE,
      FileID :=Fid); // ファイル ID

    IF (FileClose_instance.Done=TRUE) THEN
      Operating:=FALSE; // 正常終了
    END_IF;

    IF (FileClose_instance.Error=TRUE) THEN
      Stage:=INT#99; // 異常終了
    END_IF;

  99 : // 異常終了処理
    Operating:=FALSE;
  END_CASE;
END_IF;
```

時間／時刻命令

命令	名称	ページ
ADD_TIME	時間加算	P.2-599
ADD_TOD_TIME	時刻と時間の加算	P.2-601
ADD_DT_TIME	日付時刻と時間の加算	P.2-603
SUB_TIME	時間減算	P.2-605
SUB_TOD_TIME	時刻と時間の減算	P.2-607
SUB_TOD_TOD	時刻減算	P.2-609
SUB_DATE_DATE	日付減算	P.2-610
SUB_DT_DT	日付時刻減算	P.2-611
SUB_DT_TIME	日付時刻と時間の減算	P.2-613
MULTIME	時間乗算	P.2-615
DIVTIME	時間除算	P.2-617
CONCAT_DATE_TOD	日付と時刻の結合	P.2-619
DT_TO_TOD	日付時刻からの時刻切出	P.2-621
DT_TO_DATE	日付時刻からの日付切出	P.2-623
GetTime	時刻取得	P.2-625
DtToSec	日付時刻→秒変換	P.2-627
DateToSec	日付→秒変換	P.2-629
TodToSec	時刻→秒変換	P.2-631
SecToDt	秒→日付時刻変換	P.2-633
SecToDate	秒→日付変換	P.2-635
SecToTod	秒→時刻変換	P.2-637
TimeToNanoSec	時間→ナノ秒変換	P.2-639
TimeToSec	時間→秒変換	P.2-640

命令	名称	ページ
NanoSecToTime	ナノ秒→時間変換	P.2-642
SecToTime	秒→時間変換	P.2-643
ChkLeapYear	うるう年判別	P.2-645
GetDaysOfMonth	月の日数取得	P.2-646
DaysToMonth	日数→月変換	P.2-648
GetDayOfWeek	曜日取得	P.2-650
GetWeekOfYear	週取得	P.2-652
DtToDateStruct	時刻分解	P.2-654
DateStructToDt	時刻結合	P.2-656
TruncTime	時間切り捨て	P.2-658
TruncDt	日付時刻切り捨て	P.2-662
TruncTod	時刻切り捨て	P.2-666

ADD_TIME

時間と時間を加算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_TIME	時間加算	FUN	(@)ADD_TIME EN ENO In1 Out In2	Out:=ADD_TIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被加算時間	入力	被加算時間	データ型に従う	ns	T#0s
In2	加算時間		加算時間			
Out	加算結果時間	出力	加算結果時間	データ型に従う	ns	−

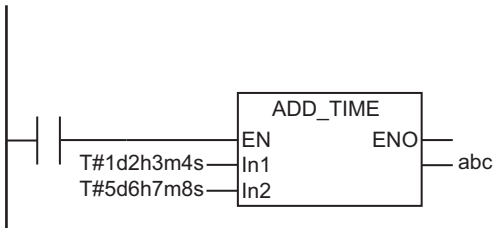
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

機能

時間「In1」と「In2」を、加算します。加算結果「Out」も、時間です。

「In1」=T#1d2h3m4s、「In2」=T#5d6h7m8s の場合の例を、以下に示します。

LD



ST

```
abc:=ADD_TIME(T#1d2h3m4s, T#5d6h7m8s);
```

	In1	T#1d2h3m4s
+	In2	T#5d6h7m8s
	Out=abc	T#6d8h10m12s

使用上の注意

加算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

時刻に時間を加算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_TOD_TIME	時刻と時間の加算	FUN	(@)ADD_TOD_TIME EN In1 In2 ENO Out	Out:=ADD_TOD_TIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被加算時刻	入力	被加算時刻	データ型に従う	時分秒	TOD#0:0:0
In2	加算時間		加算時間		ns	T#0s
Out	加算結果時刻	出力	加算結果時刻	データ型に従う	時分秒	−

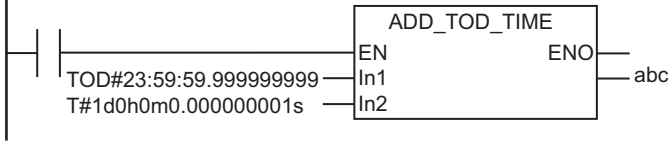
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				
Out																		○		

機能

時刻「In1」に、時間「In2」を加算します。加算結果「Out」は、時刻です。

「In1」=TOD#23:59:59.999999999、「In2」=T#1d0h0m0.000000001s の場合の例を、以下に示します。

LD



ST

```
abc:=ADD_TOD_TIME(TOD#23:59:59.999999999,
                  T#1d0h0m0.000000001s);
```

	In1	TOD#23:59:59.999999999
+	In2	T#1d0h0m0.000000001s
	Out=abc	TOD#0:0:0.000000000

使用上の注意

加算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

日付時刻に時間を加算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_DT_TIME	日付時刻と時間の加算	FUN	(@)ADD_DT_TIME EN　ENO In1　Out In2	Out:=ADD_DT_TIME(In1, In2);

変数

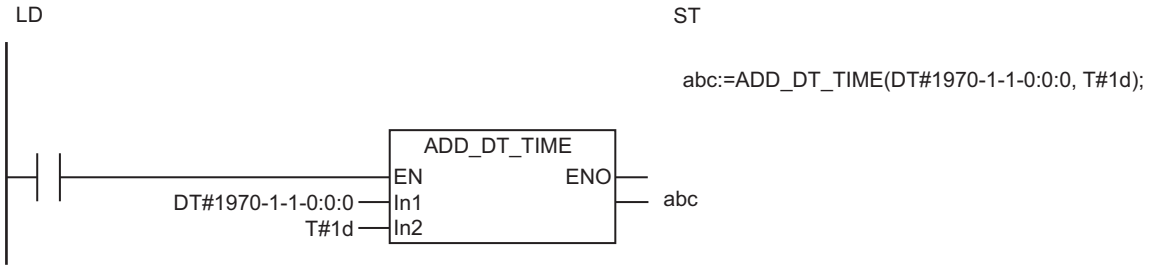
	名称	入／出力	内容	有効範囲	単位	初期値
In1	被加算日付時刻	入力	被加算日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
In2	加算時間		加算時間		ns	T#0s
Out	加算結果日付時刻	出力	加算結果日付時刻	データ型に従う	年月日時分秒	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

機能

日付時刻「In1」に、時間「In2」を加算します。加算結果「Out」は、日付時刻です。うるう年も加味します。

「In1」=DT#1970-1-1-0:0:0、「In2」=T#1d の場合の例を、以下に示します。



	In1	DT#1970-1-1-0:0:0
+	In2	T#1d
	Out=abc	DT#1970-1-2-0:0:0

関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

使用上の注意

加算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

時間から時間を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TIME	時間減算	FUN	(@)SUB_TIME EN ENO In1 Out In2	Out:=SUB_TIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算時間	入力	被減算時間	データ型に従う	ns	T#0s
In2	減算時間		減算時間			
Out	減算結果時間	出力	減算結果時間	データ型に従う	ns	−

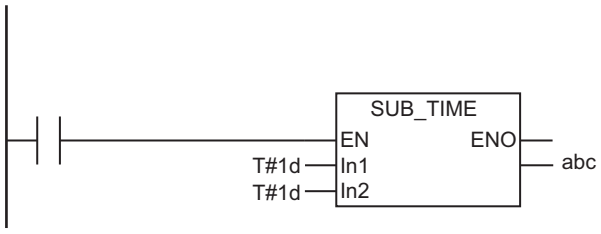
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

機能

時間「In1」から、時間「In2」を減算します。減算結果「Out」も、時間です。

「In1」＝「In2」=T#1d の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_TIME(T#1d, T#1d);
```

	In1	T#1d
-	In2	T#1d
	Out=abc	T#0s

使用上の注意

減算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

時刻から時間を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TOD_TIME	時刻と時間の減算	FUN	(@)SUB_TOD_TIME EN ENO In1 Out In2	Out:=SUB_TOD_TIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算時刻	入力	被減算時刻	データ型に従う	時分秒	TOD#0:0:0
In2	減算時間		減算時間		ns	T#0s
Out	減算結果時刻	出力	減算結果時刻	データ型に従う	時分秒	−

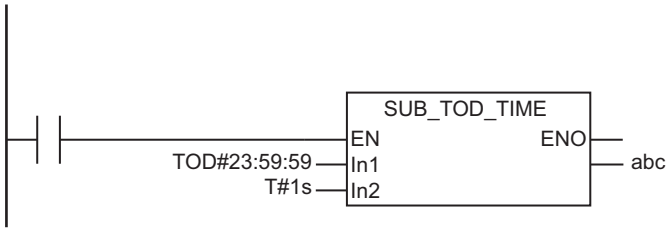
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				
Out																		○		

機能

時刻「In1」から、時間「In2」を減算します。減算結果「Out」は、時刻です。

「In1」=TOD#23:59:59、「In2」=T#1s の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_TOD_TIME(TOD#23:59:59, T#1s);
```

	In1	TOD#23:59:59
-	In2	T#1s
	Out=abc	TOD#23:59:58

使用上の注意

減算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

時刻から時刻を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TOD_TOD	時刻減算	FUN	(@)SUB_TOD_TOD EN ENO In1 Out In2	Out:=SUB_TOD_TOD(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算時刻	入力	被減算時刻	データ型に従う	時分秒	TOD#0:0:0
In2	減算時刻		減算時刻			
Out	減算結果時間	出力	減算結果時間	データ型に従う	ns	−

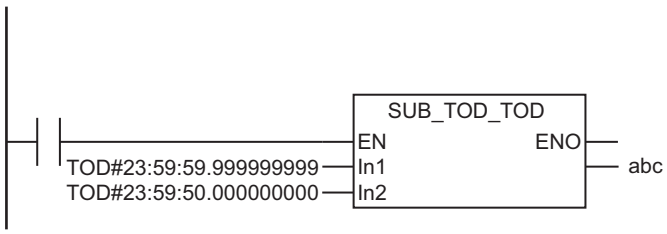
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																		○		
Out																○				

機能

時刻「In1」から、時刻「In2」を減算します。減算結果「Out」は、時間です。

「In1」=TOD#23:59:59.999999999、「In2」=TOD#23:59:50.000000000 の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_TOD_TOD(TOD#23:59:59.999999999,
                 TOD#23:59:50.000000000);
```

	In1	TOD#23:59:59.999999999
-	In2	TOD#23:59:50.000000000
	Out=abc	T#0d0h0m9.999999999s

SUB_DATE_DATE

日付から日付を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DATE_DATE	日付減算	FUN	(@)SUB_DATE_DATE EN / ENO In1 / Out In2	Out:=SUB_DATE_DATE(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算日付	入力	被減算日付	データ型に従う	年月日	D#1970-1-1
In2	減算日付		減算日付			
Out	減算結果時間	出力	減算結果時間	データ型に従う	ns	−

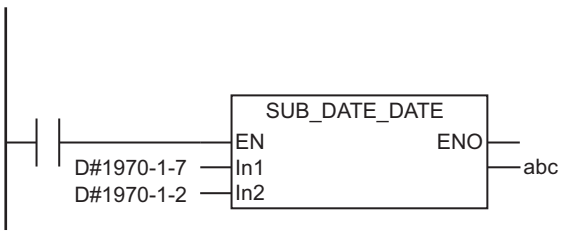
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																	○			
Out																○				

機能

日付「In1」から、日付「In2」を減算します。減算結果「Out」は、時間です。

「In1」=D#1970-1-7、「In2」=D#1970-1-2 の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_DATE_DATE(D#1970-1-7, D#1970-1-2);
```

	In1	D#1970-1-7
-	In2	D#1970-1-2
	Out=abc	T#5d0h0m0.000000000s

SUB_DT_DT

日付時刻から日付時刻を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DT_DT	日付時刻減算	FUN	(@)SUB_DT_DT EN ENO In1 Out In2	Out:=SUB_DT_DT(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算日付時刻	入力	被減算日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
In2	減算日付時刻		減算日付時刻			
Out	減算結果時間	出力	減算結果時間	データ型に従う	ns	−

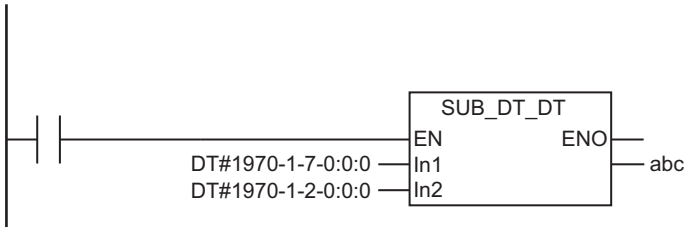
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																			○	
Out																○				

機能

日付時刻「In1」から、日付時刻「In2」を減算します。減算結果「Out」は、時間です。

「In1」=DT#1970-1-7-0:0:0、「In2」=DT#1970-1-2-0:0:0 の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_DT_DT(DT#1970-1-7-0:0:0,
               DT#1970-1-2-0:0:0);
```

	In1	DT#1970-1-7-0:0:0
-	In2	DT#1970-1-2-0:0:0
	Out=abc	T#5d

関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

使用上の注意

演算結果が「Out」の有効範囲外の場合、「Out」の値は不正値になります。

SUB_DT_TIME

日付時刻から時間を減算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DT_TIME	日付時刻と時間の減算	FUN	(@)SUB_DT_TIME EN ENO In1 Out In2	Out:=SUB_DT_TIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被減算日付時刻	入力	被減算日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
In2	減算時間		減算時間		ns	T#0s
Out	減算結果日付時刻	出力	減算結果日付時刻	データ型に従う	年月日時分秒	−

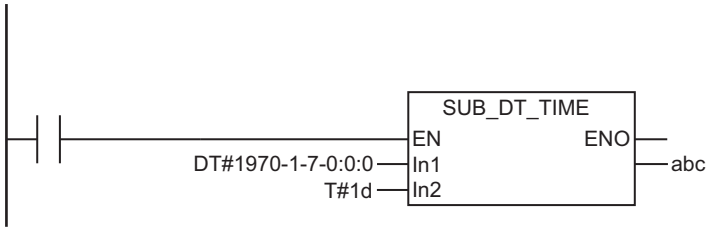
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

機能

日付時刻「In1」から、時間「In2」を減算します。減算結果「Out」は、日付時刻です。うるう年も加味します。

「In1」=DT#1970-1-7-0:0:0、「In2」=T#1d の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_DT_TIME(DT#1970-1-7-0:0:0, T#1d);
```

	In1	DT#1970-1-7-0:0:0
-	In2	T#1d
	Out=abc	DT#1970-1-6-0:0:0

関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

使用上の注意

減算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。

- 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

時間を指定乗数で乗算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
MULTIME	時間乗算	FUN	(@)MULTIME EN ENO In1 Out In2	Out:=MULTIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被乗算時間	入力	被乗算時間	データ型に従う	ns	T#0s
In2	乗数		乗数		−	*1
Out	乗算結果時間	出力	乗算結果時間	データ型に従う	ns	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

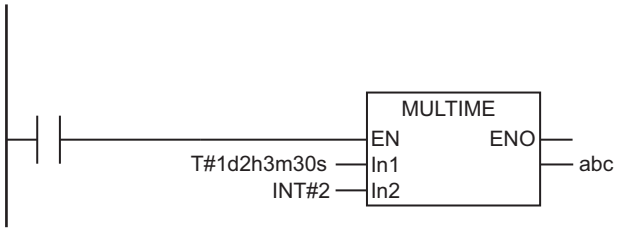
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

機能

時間「In1」と、乗数「In2」を乗算します。乗算結果「Out」は、時間です。

「In1」=T#1d2h3m30s、「In2」=INT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=MULTIME(T#1d2h3m30s, INT#2);
```

	In1	T#1d2h3m30s
×	In2	INT#2
	Out=abc	T#2d4h7m

使用上の注意

- 「In2」が実数の場合、乗算結果の 1ns 未満は五捨五入します。
- 「In2」の値が、0、+∞、-∞、非数の場合、「Out」の値は以下のようになります。

「In2」の値	「Out」の値
0	T#0s
+∞	T#-106751d23h47m16.854775808s
-∞	T#-106751d23h47m16.854775808s
非数	T#-106751d23h47m16.854775808s

- 乗算結果が「Out」の有効範囲を超える場合、異常にはならず、つぎのように動作します。
  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

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

DIVTIME

時間を指定除数で除算します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DIVTIME	時間除算	FUN	(@)DIVTIME EN ENO In1 Out In2	Out:=DIVTIME(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	被除算時間	入力	被除算時間	データ型に従う	ns	T#0s
In2	除数		除数		−	*1
Out	除算結果時間	出力	除算結果時間	データ型に従う	ns	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

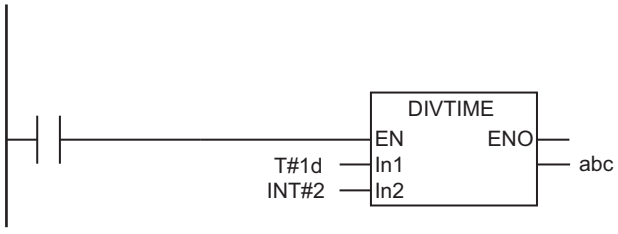
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

機能

時間「In1」を、除数「In2」で除算します。除算結果「Out」は、時間です。

「In1」=T#1d、「In2」=INT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=DIVTIME(T#1d, INT#2);
```

	In1	T#1d
/	In2	INT#2
	Out=abc	T#12h

使用上の注意

・「In2」の値が、0、+∞、-∞、非数の場合、「Out」の値は以下のようになります。

「In2」の値	「Out」の値
0	T#-106751d23h47m16.854775808s
+∞	T#0s
-∞	T#0s
非数	T#-106751d23h47m16.854775808s

・「In2」が実数の場合、数 ns の誤差が発生する場合があります。
・「In2」が実数の場合、除算結果の 1ns 未満は五捨五入します。
・ つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
a)「In2」が整数型で、値が 0 の場合。

五捨五入

五捨五入とは、下の表のような処理のことです。

小数部の値	処理	例
0.5 より小	切り捨て	1.49→1
0.5	1 の位の値が偶数なら切り捨て、奇数なら切り上げ	1.50→2 2.50→2
0.5 より大	切り上げ	1.51→2

CONCAT_DATE_TOD

日付と時刻を結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CONCAT_DATE_TOD	日付と時刻の結合	FUN	(@)CONCAT_DATE_TOD EN In1 In2 ENO Out	Out:=CONCAT_DATE_TOD(In1, In2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In1	日付	入力	日付	データ型に従う	年月日	D#1970-1-1
In2	時刻		時刻		時分秒	TOD#0:0:0
Out	結合結果日付時刻	出力	結合結果日付時刻	データ型に従う	年月日時分秒	－

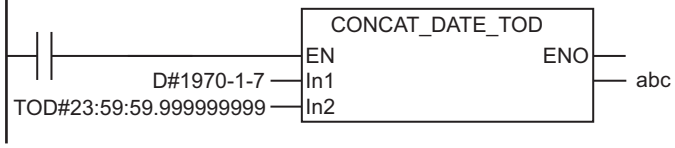
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																		○		
Out																			○	

機能

日付「In1」と、時刻「In2」を結合します。結合結果「Out」は、日付時刻です。

「In1」=D#1970-1-7、「In2」=TOD#23:59:59.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=CONCAT_DATE_TOD(D#1970-1-7,
                     TOD#23:59:59.999999999);
```

	In1	D#1970-1-7
+	In2	TOD#23:59:59.999999999
	Out=abc	DT#1970-1-7-23:59:59.999999999

関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 結合結果が、「Out」の有効範囲外の場合。
  「Out」が有効範囲外とは、「In1」の値が D#2554-7-21 で、「In2」の値が TOD#23:34:33.709551615 より大きい場合のことです。

DT_TO_TOD

日付時刻から時刻を切り出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DT_TO_TOD	日付時刻からの時刻切出	FUN	(@)DT_TO_TOD EN ENO In Out	Out:=DT_TO_TOD(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付時刻	入力	日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Out	時刻	出力	時刻	データ型に従う	時分秒	－

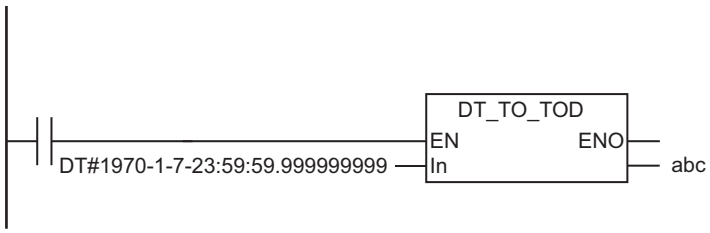
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																		○		

機能

日付時刻「In」から、時刻の部分だけを切り出します。

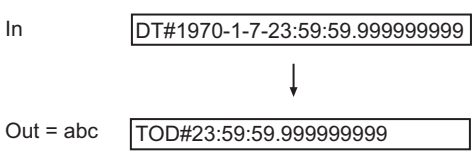
「In」=DT#1970-1-7-23:59:59.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DT_TO_TOD(
      DT#1970-1-7-23:59:59.999999999);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

DT_TO_DATE

日付時刻から日付を切り出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DT_TO_DATE	日付時刻からの日付切出	FUN	(@)DT_TO_DATE EN ENO In Out	Out:=DT_TO_DATE(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付時刻	入力	日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Out	日付	出力	日付	データ型に従う	年月日	−

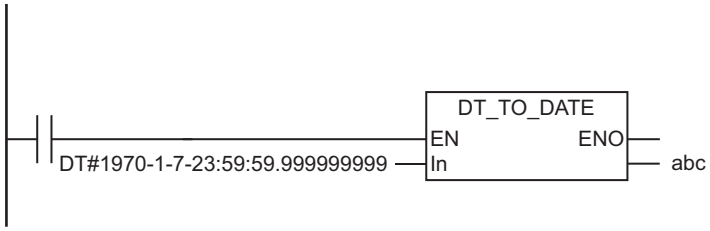
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																	○			

機能

日付時刻「In」から、日付の部分だけを切り出します。

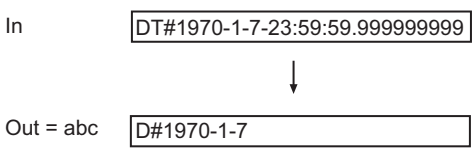
「In」=DT#1970-1-7-23:59:59.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DT_TO_DATE(
      DT#1970-1-7-23:59:59.999999999);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

GetTime

現在時刻を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetTime	時刻取得	FUN	(@)GetTime EN ENO Out	Out:=GetTime();

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	現在時刻	出力	現在時刻	*1	年月日時分秒	−

*1. 有効範囲は、GMT（グリニッジ標準時）です。
NY シリーズの有効範囲は、DT#2000-01-01-00:00:00.000000000 ～ DT#2099-12-31-23:59:59.999999999（2000 年 1 月 1 日 0 時 0 分 0.000000000 秒 ～ 2099 年 12 月 31 日 23 時 59 分 59.999999999 秒）となります。

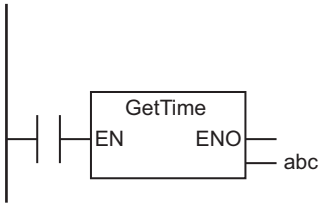
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																			○	

機能

現在時刻を読み出します。
現在時刻は、GMT（グリニッジ標準時）ではなく、設定されているタイムゾーンの標準時です。

記述例を、以下に示します。変数「abc」に、現在の時刻が代入されます。

LD



ST

```
abc:=GetTime();
```

現在時刻を読み出し、abcに代入します。

1970年1月7日23時59分59.999999999秒の場合



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

参考

- 現在時刻をシステム時刻（1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数）に変換する場合は、「DtToSec (P.2-627)」命令を使用してください。
- 現在時刻を年月日時分秒に分解する場合は、「DtToDateStruct (P.2-654)」命令を使用してください。
- 曜日を読み出す場合は、「GetDayOfWeek (P.2-650)」命令を使用してください。

DtToSec

日付時刻を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DtToSec	日付時刻→秒変換	FUN	(@)DtToSec EN ENO In Out	Out:=DtToSec(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付時刻	入力	日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Out	秒	出力	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数	0 ～ 18446744073	秒	−

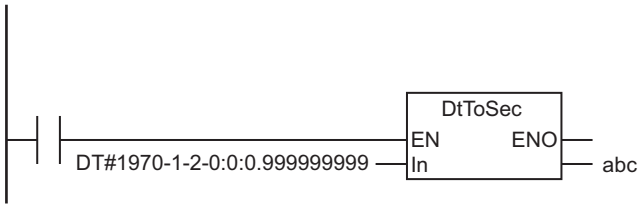
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out													○							

機能

日付時刻「In」を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。変換後の値の単位は秒です。1 秒未満の値は切り捨てます。

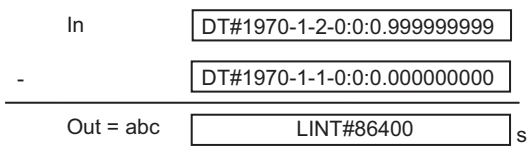
「In」=DT#1970-1-2-0:0:0.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DtToSec(DT#1970-1-2-0:0:0.999999999);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

参考

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付時刻に変換する場合は、「SecToDt (P.2-633)」命令を使用してください。

DateToSec

日付を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DateToSec	日付→秒変換	FUN	(@)DateToSec EN ENO In Out	Out:=DateToSec(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付	入力	日付	データ型に従う	年月日	D#1970-1-1
Out	秒	出力	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数	0～18446659200	秒	－

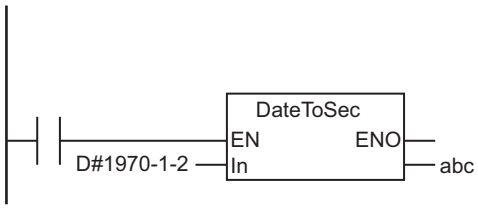
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○			
Out													○							

機能

日付「In」の 0 時 0 分 0 秒の時刻を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。変換後の値の単位は秒です。

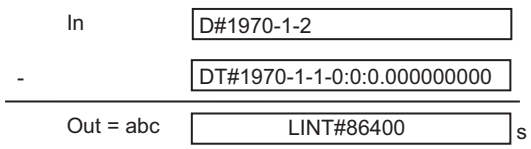
「In」=D#1970-1-2 の場合の例を、以下に示します。

LD



ST

```
abc:=DateToSec(D#1970-1-2);
```



参考

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付に変換する場合は、「SecToDate (P.2-635)」命令を使用してください。

TodToSec

時刻を、0 時 0 分 0 秒からの秒数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TodToSec	時刻→秒変換	FUN	(@)TodToSec EN ENO In Out	Out:=TodToSec(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	時刻	入力	時刻	データ型に従う	時分秒	TOD#0:0:0
Out	秒	出力	0 時 0 分 0 秒からの秒数	0 ～ 86399	秒	－

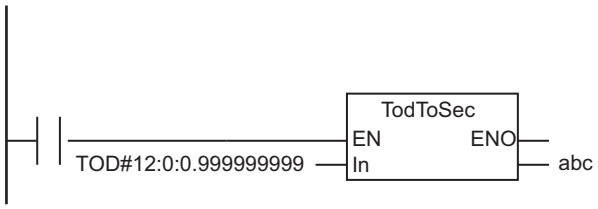
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out													○							

機能

時刻「In」を、0 時 0 分 0 秒からの秒数に変換します。変換後の値の単位は秒です。1 秒未満の値は切り捨てます。

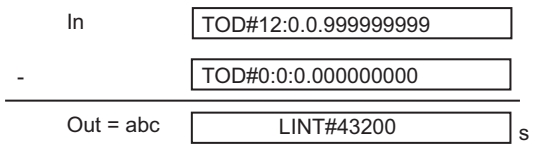
「In」=TOD#12:0:0.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=TodToSec(TOD#12:0:0.999999999);
```



参考

0 時 0 分 0 秒からの秒数を、時刻に変換する場合は、「SecToTod (P.2-637)」命令を使用してください。

SecToDt

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付時刻に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SecToDt	秒→日付時刻変換	FUN	(@)SecToDt EN ENO In Out	Out:=SecToDt(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	秒	入力	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数	0 ～ 18446744073	秒	0
Out	日付時刻	出力	日付時刻	データ型に従う	年月日時分秒	－

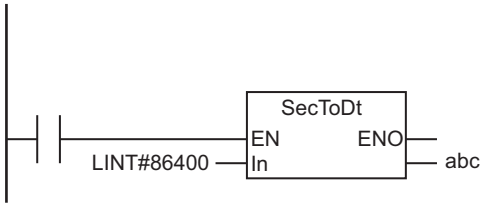
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																			○	

機能

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数「In」を、日付時刻に変換します。

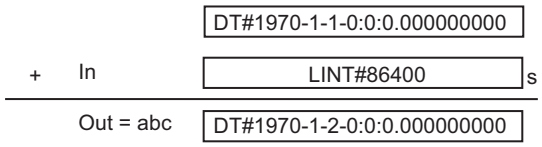
「In」=LINT#86400 の場合の例を、以下に示します。

LD



ST

```
abc:=SecToDt(LINT#86400);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

参考

日付時刻を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換する場合は、「DtToSec (P.2-627)」命令を使用してください。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
・「In」の値が、有効範囲外の場合。

SecToDate

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SecToDate	秒→日付変換	FUN	(@)SecToDate EN ENO In Out	Out:=SecToDate(In);

変数

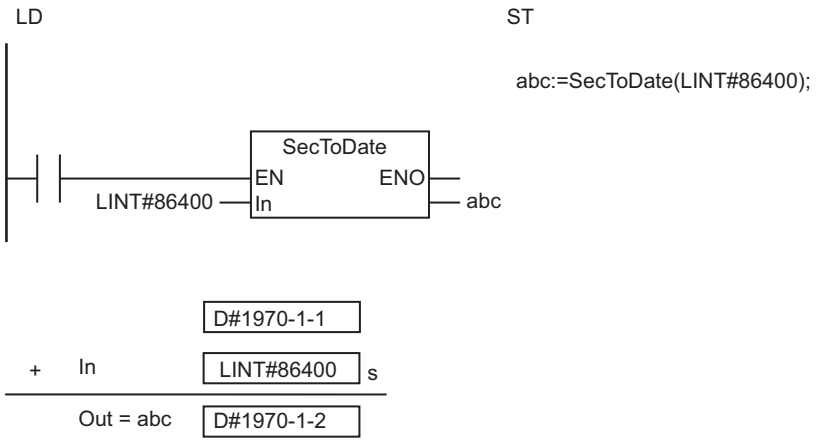
	名称	入／出力	内容	有効範囲	単位	初期値
In	秒	入力	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数	0 ～ 18446744073	秒	0
Out	日付	出力	日付	データ型に従う	年月日	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																	○			

機能

1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数「In」を、日付に変換します。1 日未満の値は、切り捨てます。

「In」=LINT#86400 の場合の例を、以下に示します。



参考

日付を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換する場合は、「DateToSec (P.2-629)」命令を使用してください。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
・「In」の値が、有効範囲外の場合。

SecToTod

0 時 0 分 0 秒からの秒数を、時刻に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SecToTod	秒→時刻変換	FUN	(@)SecToTod EN ENO In Out	Out:=SecToTod(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	秒	入力	0 時 0 分 0 秒からの秒数	データ型に従う *1	秒	0
Out	時刻	出力	時刻	データ型に従う	時分秒	−

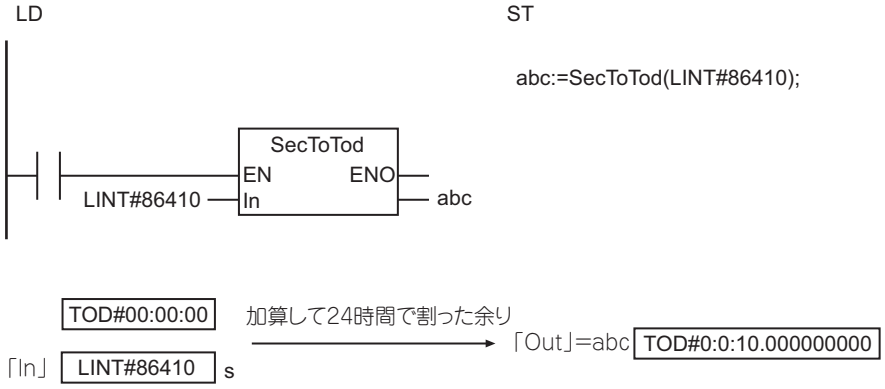
*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																		○		

機能

0 時 0 分 0 秒からの秒数「In」を、時刻に変換します。「In」の値が 24 時間以上の場合は、「In」の値を 24 時間で割った余りを時刻に変換します。

「In」=LINT#86410 の場合の例を、以下に示します。



参考

時刻を、0 時 0 分 0 秒からの秒数に変換する場合は、「TodToSec (P.2-631)」命令を使用してください。

使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
・「In」の値が、有効範囲外の場合。

TimeToNanoSec

時間を、ナノ秒数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeToNanoSec	時間→ナノ秒変換	FUN	(@)TimeToNanoSec EN ENO In Out	Out:=TimeToNanoSec(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	時間	入力	時間	データ型に従う	ns	T#0s
Out	秒数	出力	秒数	*1	ns	−

*1. -9223372036854775808 ～ 9223372036854775807

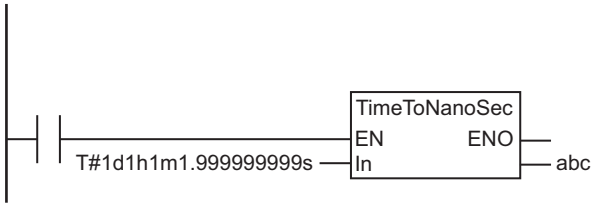
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

機能

時間「In」を、ナノ秒数に変換します。

「In」=T#1d1h1m1.999999999s の場合の例を、以下に示します。

LD



ST

```
abc:=TimeToNanoSec(T#1d1h1m1.999999999s);
```

ナノ秒数

「In」 T#1d1h1m1.999999999s → 「Out」=abc LINT#90061000000000 ns

参考

ナノ秒を、時間に変換する場合は、「NanoSecToTime (P.2-642)」命令を使用してください。

TimeToSec

時間を、秒数に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeToSec	時間→秒変換	FUN	(@)TimeToSec EN ENO In Out	Out:=TimeToSec(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	時間	入力	時間	データ型に従う	ns	T#0s
Out	秒数	出力	秒数	-9223372036 ～ 9223372036	秒	－

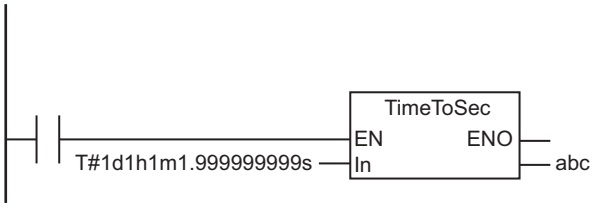
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

機能

時間「In」を、秒数に変換します。1 秒未満の値は切り捨てます。

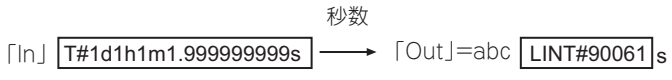
「In」=T#1d1h1m1.999999999s の場合の例を、以下に示します。

LD



ST

```
abc:=TimeToSec(T#1d1h1m1.999999999s);
```



参考

秒を、時間に変換する場合は、「SecToTime (P.2-643)」命令を使用してください。

使用上の注意

「In」の単位はナノ秒、「Out」の単位は秒です。

NanoSecToTime

ナノ秒数を、時間に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NanoSecToTime	ナノ秒→時間変換	FUN	(@)NanoSecToTime EN ENO In Out	Out:=NanoSecToTime(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	秒数	入力	秒数	*1	ns	0
Out	時間	出力	時間	データ型に従う	ns	−

*1. -9223372036854775808 ～ 9223372036854775807

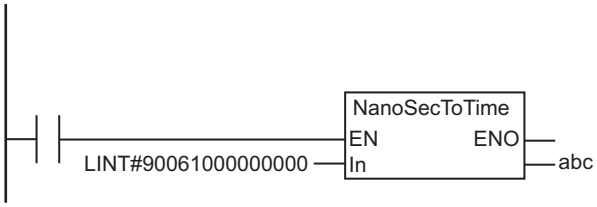
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

機能

ナノ秒数「In」を、時間に変換します。

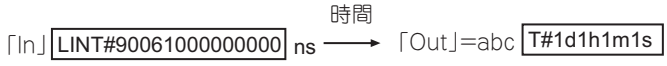
「In」=LINT#90061000000000 の場合の例を、以下に示します。

LD



ST

```
abc:=NanoSecToTime(LINT#90061000000000);
```



参考

時間を、ナノ秒に変換する場合は、「TimeToNanoSec (P.2-639)」命令を使用してください。

SecToTime

秒数を、時間に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SecToTime	秒→時間変換	FUN	(@)SecToTime EN ENO In Out	Out:=SecToTime(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	秒数	入力	秒数	-9223372036 ～ 9223372036	秒	0
Out	時間	出力	時間	データ型に従う	ns	−

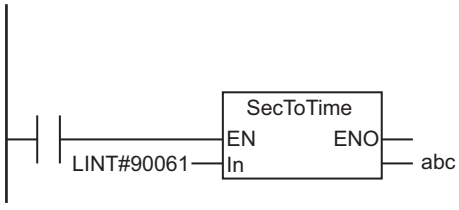
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

機能

秒数「In」を、時間に変換します。

「In」=LINT#90061 の場合の例を、以下に示します。

LD



ST

```
abc:=SecToTime(LINT#90061);
```

「In」LINT#90061 s ——時間——▶ 「Out」=abc T#1d1h1m1s

参考

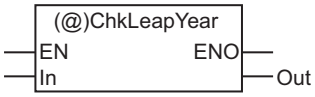
時間を、秒に変換する場合は、「TimeToSec (P.2-640)」命令を使用してください。

使用上の注意

- 「In」の単位は秒、「Out」の単位はナノ秒です。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「In」の値が、有効範囲外の場合。

ChkLeapYear

指定された年が、うるう年かどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ChkLeapYear	うるう年判別	FUN		Out:=ChkLeapYear(In);

変数

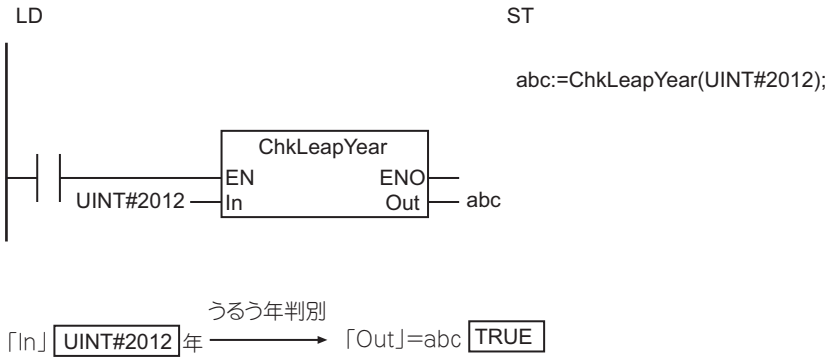
	名称	入／出力	内容	有効範囲	単位	初期値
In	西暦年	入力	西暦年	1970 ～ 2554	年	1970
Out	判定結果	出力	TRUE: うるう年である FALSE: うるう年ではない	データ型に従う	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In							○													
Out	○																			

機能

西暦年「In」が、うるう年かどうかを判定します。うるう年なら、判定結果「Out」の値が TRUE になります。うるう年でなければ「Out」は FALSE になります。

「In」=UINT#2012 の場合の例を、以下に示します。



使用上の注意

「In」の値が有効範囲外の場合、異常にはならず、「Out」の値は不正値になります。

GetDaysOfMonth

指定された月の、日数を取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetDaysOfMonth	月の日数取得	FUN	(@)GetDaysOfMonth EN ENO Year Out Month	Out:=GetDaysOfMonth(Year, Month);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Year	年	入力	西暦年	1970 ～ 2554	年	1970
Month	月		月	1 ～ 12	月	1
Out	日数	出力	日数	28 ～ 31	日	－

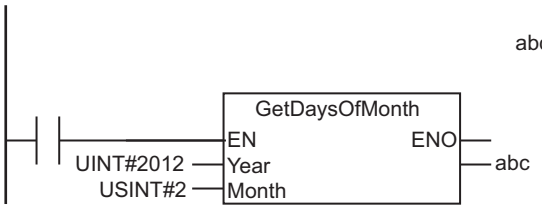
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Month						○														
Out						○														

機能

「Year」で示された年の、「Month」で示された月の、日数を取得します。

「Year」=UINT#2012、「Month」=USINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=GetDaysOfMonth(UINT#2012, USINT#2);
```

「Year」 UINT#2012 年
「Month」 USINT#2 月
→ 日数 → 「Out」=abc USINT#29 日

使用上の注意

- 「Year」の値が有効範囲外の場合、異常にはならず、「Out」の値は不正値になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Month」の値が、有効範囲外の場合。

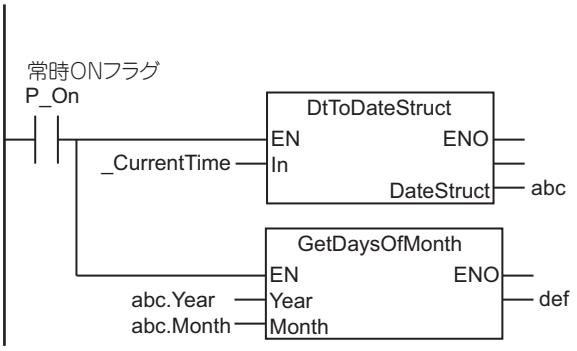
サンプルプログラム

今月の日数を取得します。

LD

内部変数	名称	データ型	初期値	コメント
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	日付時刻
	def	USINT	0	今月の日数

外部変数	名称	データ型	コンスタント	コメント
	_CurrentTime	DATE_AND_TIME	☑	システム時刻



ST

内部変数	名称	データ型	初期値	コメント
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	日付時刻
	def	USINT	0	今月の日数

外部変数	名称	データ型	コンスタント	コメント
	_CurrentTime	DATE_AND_TIME	☑	システム時刻

```
DtToDateStruct(_CurrentTime, abc);
def:=GetDaysOfMonth(abc.Year, abc.Month);
```

DaysToMonth

1 月 1 日からの日数から、その日が何月かを算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DaysToMonth	日数→月変換	FUN	(@)DaysToMonth EN ENO Year Out Days	Out:=DaysToMonth(Year, Days);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Year	年	入力	西暦年	1970 ～ 2554	年	1970
Days	日数		1 月 1 日からの日数	1 ～ 365 「Year」がうるう年の場合は、1 ～ 366	日	1
Out	月	出力	月	1 ～ 12	月	−

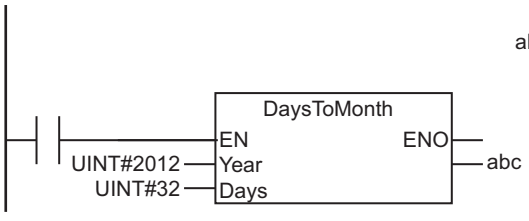
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Days							○													
Out						○														

機能

年「Year」の 1 月 1 日からの日数「Days」から、その日が何月なのかを算出します。

「Year」=UINT#2012、「Days」=UINT#32 の場合の例を、以下に示します。

LD



ST

```
abc:=DaysToMonth(UINT#2012, UINT#32);
```

「Year」 UINT#2012 年
「Days」 UINT#32 日め

月

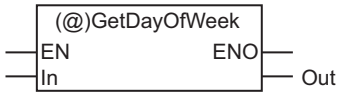
→ 「Out」=abc USINT#2 月

使用上の注意

- 「Year」の値が有効範囲外の場合、異常にはならず、「Out」の値は不正値になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Days」の値が、有効範囲外の場合。

GetDayOfWeek

指定された年月日の、曜日を取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetDayOfWeek	曜日取得	FUN		Out:=GetDayOfWeek(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	年月日	入力	年月日	データ型に従う	年月日	*1
Out	曜日	出力	曜日	_MON, _TUE, _WED, _THU, _FRI, _SAT, _SUN	曜日	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out	列挙型_eDAYOFWEEK　列挙子は「機能 (P.2-650)」参照																			

機能

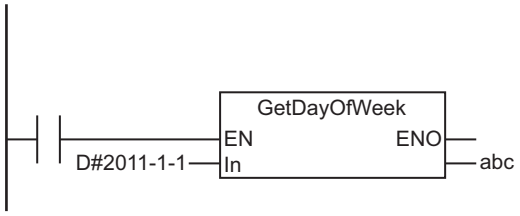
年月日「In」で示された年月日の、曜日を取得します。

「Out」のデータ型は、列挙型_eDAYOFWEEK です。列挙子の意味は、以下のとおりです。

列挙子	意味
_MON	月曜日
_TUE	火曜日
_WED	水曜日
_THU	木曜日
_FRI	金曜日
_SAT	土曜日
_SUN	日曜日

「In」=D#2011-1-1 の場合の例を、以下に示します。

LD



ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

GetWeekOfYear

指定された年月日が、その年の第何週であるかを算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetWeekOfYear	週取得	FUN	(@)GetWeekOfYear EN ENO In Out	Out:=GetWeekOfYear(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	年月日	入力	年月日	データ型に従う	年月日	*1
Out	週	出力	その年の第何週か	1 ～ 54	週	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

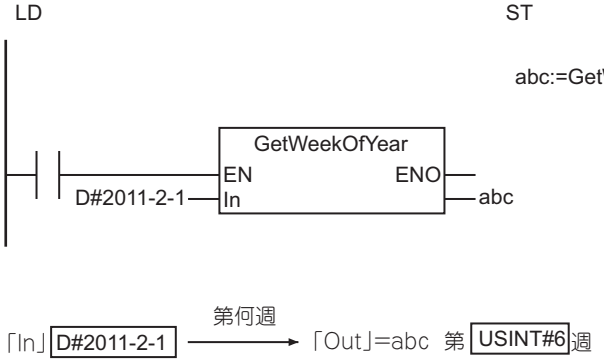
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out						○														

機能

年月日「In」で示された年月日が、その年の第何週であるかを算出します。
月曜日から日曜日までを 1 週間の単位とし、日曜日から月曜日に変わるタイミングで週数をカウントアップします。

1 月 1 日は、必ず第 1 週となります。
たとえば、1 月 1 日が木曜日とすると、1 月 1 日（木）から 1 月 4 日（日）が第 1 週となり、1 月 5 日（月）から 1 月 11 日（日）が第 2 週となります。

「In」=D#2011-2-1 の場合の例を、以下に示します。

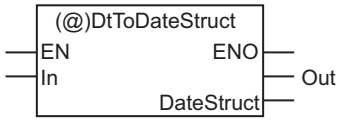


関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

DtToDateStruct

日付時刻を、年、月、日、時、分、秒、ナノ秒に分解します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DtToDateStruct	時刻分解	FUN		Out:=DtToDateStruct(In, DateStruct);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付時刻	入力	日付時刻	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－
DateStruct	日付時刻		年、月、日、時、分、秒、ナノ秒に分解された日付時刻	－		

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out	○																			
DateStruct	構造体_sDT　詳細は「機能 (P.2-654)」参照																			

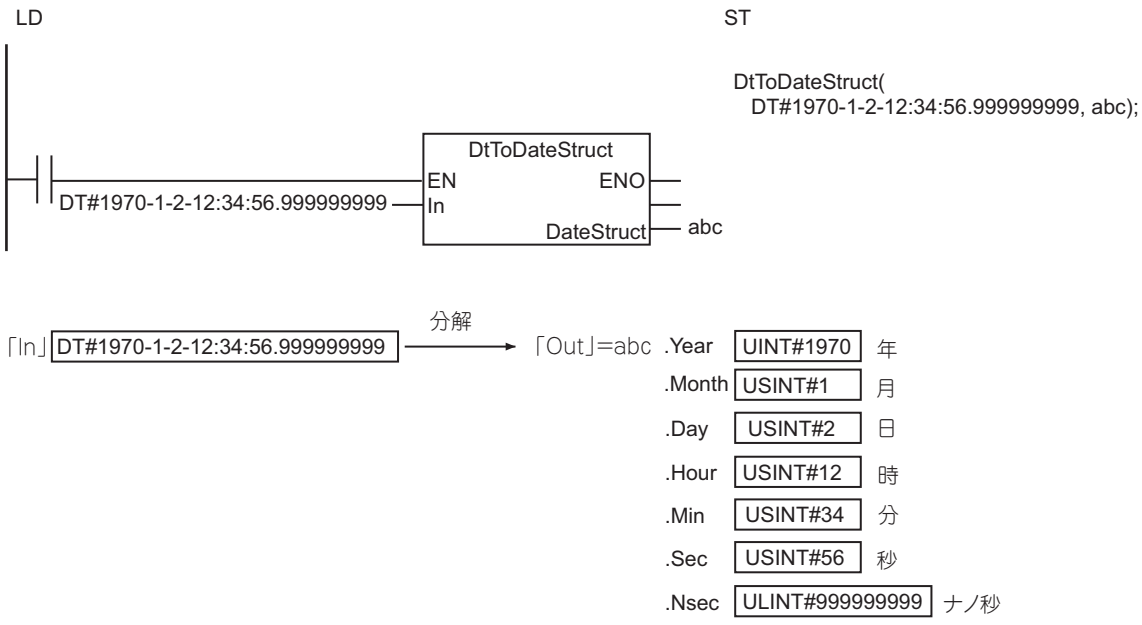
機能

日付時刻「In」を、年、月、日、時、分、秒、ナノ秒に分解します。

分解した日付時刻「DateStruct」のデータ型は、構造体_sDT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DateStruct	日付時刻	年、月、日、時、分、秒、ナノ秒に分解された日付時刻	_sDT	－	－	－
Year	年	年	UINT	1970 ～ 2554	年	－
Month	月	月	USINT	1 ～ 12	月	
Day	日	日	USINT	1 ～ 31	日	
Hour	時	時	USINT	0 ～ 23	時	
Min	分	分	USINT	0 ～ 59	分	
Sec	秒	秒	USINT	0 ～ 59	秒	
Nsec	ナノ秒	ナノ秒	ULINT	0 ～ 999999999	ナノ秒	

「In」=DT#1970-1-2-12:34:56.999999999 の場合の例を、以下に示します。

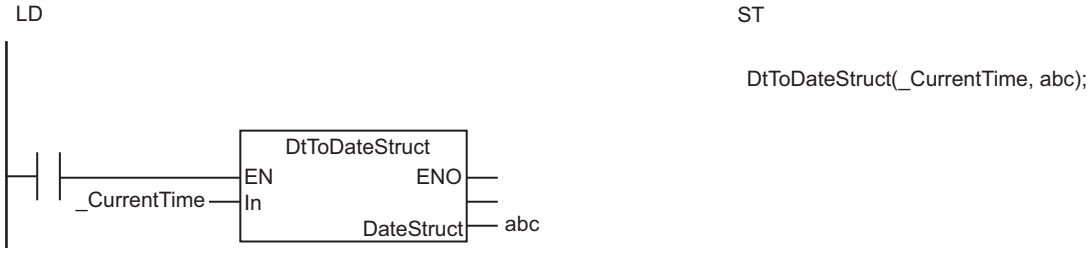


関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

参考

- 分解された年、月、日、時、分、秒、ナノ秒を結合して日付時間にする場合は、「DateStructToDt (P.2-656)」命令を使用してください。
- 現在時刻を求める例を、以下に示します。



使用上の注意

本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。

DateStructToDt

年、月、日、時、分、秒、ナノ秒に分解された日付時刻を、結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DateStructToDt	時刻結合	FUN	(@)DateStructToDt EN ENO In Out	Out:=DateStructToDt(In);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	日付時刻	入力	年、月、日、時、分、秒、ナノ秒に分解された日付時刻	−	−	−
Out	日付時刻	出力	日付時刻	データ型に従う	年月日時分秒	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	構造体_sDT　詳細は「機能 (P.2-656)」参照																			
Out																			○	

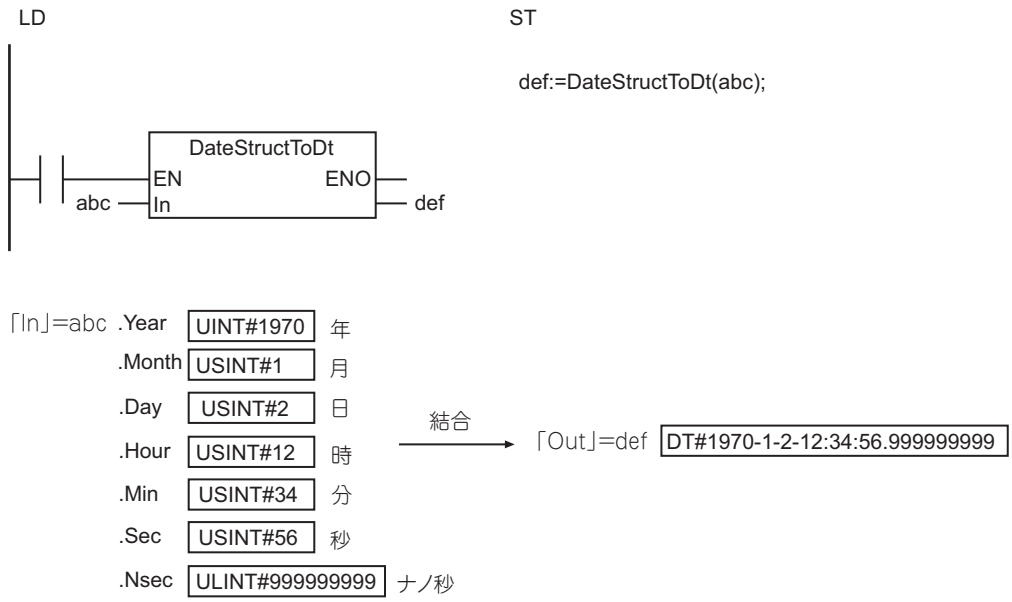
機能

年、月、日、時、分、秒、ナノ秒に分解された日付時刻「In」を、結合します。

「In」のデータ型は、構造体_sDT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
In	日付時刻	年、月、日、時、分、秒、ナノ秒に分解された日付時刻	_sDT	−	−	−
Year	年	年	UINT	1970 ～ 2554	年	1970
Month	月	月	USINT	1 ～ 12	月	1
Day	日	日	USINT	1 ～ 31	日	
Hour	時	時	USINT	0 ～ 23	時	0
Min	分	分	USINT	0 ～ 59	分	
Sec	秒	秒	USINT	0 ～ 59	秒	
Nsec	ナノ秒	ナノ秒	ULINT	0 ～ 999999999	ナノ秒	

「In」の各メンバの値が、「Year」=UINT#1970、「Month」=USINT#1、「Day」=USINT#2、「Hour」USINT#=12、「Min」=USINT#34、「Sec」=USINT#56、「Nsec」=ULINT#999999999 の場合の例を、以下に示します。



関連するシステム定義変数

変数名	名称	データ型	内容
_CurrentTime	システム時刻	DT	システムの内蔵時計の時刻。1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数。

参考

日付時刻を、年、月、日、時、分、秒、ナノ秒に分解する場合は、「DtToDateStruct (P.2-654)」命令を使用してください。

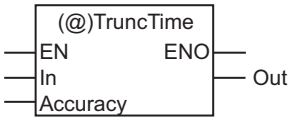
使用上の注意

つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。

- 「In」のメンバのいずれかの値が、有効範囲外の場合。
- 処理結果が、「Out」の有効範囲外の場合。

TruncTime

TIME 型変数の、指定した単位未満の値を切り捨てます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TruncTime	時間切り捨て	FUN		Out:=TruncTime(In, Accuracy);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象時間	入力	時間切り捨ての対象	データ型に従う	ns	T#0s
Accuracy	切り捨て単位		切り捨てを行わない最小の時間単位	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	－	_NANOSEC
Out	切り捨て後時間	出力	切り捨て後の時間	データ型に従う	ns	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Accuracy	列挙型_eSUBSEC　列挙子は「機能 (P.2-658)」参照																			
Out																○				

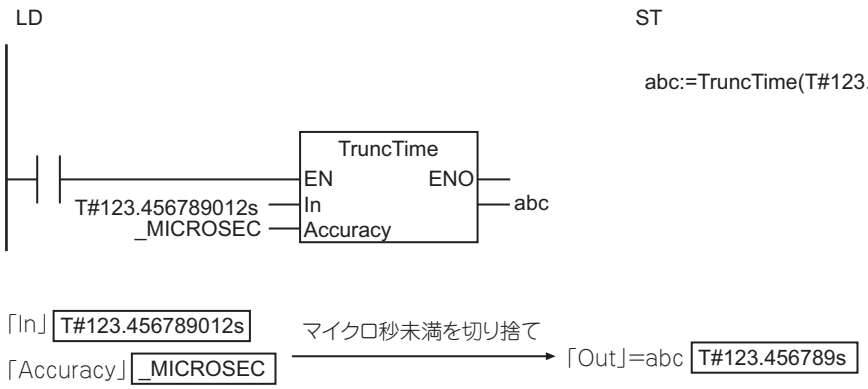
機能

対象時間「In」の、切り捨て単位「Accuracy」未満の値を、切り捨てます。切り捨て後の値は、切り捨て後時間「Out」に格納します。

「Accuracy」のデータ型は、列挙型_eSUBSEC です。列挙子の意味は、以下のとおりです。

列挙子	意味
_NANOSEC	ナノ秒
_MICROSEC	マイクロ秒
_MILLISEC	ミリ秒
_SEC	秒

「In」=TIME#123.456789012s、「Accuracy」=_MICROSEC の場合の例を、以下に示します。



参考

時間どうしを、「EQ, = (P.2-96)」命令などを使って大小比較する際に、本命令を用いて両者の精度を合わせてください。

サンプルプログラム

センサ出力の ON 時間が、しきい値以上かどうかを確認するサンプルプログラムです。
動作モードには、しきい値設定モードと実行モードがあり、それぞれ以下の動作をします。

動作モード	動作
しきい値設定	センサ出力の ON 時間を計測し、その値をしきい値として設定します。
実行	センサ出力の ON 時間を計測し、その値としきい値の比較をします。ON 時間がしきい値以上ならば、正常と判断します。

時間の比較は、ms 単位の精度で行います。そのため、TruncTime 命令を使って、計測時間の ms 未満の値を切り捨てます。
現在の動作モードは、変数「RecentMode」に格納されているとします。判断結果は、変数「Result」に格納します。
「Result」の値は、TRUE が正常、FALSE が異常を意味します。

グローバル変数の定義

● データ型　列挙型

名称	列挙値	コメント
Mode		動作モード
SET	0	しきい値設定
EXEC	1	実行

● グローバル変数

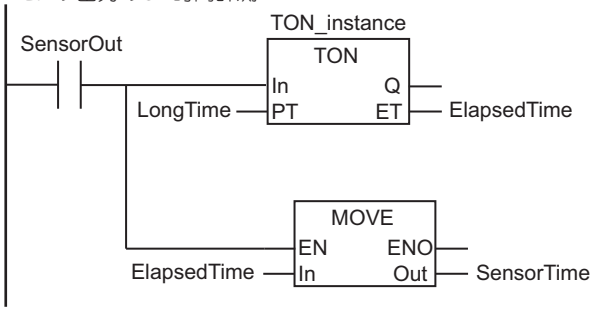
名称	データ型	初期値	コメント
RecentMode	Mode	SET	現在の動作モード

LD

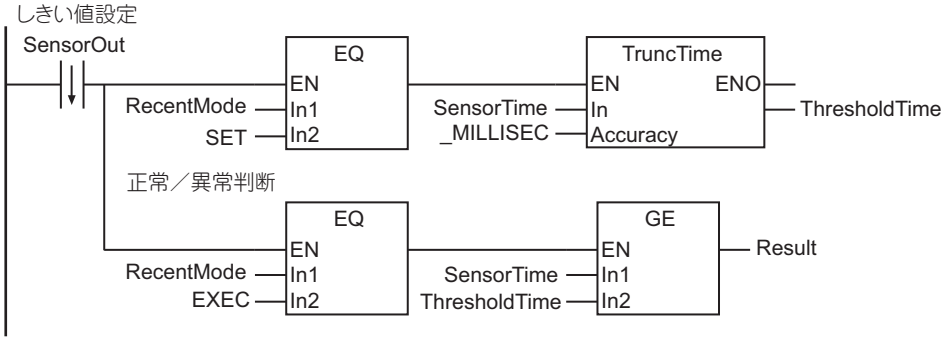
内部変数	名称	データ型	初期値	コメント
	SensorOut	BOOL	FALSE	センサ出力
	ElapsedTime	TIME	T#0s	経過時間
	SensorTime	TIME	T#0s	センサの ON 時間
	LongTime	TIME	T#1h	センサの ON 時間より十分に長い時間
	ThresholdTime	TIME	T#0s	しきい値
	Result	BOOL	FALSE	判断結果　TRUE: 正常　FALSE: 異常
	TON_instance	TON		

外部変数	名称	データ型	コメント
	RecentMode	Mode	現在の動作モード

センサ出力のON時間計測



しきい値設定



ST

内部変数	名称	データ型	初期値	コメント
	SensorOut	BOOL	FALSE	センサ出力
	ElapsedTime	TIME	T#0s	経過時間
	SensorTime	TIME	T#0s	センサの ON 時間
	LongTime	TIME	T#1h	センサの ON 時間より十分に長い時間
	SensorDone	BOOL	FALSE	センサ出力 OFF フラグ
	ThresholdTime	TIME	T#0s	しきい値
	Result	BOOL	FALSE	判断結果　TRUE: 正常　FALSE: 異常
	TON_instance	TON		
	F_TRIG_instance	F_TRIG		

外部変数	名称	データ型	コメント
	RecentMode	Mode	現在の動作モード

```
// TON 命令実行
TON_instance(
  In:=SensorOut, // タイマ入力
  PT:=LongTime, // 設定時間
  ET=>ElapsedTime); // 経過時間

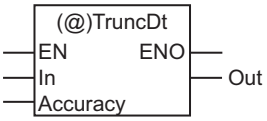
// TON の経過時間をセンサの ON 時間とする
IF (SensorOut=TRUE) THEN
  SensorTime:=ElapsedTime;
END_IF;

// センサ出力の立ち下がり検知
F_TRIG_instance(Clk:=SensorOut, Q=>SensorDone);
Result:=FALSE;

//  しきい値設定
IF (SensorDone=TRUE AND RecentMode=SET) THEN
  ThresholdTime:=TruncTime(
    In :=SensorTime,
    Accuracy:=_MILLISEC); //精度 ms 単位
// 正常/異常判断
ELSIF (SensorDone=TRUE AND RecentMode=EXEC) THEN
  IF (SensorTime >= ThresholdTime) THEN
    Result:=TRUE;
  END_IF;
END_IF;
```

TruncDt

DT 型変数の、指定した単位未満の値を切り捨てます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TruncDt	日付時刻切り捨て	FUN		Out:=TruncDt(In, Accuracy);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象日付時刻	入力	日付時刻切り捨ての対象	データ型に従う	年月日時分秒	DT#1970-1-1-0:0:0
Accuracy	切り捨て単位		切り捨てを行わない最小の時刻単位	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	－	_NANOSEC
Out	切り捨て後日付時刻	出力	切り捨て後の日付時刻	データ型に従う	年月日時分秒	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Accuracy	列挙型_eSUBSEC　列挙子は「機能 (P.2-662)」参照																			
Out																			○	

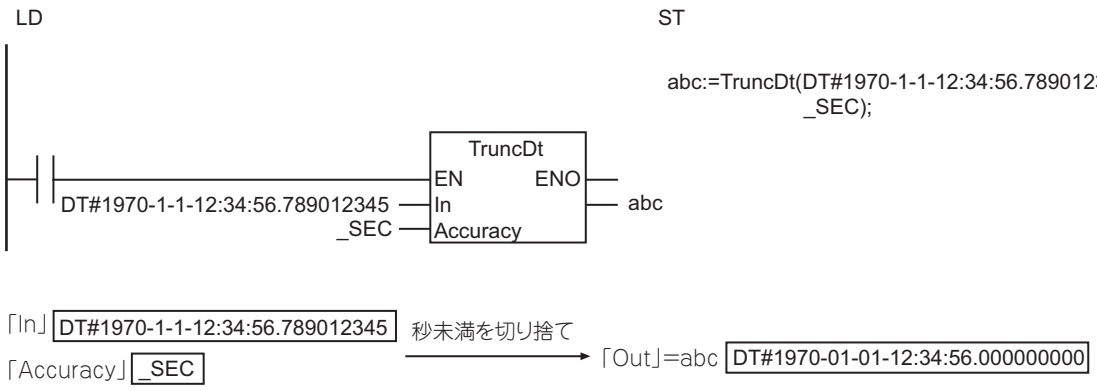
機能

対象日付時刻「In」の、切り捨て単位「Accuracy」未満の値を、切り捨てます。切り捨て後の値は、切り捨て後日付時刻「Out」に格納します。

「Accuracy」のデータ型は、列挙型_eSUBSEC です。列挙子の意味は、以下のとおりです。

列挙子	意味
_NANOSEC	ナノ秒
_MICROSEC	マイクロ秒
_MILLISEC	ミリ秒
_SEC	秒

「In」=DT#1970-1-1-12:34:56.789012345、「Accuracy」=_SEC の場合の例を、以下に示します。



参考

日付時刻どうしを、「EQ, = (P.2-96)」命令などを使って大小比較する際に、本命令を用いて両者の精度を合わせてください。

サンプルプログラム

センサ出力が ON になった日付時刻と、そのときの電圧値を記録するサンプルプログラムです。
日付時刻は、ms 単位の精度で記録します。

センサ出力は「SensorOut」、電圧値は「Voltage」に格納されているとします。現在の日付時刻は、GetTime 命令で取得します。
記録する日付時刻と電圧値は、それらをメンバとする構造体「Recent」の形で、スタック「Stack」に順番に格納していきます。

グローバル変数の定義

● データ型

名称	データ型	コメント
Record	STRUCT	構造体
DandT	DT	日付時刻
Voltage	REAL	電圧値

● グローバル変数

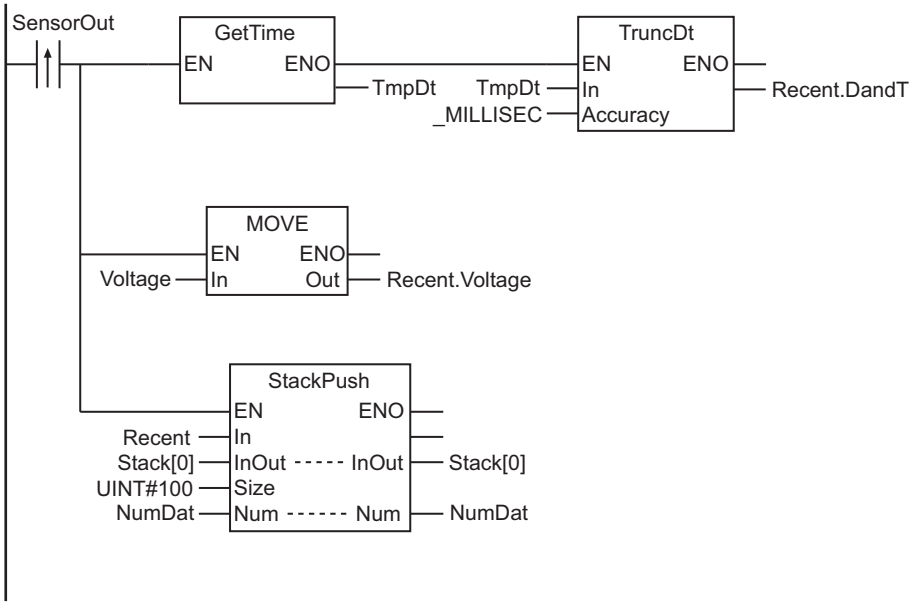
名称	データ型	初期値	コメント
Recent	Record	(DandT:=DT#1970-1-1-0:0:0, Voltage:=0.0)	現在値
Stack	ARRAY[0..99] OF Record	[100((DandT:=DT#1970-1-1-0:0:0, Voltage:=0.0))]	スタック

LD

内部変数	名称	データ型	初期値	コメント
	SensorOut	BOOL	FALSE	センサ出力
	TmpDt	DT	DT#1970-1-1-0:0:0	テンポラリ変数
	Voltage	REAL	0.0	電圧値
	NumDat	UINT	UINT#0	現在の記録データ数

外部変数	名称	データ型	コメント
	Recent	Record	現在値
	Stack	ARRAY[0..99] OF Record	スタック

日付時刻と電圧値記録



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	トリガ
	SensorOut	BOOL	FALSE	センサ出力
	Voltage	REAL	0.0	電圧値
	NumDat	UINT	UINT#0	現在の記録データ数
	R_TRIG_instance	R_TRIG		

外部変数	名称	データ型	コメント
	Recent	Record	現在値
	Stack	ARRAY[0..99] OF Record	スタック

```
// センサ出力が ON になったら、トリガ起動
R_TRIG_instance(SensorOut, Trigger);

IF (Trigger=TRUE) THEN
  // 現在の日付時刻を ms 単位の精度で格納
  Recent.DandT:=TruncDt(
    In :=GetTime(), // 日付時刻取得
    Accuracy:=_MILLISEC); // 精度  ms 単位

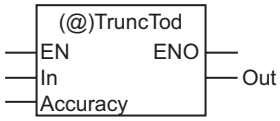
  // 電圧値の現在値を取得
  Recent.Voltage:=Voltage;

  // 日付時刻と電圧値をスタックに格納
  StackPush(
```

```
    In :=Recent, // 日付時刻と電圧値
    InOut:=Stack[0], // スタック配列
    Size :=UINT#100, // スタック配列の要素数 100
    Num :=NumDat); // 現在の記録データ数
END_IF;
```

TruncTod

TOD 型変数の、指定した単位未満の値を切り捨てます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TruncTod	時刻切り捨て	FUN		Out:=TruncTod(In, Accuracy);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	対象時刻	入力	時刻切り捨ての対象	データ型に従う	時分秒	TOD#0:0:0
Accuracy	切り捨て単位		切り捨てを行わない最小の時刻単位	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	－	_NANOSEC
Out	切り捨て後時刻	出力	切り捨て後の時刻	データ型に従う	時分秒	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Accuracy	列挙型_eSUBSEC　列挙子は「機能 (P.2-666)」参照																			
Out																		○		

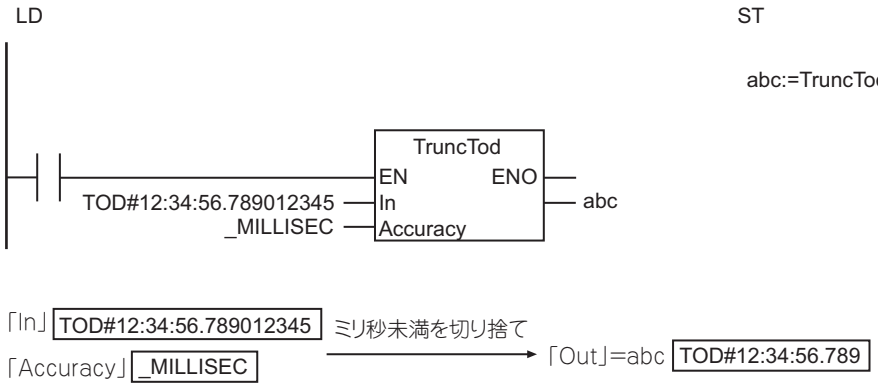
機能

対象時刻「In」の、切り捨て単位「Accuracy」未満の値を、切り捨てます。切り捨て後の値は、切り捨て後時刻「Out」に格納します。

「Accuracy」のデータ型は、列挙型_eSUBSEC です。列挙子の意味は、以下のとおりです。

列挙子	意味
_NANOSEC	ナノ秒
_MICROSEC	マイクロ秒
_MILLISEC	ミリ秒
_SEC	秒

「In」=TOD#12:34:56.789012345、「Accuracy」=_MILLISEC の場合の例を、以下に示します。



参考

時刻どうしを、「EQ, = (P.2-96)」命令などを使って大小比較する際に、本命令を用いて両者の精度を合わせてください。

サンプルプログラム

センサ出力が ON になった時刻と、そのときの電圧値を記録するサンプルプログラムです。
時刻は、秒単位の精度で記録します。

センサ出力は「SensorOut」、電圧値は「Voltage」に格納されているとします。現在時刻は、GetTime 命令と DT_TO_TOD 命令を組み合わせて取得します。
記録する時刻と電圧値は、それらをメンバとする構造体「Recent」の形で、スタック「Stack」に順番に格納していきます。

グローバル変数の定義

● データ型

名称	データ型	コメント
Record	STRUCT	構造体
TofD	TOD	時刻
Voltage	REAL	電圧値

● グローバル変数

名称	データ型	初期値	コメント
Recent	Record	(TofD:=TOD#0:0:0, Voltage:=0.0)	現在値
Stack	ARRAY[0..99] OF Record	[100((TofD:=TOD#0:0:0, Voltage:=0.0))]	スタック

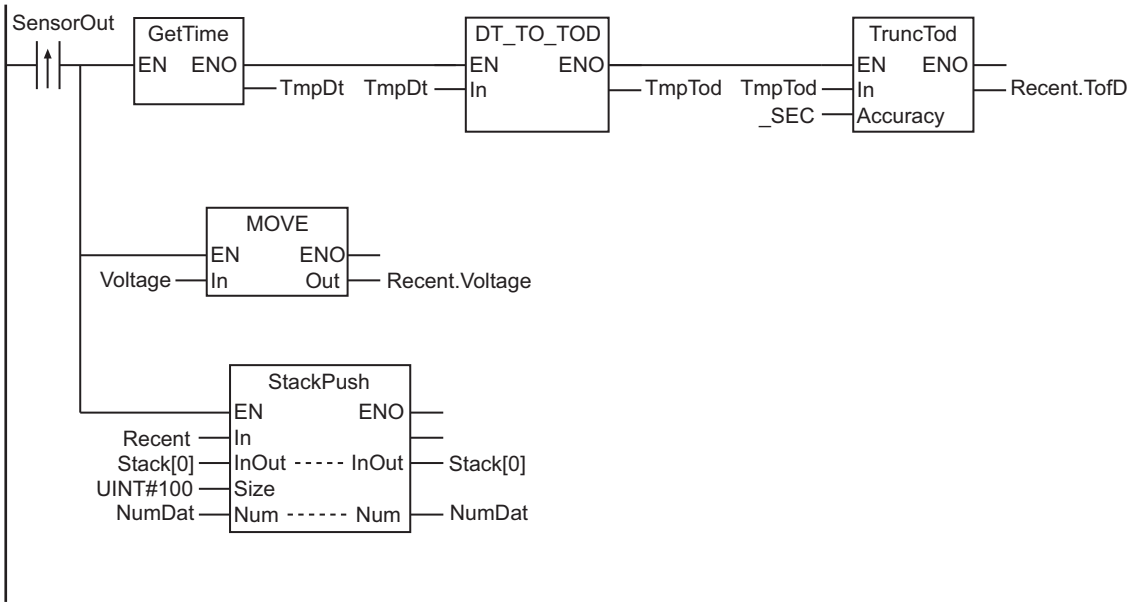
LD

内部変数	名称	データ型	初期値	コメント
	SensorOut	BOOL	FALSE	センサ出力
	TmpDt	DT	DT#1970-1-1-0:0:0	テンポラリ変数
	TmpTod	TOD	TOD#0:0:0	テンポラリ変数
	Voltage	REAL	0.0	電圧値

内部変数	名称	データ型	初期値	コメント
	NumDat	UINT	UINT#0	現在の記録データ数

外部変数	名称	データ型	コメント
	Recent	Record	現在値
	Stack	ARRAY[0..99] OF Record	スタック

時刻と電圧値記録



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	トリガ
	SensorOut	BOOL	FALSE	センサ出力
	TmpTod	TOD	TOD#0:0:0	テンポラリ変数
	Voltage	REAL	0.0	電圧値
	NumDat	UINT	UINT#0	現在の記録データ数
	R_TRIG_instance	R_TRIG		

外部変数	名称	データ型	コメント
	Recent	Record	現在値
	Stack	ARRAY[0..99] OF Record	スタック

```
// センサ出力が ON になったら、トリガ起動
R_TRIG_instance(SensorOut, Trigger);

IF (Trigger=TRUE) THEN
  // 現在の時刻を秒単位の精度で格納
  TmpTod :=DT_TO_TOD (GetTime()); // 時刻取得
  Recent.TofD:=TruncTod(
    In :=TmpTod,
    Accuracy:=_SEC); // 精度 秒単位
```

```
  // 電圧値の現在値を取得
  Recent.Voltage:=Voltage;

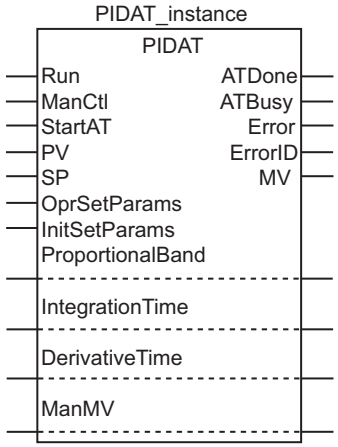
  // 時刻と電圧値をスタックに格納
  StackPush(
    In :=Recent, // 時刻と電圧値
    InOut:=Stack[0], // スタック配列
    Size :=UINT#100, // スタック配列の要素数 100
    Num :=NumDat); // 現在の記録データ数
END_IF;
```

アナログ制御命令

命令	名称	ページ
PIDAT	オートチューニング付き PID 演算	P.2-672
PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	P.2-699
TimeProportionalOut	時分割比例出力	P.2-733
LimitAlarm_**	上下限警報グループ	P.2-750
LimitAlarmDv_**	上下限偏差警報グループ	P.2-754
LimitAlarmDvStbySeq_**	待機シーケンス付き上下限偏差警報グループ	P.2-758
ScaleTrans	スケール変換	P.2-774
AC_StepProgram	ステッププログラム	P.2-777

PIDAT

オートチューニング付きの PID 演算（目標値フィルタ型 2 自由度 PID 制御） を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PIDAT	オートチューニング付き PID 演算	FB		PIDAT_instance(Run, ManCtl, StartAT, PV, SP, OprSetParams, InitSetParams, ProportionalBand, IntegrationTime, DerivativeTime, ManMV, ATDone, ATBusy, Error, ErrorID, MV);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Run	実行条件	入力	TRUE: 実行 FALSE: 停止	データ型に従う	−	FALSE
ManCtl	マニュアル／オート切替		TRUE: マニュアル運転 FALSE: オート運転			
StartAT	オートチューニング実行条件		TRUE: 実行 FALSE: 中止			
PV	測定値（現在値）		測定値（現在値）	*1		0
SP	目標値		目標値			
OprSetParams	運転中設定パラメータ		運転中の、各種設定パラメータ	−		−
InitSetParams	初期設定パラメータ		初期設定パラメータ			
ProportionalBand	比例帯	入出力	比例帯	0.01 ～ 1000.00	%FS	−
IntegrationTime	積分時間		積分時間 値が大きいほど積分効果小。 0 の場合は、積分動作なし。	T#0.0000s ～ T#10000.0000s *2	s	
DerivativeTime	微分時間		微分時間 値が大きいほど微分効果大。 0 の場合は、微分動作なし。	T#0.0000s ～ T#10000.0000s *2		
ManMV	マニュアル操作量		マニュアル操作量	-320 ～ 320	%	

	名称	入／出力	内容	有効範囲	単位	初期値
ATDone	オートチューニング正常終了	出力	TRUE: 正常終了 FALSE: *3	データ型に従う	−	−
ATBusy	オートチューニング実行中		TRUE: 実行中 FALSE: 実行中ではない			
MV	操作量		操作量	-320 ～ 320	%	

*1. 入力レンジ下限「InitSetParams.RngLowLmt」の値～入力レンジ上限「InitSetParams.RngUpLmt」の値。
*2. 0.0001s 未満は、切り捨て。
*3. 異常終了、またはオートチューニングを実行せずに PID 演算中、または PID 演算中ではない。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Run	○																			
ManCtl	○																			
StartAT	○																			
PV														○						
SP														○						
OprSetParams	構造体_sOPR_SET_PARAMS　詳細は「構造体の仕様 (P.2-673)」参照																			
InitSetParams	構造体_sINIT_SET_PARAMS　詳細は「構造体の仕様 (P.2-673)」参照																			
ProportionalBand														○						
IntegrationTime																○				
DerivativeTime																○				
ManMV														○						
ATDone	○																			
ATBusy	○																			
MV														○						

機能

PID 演算を行い、温度調節器などの操作量を制御します。
実行条件「Run」の値が TRUE になったら、PID 演算を開始します。その後、「Run」の値が TRUE である間、周期的に、測定値「PV」の取得→PID 演算→操作量「MV」の出力を繰り返します。
「Run」の値が FALSE になったら、PID 演算を終了します。

オートチューニング機能を持っており、PID 定数の最適値を、自動的に求めることができます。
オートチューニング実行条件「StartAT」の値が TRUE になったら、PID 定数のオートチューニングを実行します。

構造体の仕様

運転中設定パラメータ［OprSetParams］のデータ型は、構造体_sOPR_SET_PARAMS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
OprSetParams	運転中設定パラメータ	運転中の各種設定パラメータ	_sOPR_SET_PARAMS	–	–	–
MVLowLmt	操作量リミット下限値	「MV」をリミット制御する下限値	REAL	-320 ～ 320 *1	%	0
MVUpLmt	操作量リミット上限値	「MV」をリミット制御する上限値	REAL			100
ManResetVal	マニュアルリセット値	比例動作で、偏差 0 の場合の「MV」の値	REAL	-320 ～ 320		0
MVTrackSw	MV トラッキングスイッチ	TRUE: ON FALSE: OFF	BOOL	データ型に従う	–	FALSE
MVTrackVal	MV トラッキング値	MV トラッキング時に、「MV」にセットされる値	REAL	-320 ～ 320	%	0
StopMV	ストップ時操作量	命令実行停止時に、「MV」にセットされる値	REAL			
ErrorMV	異常時操作量	異常発生時に、「MV」にセットされる値	REAL			
Alpha	2 自由度 PID パラメータ α	目標値フィルタ係数 α 0 の場合、目標値フィルタ無効	REAL	0.00 ～ 1.00	–	0.65
ATCalcGain	オートチューニング算出ゲイン	オートチューニング結果の調整係数。 値が大きいほど安定性重視、小さいほど即応性重視。	REAL	0.1 ～ 10.0		1.0
ATHystrs	オートチューニングヒステリシス	リミットサイクルのヒステリシスの大きさ	REAL		%FS	0.2

*1. ただし、「MVLowLmt」＜「MVUpLmt」

初期設定パラメータ［InitSetParams］のデータ型は、構造体_sINIT_SET_PARAMS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
InitSetParams	初期設定パラメータ	初期設定パラメータ	_sINIT_SET_PARAMS	–	–	–
SampTime	サンプリング周期	PID 演算を行う周期	TIME	T#0.0001s ～ T#100.0000s	s	T#0.1s
RngLowLmt	入力レンジ下限	「PV」および「SP」の下限値	REAL	-32000 ～ 32000 *1	–	0
RngUpLmt	入力レンジ上限	「PV」および「SP」の上限値	REAL			100
DirOpr	動作方向	TRUE: 正動作 FALSE: 逆動作	BOOL	データ型に従う		FALSE

*1. ただし、「RngLowLmt」＜「RngUpLmt」

各変数の意味

本命令の各変数の意味を、詳しく説明します。

● Run（実行条件）

本命令の実行条件です。

値が TRUE である間、PID 演算を行います。値が FALSE になったら、PID 演算を停止します。

● ManCtl（マニュアル／オート切替）

本命令には、マニュアル運転とオート運転の 2 つのモードがあります。
「ManCtl」の値によって、どちらのモードでの運転かが決まります。

「ManCtl」の値	運転モード	「MV」の値
TRUE	マニュアル	「ManMV」の値 PID 演算は行いません。
FALSE	オート	PID 演算によって算出した値

● StartAT（オートチューニング実行条件）

PID 定数のオートチューニングの、実行条件です。
「StartAT」の値が TRUE の状態で、「Run」の値が FALSE→TRUE になると、運転開始時にオートチューニングを実行します。
「Run」の値が TRUE で PID 演算中に、「StartAT」の値が FALSE→TRUE になると、運転中にオートチューニングを開始します。

いずれの場合も、オートチューニング中に「StartAT」の値が TRUE→FALSE になると、オートチューニングを中止します。
オートチューニング機能については、「オートチューニング (P.2-685)」を参照してください。

● PV（測定値）

制御対象の測定値です。

● SP（目標値）

制御対象の目標値です。

● MVLowLmt（操作量リミット下限値）、MVUpLmt（操作量リミット上限値）

「MV」の値は、リミット制御することができます。
「MVLowLmt」と「MVUpLmt」は、リミット制御の下限値と上限値です。
必ず、「MVLowLmt」＜「MVUpLmt」となるように、設定してください。

PID 演算で得られた操作量	「MV」の値
「MVLowLmt」以下	「MVLowLmt」
「MVLowLmt」以上「MVUpLmt」以下	PID 演算で得られた操作量
「MVUpLmt」以上	「MVUpLmt」

ただし、「MV」に、ストップ時操作量「StopMV」、異常時操作量「ErrorMV」、マニュアル操作量「ManMV」のいずれかがセットされる場合は、リミット制御の対象外です。

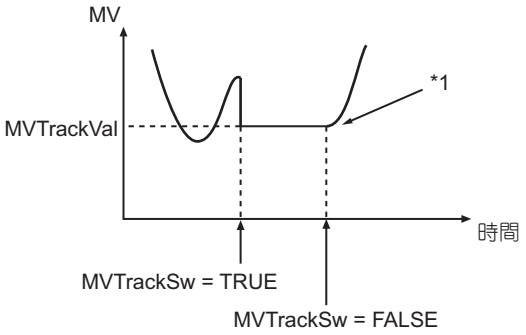
「MVLowLmt」、「MVUpLmt」は本命令の制御状態が運転中（オートチューニング非実行）であっても変更することができます。
ただし、運転中に「MVLowLmt」、「MVUpLmt」を拡大方向に変更すると、そのサンプリング周期では前回と同じ値の「MV」を出力し、バンプレス（急変なし）となります。
このため、「MVLowLmt」、「MVUpLmt」を繰り返し変更すると制御性能に影響をおよぼし、十分な制御性能を得られない場合があります。
運転中に「MVLowLmt」、「MVUpLmt」を繰り返し変更する場合は、制御性能に対する影響を確認の上、使用してください。

● ManResetVal（マニュアルリセット値）

比例動作で、偏差（「PV」と「SP」の差）が 0 になった場合の、「MV」の値です。
「ManResetVal」の値によって、比例動作範囲の位置が決まります。
積分動作を行う場合、マニュアルリセット値の設定は無効になります。したがって、「ManResetVal」の設定は、「IntegrationTime」の値が 0 の場合に有効です。

● MVTrackSw（MV トラッキングスイッチ）

オート運転時、「MV」に、外部からの入力値（MV トラッキング値）を強制的にセットする機能を、MV トラッキングと呼びます。
「MVTrackSw」の値が TRUE である間、MV トラッキングを実行します。
「MVTrackSw」の値が FALSE になると、「MV」の値は、そのサンプリング周期では「MVTrackVal」の値をそのまま引き継ぎ、次のサンプリング周期から PID 演算結果に戻ります。
これにより、「MV」の値が急変することを抑えます。



*1.「MV」の値は、「MVTrackVal」の値を引き継ぎます。

● MVTrackVal（MV トラッキング値）

MV トラッキングで、「MV」に強制的にセットする値です。
「MVTrackVal」の値は、「MVLowLmt」と「MVUpLmt」により、リミット制御されます。

● StopMV（ストップ時操作量）

「Run」の値が FALSE になり、本命令を実行停止した場合に、「MV」にセットされる値です。

● ErrorMV（異常時操作量）

異常が発生した（「Error」の値が TRUE）場合に、「MV」にセットされる値です。
「ErrorMV」の値が、有効範囲内（-320～320）でない場合、異常時の「MV」の値は 0 になります。

● Alpha（2 自由度 PID パラメータ α）

目標値フィルタの係数を決めるパラメータです。
詳細は、「目標値フィルタ型 2 自由度 PID 制御 (P.2-682)」を参照してください。
「Alpha」の値は、通常 0.65 に設定してください。

● ATCalcGain（オートチューニング算出ゲイン）

オートチューニング実行で算出した PID 定数を、実際の PID 定数に反映させる際の係数です。
1.00 を指定すると、オートチューニングの結果をそのまま反映することになります。
「ATCalcGain」の値は、安定性を重視する場合には大きくし、即応性を重視する場合には小さくします。

● ATHystrs（オートチューニングヒステリシス）

オートチューニング時の、リミットサイクルヒステリシスの大きさを表します。
「ATHystrs」の値は、小さい方が高精度のチューニングができます。ただし、測定値が安定せず正常なチューニングが困難な場合には、大きくします。
詳細は、「オートチューニング (P.2-685)」を参照してください。

● SampTime（サンプリング周期）

PID 演算を行う周期の最小値です。
詳細は、「PID 演算の実行タイミング (P.2-686)」を参照してください。
PID 演算を実行してからの経過時間が「SampTime」未満では、つぎの PID 演算が行われることはありません。

● RngLowLmt（入力レンジ下限）、RngUpLmt（入力レンジ上限）

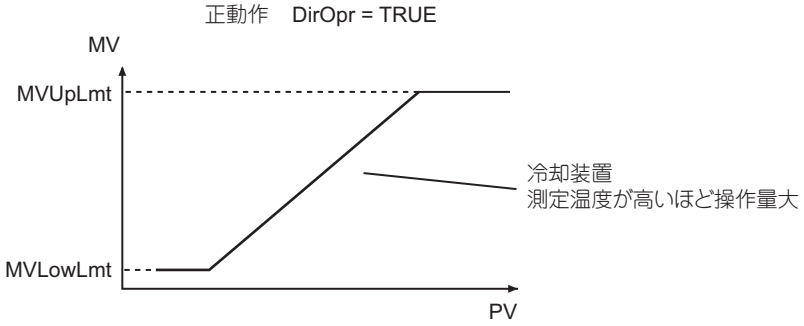
「PV」および「SP」の下限値と上限値です。
「PV」および「SP」に接続するパラメータの値が、この範囲からはずれている場合、異常になります。
必ず、「RngLowLmt」＜「RngUpLmt」となるように、設定してください。

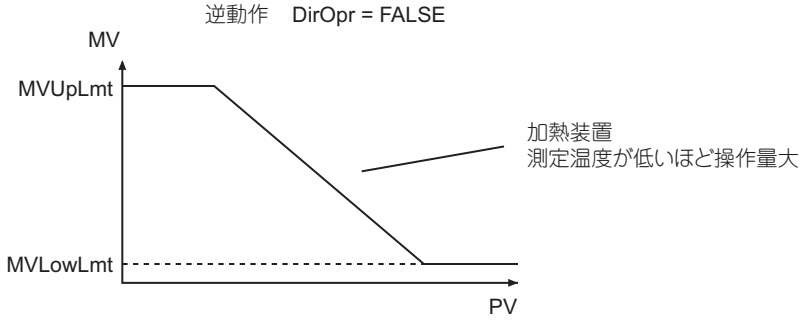
● DirOpr（動作方向）

「PV」の大小に応じて、「MV」を大きくするか小さくするかの指定です。
それぞれを、正動作、逆動作と呼びます。

「DirOpr」の値	意味	「MV」の値
TRUE	正動作	「PV」が大きいほど大
FALSE	逆動作	「PV」が大きいほど小

正動作と逆動作の違いを、温度制御の場合で説明します。
冷却装置の操作量を制御する場合が、正動作になります。つまり、測定温度が高いほど冷却装置の操作量を大きくします。これに対して、加熱装置の操作量を制御する場合が逆動作です。測定温度が低いほど加熱装置の操作量を大きくします。





● **ProportionalBand（比例帯）**

3 つある PID 定数のうちの 1 つです。詳細は、「比例動作(P) (P.2-679)」を参照してください。
「ProportionalBand」の値が大きいと、オフセットが大きくなります。小さいとハンティングが発生します。

● **IntegrationTime（積分時間）**

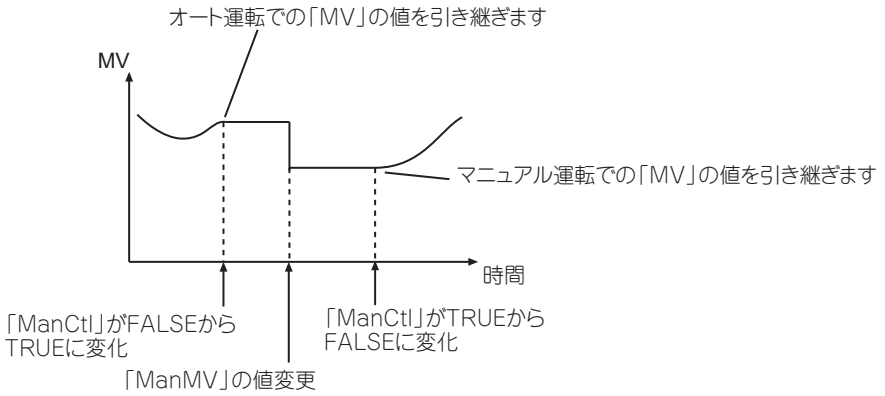
3 つある PID 定数のうちの 1 つです。詳細は、「積分動作(I) (P.2-681)」を参照してください。
「IntegrationTime」の値が大きいほど、積分動作が弱くなります。

● **DerivativeTime（微分時間）**

3 つの PID 定数のうちの 1 つです。詳細は、「微分動作(D) (P.2-681)」を参照してください。
「DerivativeTime」の値が大きいほど、微分動作が強くなります。

● **ManMV（マニュアル操作量）**

マニュアル運転時（「ManCtl」=TRUE）に、「MV」にセットされる値です。
ただし、オート運転からマニュアル運転に切り替えた直後は、オート運転での「MV」の値をそのまま引き継ぎます。
「ManMV」の値が「MV」にセットされるのは、マニュアル運転に切り替えた後に、「ManMV」の値を変更したときです。
また、マニュアル運転からオート運転に切り替えた直後も、マニュアル運転での「MV」の値を引き継ぎます。
なお、「ManMV」の値は、「MVLowLmt」と「MVUpLmt」の間でなくてもかまいません。



● **ATDone（オートチューニング正常終了）**

オートチューニングが、正常終了したことを示すフラグです。
オートチューニングが正常終了してから、「StartAT」の値が TRUE である間、TRUE になります。

FALSE になるのは、以下の場合です。
- オートチューニング異常終了時。
- オートチューニング実行中（「ATBusy」の値が TRUE）。
- オートチューニングを実行せずに、PID 演算中。
- PID 演算中ではない（「Run」の値が FALSE）。
- 「StartAT」の値が FALSE である。

● **ATBusy（オートチューニング実行中）**

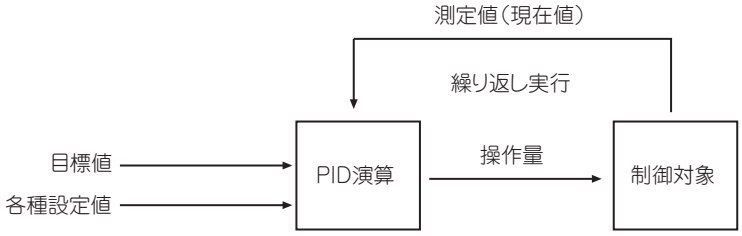
オートチューニングが、実行中であることを示すフラグです。
オートチューニング実行中に、TRUE になります。それ以外は、FALSE になります。

● **MV（操作量）**

制御対象に与える操作量です。

PID 演算の概要

PID 演算とは、制御対象の現在値を測定し、その値を目標値に近づけるために操作量を調節することを繰り返すフィードバック制御の演算方法のひとつです。
したがって、本命令の入力は、測定値、目標値、および演算のための各種設定値であり、出力は操作量です。
測定→演算→操作量出力を周期的に繰り返すことにより、現在値が目標値に近づいていきます。

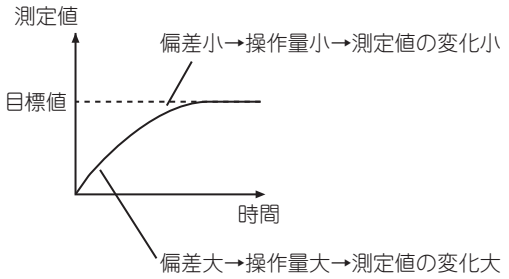


比例動作(P)、積分動作(I)、微分動作(D)

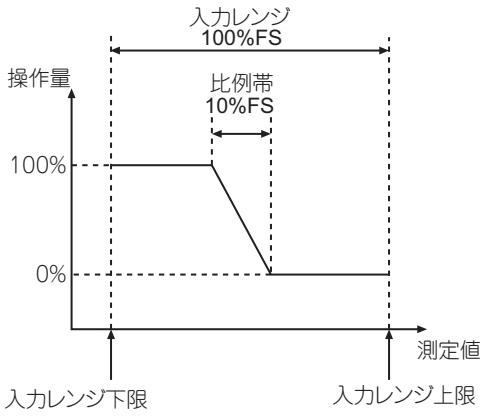
PID 演算は、比例動作(Proportional action)、積分動作(Integral action)、微分動作(Derivative action)の合成によって行われます。

● **比例動作(P)**

測定値と目標値の差（偏差）に比例して、操作量の絶対値を大きくする制御方法です。
制御対象の測定値は、下図のように推移します。

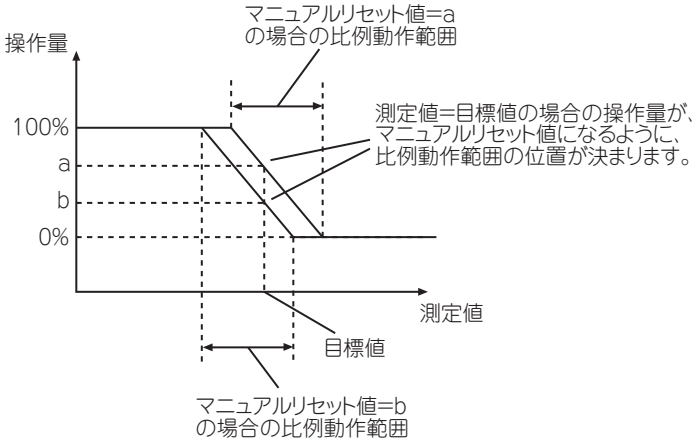


比例動作の設定値のひとつに、比例帯があります。
比例帯とは、比例動作を行う測定値の幅のことです。測定値が比例動作範囲をはずれた場合、操作量には 100%または 0%をセットします。
比例帯は、入力レンジのうち、比例動作を行う範囲の割合(%FS)で表現します。比例帯の値が 10%FS の場合の図を、以下に示します。

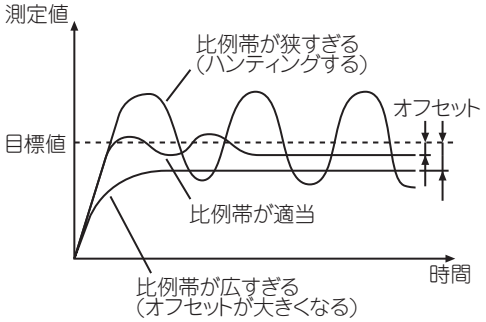


比例動作のもうひとつの設定値が、マニュアルリセット値です。
マニュアルリセット値とは、偏差が 0 の場合の操作量のことです。
マニュアルリセット値によって、測定値と操作量のグラフにおける、比例動作範囲の位置が決まります。

マニュアルリセット値と比例動作範囲の関係を以下に示します。
測定値と目標値が一致した際の操作量が、マニュアルリセット値と一致するように、比例動作範囲の位置が決まります。



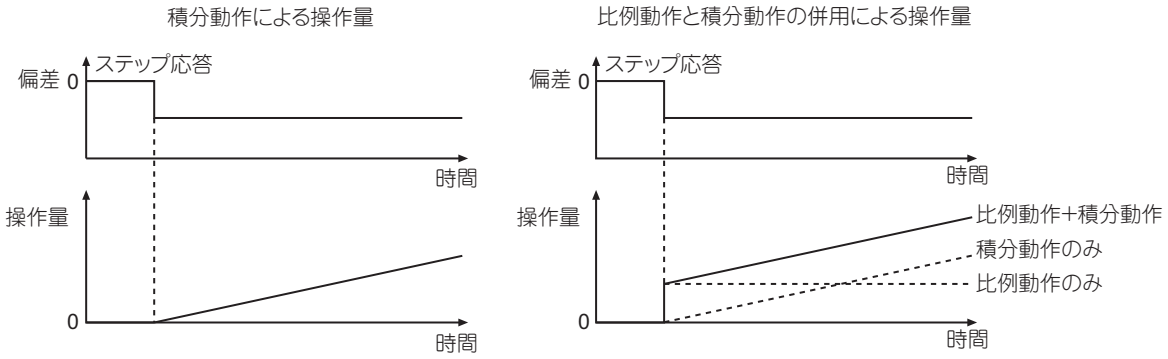
マニュアルリセット値が適正でないと、偏差を完全に 0 にすることはできません。この残った偏差のことをオフセット（残留偏差）といいます。
比例帯を狭くするとオフセットを小さくすることができます。しかし、狭くしすぎると、測定値が目標値で止まらずに行き過ぎてしまう現象が起こります。これをオーバーシュートと言います。
また、測定値が安定せず、目標値付近で振動する現象をハンティングと言います。



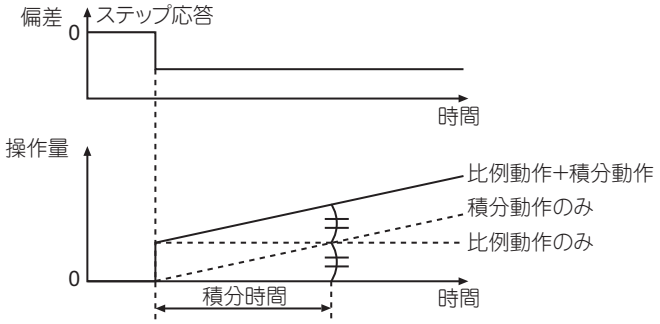
● 積分動作(I)

比例動作だけでオフセットを 0 にするには、比例帯とマニュアルリセット値の綿密な調整が必要です。また、オフセットの大きさが外乱によって変化する場合、それらの調整を頻繁にやらなければなりません。
その手間を省くために、比例動作と併用するのが積分動作です。
積分動作は、偏差を時間軸で累積（積分）し、その値に比例して操作量の絶対値を大きくする制御方法です。
積分動作を行う場合、マニュアルリセット値の設定は無効になります。

ステップ状の偏差が発生した場合の、積分動作による操作量の推移を下図左に示します。また、比例動作と積分動作を併用した場合の操作量の推移を、下図右に示します。



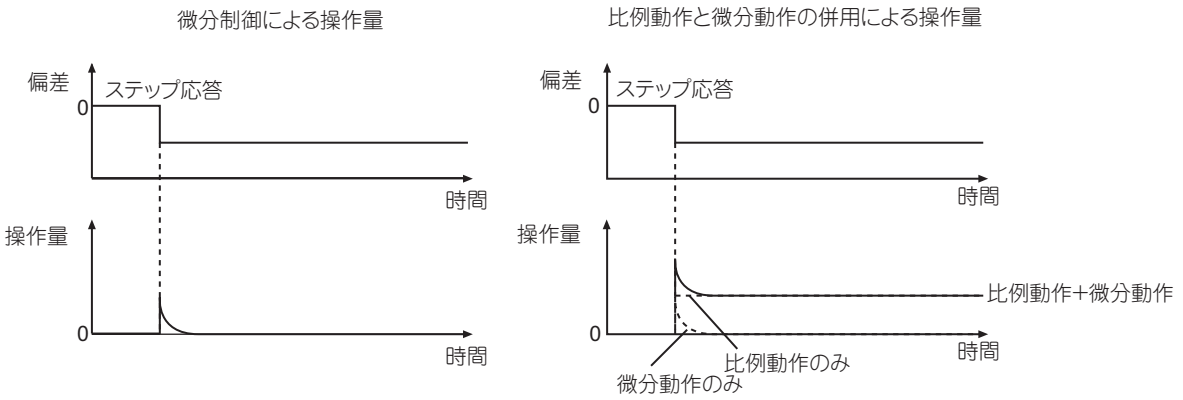
積分動作の設定値に、積分時間があります。
これは、ステップ状の偏差に対して、積分動作による操作量が、比例動作による操作量と等しくなるまでの時間のことです。
積分時間が短いほど、積分動作が強くなります。積分時間が短いと、それだけオフセットを 0 にするまでの時間が短くなりますが、その反面、ハンティング発生の要因になります。



● 微分動作(D)

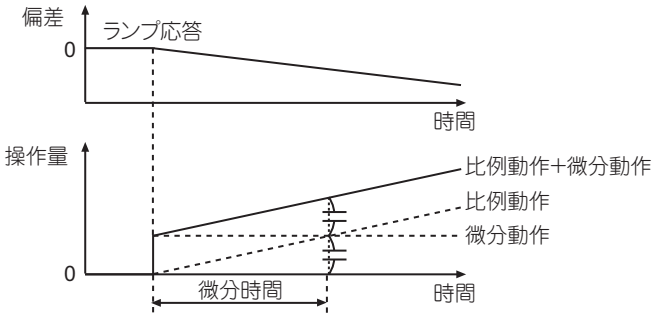
比例動作と積分動作を併用すれば、オフセットを 0 にし、測定値を目標値に一致させることができます。
しかし、外乱が発生して測定値が急に変化した場合、それを元に戻すにはある程度の時間がかかります。
外乱が発生しても、測定値を短時間で目標値に戻す目的で用いるのが、微分動作です。
微分動作は、偏差を時間軸に対して微分し、その大きさに比例して操作量の絶対値を大きくする制御方法です。つまり、測定値の変化が大きいほど、微分動作による操作量の絶対値が大きくなります。

ステップ状の偏差が発生した場合の、微分動作による操作量の推移を以下に示します。また、比例動作と微分動作を併用した場合の操作量の推移も示します。



微分動作の設定値に、微分時間があります。これは、ランプ状の偏差に対して、微分動作による操作量が、比例動作による操作量と等しくなるまでの時間のことです。

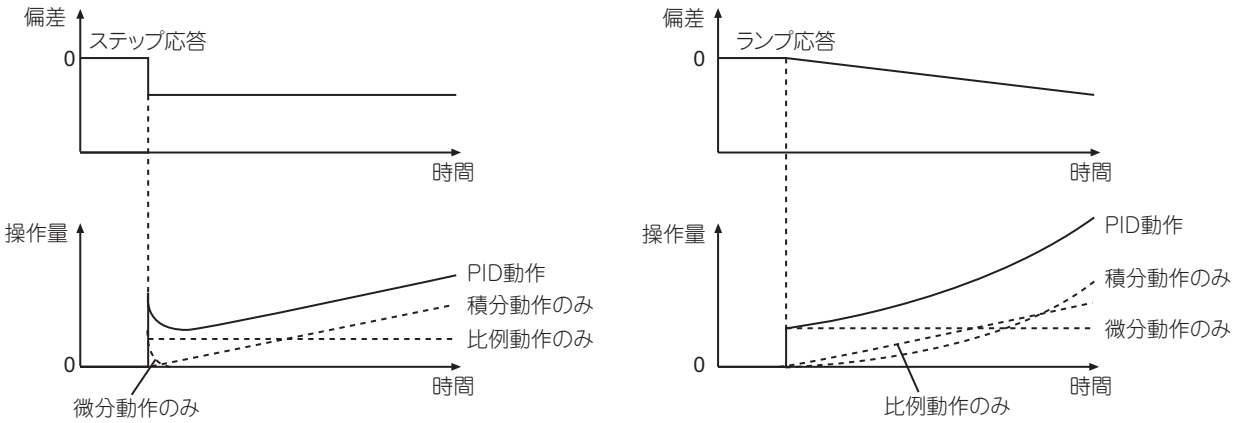
微分時間が長いほど、微分動作が強くなります。微分時間が長いと、それだけ外乱への対応は速くなりますが、その反面、ハンティング発生の要因になります。



● PID 動作

比例動作、積分動作、微分動作のすべての操作量を加算したものが、PID 動作の操作量になります。

ステップ状の偏差、ランプ状の偏差に対する PID 動作の操作量の推移を、以下に示します。



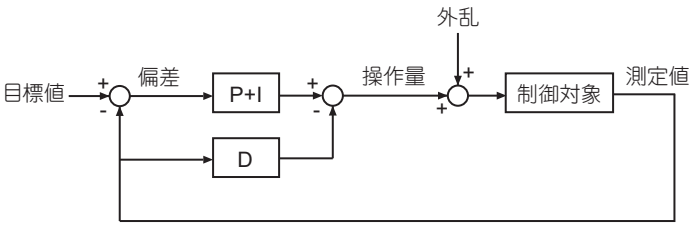
目標値フィルタ型 2 自由度 PID 制御

PID 制御で調整が必要な主な設定値は、比例帯、積分時間、微分時間の 3 つです。これらをまとめて、PID 定数といいます。
PID 定数の値は、PID 制御の以下の 2 つの性能に影響します。

- 目標値応答性能: 目標値が変化した場合に、それに追従する性能
- 外乱応答性能: 外乱が発生して測定値が大きく変化した場合に、それを修正する性能

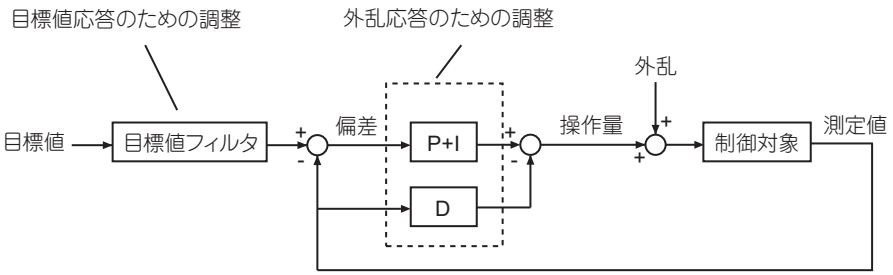
以下に、基本 PID 制御のブロック図を示します。

目標値と外乱は入力される位置が異なるため、目標値応答性能と、外乱応答性能のどちらにも最適な PID 定数を見つけることは困難です。つまり、PID 定数を、目標値応答性能を重視して設定すると外乱への対応が遅くなり、外乱応答性能を重視して設定するとオーバーシュートが発生してしまいます。

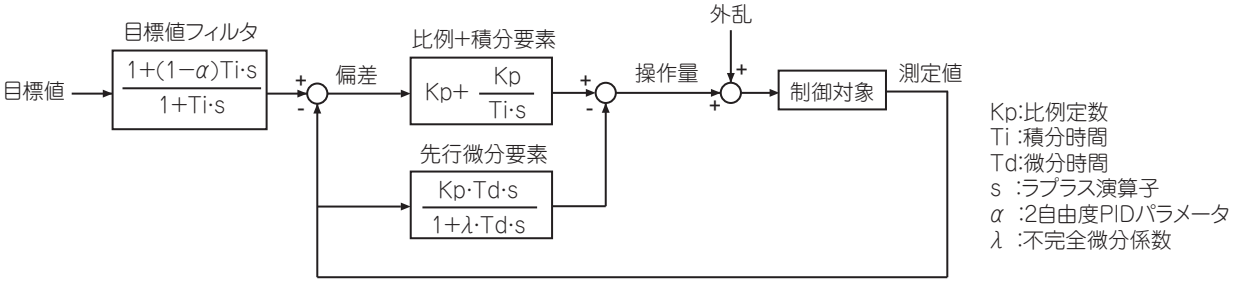


目標値応答性能と外乱応答性能を両立するのが、2 自由度 PID 制御です。
“2 自由度”とは、目標値応答性能と外乱応答性能を調整するパラメータを独立に 2 種類持つという意味です。

ブロック図は以下のようになります。
新しく目標値フィルタを設け、そこにもうひとつの調整パラメータを持ちます。
PID 定数は、外乱応答性能を最大にするように調整します。そして、その設定が目標値応答に対しても最適になるように、目標値フィルタによって目標値を加工します。
PID 定数の値と、目標値フィルタの設定値は、独立に調整可能なので、目標値応答性能と外乱応答性能の両方を高めることができます。



本命令の、各ブロックの演算式を以下に示します。
目標値フィルタの値（目標値にかける係数）は、積分時間と 2 自由度 PID パラメータ α で調整します。
α の値は、0.65 が最適であり、通常は変更することはありません。α の値が小さいほど目標値フィルタの影響は小さくなります。



運転開始の手順

本命令は、適正な PID 定数を使って運転する必要があります。そのための運転開始には、PID 定数の最適値がわかっている場合と、わからない場合の 2 つの方法があります。

PID 定数は、運転中に変更することが可能です。また、運転中にオートチューニングを実行することも可能です。その場合は、「StartAT」の値を、TRUE にします。

● PID 定数の最適値がわからない場合

PID 定数の最適値がわからない場合、運転開始時にオートチューニングを行い、PID 定数の最適値を求めます。
「StartAT」の値を TRUE にした状態で、「Run」の値を FALSE から TRUE に変化させます。
まず初めにオートチューニングを実行し、求めた PID 定数を使って PID 演算を開始します。

● PID 定数の最適値がわかっている場合

PID 定数の最適値を、「ProportionalBand」、「IntegrationTime」、「DerivativeTime」に設定して、「Run」の値を FALSE から TRUE に変化させます。
「ProportionalBand」、「IntegrationTime」、「DerivativeTime」は、入出力変数です。
定数を入力パラメータとすることはできません。必ず、適当な変数を定義して、そこに値を代入したものを入力パラメータとしてください。

制御状態と操作量

操作量「MV」は、制御状態によって、以下のように決まります。

<table>
<tr><th rowspan="2">制御状態</th><th colspan="5">変数の値</th><th rowspan="2">「MV」
操作量</th></tr>
<tr><th>「ManCtl」
マニュアル／オート切替</th><th>「Run」
実行条件</th><th>「Error」
異常終了</th><th>「MVTrack Sw」
MV トラッキングスイッチ</th><th>「ATBusy」
オートチューニング実行中</th></tr>
<tr><td>異常終了</td><td rowspan="6">FALSE</td><td rowspan="5">TRUE</td><td>TRUE</td><td>－</td><td rowspan="2">FALSE</td><td>「ErrorMV」
異常時操作量</td></tr>
<tr><td>オート運転中
（MV トラッキング）</td><td rowspan="4">FALSE</td><td>TRUE</td><td>「MVTrackVal」
MV トラッキング値</td></tr>
<tr><td>オート運転中
（オートチューニング実行中）</td><td rowspan="3">FALSE</td><td>TRUE</td><td>操作量リミット上限値と、操作量リミット下限値を繰り返し</td></tr>
<tr><td>オート運転中
（オートチューニング非実行）</td><td rowspan="3">FALSE</td><td>その時点での PID 定数によって算出された値</td></tr>
<tr><td>命令実行停止</td><td>FALSE</td><td rowspan="2">－</td><td rowspan="2">－</td><td>「StopMV」
ストップ時操作量</td></tr>
<tr><td>マニュアル運転中</td><td>TRUE</td><td>－</td><td>「ManMV」
マニュアル操作量</td></tr>
</table>

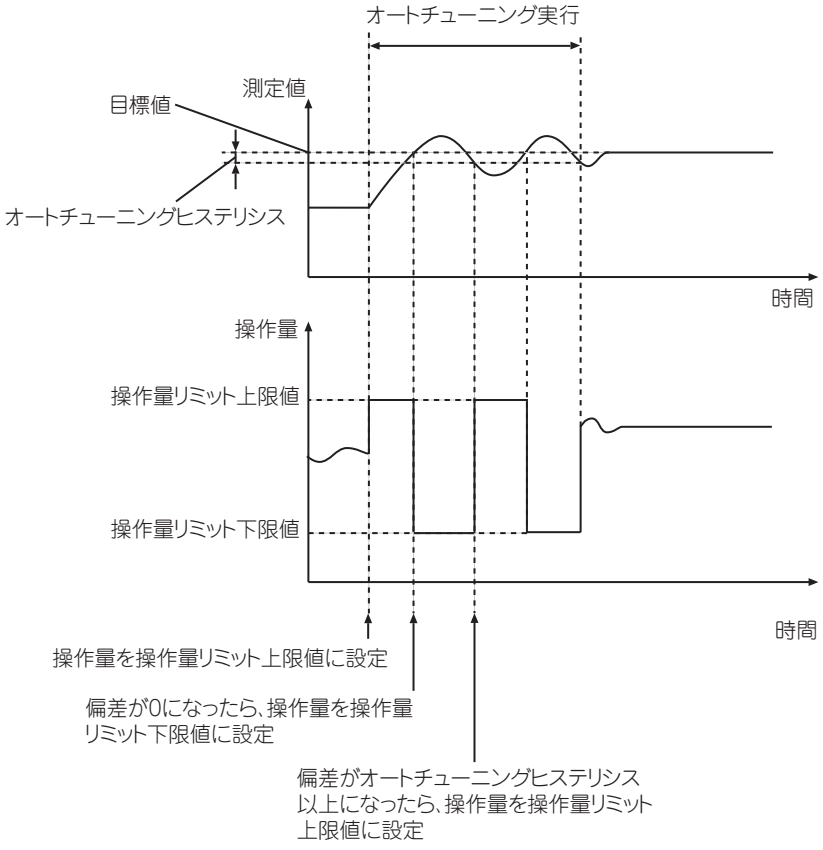
オートチューニング

2 自由度 PID パラメータ α を調整することはあまりないため、本命令で主に調整する設定値は、PID 定数となります。
本命令には、PID 定数をオートチューニングする機能があります。
オートチューニングの方法には、リミットサイクル法を使います。

リミットサイクル法とは、操作量を強制的に、一時的に操作量リミット上限値および操作量リミット下限値にして、その際の測定値の変化のしかたから、PID 定数の最適値を求める方法です。

目標値 > 測定値の場合にオートチューニング実行指示があると、操作量を、まず操作量リミット上限値に設定します。その後、偏差が 0 になったら、操作量を操作量リミット下限値に設定します。そして、偏差がオートチューニングヒステリシスの値以上になったら、操作量を、また操作量リミット上限値に設定します。これを、2 回繰り返して、PID 定数の最適値を算出します。

目標値 < 測定値の場合にオートチューニング実行指示があると、操作量を、まず操作量リミット下限値に設定します。その後は、上記と同じ手順で PID 定数の最適値を算出します。



オートチューニングは、「Run」の値が TRUE で PID 演算中に、「StartAT」の値が FALSE→TRUE になると、運転中に実行されます。または、「StartAT」の値が TRUE の状態で、「Run」の値が FALSE→TRUE になると、運転開始時に実行されます。

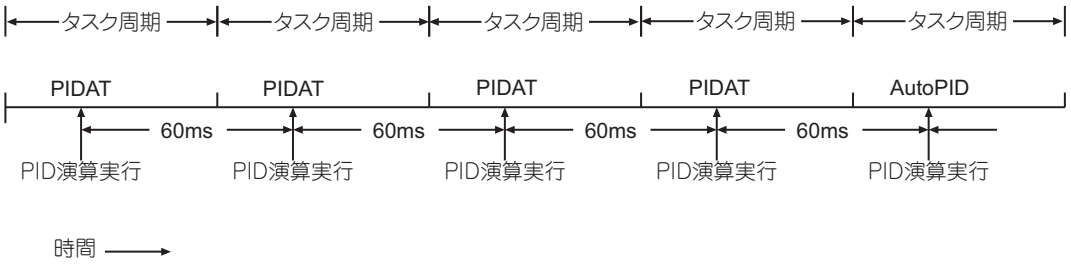
オートチューニングが正常終了すると、算出された PID 定数の最適値が、直ちに反映されます。
オートチューニング中（「ATBusy」=TRUE）に「ATBusy」の値を FALSE にすると、オートチューニングを中止します。この場合、オートチューニング開始直前の PID 定数で、PID 演算を再開します。

PID 演算の実行タイミング

PID 演算は、周期的に繰り返し実行されます。PID 演算が実行されるのは、ユーザプログラム中で本命令が実行されるときです。
ただし、前回の PID 演算実行からの経過時間が、サンプリング周期「SampTime」未満の場合は実行されません。
また、前回の PID 演算実行からの経過時間が「SampTime」以上の場合は実行され、余った時間（前回の実行からの経過時間－「SampTime」）が、次回に繰り越されます。詳しくは、下図を参照してください。

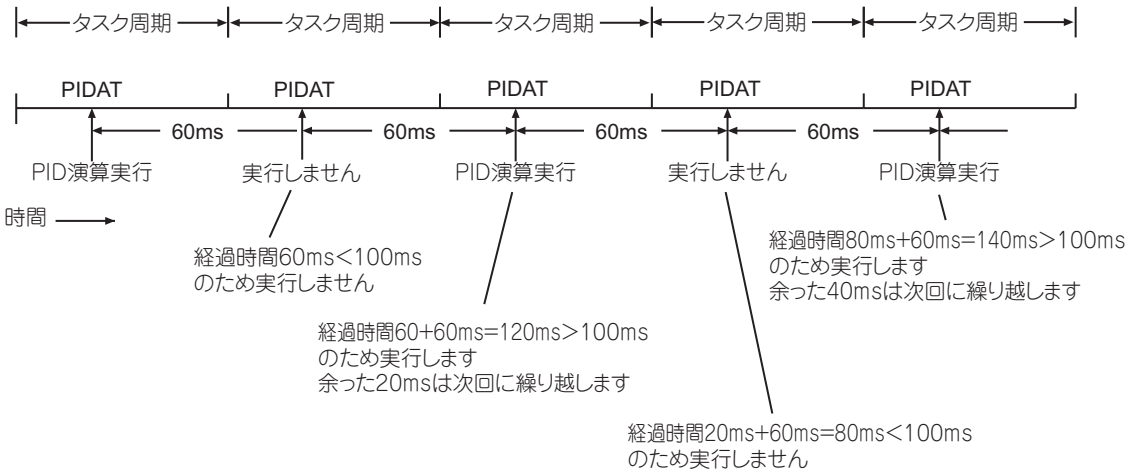
タスク周期=60ms 「SampTime」<60msの場合

タスク周期≧「SampTime」なので、1タスク周期に1回、必ず実行されます。



タスク周期=60ms 「SampTime」=100msの場合

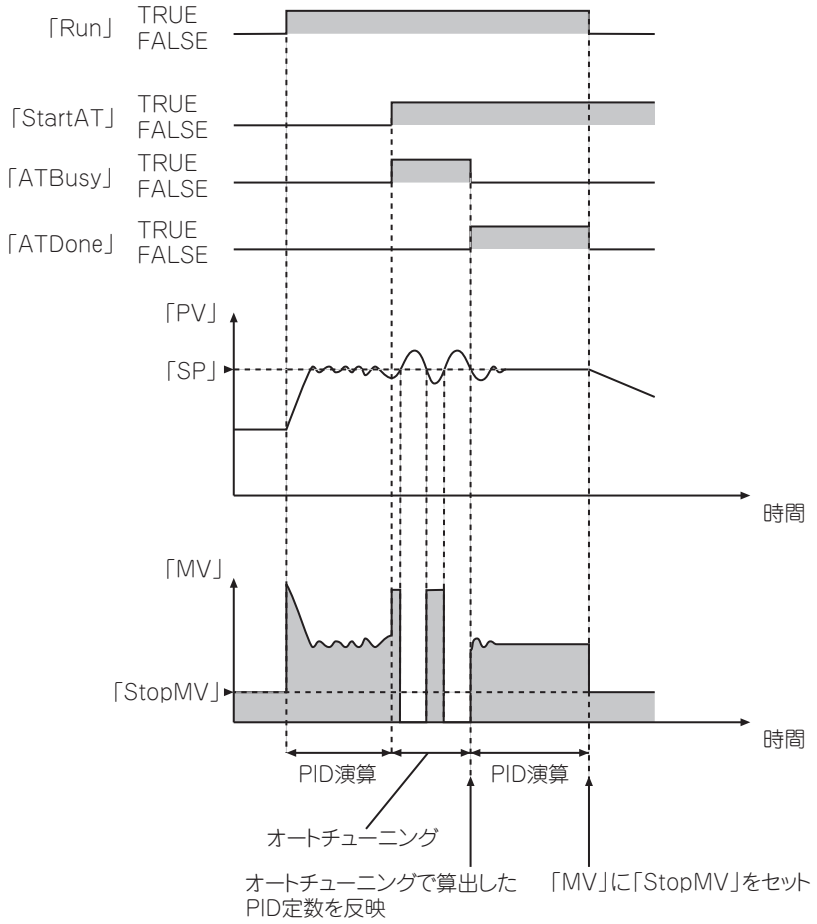
タスク周期<「SampTime」なので、実行されない場合があります。



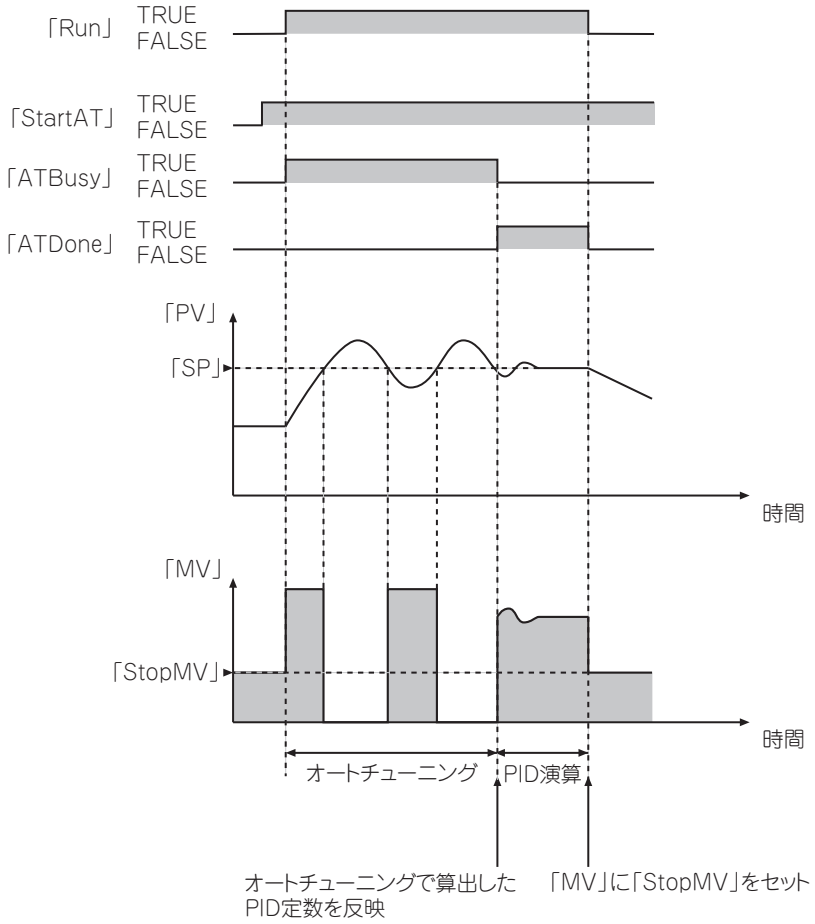
タイミングチャート

いくつかの場合の、各変数のタイミングチャートを示します。

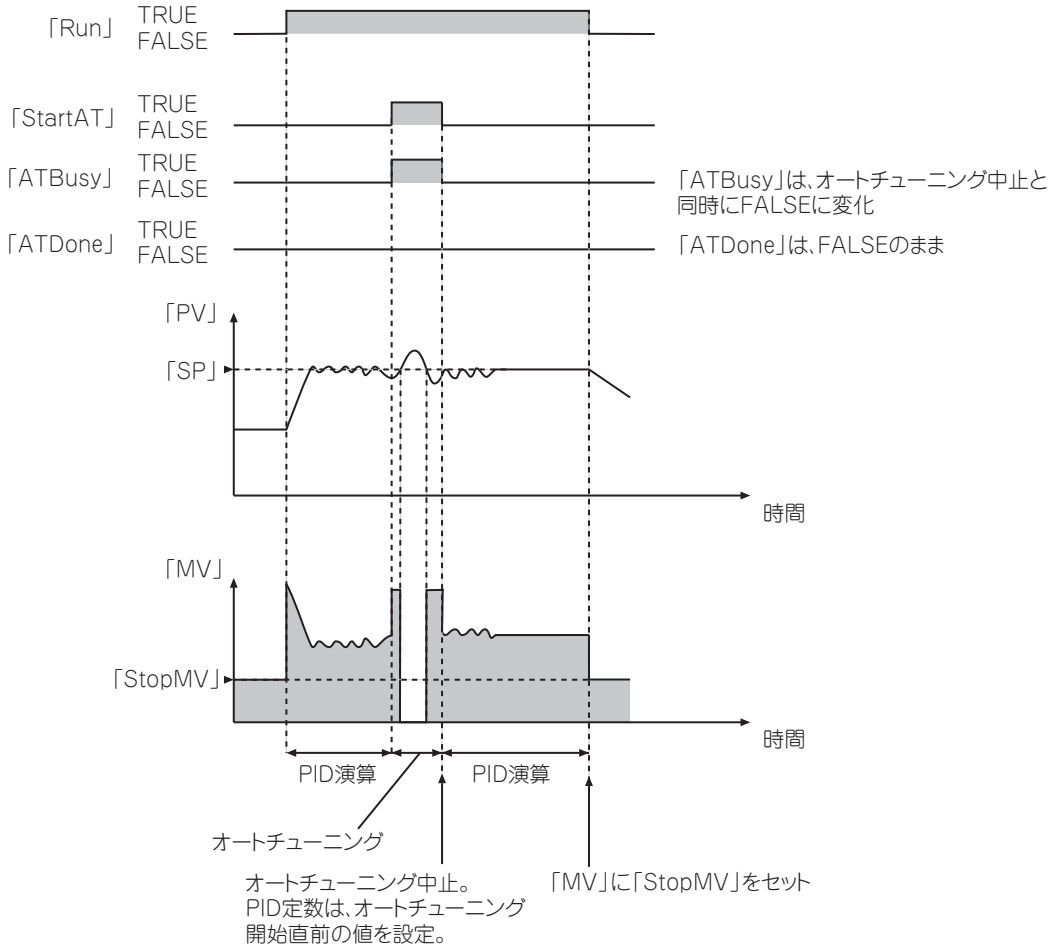
● **オート運転中に、オートチューニングを実行した場合**



● **本命令実行の最初に、オートチューニングを実行した場合**



● **オートチューニングを、中止した場合**

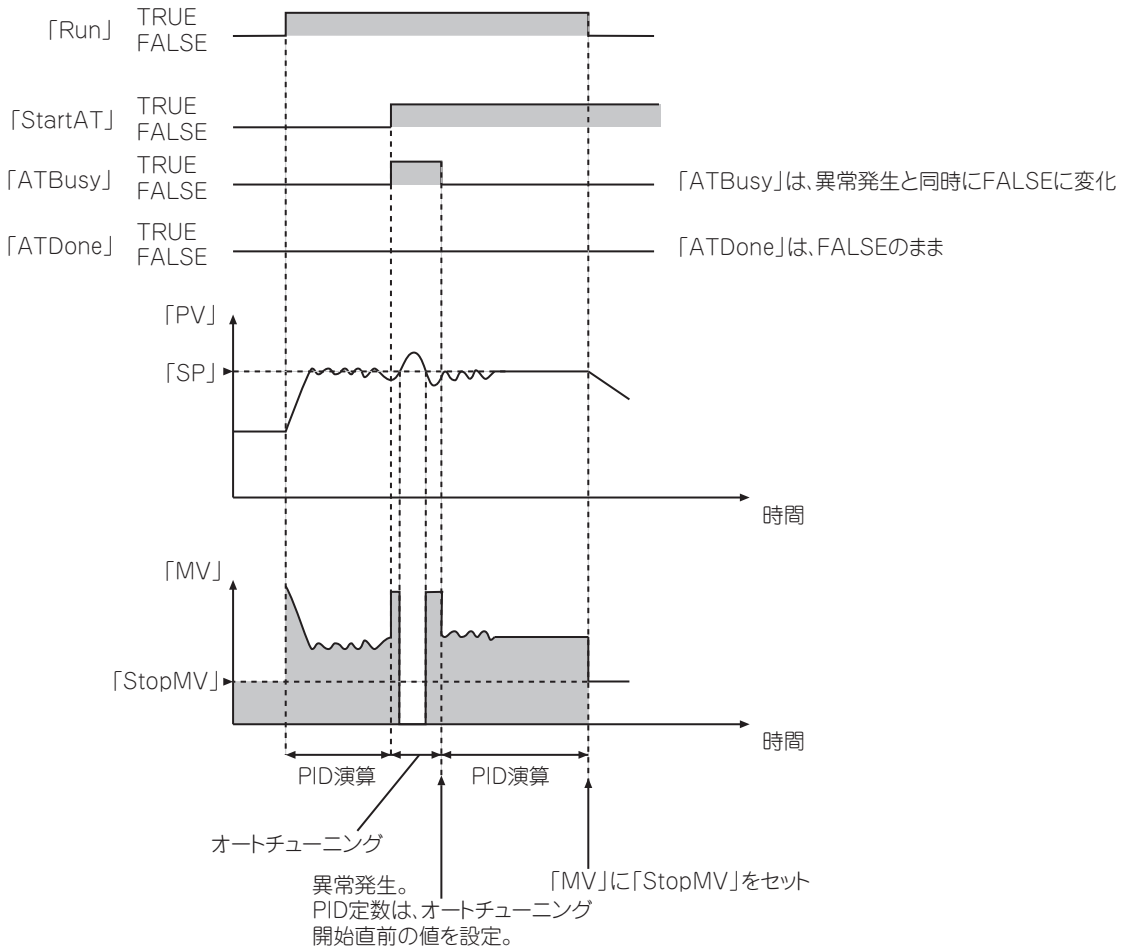


● **オートチューニング中に、オートチューニング異常が発生した場合**

つぎの場合、オートチューニング異常となり、オートチューニングを中止します。

- 操作量＝操作量リミット上限値で、偏差が 0 になるまでの時間が、19999s を超える。
- 操作量＝操作量リミット下限値で、偏差が「ATHystrs」以上になるまでの時間が、19999s を超える。

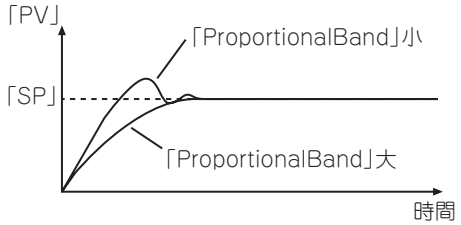
オートチューニングを中止すると、オートチューニング開始直前の PID 定数で、PID 演算を再開します。



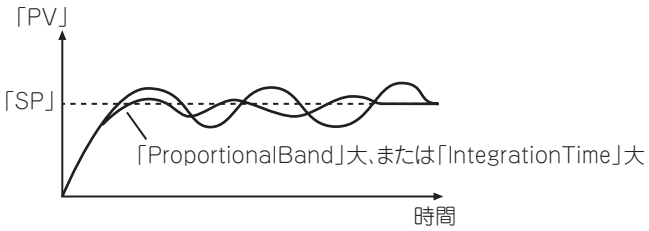
参考

PID 定数の調整方法

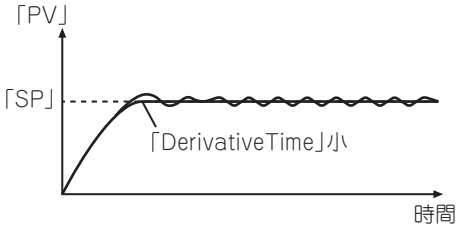
- 制御対象が安定するまで多少の時間がかかっても、ハンティングを発生させたくない場合は、「ProportionalBand」の値を大きくします。逆に、多少のハンティングが発生しても、早く安定させたい場合は、「ProportionalBand」の値を小さくします。



- ハンティングを繰り返す場合は、「ProportionalBand」を大きくするか、「IntegrationTime」を大きくします。



• ハンティングを短い周期で繰り返す場合、「DerivativeTime」を小さくします。



温度調節の場合の PID 定数初期値

本命令を、温度調節に使用する場合は、PID 定数の初期値として、以下の値を参考にしてください。PID 定数以外は、各変数の初期値を使用してください。

変数	参考初期値 *1
「ProportionalBand」	10%FS
「IntegrationTime」	233s
「DerivativeTime」	40s

*1. オートチューニングを実行した場合は、その結果に従ってください。

使用上の注意

• 「PV」および「SP」の値は、「RngLowLmt」の値以上「RngUpLmt」の値以下であることが必要です。そのために、以下に示すように、これらの変数の単位系を合わせてください。

単位系	「PV」、「SP」の値	「RngLowLmt」、「RngUpLmt」の値
%FS	「PV」＝（測定値の物理量－MIN）／（MAX－MIN）×100 *1 「SP」＝（目標値の物理量－MIN）／（MAX－MIN）×100 *1	「RngLowLmt」=0 「RngUpLmt」=100
物理量	「PV」=測定値の物理量 「SP」=目標値の物理量	「RngLowLmt」= MIN *1 「RngUpLmt」= MAX *1

*1. MAX:入力レンジ上限の物理量、MIN:入力レンジ下限の物理量

• 各変数の変更可否は、制御状態により、以下のようになります。

変数	制御状態		
	命令実行停止 *1	運転中（オートチューニング非実行） *2	運転中（オートチューニング実行中） *3
Run	○	○	○
ManCtl	○	○	○
StartAT	○	○	○
PV	○	○	○
SP	○	○	×
MVLowLmt	○	○	×
MVUpLmt	○	○	×

変数	制御状態		
	命令実行停止 *1	運転中（オートチューニング非実行） *2	運転中（オートチューニング実行中） *3
ManResetVal	○	○	×
MVTrackSw	○	○	×
MVTrackVal	○	○	×
StopMV	○	○	○
ErrorMV	○	○	○
Alpha	○	○	×
ATCalcGain	○	○	×
ATHystrs	○	○	×
SampTime	○	×	×
RngLowLmt	○	×	×
RngUpLmt	○	×	×
DirOpr	○	×	×
ProportionalBand	○	○	×
IntegrationTime	○	○	×
DerivativeTime	○	○	×
ManMV	○	○	○

*1. 「ManCtl」=TRUE または「Run」=FALSE または「Error」=TRUE または「MVTrackSw」=TRUE
*2. 「ManCtl」=FALSE かつ「Run」= TRUE かつ「Error」=FALSE かつ「MVTrackSw」= FALSE かつ「ATBusy」=FALSE
*3. 「ManCtl」=FALSE かつ「Run」= TRUE かつ「Error」=FALSE かつ「MVTrackSw」= FALSE かつ「ATBusy」=TRUE

- 「SampTime」は、100ns 未満は切り捨てられます。
- 「ManCtl」の値が TRUE 時に、「StartAT」の値を TRUE にした場合、つぎに「ManCtl」の値が FALSE になった時点でオートチューニングを開始します。
- 「ErrorMV」の値が、有効範囲内（-320～320）でない場合、異常時の「MV」の値は 0 になります。
- オートチューニング中に「ManCtl」の値を TRUE にすると、オートチューニングを中止します。
- オートチューニング異常が発生しても、「Error」の値は TRUE にはなりません。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。「ATDone」、「ATBusy」は FALSE となります。「ManCtl」と「Run」の値がともに FALSE であれば、「MV」に「ErrorMV」の値がセットされます。ただし、「ErrorMV」の値が有効範囲外の場合は、「MV」の値は 0 になります。

異常の内容	「ErrorID」の値
入力変数のいずれかの値が、有効範囲外	16#0400
「RngLowLmt」≧「RngUpLmt」	16#0401
「MVLowLmt」≧「MVUpLmt」	

- 上記以外の条件で異常停止させたい場合は、異常発生時に、「Run」の値を FALSE にする処理を設けてください。
- 「PV」または「SP」の値が有効範囲外で異常が発生した後、値を有効範囲内に変更しても、5 秒間は異常状態を保持します。つまり、5 秒間は「Error」の値が FALSE のままになります。
- 異常解除後、「Run」の値が TRUE なら自動的に PID 演算を再開します。「Run」と「StartAT」の値がともに TRUE なら、自動的にオートチューニングを再開します。
- 異常の有無の判定は、サンプリング周期ごとに行います。

サンプルプログラム

PIDAT 命令を使って、温度制御を行います。

PIDAT 命令の操作量は時分割比例出力に変換し、ヒータに出力します。
ここでは、時分割比例出力への変換にタイマ命令を使用しています。
時分割比例出力への変換に、TimeProportionalOut 命令を使用する場合は、TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

仕様

以下のような仕様で、温度制御を行います。

項目	仕様
入力種別	K 型熱電対
入力ユニット	絶縁型高分解能フルマルチ入力ユニット 形 CJ1W-PH41U
出力ユニット	トランジスタ出力ユニット 形 CJ1W-OD212
目標温度	90℃
PID 演算のサンプリング周期	100ms
出力制御周期	1s

構成・設定

入力ユニット形 CJ1W-PH41U の設定は、以下のとおりとします。

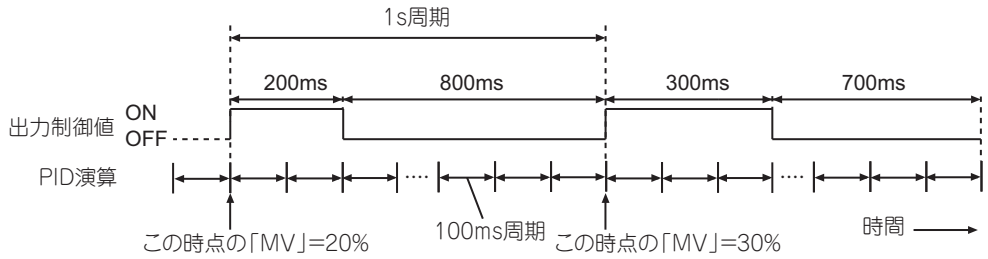
項目名	設定値
Input1:Input signal type	K(1)

I/O マップ設定は、以下のとおりとします。

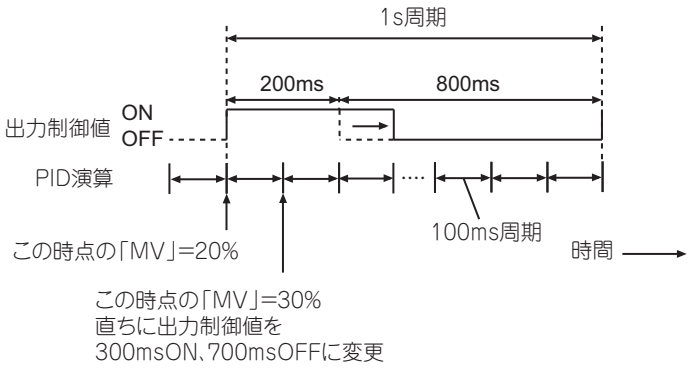
ユニット	ポート	説明	変数
形 CJ1W-PH41U	Ch1_AIInPV	入力 No.1 測定値（INT 型）	AI1
形 CJ1W-OD212	Ch1_Out00	出力 CH1 接点 00	DO1

処理内容

- PIDAT 命令の操作量「MV」を取得し、温調器への出力を制御します。温調器への出力は、ON または OFF の 2 とおりです。
- PIDAT 命令のサンプリング周期「InitSetParams.SampTime」の値は、100ms です。タスク周期は、100ms より十分に短いとします。したがって、「MV」の値は、ほぼ 100ms 周期で更新されます。
- 出力制御周期は 1s です。その間の出力制御値の ON 時間と OFF 時間を、時分割比例出力で制御します。
  たとえば、取得した「MV」の値が 20%だった場合、温調器への出力は、200ms の間 ON、続く 800ms の間 OFF とします。
  これを 1s 周期で繰り返します。



・最新の「MV」の値が、出力制御値を決めた際の「MV」の値より小さい場合、出力制御値は変化しません。
最新の「MV」の値が、出力制御値を決めた際の「MV」の値より大きい場合は、その値を直ちに出力制御値に反映します。たとえば、「MV」の値が 20%で出力制御値を決めたとします(ON200ms、OFF800ms)。
それから 100ms 後、新しい「MV」の値が 30%だった場合、出力制御値は直ちに ON300ms、OFF700ms に変化します。



・オートチューニングを実行し、「MV」の値が 100%になった場合は、制御周期に関係なく、直ちに出力制御値を ON にします。

グローバル変数の定義

● グローバル変数

名称	データ型	割付先 *1	コメント
AI1	INT	IOBus://rack#0/slot#0/Ch1_AIInPV	入力 No.1 測定値（INT 型）
DO1	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00	出力 CH1 接点 00

*1. 形 CJ1W-PH41U をラック番号 0 のスロット番号 0、形 CJ1W-OD212 をラック番号 0 のスロット番号 1 に装着した場合の割付先です。

（注）I/O マップ設定により、各ユニットのポートに割り付けたグローバル変数が自動的に生成されます。

LD

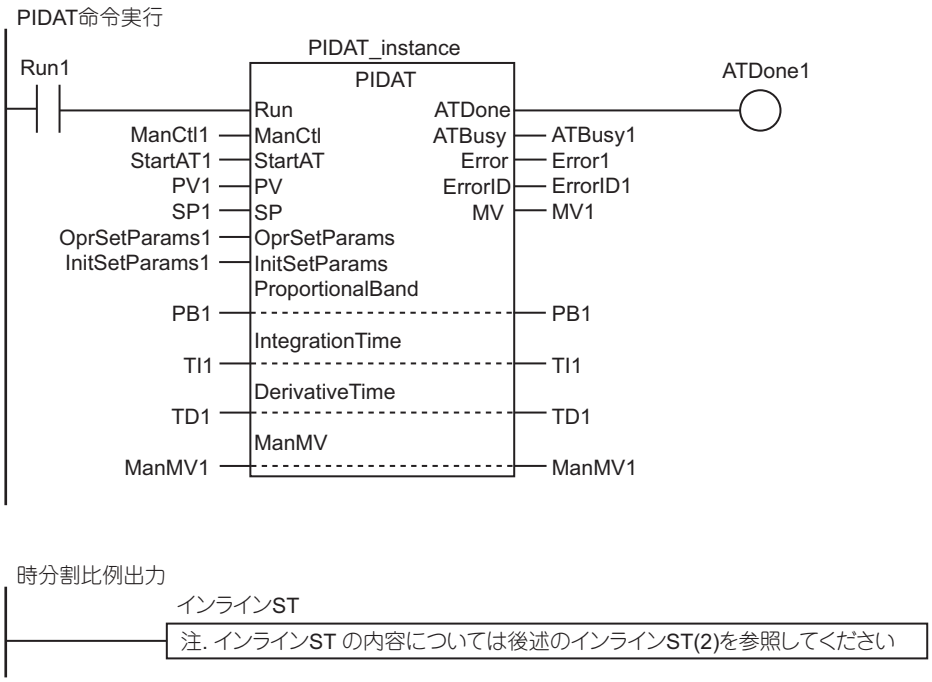
名称	データ型	初期値	保持	コメント
Run1	BOOL	FALSE	□	実行条件
ManCtl1	BOOL	FALSE	□	マニュアル／オート切替
StartAT1	BOOL	FALSE	□	オートチューニング実行条件
PV1	REAL	0.0	□	測定値（現在値）
SP1	REAL	90	□	目標値

名称	データ型	初期値	保持	コメント
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)	☐	運転中設定パラメータ
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)	☐	初期設定パラメータ
PB1	REAL	10	☑	比例帯
TI1	TIME	T#0s	☑	積分時間
TD1	TIME	T#0s	☑	微分時間
ManMV1	REAL	0.0	☐	マニュアル操作量
ATDone1	BOOL	FALSE	☐	オートチューニング正常終了
ATBusy1	BOOL	FALSE	☐	オートチューニング実行中
Error1	BOOL	FALSE	☐	異常
ErrorID1	WORD	16#0	☐	異常 ID
MV1	REAL	0.0	☐	操作量
PulseOnTime	TIME	T#0s	☐	制御出力 ON 時間
PulseCycTime	TIME	T#1s	☐	制御周期
ResetPulse	BOOL	FALSE	☐	タイマリセット
PIDAT_instance	PIDAT		☐	
TOF_instance	TOF		☐	
TON_instance	TON		☐	

現在値取得

インラインST

注. インラインST の内容については後述のインラインST(1)を参照してください



● インライン ST (1)の内容

```
PV1:=INT_TO_REAL(AI1)/REAL#10.0;  // 現在値 AI1 を実数に変換
// 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算
```

● インライン ST(2) の内容

```
// 出力制御値 ON 時間算出
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);
// TOF 命令で ON/OFF 切替
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);
// TON 命令でタイマリセット時間計測
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse);
// タイマリセット
IF (ResetPulse=BOOL#TRUE) THEN
  TOF_instance(In:=BOOL#TRUE);
  TON_instance(In:=BOOL#FALSE);
END_IF;
// オートチューニングで MV1=100%
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN
  DO1:=BOOL#TRUE;  // ならば、出力制御値を直ちに ON
END_IF;
```

ST

名称	データ型	初期値	保持	コメント
Run1	BOOL	FALSE	┌┐	実行条件
ManCtl1	BOOL	FALSE	┌┐	マニュアル／オート切替

名称	データ型	初期値	保持	コメント
StartAT1	BOOL	FALSE	☐	オートチューニング実行条件
PV1	REAL	0.0	☐	測定値（現在値）
SP1	REAL	90	☐	目標値
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)	☐	運転中設定パラメータ
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)	☐	初期設定パラメータ
PB1	REAL	10	☑	比例帯
TI1	TIME	T#0s	☑	積分時間
TD1	TIME	T#0s	☑	微分時間
ManMV1	REAL	0.0	☐	マニュアル操作量
ATDone1	BOOL	FALSE	☐	オートチューニング正常終了
ATBusy1	BOOL	FALSE	☐	オートチューニング実行中
Error1	BOOL	FALSE	☐	異常
ErrorID1	WORD	16#0	☐	異常 ID
MV1	REAL	0.0	☐	操作量
PulseOnTime	TIME	T#0s	☐	制御出力 ON 時間
PulseCycTime	TIME	T#1s	☐	制御周期
ResetPulse	BOOL	FALSE	☐	タイマリセット
PIDAT_instance	PIDAT		☐	
TOF_instance	TOF		☐	
TON_instance	TON		☐	

```
// 現在値 AI1 を実数に変換
// 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算
PV1:=INT_TO_REAL(AI1)/REAL#10.0;

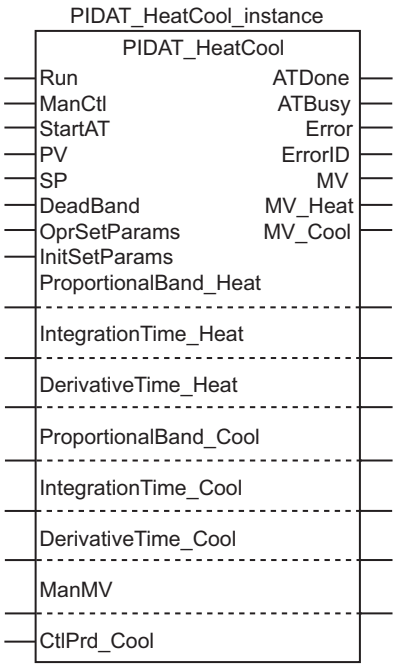
// PIDAT 命令実行
PIDAT_instance(
  Run :=Run1,
```

```
  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);

// 時分割比例出力
// 出力制御値 ON 時間算出
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);
// TOF 命令で ON/OFF 切替
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);
// TON 命令でタイマリセット時間計測
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse);
// タイマリセット
IF (ResetPulse=BOOL#TRUE) THEN
  TOF_instance(In:=BOOL#TRUE);
  TON_instance(In:=BOOL#FALSE);
END_IF;
// オートチューニングで MV1=100%
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN
  DO1:=BOOL#TRUE;  // ならば、出力制御値を直ちに ON
END_IF;
```

PIDAT_HeatCool

オートチューニング付きの加熱冷却 PID 演算（目標値フィルタ型 2 自由度 PID 制御） を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	FB		PIDAT_HeatCool_instance(Run, ManCtl, StartAT, PV, SP, DeadBand, OprSetParams, InitSetParams, ProportionalBand_Heat, IntegrationTime_Heat, DerivativeTime_Heat, ProportionalBand_Cool, IntegrationTime_Cool, DerivativeTime_Cool, ManMV, CtlPrd_Cool, ATDone, ATBusy, Error, ErrorID, MV, MV_Heat, MV_Cool);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Run	実行条件	入力	TRUE: 実行 FALSE: 停止	データ型に従う	–	FALSE
ManCtl	マニュアル／オート切替		TRUE: マニュアル運転 FALSE: オート運転			
StartAT	オートチューニング実行条件		TRUE: 実行 FALSE: 中止			
PV	測定値（現在値）		測定値（現在値）	*1		0
SP	目標値		目標値			
DeadBand	デッドバンド		デッドバンドまたはオーバーラップバンド設定	-320.0 ～ 320.0	%	
OprSetParams	運転中設定パラメータ		運転中の、各種設定パラメータ	–	–	–
InitSetParams	初期設定パラメータ		初期設定パラメータ			
CtlPrd_Cool	冷却操作の制御周期		「MV_Cool」を時分割比例出力する場合の制御周期	T#0.1s ～ T#100s		T#20s

	名称	入／出力	内容	有効範囲	単位	初期値
ProportionalBand_Heat	加熱操作の比例帯	入出力	加熱操作の比例帯	0.01 ～ 1000.00	%FS	－
IntegrationTime_Heat	加熱操作の積分時間		加熱操作の積分時間 値が大きいほど積分効果小。 0 の場合は、積分動作なし。	T#0.0000s ～ T#10000.0000s *2	s	
DerivativeTime_Heat	加熱操作の微分時間		加熱操作の微分時間 値が大きいほど微分効果大。 0 の場合は、微分動作なし。	T#0.0000s ～ T#10000.0000s *2		
ProportionalBand_Cool	冷却操作の比例帯		冷却操作の比例帯	0.01 ～ 1000.00	%FS	
IntegrationTime_Cool	冷却操作の積分時間		冷却操作の積分時間 値が大きいほど積分効果小。 0 の場合は、積分動作なし。	T#0.0000s ～ T#10000.0000s *2	s	
DerivativeTime_Cool	冷却操作の微分時間		冷却操作の微分時間 値が大きいほど微分効果大。 0 の場合は、微分動作なし。	T#0.0000s ～ T#10000.0000s *2		
ManMV	マニュアル操作量		マニュアル操作量	-320 ～ 320	%	
ATDone	オートチューニング正常終了	出力	TRUE: 正常終了 FALSE: *3	データ型に従う	－	－
ATBusy	オートチューニング実行中		TRUE: 実行中 FALSE: 実行中ではない			
MV	操作量		操作量		%	
MV_Heat	加熱操作の操作量		加熱操作の操作量	0 ～ 320		
MV_Cool	冷却操作の操作量		冷却操作の操作量	0 ～ 320		

*1. 入力レンジ下限「InitSetParams.RngLowLmt」の値 ～ 入力レンジ上限「InitSetParams.RngUpLmt」の値。
*2. 0.0001s 未満は、切り捨て。
*3. 異常終了、またはオートチューニングを実行せずに PID 演算中、または PID 演算中ではない。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Run	○																			
ManCtl	○																			
StartAT	○																			
PV														○						
SP														○						
DeadBand														○						
OprSetParams	構造体_sOPR_SET_PARAMS　詳細は「構造体の仕様 (P.2-703)」参照																			
InitSetParams	構造体_sINIT_SET_PARAMS　詳細は「構造体の仕様 (P.2-703)」参照																			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CtlPrd_Cool																○				
ProportionalBand_Heat														○						
IntegrationTime_Heat																○				
DerivativeTime_Heat																○				
ProportionalBand_Cool														○						
IntegrationTime_Cool																○				
DerivativeTime_Cool																○				
ManMV														○						
ATDone	○																			
ATBusy	○																			
MV														○						
MV_Heat														○						
MV_Cool														○						

機能

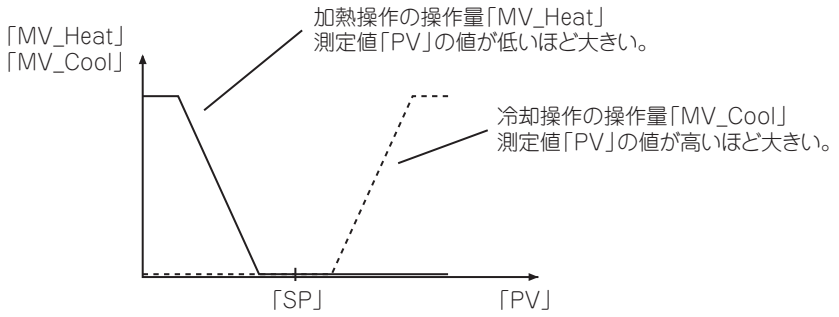
加熱冷却 PID 演算を行い、温度調節器などの操作量を制御します。
実行条件「Run」の値が TRUE になったら、加熱冷却 PID 演算を開始します。その後、「Run」の値が TRUE である間、周期的に、測定値「PV」の取得→加熱冷却 PID 演算→加熱操作の操作量「MV_Heat」と冷却操作の操作量「MV_Cool」の出力を繰り返します。
「Run」の値が FALSE になったら、加熱冷却 PID 演算を終了します。

オートチューニング機能を持っており、加熱操作および冷却操作の PID 定数の、それぞれの最適値を自動的に求めることができます。
オートチューニング実行条件「StartAT」の値が TRUE になったら、加熱操作および冷却操作の PID 定数のオートチューニングを実行します。

PIDAT_HeatCool 命令と PIDAT 命令との違い

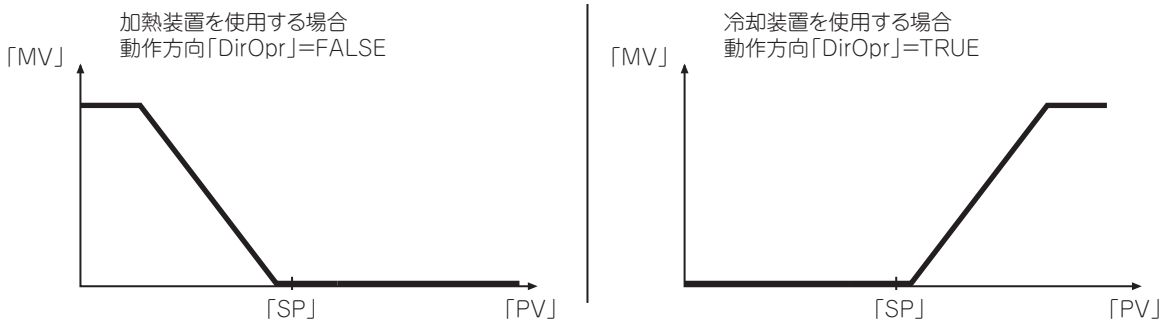
● PIDAT_HeatCool 命令

PIDAT_HeatCool 命令は、加熱装置と冷却装置の両方を用いて温度調節をする場合に使用する命令です。したがって、加熱操作の操作量「MV_Heat」と、冷却操作の操作量「MV_Cool」の、2 種類の操作量が出力されます。また、オートチューニング機能により、加熱操作の PID 定数の最適値と、冷却操作の PID 定数の最適値が、それぞれ求められます。



● **PIDAT 命令**

PIDAT 命令は、加熱装置と冷却装置のどちらかひとつを用いて温度調節をする場合に使用する命令です。したがって、出力される操作量「MV」は、1 種類だけです。また、加熱装置に対して操作量を出力するのか、冷却装置に対して操作量を出力するのかを決める、動作方向「DirOpr」というパラメータが存在します。PIDAT_HeatCool 命令では、「DirOpr」は使用しません。

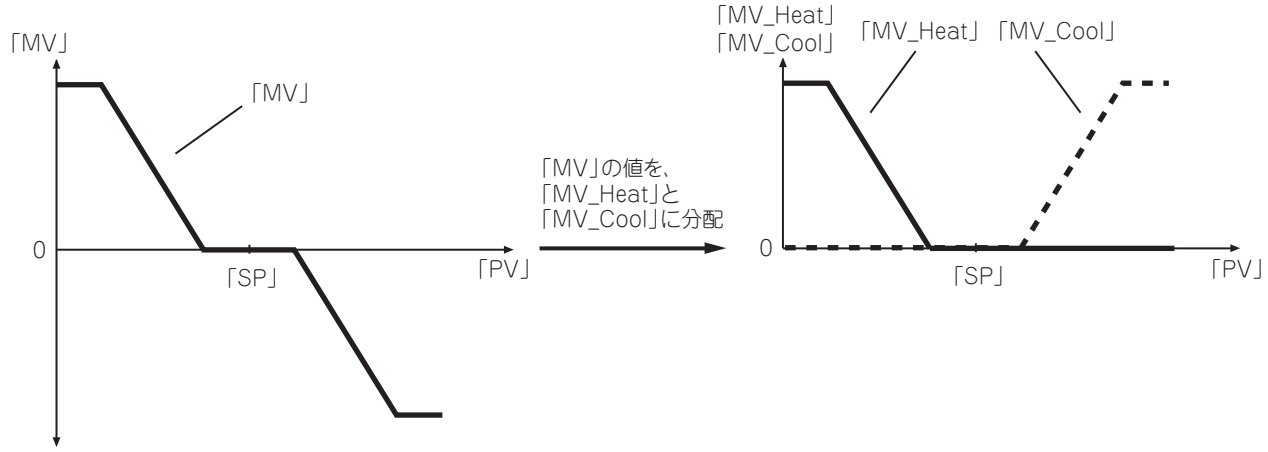


操作量「MV」と加熱装置の操作量「MV_Heat」、冷却装置の操作量「MV_Cool」

PID 命令と同様に、加熱装置と冷却装置のどちらかひとつを用いて温度調節をする場合の操作量が「MV」です。
本命令でも、まず PIDAT 命令と同じ方法で、「MV」が求められます。そして、「MV」を加熱装置の操作量と冷却装置の操作量に分配したものが、「MV_Heat」と「MV_Cool」です。

以下に、「MV」の値を「MV_Heat」と「MV_Cool」に分配する概念図を示します。

「MV_Cool」の値は、「MV」の負の値の絶対値になります。



上図は概念図です。実際の「MV_Heat」、「MV_Cool」の値は、「MV」の値の正数部分、負数部分と完全には一致しません。

「MV_Heat」、「MV_Cool」の値は、「MV」の値をもとに、特別な計算式を用いて求められます。

構造体の仕様

運転中設定パラメータ［OprSetParams］のデータ型は、構造体_sOPR_SET_PARAMS です。
仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
OprSetParams	運転中設定パラメータ	運転中の各種設定パラメータ	_sOPR_SET_PARAMS	—	—	—
MVLowLmt	操作量リミット下限値	「MV_Heat」、「MV_Cool」をリミット制御する下限値	REAL	-320 ～ 320 *1	%	-100
MVUpLmt	操作量リミット上限値	「MV_Heat」、「MV_Cool」をリミット制御する上限値	REAL			100
ManResetVal	マニュアルリセット値	使用しない	REAL	-320 ～ 320		0
MVTrackSw	MV トラッキングスイッチ	MV トラッキングスイッチ TRUE: ON FALSE: OFF	BOOL	データ型に従う	—	FALSE
MVTrackVal	MV トラッキング値	MV トラッキング時に、「MV」にセットされる値	REAL	-320 ～ 320	%	0
StopMV	ストップ時操作量	命令実行停止時に、「MV」にセットされる値	REAL			
ErrorMV	異常時操作量	異常発生時に、「MV」にセットされる値	REAL			
Alpha	2 自由度 PID パラメータ α	目標値フィルタ係数 α 0 の場合、目標値フィルタ無効	REAL	0.00 ～ 1.00	—	0.65
ATCalcGain	オートチューニング算出ゲイン	オートチューニング結果の調整係数。 値が大きいほど安定性重視、小さいほど即応性重視。	REAL	0.1 ～ 10.0		0.8
ATHystrs	オートチューニングヒステリシス	リミットサイクルのヒステリシスの大きさ	REAL	0.01 ～ 10.0	%FS	0.05

*1. ただし、「MVLowLmt」＜「MVUpLmt」

初期設定パラメータ［InitSetParams］のデータ型は、構造体_sINIT_SET_PARAMS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
InitSetParams	初期設定パラメータ	初期設定パラメータ	_sINIT_SET_PARAMS	–	–	–
SampTime	サンプリング周期	PID 演算を行う周期	TIME	T#0.0001s ～ T#100.0000s	s	T#0.05s
RngLowLmt	入力レンジ下限	「PV」および「SP」の下限値	REAL	-32000 ～ 32000 *1	–	0
RngUpLmt	入力レンジ上限	「PV」および「SP」の上限値	REAL			100
DirOpr	動作方向	使用しない	BOOL	データ型に従う		FALSE

*1. ただし、「RngLowLmt」＜「RngUpLmt」

各変数の意味

本命令の各変数の意味を、詳しく説明します。

● Run（実行条件）

本命令の実行条件です。
値が TRUE である間、加熱冷却 PID 演算を行います。値が FALSE になったら、加熱冷却 PID 演算を停止します。

● ManCtl（マニュアル／オート切替）

本命令には、マニュアル運転とオート運転の 2 つのモードがあります。
「ManCtl」の値によって、どちらのモードでの運転かが決まります。

「ManCtl」の値	運転モード	「MV」の値
TRUE	マニュアル	「ManMV」の値 加熱冷却 PID 演算は行いません。
FALSE	オート	加熱冷却 PID 演算によって算出した値

● StartAT（オートチューニング実行条件）

PID 定数のオートチューニングの、実行条件です。
「StartAT」の値が TRUE の状態で、「Run」の値が FALSE→TRUE になると、運転開始時にオートチューニングを実行します。
「Run」の値が TRUE で加熱冷却 PID 演算中に、「StartAT」の値が FALSE→TRUE になると、運転中にオートチューニングを開始します。
いずれの場合も、オートチューニング中に「StartAT」の値が TRUE→FALSE になったら、オートチューニングを中止します。
オートチューニング機能の詳細は、「オートチューニング (P.2-712)」を参照してください。

● PV（測定値）

制御対象の測定値です。

● SP（目標値）

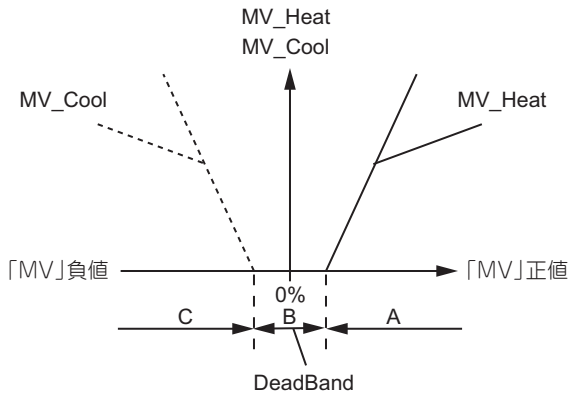
制御対象の目標値です。

● DeadBand（デッドバンド）

「MV」の値を、どのように「MV_Heat」と「MV_Cool」に分配するかを決めるのが、「DeadBand」です。

「DeadBand」の値は、「MV」=0 を中心にして、加熱操作も冷却操作も行わない「MV」の値の幅を示します。
「MV」の値と、「MV_Heat」、「MV_Cool」の値の関係は、以下のようになります。

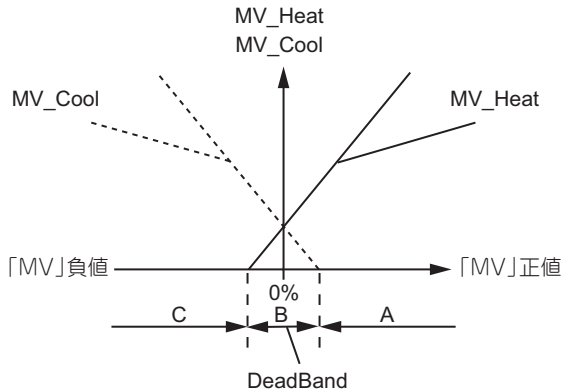
「MV」の値	「MV_Heat」の値	「MV_Cool」の値
デッドバンドより大（領域 A）	正値。「MV」の値が大きいほど大。	0
デッドバンド内（領域 B）	0	0
デッドバンドより小（領域 C）	0	正値。「MV」の値が小さいほど大。



「MV」の値がこの範囲内にある場合、「MV_Heat」と「MV_Cool」の値が、0%になります。

「DeadBand」には、負の値を設定することも可能です。
「DeadBand」の値が負の場合は、「MV」の値がデッドバンド内にあると、加熱操作と冷却操作の両方が行われます。
「DeadBand」の値が負の場合の、「MV」の値と「MV_Heat」、「MV_Cool」の関係は、以下のようになります。

「MV」の値	「MV_Heat」の値	「MV_Cool」の値
デッドバンドより大（領域 A）	正値。「MV」の値が大きいほど大。	0
デッドバンド内（領域 B）	正値。「MV」の値が大きいほど大。	正値。「MV」の値が小さいほど大。
デッドバンドより小（領域 C）	0	正値。「MV」の値が小さいほど大。



「MV」の値がこの範囲内にある場合、「MV_Heat」と「MV_Cool」の両方の値が、正になります。

● MVLowLmt（操作量リミット下限値）、MVUpLmt（操作量リミット上限値）

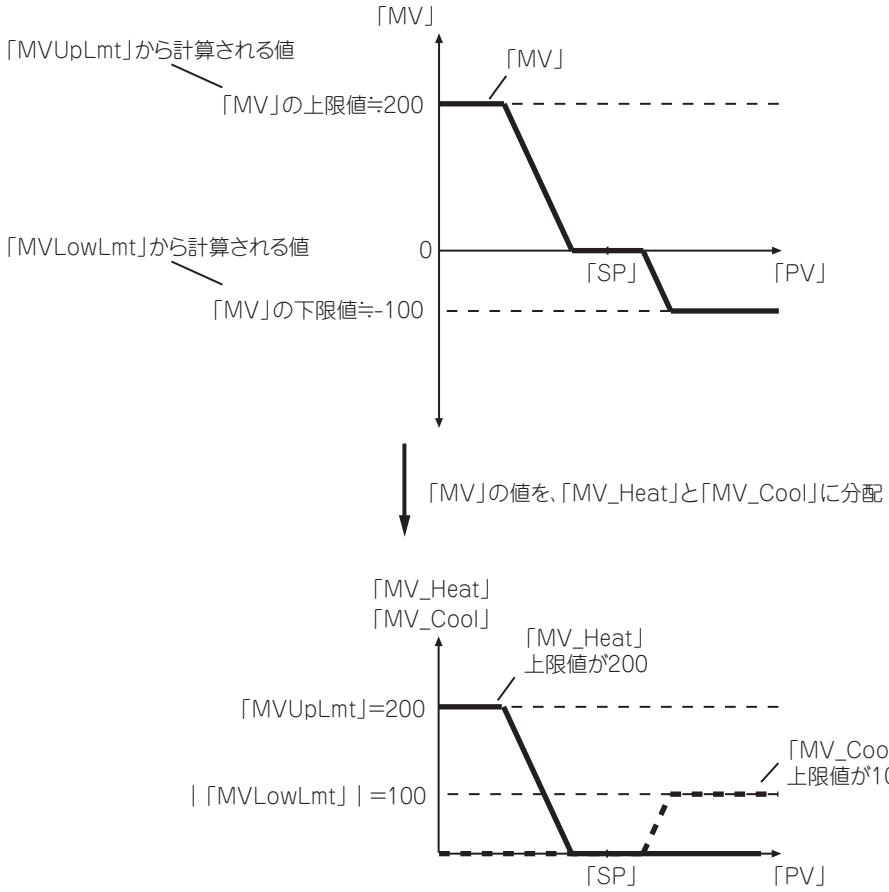
「MV_Heat」、「MV_Cool」の値は、リミット制御することができます。
「MV_Heat」と「MV_Cool」の下限値と上限値を決めるのが、「MVLowLmt」と「MVUpLmt」です。

「MV_Heat」と「MV_Cool」の値は、以下の手順で求められます。

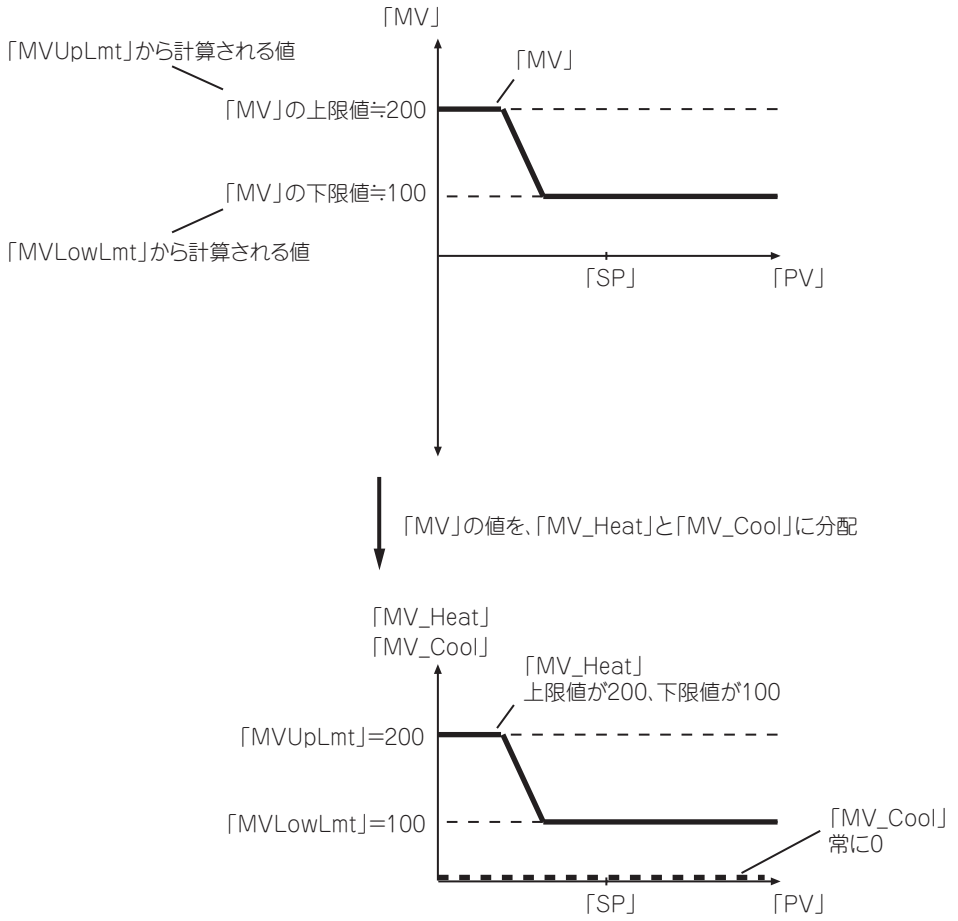
1 加熱冷却 PID 演算により、「MV」が求められます。「MV」の上限値と下限値は、「MVLowLmt」と「MVUpLmt」から、特別な計算式で求められます。

2 「MV」を分配して、「MV_Heat」と「MV_Cool」が求められます。

「MVLowLmt」=-100、「MVUpLmt」=200 の場合の、「MV」と「MV_Heat」、「MV_Cool」の関係を、以下の図に示します。
求められる「MV_Heat」は、上限値が 200、下限値が 0 になり、「MV_Cool」は、上限値が 100、下限値が 0 になります。つまり、「MV_Heat」の上限値が「MVUpLmt」の値と一致し、「MV_Cool」の上限値が「MVLowLmt」の絶対値と一致します。



「MVLowLmt」=100、「MVUpLmt」=200 の場合の、「MV」と「MV_Heat」、「MV_Cool」の関係を、以下の図に示します。
求められる「MV_Heat」は、上限値が 200、下限値が 100 になり、「MV_Cool」の値は、常に 0 になります。つまり、「MV_Heat」の上限値と下限値が、「MVUpLmt」と「MVLowLmt」のそれぞれと一致します。



上記のように、「MVLowLmt」と「MVUpLmt」が正数か負数かで、「MV_Heat」と「MV_Cool」の上限値、下限値は、以下のように変わります。

「MVLowLmt」の値	「MVUpLmt」の値	MV_Heat		MV_Cool	
		下限値	上限値	下限値	上限値
正	正	MVLowLmt	MVUpLmt	0	0
負	正	0	MVUpLmt	0	「MVLowLmt」の絶対値
負	負	0	0	「MVUpLmt」の絶対値	「MVLowLmt」の絶対値

「MVLowLmt」、「MVUpLmt」は、必ず「MVLowLmt」＜「MVUpLmt」となるように、設定してください。
また、「MV」に、ストップ時操作量「StopMV」、異常時操作量「ErrorMV」、マニュアル操作量「ManMV」のいずれかがセットされる場合は、リミット制御の対象外になります。

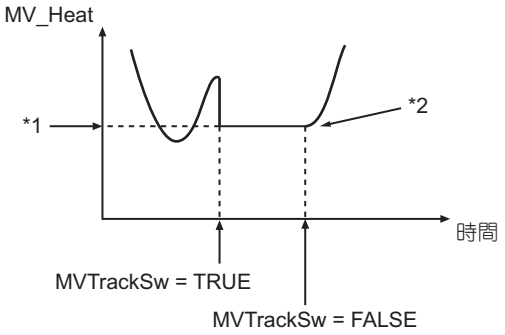
「MVLowLmt」、「MVUpLmt」は、本命令の制御状態が運転中（オートチューニング非実行）であっても変更することができます。
ただし、運転中に「MVLowLmt」、「MVUpLmt」を拡大方向に変更すると、そのサンプリング周期では前回と同じ値の「MV_Heat」あるいは「MV_Cool」を出力し、バンプレス（急変なし）となります。
このため、「MVLowLmt」、「MVUpLmt」を繰り返し変更すると制御性能に影響をおよぼし、十分な制御性能を得られない場合があります。
運転中に「MVLowLmt」、「MVUpLmt」を繰り返し変更する場合は、制御性能に対する影響を確認の上、使用してください。

● **ManResetVal（マニュアルリセット値）**

本命令では使用しません。値を設定しても、無視されます。

● **MVTrackSw（MV トラッキングスイッチ）**

オート運転時、「MV」に、外部からの入力値（MV トラッキング値）「MVTrackVal」を強制的にセットする機能を、MV トラッキングと呼びます。
「MVTrackSw」の値が TRUE である間、MV トラッキングを実行します。
「MVTrackSw」の値が FALSE になると、「MV」の値は、そのサンプリング周期では「MVTrackVal」の値をそのまま引き継ぎ、次のサンプリング周期から加熱冷却 PID 演算結果に戻ります。
これにより、「MV_Heat」や「MV_Cool」の値が急変することを抑えます。



*1.「MVTrackVal」から分配された「MV_Heat」
*2.「MV」の値は、「MVTrackVal」の値を引き継ぎます。

● **MVTrackVal（MV トラッキング値）**

MV トラッキングで、「MV」に強制的にセットする値です。
「MVTrackVal」の値は、「MVLowLmt」と「MVUpLmt」により、リミット制御されます。

● **StopMV（ストップ時操作量）**

「Run」の値が FALSE になり、本命令を実行停止した際に、「MV」にセットされる値です。

● **ErrorMV（異常時操作量）**

異常が発生した（「Error」の値が TRUE）際に、「MV」にセットされる値です。
「ErrorMV」の値が、有効範囲内（-320～320）でない場合、異常時の「MV」の値は 0 になります。

● **Alpha（2 自由度 PID パラメータ α）**

目標値フィルタの係数を決めるパラメータです。
詳細は、PIDAT 命令の、「目標値フィルタ型 2 自由度 PID 制御 (P.2-682)」を参照してください。
「Alpha」の値は、通常 0.65 に設定してください。

● **ATCalcGain（オートチューニング算出ゲイン）**

オートチューニング実行で算出した PID 定数を、実際の PID 定数に反映させる際の係数です。
1.00 を指定すると、オートチューニングの結果をそのまま反映することになります。
「ATCalcGain」の値は、安定性を重視するときには大きくし、即応性を重視するときには小さくします。

● **ATHystrs（オートチューニングヒステリシス）**

オートチューニングの際の、リミットサイクルヒステリシスの大きさを表します。

「ATHystrs」の値は、小さい方が高精度のチューニングができます。ただし、測定値が安定せず正常なチューニングが困難な場合には、大きくします。
詳細は、PIDAT 命令の、「オートチューニング (P.2-685)」を参照してください。

- **SampTime（サンプリング周期）**

  加熱冷却 PID 演算を行う周期の最小値です。
  詳細は、「加熱冷却 PID 演算の実行タイミング (P.2-713)」を参照してください。
  加熱冷却 PID 演算を実行してからの経過時間が「SampTime」未満では、つぎの加熱冷却 PID 演算が行われることはありません。

- **RngLowLmt（入力レンジ下限）、RngUpLmt（入力レンジ上限）**

  「PV」および「SP」の下限値と上限値です。
  「PV」および「SP」に接続するパラメータの値が、この範囲からはずれている場合、異常になります。
  必ず、「RngLowLmt」＜「RngUpLmt」となるように、設定してください。

- **DirOpr（動作方向）**

  本命令では使用しません。値を設定しても、無視されます。

- **CtlPrd_Cool（制御周期）**

  本命令と「TimeProportionalOut (P.2-733)」命令を組み合わせて、「MV_Cool」を時分割比例出力する場合の制御周期です。TimeProportionalOut 命令の制御周期「CtlPrd」と同じ値を設定してください。
  「MV_Cool」を時分割比例出力しない場合は、初期値 T#20s を設定してください。

- **ProportionalBand_Heat、ProportionalBand_Cool（比例帯）**

  3 つある PID 定数のうちの 1 つです。詳細は、PIDAT 命令の、「比例動作(P) (P.2-679)」を参照してください。
  「ProportionalBand_Heat」、「ProportionalBand_Cool」の値が大きいと、オフセットが大きくなります。小さいとハンティングが発生します。

- **IntegrationTime_Heat、IntegrationTime_Cool（積分時間）**

  3 つある PID 定数のうちの 1 つです。詳細は、PIDAT 命令の、「積分動作(I) (P.2-681)」を参照してください。
  「IntegrationTime_Heat」、「IntegrationTime_Cool」の値が大きいほど、積分動作が弱くなります。

- **DerivativeTime_Heat、DerivativeTime_Cool（微分時間）**

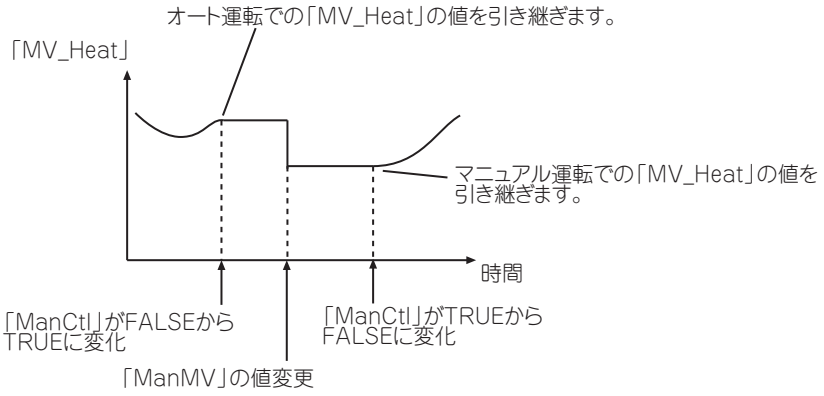
  3 つの PID 定数のうちの 1 つです。詳細は、PIDAT 命令の、「微分動作(D) (P.2-681)」を参照してください。
  「DerivativeTime_Heat」、「DerivativeTime_Cool」の値が大きいほど、微分動作が強くなります。

- **ManMV（マニュアル操作量）**

  マニュアル運転時（「ManCtl」=TRUE）に、「MV」にセットされる値です。
  ただし、「ManMV」の値が「MV」にセットされるのは、マニュアル運転に切り替えた後に、「ManMV」の値を変更したときです。

  オート運転からマニュアル運転に切り替えた直後の「MV」の値は、オート運転での「MV_Heat」の値が正値なら「MV_Heat」の値、それ以外なら「MV_Cool」の値になります。

また、マニュアル運転からオート運転に切り替えた直後の「MV」の値も、マニュアル運転での「MV_Heat」の値が正値なら「MV_Heat」の値、それ以外なら「MV_Cool」の値になります。
なお、「ManMV」の値は、「MVLowLmt」と「MVUpLmt」の間でなくてもかまいません。



- **ATDone（オートチューニング正常終了）**

  オートチューニングが、正常終了したことを示すフラグです。
  オートチューニングが正常終了してから、「StartAT」の値が TRUE である間、TRUE になります。
  FALSE になるのは、以下の場合です。
  - オートチューニング異常終了時。
  - オートチューニング実行中（「ATBusy」の値が TRUE）。
  - オートチューニングを実行せずに、加熱冷却 PID 演算中。
  - 加熱冷却 PID 演算中ではない（「Run」の値が FALSE）。
  - 「StartAT」の値が FALSE である。

- **ATBusy（オートチューニング実行中）**

  オートチューニングが、実行中であることを示すフラグです。
  オートチューニング実行中に、TRUE になります。それ以外は、FALSE になります。

- **MV（操作量）**

  加熱冷却 PID 演算によって求められる操作量です。「MV」を分配することで、「MV_Heat」と「MV_Cool」が求められます。

- **MV_Heat（加熱操作の操作量）**

  加熱装置に与える操作量です。

- **MV_Cool（冷却操作の操作量）**

  冷却装置に与える操作量です。

加熱冷却 PID 演算

PID 演算の詳細は、「PIDAT (P.2-672)」命令を参照してください。
加熱冷却 PID 演算は、加熱操作の PID 定数と冷却操作の PID 定数を用いて、操作量を求める演算です。
1 回前の演算結果が「MV」≦0 場合は加熱操作の PID 定数で演算を行い、「MV」>0 の場合は冷却操作の PID 定数で演算を行います。

比例動作(P)、積分動作(I)、微分動作(D)

比例動作(P)、積分動作(I)、微分動作(D)の詳細は、PIDAT 命令の「比例動作(P)、積分動作(I)、微分動作(D) (P.2-679)」を参照してください。

目標値フィルタ型 2 自由度 PID 制御

目標値フィルタ型 2 自由度 PID 制御の詳細は、PIDAT 命令の「目標値フィルタ型 2 自由度 PID 制御 (P.2-682)」を参照してください。

運転開始の手順

本命令は、適正な PID 定数を使って運転する必要があります。そのための運転開始には、PID 定数の最適値がわかっている場合と、わからない場合の 2 つの方法があります。

PID 定数は、運転中に変更することが可能です。また、運転中にオートチューニングを実行することも可能です。その場合は、「StartAT」の値を、TRUE にします。

● PID 定数の最適値がわからない場合

PID 定数の最適値がわからない場合、運転開始時にオートチューニングを行い、PID 定数の最適値を求めます。
「StartAT」の値を TRUE にした状態で、「Run」の値を FALSE から TRUE に変化させます。
まず初めにオートチューニングを実行し、求めた PID 定数を使って加熱冷却 PID 演算を開始します。

● PID 定数の最適値がわかっている場合

PID 定数の最適値を、「ProportionalBand_Heat」、「IntegrationTime_Heat」、「DerivativeTime_Heat」および、「ProportionalBand_Cool」、「IntegrationTime_Cool」、「DerivativeTime_Cool」に設定して、「Run」の値を FALSE から TRUE に変化させます。
「ProportionalBand_Heat」、「IntegrationTime_Heat」、「DerivativeTime_Heat」および「ProportionalBand_Cool」、「IntegrationTime_Cool」、「DerivativeTime_Cool」は、入出力変数です。定数を入力パラメータとすることはできません。必ず、適当な変数を定義して、そこに値を代入したものを入力パラメータとしてください。

制御状態と操作量

操作量「MV」は、制御状態によって、以下のように決まります。

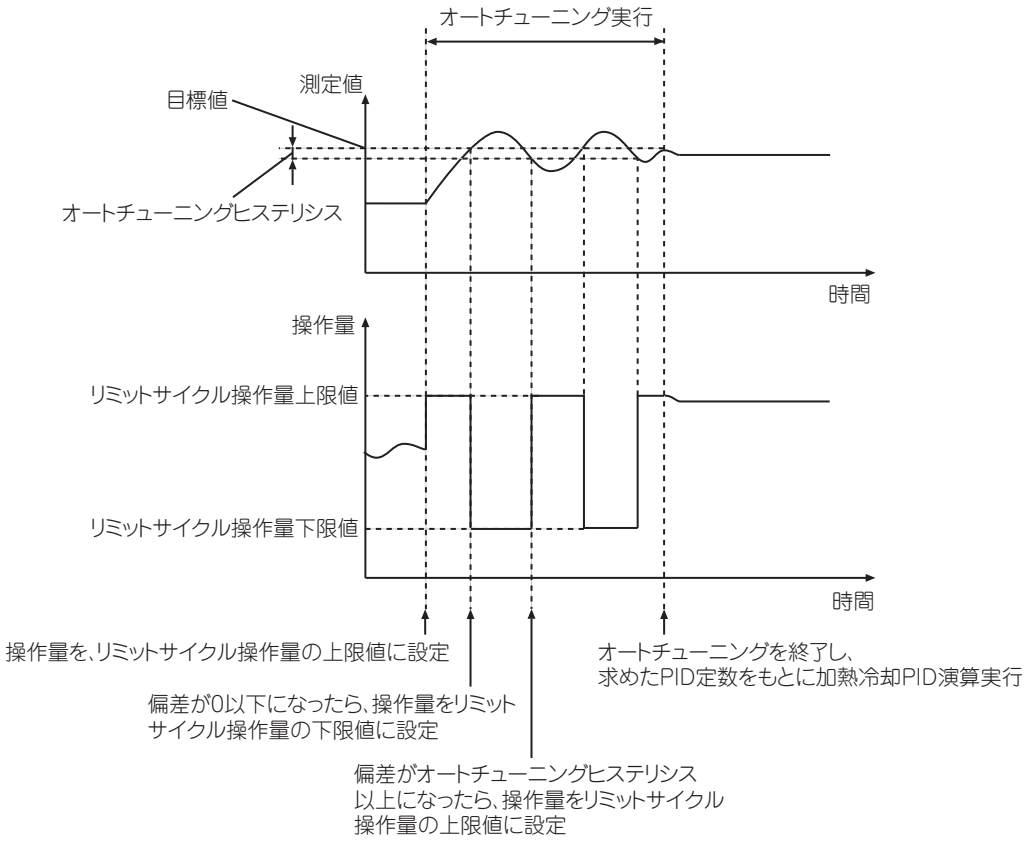
制御状態	変数の値					「MV」 操作量
	「ManCtl」 マニュアル／オート切替	「Run」 実行条件	「Error」 異常終了	「MVTrack Sw」 MV トラッキングスイッチ	「ATBusy」 オートチューニング実行中	
異常終了	FALSE	TRUE	TRUE	－	FALSE	「ErrorMV」 異常時操作量
オート運転中（MV トラッキング）			FALSE	TRUE		「MVTrackVal」 MV トラッキング値
オート運転中（オートチューニング実行中）				FALSE	TRUE	操作量リミット上限値と、操作量リミット下限値を繰り返し
オート運転中（オートチューニング非実行）					FALSE	その時点での PID 定数によって算出された値
命令実行停止		FALSE	－	－		「StopMV」*1 ストップ時操作量
マニュアル運転中	TRUE	－				「ManMV」*2 マニュアル操作量

*1. 「StopMV」の値が有効範囲外の場合、「MV」の値は 0 になります。
*2. 「ManMV」の値が有効範囲外の場合、「MV」の値は 0 になります。

オートチューニング

2 自由度 PID パラメータ α を調整することはあまりないため、本命令で主に調整する設定値は、PID 定数となります。
本命令には、PID 定数をオートチューニングする機能があります。
オートチューニングの方法には、リミットサイクル法を使います。
リミットサイクル法とは、操作量を強制的に、一時的にリミットサイクル操作量の上限値および下限値にして、その際の測定値の変化のしかたから、PID 定数の最適値を求める方法です。

オートチューニング実行指示があると、操作量を、まずリミットサイクル操作量の上限値に設定します。その後、偏差が 0 以下になったら、操作量をリミットサイクル操作量の下限値に設定します。そして、偏差がオートチューニングヒステリシスの値以上になったら、操作量を、またリミットサイクル操作量の上限値に設定します。これを、2 回半繰り返して、PID 定数の最適値を算出します。
リミットサイクル操作量の上限値と下限値は、各パラメータの値から算出します。



オートチューニングは、「Run」の値が TRUE で加熱冷却 PID 演算中に、「StartAT」の値が FALSE →TRUE になると、運転中に実行されます。または、「StartAT」の値が TRUE の状態で、「Run」の値が FALSE→TRUE になると、運転開始時に実行されます。
オートチューニングが正常終了すると、算出された PID 定数の最適値が、直ちに反映されます。

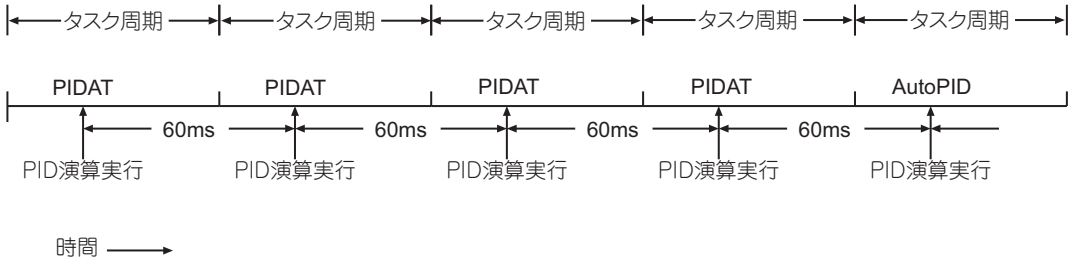
オートチューニング中（「ATBusy」=TRUE）に「StartAT」の値を FALSE にすると、オートチューニングを中止します。この場合、オートチューニング開始直前の PID 定数で、加熱冷却 PID 演算を再開します。

加熱冷却 PID 演算の実行タイミング

加熱冷却 PID 演算は、周期的に繰り返し実行されます。加熱冷却 PID 演算が実行されるのは、ユーザプログラム中で本命令が実行されるときです。
ただし、前回の加熱冷却 PID 演算実行からの経過時間が、サンプリング周期「SampTime」未満の場合は実行されません。
また、前回の加熱冷却 PID 演算実行からの経過時間が「SampTime」以上の場合は実行され、余った時間（前回の実行からの経過時間－「SampTime」）が、次回に繰り越されます。詳しくは、下図を参照してください。
PrgStop 命令や MC 命令によって本命令を実行させなかった場合でも、下図の「PID 演算実行」のタイミングで、前回の加熱冷却 PID 演算実行からの経過時間は 0 になります。

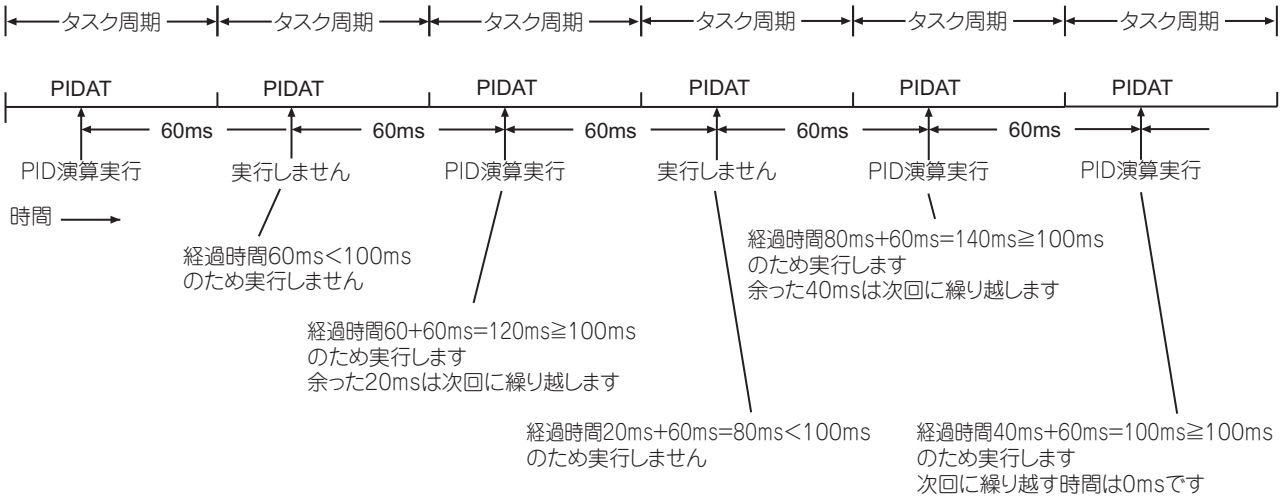
タスク周期=60ms 「SampTime」<60msの場合

タスク周期≧「SampTime」なので、1タスク周期に1回、必ず実行されます。



タスク周期=60ms 「SampTime」=100msの場合

タスク周期<「SampTime」なので、実行されない場合があります。

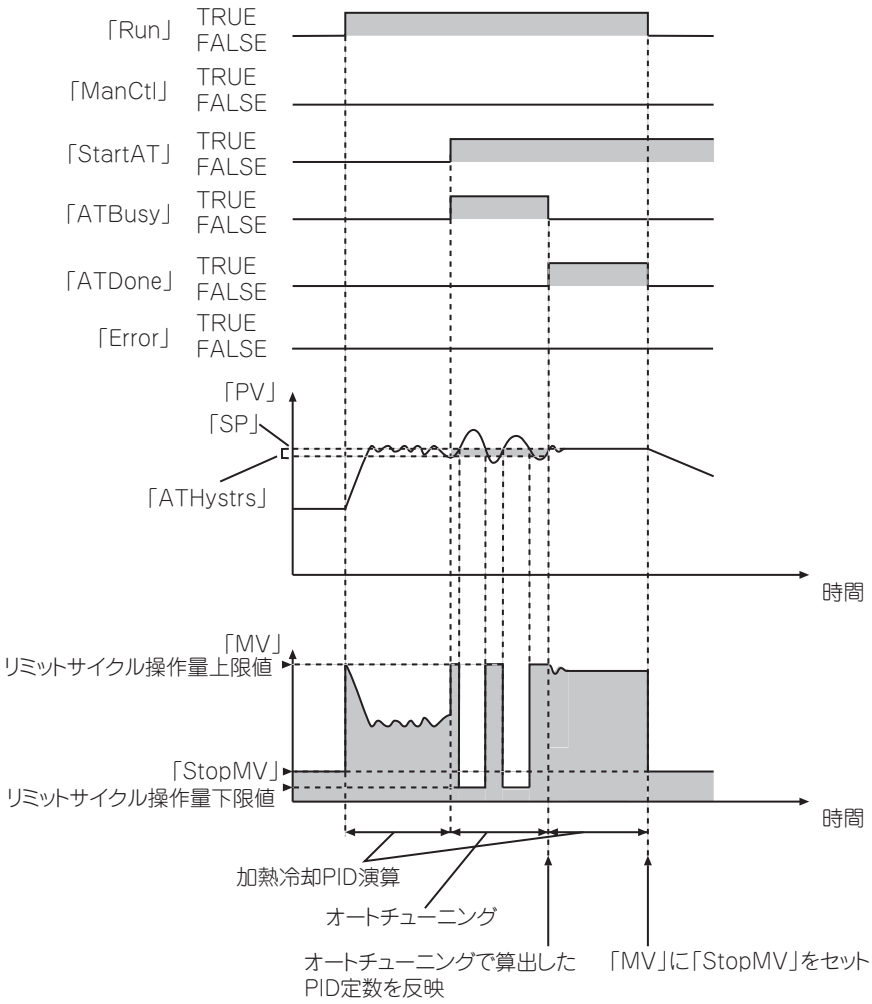


タイミングチャート

いくつかの場合の、各変数のタイミングチャートを示します。

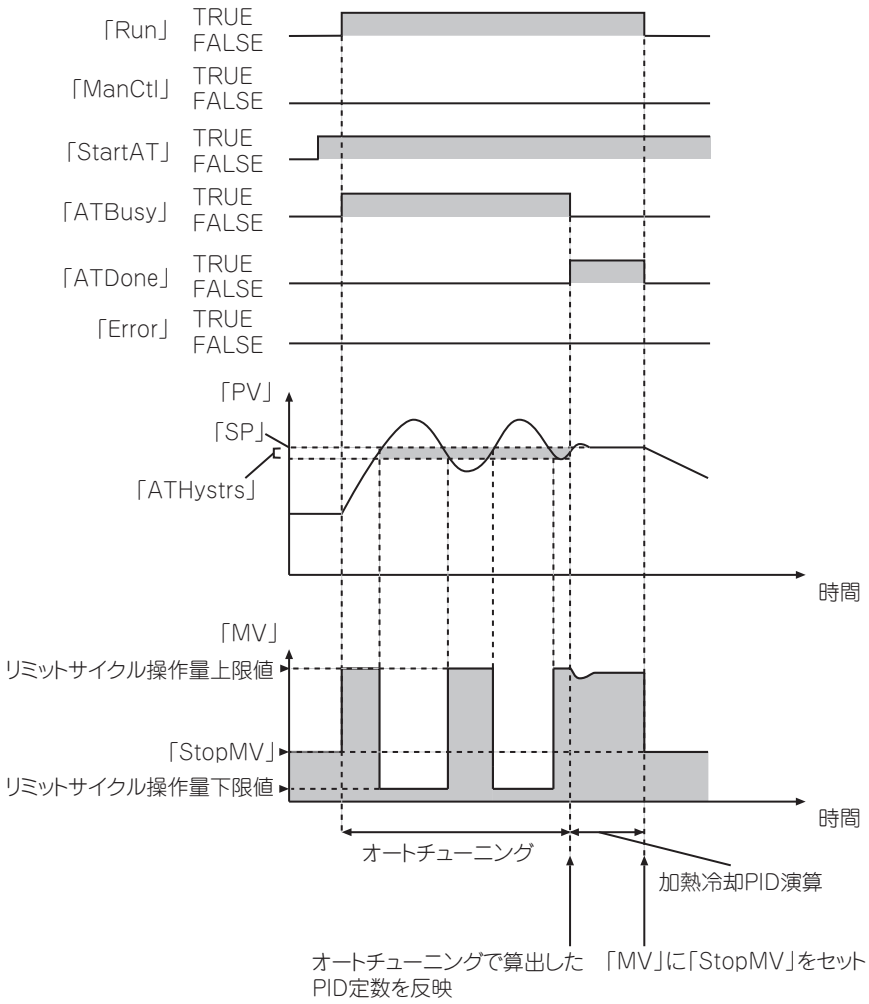
● オート運転中に、オートチューニングを実行した場合

- 下図の場合、「ManCtl」の値が FALSE なので、「Run」の値が FALSE の間、「MV」の値は「StopMV」になります。
- 「Run」の値を TRUE にすると、PID 定数に基づいた「MV」が出力されます。
- 「StartAT」の値を TRUE にすると、オートチューニングが実行されます。「ATBusy」の値が TRUE になります。
- オートチューニングが終了すると、「ATBusy」の値が FALSE になり、「ATDone」の値が TRUE になります。
- オートチューニング終了後は、オートチューニングで求められた PID 定数に基づいた「MV」が出力されます。
- 「Run」の値を FALSE にすると、「MV」の値は「StopMV」になります。また、「ATDone」の値が FALSE になります。



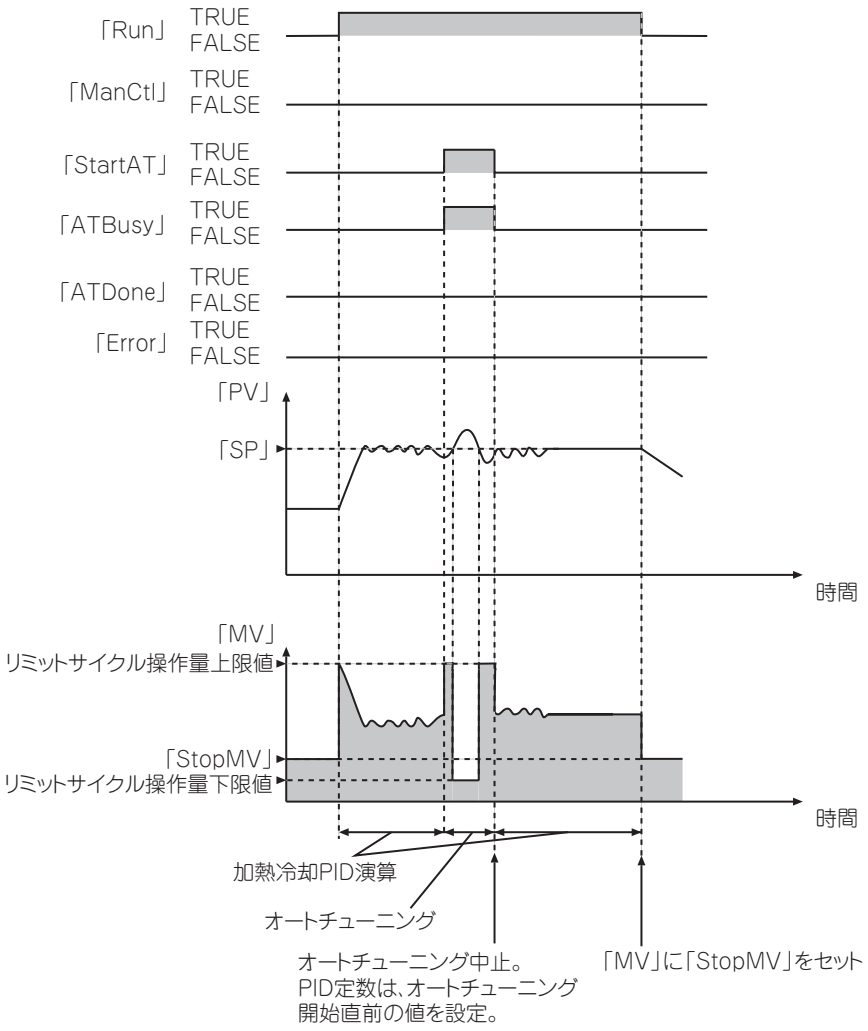
● 本命令実行の最初に、オートチューニングを実行した場合

- 下図の場合、「ManCtl」の値が FALSE なので、「Run」の値が FALSE の間、「MV」の値は「StopMV」になります。
- 「Run」の値が FALSE の間は、「StartAT」の値を TRUE にしても、オートチューニングは実行されません。
- 「StartAT」と「Run」の値を、両方とも TRUE にすると、オートチューニングが実行されます。「ATBusy」の値が、TRUE になります。
- オートチューニングが終了すると、「ATBusy」の値が FALSE になり、「ATDone」の値が TRUE になります。
- オートチューニング終了後は、オートチューニングで求められた PID 定数に基づいた「MV」が出力されます。



● オートチューニングを中止した場合

- 下図の場合、「ManCtl」の値が FALSE なので、「Run」の値が FALSE の間、「MV」の値は「StopMV」になります。
- 「Run」の値を TRUE にすると、PID 定数に基づいた「MV」が出力されます。
- 「StartAT」の値を TRUE にすると、オートチューニングが実行されます。「ATBusy」の値が TRUE になります。
- オートチューニング中に「StartAT」の値を FALSE にすると、オートチューニングが中止されます。「ATBusy」の値が、FALSE になります。
- オートチューニング中止後は、オートチューニング開始直前の PID 定数に基づいた「MV」が出力されます。
- 「Run」の値を FALSE にすると、「MV」の値は「StopMV」になります。
- オートチューニングを中止したため、「ATDone」の値は TRUE になりません。



● オートチューニング中に、オートチューニング異常が発生した場合

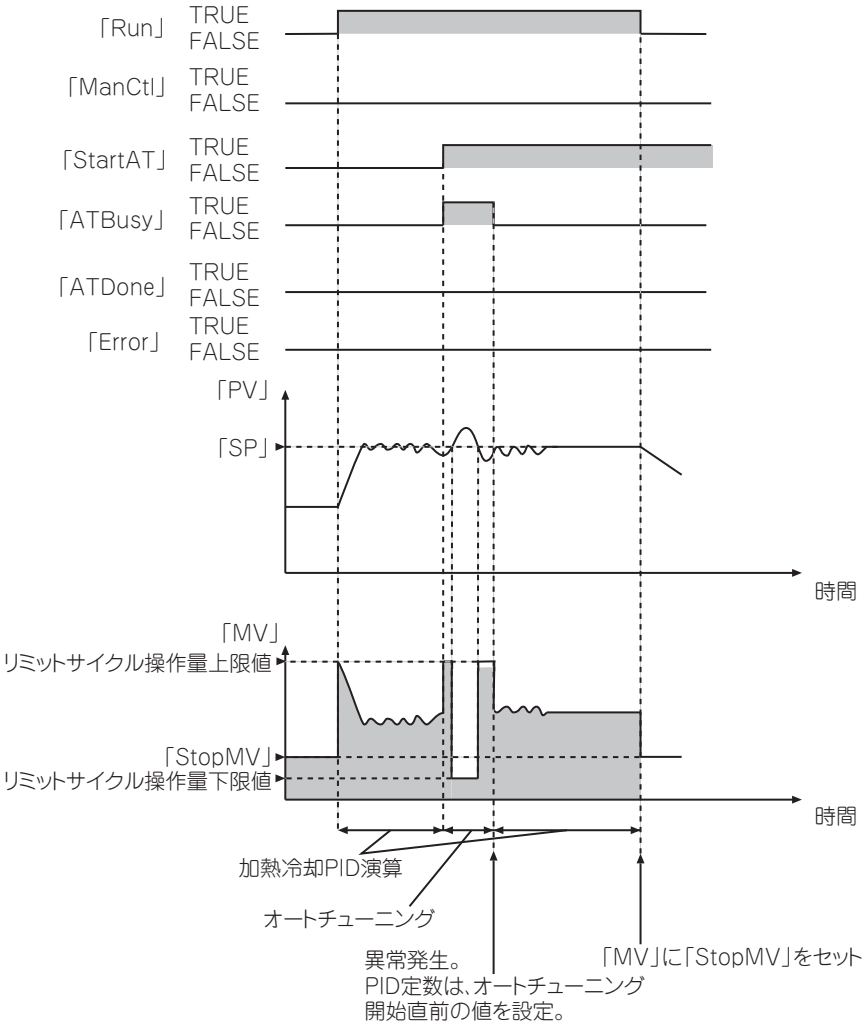
つぎの場合、オートチューニング異常となり、オートチューニングを中止します。

- 操作量＝リミットサイクル操作量の上限値で、偏差が 0 になるまでの時間が、19999s を超える。
- 操作量＝リミットサイクル操作量の下限値で、偏差が「ATHystrs」以上になるまでの時間が、19999s を超える。

オートチューニング異常が発生しても、「Error」の値は TRUE にはなりません。また、イベントログにも記録されません。
オートチューニングを中止すると、オートチューニング開始直前の PID 定数で、加熱冷却 PID 演算が再開されます。

- 下図の場合、「ManCtl」の値が FALSE なので、「Run」の値が FALSE の間、「MV」の値は「StopMV」になります。
- 「Run」の値を TRUE にすると、PID 定数に基づいた「MV」が出力されます。
- 「StartAT」の値を TRUE にすると、オートチューニングが実行されます。「ATBusy」の値が TRUE になります。
- オートチューニング実行中にオートチューニング異常が発生すると、直ちにオートチューニングが中止されます。「ATBusy」の値が、FALSE になります。
- オートチューニング異常が発生しても、「Error」の値は TRUE になりません。
- オートチューニング中止後は、オートチューニング開始直前の PID 定数に基づいた「MV」が出力されます。
- 「Run」の値を FALSE にすると、「MV」の値は「StopMV」になります。

・ オートチューニングが中止されたため、「ATDone」の値は TRUE になりません。



参考

PID 定数の調整方法

PID 定数の調整方法については、PIDAT 命令の「PID 定数の調整方法 (P.2-690)」を参照してください。

温度調節の場合の PID 定数初期値

本命令を、温度調節に使用する場合は、PID 定数の初期値として、以下の値を参考にしてください。
PID 定数以外は、各変数の初期値を使用してください。

変数	参考初期値 *1
ProportionalBand_Heat, ProportionalBand_Cool	10%FS
IntegrationTime_Heat, IntegrationTime_Cool	233s
DerivativeTime_Heat, DerivativeTime_Cool	40s

*1. オートチューニングを実行した場合は、その結果に従ってください。

使用上の注意

- 「PV」および「SP」の値は、「RngLowLmt」の値以上「RngUpLmt」の値以下であることが必要です。そのために、以下に示すように、これらの変数の単位系を合わせてください。

単位系	「PV」、「SP」の値	「RngLowLmt」、「RngUpLmt」の値
%FS	「PV」＝（測定値の物理量－MIN）／（MAX－MIN）×100 *1 「SP」＝（目標値の物理量－MIN）／（MAX－MIN）×100 *1	「RngLowLmt」=0 「RngUpLmt」=100
物理量	「PV」=測定値の物理量 「SP」=目標値の物理量	「RngLowLmt」= MIN *1 「RngUpLmt」= MAX *1

*1. MAX:入力レンジ上限の物理量、MIN:入力レンジ下限の物理量

- 各変数の変更可否は、制御状態により、以下のようになります。

変数	制御状態		
	命令実行停止 *1	運転中（オートチューニング非実行） *2	運転中（オートチューニング実行中） *3
Run	○	○	○
ManCtl	○	○	○
StartAT	○	○	○
DeadBand	○	○	○
PV	○	○	○
SP	○	○	× *4
MVLowLmt	○	○	× *4
MVUpLmt	○	○	× *4
ManResetVal *5	—	—	—
MVTrackSw	○	○	× *4
MVTrackVal	○	○	× *4
StopMV	○	○	○
ErrorMV	○	○	○
Alpha	○	○	× *4
ATCalcGain	○	○	× *4
ATHystrs	○	○	× *4
CtlPrdCool	○	○	× *4
SampTime	○	× *6	× *4
RngLowLmt	○	× *6	× *4
RngUpLmt	○	× *6	× *4
DirOpr *5	—	—	—
ProportinalBand_Heat	○	○	× *7
IntegrationTime_Heat	○	○	× *7
DerivativeTime_Heat	○	○	× *7
ProportinalBand_Cool	○	○	× *7
IntegrationTime_Cool	○	○	× *7
DerivativeTime_Cool	○	○	× *7
ManMV	○	○	○

*1. 「ManCtl」=TRUE または「Run」=FALSE または「Error」=TRUE または「MVTrackSw」=TRUE
*2. 「ManCtl」=FALSE かつ「Run」= TRUE かつ「Error」=FALSE かつ「MVTrackSw」= FALSE かつ「ATBusy」=FALSE

*3. 「ManCtl」=FALSE かつ「Run」= TRUE かつ「Error」=FALSE かつ「MVTrackSw」= FALSE かつ「ATBusy」=TRUE
*4. オートチューニング実行直前の値を使用して、オートチューニングが実行されます。
*5. 本命令では使用しません。値の変更は可能ですが、無視されます。
*6. 運転実行直前の値を使用して、運転が実行されます。
*7. 値の変更は可能ですが、無視されます。オートチューニング完了時に、オートチューニングで算出された値に上書きされます。

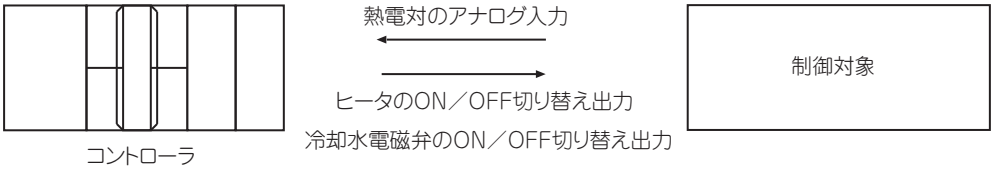
- 「SampTime」の値は、100ns 未満が切り捨てられます。
- 「ManCtl」の値が TRUE 時に、「StartAT」の値を TRUE にした場合、つぎに「ManCtl」の値が FALSE になった時点でオートチューニングを開始します。
- 「ErrorMV」の値が、有効範囲内（-320～320）でない場合、異常時の「MV」の値は 0 になります。
- オートチューニング中に「ManCtl」の値を TRUE にすると、オートチューニングを中止します。
- オートチューニング異常が発生しても、「Error」の値は TRUE にはなりません。また、イベントログにも記録されません。
- つぎの場合、異常となります。
  「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  「ATDone」、「ATBusy」は FALSE となります。
  「ManCtl」と「Run」の値がともに FALSE であれば、「MV」に「ErrorMV」の値がセットされます。ただし、「ErrorMV」の値が有効範囲外の場合は、「MV」の値は 0 になります。

異常の内容	「ErrorID」の値
入力変数のいずれかの値が、有効範囲外	16#0400
「RngLowLmt」≧「RngUpLmt」	16#0401
「MVLowLmt」≧「MVUpLmt」	

- 上記以外の条件で異常停止させたい場合は、異常発生時に、「Run」の値を FALSE にする処理を設けてください。
- 「PV」または「SP」の値が有効範囲外で異常が発生した後、値を有効範囲内に変更しても、5 秒間は異常状態を保持します。つまり、5 秒間は「Error」の値が FALSE のままになります。
- 異常解除後、「Run」の値が TRUE なら自動的に加熱冷却 PID 演算を再開します。「Run」と「StartAT」の値がともに TRUE なら、自動的にオートチューニングを再開します。
- 異常の有無の判定は、サンプリング周期ごとに行います。
- 以下の条件でバックアップ、リストアを実行すると、オートチューニングで求めた PID 定数が、バックアップ前の値に戻ります。注意してください。
  a）入出力パラメータに、保持変数を指定する。
  b）バックアップ、オートチューニング、リストアを、この順序で実行する。
- オート運転からマニュアル運転に切り替える場合、「MV_Heat」と「MV_Cool」のうち、正値である方がバンプレス（急変なし）になるように、値を引き継ぎます。したがって、もう一方の値は急変する可能性があります。

サンプルプログラム

PIDAT_HeatCool 命令を使って、温度制御を行います。
制御対象からの入力は、熱電対のアナログ入力が 1 点あるとします。
制御対象への出力は、加熱用デジタル出力と冷却用デジタル出力の 2 点があるとします。
加熱用デジタル出力は、ヒータの ON/OFF 切替、冷却用デジタル出力は、冷却水の電磁弁の ON/OFF 切替であるとします。



ユニット構成

接続するユニットは、以下のとおりです。
- 形 CJ1W-AD04U 絶縁型フルマルチ入力ユニット
- 形 CJ1W-OC201 リレー接点出力ユニット

I/O マップ

各ユニットの I/O マップは、以下のように設定します。

● 形 CJ1W-AD04U

ポート	説明	R/W	データ型	変数	変数コメント	変数種別
Ch1_AIInPV	入力 No.1 測定値	R	INT	J01_Ch1_AIInPV	熱電対入力	グローバル変数

● 形 CJ1W-OC201

ポート	説明	R/W	データ型	変数	変数コメント	変数種別
Ch1_Out00	出力 CH1 接点 00	RW	BOOL	J02_Ch1_Out00	加熱装置用出力	グローバル変数
Ch1_Out04	出力 CH1 接点 04	RW	BOOL	J02_Ch1_Out04	冷却装置用出力	グローバル変数

タッチパネルの仕様

本サンプルプログラムは、コントローラにタッチパネルを接続することを前提としています。タッチパネルを介しての入出力情報には、以下のものがあります。

入力／入出力／出力	情報
入力	本サンプルプログラムの実行フラグ マニュアル／オート切替フラグ 目標値 オートチューニング実行フラグ デッドバンド 初期設定パラメータ 運転中設定パラメータ
入出力	加熱操作の比例帯、積分時間、微分時間 冷却操作の比例帯、積分時間、微分時間 マニュアル操作量

入力／入出力／出力	情報
出力	測定値 オートチューニング正常終了フラグ オートチューニング実行中フラグ 異常発生フラグ 操作量 加熱操作の操作量 冷却操作の操作量

操作量の時分割比例出力への変換

この例では、加熱装置用出力も冷却装置用出力も ON/OFF のデジタル出力です。したがって、加熱装置の操作量も冷却装置の操作量も、時分割比例出力に変換する必要があります。
操作量を時分割比例出力に変換するには、「TimeProportionalOut (P.2-733)」命令を使用します。
ただし、オートチューニング中は、PIDAT_HeatCool 命令の出力「MV_Heat」、「MV_Cool」が変化した直後に、加熱装置用出力、冷却装置用出力を変化させなければならないため、
TimeProportionalOut 命令は使えません。
TimeProportionalOut 命令を使用すると、加熱装置用出力、冷却装置用出力は、ユーザが設定した制御周期ごとにしか変化しません。そのため本サンプルプログラムでは、オートチューニング中の操作量の時分割比例出力への変換は、タイマ命令を使って行います。

グローバル変数の定義

● グローバル変数

名称	データ型	初期値	割付先	保持	ネットワーク公開	コメント
J01_Ch1_AIInPV	INT	0	IOBus://rack#0/slot#0/Ch1_AIInPV	☐	非公開	形 CJ1W-AD04U からの熱電対入力
J02_Ch1_Out00	BOOL	FALSE	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00	☐	非公開	形 CJ1W-OC201 への加熱用出力
J02_Ch1_Out04	BOOL	FALSE	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out04	☐	非公開	形 CJ1W-OC201 への冷却用出力
PTIn_Run	BOOL	FALSE		☑	入力	タッチパネルからの本サンプルプログラム実行フラグ入力
PTIn_ManCtl	BOOL	FALSE		☑	入力	タッチパネルからのマニュアル／オート切替フラグ入力
PTIn_SP	REAL			☑	入力	タッチパネルからの目標値入力
PTIn_StartAT	BOOL	FALSE		☑	入力	タッチパネルからのオートチューニング実行フラグ入力

名称	データ型	初期値	割付先	保持	ネットワーク公開	コメント
PTIn_DeadBand	REAL	0		☑	入力	タッチパネルからのデッドバンド入力
PTIn_InitParam	_sINIT_SET_PARAMS	(SampTime := T#100ms, RngLowLmt := 0.0, RngUpLmt := 100.0, DirOpr := False)		☑	入力	タッチパネルからの初期設定パラメータ入力
PTIn_InitSetOpr_SampTime	LINT	100		☑	入力	タッチパネルからのサンプリング周期入力（ms 単位）
PTIn_OprParam	_sOPR_SET_PARAMS	(MVLowLmt := -100, MVUpLmt := 100, ManResetVal := 0.0, MVTrackSw := False, MVTrackVal := 0.0, StopMV := 0.0, ErrorMV := 0.0,Alpha := 0.65, ATCalcGain := 1.0, ATHystrs := 0.2)		☑	入力	タッチパネルからの運転中設定パラメータ入力
PTOut_PV	REAL	0		☐	出力	タッチパネルへの測定値出力
PT_PB_Heat	REAL	1		☑	入力	タッチパネルからの加熱操作の比例帯入出力
PT_TI_Heat	LINT	1000		☑	入力	タッチパネルからの、加熱操作の積分時間入出力（ms 単位）
PT_TD_Heat	LINT	1000		☑	入力	タッチパネルからの、加熱操作の微分時間入出力（ms 単位）
PT_PB_Cool	REAL	1		☑	入力	タッチパネルからの、冷却操作の比例帯入出力
PT_TI_Cool	LINT	1000		☑	入力	タッチパネルからの、冷却操作の積分時間入出力（ms 単位）
PT_TD_Cool	LINT	1000		☑	入力	タッチパネルからの、冷却操作の微分時間入出力（ms 単位）
PT_ManMV	REAL	0		☑	入力	タッチパネルからのマニュアル操作量入出力
PTOut_ATDone	BOOL	FALSE		☐	出力	タッチパネルへのオートチューニング正常終了フラグ出力
PTOut_ATBusy	BOOL	FALSE		☐	出力	タッチパネルへのオートチューニング実行中フラグ出力
PTOut_Error	BOOL	FALSE		☐	出力	タッチパネルへの異常終了フラグ出力
PTOut_MV	REAL	0		☐	出力	タッチパネルへの操作量出力

名称	データ型	初期値	割付先	保持	ネットワーク公開	コメント
PTOut_MVHeat	REAL	0		┌┐	出力	タッチパネルへの、加熱操作の操作量出力
PTOut_MVCool	REAL	0		┌┐	出力	タッチパネルへの、冷却操作の操作量出力

LD

内部変数	名称	データ型	初期値	コメント
	PB_Heat	REAL	0	加熱操作の比例帯
	PB_Cool	REAL	0	冷却操作の比例帯
	MV	REAL	0	操作量
	MV_Heat	REAL	0	加熱操作の操作量
	MV_Cool	REAL	0	冷却操作の操作量
	PIDAT_HeatCool_inst	PIDAT_HeatCool		PIDAT_HeatCool 命令のインスタンス
	TI_Heat	TIME	T#0s	加熱操作の積分時間
	TI_Cool	TIME	T#0s	冷却操作の積分時間
	TD_Heat	TIME	T#0s	加熱操作の微分時間
	TD_Cool	TIME	T#0s	冷却操作の微分時間
	ManMV	REAL	0	マニュアル操作量
	CtlPrd_Cool	TIME	T#20s	冷却操作の制御周期
	CtlPrd_Heat	TIME	T#2s	加熱操作の制御周期
	TPOHeat_inst	TimeProportionalOut		加熱操作用の TimeProportionalOut 命令のインスタンス
	TPOCool_inst	TimeProportionalOut		冷却操作用の TimeProportionalOut 命令のインスタンス
	ATHeatPhase	BOOL	FALSE	オートチューニングの加熱操作中フラグ
	ATCoolPhase	BOOL	FALSE	オートチューニングの冷却操作中フラグ
	MVHeatTime	TIME	T#0s	オートチューニング中の加熱操作時間
	MVCoolTime	TIME	T#0s	オートチューニング中の冷却操作時間
	AT_Heat_inst	TP		オートチューニング中の加熱操作の操作量出力用 TP 命令インスタンス
	AT_Cool_inst	TP		オートチューニング中の冷却操作の操作量出力用 TP 命令インスタンス
	EachCtlPrd_ATHeat_inst	TON		オートチューニング中の加熱操作の操作量出力用 TON 命令インスタンス
	EachCtlPrd_ATCool_inst	TON		オートチューニング中の冷却操作の操作量出力用 TON 命令インスタンス
	PV	REAL	0	測定値

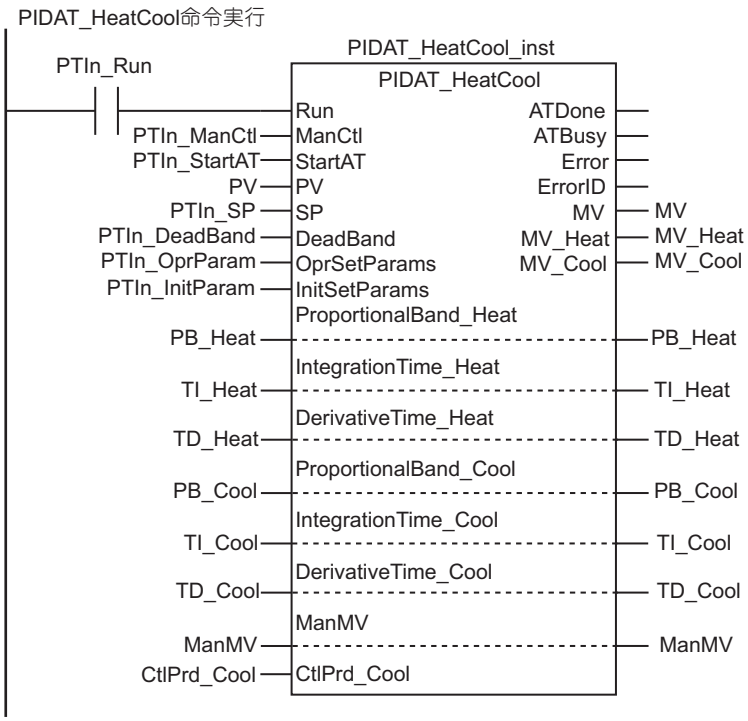
外部変数	名称	データ型	コメント
	J01_Ch1_AllnPV	INT	形 CJ1W-AD04U からの熱電対入力
	J02_Ch1_Out00	BOOL	形 CJ1W-OC201 への加熱用出力
	J02_Ch1_Out04	BOOL	形 CJ1W-OC201 への冷却用出力

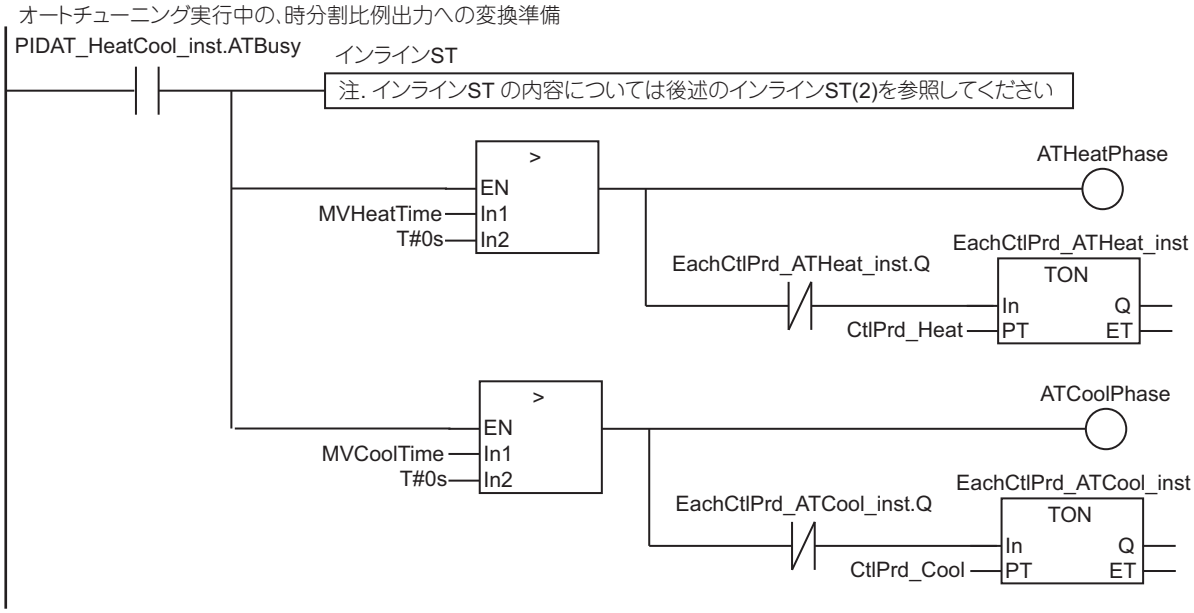
外部変数	名称	データ型	コメント
	PTIn_Run	BOOL	タッチパネルからの本サンプルプログラム実行フラグ入力
	PTIn_ManCtl	BOOL	タッチパネルからのマニュアル／オート切替フラグ入力
	PTIn_SP	REAL	タッチパネルからの目標値入力
	PTIn_StartAT	BOOL	タッチパネルからのオートチューニング実行フラグ入力
	PTIn_DeadBand	REAL	タッチパネルからのデッドバンド入力
	PTIn_InitParam	_sINIT_SET_PARAMS	タッチパネルからの初期設定パラメータ入力
	PTIn_InitSetOpr_SampTime	LINT	タッチパネルからのサンプリング周期入力（ms 単位）
	PTIn_OprParam	_sOPR_SET_PARAMS	タッチパネルからの運転中設定パラメータ入力
	PTOut_PV	REAL	タッチパネルへの測定値出力
	PT_PB_Heat	REAL	タッチパネルからの加熱操作の比例帯入出力
	PT_TI_Heat	LINT	タッチパネルからの、加熱操作の積分時間入出力（ms 単位）
	PT_TD_Heat	LINT	タッチパネルからの、加熱操作の微分時間入出力（ms 単位）
	PT_PB_Cool	REAL	タッチパネルからの、冷却操作の比例帯入出力
	PT_TI_Cool	LINT	タッチパネルからの、冷却操作の積分時間入出力（ms 単位）
	PT_TD_Cool	LINT	タッチパネルからの、冷却操作の微分時間入出力（ms 単位）
	PT_ManMV	REAL	タッチパネルからのマニュアル操作量入出力
	PTOut_ATDone	BOOL	タッチパネルへのオートチューニング正常終了フラグ出力
	PTOut_ATBusy	BOOL	タッチパネルへのオートチューニング実行中フラグ出力
	PTOut_Error	BOOL	タッチパネルへの異常終了フラグ出力
	PTOut_MV	REAL	タッチパネルへの操作量出力
	PTOut_MVHeat	REAL	タッチパネルへの、加熱操作の操作量出力
	PTOut_MVCool	REAL	タッチパネルへの、冷却操作の操作量出力

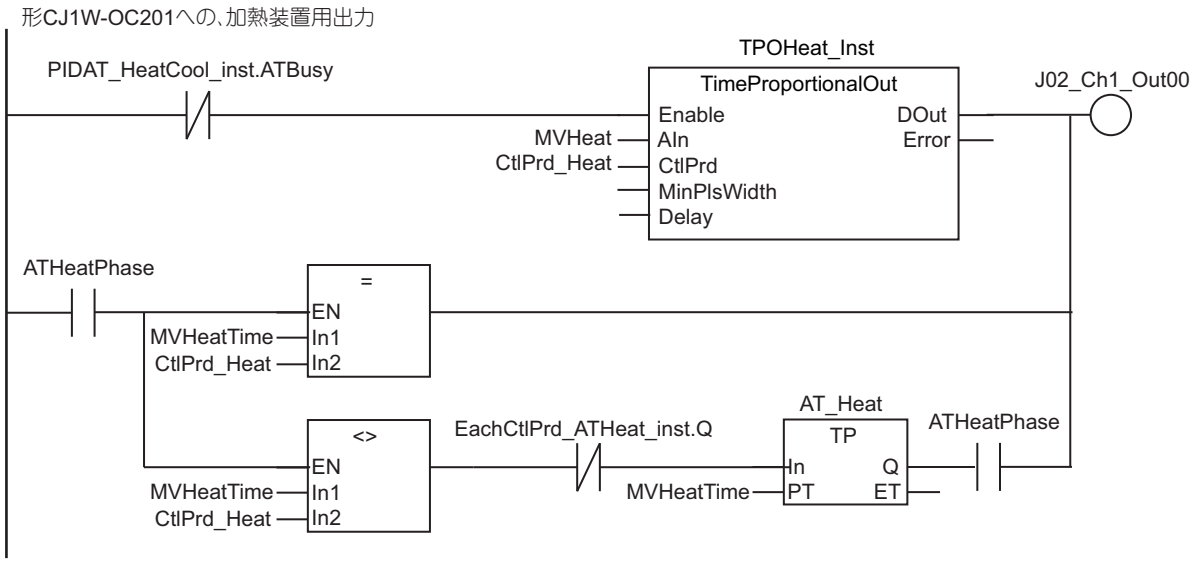
CJ1W-AD04Uおよびタッチパネルからの入力値の、単位変換

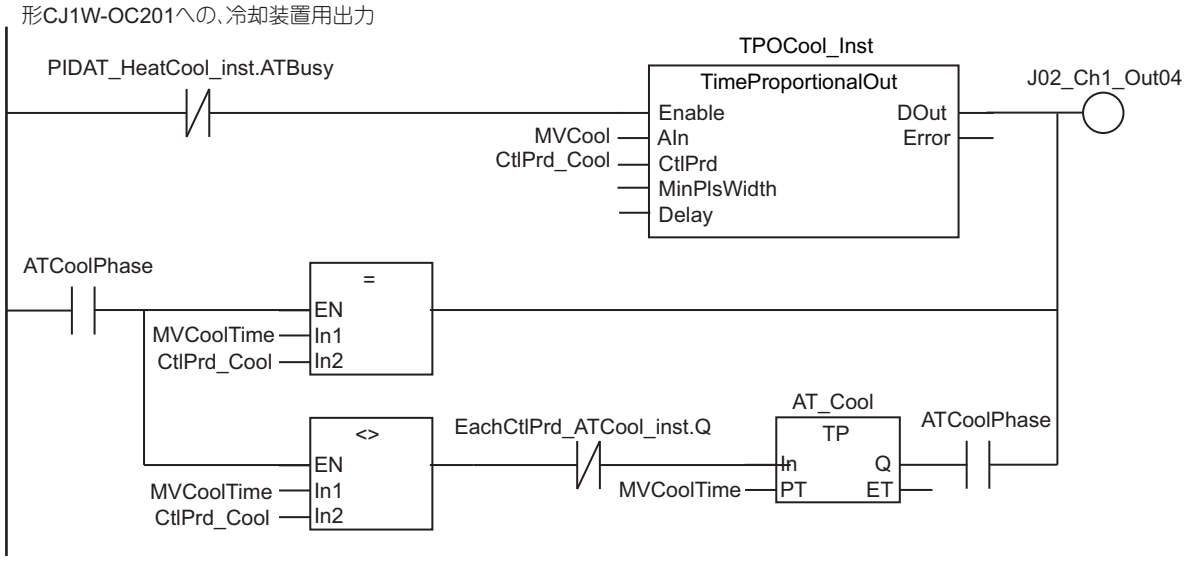
インラインST

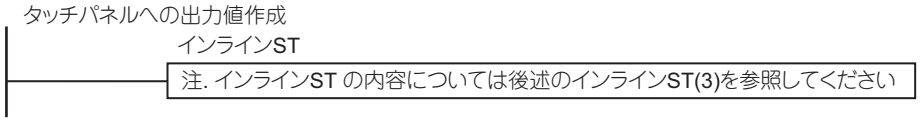
注. インラインST の内容については後述のインラインST(1)を参照してください


PIDAT_HeatCool命令実行
PIDAT_HeatCool_inst
PTIn_Run
PIDAT_HeatCool
Run
ATDone
PTIn_ManCtl
ManCtl
ATBusy
PTIn_StartAT
StartAT
Error
PV
PV
ErrorID
PTIn_SP
SP
MV
MV
PTIn_DeadBand
DeadBand
MV_Heat
MV_Heat
PTIn_OprParam
OprSetParams
MV_Cool
MV_Cool
PTIn_InitParam
InitSetParams
ProportionalBand_Heat
PB_Heat
PB_Heat
IntegrationTime_Heat
TI_Heat
TI_Heat
DerivativeTime_Heat
TD_Heat
TD_Heat
ProportionalBand_Cool
PB_Cool
PB_Cool
IntegrationTime_Cool
TI_Cool
TI_Cool
DerivativeTime_Cool
TD_Cool
TD_Cool
ManMV
ManMV
ManMV
CtlPrd_Cool
CtlPrd_Cool


オートチューニング実行中の、時分割比例出力への変換準備
PIDAT_HeatCool_inst.ATBusy
インラインST
注. インラインST の内容については後述のインラインST(2)を参照してください
>
EN
MVHeatTime
In1
T#0s
In2
ATHeatPhase
EachCtlPrd_ATHeat_inst
EachCtlPrd_ATHeat_inst.Q
TON
In
Q
CtlPrd_Heat
PT
ET
>
EN
MVCoolTime
In1
T#0s
In2
ATCoolPhase
EachCtlPrd_ATCool_inst
EachCtlPrd_ATCool_inst.Q
TON
In
Q
CtlPrd_Cool
PT
ET







● インライン ST(1)の内容

```
// 形 CJ1W-AD04U およびタッチパネルからの入力値の、単位変換
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;
```

● インライン ST(2)の内容

```
MVHeatTime := MULTIME(CtlPrd_Heat,(MV_Heat/100));
MVCoolTime := MULTIME(CtlPrd_Cool,(MV_Cool/100));
```

● インライン ST(3)の内容

```
// タッチパネルへの出力値作成
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

内部変数	名称	データ型	初期値	コメント
	PB_Heat	REAL	0	加熱操作の比例帯
	PB_Cool	REAL	0	冷却操作の比例帯
	MV	REAL	0	操作量
	MV_Heat	REAL	0	加熱操作の操作量
	MV_Cool	REAL	0	冷却操作の操作量
	PIDAT_HeatCool_inst	PIDAT_HeatCool		PIDAT_HeatCool 命令のインスタンス
	TI_Heat	TIME	T#0s	加熱操作の積分時間
	TI_Cool	TIME	T#0s	冷却操作の積分時間
	TD_Heat	TIME	T#0s	加熱操作の微分時間
	TD_Cool	TIME	T#0s	冷却操作の微分時間
	ManMV	REAL	0	マニュアル操作量
	CtlPrd_Cool	TIME	T#20s	冷却操作の制御周期
	CtlPrd_Heat	TIME	T#2s	加熱操作の制御周期
	TPOHeat_inst	TimeProportionalOut		加熱操作用の TimeProportionalOut 命令のインスタンス
	TPOCool_inst	TimeProportionalOut		冷却操作用の TimeProportionalOut 命令のインスタンス

内部変数	名称	データ型	初期値	コメント
	ATHeatPhase	BOOL	FALSE	オートチューニングの加熱操作中フラグ
	ATCoolPhase	BOOL	FALSE	オートチューニングの冷却操作中フラグ
	MVHeatTime	TIME	T#0s	オートチューニング中の加熱操作時間
	MVCoolTime	TIME	T#0s	オートチューニング中の冷却操作時間
	AT_Heat_inst	TP		オートチューニング中の加熱操作の操作量出力用 TP 命令インスタンス
	AT_Cool_inst	TP		オートチューニング中の冷却操作の操作量出力用 TP 命令インスタンス
	EachCtlPrd_ATHeat_inst	TON		オートチューニング中の加熱操作の操作量出力用 TON 命令インスタンス
	EachCtlPrd_ATCool_inst	TON		オートチューニング中の冷却操作の操作量出力用 TON 命令インスタンス
	PV	REAL	0	測定値

外部変数	名称	データ型	コメント
	J01_Ch1_AIInPV	INT	形 CJ1W-AD04U からの熱電対入力
	J02_Ch1_Out00	BOOL	形 CJ1W-OC201 への加熱用出力
	J02_Ch1_Out04	BOOL	形 CJ1W-OC201 への冷却用出力
	PTIn_Run	BOOL	タッチパネルからの本サンプルプログラム実行フラグ入力
	PTIn_ManCtl	BOOL	タッチパネルからのマニュアル／オート切替フラグ入力
	PTIn_SP	REAL	タッチパネルからの目標値入力
	PTIn_StartAT	BOOL	タッチパネルからのオートチューニング実行フラグ入力
	PTIn_DeadBand	REAL	タッチパネルからのデッドバンド入力
	PTIn_InitParam	_sINIT_SET_PARAMS	タッチパネルからの初期設定パラメータ入力
	PTIn_InitSetOpr_SampTime	LINT	タッチパネルからのサンプリング周期入力（ms 単位）
	PTIn_OprParam	_sOPR_SET_PARAMS	タッチパネルからの運転中設定パラメータ入力
	PTOut_PV	REAL	タッチパネルへの測定値出力
	PT_PB_Heat	REAL	タッチパネルからの加熱操作の比例帯入出力
	PT_TI_Heat	LINT	タッチパネルからの、加熱操作の積分時間入出力（ms 単位）
	PT_TD_Heat	LINT	タッチパネルからの、加熱操作の微分時間入出力（ms 単位）
	PT_PB_Cool	REAL	タッチパネルからの、冷却操作の比例帯入出力
	PT_TI_Cool	LINT	タッチパネルからの、冷却操作の積分時間入出力（ms 単位）
	PT_TD_Cool	LINT	タッチパネルからの、冷却操作の微分時間入出力（ms 単位）
	PT_ManMV	REAL	タッチパネルからのマニュアル操作量入出力

外部変数	名称	データ型	コメント
	PTOut_ATDone	BOOL	タッチパネルへのオートチューニング正常終了フラグ出力
	PTOut_ATBusy	BOOL	タッチパネルへのオートチューニング実行中フラグ出力
	PTOut_Error	BOOL	タッチパネルへの異常終了フラグ出力
	PTOut_MV	REAL	タッチパネルへの操作量出力
	PTOut_MVHeat	REAL	タッチパネルへの、加熱操作の操作量出力
	PTOut_MVCool	REAL	タッチパネルへの、冷却操作の操作量出力

```
// 形 CJ1W-AD04U およびタッチパネルからの入力値の、単位変換
PV := INT_TO_REAL(J01_Ch1_AllnPV)/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;

// PIDAT_HeatCool 命令実行
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);

//オートチューニング実行中の、時分割比例出力への変換準備
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);

// 形 CJ1W-OC201 への、加熱装置用出力
TPOHeat_inst(Enable :=NOT(PIDAT_HeatCool_inst.ATBusy),
  AIn :=MV_Heat,
  CtlPrd :=CtlPrd_Heat );
AT_Heat_inst(In:= ATHeatPhase & (MVHeatTime<>CtlPrd_Heat) & NOT(EachCtlPrd_ATHeat_inst.Q) ,
  PT:= MVHeatTime);
JO2_Ch1_Out00 :=( TPOHeat_inst.DOut ) OR
  ( ATHeatPhase & (MVHeatTime=CtlPrd_Heat)) OR
  ( AT_Heat_inst.Q & ATHeatPhase );

// 形 CJ1W-OC201 への、冷却装置用出力
TPOCool_inst(Enable :=NOT(PIDAT_HeatCool_inst.ATBusy),
  AIn :=MV_Cool,
  CtlPrd :=CtlPrd_Cool );
AT_Cool_inst(In:= ATCoolPhase & (MVCoolTime<>CtlPrd_Cool) & NOT(EachCtlPrd_ATCool_inst.Q) ,
  PT:= MVCoolTime);
JO2_Ch1_Out04 :=( TPOCool_inst.DOut ) OR
  ( ATCoolPhase & (MVCoolTime=CtlPrd_Cool)) OR
  ( AT_Cool_inst.Q & ATCoolPhase );

// タッチパネルへの出力値作成
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

操作量を時分割比例出力に変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeProportionalOut	時分割比例出力	FB	TimeProportionalOut_Instance TimeProportionalOut Enable AIn CtlPrd MinPlsWidth Delay DOut Error	TimeProportionalOut_instance(Enable, AIn, CtlPrd, MinPlsWidth, Delay, DOut, Error);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	動作指令	入力	TRUE: 実行 FALSE: 時分割比例出力リセット	データ型に従う	−	FALSE
AIn	操作量		操作量	0 ～ 100	%	0
CtlPrd	制御周期		時分割比例出力の制御周期	T#0.1s ～ T#100s	s	T#2s
MinPlsWidth	最小パルス幅		最小パルス幅	0 ～ 50	%	1
Delay	ディレー		オンディレー時間	0 ～ 100	%	0
DOut	時分割比例出力	出力	TRUE: 時分割比例出力 ON FALSE: 時分割比例出力 OFF	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
AIn														○						
CtlPrd																○				
MinPlsWidth														○						
Delay														○						
DOut	○																			

機能

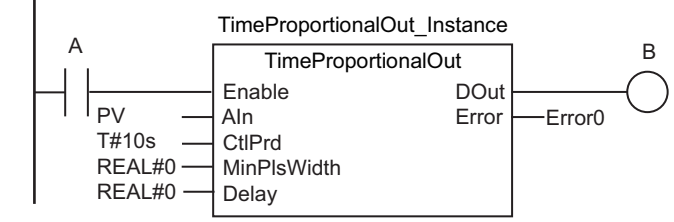
PID 制御などの操作量を、時分割比例出力に変換します。
時分割比例出力とは、操作量を、ON と OFF の時間比に比例させて変換した出力です。

動作指令「Enable」が TRUE の間、操作量「AIn」の値を、制御周期「CtlPrd」の時分割比例出力「DOut」に変換します。
「Enable」が FALSE のときは、時分割比例出力をリセットします。

「DOut」と「Error」は FALSE となります。「CtlPrd」、「MinPlsWidth」、および「Delay」の値は、「Enable」が FALSE から TRUE に変化したときに反映されます。

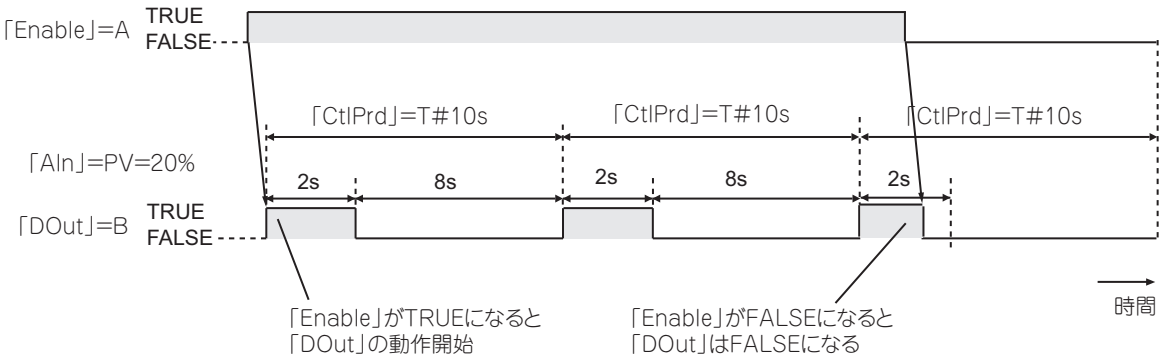
「CtlPrd」の値が 10s、「AIn」の値が 20%の場合の記述例を以下に示します。「Enable」が TRUE の間、「DOut」は 2s の間 TRUE、続く 8s の間 FALSE になります。これを 10s 周期で繰り返します。

LD



ST

```
TimeProportionalOut_instance(A,PV,T#10s,REAL#0,REAL#0,B,Error0);
```



時分割比例出力「DOut」の分解能

「AIn」の値を「DOut」に変換する最小単位のことを「DOut」の分解能と呼びます。
「DOut」の分解能より「AIn」の値の分解能が高い場合、「AIn」を「DOut」の分解能に応じて丸めた値を「DOut」に変換します。

「DOut」の分解能は、以下の式で算出します。

「DOut」の分解能(%)＝タスク周期÷「CtlPrd」×100

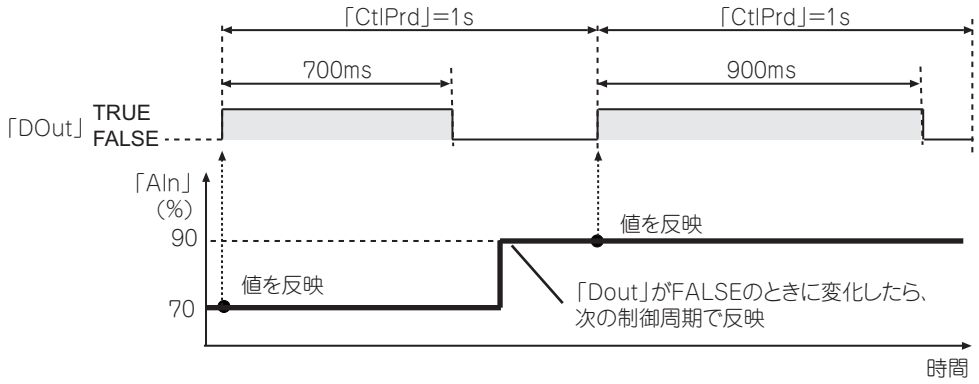
たとえば、タスク周期が 1ms、「CtlPrd」の値が 1s の場合は、「DOut」の分解能は 0.1%になります。この場合、「AIn」の値の小数点以下 2 けため以降は丸められます。

操作量「AIn」の値の反映タイミング

「AIn」の値の反映タイミングは、「DOut」が FALSE であるか TRUE であるかで異なります。

●「DOut」が FALSE の場合

「DOut」が FALSE の間に「AIn」の値が変化した場合は、次の制御周期で反映されます。

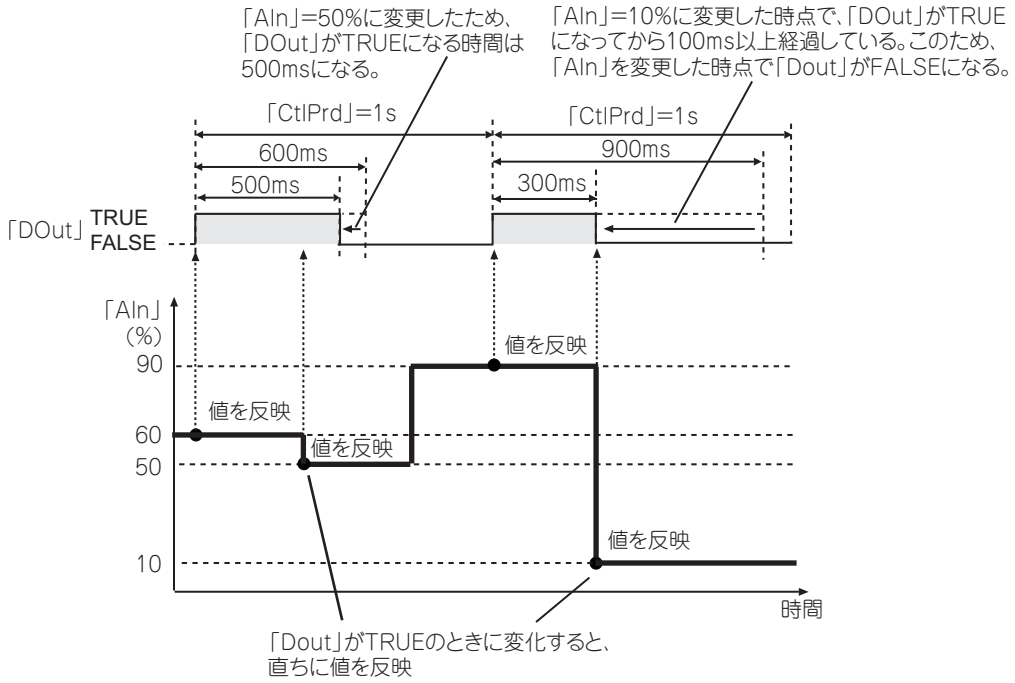


●「DOut」が TRUE の場合

「DOut」が TRUE の間に「AIn」の値が変化した場合は、直ちに反映されます。

たとえば、制御周期「CtlPrd」の値が 1s の場合、以下のように動作します。

- 制御周期の開始時に「AIn」の値が 60%の場合、「DOut」が TRUE の間に「AIn」の値が 50%に変化すると、「DOut」は 500ms の間だけ TRUE になります。
- 制御周期の開始時に「AIn」の値が 90%の場合、「DOut」が TRUE になり 300ms 経過後に「AIn」の値が 10%に変化するとします。この場合、10%に相当する 100ms はすでに経過しているため、「DOut」は直ちに FALSE になります。



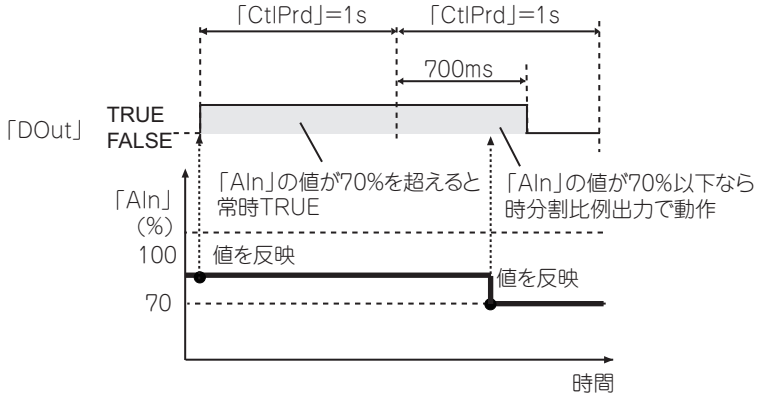
最小パルス幅「MinPlsWidth」による時分割比例出力「DOut」の動作

最小パルス幅とは、「DOut」が TRUE または FALSE の値を保持する時間の最小値のことです。
最小パルス幅「MinPlsWidth」を設定することにより、「DOut」のチャタリングを低減する効果があります。
たとえば、冷却制御では、ファンの ON/OFF を切り替える回数を減らすことで、消費電力を抑えることができます。

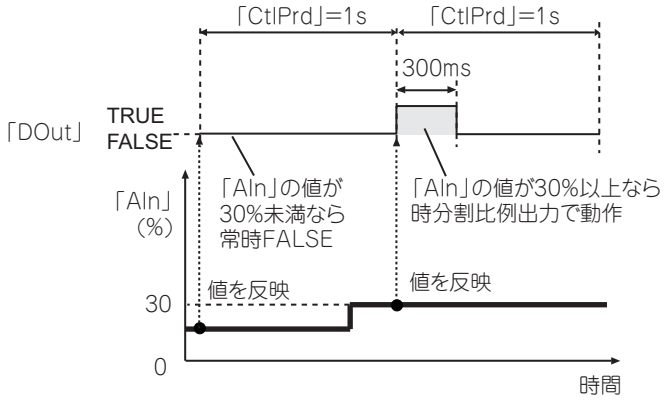
「MinPlsWidth」と「AIn」の値の関係によって、「DOut」は以下のように動作します。

「MinPlsWidth」と「AIn」の値の関係	「DOut」の動作
「AIn」＜「MinPlsWidth」	常時 FALSE
「MinPlsWidth」≦「AIn」≦100－「MinPlsWidth」	時分割比例で出力
「AIn」＞100－「MinPlsWidth」	常時 TRUE

たとえば、「MinPlsWidth」が 30%の場合、「DOut」は下図のように動作します。
「AIn」の値が 70%を超えている間は常時 TRUE、70%以下になると時分割比例出力になります。



「AIn」の値が 30%未満の間は常時 FALSE、30%以上になると時分割比例出力になります。



ディレー「Delay」による時分割比例出力「DOut」の動作

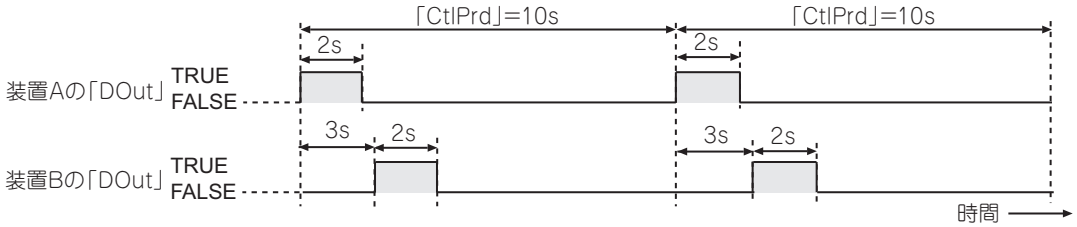
制御周期の開始から設定時間が経過した後に「DOut」を TRUE にすることを、ディレーといいます。
複数の本命令を使用する場合、それぞれの「DOut」が TRUE になるタイミングを、「Delay」の設定によってずらすことでができます。
これにより、「DOut」が同時に ON になる可能性を低減する効果があります。
たとえば、複数のヒータを動作させる場合、「Delay」でそれぞれのヒータへの出力が ON になるタイミングをずらすことで、同時に使用する電力を低減できます。

制御周期の開始から、ディレー「Delay」の割合の時間が経過した後に、「DOut」が TRUE になります。

たとえば、同じ制御周期の装置 A、装置 B に対して、以下の値を設定するとします。

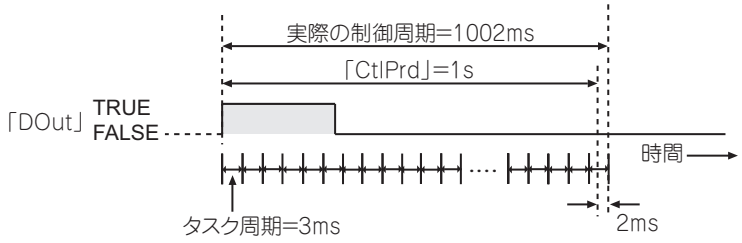
装置	「Delay」の値	「AIn」の値	「CtlPrd」の値
装置 A	0%	20%	10s
装置 B	30%		

装置 A の「DOut」は、制御周期の開始時に TRUE になります。装置 B の「DOut」は、制御周期の開始後 3s 後に TRUE になります。

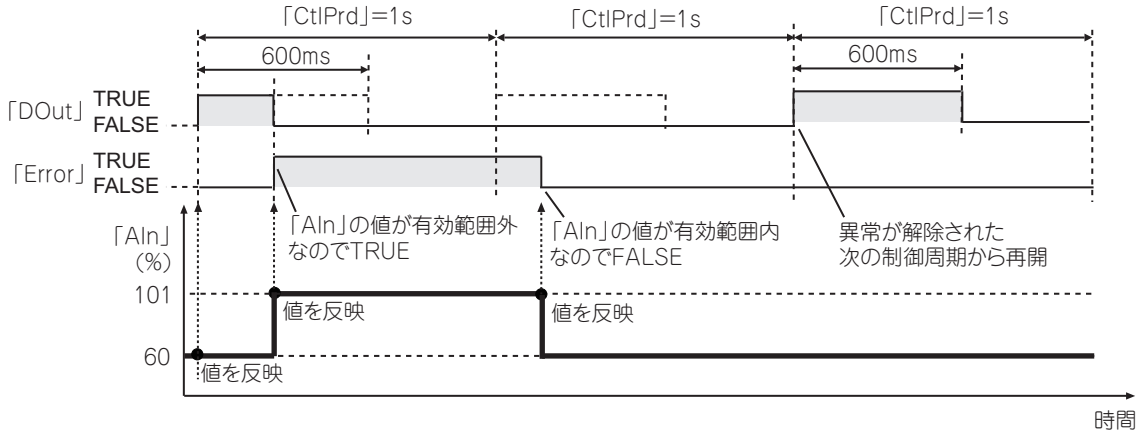


使用上の注意

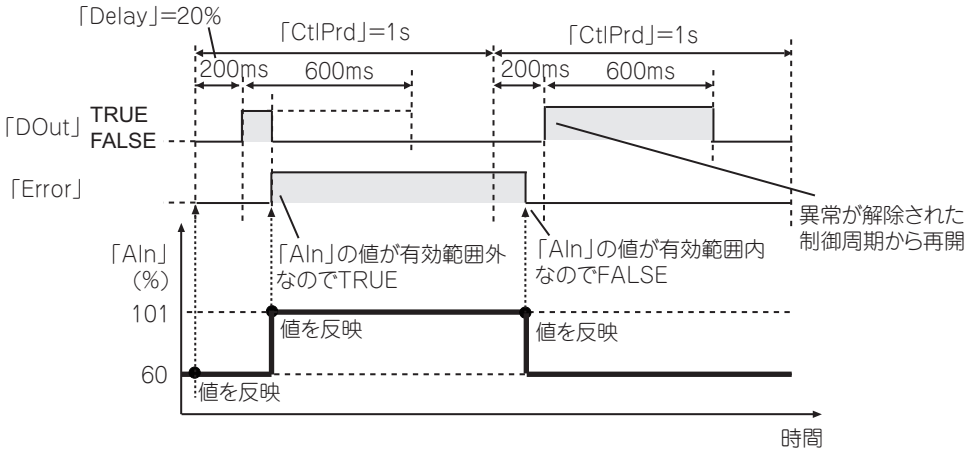
- プログラムを割り付けたタスク周期の整数倍を制御周期「CtlPrd」の値に設定してください。
  タスク周期の整数倍以外を設定した場合は、制御周期「CtlPrd」の終了後の次のタスク実行タイミングまでが実際の制御周期になります。たとえば、タスク周期が 3ms、「CtlPrd」の値が 1s の場合は、「CtlPrd」の終了後の次のタスク実行タイミングまでの 1002ms が実際の制御周期となります。



- 「DOut」の分解能が 0.1％以下になるように、タスク周期と制御周期「CtlPrd」の値を設定してください。「DOut」の分解能が 0.1%を超えると、「DOut」が TRUE になる割合と「AIn」の値との誤差が大きくなり、制御性能が低下します。たとえば、「CtlPrd」が 10s の場合は、タスク周期を 10ms 以下に設定してください。
- 本命令を複数使用して、それぞれの制御周期を同期させたい場合は、本命令を同じプログラムの中で使用してください。異なるプログラムで本命令を使用すると、プログラムの実行タイミングに依存して制御周期がずれます。
- 「Enable」の値が TRUE になってから「DOut」が動作を開始するまでの時間差は不定です。
- 「AIn」、「CtlPrd」、「MinPlsWidth」、「Delay」の値が有効範囲外の場合、異常となります。「Error」は TRUE となり、「DOut」は FALSE となります。「AIn」の値が有効範囲外である場合、異常を解除するタイミングによって、「DOut」の動作が以下のように異なります。
  a)「DOut」が TRUE になるタイミングを過ぎて異常が解除された場合は、次の制御周期から「DOut」は時分割比例出力を再開します。

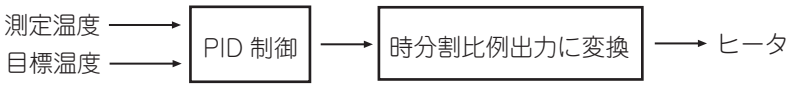


b)「DOut」が TRUE になるタイミングより前に異常が解除された場合は、異常が解除された制御周期から「DOut」は時分割比例出力を再開します。



サンプルプログラム

上下限警報、上下限偏差警報付きの 4 点の温度制御を行います。制御方式は PID 制御を用い、PID 制御の操作量を時分割比例出力に変換し、ヒータに出力します。



仕様

以下のような仕様で、温度制御を行います。

項目	仕様
入力ユニット	絶縁型高分解能フルマルチ入力ユニット 形 CJ1W-PH41U
入力種別	すべて K 型熱電対
出力ユニット	トランジスタ出力ユニット 形 CJ1W-OD212
目標温度	100℃
上限温度	200℃
下限温度	0℃
上下限警報のヒステリシス	5℃
上限偏差温度	50℃
下限偏差温度	50℃
上下限偏差警報のヒステリシス	3℃
PID 演算のサンプリング周期	100ms
出力制御周期	1s

構成・設定

入力ユニット形 CJ1W-PH41U の設定は、以下のとおりとします。

項目名	設定値
Input1:Input signal type	K(1)
Input2:Input signal type	K(1)

項目名	設定値
Input3:Input signal type	K(1)
Input4:Input signal type	K(1)

I/O マップ設定は、以下のとおりとします。

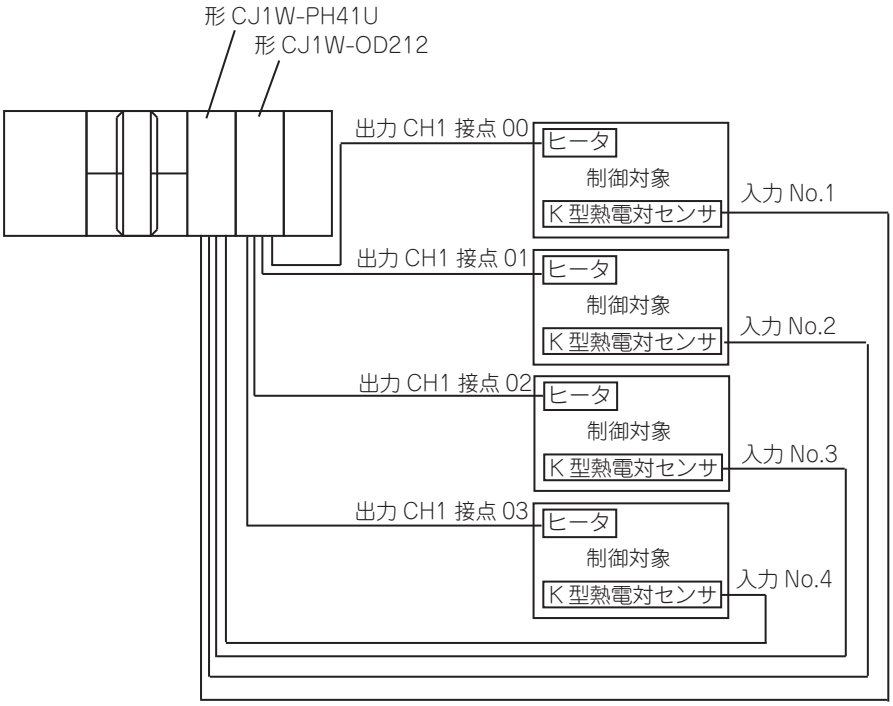
ユニット	ポート	説明	変数
形 CJ1W-PH41U	Ch1_AIInPV	入力 No.1 測定値（INT 型）	AI1
	Ch2_AIInPV	入力 No.2 測定値（INT 型）	AI2
	Ch3_AIInPV	入力 No.3 測定値（INT 型）	AI3
	Ch4_AIInPV	入力 No.4 測定値（INT 型）	AI4
形 CJ1W-OD212	Ch1_Out00	出力 CH1 接点 00	DO1
	Ch1_Out01	出力 CH1 接点 01	DO2
	Ch1_Out02	出力 CH1 接点 02	DO3
	Ch1_Out03	出力 CH1 接点 03	DO4

4 点の温度制御の、入力と出力の対応は、以下のとおりとします。

入力	出力
AI1	DO1
AI2	DO2
AI3	DO3
AI4	DO4

プログラムを割り付けるタスクの周期は、1ms とします。

● **構成図**



処理内容

以下の処理を 4 点分行います。

1 **測定温度を取得します。**

2 **LimitAlarm_REAL 命令を使って、測定温度に対して上下限警報を出力します。**

3 **LimitAlarm_REAL 命令で異常が発生した場合、または上下限警報を出力した場合は、安全対策のための出力をします。**

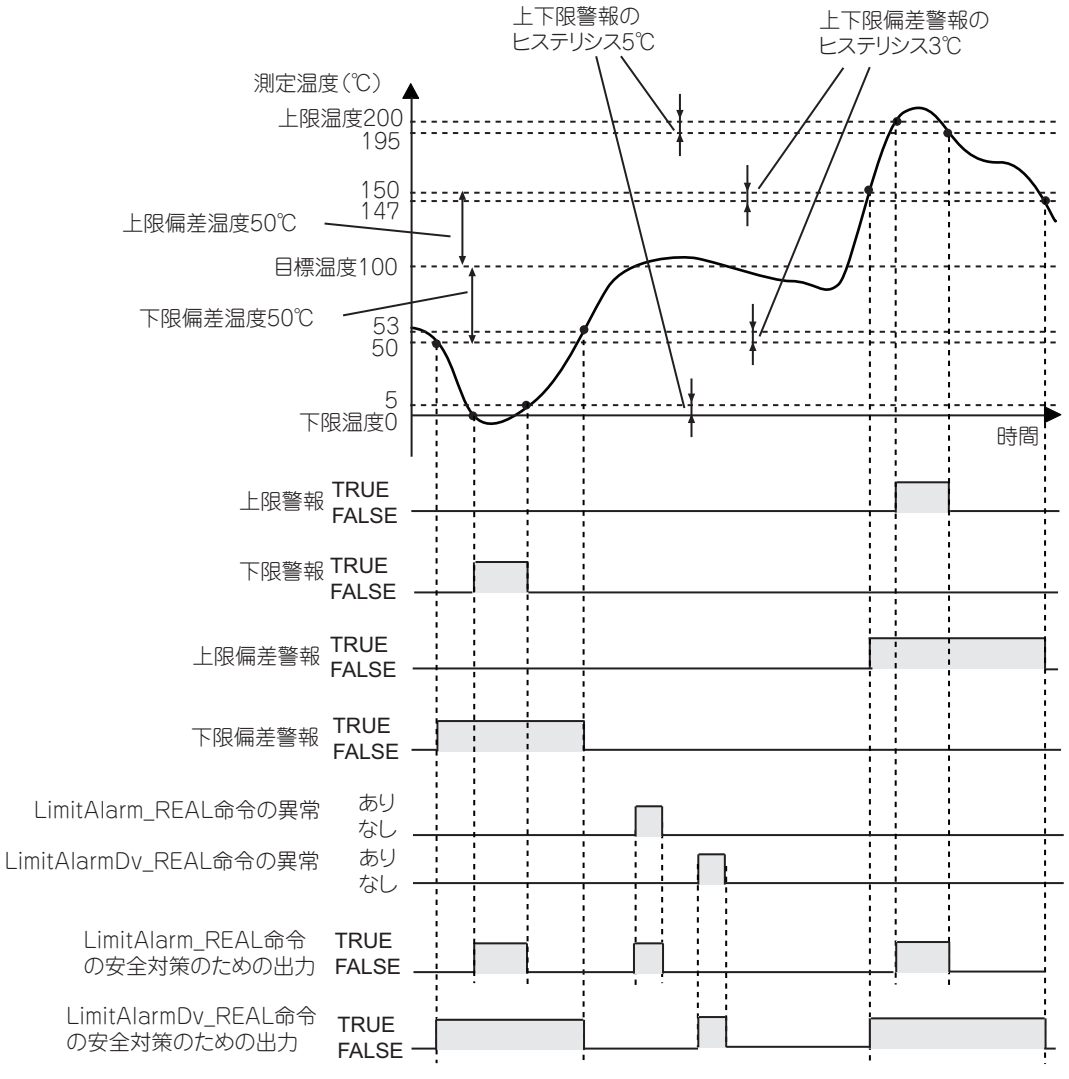
4 **LimitAlarmDv_REAL 命令を使って、目標温度からの測定温度の偏差に対して上下限偏差警報を出力します。**

5 **LimitAlarmDv_REAL 命令で異常が発生した場合、または上下限偏差警報を出力した場合は、安全対策のための出力をします。**

6 **PIDAT 命令を使って温度制御します。**

7 **TimeProportionalOut 命令を使って、操作量を時分割比例でヒータに出力します。**

● 上下限警報、上下限偏差警報の動作



グローバル変数の定義

● グローバル変数

名称	データ型	割付先 *1	コメント
AI1	INT	IOBus://rack#0/slot#0/Ch1_AllnPV	入力 No.1 測定値（INT 型）
AI2	INT	IOBus://rack#0/slot#0/Ch2_AllnPV	入力 No.2 測定値（INT 型）
AI3	INT	IOBus://rack#0/slot#0/Ch3_AllnPV	入力 No.3 測定値（INT 型）
AI4	INT	IOBus://rack#0/slot#0/Ch4_AllnPV	入力 No.4 測定値（INT 型）
DO1	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00	出力 CH1 接点 00
DO2	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out01	出力 CH1 接点 01
DO3	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out02	出力 CH1 接点 02
DO4	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out03	出力 CH1 接点 03

*1. 形 CJ1W-PH41U をラック番号 0 のスロット番号 0、形 CJ1W-OD212 をラック番号 0 のスロット番号 1 に装着した場合の割付先です。

（注）I/O マップ設定により、各ユニットのポートに割り付けたグローバル変数が自動的に生成されます。

LD

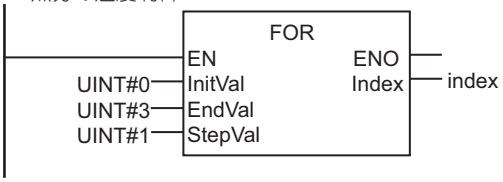
内部変数	名称	データ型	初期値	保持	コメント
	index	UINT	0	□	ループインデックス
	LimitAlarm_ON	BOOL	True	□	上下限警報実行
	LimitAlarmDv_ON	BOOL	True	□	上下限偏差警報実行
	TimeProportionalOut_ON	BOOL	True	□	時分割比例出力実行
	AI	INT	0	□	現在値
	PV	ARRAY[0..3] OF REAL	[4(0.0)]	□	測定値
	SP	ARRAY[0..3] OF REAL	[4(100)]	□	目標温度
	DOut_TPO	BOOL	False	□	時分割比例出力
	HighVal	ARRAY[0..3] OF REAL	[4(200)]	□	上下限警報の上限設定値
	LowVal	ARRAY[0..3] OF REAL	[4(0.0)]	□	上下限警報の下限設定値
	Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]	□	上下限警報のヒステリシス
	Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	上下限警報出力
	HighAlm	ARRAY[0..3] OF BOOL	[4(False)]	□	上限警報
	LowAlm	ARRAY[0..3] OF BOOL	[4(False)]	□	下限警報
	Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	LimitAlarm_REAL 命令の異常
	Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	上下限警報の安全対策のための出力

内部変数	名称	データ型	初期値	保持	コメント
	DvHighVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の上限偏差設定値
	DvLowVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の下限偏差設定値
	Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報出力
	HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限偏差警報
	LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限偏差警報
	Error_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarmDv_REAL 命令の異常
	Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]	☐	上下限偏差警報のヒステリシス
	Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報の安全対策のための出力
	Run	ARRAY[0..3] OF BOOL	[4(False)]	☐	実行条件
	ManCtl	ARRAY[0..3] OF BOOL	[4(False)]	☐	マニュアル/オート切替
	StartAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行条件
	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)	☐	運転中設定パラメータ
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=-10.0, RngUpLmt:=1000.0, DirOpr:=False)	☐	初期設定パラメータ
	PB	ARRAY[0..3] OF REAL	[4(10)]	☑	比例帯
	TI	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	積分時間
	TD	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	微分時間
	ManMV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	マニュアル操作量
	ATDone	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング正常終了
	ATBusy	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行中
	Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	PIDAT 命令の異常
	ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]	☐	PIDAT 命令の異常 ID
	MV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	操作量
	CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]	☐	制御周期

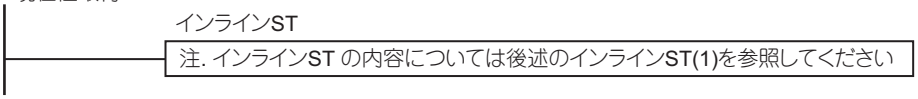
内部変数	名称	データ型	初期値	保持	コメント
	MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]	□	最小パルス幅
	Delay	ARRAY[0..3] OF REAL	[4(0.0)]	□	オンディレー時間
	Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]	□	TimeProportionalOut 命令の異常
	LimitAlarm_REAL_instance	ARRAY[0..3] OF LimitAlarm_REAL		□	
	LimitAlarmDv_REAL_instance	ARRAY[0..3] OF LimitAlarmDv_REAL		□	
	PIDAT_instance	ARRAY[0..3] OF PIDAT		□	
	TimeProportionalOut_instance	ARRAY[0..3] OF TimeProportionalOut		□	

外部変数	名称	データ型	コメント
	AI1	INT	入力 No.1（測定値）
	AI2	INT	入力 No.2（測定値）
	AI3	INT	入力 No.3（測定値）
	AI4	INT	入力 No.4（測定値）
	DO1	BOOL	出力 CH1（接点 00）
	DO2	BOOL	出力 CH1（接点 01）
	DO3	BOOL	出力 CH1（接点 02）
	DO4	BOOL	出力 CH1（接点 03）

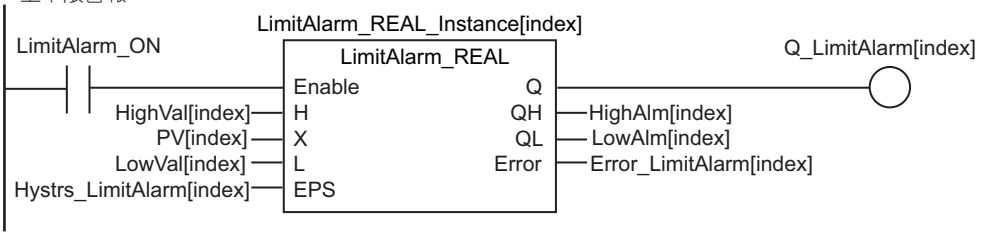
4点分の温度制御



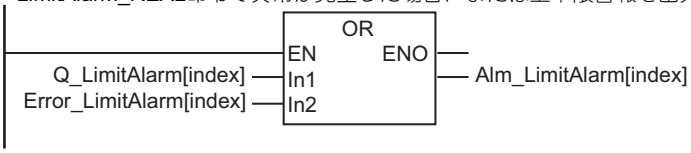
現在値取得

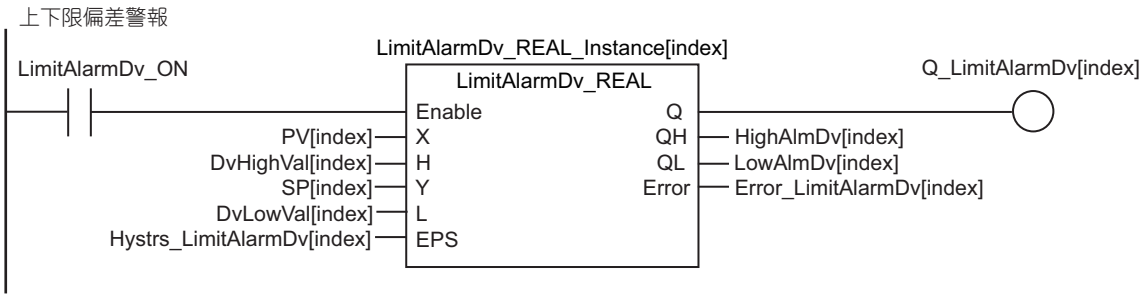


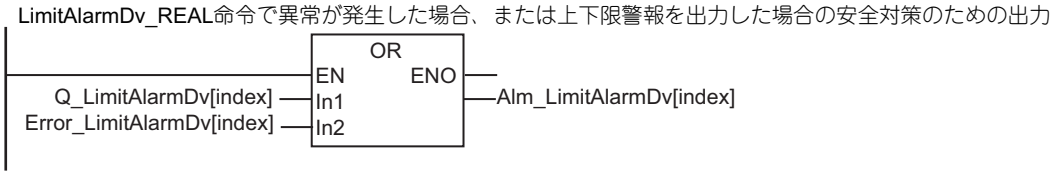
上下限警報

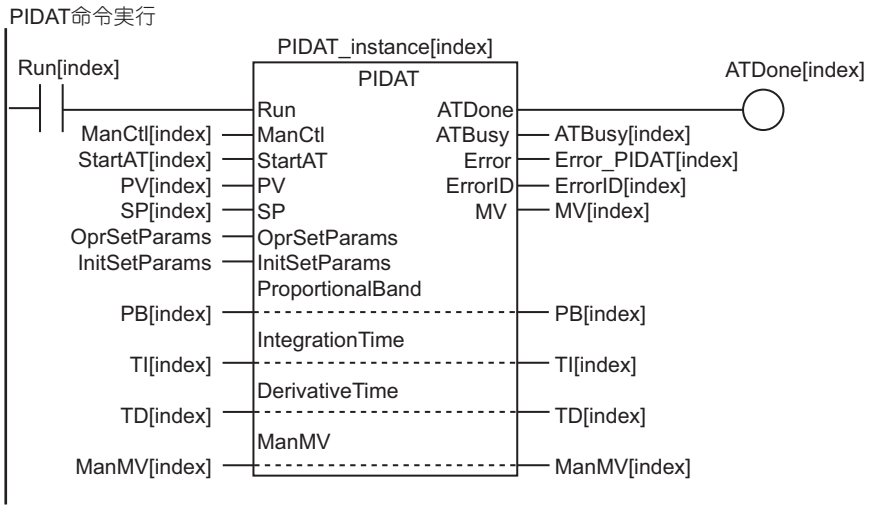


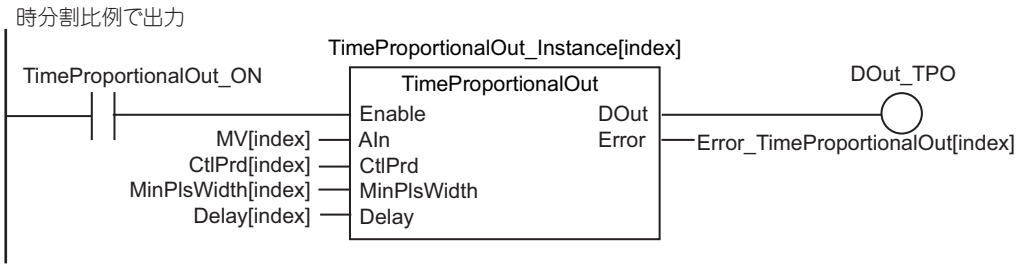
LimitAlarm_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力

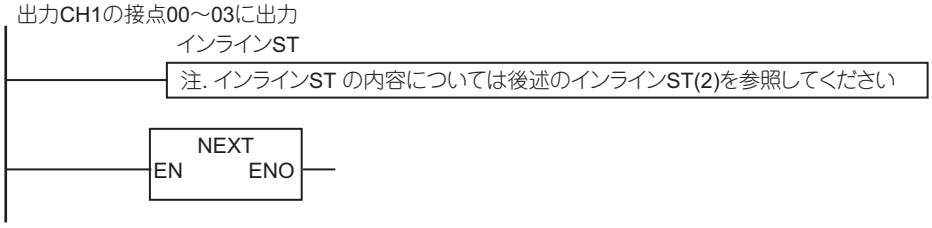












● インライン ST (1)の内容

```
//入力 1～4 の値を取得
CASE index OF
INT#0:
```

```
  AI:=AI1;
INT#1:
  AI:=AI2;
INT#2:
  AI:=AI3;
ELSE
  AI:=AI4;
END_CASE;

//現在値 AI を実数に変換
PV[index]:=INT_TO_REAL(AI)/REAL#10.0;  // 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算
```

● インライン ST (2)の内容

```
//出力 CH1 の接点 00～03 に出力
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

内部変数	名称	データ型	初期値	保持	コメント
	index	UINT	0	□	ループインデックス
	LimitAlarm_ON	BOOL	True	□	上下限警報実行
	LimitAlarmDv_ON	BOOL	True	□	上下限偏差警報実行
	TimeProportionalOut_ON	BOOL	True	□	時分割比例出力実行
	AI	INT	0	□	現在値
	PV	ARRAY[0..3] OF REAL	[4(0.0)]	□	測定値
	SP	ARRAY[0..3] OF REAL	[4(100)]	□	目標温度
	DOut_TPO	BOOL	False	□	時分割比例出力
	HighVal	ARRAY[0..3] OF REAL	[4(200)]	□	上下限警報の上限設定値
	LowVal	ARRAY[0..3] OF REAL	[4(0.0)]	□	上下限警報の下限設定値
	Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]	□	上下限警報のヒステリシス
	Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	上下限警報出力

内部変数	名称	データ型	初期値	保持	コメント
	HighAlm	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限警報
	LowAlm	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限警報
	Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarm_REAL 命令の異常
	Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限警報の安全対策のための出力
	DvHighVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の上限偏差設定値
	DvLowVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の下限偏差設定値
	Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報出力
	HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限偏差警報
	LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限偏差警報
	Error_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarmDv_REAL 命令の異常
	Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]	☐	上下限偏差警報のヒステリシス
	Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報の安全対策のための出力
	Run	ARRAY[0..3] OF BOOL	[4(False)]	☐	実行条件
	ManCtl	ARRAY[0..3] OF BOOL	[4(False)]	☐	マニュアル/オート切替
	StartAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行条件
	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)	☐	運転中設定パラメータ
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=-10.0,RngUpLmt:=1000.0, DirOpr:=False)	☐	初期設定パラメータ
	PB	ARRAY[0..3] OF REAL	[4(10)]	☑	比例帯
	TI	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	積分時間
	TD	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	微分時間
	ManMV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	マニュアル操作量
	ATDone	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング正常終了
	ATBusy	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行中

内部変数	名称	データ型	初期値	保持	コメント
	Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]	□	PIDAT 命令の異常
	ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]	□	PIDAT 命令の異常 ID
	MV	ARRAY[0..3] OF REAL	[4(0.0)]	□	操作量
	CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]	□	制御周期
	MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]	□	最小パルス幅
	Delay	ARRAY[0..3] OF REAL	[4(0.0)]	□	オンディレー時間
	Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]	□	TimeProportionalOut 命令の異常
	LimitAlarm_REAL_instance	ARRAY[0..3] OF LimitAlarm_REAL		□	
	LimitAlarmDv_REAL_instance	ARRAY[0..3] OF LimitAlarmDv_REAL		□	
	PIDAT_instance	ARRAY[0..3] OF PIDAT		□	
	TimeProportionalOut_instance	ARRAY[0..3] OF TimeProportionalOut		□	

外部変数	名称	データ型	コメント
	AI1	INT	入力 No.1（測定値）
	AI2	INT	入力 No.2（測定値）
	AI3	INT	入力 No.3（測定値）
	AI4	INT	入力 No.4（測定値）
	DO1	BOOL	出力 CH1（接点 00）
	DO2	BOOL	出力 CH1（接点 01）
	DO3	BOOL	出力 CH1（接点 02）
	DO4	BOOL	出力 CH1（接点 03）

```
//4 点分の温度制御
FOR index:=UINT#0 TO UINT#3 BY UINT#1 DO

  //入力 1～4 の値を取得
  CASE index OF
  INT#0:
    AI:=AI1;
  INT#1:
    AI:=AI2;
  INT#2:
    AI:=AI3;
  ELSE
    AI:=AI4;
  END_CASE;

  //現在値 AI を実数に変換
  PV[index]:=INT_TO_REAL(AI)/REAL#10.0; // 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算
```

```
//上下限警報
LimitAlarm_REAL_instance[index](
  Enable :=LimitAlarm_ON,
  H :=HighVal[index],
  X :=PV[index],
  L :=LowVal[index],
  EPS :=Hystrs_LimitAlarm[index],
  Q =>Q_LimitAlarm[index],
  QH =>HighAlm[index],
  QL =>LowAlm[index],
  Error =>Error_LimitAlarm[index]);

//LimitAlarm_REAL 命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力
Alm_LimitAlarm[index]:=Q_LimitAlarm[index] OR Error_LimitAlarm[index];

//上下限偏差警報
LimitAlarmDv_REAL_instance[index](
  Enable :=LimitAlarmDv_ON,
  X :=PV[index],
  H :=DvHighVal[index],
  Y :=SP[index],
  L :=DvLowVal[index],
  EPS :=Hystrs_LimitAlarmDv[index],
  Q =>Q_LimitAlarmDv[index],
  QH =>HighAlmDv[index],
  QL =>LowAlmDv[index],
  Error =>Error_LimitAlarmDv[index]);

//LimitAlarmDv_REAL 命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力
Alm_LimitAlarmDv[index]:=Q_LimitAlarmDv[index] OR Error_LimitAlarmDv[index];

// PIDAT 命令実行
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]);

// 時分割比例で出力
```

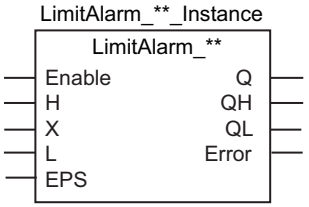
```
  TimeProportionalOut_instance[index](
    Enable :=TimeProportionalOut_ON,
    AIn :=MV[index],
    CtlPrd :=CtlPrd[index],
    MinPlsWidth :=MinPlsWidth[index],
    Delay :=Delay[index],
    DOut =>DOut_TPO,
    Error =>Error_TimeProportionalOut[index]);

  //出力 Ch1 の接点 00～03 に出力
  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_**

入力値が、下限設定値未満の値、または上限設定値を超える値の場合に警報を出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LimitAlarm_**	上下限警報グループ	FB	 ただし、“**”は REAL, LREAL のいずれか	LimitAlarm_**_instance(Enable, H, X, L, EPS, Q, QH, QL, Error); ただし、“**”は REAL, LREAL のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	動作指令	入力	TRUE: 実行 FALSE: 警報リセット	データ型に従う	–	FALSE
H	上限設定値		入力値の上限設定値			0
X	入力値		監視の対象となる値			
L	下限設定値		入力値の下限設定値			
EPS	ヒステリシス		警報のヒステリシス	データ型に従う *1		
Q	警報出力	出力	TRUE: 上限警報と下限警報のいずれかが ON FALSE: 上限警報と下限警報がいずれも OFF	データ型に従う	–	–
QH	上限警報		TRUE: 上限警報 ON FALSE: 上限警報 OFF			
QL	下限警報		TRUE: 下限警報 ON FALSE: 下限警報 OFF			

*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
H														○	○					
X	「H」と同じデータ型																			
L	「H」と同じデータ型																			
EPS	「H」と同じデータ型																			
Q	○																			
QH	○																			
QL	○																			

機能

入力値が、下限設定値から上限設定値の間の値であるかを監視ます。
入力値が、下限設定値未満の値、または上限設定値を超える値の場合は警報を出力します。
温度制御で測定温度の監視などに使用します。

動作指令「Enable」が TRUE の間、入力値「X」の値を監視します。

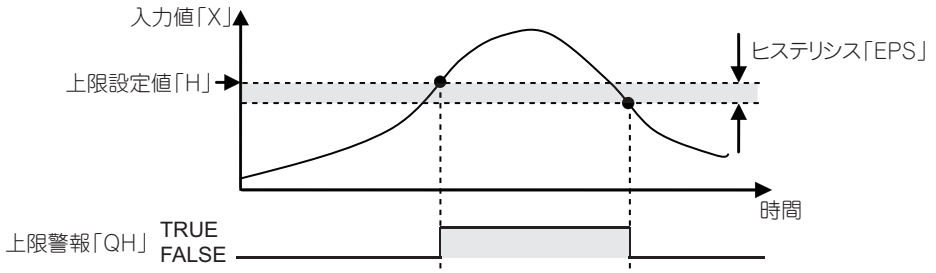
「X」の値が上限設定値「H」の値を超える場合、上限警報「QH」は TRUE となります。
「X」の値が下限設定値「L」の値未満の場合、下限警報「QL」は TRUE となります。
「QH」、「QL」のいずれかの値が TRUE の場合、警報出力「Q」が TRUE になります。
「X」、「H」、「L」、ヒステリシス「EPS」の値は、「Enable」が TRUE の間、常時反映されます。

「Enable」が FALSE の場合は、警報をリセットします。リセットにより、「Q」、「QH」、「QL」はすべて FALSE となります。

「H」、「X」、「L」、「EPS」のデータ型は、REAL、LREAL のいずれかです。
命令名称は、「H」、「X」、「L」、「EPS」のデータ型によって異なります。
たとえば、命令名称が LimitAlarm_LREAL の場合、「H」、「X」、「L」、「EPS」のデータ型は、すべて LREAL となります。

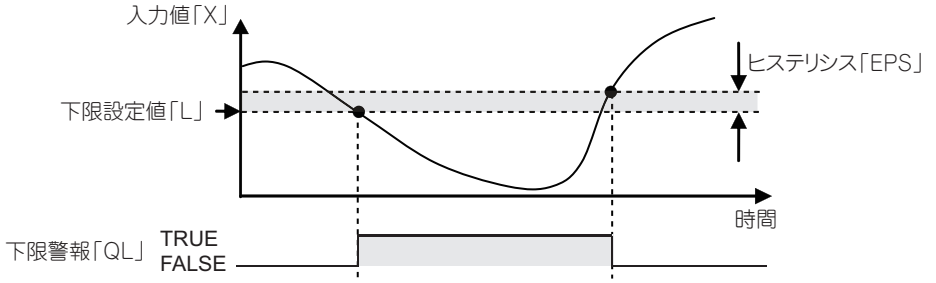
上限警報「QH」の動作

上限警報「QH」の値は、以下のように変化します。
ヒステリシスの設定により、警報がハンティングすることを防止できます。
- 入力値「X」＞上限設定値「H」の場合、TRUE となります。
- 入力値「X」＜上限設定値「H」－ヒステリシス「EPS」の場合、FALSE となります。



下限警報「QL」の動作

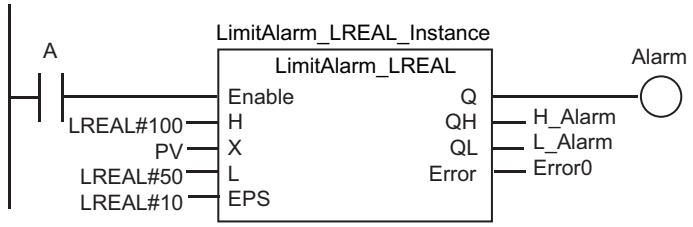
下限警報「QL」の値は、以下のように変化します。
ヒステリシスの設定により、警報がハンティングすることを防止できます。
- 入力値「X」＜下限設定値「L」の場合、TRUE となります。
- 入力値「X」＞下限設定値「L」＋ヒステリシス「EPS」の場合、FALSE となります。



記述例

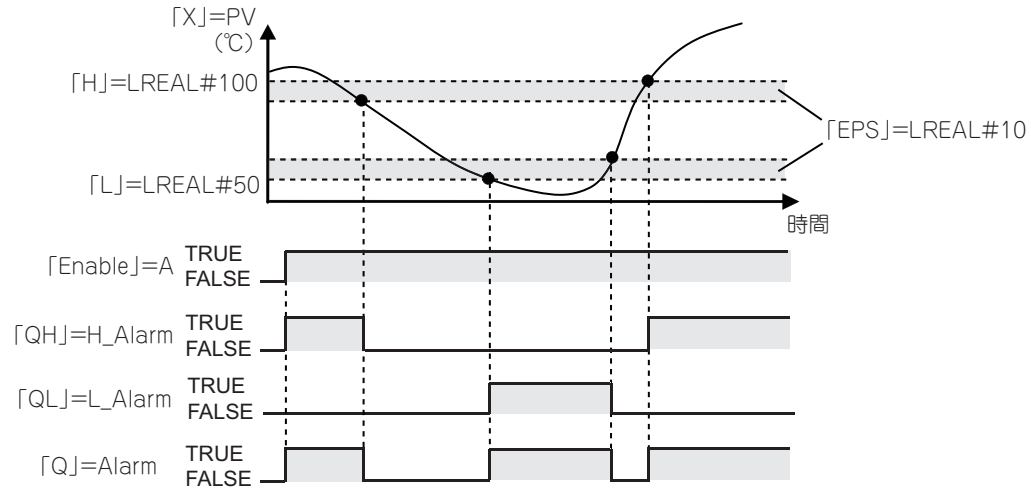
上限設定値「H」が 100°C、下限設定値「L」が 50°C、ヒステリシス「EPS」が 10°C の場合の記述例を以下に示します。

LD



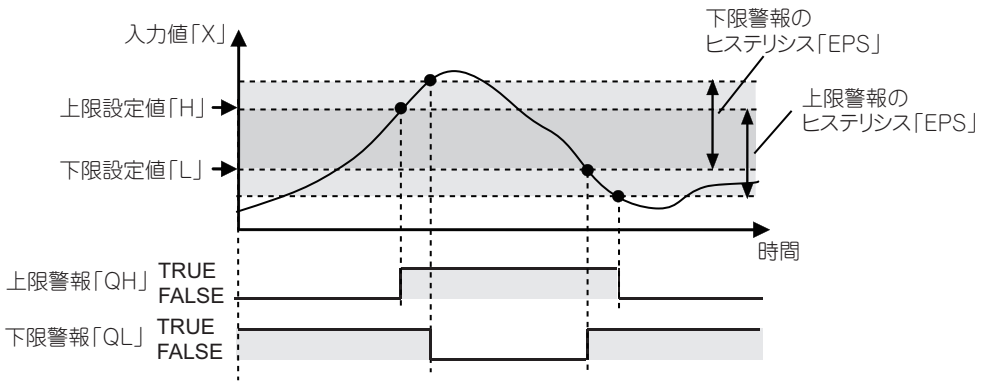
ST

```
LimitAlarm_LREAL_instance(A,LREAL#100,PV,LREAL#50,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

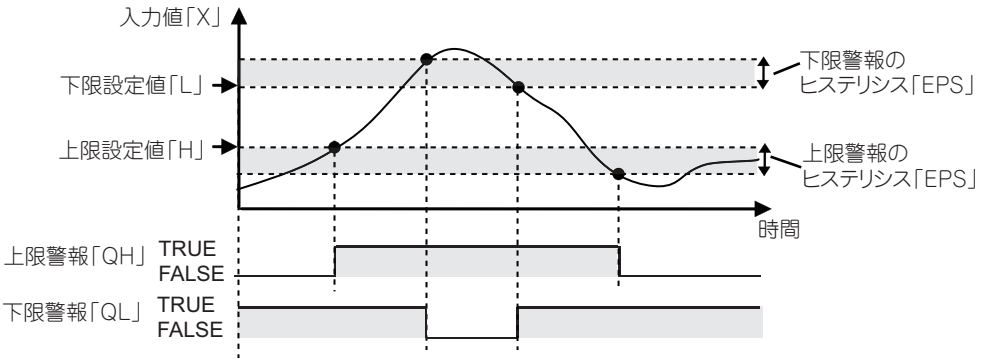


参考

- 命令実行時間を短縮したい場合は、LimitAlarm_REAL 命令を使用してください。
- 「H」－「L」＜「EPS」となる設定も可能です。この場合、「QH」と「QL」が両方とも TRUE になることがあります。



- 「H」＜「L」となる設定も可能です。この場合、「QH」か「QL」のいずれかが常に TRUE になります。



使用上の注意

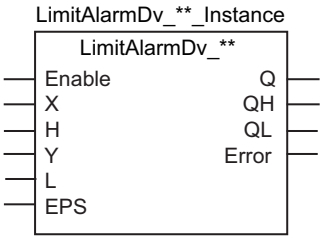
- 「EPS」の値が有効範囲外の場合、異常となります。「Error」は TRUE となり、「Q」、「QH」、「QL」は FALSE となります。
- 警報を出力した場合に温度制御の出力を OFF するなど、本命令を安全対策の用途に使用することができます。この場合は、異常により「Q」、「QH」、および「QL」が FALSE になっても安全対策ができるように使用してください。使用例は、TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

サンプルプログラム

TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

LimitAlarmDv_**

基準値からの入力値の偏差が、下限偏差設定値または上限偏差設定値を超える値の場合に警報を出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LimitAlarmDv_**	上下限偏差警報グループ	FB	 ただし、“**”は REAL, LREAL のいずれか	LimitAlarmDv_**instance(Enable, X, H, Y, L, EPS, Q, QH, QL, Error); ただし、“**”は REAL, LREAL のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	動作指令	入力	TRUE: 実行 FALSE: 警報リセット	データ型に従う	–	FALSE
X	入力値		監視の対象となる値			0
H	上限偏差設定値		基準値に対して上方の偏差警報の設定値			
Y	基準値		偏差の基準値			
L	下限偏差設定値		基準値に対して下方の偏差警報の設定値			
EPS	ヒステリシス		警報のヒステリシス	データ型に従う *1		
Q	偏差警報出力	出力	TRUE: 上限偏差警報と下限偏差警報のいずれかが ON FALSE: 上限偏差警報と下限偏差警報がいずれも OFF	データ型に従う	–	–
QH	上限偏差警報		TRUE: 上限偏差警報 ON FALSE: 上限偏差警報 OFF			
QL	下限偏差警報		TRUE: 下限偏差警報 ON FALSE: 下限偏差警報 OFF			

*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
X														○	○					
H	「X」と同じデータ型																			
Y	「X」と同じデータ型																			
L	「X」と同じデータ型																			
EPS	「X」と同じデータ型																			
Q	○																			
QH	○																			
QL	○																			

機能

基準値からの入力値の偏差が、下限偏差設定値または上限偏差設定値までの値であるかを監視します。
偏差が、下限偏差設定値または上限偏差設定値を超える値の場合は警報を出力します。
温度制御で、目標温度からの測定温度の偏差を監視する場合などに使用します。

動作指令「Enable」が TRUE の間、基準値「Y」からの入力値「X」の偏差を監視します。
「Y」からの「X」の上方の偏差が上限偏差設定値「H」の値を超える場合、上限偏差警報「QH」は TRUE となります。
「Y」からの「X」の下方の偏差が下限偏差設定値「L」の値を超える場合、下限偏差警報「QL」は TRUE となります。
「QH」、「QL」のいずれかの値が TRUE の場合、警報出力「Q」が TRUE になります。
「X」、「H」、「Y」、「L」、ヒステリシス「EPS」の値は、「Enable」が TRUE の間、常時反映されます。

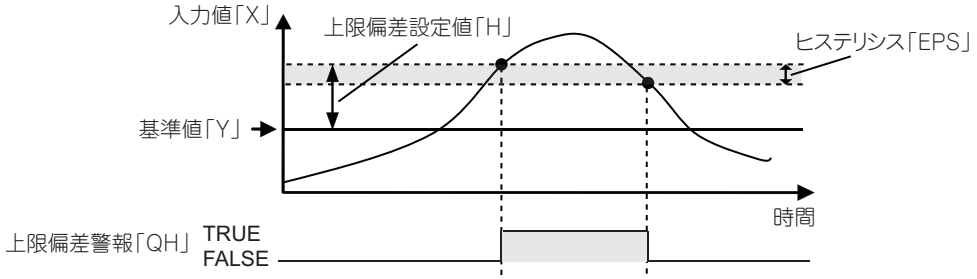
「Enable」が FALSE の場合は、警報をリセットします。リセットにより、「Q」、「QH」、「QL」はすべて FALSE となります。

「X」、「H」、「Y」、「L」、「EPS」のデータ型は、REAL、LREAL のいずれかです。
命令名称は、「X」、「H」、「Y」、「L」、「EPS」のデータ型によって異なります。
たとえば、命令名称が LimitAlarmDv_LREAL の場合、「X」、「H」、「Y」、「L」、「EPS」のデータ型は、すべて LREAL となります。

上限偏差警報「QH」の動作

上限偏差警報「QH」は、基準値「Y」に対する上方の偏差警報です。「QH」の値は以下のように変化します。ヒステリシスにより、偏差警報がハンティングすることを防止できます。

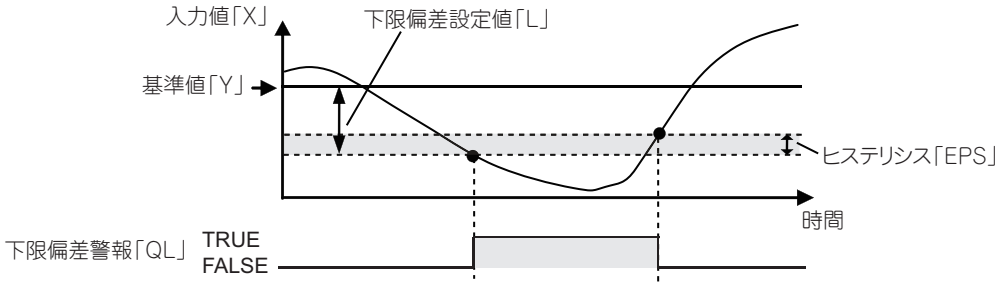
- 入力値「X」－基準値「Y」＞上限偏差設定値「H」の場合、TRUE となります。
- 入力値「X」－基準値「Y」＜上限偏差設定値「H」－ヒステリシス「EPS」の場合、FALSE となります。



下限偏差警報「QL」の動作

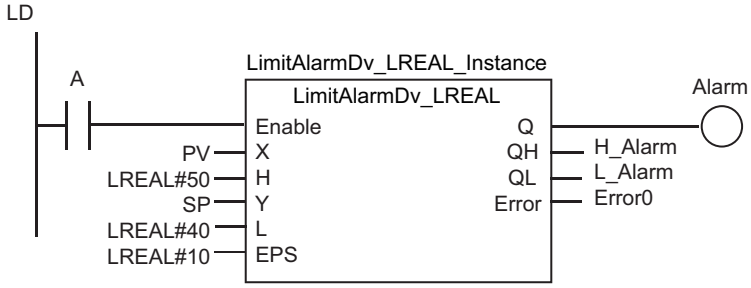
下限偏差警報「QL」は、基準値「Y」に対する下方の偏差警報です。「QL」の値は以下のように変化します。ヒステリシスにより、偏差警報がハンティングすることを防止できます。

- －（入力値「X」－基準値「Y」）＞下限偏差設定値「L」の場合、TRUE となります。
- －（入力値「X」－基準値「Y」）＜下限偏差設定値「L」－ヒステリシス「EPS」の場合、FALSE となります。



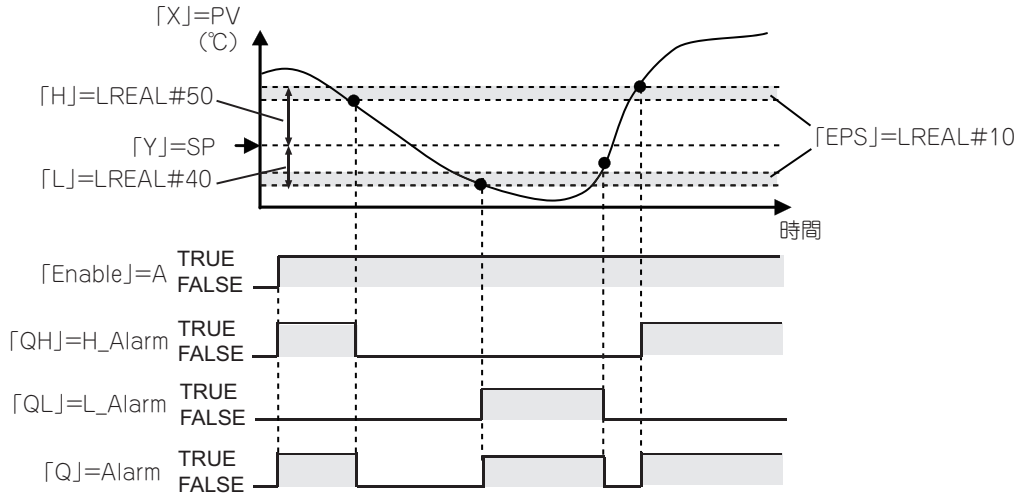
記述例

上限偏差設定値「H」が 50℃、下限偏差設定値「L」が 40℃、ヒステリシス「EPS」が 10℃ の場合の記述例を以下に示します。



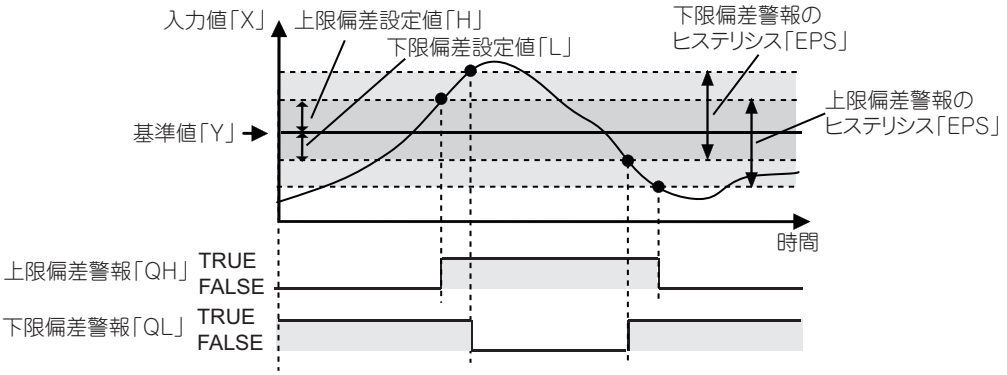
ST

```
LimitAlarmDv_LREAL_instance(A,PV,LREAL#50,SP,LREAL#40,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

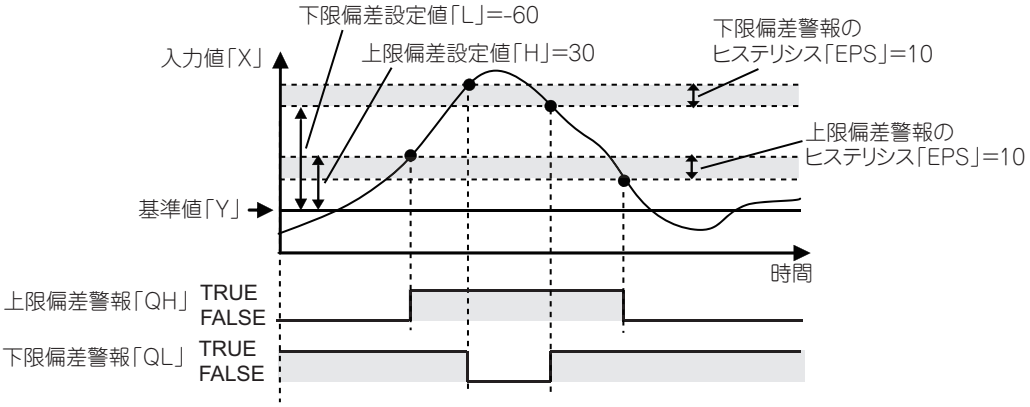


参考

- 命令実行時間を短縮したい場合は、LimitAlarmDv_REAL 命令を使用してください。
- 「H」＋「L」＜「EPS」となる設定も可能です。この場合、「QH」と「QL」が両方とも TRUE になることがあります。



- 「H」＋「L」＜0 となる設定も可能です。この場合、「QH」と「QL」のいずれかが常に TRUE になります。たとえば、「L」の値が-60、「H」の値が 30 の場合は、下図のように動作します。



使用上の注意

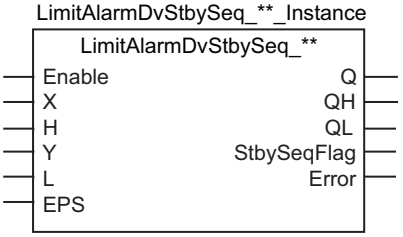
- 「EPS」の値が有効範囲外の場合、異常となります。「Error」は TRUE となり、「Q」、「QH」、「QL」は FALSE となります。
- 偏差警報を出力した場合に温度制御の出力を OFF するなど、本命令を安全対策の用途に使用することができます。この場合は、異常により「Q」、「QH」、および「QL」が FALSE になっても安全対策ができるように使用してください。使用例は、TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

サンプルプログラム

TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

LimitAlarmDvStbySeq_**

待機シーケンス付きで上下限偏差警報を出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
LimitAlarmDvStbySeq_**	待機シーケンス付き上下限偏差警報グループ	FB	 ただし、“**”は REAL、LREAL のいずれか	LimitAlarmDvStbySeq_**_instance(Enable, X, H, Y, L, EPS, Q, QH, QL, StbySeqFlag, Error); ただし、“**”は REAL、LREAL のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	動作指令	入力	TRUE: 実行 FALSE: 警報リセット	データ型に従う	–	FALSE
X	入力値		偏差警報の対象となる値			
H	上限偏差設定値		基準値に対して上方の偏差警報の設定値			0
Y	基準値		偏差の基準値			
L	下限偏差設定値		基準値に対して下方の偏差警報の設定値			
EPS	ヒステリシス		警報のヒステリシス	データ型に従う *1		
Q	偏差警報出力	出力	TRUE: 上限偏差警報と下限偏差警報のいずれかが ON FALSE: 上限偏差警報と下限偏差警報がいずれも OFF	データ型に従う	–	–
QH	上限偏差警報		TRUE: 上限偏差警報 ON FALSE: 上限偏差警報 OFF			
QL	下限偏差警報		TRUE: 下限偏差警報 ON FALSE: 下限偏差警報 OFF			
StbySeqFlag	待機シーケンス有効フラグ		TRUE: 有効 FALSE: 無効			

*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
X														○	○					
H	「X」と同じデータ型																			
Y	「X」と同じデータ型																			
L	「X」と同じデータ型																			
EPS	「X」と同じデータ型																			
Q	○																			
QH	○																			
QL	○																			
StbySeqFlag	○																			

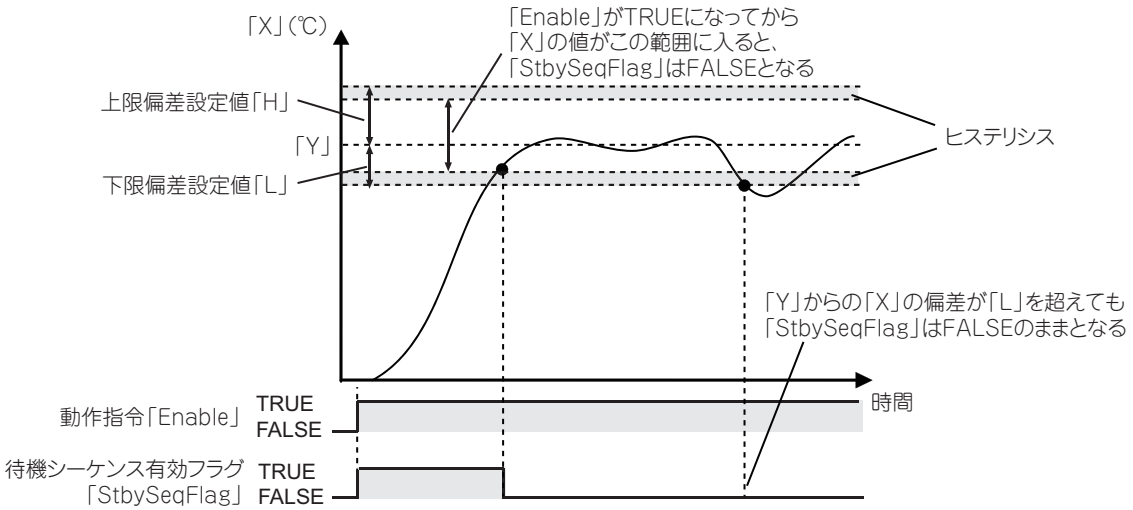
機能

基準値からの入力値の偏差が、下限偏差設定値まで、かつ上限偏差設定値までの値であるかを監視します。
偏差が、下限偏差設定値または上限偏差設定値を超える値の場合は警報を出力します。ただし、いったん偏差が、下限偏差設定値まで、かつ上限偏差設定値までの値になるまでは警報を出力しません。温度制御で、測定温度が安定するまでは偏差警報を出力させない場合などに使用します。

動作指令「Enable」が TRUE の間、基準値「Y」からの入力値「X」の偏差を監視します。ただし、待機シーケンス有効フラグ「StbySeqFlag」が TRUE の間は、偏差を監視しません。
「Y」からの「X」の上方の偏差が上限偏差設定値「H」の値を超える場合、上限偏差警報「QH」は TRUE となります。
「Y」からの「X」の下方の偏差が下限偏差設定値「L」の値を超える場合、下限偏差警報「QL」は TRUE となります。
「QH」、「QL」のいずれかの値が TRUE の場合、警報出力「Q」が TRUE になります。
「X」、「H」、「Y」、「L」、「EPS」の値は、「Enable」が TRUE の間、常時反映されます。

「Enable」が FALSE の場合は、警報をリセットします。リセットにより、「Q」、「QH」、「QL」、「StbySeqFlag」はすべて FALSE となります。

「StbySeqFlag」は、「Enable」が TRUE になってから以下の条件をいずれも満たすと、FALSE になります。
「StbySeqFlag」は、一度 FALSE になると、「Enable」が FALSE から TRUE に変化するまでは TRUE になりません。
- 入力値「X」－基準値「Y」＜上限偏差設定値「H」－ヒステリシス「EPS」
- －（入力値「X」－基準値「Y」）＜下限偏差設定値「L」－ヒステリシス「EPS」

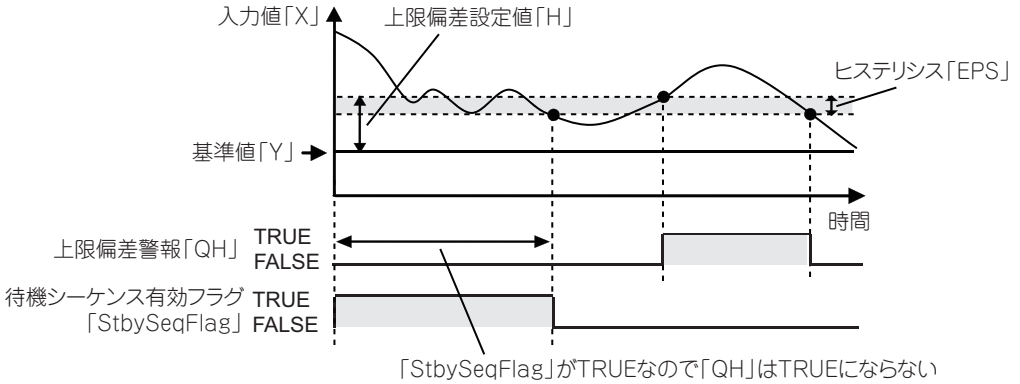


「X」、「H」、「Y」、「L」、「EPS」のデータ型は、REAL、LREAL のいずれかです。
命令名称は、「X」、「H」、「Y」、「L」、「EPS」のデータ型によって異なります。
たとえば、命令名称が LimitAlarmDvStbySeq_LREAL の場合、「X」、「H」、「Y」、「L」、「EPS」のデータ型は、すべて LREAL となります。

上限偏差警報「QH」の動作

上限偏差警報「QH」は、基準値「Y」に対する上方の偏差警報です。
「StbySeqFlag」が FALSE の場合、「QH」の値は、以下のように変化します。
ヒステリシスを設定することで、偏差警報がハンティングすることを防止できます。

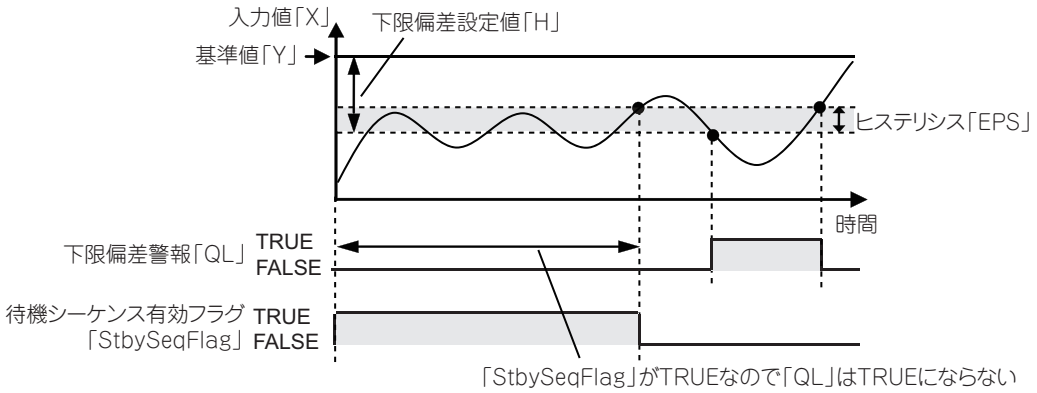
- 入力値「X」－基準値「Y」＞上限偏差設定値「H」の場合、TRUE となります。
- 入力値「X」－基準値「Y」＜上限偏差設定値「H」－ヒステリシス「EPS」の場合、FALSE となります。



下限偏差警報「QL」の動作

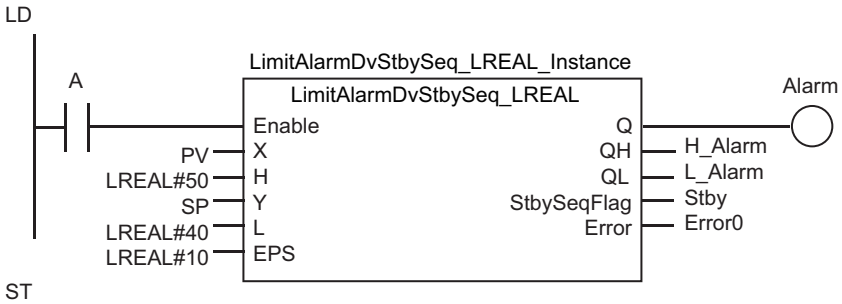
下限偏差警報「QL」は、基準値「Y」に対する下方の偏差警報です。
「StbySeqFlag」が FALSE の場合、「QL」の値は、以下のように変化します。
ヒステリシスを設定することで、偏差警報がハンティングすることを防止できます。

- －（入力値「X」－基準値「Y」）＞下限偏差警報「L」の場合、TRUE となります。
- －（入力値「X」－基準値「Y」）＜下限偏差警報「L」－ヒステリシス「EPS」の場合、FALSE となります。

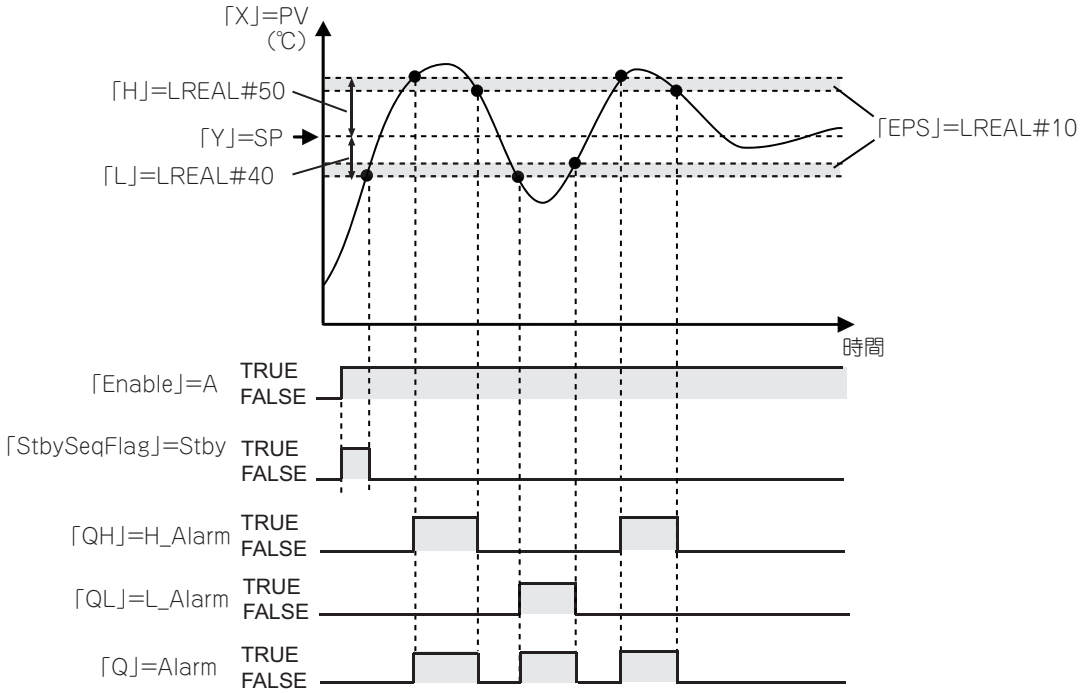


記述例

上限偏差設定値「H」が 50℃、下限偏差設定値「L」が 40℃、ヒステリシス「EPS」が 10℃ の場合の記述例を以下に示します。




```
ST
LimitAlarmDvStbySeq_LREAL_Instance(A,PV,LREAL#50,SP,LREAL#40,LREAL#10,Alarm,H_Alarm,L_Alarm,Stby,Error0);
```



参考

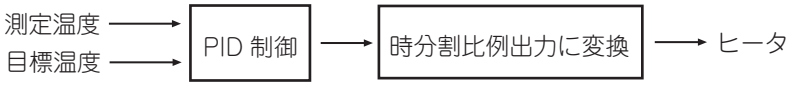
- 命令実行時間を短縮したい場合は、LimitAlarmDvStbySeq_REAL 命令を使用してください。
- 「H」＋「L」＜「EPS」となる設定も可能です。この場合、「QH」と「QL」が両方とも TRUE になることがあります。「LimitAlarmDv_** (P.2-754)」命令を参照してください。
- 「H」＋「L」＜ 0 となる設定も可能です。この場合、「StbySeqFlag」が FALSE の間は、「QH」と「QL」のいずれかが常に TRUE になります。「LimitAlarmDv_** (P.2-754)」命令を参照してください。

使用上の注意

- 「EPS」の値が有効範囲外の場合、異常となります。「Error」は TRUE となり、「Q」、「QH」、「QL」は FALSE となります。
- 偏差警報を出力した場合に温度制御の出力を OFF するなど、本命令を安全対策の用途に使用することができます。この場合は、異常により「Q」、「QH」、および「QL」が FALSE になっても安全対策ができるように使用してください。使用例は、「サンプルプログラム (P.2-762)」を参照してください。

サンプルプログラム

上下限警報、待機シーケンス付き上下限偏差警報付きの、4 点の温度制御を行います。
制御方式は PID 制御を用い、PID 制御の操作量を時分割比例出力に変換し、ヒータに出力します。



仕様

以下のような仕様で、温度制御を行います。

項目	仕様
入力ユニット	絶縁型高分解能フルマルチ入力ユニット 形 CJ1W-PH41U
入力種別	すべて K 型熱電対
出力ユニット	トランジスタ出力ユニット 形 CJ1W-OD212
目標温度	100℃
上限温度	200℃
下限温度	0℃
上下限警報のヒステリシス	5℃
上限偏差温度	50℃
下限偏差温度	50℃
上下限偏差警報のヒステリシス	3℃
PID 演算のサンプリング周期	100ms
出力制御周期	1s

構成・設定

入力ユニット形 CJ1W-PH41U の設定は、以下のとおりとします。

項目名	設定値
Input1:Input signal type	K(1)
Input2:Input signal type	K(1)
Input3:Input signal type	K(1)
Input4:Input signal type	K(1)

I/O マップ設定は、以下のとおりとします。

ユニット	ポート	説明	変数
形 CJ1W-PH41U	Ch1_AIInPV	入力 No.1 測定値（INT 型）	AI1
	Ch2_AIInPV	入力 No.2 測定値（INT 型）	AI2
	Ch3_AIInPV	入力 No.3 測定値（INT 型）	AI3
	Ch4_AIInPV	入力 No.4 測定値（INT 型）	AI4
形 CJ1W-OD212	Ch1_Out00	出力 CH1 接点 00	DO1
	Ch1_Out01	出力 CH1 接点 01	DO2
	Ch1_Out02	出力 CH1 接点 02	DO3
	Ch1_Out03	出力 CH1 接点 03	DO4

4 点の温度制御の、入力と出力の対応は、以下のとおりとします。

入力	出力
AI1	DO1
AI2	DO2
AI3	DO3
AI4	DO4

プログラムを割り付けるタスクの周期は、1ms とします。

● **構成図**

TimeProportionalOut 命令の「サンプルプログラム (P.2-738)」を参照してください。

処理内容

以下の処理を 4 点分行います。

1 **測定温度を取得します。**

2 **LimitAlarm_REAL 命令を使って、測定温度に対して上下限警報を出力します。**

3 **LimitAlarm_REAL 命令で異常が発生した場合、または上下限警報を出力した場合は、安全対策のための出力をします。**

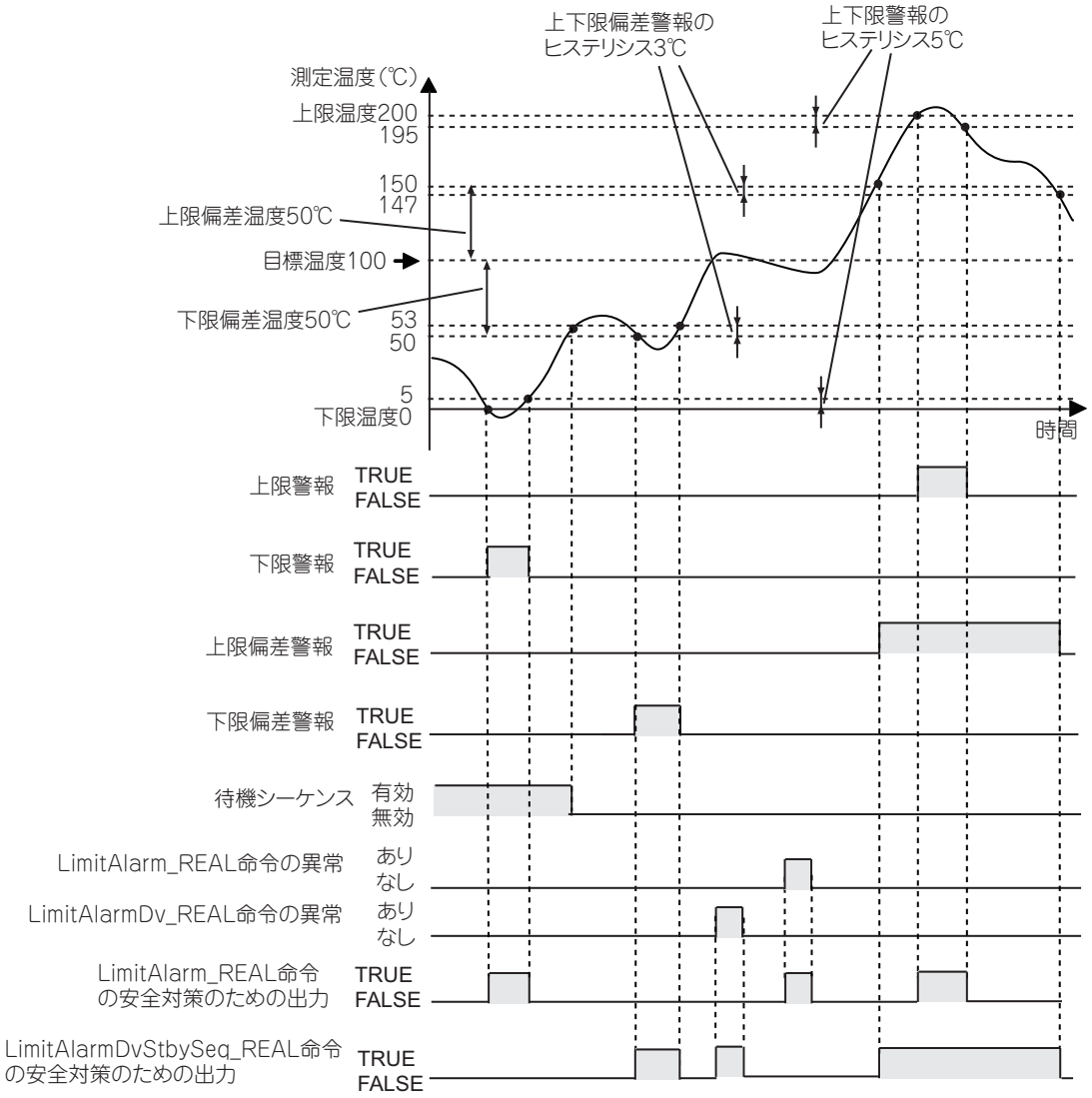
4 **LimitAlarmDvStbySeq_REAL 命令を使って、目標温度からの測定温度の偏差に対して待機シーケンス付き上下限偏差警報を出力します。**

5 **LimitAlarmDvStbySeq_REAL 命令で異常が発生した場合、または上下限偏差警報を出力した場合は、安全対策のための出力をします。**

6 **PIDAT 命令を使って温度制御します。**

7 **TimeProportionalOut 命令を使って、操作量を時分割比例でヒータに出力します。**

● 上下限警報、待機シーケンス付き上下限偏差警報の動作



グローバル変数の定義

● グローバル変数

名称	データ型	割付先 *1	コメント
AI1	INT	IOBus://rack#0/slot#0/Ch1_AIInPV	入力 No.1 測定値（INT 型）
AI2	INT	IOBus://rack#0/slot#0/Ch2_AIInPV	入力 No.2 測定値（INT 型）
AI3	INT	IOBus://rack#0/slot#0/Ch3_AIInPV	入力 No.3 測定値（INT 型）
AI4	INT	IOBus://rack#0/slot#0/Ch4_AIInPV	入力 No.4 測定値（INT 型）
DO1	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00	出力 CH1 接点 00
DO2	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out01	出力 CH1 接点 01
DO3	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out02	出力 CH1 接点 02
DO4	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out03	出力 CH1 接点 03

*1. 形 CJ1W-PH41U をラック番号 0 のスロット番号 0、形 CJ1W-OD212 をラック番号 0 のスロット番号 1 に装着した場合の割付先です。

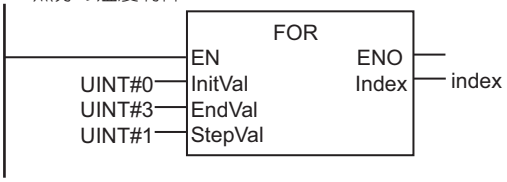
（注）I/O マップ設定により、各ユニットのポートに割り付けたグローバル変数が自動的に生成されます。

LD

名称	データ型	初期値	保持	コメント
index	UINT	0	☐	ループインデックス
LimitAlarm_ON	BOOL	True	☐	上下限警報実行
LimitAlarmDvStbySeq_ON	BOOL	True	☐	待機シーケンス付き上下限偏差警報実行
TimeProportionalOut_ON	BOOL	True	☐	時分割比例出力実行
AI	INT	0	☐	現在値
PV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	測定値
SP	ARRAY[0..3] OF REAL	[4(100)]	☐	目標温度
DOut_TPO	BOOL	False	☐	時分割比例出力
HighVal	ARRAY[0..3] OF REAL	[4(200)]	☐	上下限警報の上限設定値
LowVal	ARRAY[0..3] OF REAL	[4(0.0)]	☐	上下限警報の下限設定値
Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]	☐	上下限警報のヒステリシス
Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限警報出力
HighAlm	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限警報
LowAlm	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限警報
Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarm_REAL 命令の異常
Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限警報の安全対策のための出力
DvHighVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の上限偏差設定値
DvLowVal	ARRAY[0..3] OF REAL	[4(50)]	☐	上下限偏差警報の下限偏差設定値
Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報出力
HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限偏差警報
LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限偏差警報
StbySeqFlag	ARRAY[0..3] OF BOOL	[4(False)]	☐	待機シーケンス有効フラグ
Error_LimitAlarmDvStbySeq	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarmDvStbySeq_REAL 命令の異常
Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]	☐	上下限偏差警報のヒステリシス
Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報の安全対策のための出力

名称	データ型	初期値	保持	コメント
Run	ARRAY[0..3] OF BOOL	[4(False)]	☐	実行条件
ManCtl	ARRAY[0..3] OF BOOL	[4(False)]	☐	マニュアル/オート切替
StartAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行条件
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)	☐	運転中設定パラメータ
InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=-10.0,RngUpLmt:=1000.0, DirOpr:=False)	☐	初期設定パラメータ
PB	ARRAY[0..3] OF REAL	[4(10)]	☑	比例帯
TI	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	積分時間
TD	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	微分時間
ManMV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	マニュアル操作量
ATDone	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング正常終了
ATBusy	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行中
Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	PIDAT 命令の異常
ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]	☐	PIDAT 命令の異常 ID
MV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	操作量
CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]	☐	制御周期
MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]	☐	最小パルス幅
Delay	ARRAY[0..3] OF REAL	[4(0.0)]	☐	オンディレー時間
Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]	☐	TimeProportionalOut 命令の異常
LimitAlarm_REAL_instance	LimitAlarm_REAL		☐	
LimitAlarmDvStbySeq_REAL_instance	LimitAlarmDvStbySeq_REAL		☐	
PIDAT_instance	PIDAT		☐	
TimeProportionalOut_instance	TimeProportionalOut		☐	

4点分の温度制御

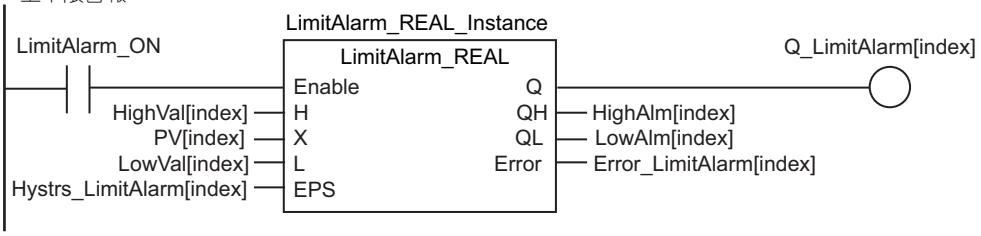


現在値取得

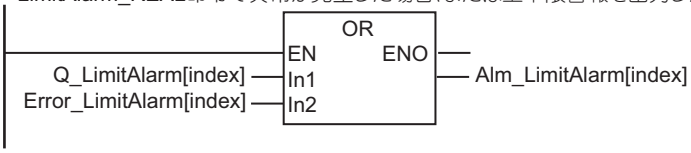
インラインST

注. インラインST の内容については後述のインラインST(1)を参照してください

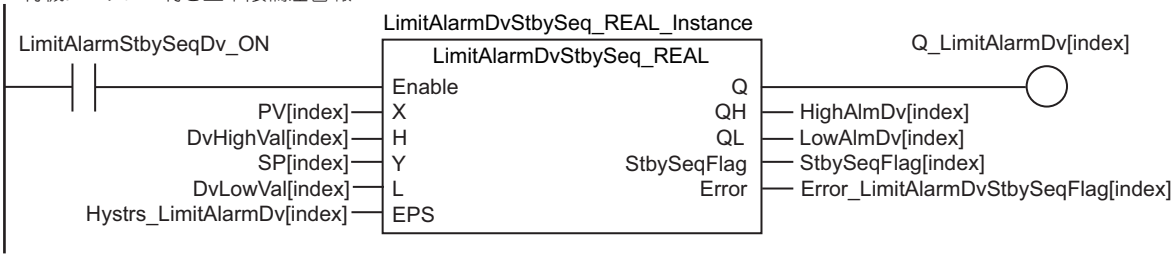
上下限警報



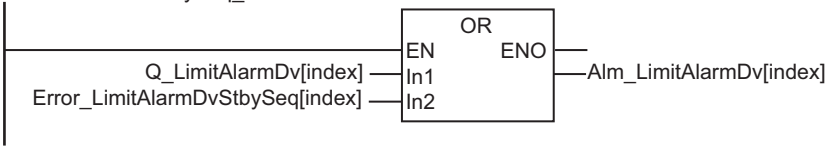
LimitAlarm_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力

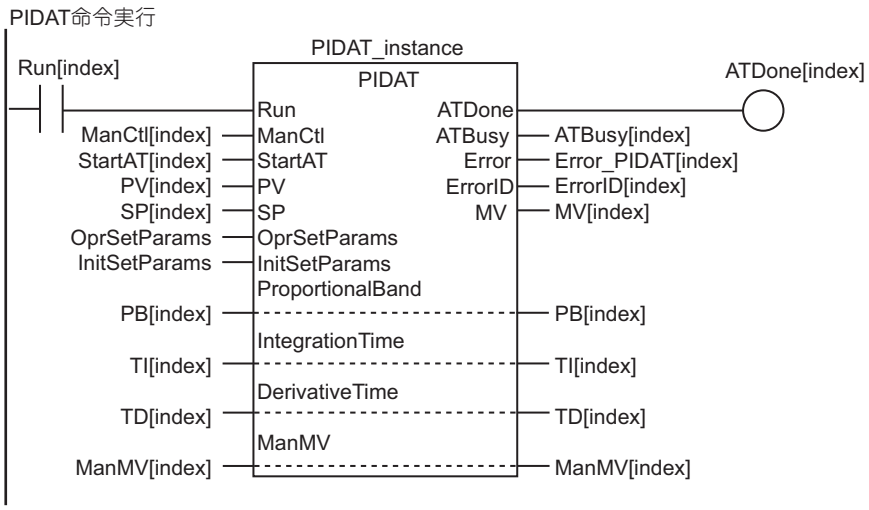


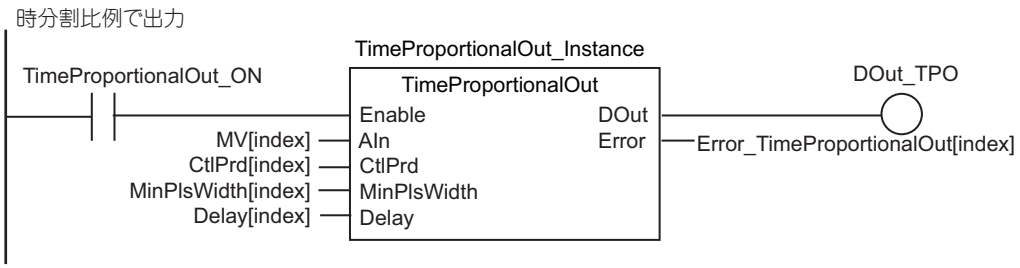
待機シーケンス付き上下限偏差警報

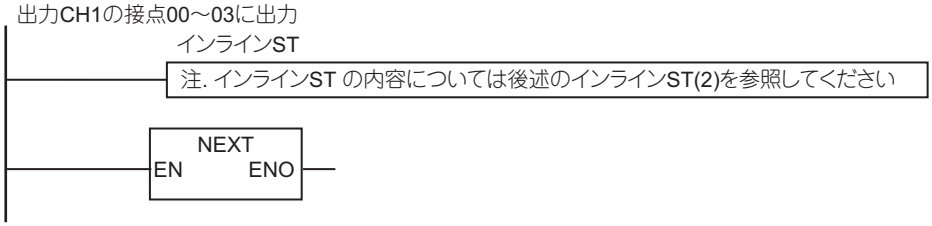


LimitAlarmDvStbySeq_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力









● インライン ST (1)の内容

```
//入力 1～4 の値を取得
CASE index OF
INT#0:
  AI:=AI1;
INT#1:
  AI:=AI2;
INT#2:
  AI:=AI3;
ELSE
  AI:=AI4;
END_CASE;

//現在値 AI を実数に変換
PV[index]:=INT_TO_REAL(AI)/REAL#10.0;  // 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算
```

● **インライン ST (2)の内容**

```
//出力 CH1 の接点 00～03 に出力
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

名称	データ型	初期値	保持	コメント
index	UINT	0	□	ループインデックス
LimitAlarm_ON	BOOL	True	□	上下限警報実行
LimitAlarmDvStbySeq_ON	BOOL	True	□	待機シーケンス付き上下限偏差警報実行
TimeProportionalOut_ON	BOOL	True	□	時分割比例出力実行
AI	INT	0	□	現在値
PV	ARRAY[0..3] OF REAL	0.0	□	測定値
SP	ARRAY[0..3] OF REAL	[4(100)]	□	目標温度
DOut_TPO	BOOL	False	□	時分割比例出力
HighVal	ARRAY[0..3] OF REAL	[4(200)]	□	上下限警報の上限設定値
LowVal	ARRAY[0..3] OF REAL	[4(0.0)]	□	上下限警報の下限設定値
Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]	□	上下限警報のヒステリシス
Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	上下限警報出力
HighAlm	ARRAY[0..3] OF BOOL	[4(False)]	□	上限警報
LowAlm	ARRAY[0..3] OF BOOL	[4(False)]	□	下限警報
Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	LimitAlarm_REAL 命令の異常
Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]	□	上下限警報の安全対策のための出力
DvHighVal	ARRAY[0..3] OF REAL	[4(50)]	□	上下限偏差警報の上限偏差設定値
DvLowVal	ARRAY[0..3] OF REAL	[4(50)]	□	上下限偏差警報の下限偏差設定値

名称	データ型	初期値	保持	コメント
Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報出力
HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上限偏差警報
LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	下限偏差警報
StbySeqFlag	ARRAY[0..3] OF BOOL	[4(False)]	☐	待機シーケンス有効フラグ
Error_LimitAlarmDvStbySeq	ARRAY[0..3] OF BOOL	[4(False)]	☐	LimitAlarmDvStbySeq_REAL 命令の異常
Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]	☐	上下限偏差警報のヒステリシス
Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]	☐	上下限偏差警報の安全対策のための出力
Run	ARRAY[0..3] OF BOOL	[4(False)]	☐	実行条件
ManCtl	ARRAY[0..3] OF BOOL	[4(False)]	☐	マニュアル/オート切替
StartAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行条件
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)	☐	運転中設定パラメータ
InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:= -10.0,RngUpLmt:=1000.0, DirOpr:=False)	☐	初期設定パラメータ
PB	ARRAY[0..3] OF REAL	[4(10)]	☑	比例帯
TI	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	積分時間
TD	ARRAY[0..3] OF TIME	[4(T#0s)]	☑	微分時間
ManMV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	マニュアル操作量
ATDone	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング正常終了
ATBusy	ARRAY[0..3] OF BOOL	[4(False)]	☐	オートチューニング実行中
Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]	☐	PIDAT 命令の異常
ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]	☐	PIDAT 命令の異常 ID
MV	ARRAY[0..3] OF REAL	[4(0.0)]	☐	操作量
CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]	☐	制御周期
MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]	☐	最小パルス幅

名称	データ型	初期値	保持	コメント
Delay	ARRAY[0..3] OF REAL	[4(0.0)]	□	オンディレー時間
Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]	□	TimeProportionalOut 命令の異常
LimitAlarm_REAL_instance	LimitAlarm_REAL		□	
LimitAlarmDvStbySeq_REAL_instance	LimitAlarmDvStbySeq_REAL		□	
PIDAT_instance	PIDAT		□	
TimeProportionalOut_instance	TimeProportionalOut		□	

```
//4 点分の温度制御
FOR index:=UINT#0 TO UINT#3 BY UINT#1 DO

  //入力 1～4 の値を取得
  CASE index OF
  INT#0:
    AI:=AI1;
  INT#1:
    AI:=AI2;
  INT#2:
    AI:=AI3;
  ELSE
    AI:=AI4;
  END_CASE;

  //現在値 AI を実数に変換
  PV[index]:=INT_TO_REAL(AI)/REAL#10.0; // 形 CJ1W-PH41U の出力は測定温度の 10 倍なので、10.0 で除算

  //上下限警報
  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]);

  //LimitAlarm_REAL 命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力
  Alm_LimitAlarm[index]:=Q_LimitAlarm[index] OR Error_LimitAlarm[index];

  //待機シーケンス付き上下限偏差警報
  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]);

//LimitAlarmDvStbySeq_REAL 命令で異常が発生した場合、
//または上下限警報を出力した場合の安全対策のための出力
Alm_LimitAlarmDv[index]:=Q_LimitAlarmDv[index] OR Error_LimitAlarmDvStbySeq[index];

// PIDAT 命令実行
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]);

// 時分割比例で出力
TimeProportionalOut_instance(
  Enable :=TimeProportionalOut_ON,
  AIn :=MV[index],
  CtlPrd :=CtlPrd[index],
  MinPlsWidth :=MinPlsWidth[index],
  Delay :=Delay[index],
  DOut =>DOut_TPO,
  Error =>Error_TimeProportionalOut[index]);

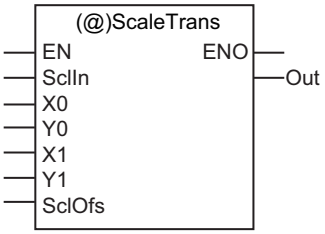
//出力 Ch1 の接点 00～03 に出力
CASE index OF
INT#0:
  DO1:=DOut_TPO;
INT#1:
  DO2:=DOut_TPO;
INT#2:
```

```
      DO3:=DOut_TPO;
    ELSE
      DO4:=DOut_TPO;
    END_CASE;

END_FOR;
```

ScaleTrans

入力値を入力レンジから出力レンジに変換します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ScaleTrans	スケール変換	FUN		Out :=ScaleTrans(ScIIn, X0, Y0, X1, Y1, SclOfs);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ScIIn	入力値	入力	変換の対象となる値	データ型に従う	−	*1
X0	入力レンジ下限値		入力レンジの下限値			0
Y0	出力レンジ下限値		出力レンジの下限値			
X1	入力レンジ上限値		入力レンジの上限値			
Y1	出力レンジ上限値		出力レンジの上限値			
SclOfs	オフセット値		出力値に加えるオフセット値			
Out	出力値	出力	スケール変換後の値		−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

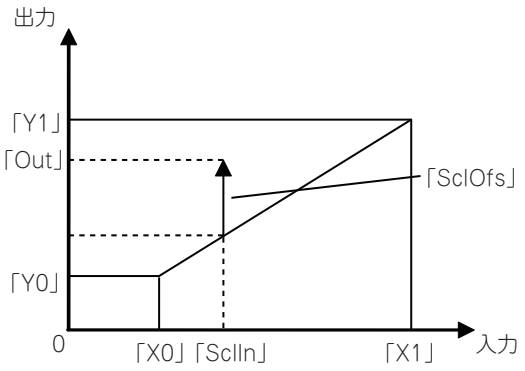
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ScIIn														○	○					
X0														○	○					
X1														○	○					
Y0														○	○					
Y1														○	○					
SclOfs														○	○					
Out														○	○					

機能

入力値「ScIIn」の値を、入力レンジから出力レンジに変換します。
入力レンジは、入力レンジ下限値「X0」と入力レンジ上限値「X1」で指定します。出力レンジは、出力レンジ下限値「Y0」と出力レンジ上限値「Y1」で指定します。
さらに、出力レンジに変換した値にオフセット値「SclOfs」を加算した値を、出力値「Out」として出力します。「SclOfs」は、温度制御の誤差の調整などに使用します。

変換式は、つぎのとおりです。

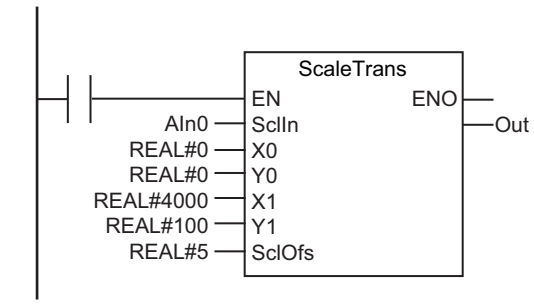
$$「Out」=\frac{「Y1」-「Y0」}{「X1」-「X0」}(「SclIn」-「X0」)+「Y0」+「SclOfs」$$



記述例

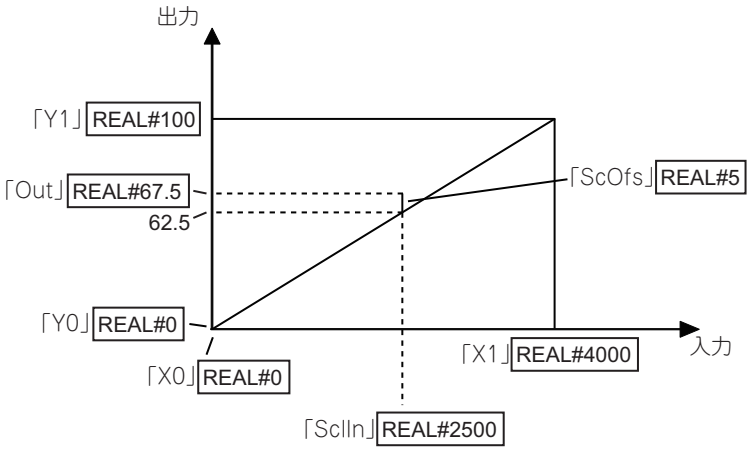
入力レンジが 0～4000、出力レンジが 0～100％で、入力値 2500 を変換する場合の記述例を以下に示します。出力に加えるオフセット値は 5％とします。
「SclIn」=REAL#2500、「X0」=REAL#0、「X1」= REAL#4000、「Y0」=REAL#0、「Y1」=REAL#100、「SclOfs」=REAL#5 とします。
「Out」の値は、REAL#67.5 となります。

LD



ST

```
Out :=ScaleTrans(AIn0,REAL#0,REAL#0,REAL#4000,
                 REAL#100,REAL#5);
```



入力レンジ0～4000、出力レンジ0～100で、入力値2500を変換すると62.5。
それにオフセット値5を加えて、「Out」=REAL#67.5になります。

参考

- 「SclIn」を、PIDAT 命令の「PV」および「SP」の値の範囲にスケール変換する場合は、「Y0」、「Y1」に以下のパラメータを渡してください。

変数	パラメータ
Y0	PIDAT 命令の入力レンジ下限「InitSetParams.RngLowLmt」
Y1	PIDAT 命令の入力レンジ上限「InitSetParams.RngUpLmt」

・「X1」＜「X0」や、「Y1」＜「Y0」となる設定も可能です。

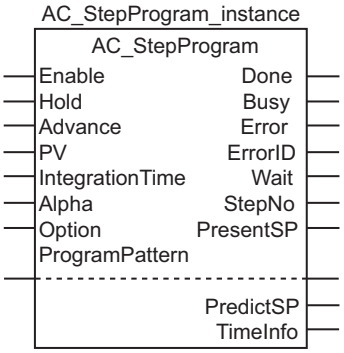
使用上の注意

「SclIn」に整数型のパラメータを渡した場合、以下のようにデータ型変換して入力されます。

「SclIn」に渡すパラメータのデータ型	「SclIn」,「X0」,「X1」,「Y0」,「Y1」,「SclOfs」のデータ型
USINT, UINT, SINT, INT	REAL
UDINT, DINT	LREAL
ULINT, LINT	ビルド時に異常

AC_StepProgram

指定したプログラムパターンに従い、タスク周期ごとに現在目標値と予測目標値を算出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
AC_StepProgram	ステッププログラム	FB		AC_StepProgram_instance(Enable, Hold, Advance, PV, IntegrationTime, Alpha, Option, ProgramPattern, Done, Busy, Error, ErrorID, Wait, StepNo, PresentSP, PredictSP, TimeInfo);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	動作指令	入力	TRUE: 実行 FALSE: 非実行	データ型に従う	−	FALSE
Hold	ホールド		TRUE: ホールド機能を実行 FALSE: ホールド機能を非実行			
Advance	前進		FALSE から TRUE への立ち上がり時に、実行するステップ番号をインクリメント			
PV	測定値（現在値）		測定値（現在値） *1			0
Integration Time	積分時間		積分時間 *2	T#0.0000s ～ T#10000.0000s *3	s	T#0s
Alpha	2 自由度 PID パラメータ α		2 自由度 PID パラメータ α *4	0.00 ～ 1.00	−	0
Option	オプション		オプション *5	−		−
ProgramPattern[] 配列	プログラムパターン	入出力	プログラムパターン	−	−	−
Wait	ウェイト中	出力	TRUE: ウェイト中 FALSE: 非ウェイト中	データ型に従う	−	−
StepNo	現在ステップ番号		現在実行中のステップ番号	0 ～ 255*6		
PresentSP	現在目標値		算出した現在目標値	データ型に従う		
PredictSP	予測目標値		算出した予測目標値			
TimeInfo	時間情報		本命令の進行状況を監視するための時間情報	−		

*1. PIDAT 命令の「PV」と同じ。詳細は、「PV（測定値、現在値）（P.2-781）」を参照してください。
*2. PIDAT 命令の「IntegrationTime」と同じ。詳細は、「IntegrationTime（積分時間）（P.2-781）」を参照してください。
*3. 0.0001s 未満は、切り捨て。

*4. PIDAT 命令の「OprSetParams.Alpha」と同じ。詳細は、「Alpha（2 自由度 PID パラメータ α）（P.2-782)」を参照してください。
*5. 詳細は、「構造体の仕様 (P.2-780)」を参照してください。
*6. ユニットバージョン Ver.1.20 以前は、「0～99」となります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
Hold	○																			
Advance	○																			
PV														○						
Integration Time																○				
Alpha														○						
Option	構造体_sAC_STEP_OPTION　詳細は「構造体の仕様 (P.2-780)」参照																			
ProgramPattern[] 配列 *1 *2 *3	構造体_sAC_STEP_DATA　詳細は「構造体の仕様 (P.2-780)」参照 配列全体を指定																			
Wait	○																			
StepNo						○														
PresentSP														○						
PredictSP														○						
TimeInfo	構造体_sAC_STEP_TIME　詳細は「構造体の仕様 (P.2-780)」参照																			

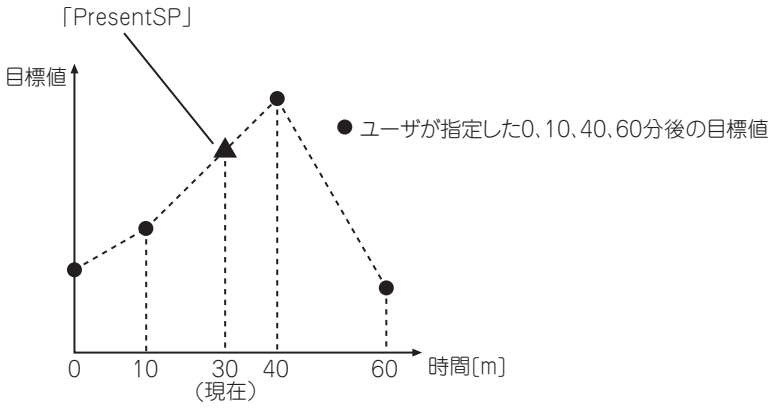
*1. 配列の最大要素数は、ユニットバージョン Ver.1.21 以降、「256」となります。Ver.1.20 以前は、「100」となります。
*2. 1 次元の配列です。2 次元以上の配列を指定した場合は、ビルド時に異常になります。
*3. 先頭の要素番号は 0 です。先頭の要素番号が 0 以外の配列を指定した場合は、ビルド時に異常になります。

機能

PIDAT 命令と連動させて、温度調節器などの操作量制御を行う場合の、現在目標値「PresentSP」と予測目標値「PredictSP」を、タスク周期ごとに算出します。
現在目標値とは、現タスク周期での目標値のことです。
予測目標値とは、現在目標値に 2 自由度 PID 制御の遅れ補正を施した目標値のことです。
予測目標値「PredictSP」を PIDAT 命令の目標値「SP」に渡すことで、PIDAT 命令によるプログラム制御の追従性を改善することができます。

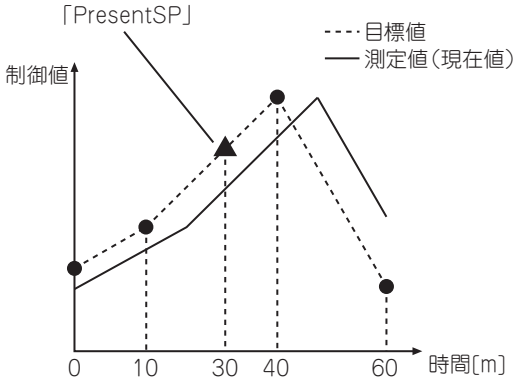
現在目標値「PresentSP」

現在目標値「PresentSP」とは、現タスク周期での目標値のことです。
たとえば、ユーザが、制御開始 0、10、40、60 分後の目標値を、下図のように設定しているとします。現在の時刻は、制御開始から 30 分後であるとします。本命令は、制御開始 10 分後と 40 分後の目標値を線形補間することにより、「PresentSP」を算出します。

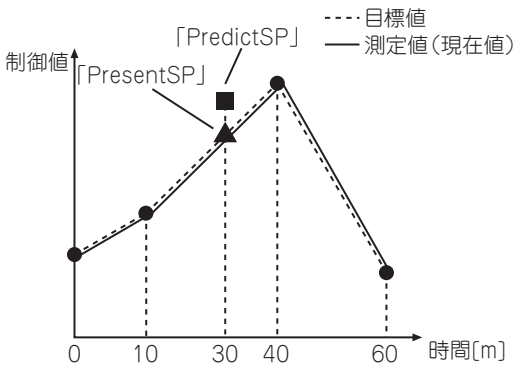


予測目標値「PredictSP」

予測目標値「PredictSP」とは、「PresentSP」に 2 自由度 PID 制御の遅れ補正を施した目標値のことです。
「PresentSP」を補正なしに PIDAT 命令の「SP」に渡すと、PIDAT 命令の「PV」は、目標値と一致しません。その様子を下図に示します。



この遅れを補正するのが、「PredictSP」です。
本命令は、積分時間「IntegrationTime」や 2 自由度 PID パラメータ α「Alpha」の値をもとに、「PredictSP」を算出します。
「PredictSP」を PIDAT 命令の「SP」に渡すことにより、PIDAT 命令のプログラム制御の追従性を改善することができます。



構造体の仕様

オプション「Option」のデータ型は、構造体_sAC_STEP_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sAC_STEP_OPTION	－	－	－
StartAtPV	現在値開始	TRUE: 現在値開始機能実行 FALSE: 現在値開始機能非実行	BOOL	データ型に従う	－	FALSE
StartStepNo	開始ステップ番号	処理を開始するステップ番号	USINT	0 ～ 255 *1	－	0
EndStepNo	終了ステップ番号	処理を終了するステップ番号 *2	USINT			
Reserved	予備	予備	ARRAY[0..31] OF BYTE	データ型に従う		32 個の要素がすべて 0

*1. ユニットバージョン Ver.1.20 以前は、「0～99」となります。
*2. 「0」を設定した場合、ProgramPattern[]の最大要素番号を終了ステップ番号とみなします。

プログラムパターン ProgramPattern[]の要素のデータ型は、構造体_sAC_STEP_DATA です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ProgramPattern	プログラムパターン	プログラムパターン	_sAC_STEP_DATA	－	－	－
ReachSP	到達目標値	当該ステップの到達目標値	REAL	データ型に従う		0
TimeWidth	時間幅	当該ステップの時間幅 *1	TIME		s	T#0s
WaitWidth	ウェイト判定幅	当該ステップのウェイト判定幅 *2	REAL		－	0
WaitTimeLimit	ウェイト時間上限	当該ステップのウェイト時間上限 *1 *3	TIME		s	T#0s

*1. 分解能は、1 タスク周期です。
*2. 0 以下の値を設定した場合、0 とみなします。
*3. 0 以下の値を設定した場合、T#0s とみなします。

時間情報「TimeInfo」のデータ型は、構造体_sAC_STEP_TIME です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
TimeInfo	時間情報	時間情報	_sAC_STEP_TIME	–	–	–
ProgramTime	プログラム時間	ステップ 0 から「EndStepNo」までの「TimeWidth」の合計	TIME	0 または正値	s	T#0s
ElapseTime	経過時間	本命令の実行開始から現在までの時間 *1	TIME			
ProgressTime	進捗時間	本命令の実行開始から現在までの時間 *2	TIME			
LeftTime	残時間	現在からすべての処理終了までの時間 *2	TIME			
StepProgressTime	ステップ内進捗時間	実行中のステップの開始から現在までの時間 *2	TIME			
StepLeftTime	ステップ内残時間	現在から、実行中のステップの終了までの時間 *2	TIME			

*1. ウェイト時間を含みます。ホールド時間は含みません。
*2. ウェイト時間やホールド時間を含みません。

各変数の意味

本命令の各変数の意味を、詳しく説明します。

● Enable（動作指令）

本命令の実行条件です。
「Enable」の値を FALSE から TRUE に変更すると、本命令が実行されます。「Enable」の値を FALSE にすると、本命令の実行が停止されます。

● Hold（ホールド）

ホールド機能を実行するためのフラグです。
「Hold」の値を TRUE にすると、ホールド機能が実行されます。
ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

● Advance（前進）

本命令実行中に値が FALSE から TRUE に変化すると、処理がつぎのステップに移ります。
前進機能の詳細は、「前進機能 (P.2-788)」を参照してください。

● PV（測定値、現在値）

制御対象の測定値です。PIDAT 命令の「PV」と同じ意味です。

● IntegrationTime（積分時間）

PIDAT 命令の「IntegrationTime」と同じ意味です。
PIDAT 命令または PIDAT_HeatCool 命令の、「IntegrationTime」の値またはその変数を入力してください。

- **Alpha（2 自由度 PID パラメータ α）**

  PIDAT 命令の「OprSetParams.Alpha」と同じ意味です。
  PIDAT 命令または PIDAT_HeatCool 命令の、「OprSetParams.Alpha」の値またはその変数を入力してください。

- **StartAtPV（現在値開始）**

  現在値開始機能実行フラグです。
  「StartAtPV」の値を TRUE にすると、現在値開始機能が実行されます。
  現在値開始機能の詳細は、「現在値開始機能 (P.2-787)」を参照してください。

- **StartStepNo（開始ステップ番号）、EndStepNo（終了ステップ番号）**

  プログラムパターンのステップのうち、処理を開始するステップの番号と終了するステップの番号のことです。
  「EndStepNo」=0 の場合は、ProgramPattern[]の最大要素番号を終了ステップ番号とみなします。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。

- **ReachSP（到達目標値）**

  プログラムパターン内の、各ステップの終了時に到達すべき目標値です。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。

- **TimeWidth（時間幅）**

  プログラムパターン内の、各ステップの時間幅です。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。

- **WaitWidth（ウェイト判定幅）**

  プログラムパターン内の、各ステップでウェイト機能を実行するかどうかを判定するためのしきい値です。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を参照してください。

- **WaitTimeLimit（ウェイト時間上限）**

  プログラムパターン内の、各ステップでウェイト機能を実行した場合の、ウェイト時間の上限値です。
  「WaitTimeLimit」の値が T#0s の場合は、ウェイト時間の上限値が無限大の意味になります。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を参照してください。

- **Wait（ウェイト中）**

  ウェイト機能実行中かどうかを示すフラグです。
  「Wait」の値が TRUE なら、ウェイト機能実行中です。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を参照してください。

- **StepNo（現在ステップ番号）**

  現在実行中のステップ番号です。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。

- **PresentSP（現在目標値）**

  現在目標値です。
  現在目標値の詳細は、「現在目標値「PresentSP」 (P.2-778)」を参照してください。

- **PredictSP（予測目標値）**

  予測目標値です。
  予測目標値の詳細は、「予測目標値「PredictSP」 (P.2-779)」を参照してください。

- **ProgramTime（プログラム時間）**

  プログラムパターンの、ステップ 0 から「EndStepNo」までの「TimeWidth」の合計値です。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。

- **ElapseTime（経過時間）**

  本命令の実行開始から現在までの時間です。ただし、ウェイト時間は含みますが、ホールド時間は含みません。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を、ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

- **ProgressTime（進捗時間）**

  本命令の実行開始から現在までの時間です。ただし、ウェイト時間やホールド時間を含みません。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を、ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

- **LeftTime（残時間）**

  現在から、すべての処理終了までの時間です。ただし、ウェイト時間やホールド時間は含みません。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を、ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

- **StepProgressTime（ステップ内進捗時間）**

  プログラムパターン内の、実行中のステップの開始から現在までの時間です。ただし、ウェイト時間やホールド時間を含みません。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を、ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

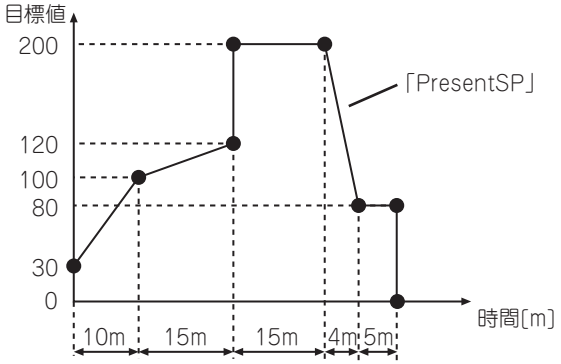
- **StepLeftTime（ステップ内残時間）**

  プログラムパターン内の、現在から、実行中のステップの処理終了までの時間です。ただし、ウェイト時間やホールド時間は含みません。
  プログラムパターンやステップの詳細は、「プログラムパターン (P.2-783)」を参照してください。
  ウェイト機能の詳細は、「ウェイト機能 (P.2-785)」を、ホールド機能の詳細は、「ホールド機能 (P.2-786)」を参照してください。

プログラムパターン

プログラムパターンとは、本命令実行開始から終了までの処理を複数のステップに分割し、ステップごとの到達目標値と時間幅を、時系列で並べたものです。
プログラムパターンは、データ型_sAC_STEP_DATA を要素とする配列 ProgramPattern[]で表現します。ProgramPattern[]の各要素が、各ステップに対応します。
以下に、プログラムパターンの例を示します。ProgramPattern[]の各要素の、「ReachSP」と「TimeWidth」の値が下表の場合、本命令実行開始からの時間と目標値との関係は、下図のようになります。

	ProgramPattern[]の要素番号							
	0	1	2	3	4	5	6	7
ステップ番号	0	1	2	3	4	5	6	7
「ReachSP」の値	30	100	120	200	200	80	80	0
「TimeWidth」の値	T#0s	T#10m	T#15m	T#0s	T#15m	T#4m	T#5m	T#0s



各ステップの目標値を線形補間し、各時点の「PresentSP」の値が算出されます。

上図の実線で示したものが、「PresentSP」です。タスク周期ごとに、その時点での「PresentSP」の値が出力されます。

●「TimeWidth」の値とステップの時間幅の関係

「TimeWidth」の値とステップの時間幅の関係は、以下のとおりです。

「TimeWidth」の値	ステップ番号	ステップの時間幅
T#0s	0	T#0s とみなします。
	0 以外	1 タスク周期とみなします。
正値	—	「TimeWidth」の値が、ステップの時間幅になります。
負値	—	1 タスク周期とみなします。

● ステップの時間幅が 1 タスク周期未満の場合の動作

ステップの時間幅の分解能は、1 タスク周期です。ステップの時間幅が 1 タスク周期未満の場合の動作は、以下のとおりです。

ステップ番号	ステップの時間幅	動作
0	T#0s	ステップ 0 の「ReachSP」の値が、「PresentSP」の初期値になります。実際の処理は、ステップ 1 から始まります。
	T#0s 以外	当該ステップの処理を 1 タスク周期だけ行った後、つぎのステップに処理が移ります。
0 以外	—	

開始ステップ番号「StartStepNo」と終了ステップ番号「EndStepNo」

プログラムパターン内の任意のステップを、処理の開始ステップおよび終了ステップに設定することができます。
開始ステップ番号を「StartStepNo」に、終了ステップ番号を「EndStepNo」に設定します。
たとえば、「StartStepNo」=3、「EndStepNo」=6 として本命令を実行すると、処理がステップ 3 から始まり、ステップ 6 で終了します。

● 命令実行中の「StartStepNo」、「EndStepNo」の値の変更

命令実行中に、「StartStepNo」および「EndStepNo」の値を変更することが可能です。値を変更すると、以下のように動作します。

変数	変更するステップ番号	動作
StartStepNo	―	変更後の「StartStepNo」の先頭から処理を開始します。
EndStepNo	実行中のステップ番号以上の値への変更	変更後の「EndStepNo」が終了した時点で、処理が終了します。
	実行中のステップ番号未満の値への変更	変更した時点で、処理が終了します。 「Done」の値が TRUE になります。

ウェイト機能

制御対象の遅れにより、「PV」の値が、実行中のステップの「TimeWidth」内に「ReachSP」の値に到達しない場合があります。
このような場合に、「TimeWidth」の時間幅を超えて、実行中のステップの処理を保持する機能を、ウェイト機能と呼びます。
ウェイト機能に関係する変数は、ProgramPattern[]内の、ウェイト判定値「WaitWidth」とウェイト時間上限「WaitTimeLimit」、およびウェイト中「Wait」です。

● ウェイト機能実行の判定

実行中のステップの終了時間に、「ReachSP」と「PV」の差が「WaitWidth」を超えていたら、ウェイト機能が実行されます。

● ウェイト機能終了のタイミング

ウェイト機能実行開始から「WaitTimeLimit」以内の時間に、「ReachSP」と「PV」の差が「WaitWidth」以下になったら、その時点でウェイト機能実行を終了し、つぎのステップに処理が移ります。
ウェイト機能実行開始から「WaitTimeLimit」以内の時間に、「ReachSP」と「PV」の差が「WaitWidth」以下にならなかったら、「WaitTimeLimit」の時間経過後に、ウェイト機能実行を終了し、つぎのステップに処理が移ります。ただし、「WaitTimeLimit」の値が T#0s の場合は、ウェイト時間上限が無限大になります。したがって、「ReachSP」と「PV」の差が「WaitWidth」以下になるまで、時間制限なしにウェイト機能を実行し続けます。

● ウェイト機能実行の監視

ウェイト機能実行中かどうかは、ウェイト中「Wait」の値を監視することでわかります。
ウェイト機能実行中は、「Wait」の値が TRUE になります。
ウェイト機能実行が終了したら、「Wait」の値が FALSE になります。

● ウェイト機能実行中の計時

ウェイト機能実行中の、時間関係の各変数の動作は、以下のようになります。

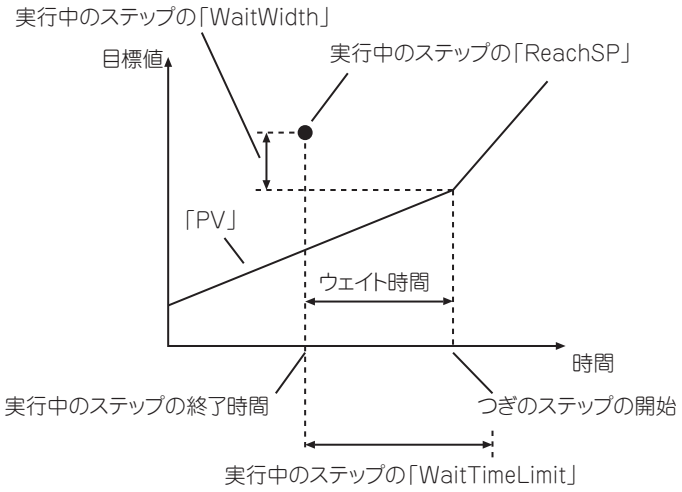
変数	動作
ElapseTime	計時を継続します。
ProgressTime	計時を停止し、ウェイト機能実行開始時点の値を保持します。ウェイト機能実行が終了したら、その値から計時を再開します。
LeftTime	
StepProgressTime	実行中のステップの「TimeWidth」の値になり、その値を保持します。
StepLeftTime	0 になり、その値を保持します。

● ウェイト機能実行中の「PresentSP」と「PredictSP」

ウェイト機能実行中の「PresentSP」と「PredictSP」は、どちらも「ReachSP」の値を保持します。

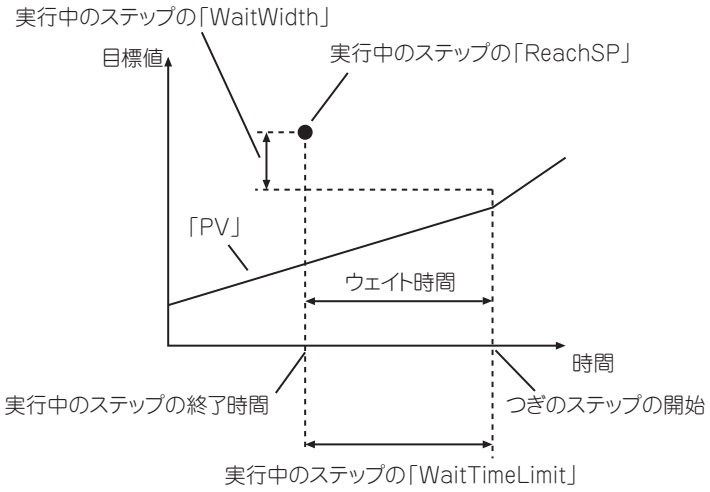
● ウェイト機能の例

「WaitTimeLimit」以内の時間に、「ReachSP」と「PV」の差が「WaitWidth」以下になった場合の、「PV」のグラフを示します。
実行中のステップの終了時間が来ても、「ReachSP」と「PV」の差が「WaitWidth」を超えているので、ウェイト機能を実行します。
「ReachSP」と「PV」の差が「WaitWidth」以下になった時点で、つぎのステップの処理を開始します。



つぎに、「WaitTimeLimit」以内の時間に、「ReachSP」と「PV」の差が「WaitWidth」以下にならなかった場合の、「PV」のグラフを示します。

「WaitTimeLimit」の時間経過後に、つぎのステップの処理を開始します。



ホールド機能

ホールド機能は、ホールド「Hold」の値を TRUE にすることにより、実行中のステップの処理を強制的に保持させる機能です。ホールド機能実行中は、時間関係のすべての変数の計時が停止します。

「Hold」の値を FALSE にすると、時間関係の変数の計時が再開されます。

● ホールド機能実行中の計時

ホールド機能実行中の、時間関係の各変数の動作は、以下のようになります。

変数	動作
ElapseTime	計時を停止し、ホールド機能実行開始時点の値を保持します。 ホールド機能実行が終了したら、その値から計時を再開します。
ProgressTime	
LeftTime	
StepProgressTime	
StepLeftTime	

● ホールド機能実行中の「PresentSP」と「PredictSP」

ホールド機能実行中の「PresentSP」は、ホールド機能実行開始時点の値を保持します。
ホールド機能実行中の「PredictSP」は、「PresentSP」と同じ値になります。

● ウェイト機能実行中のホールド機能実行

ウェイト機能実行中にホールド機能を実行すると、ウェイト機能の実行がいったん終了します。したがって、「Wait」の値は FALSE になります。その後、ホールド機能実行を終了した時点で、あらためてウェイト機能実行の判定が行われます。

現在値開始機能

現在値開始機能は、「PV」の現在値と「PresentSP」の値が一致する時点から、処理を開始する機能です。
現在値開始「StartAtPV」の値を TRUE にした状態で「Enable」の値を TRUE にすると、現在値開始機能が実行されます。
現在値開始機能が実行されると、以下の手順で処理が行われます。

1 **「PV」の現在値が、取得されます。**

2 **ステップ 0 から最終ステップに向かって、「PV」の現在値と「PresentSP」の値が最初に一致する時刻が探索されます。**
ただし、「PresentSP」の値がステップ 0 の先頭から上昇している場合は、「PresentSP」の値が下降する直前までしか探索しません。同様に、「PresentSP」の値がステップ 0 の先頭から下降している場合は、「PresentSP」の値が上昇する直前までしか探索しません。

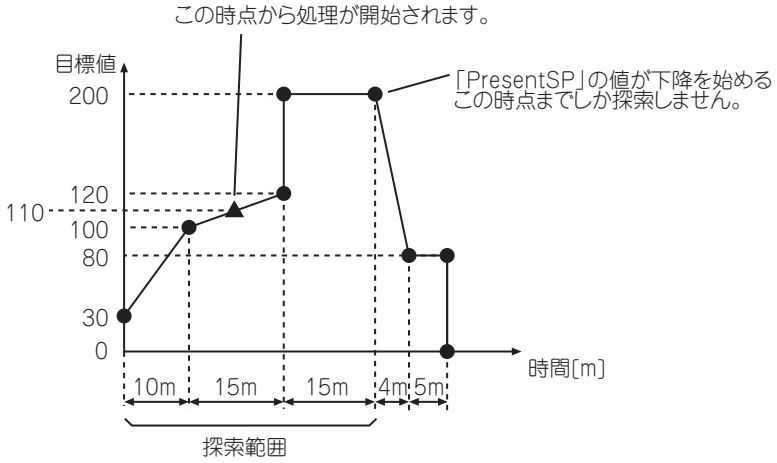
3 **上で探索した時点から、処理が開始されます。**

探索範囲の中に、「PV」の現在値と「PresentSP」の値が一致する時刻が存在しない場合は、ステップ 0 から処理が開始されます。

以下に、現在値開始機能実行の例を示します。ProgramPattern[]の内容は下表のとおりだとします。

	ProgramPattern[]の要素番号							
	0	1	2	3	4	5	6	7
ステップ番号	0	1	2	3	4	5	6	7
「ReachSP」の値	30	100	120	200	200	80	80	0
「TimeWidth」の値	T#0s	T#10m	T#15m	T#0s	T#15m	T#4m	T#5m	T#0s

「PresentSP」の値は、ステップ 0 の値から上昇しています。したがって、「PresentSP」の値が下降を始める、処理開始後 40m の時点までしか探索しません。
本命令実行開始時の「PV」の値が 110 であったとします。その場合、「PresentSP」=110 になる、下図で示した時点から処理が開始されます。



● 現在値開始機能実行時の計時

現在値開始機能実行時の、時間関係の各変数の動作は、以下のようになります。

変数	動作
ElapseTime	0 になります。
ProgressTime	ステップ 0 から、探索した時点までの時間になります。
LeftTime	現在から、「EndStepNo」の終了までの時間になります。
StepProgressTime	実行中のステップの先頭から、探索した時点までの時間になります。
StepLeftTime	現在から、実行中のステップの終了までの時間になります。

● 命令実行中の「StartAtPV」の値変更

本命令の実行中に「StartAtPV」の値を変更しても、無視されます。

前進機能

本命令を実行中に、「Advance」の値を FALSE から TRUE に変更すると、処理がつぎのステップの先頭に移ります。

● 前進機能実行時の計時

前進機能実行時の、時間関係の各変数の動作は、以下のようになります。

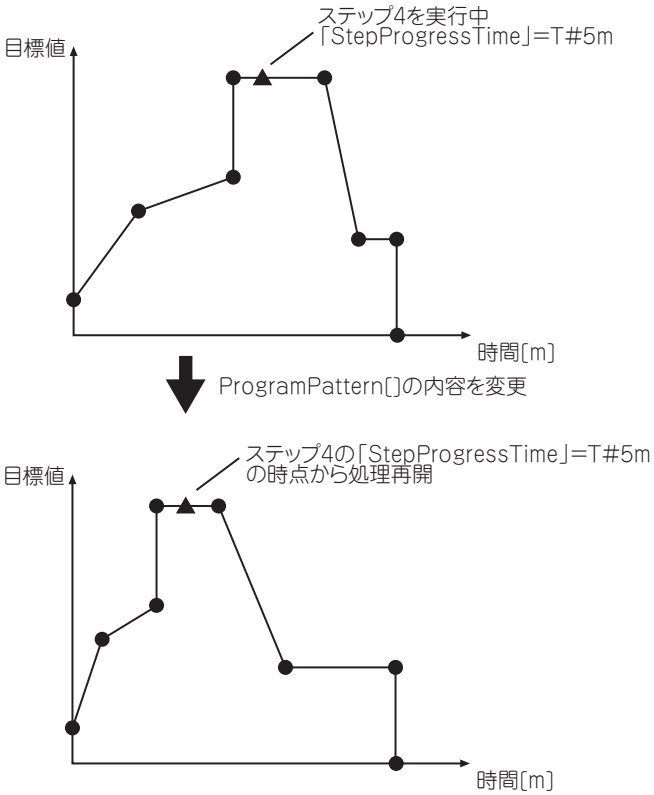
変数	動作
ElapseTime	計時を継続します。
ProgressTime	ステップ 0 から実行中のステップまでの「TimeWidth」の合計になります。
LeftTime	つぎのステップから「EndStepNo」の終了までの時間になります。
StepProgressTime	処理がつぎのステップの先頭に移るため、0 になります。
StepLeftTime	つぎのステップの「TimeWidth」の値になります。

●「StartStepNo」の値変更と前進機能の併用

「StartStepNo」の値を変更すると同時に「Advance」の値を FALSE から TRUE に変更した場合は、「StartStepNo」の値の変更が優先されます。したがって、処理は「StartStepNo」の先頭に移ります。

命令実行中のプログラムパターンの変更

命令実行中に、ProgramPattern[]の内容を変更することが可能です。
ProgramPattern[]の内容を変更すると、「PresentSP」が算出しなおされます。
処理は、変更前に実行していたステップ番号の「StepProgressTime」の時点から再開されます。
過去のステップの内容を変更してもかまいません。
たとえば、ステップ 4 を実行中に ProgramPattern[]の内容を変更したとします。
変更前の「StepProgressTime」の値は、T#5m だったとします。変更後は、ステップ 4 の「StepProgressTime」=T#5m の時点から処理が再開されます。



変更後の当該ステップの「TimeWidth」の値が、「StepProgressTime」の値より小さい場合は、つぎのステップの先頭から処理が再開されます。

● 命令実行中にプログラムパターンを変更した場合の計時

命令実行中にプログラムパターンを変更した場合の、時間関係の各変数の動作は、以下のようになります。

変数	動作
ProgramTime	変更後の、ステップ 0 から「EndStepNo」までの「TimeWidth」の合計になります。
ElapseTime	計時を継続します。
ProgressTime	変更後の、ステップ 0 から実行中のステップ番号の 1 個前のステップまでの「TimeWidth」の合計に、「StepProgressTime」を加算した値になります。
LeftTime	変更後の、現在から「EndStepNo」の終了までの時間になります。

変数	動作
StepProgressTime	変更前の値から、計時を継続します。
StepLeftTime	変更後の、実行中のステップの現在からステップ終了までの時間になります。

● ウェイト機能実行中のプログラムパターン変更

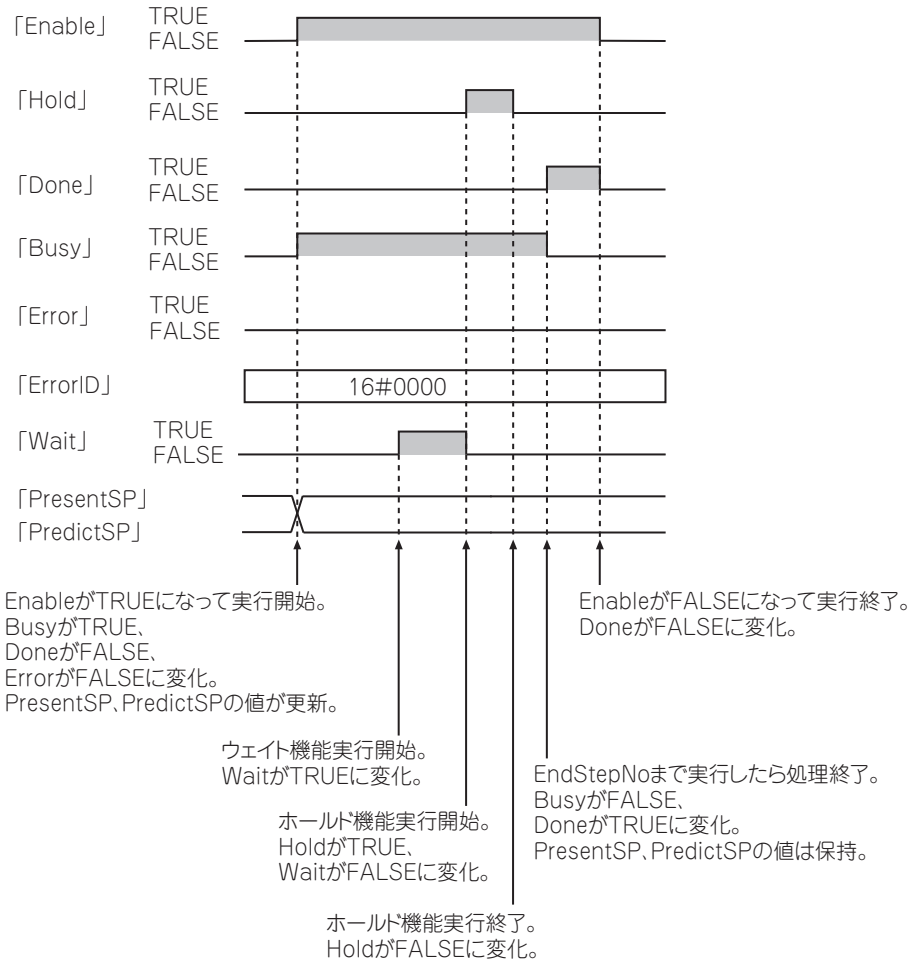
ウェイト機能実行中にプログラムパターンを変更した場合、算出しなおされた「PresentSP」に対して、あらためてウェイト機能実行の判定が行われます。
ただし、「StepProgressTime」の値が、変更後の「WaitTimeLimit」の値より大きい場合は、ただちにウェイト機能実行を終了し、つぎのステップに処理が移ります。

● ホールド機能実行中のプログラムパターン変更

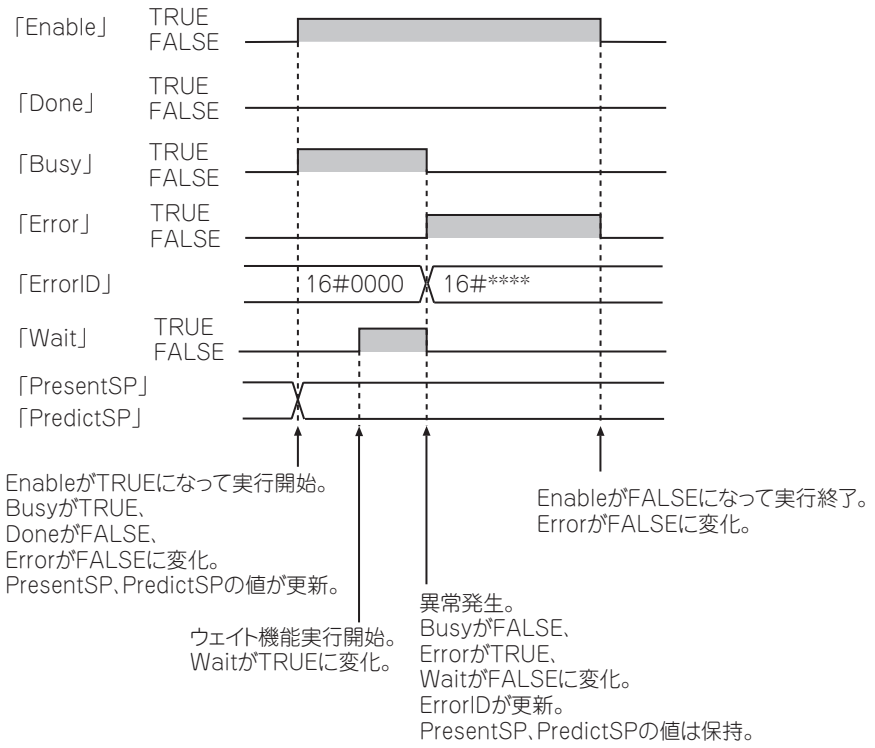
ホールド機能実行中にプログラムパターンを変更した場合、算出しなおされた「PresentSP」に対してホールド機能が継続されます。

タイミングチャート

以下に、正常時のタイミングチャートを示します。



以下に、異常発生時のタイミングチャートを示します。



使用上の注意

つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。

異常の内容	「ErrorID」の値
「IntegrationTime」、「Alpha」、「StartStepNo」、「EndStepNo」のいずれかの値が、有効範囲外。	16#0400
ProgramPattern[]の配列の最終要素番号が 255 *1 を超えている。	16#0416

*1. ユニットバージョン Ver.1.20 以前は、「99」となります。

サンプルプログラム

AC_StepProgram 命令の、ステップごとの最適な PID パラメータを使用して、温度制御を行うサンプルプログラムです。

処理内容

本サンプルプログラムは、以下の 2 つの処理を行います。
- ステップごとの、最適 PID パラメータ算出
- プログラムパターンに従った温度制御

それぞれの処理について説明します。

● ステップごとの、最適 PID パラメータ算出

プログラムパターンに従った温度制御を行う前に、あらかじめ各ステップに対する最適な PID パラメータを算出しておきます。PID パラメータの算出には、PIDAT 命令のオートチューニング機能を使用します。

算出された PID パラメータは、ステップ番号を添え字に持つ構造体型配列 PIDbank[]に格納されます。PIDbank[]の要素の各メンバは、比例帯、積分時間、微分時間の値を示します。

処理の手順は、以下のとおりです。

1 **ユーザが、AC_StepProgram 命令の動作指令「ACSP_Enable」の値を TRUE にします。**
AC_StepProgram 命令が起動し、現在ステップ番号「StepNo」の値が 0 になります。

2 **ユーザが、PIDAT 命令の実行条件「Run」の値を TRUE にします。**
PIDAT 命令が起動します。

3 **ユーザが、オートチューニング実行条件「StartAT」の値を FALSE から TRUE にします。**
AC_StepProgram 命令のホールド「Hold」の値が TRUE になり、ホールド機能が実行されます。
PIDAT 命令のオートチューニング機能が実行され、ステップ 0 に対する最適な PID パラメータが算出されます。

4 **オートチューニングが終了します。**
PIDAT 命令のオートチューニング正常終了「ATDone」の値が、TRUE になります。
算出された PID パラメータが、PIDbank[0]に格納されます。

5 **ユーザが、AC_StepProgram 命令の「Hold」の値を FALSE にします。**
AC_StepProgram 命令のホールド機能が解除されます。
しばらく時間が経つと、ステップがつぎに移り、「StepNo」の値が 1 になります。

6 **ユーザが、ステップの数だけ上記 3～5 を繰り返します。**
すべてのステップに対する最適な PID パラメータが、PIDbank[]に格納されます。

- **プログラムパターンに従った温度制御**

各ステップに対する最適な PID パラメータを使用して、プログラムパターンに従った温度制御を実行します。

処理の手順は、以下のとおりです。

1 **ユーザが、AC_StepProgram 命令の動作指令「ACSP_Enable」の値を TRUE にします。**
AC_StepProgram 命令が起動し、ステップ番号「StepNo」の値が 0 になります。

2 **ユーザが、PIDAT 命令の実行条件「Run」の値を TRUE にします。**
PIDAT 命令が起動します。

3 **タスク周期ごとに、PIDAT 命令の操作量「MV」が出力されます。**

4 **TimeProportionalOut 命令により、「MV」の値に対する時分割比例出力が実行されます。**

5 **しばらく時間が経つと、ステップがつぎに移ります。**

6 **上記 3～5 が、終了ステップまで繰り返されます。**

Sysmac Studio の設定

本サンプルプログラムを実行するためには、Sysmac Studio で、ネットワーク構成、I/O マップ、データ型定義の設定をしておくことが必要です。

● ネットワーク設定

ネットワーク構成は、以下のとおりです。EtherCAT のノードアドレス 1 に、以下の構成のスレーブターミナルが接続しています。それぞれに、以下のデバイス名を割り付けます。

ユニット番号	形式	ユニット	デバイス名
0	形 NX-ECC201	EtherCAT カプラユニット	E001
1	形 NX-TS2101	温度入力ユニット	N1
2	形 NX-OD3121	デジタル出力ユニット	N2

● I/O マップ

I/O マップの設定は、以下のとおりです。

位置	ポート	説明	R/W	データ型	変数	変数種別
Unit1	Ch1 Measured Value REAL *1	チャネルの測定値（REAL 型）	R	REAL	N1_Ch1_Measured_Value_REAL	グローバル変数
Unit1	Ch2 Measured Value REAL *2	チャネルの測定値（REAL 型）	R	REAL	N1_Ch2_Measured_Value_REAL	グローバル変数
Unit2	Output Bit 00	出力接点 00	W	BOOL	N2_Output_Bit_00	グローバル変数
Unit2	Output Bit 01	出力接点 01	W	BOOL	N2_Output_Bit_01	グローバル変数
Unit2	Output Bit 02	出力接点 02	W	BOOL	N2_Output_Bit_02	グローバル変数
Unit2	Output Bit 03	出力接点 03	W	BOOL	N2_Output_Bit_03	グローバル変数

*1. 温度入力ユニット形 NX-TS2101 の I/O エントリに、0x6003:01(Ch1 Measured Value REAL)を追加する必要があります。
*2. 温度入力ユニット形 NX-TS2101 の I/O エントリに、0x6003:02(Ch2 Measured Value REAL)を追加する必要があります。

● データ型定義

以下のように、構造体型 sPID_BANK を定義します。

構造体型	名称	データ型	コメント
▼	sPID_BANK	STRUCT	PID パラメータ構造体
	PB	REAL	比例帯
	TI	TIME	積分時間
	TD	TIME	微分時間

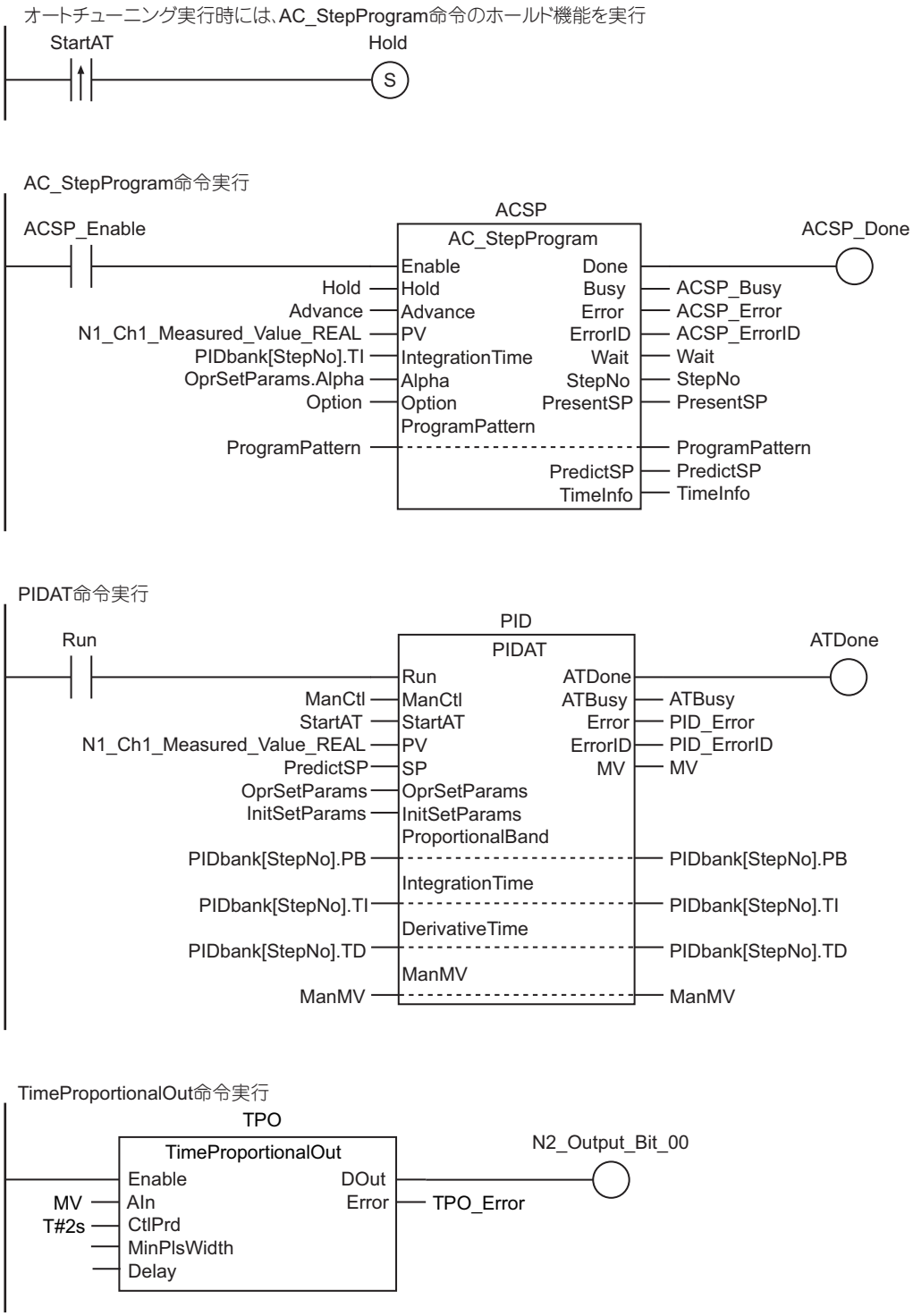
LD

内部変数	名称	データ型	初期値	コメント
	ACSP_Enable	BOOL	FALSE	AC_StepProgram の動作指令
	Hold	BOOL	FALSE	ホールド
	Advance	BOOL	FALSE	前進

内部変数	名称	データ型	初期値	コメント
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	オプション
	ProgramPattern	ARRAY[0..7] OF _sAC_STEP_DATA	[(ReachSP:=30.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=100.0, TimeWidth:=T#10m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=120.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#4m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#5m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=10.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m)]	プログラムパターン
	ACSP_Busy	BOOL	FALSE	AC_StepProgram 実行中
	ACSP_Error	BOOL	FALSE	AC_StepProgram エラー
	ACSP_ErrorID	WORD	WORD#16#0	AC_StepProgram エラーコード
	Wait	BOOL	FALSE	ウェイト中
	StepNo	USINT	0	現在ステップ番号
	PresentSP	REAL	0.0	現在目標値
	PredictSP	REAL	0.0	予測目標値

内部変数	名称	データ型	初期値	コメント
	TimeInfo	_sAC_STEP_TIME	(ProgramTime:=T#0s, ElapseTime:=T#0s, ProgressTime:=T#0s, LeftTime:=T#0s, StepProgressTime:=T#0s, StepLeftTime:=T#0s)	時間情報
	ACSP_Done	BOOL	FALSE	AC_StepProgram 実行完了
	Run	BOOL	FALSE	PIDAT 命令実行条件
	ManCtl	BOOL	FALSE	マニュアル／オート切替
	StartAT	BOOL	FALSE	オートチューニング実行条件
	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)	運転中設定パラメータ
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#250ms, RngLowLmt:=-200.0, RngUpLmt:=1300.0, DirOpr:=FALSE)	初期設定パラメータ
	ManMV	REAL	0.0	マニュアル操作量
	ATBusy	BOOL	FALSE	オートチューニング実行中
	PID_ErrorID	WORD	WORD#16#0	PIDAT エラーコード
	PID_Error	BOOL	FALSE	PIDAT エラー
	MV	REAL	0.0	操作量
	ATDone	BOOL	FALSE	オートチューニング正常終了
	TPO_Error	BOOL	FALSE	TimeProportional Out エラー
	PIDbank	ARRAY[0..7] OF sPID_BANK	[8((PB:=10, TI:=T#233s, TD:=T#60s))]	最適 PID パラメータ格納配列
	ACSP	AC_StepProgram		
	PID	PIDAT		
	TPO	TimeProportional Out		

外部変数	名称	データ型	コンスタント	コメント
	N1_Ch1_Measured_Value_REAL	REAL	☐	チャネルの測定値（REAL 型）
	N2_Output_Bit_00	BOOL	☐	出力接点



ST

内部変数	名称	データ型	初期値	コメント
	ACSP_Enable	BOOL	FALSE	AC_StepProgramの動作指令
	Hold	BOOL	FALSE	ホールド
	Advance	BOOL	FALSE	前進

内部変数	名称	データ型	初期値	コメント
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	オプション
	ProgramPattern	ARRAY[0..7] OF _sAC_STEP_DATA	[(ReachSP:=30.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=100.0, TimeWidth:=T#10m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=120.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#4m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#5m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=10.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m)]	プログラムパターン
	ACSP_Busy	BOOL	FALSE	AC_StepProgram 実行中
	ACSP_Error	BOOL	FALSE	AC_StepProgram エラー
	ACSP_ErrorID	WORD	WORD#16#0	AC_StepProgram エラーコード
	Wait	BOOL	FALSE	ウェイト中
	StepNo	USINT	0	現在ステップ番号
	PresentSP	REAL	0.0	現在目標値
	PredictSP	REAL	0.0	予測目標値

内部変数	名称	データ型	初期値	コメント
	TimeInfo	_sAC_STEP_TIME	(ProgramTime:=T#0s, ElapseTime:=T#0s, ProgressTime:=T#0s, LeftTime:=T#0s, StepProgressTime:=T#0s, StepLeftTime:=T#0s)	時間情報
	ACSP_Done	BOOL	FALSE	AC_StepProgram 実行完了
	Run	BOOL	FALSE	PIDAT 命令実行条件
	ManCtl	BOOL	FALSE	マニュアル／オート切替
	StartAT	BOOL	FALSE	オートチューニング実行条件
	PreStartAT	BOOL	TRUE	前タスク周期のオートチューニング実行条件
	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)	運転中設定パラメータ
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#250ms, RngLowLmt:=-200.0, RngUpLmt:=1300.0, DirOpr:=FALSE)	初期設定パラメータ
	ManMV	REAL	0.0	マニュアル操作量
	ATBusy	BOOL	FALSE	オートチューニング実行中
	PID_ErrorID	WORD	WORD#16#0	PIDAT エラーコード
	PID_Error	BOOL	FALSE	PIDAT エラー
	MV	REAL	0.0	操作量
	ATDone	BOOL	FALSE	オートチューニング正常終了
	TPO_Error	BOOL	FALSE	TimeProportional Out エラー
	PIDbank	ARRAY[0..7] OF sPID_BANK	[8((PB:=10, TI:=T#233s, TD:=T#60s))]	最適 PID パラメータ格納配列
	TPO_Enable	BOOL	FALSE	TimeProportional Out の動作指令
	MinPlsWidth	REAL	0.0	最小パルス幅
	Delay	REAL	0.0	ディレー
	ACSP	AC_StepProgram		
	PID	PIDAT		
	TPO	TimeProportional Out		

外部変数	名称	データ型	コンスタント	コメント
	N1_Ch1_Measured_Value_REAL	REAL	┌┐	チャネルの測定値（REAL 型）
	N2_Output_Bit_00	BOOL	┌┐	出力接点

```
TPO_Enable := TRUE;

// オートチューニング実行時には、AC_StepProgram 命令のホールド機能を実行
IF StartAT AND PreStartAT=FALSE THEN
  Hold := TRUE;
END_IF;
PreStartAT := StartAT;

// AC_StepProgram 命令実行
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;

// PIDAT 命令実行
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;

// TimeProportionalOut 命令実行
TPO(Enable :=TPO_Enable,
  AIn :=MV,
  CtlPrd :=T#2s,
  MinPlsWidth:=MinPlsWidth,
  Delay :=Delay,
  DOut =>N2_Output_Bit_00,
  Error =>TPO_Error);
```

システム制御命令

命令	名称	ページ
TraceSamp	データトレースサンプリング	P.2-802
TraceTrig	データトレーストリガ条件成立	P.2-805
GetTraceStatus	データトレースステータス読出	P.2-808
SetAlarm	ユーザ異常発生	P.2-811
ResetAlarm	ユーザ異常解除	P.2-816
GetAlarm	ユーザ異常ステータス取得	P.2-818
ResetPLCError	PLC 異常解除	P.2-820
GetPLCError	PLC 異常ステータス取得	P.2-823
GetEIPError	EtherNet/IP 異常ステータス取得	P.2-825
ResetMCError	モーション制御異常解除	P.2-827
GetMCError	モーション制御異常ステータス取得	P.2-832
ResetECError	EtherCAT 異常解除	P.2-834
GetECError	EtherCAT 異常ステータス取得	P.2-836
SetInfo	ユーザ情報発生	P.2-838
RestartNXUnit	NX ユニットリスタート	P.2-840
NX_ChangeWriteMode	NX ユニット書込モード変更	P.2-845
NX_SaveParam	NX ユニットパラメータ保存	P.2-850
NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	P.2-855

TraceSamp

データトレースのサンプリングを実行します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TraceSamp	データトレースサンプリング	FUN	(@)TraceSamp EN ENO TraceNo Out Point	TraceSamp(TraceNo, Point);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
TraceNo	トレース番号	入力	トレース番号	0 ～ 3	−	0
Point	サンプリングポイント番号		サンプリングポイント番号	データ型に従う		
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

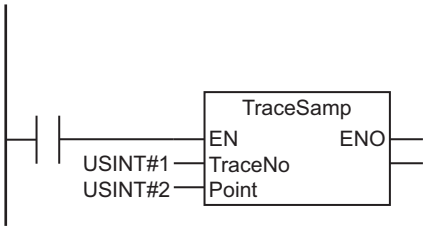
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						○														
Point						○														
Out	○																			

機能

データトレースの、サンプリングを実行します。
Sysmac Studio により指定されたサンプリング設定に従い、サンプリング対象の全変数の現在値を読み出し、トレース番号「TraceNo」とサンプリングポイント番号「Point」の値を付加して、トレースメモリに格納します。
ただし、本命令が実行されるのは、データトレースが実行中であり、かつ、Sysmac Studio によるトレースのサンプリングタイミングの設定が、［サンプリング命令］となっている場合だけです。

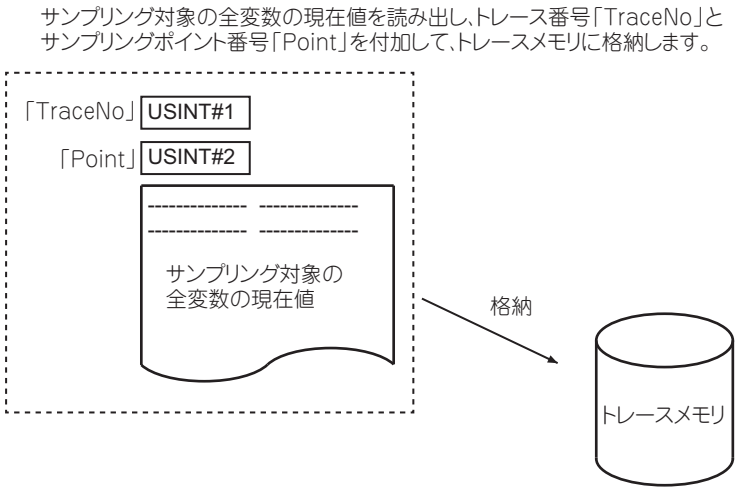
記述例を、以下に示します。トレース番号 1、サンプリングポイント番号 2 の値を付加して、サンプリング対象の全変数の現在値を、トレースメモリに格納します。

LD



ST

```
TraceSamp(USINT#1, USINT#2);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_PLC_TraceSta[0..3]	トレース情報	_sTRACE_STA[]	トレース情報 *1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。

参考

- データトレース機能の詳細については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。
- トレースとは、指定した条件時に、指定した変数の値をサンプリングする機能です。各条件は、Sysmac Studio で指定します。
- 本命令は、プログラム上に複数個配置することができます。条件に応じて、サンプリングを行うプログラムを記述することができます。
- 「Point」を適切に設定することで、Sysmac Studio のデータトレースウィンドウから、どの TraceSamp 命令でサンプリングした値かを特定できます。「Point」を省略した場合の値は 0 です。

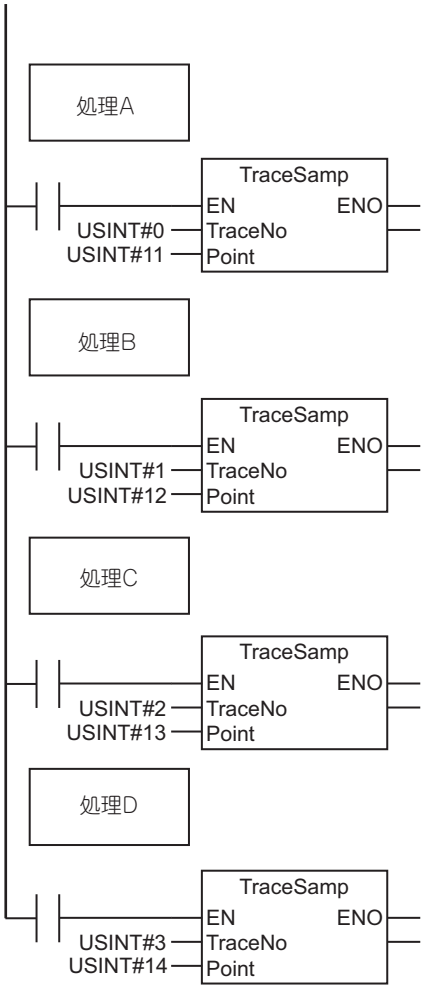
使用上の注意

- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- 以下の場合は、何も行わず正常終了します。
  a）データトレースが停止している場合。
  b）トレース設定のサンプリングのタイミングが、［サンプリング命令］以外の場合。
  c）「TraceNo」の値が、Sysmac Studio で設定されているトレース番号ではない場合。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a）「TraceNo」の値が、有効範囲外の場合。

サンプルプログラム

A～D の各処理の完了時点で、サンプリングを実行します。
それぞれの時点での、各変数の値が記録されます。

LD



ST

```
処理A
TraceSamp(USINT#0, USINT#11);
処理B
TraceSamp(USINT#1, USINT#12);
処理C
TraceSamp(USINT#2, USINT#13);
処理D
TraceSamp(USINT#3, USINT#14);
```

TraceTrig

データトレースのトリガを発生させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
TraceTrig	データトレーストリガ条件成立	FUN	(@)TraceTrig EN ENO TraceNo Out	TraceTrig(TraceNo);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
TraceNo	トレース番号	入力	トレース番号	0 ～ 3	−	0
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

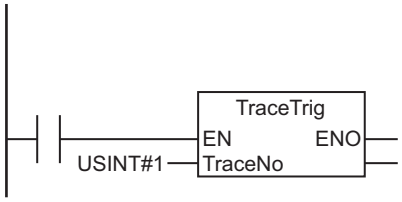
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						○														
Out	○																			

機能

データトレースのトリガを、強制的に発生させます。
Sysmac Studio によって指定されたトリガ条件が、成立しているかどうかは無関係です。
本命令実行時に、トレース番号「TraceNo」のデータトレースが実行中であれば、サンプリングを実行します。

記述例を、以下に示します。トレース番号 1 のデータトレースのトリガを、発生させます。

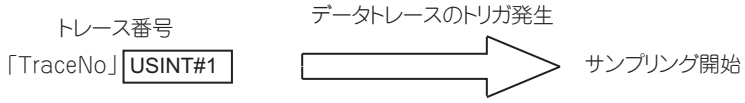
LD



ST

```
TraceTrig(USINT#1);
```

トレース番号「TraceNo」のデータトレースのトリガを、発生させます。



関連するシステム定義変数

変数名	名称	データ型	内容
_PLC_TraceSta[0..3]	トレース情報	_sTRACE_STA[]	トレース情報 *1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

参考

- データトレース機能の詳細については、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。
- 本命令は、プログラム上に複数配置することが可能です。条件に応じて、トリガ条件を成立させるプログラムを記述することができます。
- 変数と変数の比較など、通常のトリガ条件では設定できないタイミングで、トリガ条件を成立させるプログラムを記述することができます。

使用上の注意

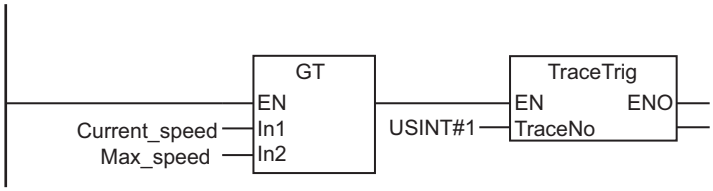
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- 以下の場合は、何も行わず正常終了します。
  a）データトレースが停止している場合。
  b）すでにトリガ条件が成立している場合。
  c）「TraceNo」の値が、Sysmac Studio で設定されているトレース番号ではない場合。
  d）「TraceNo」で指定しているトレース番号のトレースタイプが、［連続トレース］の場合。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a）「TraceNo」の値が、有効範囲外の場合。

サンプルプログラム

速度の現在値が、最大限度値を超えていたら、強制的にデータトレースのトリガを発生させて、各変数の値を記録します。
現在速度「Current_speed」の値が、速度上限値「Max_speed」の値を超えていたら、TraceTrig 命令を実行します。

LD

名称	データ型	初期値	コメント
Current_speed	INT	0	現在速度
Max_speed	INT	20	速度上限値



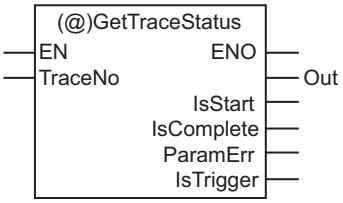
ST

名称	データ型	初期値	コメント
Current_speed	INT	0	現在速度
Max_speed	INT	20	速度上限値

```
IF (Current_speed > Max_speed) THEN
  TraceTrig(USINT#1);
END_IF;
```

GetTraceStatus

データトレースの実行ステータスを、読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetTraceStatus	データトレースステータス読出	FUN		GetTraceStatus(TraceNo, IsStart, IsComplete, ParamErr, IsTrigger);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
TraceNo	トレース番号	入力	トレース番号	0 ～ 3	−	0
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−
IsStart	実行中フラグ		TRUE: トレース実行中 FALSE: トレース実行中ではない	データ型に従う		
IsComplete	完了フラグ		TRUE: トレース実行完了 FALSE: トレース実行中または未実行			
ParamErr	パラメータ異常フラグ		TRUE: トレース設定異常あり FALSE: トレース設定異常なし			
IsTrigger	トリガフラグ		TRUE: トレーストリガ条件成立 FALSE: トレーストリガ条件非成立			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						○														
Out	○																			
IsStart	○																			
IsComplete	○																			
ParamErr	○																			
IsTrigger	○																			

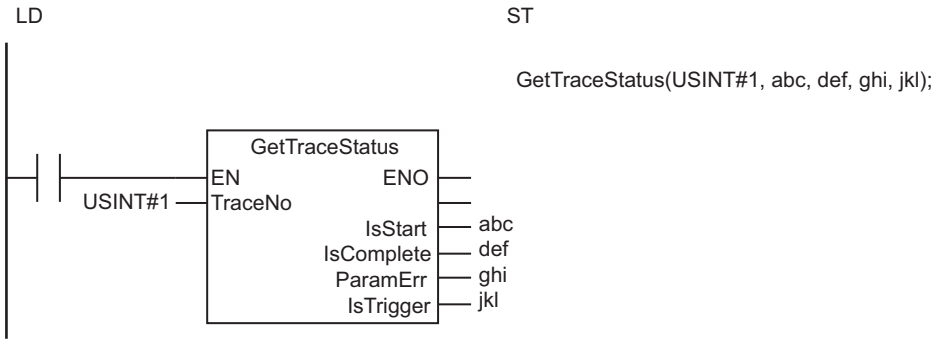
機能

トレース番号「TraceNo」で指定したデータトレースの、実行ステータスを読み出します。

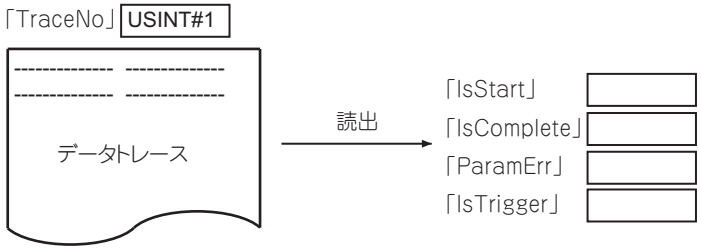
読み出す情報は、実行中フラグ「IsStart」、完了フラグ「IsComplete」、パラメータ異常フラグ「ParamErr」、トリガフラグ「IsTrigger」です。

「ParamErr」の値が TRUE になるのは、以下のように、トレース設定に異常がある場合です。
- トリガ設定、サンプリング設定に指定している変数が、存在しない。
- サンプリングタイミングの設定が、［指定タスク周期］になっていて、指定したタスクが存在しない。

記述例を、以下に示します。トレース番号 1 の実行ステータスを、読み出します。



トレース番号「TraceNo」で指定したデータトレースの、実行ステータスを読み出します。



関連するシステム定義変数

変数名	名称	データ型	内容
_PLC_TraceSta[0..3]	トレース情報	_sTRACE_STA[]	トレース情報 *1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

参考

データトレース機能の詳細については、 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。

使用上の注意

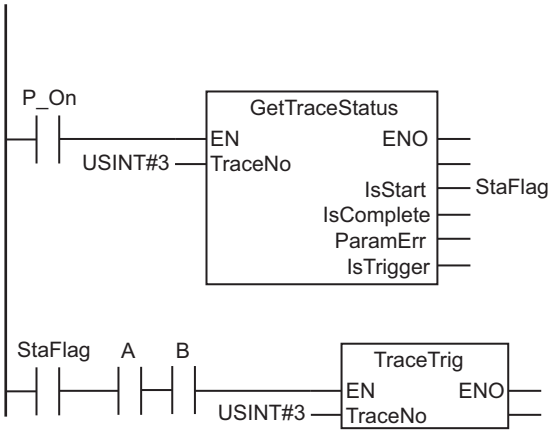
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- 本命令で読み出す情報は、システム定義変数_PLC_TraceSta[]の内容です。この変数には、直接アクセスできません。この変数の内容を読み出すには、必ず本命令を使用してください。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a)「TraceNo」の値が、有効範囲外の場合。

サンプルプログラム

GetTraceStatus 命令でトレース番号 3 の実行ステータスを読み出します。データトレース実行中であり、かつ変数 A と B の値がどちらも TRUE ならば、TraceTrig 命令を実行して、データトレースのトリガを発生させます。

LD

名称	データ型	初期値	コメント
StaFlag	BOOL	FALSE	トレースの実行ステータス
A	BOOL	FALSE	
B	BOOL	FALSE	



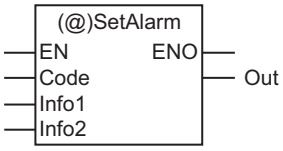
ST

名称	データ型	初期値	コメント
StaFlag	BOOL	FALSE	トレースの実行ステータス
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

ユーザ異常を発生させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SetAlarm	ユーザ異常発生	FUN		SetAlarm(Code, Info1, Info2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Code	イベントコード	入力	発生させるユーザ異常のイベントコード	1 ～ 40000	−	1
Info1	付属情報 1		ユーザ異常発生とともに、イベントログに記録する値	データ型に従う		*1
Info2	付属情報 2					
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							○													
Info1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Info2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Out	○																			

機能

Sysmac Studio の、イベント設定テーブルに定義したイベントコード「Code」に対応するユーザ異常を発生させます。
また、イベントコードの重要度に対応するユーザイベントログ領域に、発生時刻、イベント名称、イベントグループ、イベントコード「Code」、イベント重要度、付属情報「Info1」、「Info2」、詳細情報の値を記録します。発生時刻は自動的に取得した値を記録します。イベント名称、イベントグループ、詳細情報は、SysmacStudio で設定された情報を記録します。

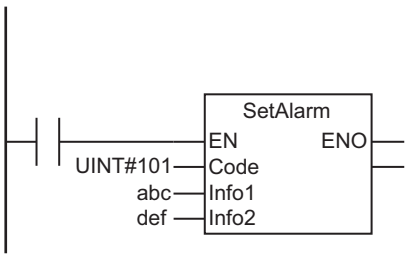
イベント重要度は、以下のように、イベントコードごとに定められた分類を記録します。分類番号が小さいほど、イベント重要度は高くなります。

イベントコード	分類(User fault Level)
1 ～ 5000	1
5001 ～ 10000	2
10001 ～ 15000	3
15001 ～ 20000	4
20001 ～ 25000	5

イベントコード	分類(User fault Level)
25001 ～ 30000	6
30001 ～ 35000	7
35001 ～ 40000	8

記述例を、以下に示します。イベントコード 101 に対応するユーザ異常を、発生させます。付属情報として、変数「abc」、「def」の値を記録します。

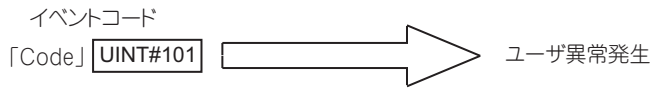
LD

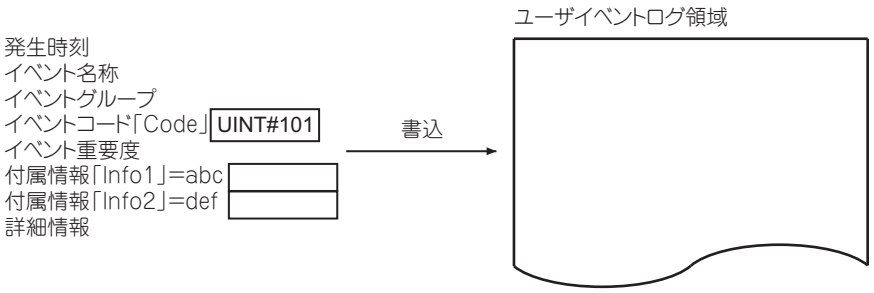


ST

```
SetAlarm(UINT#101, abc, def);
```

イベントコード「Code」に対応するユーザ異常を発生させます。
また、ユーザイベントログ領域に、発生時刻、イベント名称、イベントグループ、イベントコード「Code」、イベント重要度、付属情報「Info1」、「Info2」、詳細情報の値を記録します。





関連するシステム定義変数

変数名	名称	データ型	内容
_AlarmFlag	ユーザ異常の異常ステータス	WORD	ユーザ異常の検知有無を示すフラグ。 ビット 0 からビット 7 が、ユーザ異常の各重要度(1～8)の発生状況を示します。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

参考

「Info1」、「Info2」には、グローバル変数を指定することも、ローカル変数を指定することも可能です。

使用上の注意

- 同時に発生させられるユーザ異常は、8 つのイベント重要度ごとに 32 件ずつ（最大 256 件）です。
- すでに同じイベントコードのユーザ異常が発生中の場合は、イベントログへの記録を行いません。
- 「Info1」、「Info2」に渡す入力パラメータは、必ず変数にしてください。定数にすると、ビルド時に異常になります。
- 「Code」の値に、Sysmac Studio で設定していないイベントコードを設定した場合、異常にはなりません。この場合、イベントグループ、詳細情報は、ユーザイベントログ領域に記録されません。また、イベント名称としては、「Code」の値が記録されます。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a)「Code」の値が、有効範囲外の場合。
  b）最大件数以上のユーザ異常を、発生させようとした場合。

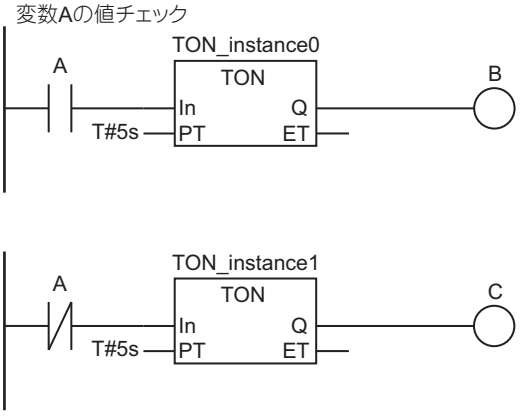
サンプルプログラム

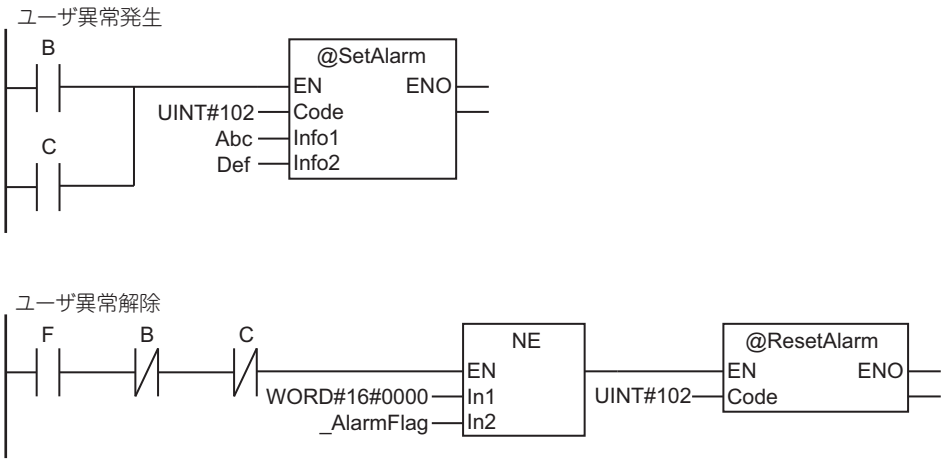
変数「A」の値は、正常時には、5s ごとに TRUE と FALSE を繰り返すものとします。
「A」の値を監視し、5s 以上値が変化しなかった場合に、イベントコード 102 のユーザ異常を発生させます。付属情報は、UINT#123 と UINT#456 とします。
変数「F」の値が TRUE になったら、ユーザ異常を解除します。

LD

内部変数	名称	データ型	初期値
	A	BOOL	FALSE
	B	BOOL	FALSE
	C	BOOL	FALSE
	F	BOOL	FALSE
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

外部変数	名称	データ型	コンスタント	コメント
	_AlarmFlag	WORD	☑	ユーザ異常の異常ステータス





ST

内部変数	名称	データ型	初期値
	A	BOOL	FALSE
	B	BOOL	FALSE
	C	BOOL	FALSE
	F	BOOL	FALSE
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

外部変数	名称	データ型	コンスタント	コメント
	_AlarmFlag	WORD	☑	ユーザ異常の異常ステータス

```
// 変数 A の値チェック
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;

// ユーザ異常発生
IF (B=TRUE) OR (C=TRUE) THEN
  SetAlarm(
    Code:=UINT#102,
    Info1 :=Abc,
    info2 :=Def);
END_IF;
```

```
// ユーザ異常解除
IF (F=TRUE) & (B=FALSE) & (C=FALSE) & (_AlarmFlag<>WORD#16#0000) THEN
  ResetAlarm(Code:=UINT#102);
END_IF;
```

ResetAlarm

ユーザ異常を解除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetAlarm	ユーザ異常解除	FUN	(@)ResetAlarm EN ENO Code Out	ResetAlarm(Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Code	イベントコード	入力	解除するユーザ異常のイベントコード。 0: すべてのユーザ異常解除	0 ～ 40000	−	0
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							○													
Out	○																			

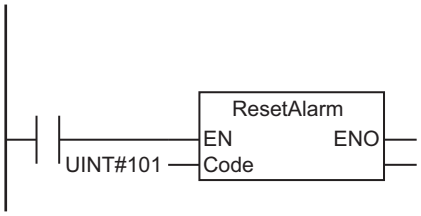
機能

イベントコード「Code」で指定したユーザ異常を、解除します。
そして、ユーザイベントログ領域に、ユーザ異常の個別解除を意味するイベントを記録します。このイベントのイベントコードは No.65533、重要度は“User Information”です。

「Code」の値が 0 の場合、発生しているすべてのユーザ異常を解除します。
この場合は、ユーザイベントログ領域に、ユーザ異常の全解除を意味するイベントを記録します。このイベントのイベントコードは No.65534、重要度は“User Information”です。

記述例を、以下に示します。イベントコード 101 に対するユーザ異常を、解除します。

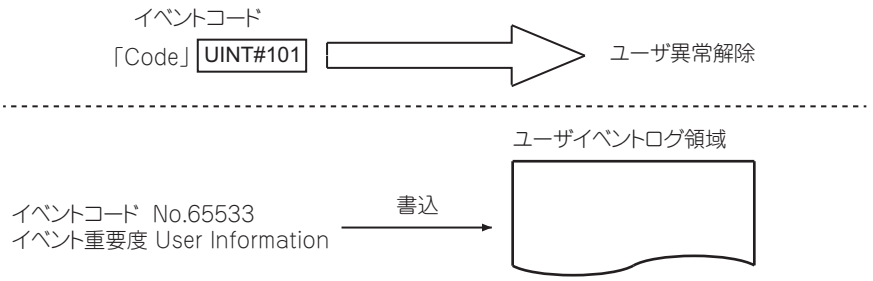
LD



ST

```
ResetAlarm(UINT#101);
```

イベントコード「Code」で指定したユーザ異常を解除します。
また、ユーザイベントログ領域に、ユーザ異常の個別解除を意味するイベントを記録します。



関連するシステム定義変数

変数名	名称	データ型	内容
_AlarmFlag	ユーザ異常の異常ステータス	WORD	ユーザ異常の検知有無を示すフラグ。 ビット 0 からビット 7 が、ユーザ異常の各重要度(1～8)の発生状況を示します。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

使用上の注意

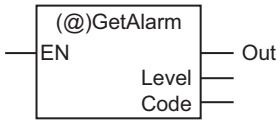
- 「Code」で指定したユーザ異常が発生していない場合、異常にはなりません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a)「Code」の値が、有効範囲外の場合。

サンプルプログラム

SetAlarm 命令の「サンプルプログラム (P.2-813)」を参照してください。

GetAlarm

発生中のユーザ異常の、最重要のイベント重要度（ユーザ異常レベル 1～8）と、最重要のイベントコードを取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetAlarm	ユーザ異常ステータス取得	FUN		Out:=GetAlarm(Level, Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	異常有無	出力	TRUE: 発生中のユーザ異常あり FALSE: 発生中のユーザ異常なし	データ型に従う	–	–
Level	最重要イベント重要度		発生中のユーザ異常のうち、最重要のイベント重要度 0: ユーザ異常なし 1 ～ 8: イベント重要度	0 ～ 8		
Code	最重要イベントコード		発生中のユーザ異常のうち、最重要のイベントコード 0: ユーザ異常なし 1 ～ 40000: イベントコード	0 ～ 40000		

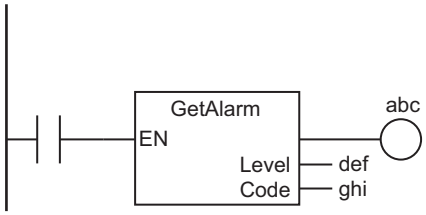
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Level							○													
Code							○													

機能

発生中のユーザ異常の、最重要のイベント重要度「Level」と、最重要のイベントコード「Code」を取得します。
発生中のユーザ異常がない場合、異常有無「Out」の値が FALSE になります。
最重要のイベント重要度のユーザ異常が複数発生している場合、「Code」の値は、最も早く発生したユーザ異常のイベントコードになります。

記述例を、以下に示します。

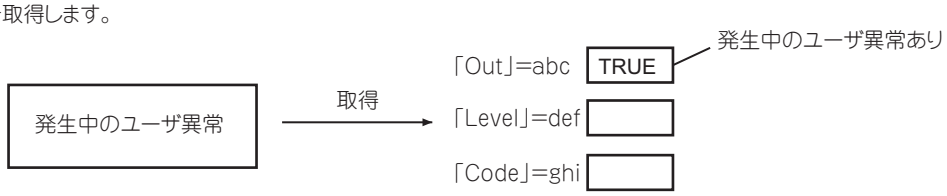
LD



ST

```
abc:=GetAlarm(def, ghi);
```

発生中のユーザ異常の最重要のイベント重要度「Level」と、最重要のイベントコード「Code」を取得します。



関連するシステム定義変数

変数名	名称	データ型	内容
_AlarmFlag	ユーザ異常の異常ステータス	WORD	ユーザ異常の検知有無を示すフラグ。 ビット 0 からビット 7 が、ユーザ異常の各重要度(1～8)の発生状況を示します。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

使用上の注意

本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。

ResetPLCError

PLC 機能モジュールで発生中のコントローラ異常を解除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetPLCError	PLC 異常解除	FB	ResetPLCError_instance ResetPLCError Execute / Done / Busy / Error / ErrorID	ResetPLCError(Execute, Done, Busy, Error, ErrorID);

変数

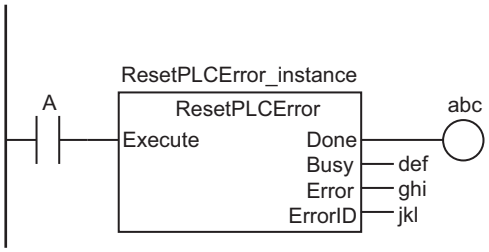
共通の変数だけ

機能

PLC 機能モジュールで発生中のコントローラ異常を解除します。

記述例を、以下に示します。

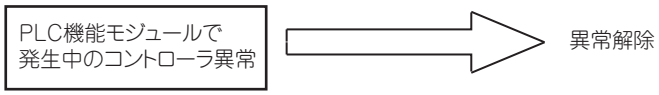
LD



ST

```
ResetPLCError_instance(A, abc, def, ghi, jkl);
```

PLC機能モジュールで発生中のコントローラ異常を解除します。



関連するシステム定義変数

変数名	名称	データ型	内容
_PLC_ErrSta	PLC 機能モジュール異常ステータス	WORD	PLC 機能モジュールで検知した異常の状態。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

使用上の注意

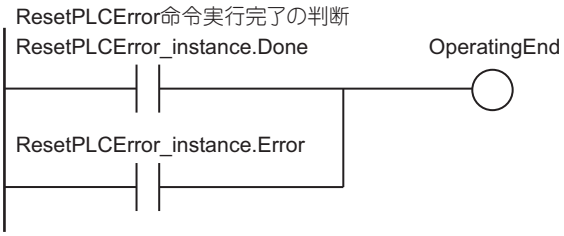
本命令の実行直後に、異常が解除されるとは限りません。異常が解除されたかどうかは、GetPLCError 命令で確認してください。

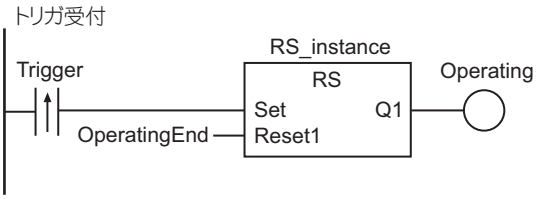
サンプルプログラム

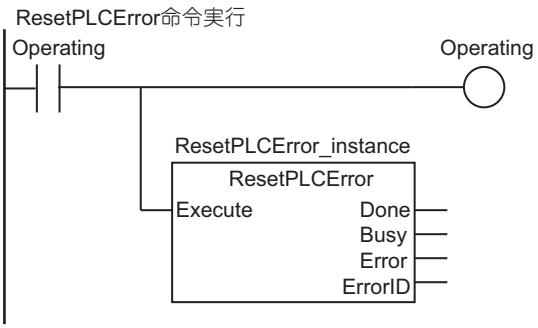
「Trigger」の値が FALSE→TRUE になったら、ResetPLCError 命令を実行します。ResetPLCError 命令が正常終了（「Done」の値が TRUE）したら、正常終了時処理を実行します。異常終了（「Error」の値が TRUE）したら、異常終了時処理を実行します。

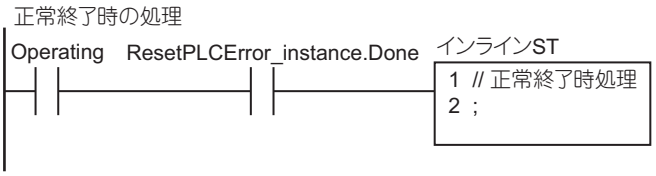
LD

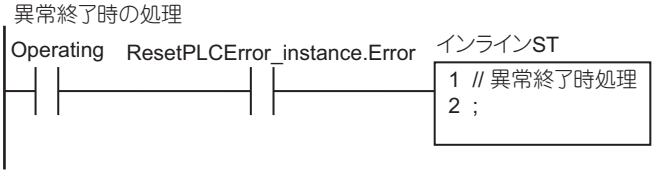
名称	データ型	初期値	コメント
OperatingEnd	BOOL	FALSE	処理終了
Trigger	BOOL	FALSE	実行条件
Operating	BOOL	FALSE	処理中
RS_instance	RS		
ResetPLCError_instance	ResetPLCError		











ST

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
OperatingStart	BOOL	FALSE	処理開始
Operating	BOOL	FALSE	処理中
ResetPLCError_instance	ResetPLCError		

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
  OperatingStart:=TRUE;
  Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// ResetPLCError_instance 初期化
IF (OperatingStart=TRUE) THEN
  ResetPLCError_instance(Execute:=FALSE);
  OperatingStart:=FALSE;
END_IF;

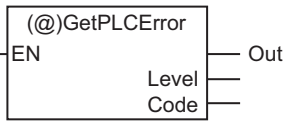
// ResetPLCError 命令実行
IF (Operating=TRUE) THEN
  ResetPLCError_instance(Execute:=TRUE);

  IF (ResetPLCError_instance.Done=TRUE) THEN
    // 正常終了時処理
    Operating:=FALSE;
  END_IF;

  IF (ResetPLCError_instance.Error=TRUE) THEN
    // 異常終了時処理
    Operating:=FALSE;
  END_IF;
END_IF;
```

GetPLCError

PLC 機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetPLCError	PLC 異常ステータス取得	FUN		Out:=GetPLCError(Level, Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	異常有無	出力	TRUE: 発生中のコントローラ異常あり FALSE: 発生中のコントローラ異常なし	データ型に従う	–	–
Level	最重要ステータス		PLC 機能モジュールで発生中のコントローラ異常のうち、最重要のステータス 0: コントローラ異常なし 2: 部分停止フォールトレベル 3: 軽度フォールトレベル	0, 2, 3		
Code	最重要イベントコード		PLC 機能モジュールで発生中のコントローラ異常のうち、最重要のイベントコード 16#0000_0000: コントローラ異常なし 16#0007_0000 ～ 16#FFFF_FFFF: イベントコード	16#00000000, 16#00070000 ～ 16#FFFFFFFF		

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Level							○													
Code				○																

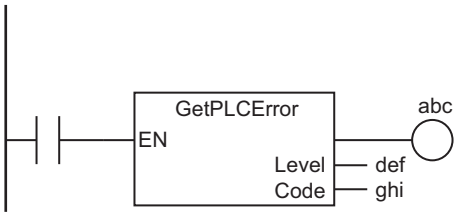
機能

PLC 機能モジュールで発生中のコントローラ異常の、最重要のステータス「Level」と、最重要のイベントコード「Code」を取得します。

発生中のコントローラ異常がない場合、異常有無「Out」の値が FALSE になります。
最重要のイベント重要度のコントローラ異常が複数発生している場合、「Code」の値は、最も早く発生したコントローラ異常のイベントコードになります。

記述例を、以下に示します。

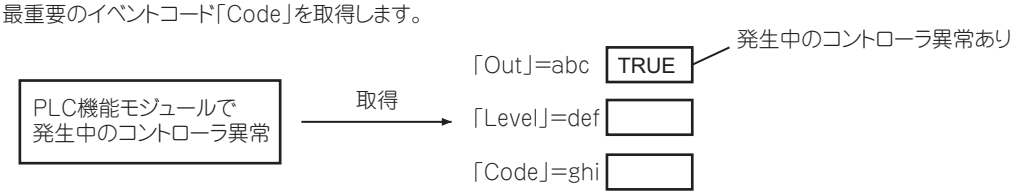
LD



ST

```
abc:=GetPLCError(def, ghi);
```

PLC機能モジュールで発生中のコントローラ異常の、最重要のステータス「Level」と、
最重要のイベントコード「Code」を取得します。



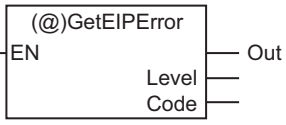
関連するシステム定義変数

変数名	名称	データ型	内容
_PLC_ErrSta	PLC 機能モジュール異常ステータス	WORD	PLC 機能モジュールで検知した異常の状態。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

GetEIPError

EtherNet/IP 機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetEIPError	EtherNet/IP 異常ステータス取得	FUN		Out:=GetEIPError(Level, Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	異常有無	出力	TRUE: 発生中のコントローラ異常あり FALSE: 発生中のコントローラ異常なし	データ型に従う	–	–
Level	最重要ステータス		EtherNet/IP 機能モジュールで発生中のコントローラ異常のうち、最重要のステータス 0: コントローラ異常なし 2: 部分停止フォールトレベル 3: 軽度フォールトレベル	0, 2, 3		
Code	最重要イベントコード		EtherNet/IP 機能モジュールで発生中のコントローラ異常のうち、最重要のイベントコード 16#0000_0000: コントローラ異常なし 16#0007_0000 ～ 16#FFFF_FFFF : イベントコード	16#00000000, 16#00070000 ～ 16#FFFFFFFF		

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Level							○													
Code				○																

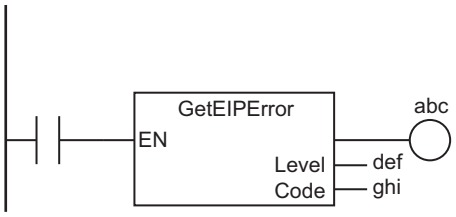
機能

EtherNet/IP 機能モジュールで発生中のコントローラ異常の、最重要のステータス「Level」と、最重要のイベントコード「Code」を取得します。

発生中のコントローラ異常がない場合、異常有無「Out」の値が FALSE になります。
最重要のイベント重要度のコントローラ異常が複数発生している場合、「Code」の値は、最も早く発生したコントローラ異常のイベントコードになります。

記述例を、以下に示します。

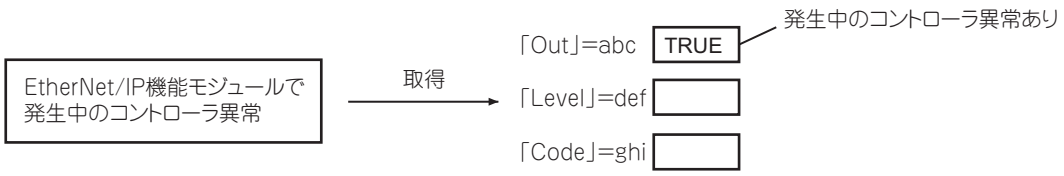
LD



ST

```
abc:=GetEIPError(def, ghi);
```

EtherNet/IP機能モジュールで発生中のコントローラ異常の、最重要のステータス「Level」と、
最重要のイベントコード「Code」を取得します。



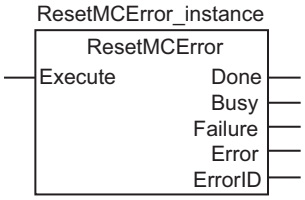
関連するシステム定義変数

変数名	名称	データ型	内容
_EIP_ErrSta	EtherNet/IP 機能モジュール異常ステータス	WORD	EtherNet/IP 機能モジュールで検知した異常の状態。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

ResetMCError

モーション制御機能モジュールで発生中のコントローラ異常を解除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetMCError	モーション制御異常解除	FB		ResetMCError_instance(Execute, Done, Busy, Failure Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Failure	不正終了	出力	TRUE: 異常解除失敗 FALSE: 異常解除成功	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Failure	○																			

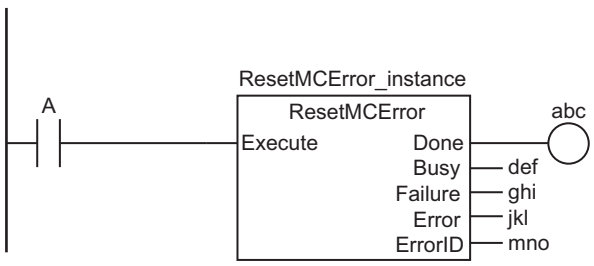
機能

モーション制御機能モジュールで発生中の、コントローラ異常を解除します。
解除に失敗した場合、「Failure」の値が TRUE になります。

ResetMCError 命令を実行するプログラムが、どのタスクに記述されていても、全軸、全軸グループが対象となります。

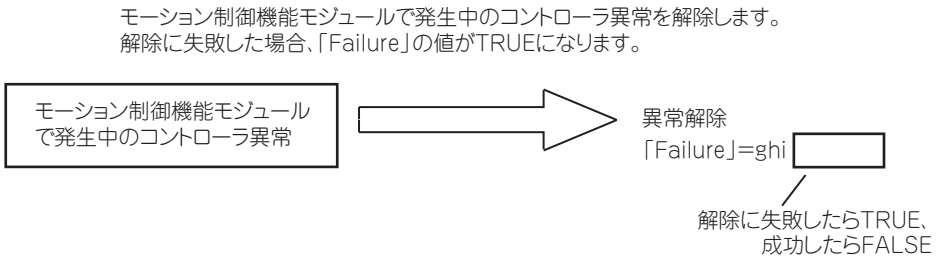
記述例を、以下に示します。

LD



ST

```
ResetMCError_instance(A, abc, def,
                      ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_MC_ErrSta	モーション制御異常ステータス	WORD	モーション制御機能モジュールで検知した異常の状態。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

使用上の注意

- 本命令の実行直後に、異常が解除されるとは限りません。異常が解除されたかどうかは、GetMCError 命令で確認してください。
- MC 試運転中に本命令を実行しようとすると、「Busy」の値が TRUE になったまま、実行できません。
- オムロン製サーボドライバ G5 シリーズに対して本命令を使用する場合、ResetECError 命令が同時に実行されないように、命令の排他制御を行ってください。本命令と ResetECError 命令を同時に実行すると、G5 シリーズがそれ以降の SDO を受け付けられなくなることがあります。

サンプルプログラム

EtherCAT マスタ機能モジュールおよびモーション制御機能モジュールで発生中の、コントローラ異常を検知します。もし、異常が発生していたら、その異常を解除します。

処理手順は、以下のとおりです。

1 **GetECError 命令で、EtherCAT マスタ機能モジュールで発生中の、コントローラ異常を検知します。**

2 **もし異常が発生していたら、ResetECError 命令で、異常を解除します。**

3 **GetMCError 命令で、モーション制御機能モジュールで発生中の、コントローラ異常を検知します。**

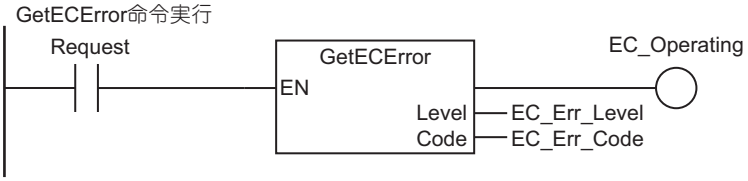
4 **もし異常が発生していたら、ResetMCError 命令で、異常を解除します。**

LD

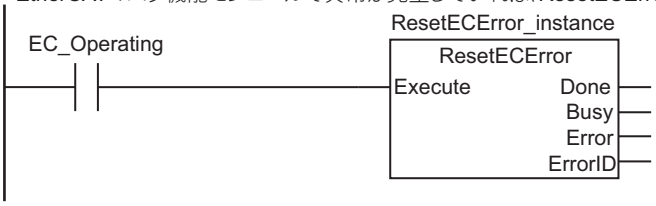
名称	データ型	初期値	コメント
Request	BOOL	FALSE	異常検知解除要求
EC_Err_Level	UINT	0	EtherCAT マスタ機能モジュール 最重要イベント重要度
EC_Err_Code	DWORD	DWORD#16#0	EtherCAT マスタ機能モジュール 最重要イベントコード

名称	データ型	初期値	コメント
EC_Operating	BOOL	FALSE	EtherCAT マスタ機能モジュール異常解除中
MC_Err_Level	UINT	0	モーション制御機能モジュール 最重要イベント重要度
MC_Err_Code	DWORD	DWORD#16#0	モーション制御機能モジュール 最重要イベントコード
MC_Operating	BOOL	FALSE	モーション制御機能モジュール異常解除中
Normal_End	BOOL	FALSE	正常終了
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		

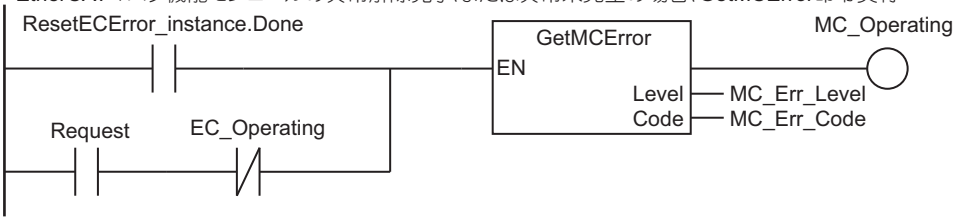
GetECError命令実行



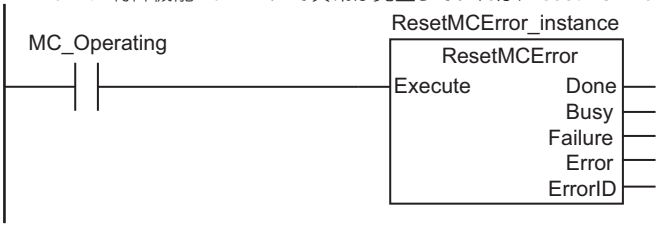
EtherCATマスタ機能モジュールで異常が発生していれば、ResetECError命令実行



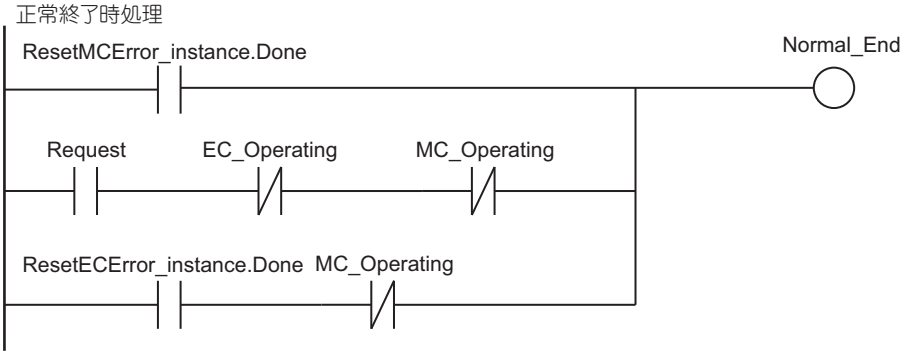
EtherCATマスタ機能モジュールの異常解除完了、または異常未発生の場合、GetMCError命令実行



モーション制御機能モジュールで異常が発生していれば、ResetMCError命令実行



正常終了時処理



ST

名称	データ型	初期値	コメント
Request	BOOL	FALSE	異常検知解除要求
EC_Error	BOOL	FALSE	EtherCAT マスタ機能モジュール異常あり
EC_Err_Level	UINT	0	EtherCAT マスタ機能モジュール最重要イベント重要度
EC_Err_Code	DWORD	DWORD#16#0	EtherCAT マスタ機能モジュール最重要イベントコード
EC_Stage	INT	0	EtherCAT マスタ機能モジュール異常解除状態
MC_Error	BOOL	FALSE	モーション制御機能モジュール異常あり
MC_Err_Level	UINT	0	モーション制御機能モジュール最重要イベント重要度
MC_Err_Code	DWORD	DWORD#16#0	モーション制御機能モジュール最重要イベントコード
MC_Stage	INT	0	モーション制御機能モジュール異常解除状態
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		

```
// 異常解除要求を判定
IF (Request=TRUE) THEN

// EtherCAT マスタ機能モジュールのコントローラ異常検知
  EC_Error:=GetECError(EC_Err_Level, EC_Err_Code);

// モーション制御機能モジュールのコントローラ異常検知
  MC_Error:=GetMCError(MC_Err_Level, MC_Err_Code);

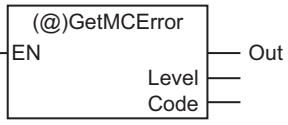
  IF (EC_Error=TRUE) THEN // EtherCAT マスタ機能モジュールのコントローラ異常あり
    CASE EC_Stage OF
    0 : // 初期化
      ResetECError_instance(Execute:=FALSE);
      EC_Stage:=INT#1;
    1 : // EtherCAT マスタ機能モジュールのコントローラ異常解除中
      ResetECError_instance(Execute:=TRUE);
      IF (ResetECError_instance.Done=TRUE) THEN
        EC_Stage:=INT#99; // 正常終了
      END_IF;
      IF (ResetECError_instance.Error=TRUE) THEN
        EC_Stage:=INT#98; // 異常終了
      END_IF;
    99 : // 正常終了時処理
      EC_Stage:=INT#0;
    98 : // 異常終了時処理
      EC_Stage:=INT#0;
    END_CASE;
  END_IF;

  IF (MC_Error=TRUE) THEN // モーション制御機能モジュールのコントローラ異常あり
    CASE MC_Stage OF
```

```
    0 : // 初期化
      ResetMCError_instance(Execute:=FALSE);
      MC_Stage:=INT#1;
    1 : // モーション制御機能モジュールのコントローラ異常解除中
      IF (EC_Error=FALSE) THEN
        ResetMCError_instance(Execute:=TRUE); // 全スレーブ復帰
        IF (ResetMCError_instance.Done=TRUE) THEN
          MC_Stage:=INT#99; // 正常終了
        END_IF;
        IF ( (ResetMCError_instance.Error=TRUE) OR (ResetMCError_instance.Failure=TRUE) ) THEN
          MC_Stage:=INT#98; // 異常終了
        END_IF;
      END_IF;
    99 : // 正常終了時処理
      MC_Stage:=INT#0;
    98 : // 異常終了時処理
      MC_Stage:=INT#0;
    END_CASE;
  END_IF;
END_IF;
```

GetMCError

モーション制御機能モジュールで発生中のコントローラ異常の、最重要のステータス（部分停止フォールト、軽度フォールト）と、最重要のイベントコードを取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetMCError	モーション制御異常ステータス取得	FUN		Out:=GetMCError(Level, Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	異常有無	出力	TRUE: 発生中のコントローラ異常あり FALSE: 発生中のコントローラ異常なし	データ型に従う	–	–
Level	最重要ステータス		モーション制御機能モジュールで発生中のコントローラ異常のうち、最重要のステータス 0: コントローラ異常なし 2: 部分停止フォールトレベル 3: 軽度フォールトレベル	0, 2, 3		
Code	最重要イベントコード		モーション制御機能モジュールで発生中のコントローラ異常のうち、最重要のイベントコード 16#0000_0000: コントローラ異常なし 16#0007_0000 ～ 16#FFFF_FFFF: イベントコード	16#00000000, 16#00070000 ～ 16#FFFFFFFF		

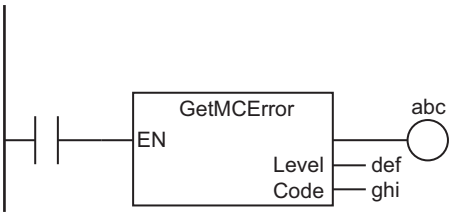
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Level							○													
Code				○																

機能

モーション制御機能モジュールで発生中のコントローラ異常の、最重要のステータス「Level」と、最重要のイベントコード「Code」を取得します。
発生中のコントローラ異常がない場合、異常有無「Out」の値が FALSE になります。
最重要のイベント重要度のコントローラ異常が複数発生している場合、「Code」の値は、最も早く発生したコントローラ異常のイベントコードになります。

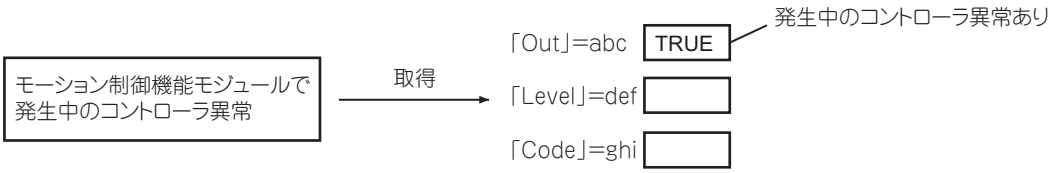
記述例を、以下に示します。

LD



ST

```
abc:=GetMCError(def, ghi);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_MC_ErrSta	モーション制御異常ステータス	WORD	モーション制御機能モジュールで検知した異常の状態。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.2-828)」を参照してください。

ResetECError

EtherCAT マスタ機能モジュールで発生中のコントローラ異常を解除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetECError	EtherCAT 異常解除	FB	ResetECError_instance ResetECError Execute / Done / Busy / Error / ErrorID	ResetECError_instance(Execute, Done, Busy, Error, ErrorID);

変数

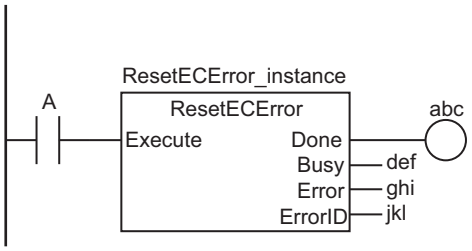
共通の変数だけ

機能

EtherCAT マスタ機能モジュールで発生中のコントローラ異常を解除します。

記述例を、以下に示します。

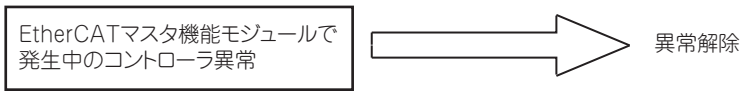
LD



ST

```
ResetECError_instance(A, abc, def, ghi, jkl);
```

EtherCATマスタ機能モジュールで発生中のコントローラ異常を解除します。



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_ErrSta	EtherCAT 異常	WORD	EtherCAT マスタ機能モジュールの異常の集約。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

使用上の注意

- 本命令の実行直後に、異常が解除されるとは限りません。異常が解除されたかどうかは、GetECError 命令で確認してください。

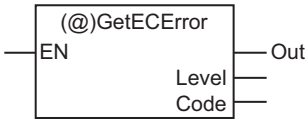
- オムロン製サーボドライバ G5 シリーズに対して本命令を使用する場合、ResetMCError 命令、MC_Reset 命令、MC_GroupReset 命令のいずれかが同時に実行されないように、命令の排他制御を行ってください。本命令と、これら 3 つの命令のいずれかを同時に実行すると、G5 シリーズがそれ以降の SDO を受け付けられなくなることがあります。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) EtherCAT マスタ機能モジュールで発生中のコントローラ異常解除処理中に、さらに本命令が実行された場合。
  b) EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令のいずれかが実行中の場合。

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.2-828)」を参照してください。

GetECError

EtherCAT マスタ機能モジュールで発生中の異常を検出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetECError	EtherCAT 異常ステータス取得	FUN		Out:=GetECError(Level, Code);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	異常有無	出力	TRUE: 発生中の異常あり FALSE: 発生中の異常なし	データ型に従う	–	–
Level	最重要ステータス		発生中の異常のうち、最重要のステータス 0: 異常なし 2: 部分停止フォールトレベル 3: 軽度フォールトレベル	0, 2, 3		
Code	最重要イベントコード		発生中の異常のうち、最重要のイベントコード	16#00000000, 16#00070000～ 16#FFFFFFFF		

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Level							○													
Code				○																

機能

EtherCAT マスタ機能モジュールで発生中の異常を検出します。
「Out」の値は、異常ありの場合に TRUE、異常なしの場合に FALSE になります。
「Level」は、発生中の異常のうち、最重要のステータスを示します。
「Code」は、発生中の異常のうち、最重要のイベントコードを示します。

検出する異常の種類と出力変数の値

本命令で検出できる異常の種類は、通信ポート異常、マスタ異常、スレーブ異常です。

EtherCAT マスタ機能モジュールの状態、および各出力変数の値との関係は、以下のとおりです。

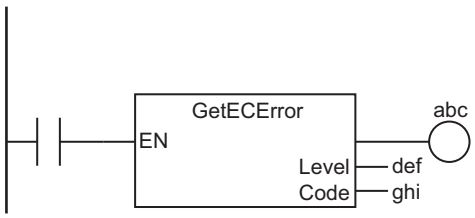
EtherCAT マスタ機能モジュールの状態	「Out」の値	「Level」の値	「Code」の値
異常なし	FALSE	0	16#0000_0000
通信ポート異常またはマスタ異常あり	TRUE	発生中の異常のうち最重要のステータス 2: 部分停止フォールトレベル 3: 軽度フォールトレベル	発生中の異常のうち最重要のイベントコード *1 16#0007_0000 ～ 16#FFFF_FFFF
スレーブ異常あり			16#0000_0000

*1. 最重要のイベント重要度の異常が複数発生している場合、最も早く発生した異常のイベントコードになります。

記述例

記述例を、以下に示します。

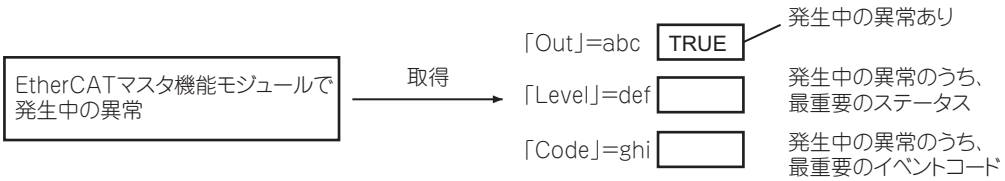
LD



ST

```
abc:=GetECError(def, ghi);
```

EtherCATマスタ機能モジュールで発生中の通信ポート異常、マスタ異常、スレーブ異常を検出します。



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_ErrSta	EtherCAT 異常	WORD	EtherCAT マスタ機能モジュールの異常の集約。*1
_EC_PortErr *2	通信ポート異常	WORD	EtherCAT マスタ用通信ポートの異常の集約。*1
_EC_MstrErr *2	マスタ異常	WORD	EtherCAT マスタの異常と、EtherCAT マスタが検知したスレーブの異常の集約。*1
_EC_SlavErr	スレーブ異常	WORD	EtherCAT スレーブ全体の異常ステータスの集約。*1

*1. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート (SBCD-368)』を参照してください。
*2. 本命令が取得するのは、_EC_PortErr（通信ポート異常）と_EC_MstrErr（マスタ異常）に該当する異常です。

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.2-828)」を参照してください。

SetInfo

ユーザ情報を発生させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SetInfo	ユーザ情報発生	FUN	(@)SetInfo EN ENO Code Out Info1 Info2	SetInfo(Code, Info1, Info2);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Code	イベントコード	入力	発生させるユーザ情報のイベントコード	40001 ～ 60000	−	40001
Info1	付属情報 1		ユーザ情報発生とともに、イベントログに記録する値	データ型に従う		*1
Info2	付属情報 2					
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

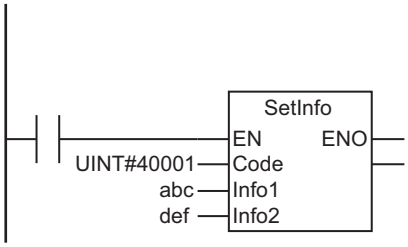
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							○													
Info1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Info2	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	
Out	○																			

機能

イベントコード「Code」で指定したユーザ情報を、発生させます。
また、イベントコードの重要度に対応するユーザイベントログ領域に、発生時刻、イベントコード「Code」、イベント重要度、付属情報「Info1」、「Info2」の値を記録します。

記述例を、以下に示します。イベントコード 40001 に対応するユーザ情報を、発生させます。付属情報として、変数「abc」、「def」の値を記録します。

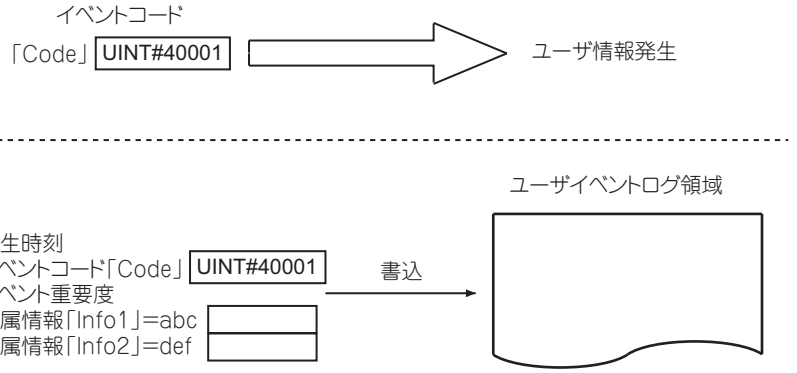
LD



ST

```
SetInfo(UINT#40001, abc, def);
```

イベントコード「Code」で指定したユーザ情報を発生させます。
また、イベントコードの重要度に対するユーザイベントログ領域に、発生時刻、
イベントコード「Code」、イベント重要度、付属情報「Info1」、「Info2」の値を記録します。

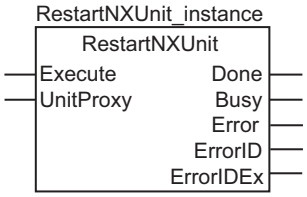


使用上の注意

- 「Info1」、「Info2」に渡す入力パラメータは、必ず変数を指定してください。付属情報を使用しない場合でも、ダミーの変数を指定してください。定数を指定するとビルド時に異常になります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となります。
  a)「Code」の値が、有効範囲外の場合。

RestartNXUnit

EtherCAT カプラユニット、または NX ユニットをリスタートします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
RestartNXUnit	NX ユニットリスタート	FB		RestartNXUnit_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx);

バージョン関連情報

以下は、バージョンによって「指定した NX ユニットをリスタートする機能」に対応していないものがあります。
- EtherCAT カプラユニット
- NX ユニット

「指定した NX ユニットをリスタートする機能」に対応していないバージョンを使用する場合は、リスタートするユニットに EtherCAT カプラユニットだけ指定することが可能です。
「指定した NX ユニットをリスタートする機能」に対応しているバージョンは、『NX シリーズ EtherCAT カプラユニットユーザーズマニュアル(SBCD-361)』を参照してください。

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	指定ユニット	入力	リスタートする EtherCAT カプラユニット、NX バス機能モジュールまたは NX ユニット	－	－	*1

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID 詳細は「機能 (P.2-840)」参照																			

機能

EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットをリスタートします。
指定したユニットを単独でリスタートすることができます。
ただし、EtherCAT カプラユニットだけ、単独でリスタートすることはできません。
EtherCAT カプラユニットを指定した場合は、それに接続されている NX ユニットもすべてリスタートします。
リスタートするユニットは、「UnitProxy」で指定します。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

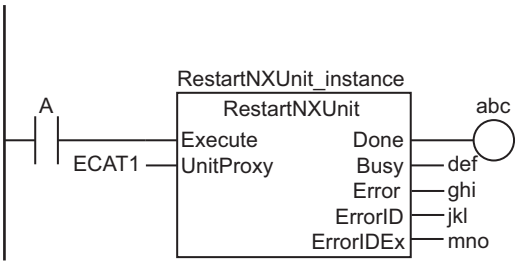
変数		名称	内容	データ型
UnitProxy		指定ユニット	指定するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定するユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定する EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに割り付けられたデバイス変数を渡してください。

記述例

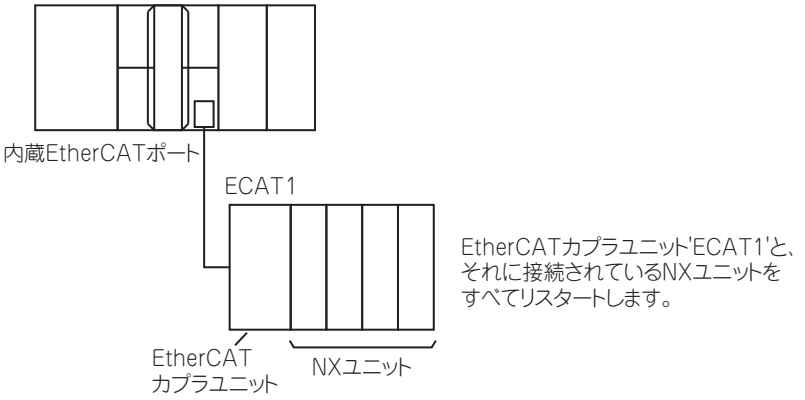
EtherCAT スレーブターミナルをすべてリスタートする場合の例を、以下に示します。EtherCAT カプラユニットには、_sNXUNIT_ID 型の「ECAT1」という変数が割り付けられているとします。

LD



ST

```
RestartNXUnit_instance(A, ECAT1,
                       abc, def, ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可
_NXB_UnitMsgActiveTbl[i]	NX ユニットメッセージ通信可能ステータス	BOOL	メッセージ通信が可能なスレーブの一覧です。該当スレーブと、通信が可能かどうかを確認する場合に使用します。

参考

EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに、以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性あり
- 値が反映されるのはユニットのリスタート時

1 **「NX_ChangeWriteMode (P.2-845)」命令を使って、対象ユニットを、データの書込ができるモードに変更します。**

2 **「NX_WriteObj (P.2-953)」命令を使って、対象ユニットにデータを書き込みます。**

3 **「NX_SaveParam (P.2-850)」命令を使って、書き込んだデータを保存します。**

4 **本命令を使って、対象ユニットをリスタートさせます。**

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「UnitProxy」に、モーション制御の軸（データ型_sAXIS_REF）に割り付けたユニットを指定すると、モーション制御モジュールのコントローラ異常が発生します。「ResetMCError (P.2-827)」命令で、コントローラ異常を解除してください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニット、または EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 本命令または「NX_ChangeWriteMode (P.2-845)」命令が実行中に本命令を実行しようとした場合、後から実行しようとした本命令はキューイングされます。キューイングの最大数は 192 です。192 を超えると異常になります。また、キューイングされている時間は、タイムアウトの対象外です。
- 本命令がキューイングされている間、「Busy」の値は TRUE となります。
- 本命令は、NX メッセージ通信異常に関連する命令です。NX メッセージ通信異常に関連する命令を一度に多数実行すると、NX メッセージ通信異常が発生する可能性があります。NX メッセージ通信異常に関連する命令については、「A-3 NX メッセージ通信異常に関連する命令 (P.A-22)」を参照してください。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。実行すると、異常が発生します。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0419	16#00000000	「UnitProxy」のデータ型が、不正です。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。

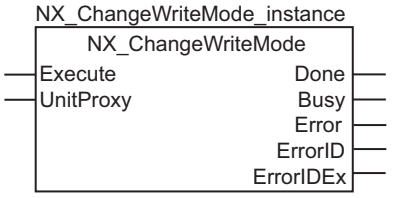
「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。
16#2C01	16#00000000	本命令または NX_ChangeWriteMode 命令が、192 を超えてキューイングされました。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。
16#2C05	－	EtherCAT ネットワーク上で、異常が発生しました。 「UnitProxy」の値や、EtherCAT ネットワーク構成を確認してください。
16#2C06	16#00000000	指定したユニットは、Sysmac Studio からの指示で、すでにリスタートが実行されています。したがって、本命令の実行は不要です。
16#2C07	16#00000000	指定したユニットのスレーブノードアドレスに、本命令の対象外のスレーブが接続されています。「UnitProxy」の値や、EtherCAT ネットワーク構成を確認してください。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.2-958)」を、参照してください。

NX_ChangeWriteMode

EtherCAT カプラユニット、または NX ユニットを、データの書込ができるモードに変更します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ChangeWriteMode	NX ユニット書込モード変更	FB		NX_ChangeWriteMode_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	指定ユニット	入力	モード変更するユニット	–	–	*1

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID　詳細は「機能 (P.2-845)」参照																			

機能

EtherCAT カプラユニット、または EtherCAT カプラユニット上の NX ユニットを、データの書き込みができるモードに変更します。
モード変更するユニットを、「UnitProxy」で指定します。
「Done」の値が TRUE になったら、データの書込ができる状態になっています。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

変数		名称	内容	データ型
UnitProxy		指定ユニット	書込モードを変更するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定するユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定するユニットに割り付けたデバイス変数を渡してください。

関連命令との実行手順

本命令は、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに、以下の属性をもつデータを書き込む場合に使用します。

- 電源断保持属性あり
- 値が反映されるのはユニットのリスタート時

関連命令との実行手順は、以下のとおりです。

1 **本命令を使って、対象ユニットを、データの書込ができるモードに変更します。**

2 **「NX_WriteObj (P.2-953)」命令を使って、対象ユニットにデータを書き込みます。**

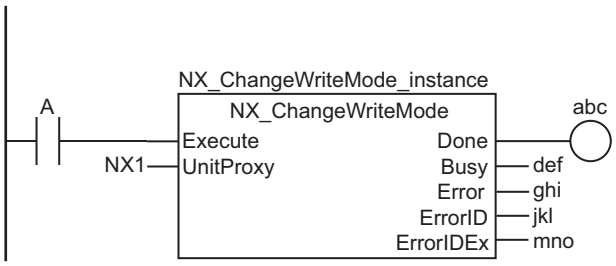
3 **「NX_SaveParam (P.2-850)」命令を使って、書き込んだデータを保存します。**

4 **「RestartNXUnit (P.2-840)」命令を使って、対象ユニットをリスタートさせます。**

記述例

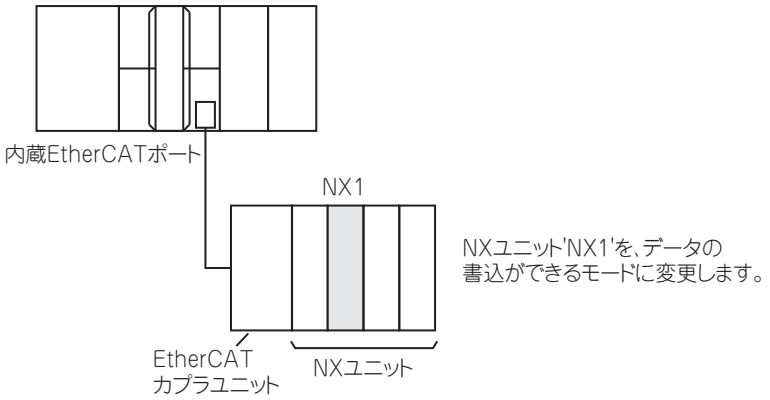
NX ユニット「NX1」を、データの書込ができるモードに変更する場合の例を、以下に示します。対象となる NX ユニットには、_sNXUNIT_ID 型の「NX1」という変数が割り付けられているとします。

LD



ST

```
NX_ChangeWriteMode_instance(A, NX1, abc, def,
                            ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可
_NXB_UnitMsgActiveTbl[i]	NX ユニットメッセージ通信可能ステータス	BOOL	メッセージ通信が可能なスレーブの一覧です。該当スレーブと、通信が可能かどうかを確認する場合に使用します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「UnitProxy」に、モーション制御の軸（データ型_sAXIS_REF）に割り付けたユニットを指定すると、モーション制御モジュールのコントローラ異常が発生します。その場合は、「ResetMCError (P.2-827)」命令で、コントローラ異常を解除してください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 本命令または「RestartNXUnit (P.2-840)」命令が実行中に本命令を実行しようとした場合、後から実行しようとした本命令はキューイングされます。キューイングの最大数は 192 です。192 を超えると異常になります。また、キューイングされている時間は、タイムアウトの対象外です。
- 本命令がキューイングされている間、「Busy」の値は TRUE となります。
- 本命令は、NX メッセージ通信異常に関連する命令です。NX メッセージ通信異常に関連する命令を一度に多数実行すると、NX メッセージ通信異常が発生する可能性があります。NX メッセージ通信異常に関連する命令については、「A-3 NX メッセージ通信異常に関連する命令 (P.A-22)」を参照してください。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。実行すると、異常が発生します。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0419	16#00000000	「UnitProxy」のデータ型が、不正です。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。

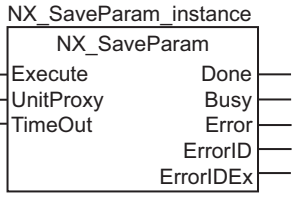
「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#2C01	16#00000000	本命令または RestartNXUnit 命令が、192 を超えてキューイングされました。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。
16#2C05	−	EtherCAT ネットワーク上で、異常が発生しました。 「UnitProxy」の値や、EtherCAT ネットワーク構成を確認してください。
16#2C07	16#00000000	指定したユニットのスレーブノードアドレスに、本命令の対象外のスレーブが接続されています。 「UnitProxy」の値や、EtherCAT ネットワーク構成を確認してください。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.2-958)」を、参照してください。

NX_SaveParam

EtherCAT カプラユニット、または NX ユニットに書き込んだデータを、保存します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SaveParam	NX ユニットパラメータ保存	FB		NX_SaveParam_instance(Execute, UnitProxy, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	指定ユニット	入力	データを保存するユニット	–	–	*1
TimeOut	タイムアウト時間		タイムアウト時間 0 の場合は、2.0s	0 ～ 60000	ms	2000 (2.0s)

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、Line break 日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID　詳細は「機能 (P.2-850)」参照																			
TimeOut							○													

機能

EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに書き込んだデータを、保存します。
NX ユニットのデータは、EtherCAT カプラユニットにデータを保存するものと、NX ユニット自体にデータを保存するものがあります。

データの保存先にかかわらず、データを保存させる EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットを、「UnitProxy」で指定します。

「Done」の値が TRUE になったら、データの保存が完了しています。
データを書き込む命令は、「NX_WriteObj (P.2-953)」命令です。
本命令実行後は、電源断が発生しても、電源断保持属性ありのデータの値は保持されます。

タイムアウト時間は、「TimeOut」で指定します。タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。その場合、ユニットのデータ保存は行われません。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

変数		名称	内容	データ型
UnitProxy		指定ユニット	データを保存するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定するユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定するユニットに割り付けられたデバイス変数を渡してください。

関連命令との実行手順

本命令は、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに書き込むデータの属性によって、他の関連命令と連携して実行する必要があります。

それぞれの場合の実行手順は、以下のとおりです。

● 実行手順 1

以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性あり
- 値が反映されるのはユニットのリスタート時

1 **「NX_ChangeWriteMode (P.2-845)」命令を使って、対象ユニットをデータの書込ができるモードに変更します。**

2 **「NX_WriteObj (P.2-953)」命令を使って、対象ユニットにデータを書き込みます。**

3 **本命令を使って、書き込んだデータを保存します。**

4 **「RestartNXUnit (P.2-840)」命令を使って、対象ユニットをリスタートさせます。**

● 実行手順 2

以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性あり
- 値が反映されるのは書込直後

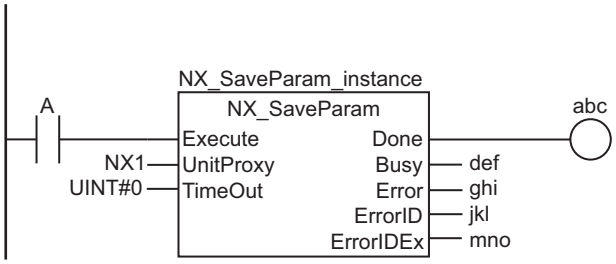
1 **「NX_WriteObj (P.2-953)」命令を使って、対象ユニットにデータを書き込みます。**

2 **本命令を使って、書き込んだデータを保存します。**

記述例

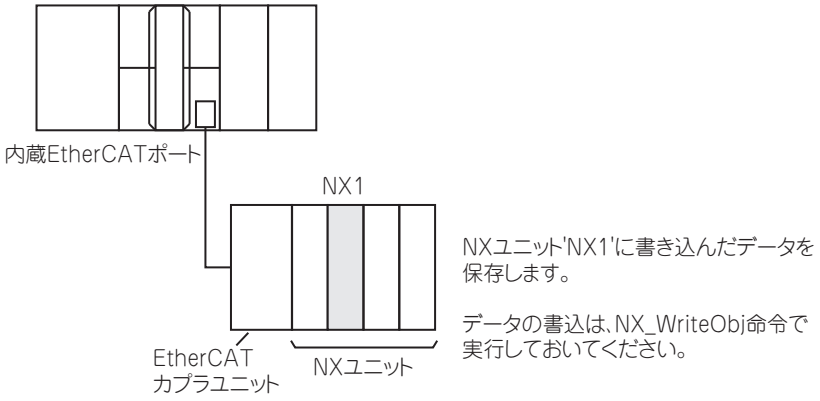
NX ユニット「NX1」に書き込んだデータを保存する場合の例を、以下に示します。対象となる NX ユニットには、_sNXUNIT_ID 型の「NX1」という変数が割り付けられているとします。

LD



ST

```
NX_SaveParam_instance(A, NX1, UINT#0,
                      abc, def, ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「UnitProxy」で指定したユニットがデータ保存中の場合、本命令は異常にはなりません。そのまま「Busy」の値が TRUE になり、データ保存完了時に「Done」の値が TRUE になります。データ保存の要求は、キューイングされません。
- ユニットにデータ書込を行わずに本命令を実行しても、異常にはなりません。
- ユニットには、データの書込可能回数が制限されているものがあります。詳細は、各ユニットのマニュアルを参照してください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 電源断時の保持属性ありのデータを書き込み、保持させたい場合は、「NX_WriteObj (P.2-953)」命令の実行後に本命令を実行してください。本命令を実行せずに対象ユニットがリスタートされると、NX オブジェクトのデータは書込前の値に戻ります。

- 本命令は、NX メッセージ通信異常に関連する命令です。NX メッセージ通信異常に関連する命令を一度に多数実行すると、NX メッセージ通信異常が発生する可能性があります。NX メッセージ通信異常に関連する命令については、「A-3 NX メッセージ通信異常に関連する命令 (P.A-22)」を参照してください。
- 本命令実行中（「Busy」の値が TRUE）にコントローラの電源を OFF すると、次回電源投入時に全停止フォールトが発生する場合があります。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0400	16#00000000	• 「UnitProxy」の値が、有効範囲外です。 • 「TimeOut」の値が、有効範囲外です。
16#0419	16#00000000	「UnitProxy」のデータ型が、不正です。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。
16#2C01	16#00000000	同時に実行できる命令数を超えています。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.2-958)」を、参照してください。

NX_ReadTotalPowerOnTime

通信カプラユニット、および NX ユニットが保有する累積通電時間を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	FB	NX_ReadTotalPowerOnTime_instance NX_ReadTotalPowerOnTime Execute / UnitProxy Done / Busy / Error / ErrorID / ErrorIDEx / TotalPowerOnTime	NX_ReadTotalPowerONTime_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx, TotalPowerOnTime);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	ユニット指定	入力	対象とする NX ユニットを指定します。	–	–	*1
TotalPowerOnTime	累積通電時間	出力	読み出した累積通電時間を出力します。	データ型に従う		0

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID　詳細は「機能 (P.2-855)」参照																			
TotalPowerOnTime																○				

機能

通信カプラユニットまたは通信カプラユニット上の NX ユニットの累積通電時間を目安として読み出します。
精度は 1 時間／月です。

累積通電時間を読み出すユニットを、「UnitProxy」で指定します。
「Done」の値が TRUE になると、累積通電時間が読み出されています。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

変数		名称	内容	データ型
UnitProxy		指定ユニット	指定するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定する NX ユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定する通信カプラユニットまたは通信カプラユニット上の NX ユニットに割り付けられたデバイス変数を渡してください。

バージョンによる組み合わせ

CPU ユニットに接続された通信カプラユニットや通信カプラユニット上の NX ユニットのバージョンにより、累積通電時間を読み出し可能な組み合わせがあります。

● EtherCAT スレーブターミナルの場合

ユニット	NX ユニットのバージョン	EtherCAT カプラユニットのバージョン
デジタル I/O ユニット	Ver.1.0 以降	Ver.1.2 以降
アナログ I/O ユニット		
システムユニット		
位置インタフェースユニット	Ver.1.1 以降	
温度入力ユニット		

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可
_NXB_UnitMsgActiveTbl[i]	NX ユニットメッセージ通信可能ステータス	BOOL	メッセージ通信が可能なスレーブの一覧です。該当スレーブと、通信が可能かどうかを確認する場合に使用します。

参考

本命令をシミュレータで動作させた場合、「Execute」が FALSE から TRUE に変化したときに、1 タスク周期の間だけ、「Busy」が TRUE となります。
次のタスク周期では、「Busy」は FALSE となり、「Done」は TRUE となります。
なお、読み出される「TotalPowerOnTime」は「0」になります。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 通信カプラユニットにより、台数制限が異なる場合があります。詳細は、通信カプラユニットのマニュアルを参照してください。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0400	16#00000000	「UnitProxy」の値が、有効範囲外です。
16#0419	16#00000000	「UnitProxy」のデータ型が、不正です。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。
16#2C01	16#00000000	同時に実行できる命令数を超えています。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。
16#2C08	16#00000000	通電時間の読み出しができませんでした。

サンプルプログラム

プログラム上でメンテナンスモードと RUN モードの 2 つを作ります。
このうち、メンテナンスモード時に累積時間読み出しボタンを押すと、あらかじめ決めているユニット 3 の累積通電時間を読み出します。
累積通電時間が 5 年を超えている場合、ユニット交換お知らせランプが点灯します。
ユニット交換後、ユニット交換済みボタンを押すと、ユニット交換お知らせランプは消灯します。

次のようなシステム構成とします。

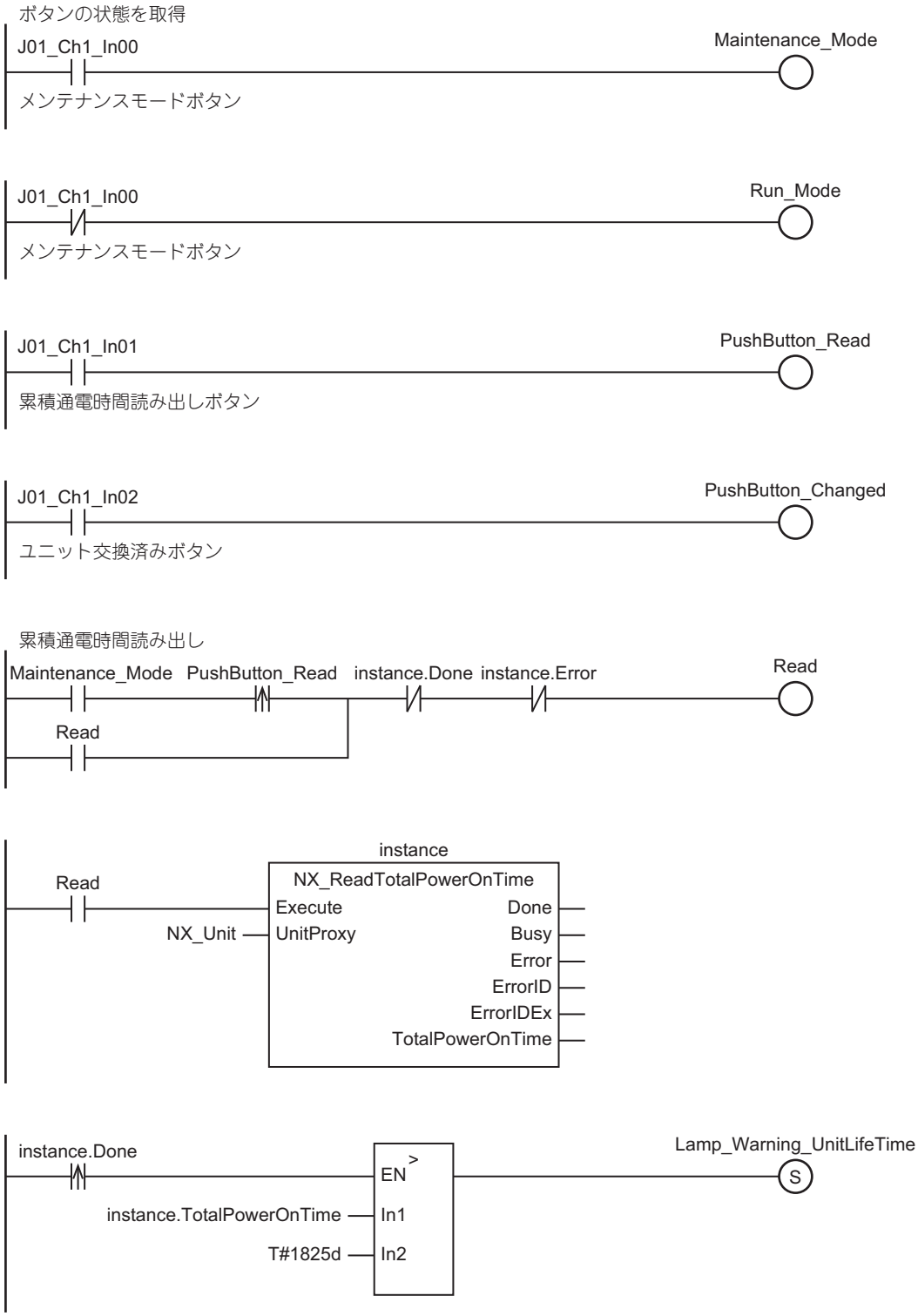
ユニット	内容
ユニット 1	NX ユニット (ID)
ユニット 2	NX ユニット (OD)
ユニット 3	NX ユニット （累積通電時間を読み出す対象）

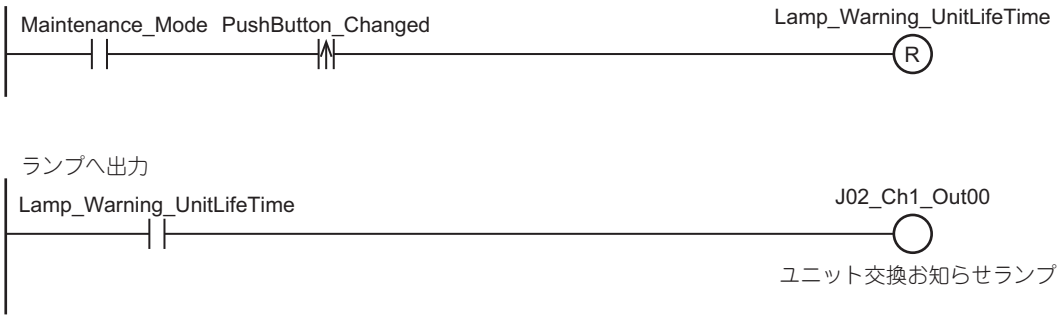
LD

内部変数	名称	データ型	初期値	コメント
	Maintenance_Mode	BOOL	FALSE	メンテナンスモード
	Run_Mode	BOOL	FALSE	RUN モード
	PushButton_Read	BOOL	FALSE	累積時間読み出し
	PushButton_Changed	BOOL	FALSE	ユニット交換済み
	Lamp_Warning_UnitLifeTime	BOOL	FALSE	ユニット交換お知らせ
	Read	BOOL	FALSE	

内部変数	名称	データ型	初期値	コメント
	instance	NX_ReadTotalPowerOnTime		

外部変数	名称	データ型	コメント
	NX_Unit	_sNXUNIT_ID	
	J01_Ch1_In00	BOOL	メンテナンスモードボタン
	J01_Ch1_In01	BOOL	累積通電時間読み出しボタン
	J01_Ch1_In02	BOOL	ユニット交換済みボタン
	J02_Ch1_Out00	BOOL	ユニット交換お知らせランプ





ST

内部変数	名称	データ型	初期値	コメント
	Maintenance_Mode	BOOL	FALSE	メンテナンスモード
	Run_Mode	BOOL	FALSE	RUN モード
	PushButton_Read	BOOL	FALSE	累積時間読み出し
	PushButton_Changed	BOOL	FALSE	ユニット交換済み
	Lamp_Warning_UnitLifeTime	BOOL	FALSE	ユニット交換お知らせ
	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		

外部変数	名称	データ型	コメント
	NX_Unit	_sNXUNIT_ID	
	J01_Ch1_In00	BOOL	メンテナンスモードボタン
	J01_Ch1_In01	BOOL	累積通電時間読み出しボタン
	J01_Ch1_In02	BOOL	ユニット交換済みボタン
	J02_Ch1_Out00	BOOL	ユニット交換お知らせランプ

```
// ボタンの状態を取得
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);

// 累計通電時間読み出し
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);

// ランプへ出力
JO2_Ch1_Out00 := Lamp_Warning_UnitLifeTime;
```

プログラム制御命令

命令	名称	ページ
PrgStart	プログラム実行指令	P.2-864
PrgStop	プログラム停止指令	P.2-872
PrgStatus	プログラム状態読出	P.2-889

PrgStart

指定したプログラムの実行を指令します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PrgStart	プログラム実行指令	FUN	(@)PrgStart EN PrgName isFirstRun Out	Out:=PrgStart(PrgName, isFirstRun);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
PrgName	プログラム名	入力	指定するプログラム名	最大 128 バイト（半角英数字 127 文字 + 終端 NULL 文字）	–	*1
isFirstRun	プログラム起動時 1 周期 ON フラグ有効		プログラム実行指定の場合の、システム定義変数「P_First_Run」の、最初の 1 タスク周期の動作 TRUE: TRUE にする FALSE: FALSE にする	データ型に従う		TRUE
Out	正常終了フラグ	出力	正常終了フラグ TRUE: 正常終了 FALSE: 異常終了	データ型に従う	–	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				○
isFirstRun	○																			
Out	○																			

機能

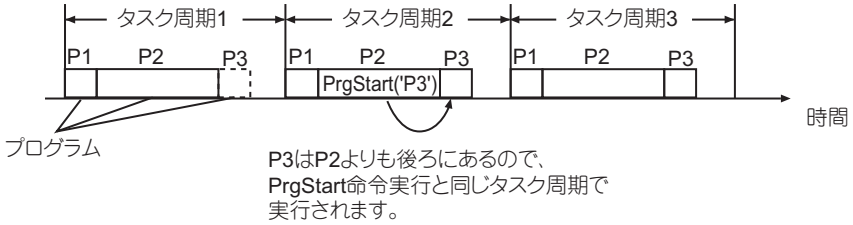
プログラム名「PrgName」で指定したプログラムの、実行を指令します。
指定されたプログラムは、つぎにプログラムを実行するタイミングで実行されます。指定されたプログラムがすでに実行中の場合、異常にはなりません。
指定するプログラムは、本命令を実行するタスクと同じタスク内にあっても、別タスク内にあってもかまいません。
正常終了フラグ「Out」の値は、本命令が正常終了すれば TRUE となり、異常終了すれば FALSE となります。

自タスクのプログラムを指定する場合の動作例

本命令を実行するタスクと同じタスク内のプログラムを指定する場合の動作例を、示します。

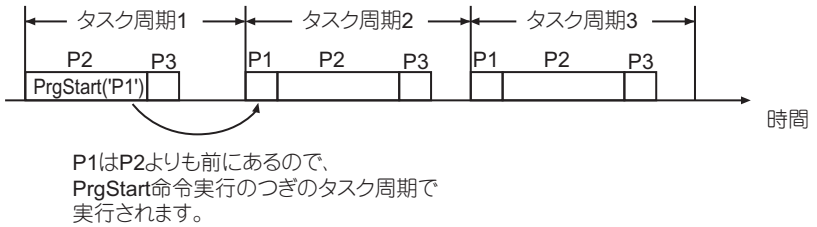
● PrgStart 命令よりも後ろにあるプログラムの実行を指令する場合

- 同じタスク内に、P1、P2 および P3 の 3 つのプログラムがあります。
- タスク周期 1 で、P3 は停止しています。
- タスク周期 2 の P2 内で、P3 を指定して PrgStart 命令を実行します。
- P3 は P2 よりも後ろにあるので、タスク周期 2 で P3 が実行されます。
- その後は、あらためて P3 を指定して PrgStart 命令を実行しなくても、P3 が実行されます。



● PrgStart 命令よりも前にあるプログラムの実行を指令する場合

- 同じタスク内に、P1、P2 および P3 の 3 つのプログラムがあります。
- タスク周期 1 で、P1 は停止しています。
- タスク周期 1 の P2 内で、P1 を指定して PrgStart 命令を実行します。
- P1 は P2 よりも前にあるので、タスク周期 2 で P1 が実行されます。
- その後は、あらためて P1 を指定して PrgStart 命令を実行しなくても、P1 が実行されます。

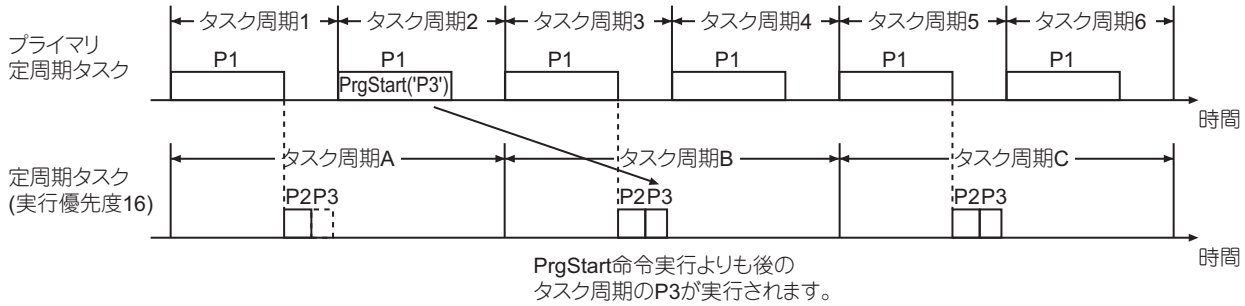


別タスクのプログラムを指定する場合の動作例

本命令を実行するタスクとは別のタスク内のプログラムを指定する場合の動作例を、示します。

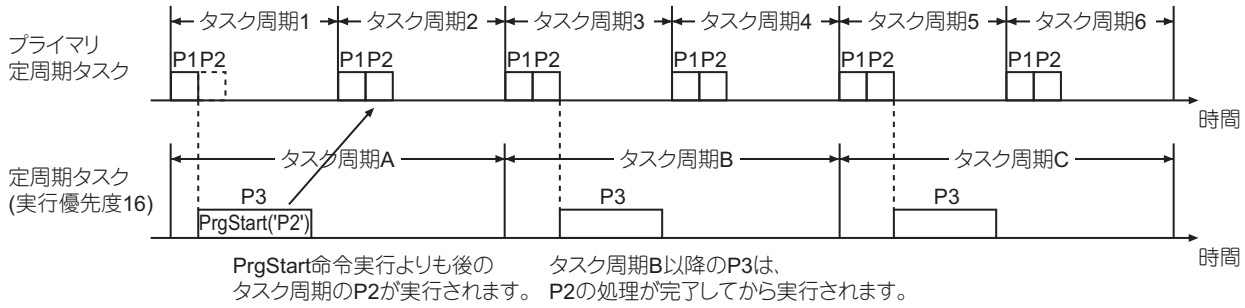
● 自タスクよりも実行優先度の低いタスク内の、プログラムの実行を指令する場合

- プライマリ定周期タスクに P1、定周期タスクに P2、P3 の 3 つのプログラムがあります。
- 定周期タスクのタスク周期 A で、P3 は停止しています。
- プライマリ定周期タスクのタスク周期 2 の P1 内で、P3 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、定周期タスクのタスク周期 B の P3 が実行されます。
- その後は、あらためて P3 を指定して PrgStart 命令を実行しなくても、P3 が実行されます。



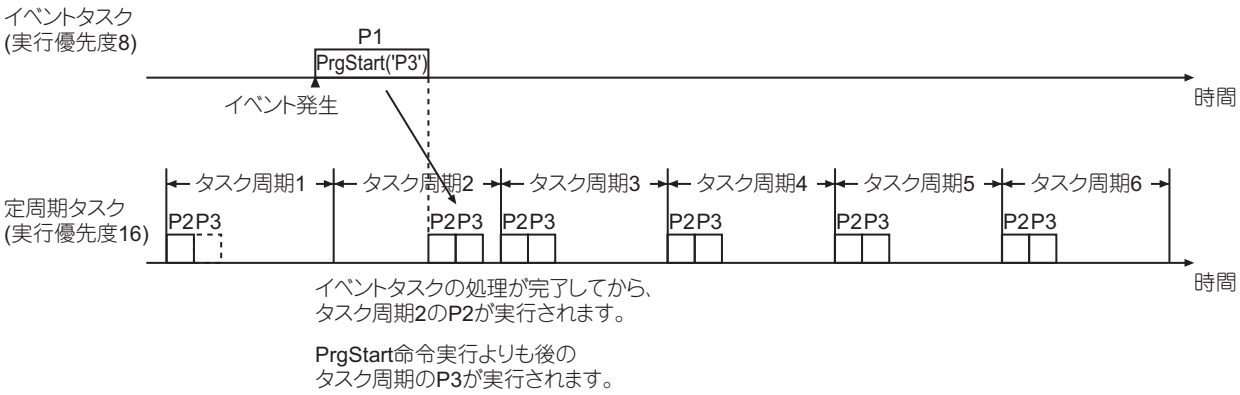
● 自タスクよりも実行優先度の高いタスク内の、プログラムの実行を指令する場合

- プライマリ定周期タスクに P1、P2、定周期タスクに P3 の 3 つのプログラムがあります。
- プライマリ定周期タスクのタスク周期 1 で、P2 は停止しています。
- 定周期タスクのタスク周期 A の P3 内で、P2 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、プライマリ定周期タスクのタスク周期 2 の P2 が実行されます。
- その後は、あらためて P2 を指定して PrgStart 命令を実行しなくても、P2 が実行されます。
- プライマリ定周期タスクは定周期タスクより実行優先度が高いので、タスク周期 B 以降の P3 は、P2 の処理が完了した後に実行されます。



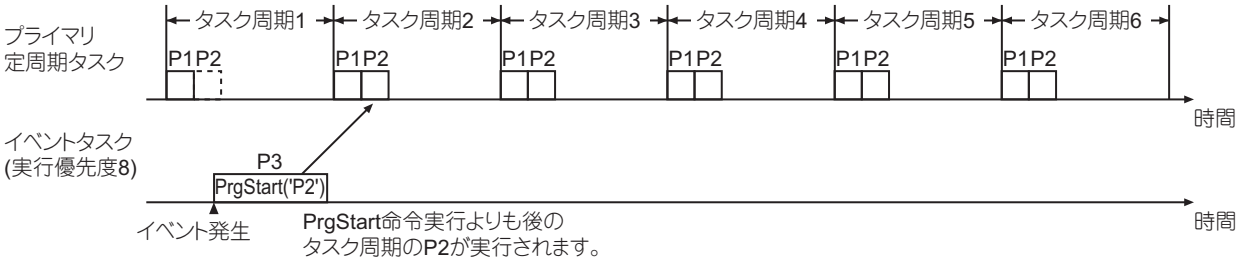
● イベントタスクから、実行優先度が低いタスク内のプログラムの実行を指令する場合

- イベントタスク（実行優先度 8）に P1、定周期タスク（実行優先度 16）に P2、P3 の 3 つのプログラムがあります。
- 定周期タスクのタスク周期 1 で、P3 は停止しています。
- イベントタスク 内で、P3 を指定して PrgStart 命令を実行します。
- イベントタスクが実行されたことにより、定周期タスクのタスク周期 2 の P2,P3 は、イベントタスクの処理が完了してから実行されます。
- その結果、定周期タスクのタスク周期 2 の P3 は、PrgStart 命令実行の後になるため、実行されます。
- その後は、あらためて P3 を指定して PrgStart 命令を実行しなくても、P3 が実行されます。



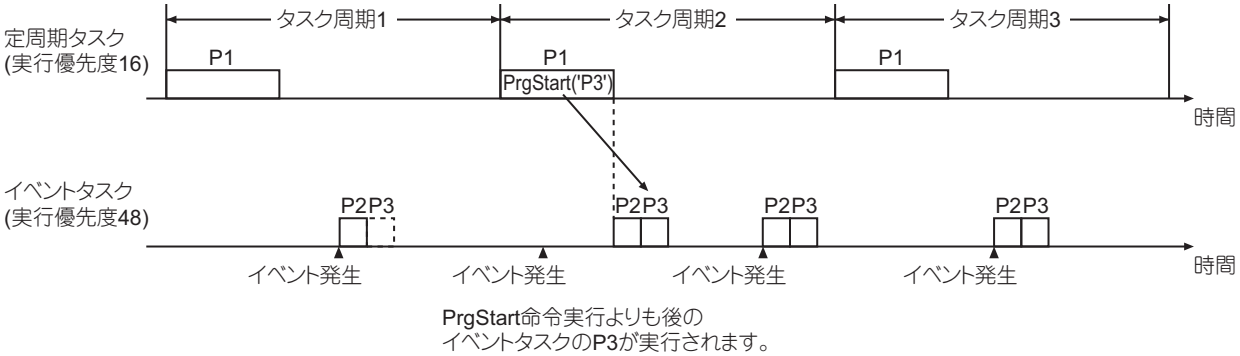
● イベントタスクから、実行優先度が高いタスク内のプログラムの実行を指令する場合

- プライマリ定周期タスクに P1、P2、イベントタスクに P3 の 3 つのプログラムがあります。
- プライマリ定周期タスクのタスク周期 1 で、P2 は停止しています。
- イベントタスク 内で、P2 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、プライマリ定周期タスクのタスク周期 2 の P2 が実行されます。
- その後は、あらためて P2 を指定して PrgStart 命令を実行しなくても、P2 が実行されます。



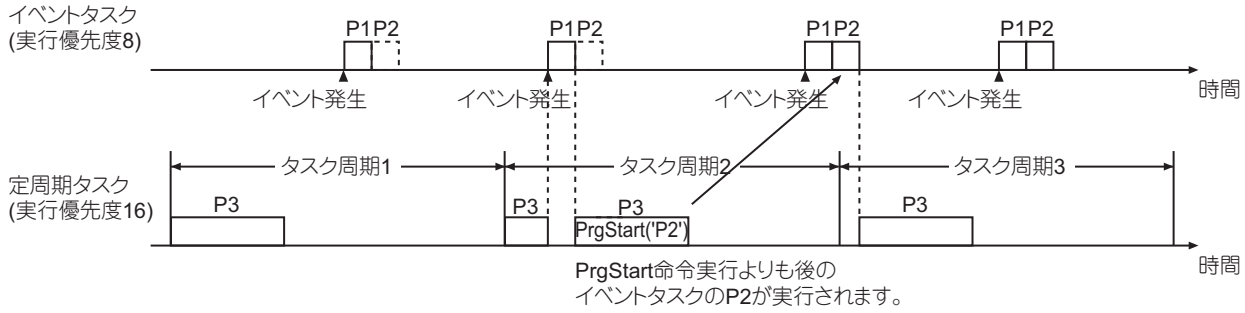
● 定周期タスクから、優先度の低いイベントタスク内のプログラムの実行を指令する場合

- 定周期タスク（実行優先度 16）に P1、イベントタスク（実行優先度 48）に P2、P3 の 3 つのプログラムがあります。
- イベントタスクの P3 は、停止しています。
- 定周期タスク 内で、P3 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、イベントタスクの P3 が実行されます。
- その後は、あらためて P3 を指定して PrgStart 命令を実行しなくても、P3 が実行されます。



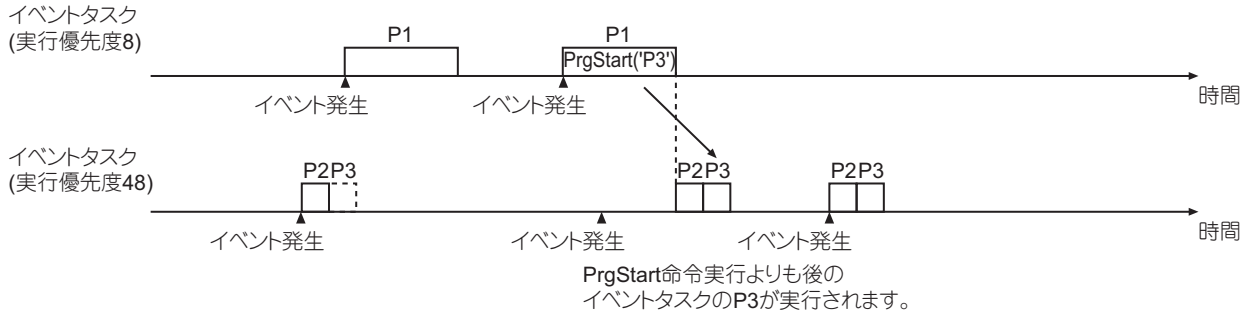
● 定周期タスクから、優先度の高いイベントタスク内のプログラムの実行を指令する場合

- イベントタスク（実行優先度 8）に P1、P2、定周期タスク（実行優先度 16）に P3 の 3 つのプログラムがあります。
- イベントタスクの P2 は、停止しています。
- 定周期タスク 内で、P2 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、イベントタスクの P2 が実行されます。
- その後は、あらためて P2 を指定して PrgStart 命令を実行しなくても、P2 が実行されます。



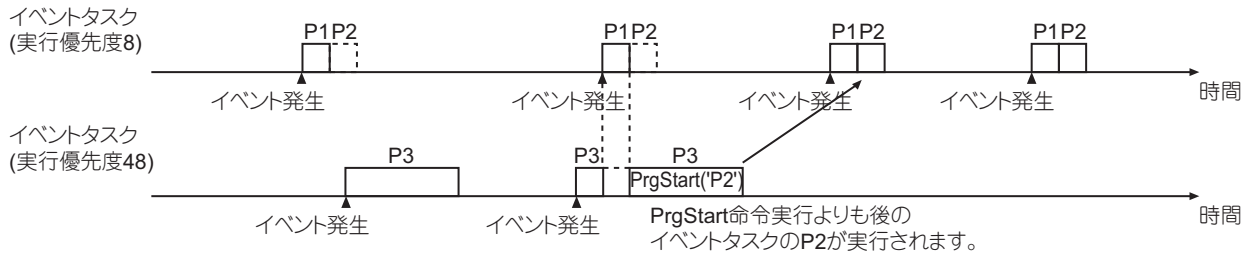
● イベントタスクから、優先度の低いイベントタスク内のプログラムの実行を指令する場合

- イベントタスク（実行優先度 8）に P1、イベントタスク（実行優先度 48）に P2、P3 の 3 つのプログラムがあります。
- イベントタスク（実行優先度 48）の P3 は、停止しています。
- イベントタスク（実行優先度 8） 内で、P3 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、イベントタスク（実行優先度 48）の P3 が実行されます。
- その後は、あらためて P3 を指定して PrgStart 命令を実行しなくても、P3 が実行されます。



● イベントタスクから、優先度の高いイベントタスク内のプログラムの実行を指令する場合

- イベントタスク（実行優先度 8）に P1、P2、イベントタスク（実行優先度 48）に P3 の 3 つのプログラムがあります。
- イベントタスク（実行優先度 8）の P2 は、停止しています。
- イベントタスク（実行優先度 48） 内で、P2 を指定して PrgStart 命令を実行します。
- PrgStart 命令の実行よりも後に実行される、イベントタスク（実行優先度 8）の P2 が実行されます。
- その後は、あらためて P2 を指定して PrgStart 命令を実行しなくても、P2 が実行されます。



プログラム起動時 1 周期 ON フラグ有効「isFirstRun」

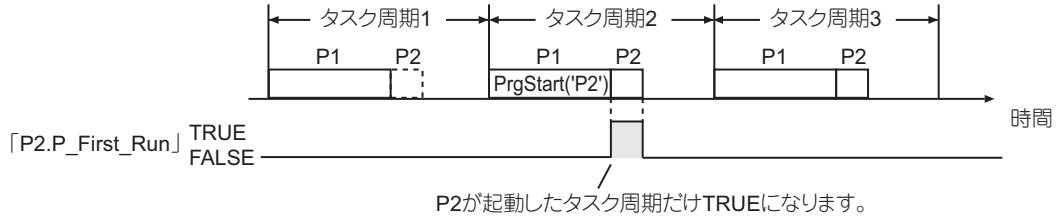
プログラム起動時 1 周期 ON フラグ有効「isFirstRun」は、プログラム起動時 1 周期 ON フラグのシステム定義変数「P_First_Run」を有効にするかどうかを指定する変数です。
「isFirstRun」の値を TRUE にして本命令を実行すると、「P_First_Run」は、当該プログラムが起動したときに、1 タスク周期の間だけ値が TRUE になります。
「isFirstRun」の値が FALSE の場合、当該プログラムが起動しても、「P_First_Run」の値は、FALSE のまま変化しません。
「isFirstRun」は、ある条件が成立した場合にだけ、当該プログラムが起動したときに特定の処理を行う場合に使用します。
条件が成立した場合には、「isFirstRun」の値を TRUE にして本命令を実行します。
当該プログラムは、「P_First_Run」の値が TRUE の場合に、特定の処理を行うアルゴリズムにしておきます。

「isFirstRun」と「P_First_Run」の関係は、以下のとおりです。
当該プログラムが停止しているかすでに起動しているかで、「P_First_Run」の動作が変わります。

「isFirstRun」の値	当該プログラムの状態	「P_First_Run」の値
TRUE	停止している	プログラムの起動後、1 タスク周期の間、TRUE になります。そのつぎのタスク周期からは、FALSE になります。
	すでに起動している	FALSE のまま、変化しません。
FALSE	ー	FALSE のまま、変化しません。

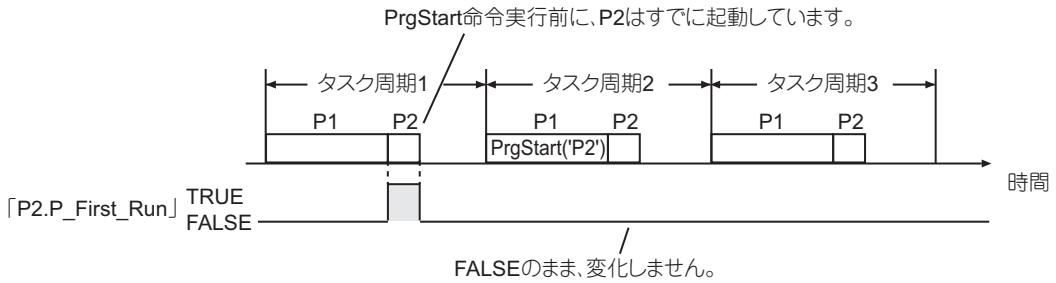
●「isFirstRun」の値が TRUE で、当該プログラムが停止している場合

当該プログラムが起動したときに、1 タスク周期の間、「P_First_Run」の値が TRUE になります。その後、「P_First_Run」の値は FALSE になります。



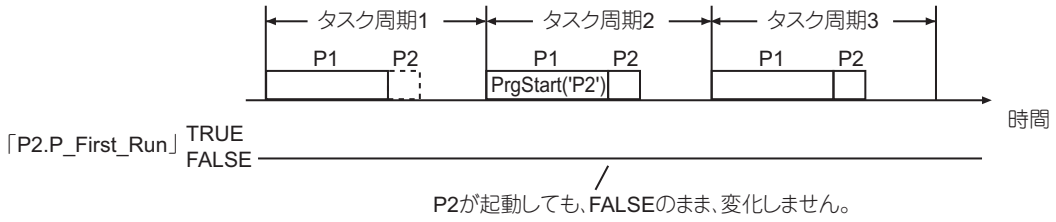
●「isFirstRun」の値が TRUE で、当該プログラムがすでに起動している場合

PrgStart 命令を実行しても、「P_First_Run」の値は、FALSE のまま変化しません。



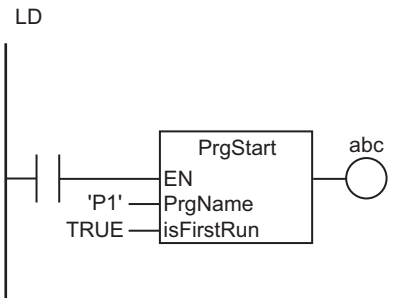
●「isFirstRun」の値が FALSE の場合

当該プログラムが起動しても、「P_First_Run」の値は、FALSE のまま変化しません。



記述例

プログラム'P1'の実行を指令する場合の記述例を、以下に示します。



ST

```
abc:=PrgStart('P1', TRUE);
```

関連するシステム定義変数

変数名	名称	データ型	内容
P_First_Run	プログラム起動時 1 周期 ON フラグ	BOOL	当該プログラムが実行を開始した場合、1 タスク周期の間 TRUE になります。それ以外は FALSE です。 ただし、「isFirstRun」の値を FALSE にして PrgStart 命令を実行した場合は、当該プログラムが実行を開始しても、FALSE のままです。 当該プログラムが実行を開始するごとに、特定の処理を行いたい場合に使用します。
P_First_RunMode	運転開始時 1 周期 ON フラグ	BOOL	コントローラの動作モードを、プログラムモードから運転モードに変更したときに、当該プログラムが実行中である場合、1 タスク周期だけ TRUE となります。 当該プログラムが実行中でなければ、FALSE のままです。 CPU ユニットが運転を開始したときに初期処理を行う場合などに使用します。

参考

- ユーザプログラムから、指定したプログラムを停止させるには、「PrgStop (P.2-872)」命令を使用してください。
- ユーザプログラムから、指定したプログラムの状態を確認するには、「PrgStatus (P.2-889)」命令を使用してください。

使用上の注意

- すでに実行状態にあるプログラムを指定して本命令を実行した場合、異常にはなりません。
- 同一プログラムを指定して複数の本命令を実行した場合、「isFirstRun」の指定は、最も早く実行した命令に従います。
- PrgStart 命令を実行し、当該プログラムが実行される前に PrgStop 命令を実行した場合、当該プログラムは実行されません。
- PrgStop 命令を実行し、当該プログラムが停止する前に PrgStart 命令を実行した場合、当該プログラムは停止しません。
- コントローラの動作モードが運転モードに移行した直後の各プログラムの動作は、Sysmac Studio の、各プログラムの「初期状態」設定に従います。つまり、運転モード移行前に実行した PrgStart 命令や PrgStop 命令は、運転モード移行後には無効です。
- 本命令を実行するタスクとは別のタスク内にあるプログラムを指定した場合、双方のタスクの実行優先度によって、指定されたプログラムの実行タイミングが異なり、コントローラが意図しない動作をする可能性があります。本命令を、指定するプログラムが割り付けられたタスクの先頭プログラムで実行することで、確実に、本命令を実行したタスク周期中に、指定プログラムを実行させることができます。
- 指定されたプログラムの内部変数、入力変数、出力変数、入出力変数は、そのプログラムが前回実行されたときの値を保持します。それらの変数を初期化してからプログラムを実行したい場合は、「isFirstRun」の値を TRUE にして本命令を実行し、かつ、指定するプログラム内に、「P_First_Run」の値が TRUE なら、変数を初期化する処理を入れてください。
- つぎの場合、異常となります。「Out」は FALSE となります。
  a)「PrgName」で指定したプログラムが、存在しない場合。

サンプルプログラム

PrgStop 命令の「サンプルプログラム (P.2-878)」を、参照してください。

PrgStop

指定したプログラムの停止を指令します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PrgStop	プログラム停止指令	FUN	(@)PrgStop EN PrgName Out	Out:=PrgStop(PrgName);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
PrgName	プログラム名	入力	指定するプログラム名	最大 128 バイト（半角英数字 127 文字 + 終端 NULL 文字）	−	*1
Out	正常終了フラグ	出力	正常終了フラグ TRUE: 正常終了 FALSE: 異常終了	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				○
Out	○																			

機能

プログラム名「PrgName」で指定したプログラムの、停止を指令します。
指定されたプログラムは、つぎにプログラムを実行するタイミングで停止されます。
指定されたプログラムがすでに停止中の場合、異常にはなりません。
指定するプログラムは、本命令を実行するタスクと同じタスク内にあっても、別タスク内にあってもかまいません。
本命令自身を含むプログラムを指定することも可能です。その場合、本命令自身を含むプログラムは、本命令を実行したタスク周期の最後まで実行され、つぎのタスク周期から停止します。
正常終了フラグ「Out」の値は、本命令が正常終了したら TRUE になり、異常終了したら FALSE になります。

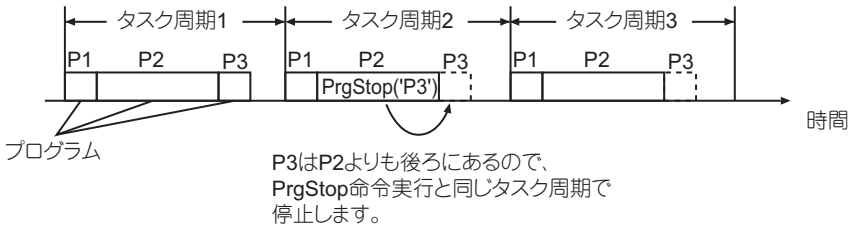
自タスクのプログラムを指定する場合の動作例

本命令を実行するタスクと同じタスク内のプログラムを指定する場合の動作例を、示します。

● PrgStop 命令よりも後ろにあるプログラムの停止を指令する場合

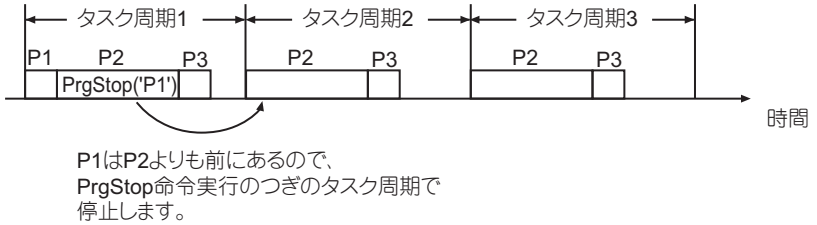
・ 同じタスク内に、P1、P2 および P3 の 3 つのプログラムがあります。

・タスク周期 1 で、P3 は実行されます。
・タスク周期 2 の P2 内で、P3 を指定して PrgStop 命令を実行します。
・P3 は P2 よりも後ろにあるので、タスク周期 2 で P3 が停止します。
・その後は、あらためて P3 を指定して PrgStop 命令を実行しなくても、P3 は停止したままです。



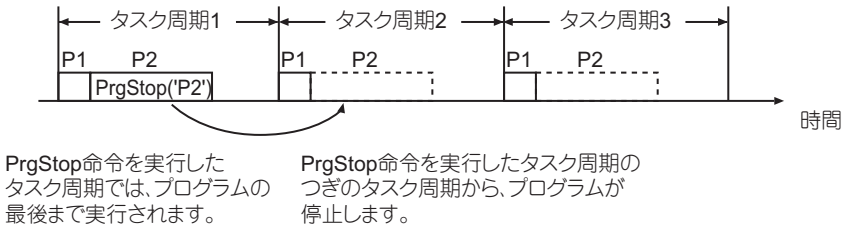
● PrgStop 命令よりも前にあるプログラムの停止を指令する場合

・同じタスク内に、P1、P2 および P3 の 3 つのプログラムがあります。
・タスク周期 1 で、P1 は実行されます。
・タスク周期 1 の P2 内で、P1 を指定して PrgStop 命令を実行します。
・P1 は P2 よりも前にあるので、タスク周期 2 で P1 が停止します。
・その後は、あらためて P1 を指定して PrgStop 命令を実行しなくても、P1 は停止したままです。



● PrgStop 命令自身を含むプログラムの停止を指令する場合

・同じタスク内に、P1 および P2 の 2 つのプログラムがあります。
・タスク周期 1 で、P2 は実行されます。
・タスク周期 1 の P2 内で、P2 を指定して PrgStop 命令を実行します。
・タスク周期 1 の P2 は、最後まで実行されます。
・タスク周期 2 から、P2 は停止します。
・その後は、あらためて P2 を指定して PrgStop 命令を実行しなくても、P2 は停止したままです。



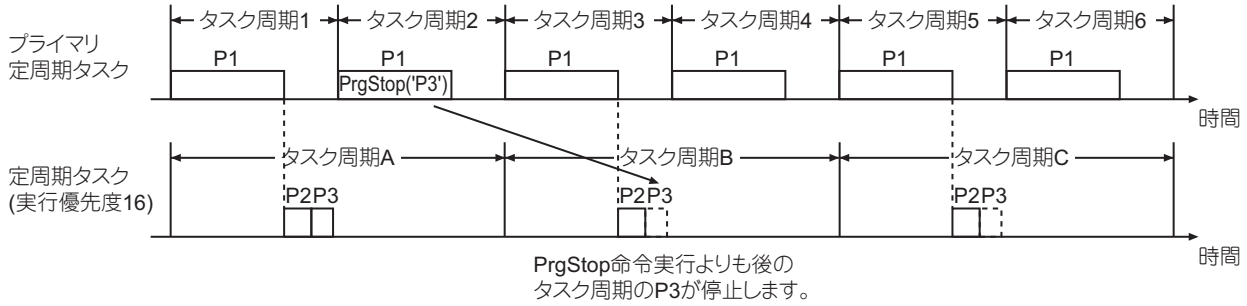
別タスクのプログラムを指定する場合の動作例

本命令を実行するタスクとは別のタスク内のプログラムを指定する場合の動作例を、示します。

● 自タスクよりも実行優先度の低いタスク内の、プログラムの停止を指令する場合

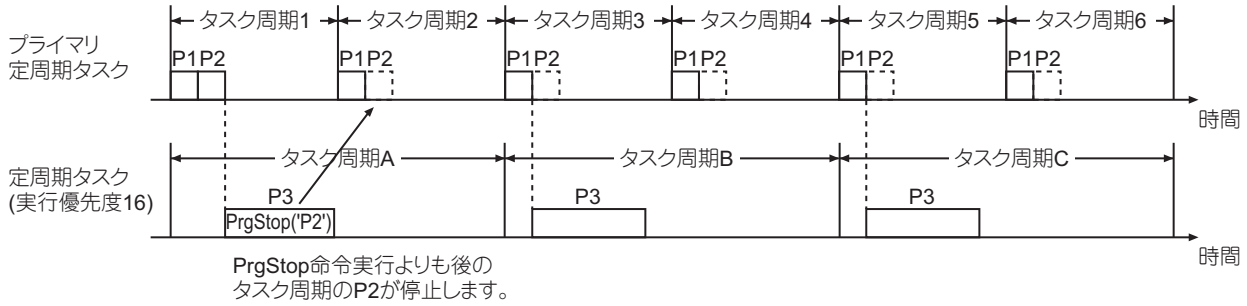
・プライマリ定周期タスクに P1、定周期タスクに P2、P3 の 3 つのプログラムがあります。

- 定周期タスクのタスク周期 A で、P3 は実行されます。
- プライマリ定周期タスクのタスク周期 2 の P1 内で、P3 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、定周期タスクのタスク周期 B の P3 が停止します。
- その後は、あらためて P3 を指定して PrgStop 命令を実行しなくても、P3 は停止したままです。



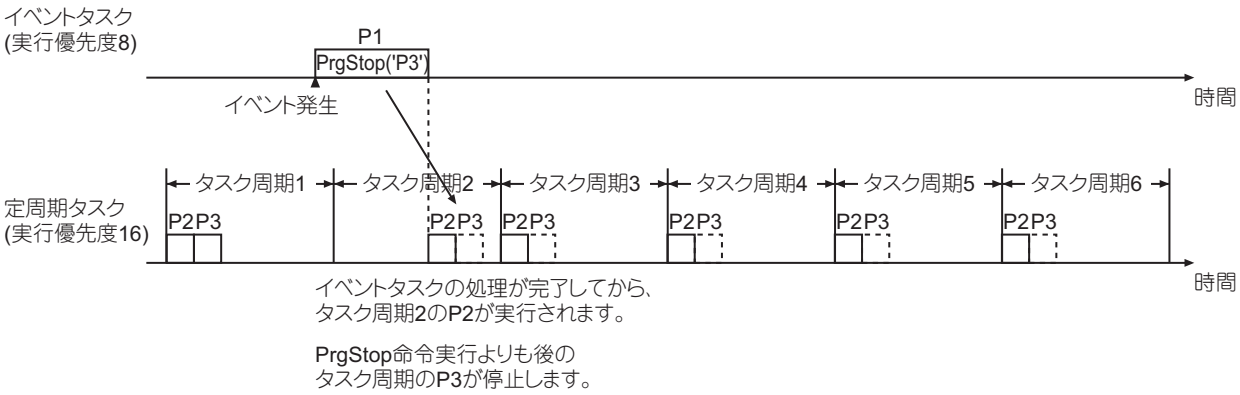
● 自タスクよりも実行優先度の高いタスク内の、プログラムの停止を指令する場合

- プライマリ定周期タスクに P1、P2、定周期タスクに P3 の 3 つのプログラムがあります。
- プライマリ定周期タスクのタスク周期 1 で、P2 は実行されます。
- 定周期タスクのタスク周期 A の P3 内で、P2 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、プライマリ定周期タスクのタスク周期 2 の P2 が停止します。
- その後は、あらためて P2 を指定して PrgStop 命令を実行しなくても、P2 は停止したままです。



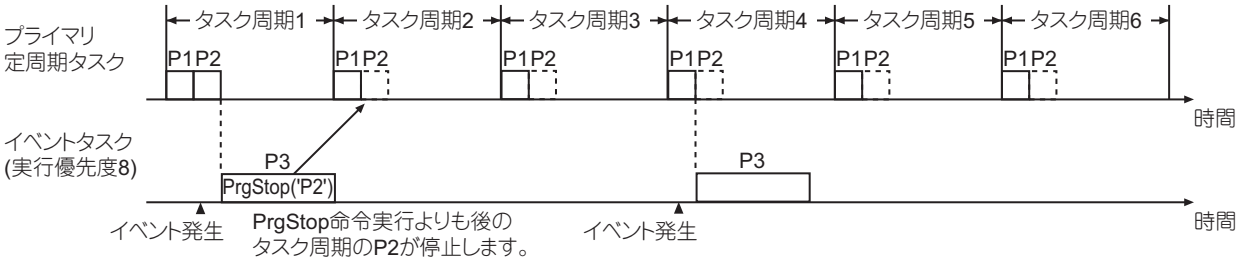
● イベントタスクから、実行優先度が低いタスク内のプログラムの停止を指令する場合

- イベントタスク（実行優先度 8）に P1、定周期タスク（実行優先度 16）に P2、P3 の 3 つのプログラムがあります。
- 定周期タスクのタスク周期 1 で、P3 は実行されます。
- イベントタスク 内で、P3 を指定して PrgStop 命令を実行します。
- イベントタスクが実行されたことにより、定周期タスクのタスク周期 2 の P2,P3 は、イベントタスクの処理が完了してから実行されます。
- その結果、定周期タスクのタスク周期 2 の P3 は、PrgStop 命令実行の後になるため、停止します。
- その後は、あらためて P3 を指定して PrgStop 命令を実行しなくても、P3 は停止したままです。



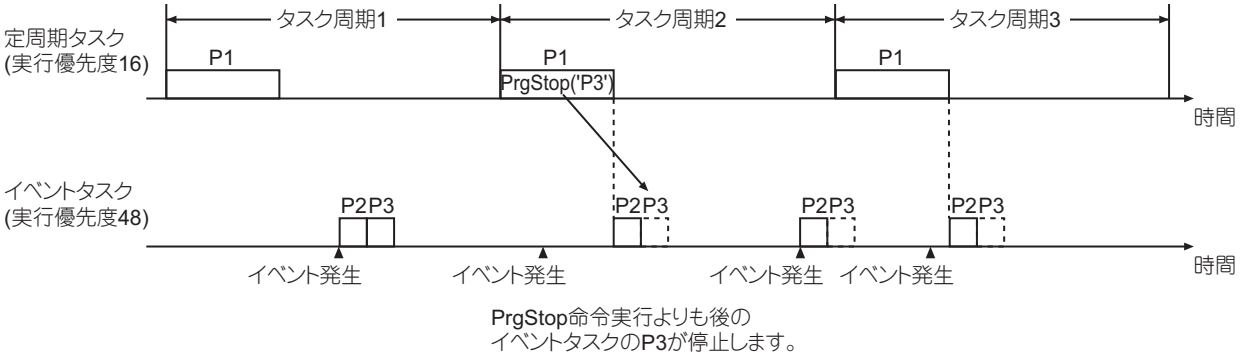
● イベントタスクから、実行優先度が高いタスク内のプログラムの停止を指令する場合

- プライマリ定周期タスクに P1、P2、イベントタスクに P3 の 3 つのプログラムがあります。
- プライマリ定周期タスクのタスク周期 1 で、P2 は実行されます。
- イベントタスク 内で、P2 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、プライマリ定周期タスクのタスク周期 2 の P2 が停止します。
- その後は、あらためて P2 を指定して PrgStop 命令を実行しなくても、P2 は停止したままです。



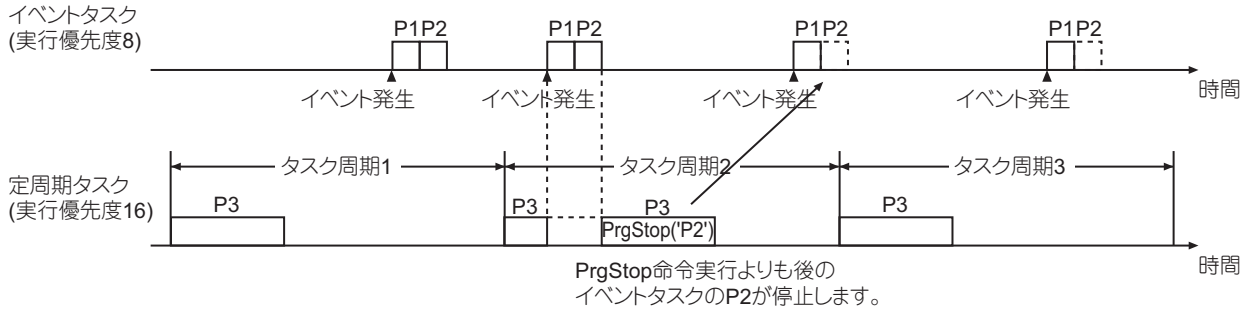
● 定周期タスクから、優先度の低いイベントタスク内のプログラムの停止を指令する場合

- 定周期タスク（実行優先度 16）に P1、イベントタスク（実行優先度 48）に P2、P3 の 3 つのプログラムがあります。
- イベントタスクの P3 は、実行されます。
- 定周期タスク 内で、P3 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、イベントタスクの P3 が停止します。
- その後は、あらためて P3 を指定して PrgStop 命令を実行しなくても、P3 は停止したままです。



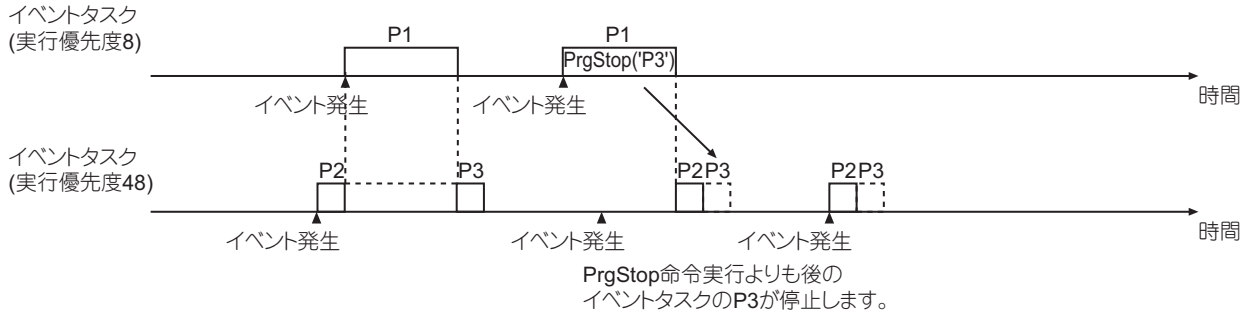
● 定周期タスクから、優先度の高いイベントタスク内のプログラムの停止を指令する場合

- イベントタスク（実行優先度 8）に P1、P2、定周期タスク（実行優先度 16）に P3 の 3 つのプログラムがあります。
- イベントタスクの P2 は、実行されます。
- 定周期タスク 内で、P2 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、イベントタスクの P2 が停止します。
- その後は、あらためて P2 を指定して PrgStop 命令を実行しなくても、P2 は停止したままです。



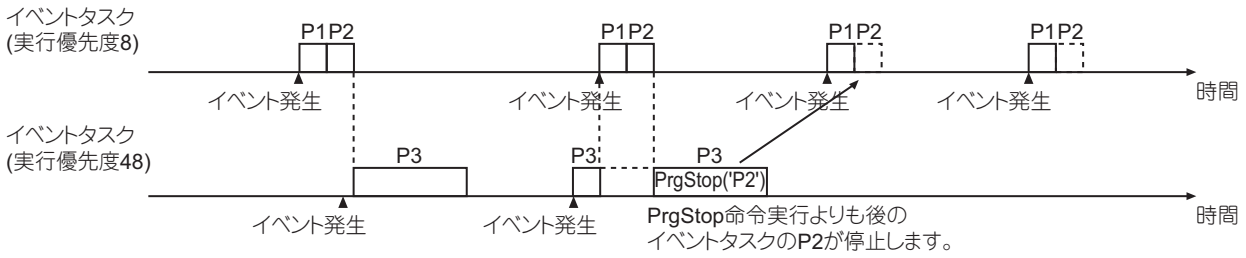
● イベントタスクから、優先度の低いイベントタスク内のプログラムの停止を指令する場合

- イベントタスク（実行優先度 8）に P1、イベントタスク（実行優先度 48）に P2、P3 の 3 つのプログラムがあります。
- イベントタスク（実行優先度 48）の P3 は、実行します。
- イベントタスク（実行優先度 8） 内で、P3 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、イベントタスク（実行優先度 48）の P3 が停止します。
- その後は、あらためて P3 を指定して PrgStop 命令を実行しなくても、P3 は停止したままです。



● イベントタスクから、優先度の高いイベントタスク内のプログラムの停止を指令する場合

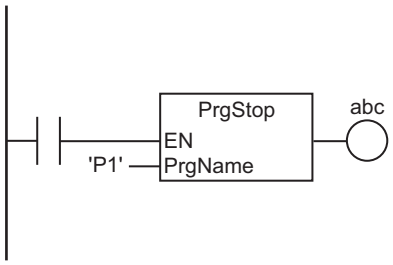
- イベントタスク（実行優先度 8）に P1、P2、イベントタスク（実行優先度 48）に P3 の 3 つのプログラムがあります。
- イベントタスク（実行優先度 8）の P2 は、実行されます。
- イベントタスク（実行優先度 48） 内で、P2 を指定して PrgStop 命令を実行します。
- PrgStop 命令の実行よりも後に実行される、イベントタスク（実行優先度 8）の P2 が停止します。
- その後は、あらためて P2 を指定して PrgStop 命令を実行しなくても、P2 は停止したままです。



記述例

プログラム'P1'の停止を指令する場合の記述例を、以下に示します。

LD



ST

```
abc:=PrgStop('P1');
```

参考

- ユーザプログラムから、指定したプログラムを実行させるには、「PrgStart (P.2-864)」命令を使用してください。
- ユーザプログラムから、指定したプログラムの状態を確認するには、「PrgStatus (P.2-889)」命令を使用してください。

使用上の注意

- すでに停止状態にあるプログラムを指定して本命令を実行した場合、異常にはなりません。
- PrgStart 命令を実行し、当該プログラムが実行される前に PrgStop 命令を実行した場合、当該プログラムは実行されません。
- PrgStop 命令を実行し、当該プログラムが停止する前に PrgStart 命令を実行した場合、当該プログラムは停止しません。
- 入力変数に「Execute」をもつ命令は、実行時間がタスク周期を超えても、処理を最後まで続けます。このような命令を含むプログラムの停止を指令する場合は、その命令の出力変数「Busy」の値が FALSE であり、その命令の実行が停止していることを事前に確認してください。
- NX_DOutTimeStamp 命令および NX_AryDOutTimeStamp 命令は、複数のタスクにまたがって処理を行う場合があります。これらの命令を含むプログラムの停止を指令する場合は、これらの命令の入力変数「Enable」の値が FALSE になっていることを事前に確認してください。
- コントローラの動作モードが運転モードに移行した直後の各プログラムの動作は、Sysmac Studio の、各プログラムの「初期状態」設定に従います。つまり、運転モード移行前に実行した PrgStart 命令や PrgStop 命令は、運転モード移行後には無効です。
- 本命令を実行するタスクとは別のタスク内にあるプログラムを指定した場合、双方のタスクの実行優先度によって、指定されたプログラムの停止タイミングが異なり、コントローラが意図しない動作をする可能性があります。本命令を、指定するプログラムが割り付けられたタスクの先頭プログ

ラムで実行することで、確実に、本命令を実行したタスク周期中に、指定プログラムを停止させることができます。

- 本命令は、指定するプログラムに対して、以下のことを確認してから実行してください。
  a）モーション制御命令が実行中でないこと。
  b）入力変数に「Execute」をもつ命令、つまり、実行時間がタスク周期を超えても、処理を最後まで続ける命令が実行中でないこと。
  c）タイムスタンプ命令が、指定時刻待ちの状態でないこと。
- 指定したプログラムの停止時に、プログラムの出力はリセットされません。停止直前の値を保持します。プログラムの停止時に出力をリセットする場合は、あらかじめ指定したプログラム内でマスタコントロールによるリセットを実行してください。
- 本命令で指定したプログラムが停止しても、プログラム内の、入力変数「Execute」を持つ FB 型の命令は、最後まで処理を続けます。
- 本命令で指定したプログラムが停止しても、プログラム内のモーション制御命令は、最後まで処理を続けます。
- つぎの場合、異常となります。「Out」は FALSE となります。
  a）「PrgName」で指定したプログラムが、存在しない場合。

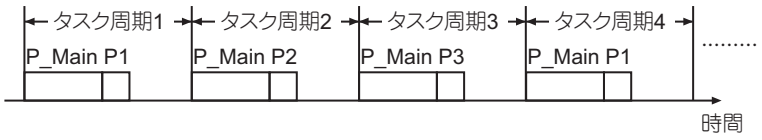
サンプルプログラム

2 つのプログラムを例に上げて、説明します。

順番に実行させる例

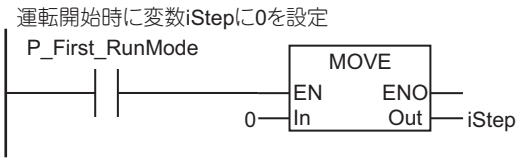
3 つのプログラムを、タスク周期ごとに順番に実行させる例を示します。

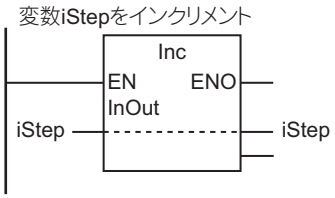
P1、P2 および P3 の 3 つのプログラムがあります。
これらを、タスク周期ごとに順番に繰り返し実行させます。
3 つのプログラムへの実行と停止の指令を出すプログラムは、P_Main です。

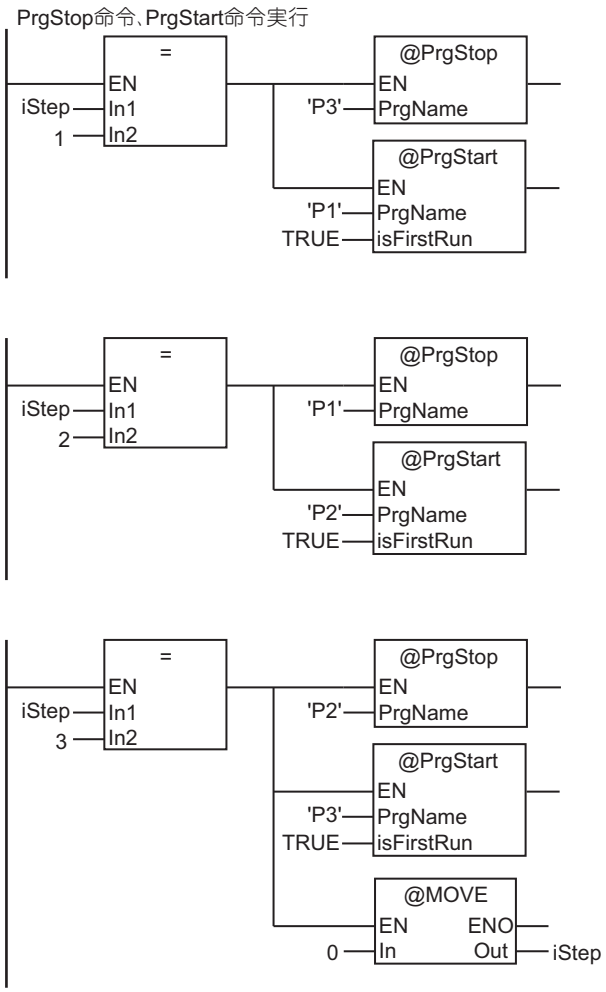


● **LD**

名称	データ型	初期値	コメント
iStep	DINT	0	実行させるプログラムの番号







● ST

名称	データ型	初期値	コメント
iStep	DINT	0	実行させるプログラムの番号

```
// 運転開始時に変数 iStep に 0 を設定
IF P_First_RunMode THEN
  iStep:=0;
END_IF;

// 変数 iStep をインクリメント
iStep:=iStep+1;

// PrgStop 命令、PrgStart 命令実行
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;
```

指定したプログラムを、次回運転時に実行させる例

複数のプログラムのうち、指定したプログラムだけを、次回運転時に実行させる例を示します。

次回運転時に実行させたいプログラムを指定してから、コントローラの電源を OFF します。
つぎに電源を ON すると、指定しておいたプログラムだけが実行されます。

● プログラム、モジュール、モジュール構成

プログラムは Program1 から Program8 まで、8 種類あります。
各プログラムは、ModuleA から ModuleE までの 5 種類のモジュールに、以下のように属しています。

モジュール	属するプログラム
ModuleA	Program1
ModuleB	Program2
ModuleC	Program3、Program4
ModuleD	Program5、Program6、Program7
ModuleE	Program8

実行させるプログラムは、モジュール単位で指定します。また、実行させるモジュールの組み合わせを示したものを、モジュール構成と呼びます。
たとえば、ModuleA と ModuleC を実行させるモジュール構成を指定した場合、Program1、Program3、Program4 の 3 つのプログラムが実行されます。

● 実行されるモジュール構成の指定方法

モジュール構成は、設定ファイルにテキストデータとして記述します。設定ファイルのファイル名は Config.txt とし、SD メモリカードのルートディレクトリに格納しておきます。設定ファイルには、モジュール構成を複数記述することができます。
電源を OFF する前に、タッチパネルを使用して、設定ファイルに記述された中から、次回運転時に実行されるモジュール構成番号を指定します。

● 設定ファイルの書式

設定ファイルの書式は、以下のとおりです。

行	記述内容
1 行め	モジュール構成の数
2 行め以降	モジュール構成番号、ModuleA の実行フラグ *1、ModuleB の実行フラグ、ModuleC の実行フラグ、ModuleD の実行フラグ、ModuleE の実行フラグ

*1.　実行なら TRUE、非実行なら FALSE を記述します。

設定ファイルの記述例を、以下に示します。

```
3
Config1, TRUE, TRUE, TRUE, FALSE, FALSE
Config2, TRUE, TRUE, FALSE, TRUE, FALSE
Config3, TRUE, TRUE, TRUE, FALSE, TRUE
```

この設定ファイルには、Config1、Config2 および Config3 の 3 つのモジュール構成が記述されています。

そのうち、Config1 のモジュール構成は、ModuleA、ModuleB および ModuleC は実行、ModuleD と ModuleE は非実行です。

● データ型定義

以下のように、構造体型「myConfig」を定義します。

構造体型		名称	データ型	オフセット種別	コメント
	▼	myConfig	STRUCT	NJ	モジュール構成
		configName	STRING[32]		モジュール構成名
		moduleA	BOOL		ModuleA の実行フラグ
		moduleB	BOOL		ModuleB の実行フラグ
		moduleC	BOOL		ModuleC の実行フラグ
		moduleD	BOOL		ModuleD の実行フラグ
		moduleE	BOOL		ModuleE の実行フラグ

● LD

名称	データ型	初期値	保持	コメント
Open	FileOpen		☐	FileOpen 命令のインスタンス
TopLineGetter	FileGets		☐	FileGets 命令のインスタンス
LineGetter	FileGets		☐	FileGets 命令のインスタンス
Close	FileClose		☐	FileClose 命令のインスタンス
PTInput_TargetConfigNum_Retain	USINT	0	☑	次回運転時に実行させるモジュール構成番号
CurrentLineNum	USINT	1	☐	設定ファイルの、現在読出中の行
TargetLineNum	USINT	0	☐	「CurrentConfig」の、設定ファイル内での行
ConfigNum	USINT	1	☐	設定ファイルの 1 行めに書かれた数
LineMax	USINT	3	☐	「ConfigNum」から求めた、設定ファイルの行数
isOverLine	BOOL	FALSE	☐	「LineMax」の値より、「PTInput_TargetConfigNum_Retain」の値の方が大きい異常発生フラグ
Busy	BOOL	FALSE	☐	処理中フラグ
SubDeliNG	BOOL	FALSE	☐	「CurrentConfig」の読出異常終了フラグ
Error	BOOL	FALSE	☐	異常フラグ
opening	BOOL	FALSE	☐	設定ファイルオープンの実行フラグ
myFileID	DWORD	0	☐	設定ファイルのファイル ID
TopLineGetting	BOOL	FALSE	☐	「ConfigNum」の読出実行フラグ
GetConfigNumDone	BOOL	FALSE	☐	「ConfigNum」の読出完了フラグ
SelectDone	BOOL	FALSE	☐	「CurrentConfig」の読出完了フラグ

名称	データ型	初期値	保持	コメント
reading	BOOL	FALSE	┌┐	設定ファイルの 2 行め以降読出の実行フラグ
CurrentConfig	myConfig	(configName:='', moduleA:=FALSE, moduleB:=FALSE, moduleC:=FALSE, moduleD:=FALSE)	┌┐	次回運転時に実行させるモジュール構成
Error_exceptOpen	BOOL	FALSE	┌┐	異常発生時の、設定ファイルクローズ実行フラグ

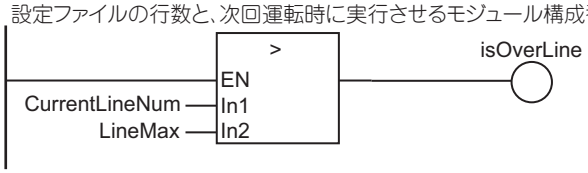
次回運転時に実行させるモジュール構成番号取得



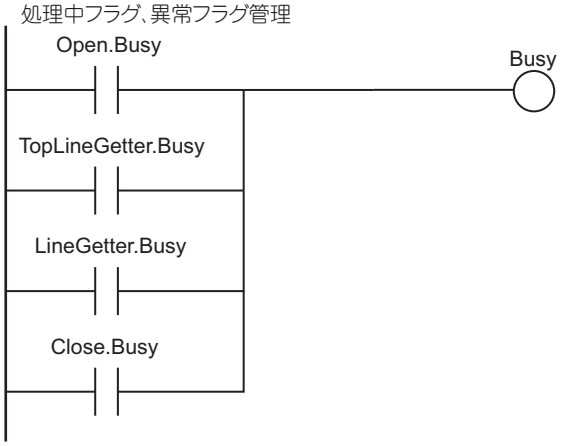
設定ファイルの1行めの記述から、行数を算出

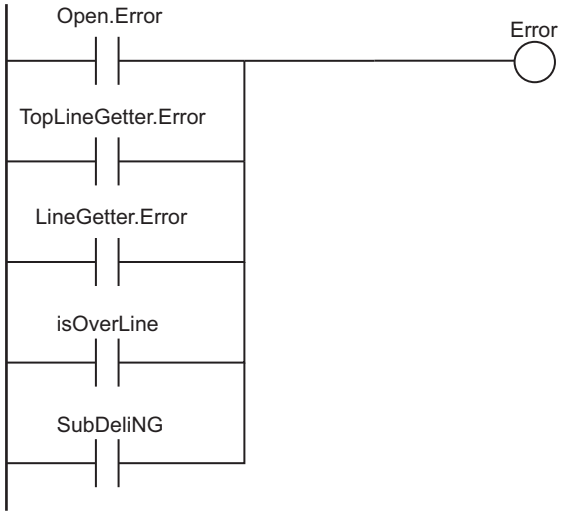


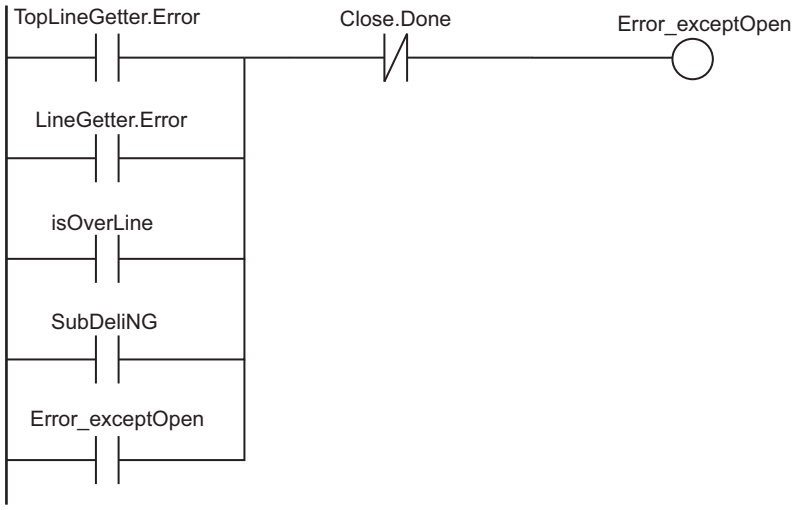
設定ファイルの行数と、次回運転時に実行させるモジュール構成番号が合わない異常検知

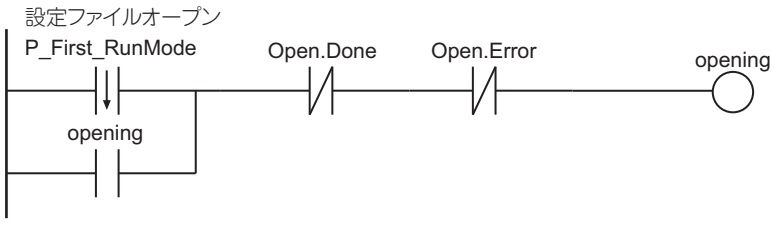


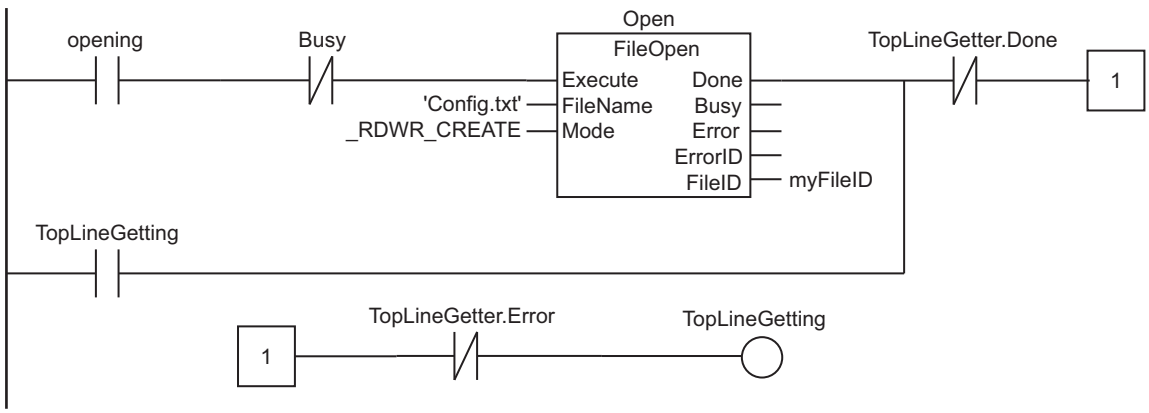
処理中フラグ、異常フラグ管理

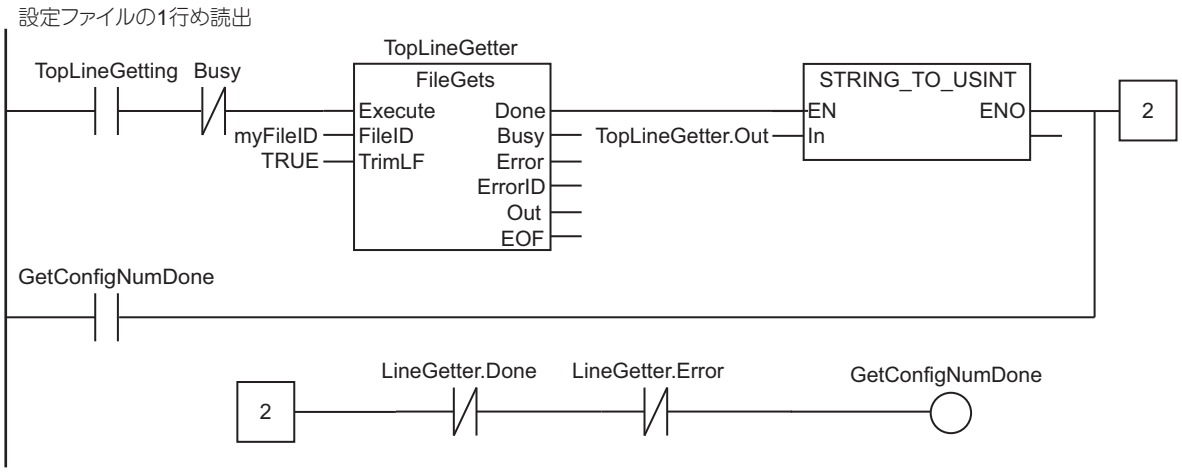


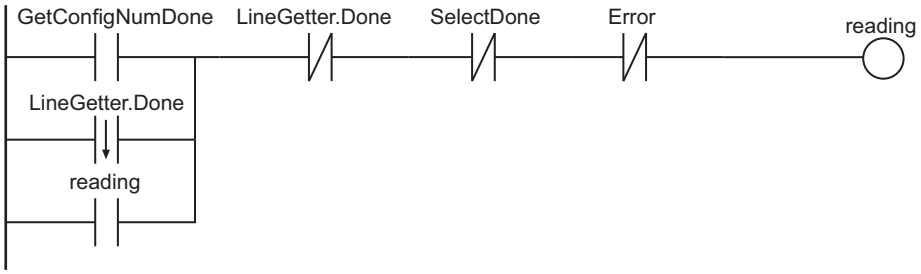

Open.Error
Error
TopLineGetter.Error
LineGetter.Error
isOverLine
SubDeliNG

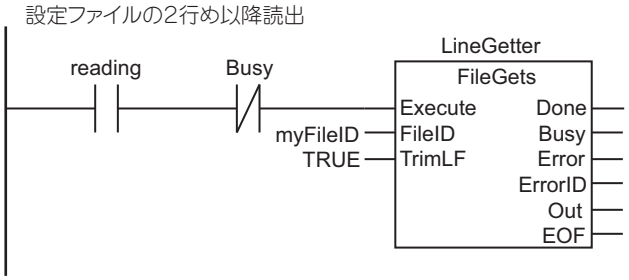

TopLineGetter.Error
Close.Done
Error_exceptOpen
LineGetter.Error
isOverLine
SubDeliNG
Error_exceptOpen

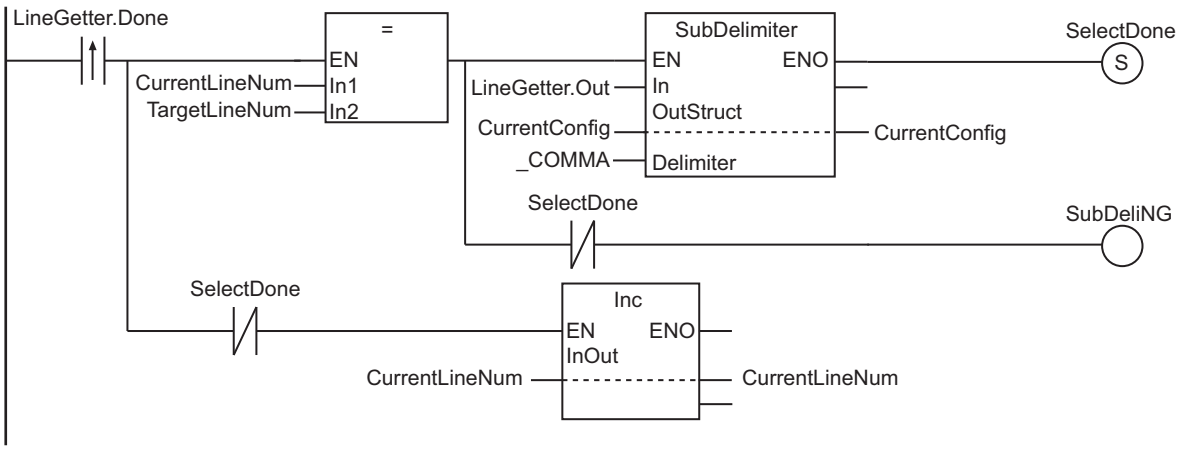

設定ファイルオープン
P_First_RunMode
Open.Done
Open.Error
opening
opening

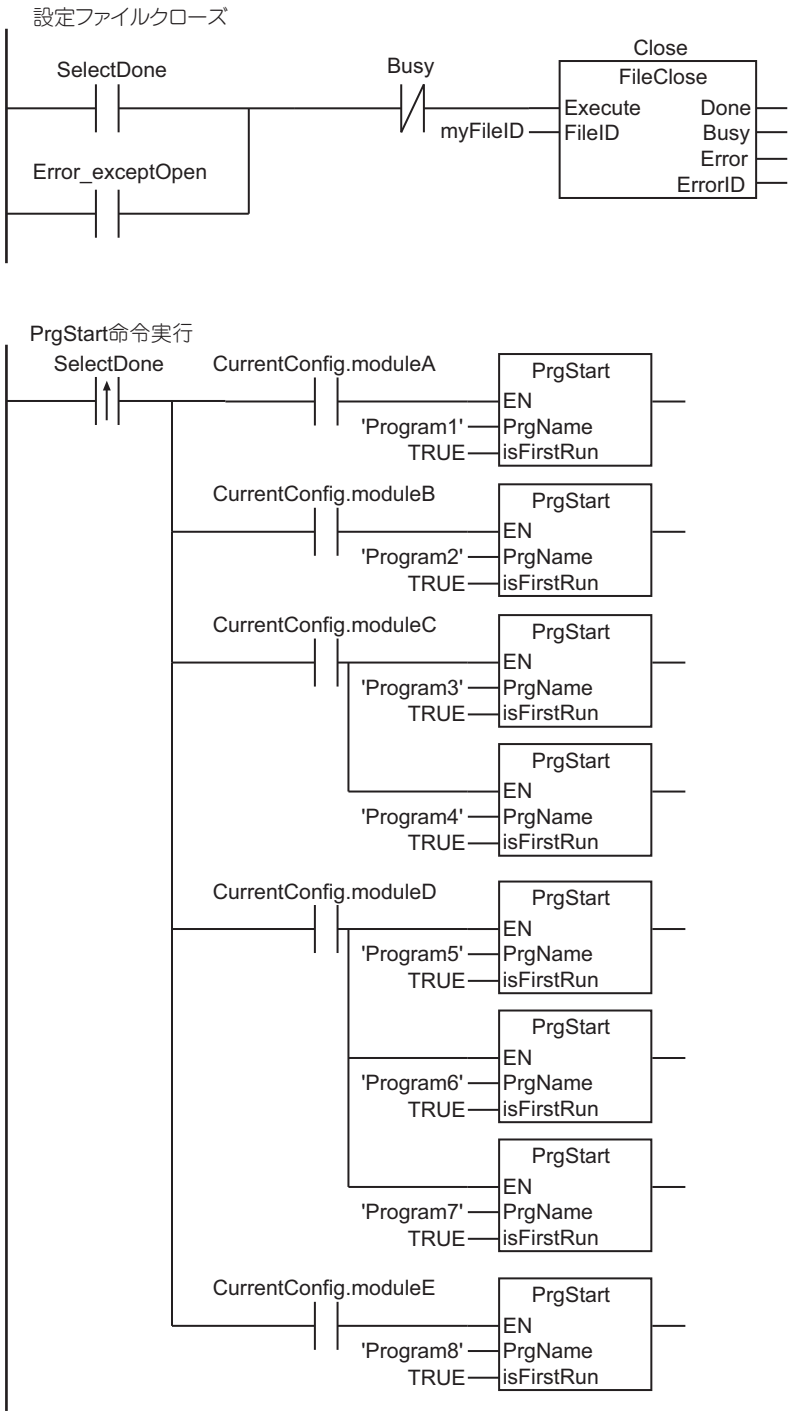

opening
Busy
Open
FileOpen
Execute
Done
'Config.txt'
FileName
Busy
_RDWR_CREATE
Mode
Error
ErrorID
FileID
myFileID
TopLineGetter.Done
1
TopLineGetting
TopLineGetter.Error
TopLineGetting
1


設定ファイルの1行め読出
TopLineGetter
FileGets
TopLineGetting
Busy
Execute
FileID
TrimLF
myFileID
TRUE
Done
Busy
Error
ErrorID
Out
EOF
TopLineGetter.Out
STRING_TO_USINT
EN
In
ENO
2
GetConfigNumDone
LineGetter.Done
LineGetter.Error
GetConfigNumDone
2


GetConfigNumDone
LineGetter.Done
SelectDone
Error
reading
LineGetter.Done
reading


設定ファイルの2行め以降読出
LineGetter
FileGets
reading
Busy
Execute
FileID
TrimLF
myFileID
TRUE
Done
Busy
Error
ErrorID
Out
EOF


LineGetter.Done
=
EN
In1
In2
CurrentLineNum
TargetLineNum
SubDelimiter
EN
ENO
In
OutStruct
Delimiter
LineGetter.Out
CurrentConfig
_COMMA
CurrentConfig
SelectDone
S
SelectDone
SubDeliNG
SelectDone
Inc
EN
ENO
InOut
CurrentLineNum
CurrentLineNum



● ST

名称	データ型	初期値	保持	コメント
Open	FileOpen		□	FileOpen 命令のインスタンス
TopLineGetter	FileGets		□	FileGets 命令のインスタンス
LineGetter	FileGets		□	FileGets 命令のインスタンス
Close	FileClose		□	FileClose 命令のインスタンス
PTInput_TargetConfigNum_Retain	USINT	0	☑	次回運転時に実行させるモジュール構成番号

名称	データ型	初期値	保持	コメント
CurrentLineNum	USINT	1	☐	設定ファイルの、現在読出中の行
TargetLineNum	USINT	0	☐	「CurrentConfig」の、設定ファイル内での行
ConfigNum	USINT	1	☐	設定ファイルの 1 行めに書かれた数
LineMax	USINT	3	☐	「ConfigNum」から求めた、設定ファイルの行数
isOverLine	BOOL	FALSE	☐	「LineMax」の値より、「PTInput_TargetConfigNum_Retain」の値の方が大きい異常発生フラグ
Busy	BOOL	FALSE	☐	処理中フラグ
SubDeliNG	BOOL	FALSE	☐	「CurrentConfig」の読出異常終了フラグ
Error	BOOL	FALSE	☐	異常フラグ
opening	BOOL	FALSE	☐	設定ファイルオープンの実行フラグ
myFileID	DWORD	0	☐	設定ファイルのファイル ID
TopLineGetting	BOOL	FALSE	☐	「ConfigNum」の読出実行フラグ
GetConfigNumDone	BOOL	FALSE	☐	「ConfigNum」の読出完了フラグ
SelectDone	BOOL	FALSE	☐	「CurrentConfig」の読出完了フラグ
reading	BOOL	FALSE	☐	設定ファイルの 2 行め以降読出の実行フラグ
CurrentConfig	myConfig	(configName:='', moduleA:=FALSE, moduleB:=FALSE, moduleC:=FALSE, moduleD:=FALSE)	☐	次回運転時に実行させるモジュール構成
Error_exceptOpen	BOOL	FALSE	☐	異常発生時の、設定ファイルクローズ実行フラグ
R_GetConfigNumDone	R_TRIG		☐	R_TRIG 命令のインスタンス
RS_1	RS		☐	RS 命令のインスタンス
RS_2	RS		☐	RS 命令のインスタンス
SecondCycle	F_TRIG		☐	F_TRIG 命令のインスタンス
RS_3	RS		☐	RS 命令のインスタンス
ConvertDone	BOOL	FALSE	☐	設定ファイルの 1 行目に書かれた文字から数への変換完了フラグ
RS_4	RS		☐	RS 命令のインスタンス
F_LineGetterDone	F_TRIG		☐	F_TRIG 命令のインスタンス
R_LineGetterDone	R_TRIG		☐	R_TRIG 命令のインスタンス

名称	データ型	初期値	保持	コメント
isTargetLine	BOOL	FALSE	☐	読出中の行が、次回運転時に実行させるモジュール構成の行であることを示すフラグ
SubDeliCondition	BOOL	FALSE	☐	モジュール構成の記述から「CurrentConfig」への展開実行フラグ
RS_5	RS		☐	RS 命令のインスタンス
SubDeliDone	BOOL	FALSE	☐	モジュール構成の記述から「CurrentConfig」への展開完了フラグ
R_SelectDone	R_TRIG		☐	R_TRIG 命令のインスタンス

```
// 次回運転時に実行させるモジュール構成番号取得
IF P_First_RunMode THEN
  TargetLineNum := PTInput_TargetConfigNum_Retain + USINT#1;
END_IF;

// 設定ファイルの 1 行めの記述から、行数を算出
R_GetConfigNumDone(Clk:=GetConfigNumDone);
IF R_GetConfigNumDone.Q THEN
  LineMax := ConfigNum + USINT#1;
END_IF;

// 設定ファイルの行数と、次回運転時に実行させるモジュール構成番号が合わない異常検知
isOverLine := (CurrentLineNum > LineMax);

// 処理中フラグ、異常フラグ管理
Busy := Open.Busy OR TopLineGetter.Busy OR LineGetter.Busy OR Close.Busy;

Error := Open.Error OR TopLineGetter.Error OR LineGetter.Error OR isOverLine OR SubDeliNG;

RS_1(Set:= (TopLineGetter.Error OR LineGetter.Error OR isOverLine OR SubDeliNG), reset1 := Close.Done, Q1 => Error_exceptOpen);

// 設定ファイルオープン
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=>TopLineGetting);

// 設定ファイルの 1 行め読出
TopLineGetter(Execute :=(TopLineGetting & NOT(Busy)), FileID := myFileID, TrimLF := TRUE);
ConfigNum := STRING_TO_USINT(EN:= TopLineGetter.Done, IN:=TopLineGetter.Out, ENO=>ConvertDone);
RS_4(Set := ConvertDone, Reset1:=(LineGetter.Done OR LineGetter.Error), Q1=>GetConfigNumDone);
F_LineGetterDone(Clk:=LineGetter.Done);
RS_5(Set := (GetConfigNumDone OR F_LineGetterDone.Q), Reset1:=(LineGetter.Done OR SelectDone OR Error), Q1=>reading);

// 設定ファイルの 2 行め以降読出
```

```
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 := CurrentConfig, Delimiter :=
_COMMA, ENO => SubDeliDone);
IF SubDeliDone THEN
  SelectDone := TRUE;
END_IF;
SubDeliNG := (SubDeliCondition & NOT(SubDeliDone));
Inc(EN := (R_LineGetterDone.Q & NOT(SelectDone)), InOut:= CurrentLineNum);

// 設定ファイルクローズ
Close(Execute := ((SelectDone OR Error_exceptOpen) & NOT(Busy)), FileID := myFileID);

// PrgStart 命令実行
R_SelectDone(Clk:=SelectDone);
//moduleA
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleA), PrgName :='Program1', isFirstRun:=TRUE);
//moduleB
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleB), PrgName :='Program2', isFirstRun:=TRUE);
//moduleC
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program3', isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program4', isFirstRun:=TRUE);
//moduleD
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program5', isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program6', isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program7', isFirstRun:=TRUE);
//moduleE
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleE), PrgName :='Program8', isFirstRun:=TRUE);
```

PrgStatus

指定したプログラムの状態を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
PrgStatus	プログラム状態読出	FUN	(@)PrgStatus EN　ENO PrgName　Out	Out:=PrgStatus(PrgName);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
PrgName	プログラム名	入力	指定するプログラム名	最大 128 バイト（半角英数字 127 文字 + 終端 NULL 文字）	−	*1
Out	プログラムの状態	出力	つぎに実行されるタイミングでのプログラムの状態 TRUE: 実行される FALSE: 停止する	データ型に従う	−	−

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				○
Out	○																			

機能

プログラム名「PrgName」で指定したプログラムの、つぎに実行されるタイミングでの状態を読み出します。
指定したプログラムが、つぎに実行されるタイミングで実行される場合は、「Out」の値が TRUE になります。
指定したプログラムが、つぎに実行されるタイミングで停止する場合は、「Out」の値が FALSE になります。

つぎに実行されるタイミングで実行される、および停止するとは、以下のような意味です。

プログラムの状態	意味
つぎに実行されるタイミングで実行される	• Sysmac Studio の、当該プログラムの「初期状態」に、［起動］が設定されている。 • PrgStart 命令が、当該プログラムに対して実行された。
つぎに実行されるタイミングで停止する	• Sysmac Studio の、当該プログラムの「初期状態」に、［停止］が設定されている。 • PrgStop 命令が、当該プログラムに対して実行された。

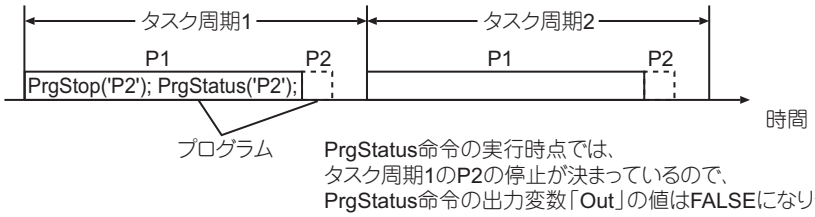
指定するプログラムは、本命令を実行するタスクと同じタスク内にあっても、別タスク内にあってもかまいません。

動作例

本命令のいくつかの場合の動作例を、示します。

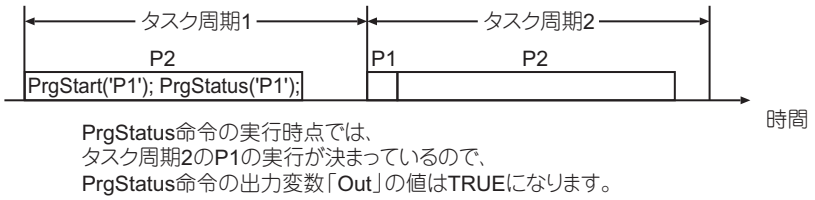
● 自タスク内で、PrgStatus 命令よりも後ろにあるプログラムの状態を読み出す場合

- 同じタスク内に、P1 および P2 の 2 つのプログラムがあります。
- タスク周期 1 の P1 内で、P2 を指定して PrgStop 命令を実行します。
- その後に、タスク周期 1 の P1 内で、P2 を指定して PrgStatus 命令を実行します。
- タスク周期 1 の P2 の停止が決まっているので、PrgStatus 命令の出力変数「Out」の値は、FALSE になります。



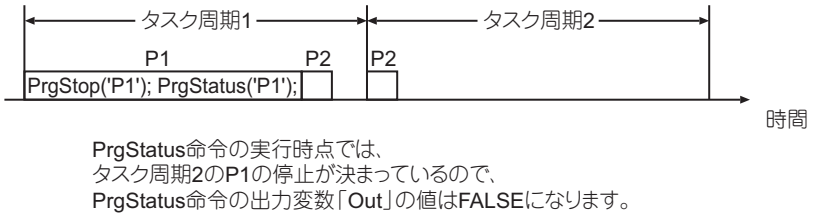
● 自タスク内で、PrgStatus 命令よりも前にあるプログラムの状態を読み出す場合

- 同じタスク内に、P1 および P2 の 2 つのプログラムがあります。
- タスク周期 1 の P2 内で、P1 を指定して PrgStart 命令を実行します。
- その後に、タスク周期 1 の P2 内で、P1 を指定して PrgStatus 命令を実行します。
- タスク周期 2 の P1 の実行が決まっているので、PrgStatus 命令の出力変数「Out」の値は、TRUE になります。



● PrgStatus 命令自身を含むプログラムの状態を読み出す場合

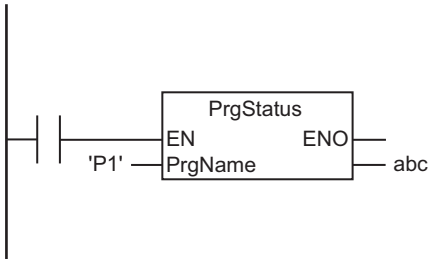
- タスク周期 1 の P1 内で、P1 を指定して PrgStop 命令を実行します。
- その後に、タスク周期 1 の P1 内で、P1 を指定して PrgStatus 命令を実行します。
- タスク周期 2 の P1 の停止が決まっているので、PrgStatus 命令の出力変数「Out」の値は、FALSE になります。



記述例

プログラム'P1'の状態を読み出す場合の記述例を、以下に示します。

LD



ST

```
abc:=PrgStatus('P1');
```

参考

- ユーザプログラムから、指定したプログラムを実行させるには、「PrgStart (P.2-864)」命令を使用してください。
- ユーザプログラムから、指定したプログラムを停止させるには、「PrgStop (P.2-872)」命令を使用してください。

使用上の注意

- つぎの場合、異常となります。「Out」は FALSE となります。
  a)「PrgName」で指定したプログラムが、存在しない場合。

サンプルプログラム

P1、P2 および P3 の 3 つのプログラムがあります。タッチパネルからの指示によって、実行されるプログラムが切り替わります。

タッチパネルの仕様

本プログラムは、コントローラにタッチパネルを接続することを前提としています。
タッチパネルには、以下のランプがあります。

ランプ名	意味
P1 実行中ランプ	P1 が実行中である場合に点灯します。
P2 実行中ランプ	P2 が実行中である場合に点灯します。
P3 実行中ランプ	P3 が実行中である場合に点灯します。

また、タッチパネルには以下のボタンがあります。

ボタン名	押下した場合の動作
実行プログラム切替ボタン	ボタンを押下するごとに、実行されるプログラムが P1→P2→P3→P1‥と順番に切り替えられます。

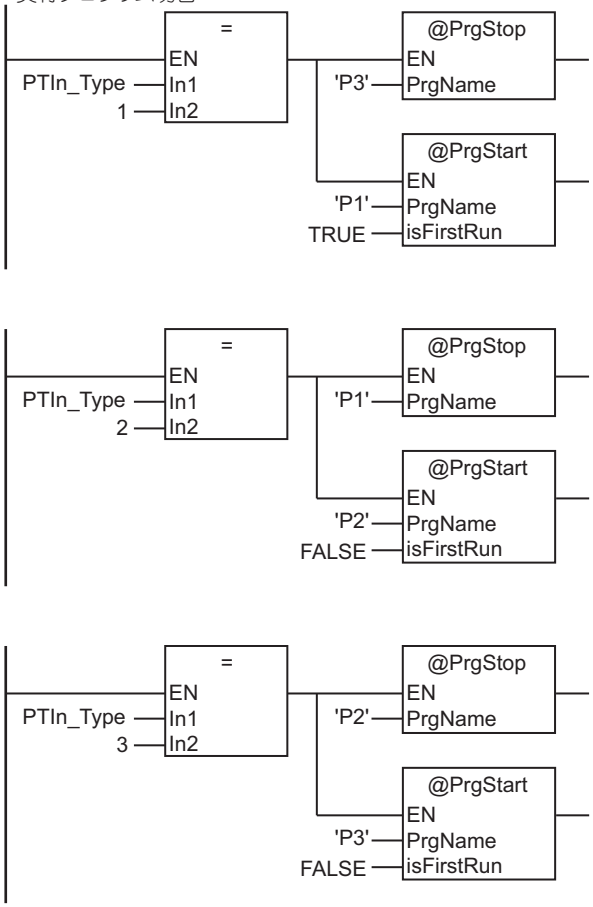
グローバル変数

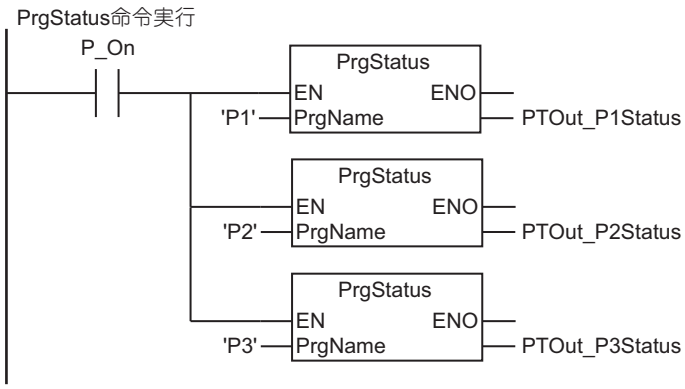
名称	データ型	初期値	コメント
PTIn_Type	INT	0	実行プログラム切替ボタン入力
PTOut_P1Status	BOOL	FALSE	P1 実行中ランプ出力
PTOut_P2Status	BOOL	FALSE	P2 実行中ランプ出力
PTOut_P3Status	BOOL	FALSE	P3 実行中ランプ出力

LD

外部変数	名称	データ型	コメント
	PTIn_Type	INT	実行プログラム切替ボタン入力
	PTOut_P1Status	BOOL	P1 実行中ランプ出力
	PTOut_P2Status	BOOL	P2 実行中ランプ出力
	PTOut_P3Status	BOOL	P3 実行中ランプ出力

実行プログラム切替





ST

外部変数	名称	データ型	コメント
	PTIn_Type	INT	実行プログラム切替ボタン入力
	PTOut_P1Status	BOOL	P1 実行中ランプ出力
	PTOut_P2Status	BOOL	P2 実行中ランプ出力
	PTOut_P3Status	BOOL	P3 実行中ランプ出力

```
// 実行プログラム切替
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;

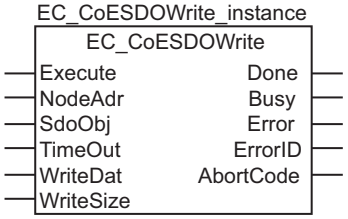
// PrgStatus 命令実行
IF P_On THEN
  PTOut_P1Status:=PrgStatus('P1');
  PTOut_P2Status:=PrgStatus('P2');
  PTOut_P3Status:=PrgStatus('P3');
END_IF;
```

EtherCAT 通信命令

命令	名称	ページ
EC_CoESDOWrite	CoE SDO 書込	P.2-896
EC_CoESDORead	CoE SDO 読出	P.2-899
EC_StartMon	EtherCAT パケットモニタ機能開始	P.2-904
EC_StopMon	EtherCAT パケットモニタ機能停止	P.2-910
EC_SaveMon	EtherCAT パケット保存	P.2-912
EC_CopyMon	EtherCAT パケット転送	P.2-914
EC_DisconnectSlave	EtherCAT スレーブ離脱	P.2-916
EC_ConnectSlave	EtherCAT スレーブ再加入	P.2-923
EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	P.2-925
EC_GetMasterStatistics	EtherCAT マスタ診断・統計情報読出	P.2-943
EC_ClearMasterStatistics	EtherCAT マスタ診断・統計情報クリア	P.2-946
EC_GetSlaveStatistics	EtherCAT スレーブ診断・統計情報読出	P.2-948
EC_ClearSlaveStatistics	EtherCAT スレーブ診断・統計情報クリア	P.2-951
NX_WriteObj	NX オブジェクト書込	P.2-953
NX_ReadObj	NX オブジェクト読出	P.2-968

EC_CoESDOWrite

EtherCAT ネットワーク上の、指定したスレーブが持つ CoE（CAN application protocol over EtherCAT の略）のオブジェクトに、値を書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_CoESDOWrite	CoE SDO 書込	FB		EC_CoESDOWrite_instance(Execute, NodeAdr, SdoObj, TimeOut, WriteDat, WriteSize, Done, Busy, Error, ErrorID, AbortCode);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
NodeAdr	スレーブノードアドレス	入力	アクセスするスレーブのノードアドレス	1 ～ 512	–	–
SdoObj	SDO パラメータ		SDO パラメータ	–		
TimeOut	タイムアウト時間		0: 2.0s 1 ～ 65535: 0.1 ～ 6553.5s	データ型に従う	0.1s	20 (2.0s)
WriteDat	書込データ		書込データ		–	–
WriteSize	書込データサイズ		書込データサイズ *1	1 ～ 2048	バイト	
AbortCode	アボートコード	出力	CoE で規定される SDO アクセスのレスポンスコード 0: 正常終了	データ型に従う	–	–

*1. 書込データが、BOOL 型や BOOL 型配列などの場合、書込データサイズが 1 バイト未満であることがあります。その場合、「WriteSize」の値は 1 にしてください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							○													
SdoObj	構造体_sSDO_ACCESS　詳細は「機能 (P.2-896)」参照																			
TimeOut							○													
WriteDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			
WriteSize							○													
AbortCode				○																

機能

スレーブノードアドレス「NodeAdr」で指定したノードが持つ CoE のオブジェクトに対して、データを書き込みます。
書き込むデータは、「WriteDat」の中身です。書き込むデータサイズは、「WriteSize」で指定します。

SDO パラメータは、「SdoObj」で指定します。

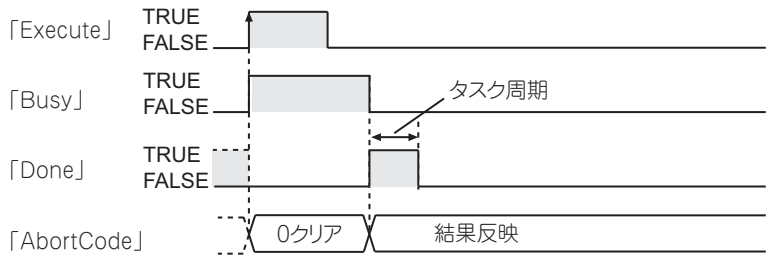
「SdoObj」のデータ型は、構造体_sSDO_ACCESS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SdoObj	SDO パラメータ	SDO パラメータ	_sSDO_ACCESS	–	–	–
Index	インデックス	CoE で定義されるオブジェクトディクショナリのインデックス番号	UINT	1 ～ 65535	–	–
Subindex	サブインデックス	CoE で定義されるオブジェクトディクショナリのサブインデックス番号	USINT	データ型に従う		
IsComplete Access	Complete アクセス	SDO の Complete アクセス機能指定 TRUE: 全サブインデックスのデータにアクセス FALSE: 指定したサブインデックスのデータにアクセス	BOOL			

書込完了後、タイムアウト時間「TimeOut」で指定した時間だけ、レスポンスを待ちます。
レスポンスは、「AbortCode」に格納されます。

正常終了時の「AbortCode」の値は 0 です。ただし、「AbortCode」に値が格納されるのは、「ErrorID」の値が 16#1804（SDO アボートレスポンス）の場合だけです。
「AbortCode」の値と意味は、スレーブによって規定されます。スレーブ側のマニュアルを参照してください。

タイミングチャートを、以下に示します。「AbortCode」に値が格納されるのは、命令の処理が終了し、「Busy」の値が FALSE になる時です。



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可

参考

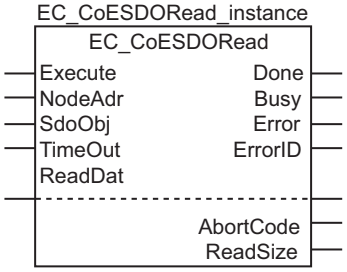
- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。
- SDO アボートコードは、「A-4 SDO アボートコード一覧 (P.A-23)」を参照してください。

使用上の注意

- 「WriteDat」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）EtherCAT マスタがメッセージ通信可能な状態になっていない場合。
  b）「NodeAdr」で指定したスレーブが、存在しない場合。
  c）「NodeAdr」で指定したスレーブが、通信可能な状態になっていない場合。
  d）スレーブが、異常レスポンスを返した場合。
  e）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

EC_CoESDORead

EtherCAT ネットワーク上の、指定したスレーブが持つ CoE（CAN application protocol over EtherCAT の略）のオブジェクトから、値を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_CoESDORead	CoE SDO 読出	FB		EC_CoESDORead_instance(Execute, NodeAdr, SdoObj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, AbortCode, ReadSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
NodeAdr	スレーブノードアドレス	入力	アクセスするスレーブのノードアドレス	1～512	–	–
SdoObj	SDO パラメータ		SDO パラメータ	–		
TimeOut	タイムアウト時間		0: 2.0s 1 ～ 65535: 0.1 ～ 6553.5s	データ型に従う	0.1s	0 (2.0s)
AbortCode	アボートコード	出力	CoE で規定される SDO アクセスのレスポンスコード 0: 正常終了	データ型に従う	–	–
ReadSize	読出データサイズ		読み出して、「ReadDat」に格納したデータのサイズ *1		バイト	
ReadDat	読出データ	入出力	読出データ格納用バッファ	データ型に従う	–	–

*1. 読出データが、BOOL 型や BOOL 型配列などの場合、読出データサイズが 1 バイト未満であることがあります。その場合、「ReadSize」の値は 1 になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							○													
SdoObj	構造体_sSDO_ACCESS　詳細は「機能 (P.2-900)」参照																			
TimeOut							○													
AbortCode				○																
ReadSize							○													
ReadDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			

機能

スレーブノードアドレス「NodeAdr」で指定したノードが持つ CoE のオブジェクトから、データを読み出します。
読み出したデータは、「ReadDat」 に格納します。そして、格納したデータのサイズを、「ReadSize」に格納します。「ReadSize」の値は、格納が成功した場合だけ有効です。
SDO パラメータは、「SdoObj」で指定します。

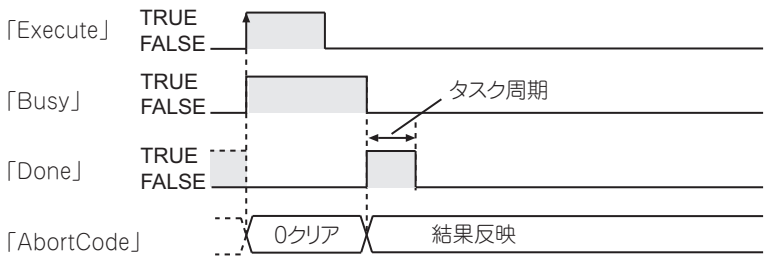
「SdoObj」のデータ型は、構造体_sSDO_ACCESS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SdoObj	SDO パラメータ	SDO パラメータ	_sSDO_ACCESS	–	–	–
Index	インデックス	CoE で定義されるオブジェクトディクショナリのインデックス番号	UINT	1 ～ 65535	–	–
Subindex	サブインデックス	CoE で定義されるオブジェクトディクショナリのサブインデックス番号	USINT	データ型に従う		
IsComplete Access	Complete アクセス	SDO の Complete アクセス機能指定 TRUE: 全サブインデックスのデータにアクセス FALSE: 指定したサブインデックスのデータにアクセス	BOOL			

読出完了後、タイムアウト時間「TimeOut」で指定した時間だけ、レスポンスを待ちます。
レスポンスは、「AbortCode」に格納されます。

正常終了時の「AbortCode」の値は 0 です。ただし、「AbortCode」に値が格納されるのは、「ErrorID」の値が 16#1804（SDO アボートレスポンス）の場合だけです。
「AbortCode」の値と意味は、スレーブによって規定されます。スレーブ側のマニュアルを参照してください。

タイミングチャートを、以下に示します。「AbortCode」に値が格納されるのは、命令の処理が終了し、「Busy」の値が FALSE になる時です。



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可

参考

- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。
- SDO アボートコードは、「A-4 SDO アボートコード一覧 (P.A-23)」を参照してください。

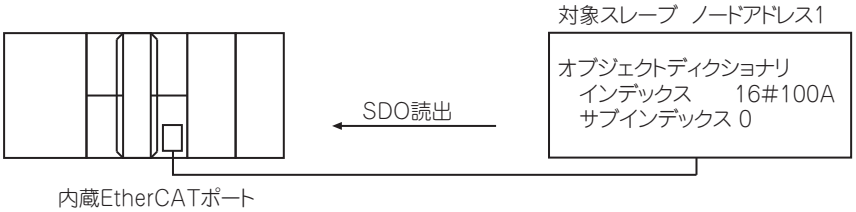
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a) EtherCAT マスタがメッセージ通信可能な状態になっていない場合。
  b)「NodeAdr」で指定したスレーブが、存在しない場合。
  c)「NodeAdr」で指定したスレーブが、通信可能な状態になっていない場合。
  d) スレーブが、異常レスポンスを返した場合。
  e)「ReadDat」のサイズより、読み出したデータサイズの方が大きい場合。
  f) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EtherCAT の SDO メッセージを使用して、オムロン製サーボドライバ 1S シリーズから、ソフトウェアバージョンを読み出します。このスレーブのノードアドレスは 1 とします。

ソフトウェアバージョンのオブジェクトインデックスは 16#100A、サブインデックスは 0 です。
読み出した値は、STRING 型変数「VersionInfo」に格納します。

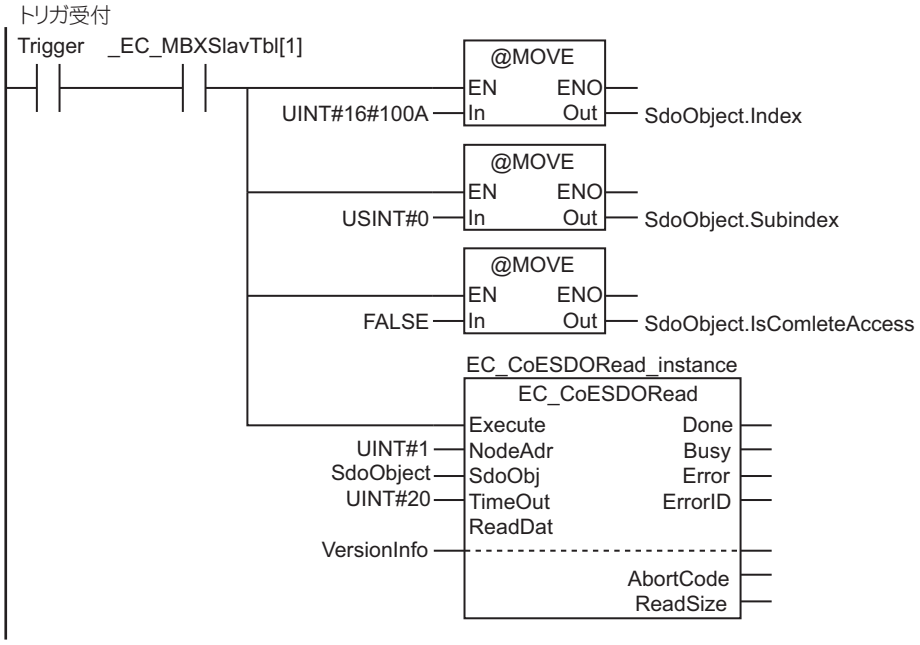


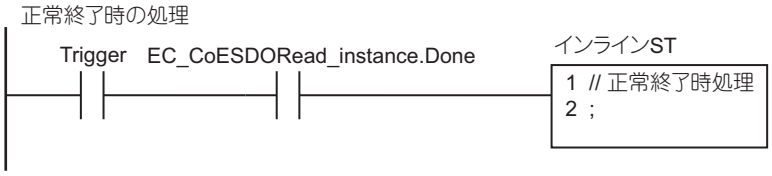
LD

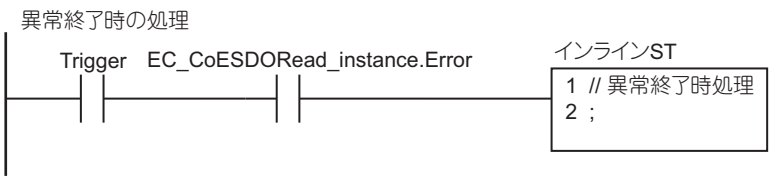
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件

内部変数	名称	データ型	初期値	コメント
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO パラメータ
	VersionInfo	STRING[256]	''	読出データ
	EC_CoESDORead_instance	EC_CoESDORead		

外部変数	名称	データ型	コンスタント	コメント
	_EC_MBXSlavTbl	ARRAY[1..512] OF BOOL	☑	メッセージ通信可能スレーブテーブル







ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO パラメータ

内部変数	名称	データ型	初期値	コメント
	DoSdoRead	BOOL	FALSE	処理中
	VersionInfo	STRING[256]	''	読出データ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	EC_CoESDORead_instance	EC_CoESDORead		

外部変数	名称	データ型	コンスタント	コメント
	_EC_MBXSlavTbl	ARRAY[1..512] OF BOOL	☑	メッセージ通信可能スレーブテーブル

```
// Trigger の立ち上がり検出
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, // インスタンス初期化
    ReadDat:=VersionInfo); // ダミー
END_IF;

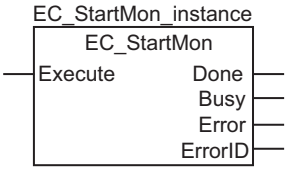
// EC_CoESDORead 命令実行
IF (DoSdoRead=TRUE) THEN
  EC_CoESDORead_instance(
    Execute:=TRUE,
    NodeAdr:=UINT#1, // ノードアドレス 1
    SdoObj :=SdoObject, // SDO パラメータ
    TimeOut:=UINT#20, // タイムアウト時間 2.0s
    ReadDat:=VersionInfo); // 読出データ

  IF (EC_CoESDORead_instance.Done=TRUE) THEN
    // 正常終了時処理
    NormalEnd:=NormalEnd+UINT#1;
  ELSIF (EC_CoESDORead_instance.Error=TRUE) THEN
    // 異常終了時処理
    ErrorEnd :=ErrorEnd+UINT#1;
  END_IF;

END_IF;
```

EC_StartMon

EtherCAT 通信の、パケットモニタ機能を実行開始します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_StartMon	EtherCAT パケットモニタ機能開始	FB		EC_StartMon_instance(Execute, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、Sysmac Studio のバージョンにより、以下のような制約があります。
- NY シリーズコントローラは、Sysmac Studio Ver.1.17 以降で使用可能です。

変数

共通の変数だけ

機能

EtherCAT 通信の、パケットモニタ機能を実行開始します。
パケットモニタ機能とは、EtherCAT 通信のパケットを、直近の一定数だけ収集する機能のことです。
収集数が一定数を超えると、古いパケットから順に破棄します。
本命令が実行されると、EC_StopMon 命令が実行されるまで、パケットモニタ機能を実行し続けます。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_PktMonStop	パケットモニタ停止中	BOOL	パケットモニタ機能が、停止しているかどうかを示します。 TRUE: 停止している FALSE: 停止していない
_EC_PktSaving	パケットデータファイル保存中	BOOL	パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに、保存中であるかどうかを示します。 TRUE: 保存中 FALSE: 保存中ではない

参考

- 本命令実行中は、収集したパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存することはできません。
- パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存する場合は、EC_StopMon 命令でパケットモニタ機能を停止させ、つぎに EC_SaveMon 命令で、保存を実行します。
- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a) パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存中の場合。
  b) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EtherCAT のスレーブ異常が発生した場合に、EtherCAT 通信のパケットを、SD メモリカードに保存します。ファイル名は、'PacketFile'とします。
処理手順は、以下のとおりです。

1 **EtherCAT 異常のシステム定義変数「_EC_ErrSta」を監視しておき、異常が発生したら処理を開始します。**

2 **EC_StopMon 命令を使って、EtherCAT 通信のパケットモニタ機能を停止します。**

3 **EC_SaveMon 命令を使って、EtherCAT 通信のパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存します。**

4 **そのファイルを、EC_CopyMon 命令を使って、SD メモリカードにコピーします。**

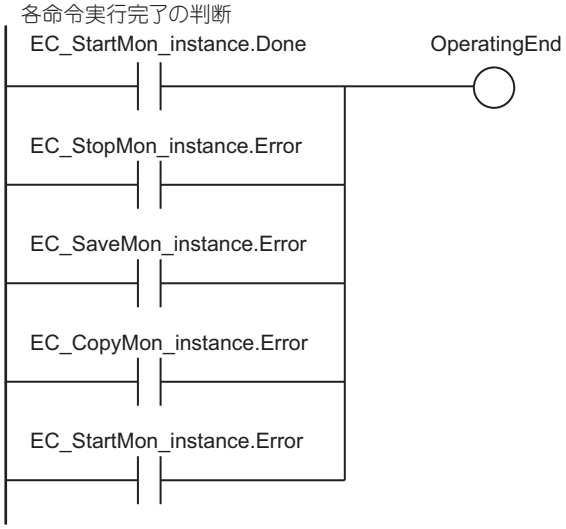
5 **EC_StartMon 命令を使って、再び EtherCAT 通信のパケットモニタ機能を実行開始します。**

LD

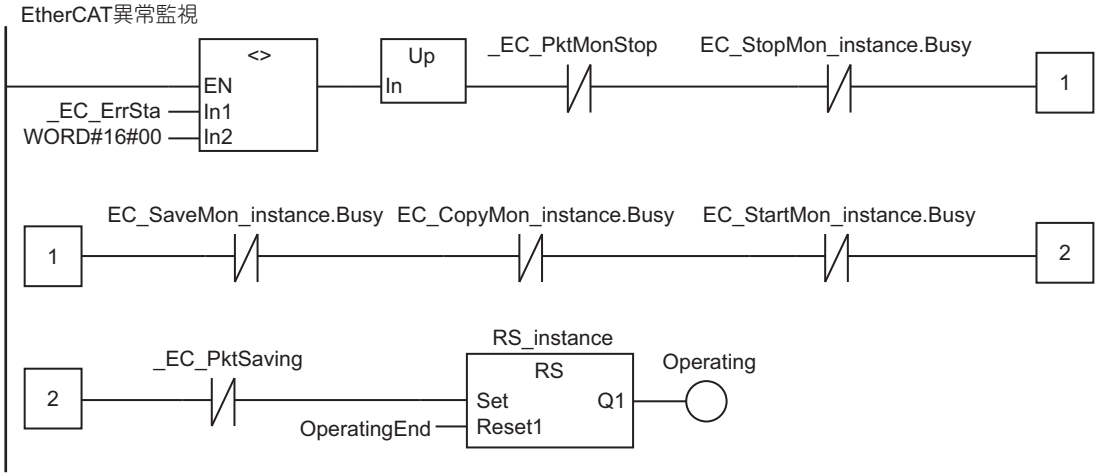
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Operating	BOOL	FALSE	実行条件
	RS_instance	RS		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

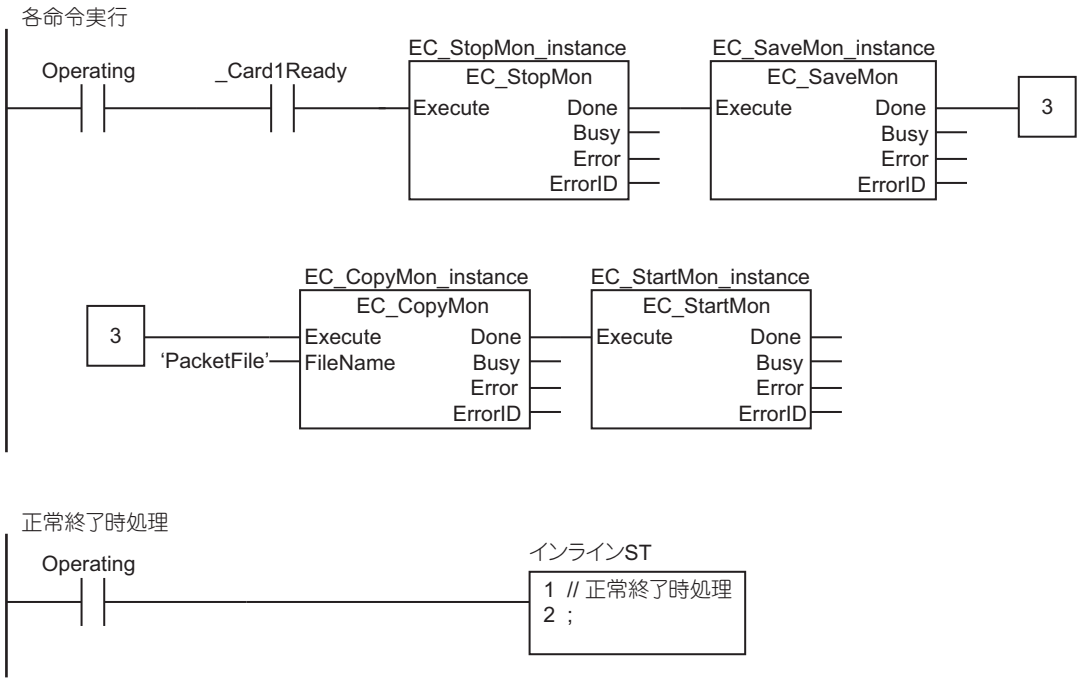
外部変数	名称	データ型	コンスタント	コメント
	_EC_ErrSta	WORD	☑	EtherCAT 異常
	_EC_PktMonStop	BOOL	☑	パケットモニタ停止中
	_EC_PktSaving	BOOL	☑	パケットデータファイル保存中
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ

各命令実行完了の判断



EtherCAT異常監視





ST

内部変数	名称	データ型	初期値	コメント
	EC_Err	BOOL	FALSE	EtherCAT マスタ機能モジュールのコントローラ異常発生
	EC_Err_Trigger	BOOL	FALSE	EC_Err の立ち上がり
	DoEC_PktSave	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	R_TRIG_instance	R_TRIG		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

外部変数	名称	データ型	コンスタント	コメント
	_EC_ErrSta	WORD	☑	EtherCAT 異常
	_EC_PktMonStop	BOOL	☑	パケットモニタ停止中
	_EC_PktSaving	BOOL	☑	パケットデータファイル保存中
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ

```
// _EC_ErrSta の立ち上がりでシーケンス開始
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); // インスタンス初期化
  EC_SaveMon_instance(Execute:=FALSE);
  EC_CopyMon_instance(Execute:=FALSE);
  EC_StartMon_instance(Execute:=FALSE);
END_IF;

// 各命令実行
IF (DoEC_PktSave=TRUE) THEN
  CASE Stage OF
  1 : // EtherCAT パケットモニタ機能停止
    EC_StopMon_instance(
      Execute:=TRUE);
    IF (EC_StopMon_instance.Done=TRUE) THEN
      Stage:=INT#2; // 正常終了
    ELSIF (EC_StopMon_instance.Error=TRUE) THEN
      Stage:=INT#10; // 異常終了
    END_IF;

  2 : // EtherCAT パケットデータを内部ファイルに保存
    EC_SaveMon_instance(
      Execute:=TRUE);
    IF (EC_SaveMon_instance.Done=TRUE) THEN
      Stage:=INT#3; // 正常終了
    ELSIF (EC_SaveMon_instance.Error=TRUE) THEN
      Stage:=INT#20; // 異常終了
    END_IF;

  3 : // EtherCAT パケットデータファイルを SD メモリカードにコピー
    EC_CopyMon_instance(
      Execute :=TRUE,
      FileName:='PacketFile');
    IF (EC_CopyMon_instance.Done=TRUE) THEN
      Stage:=INT#4; // 正常終了
    ELSIF (EC_CopyMon_instance.Error=TRUE) THEN
      Stage:=INT#30; // 異常終了
    END_IF;

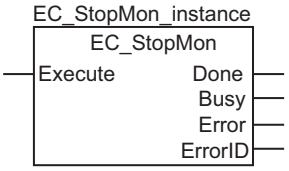
  4 : // EtherCAT パケットモニタ機能再開
    EC_StartMon_instance(
      Execute:=TRUE);
    IF (EC_StartMon_instance.Done=TRUE) THEN
      Stage:=INT#0; // 正常終了
    ELSIF (EC_StartMon_instance.Error=TRUE) THEN
      Stage:=INT#40; // 異常終了
    END_IF;

  0 : // 正常終了時処理
    DoEC_PktSave:=FALSE;
```

```
  ELSE // 異常終了時処理
    DoEC_PktSave:=FALSE;
  END_CASE;
END_IF;
```

EC_StopMon

EtherCAT 通信の、パケットモニタ機能を停止します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_StopMon	EtherCAT パケットモニタ機能停止	FB		EC_StopMon_instance(Execute, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、Sysmac Studio のバージョンにより、以下のような制約があります。
- NY シリーズコントローラは、Sysmac Studio Ver.1.17 以降で使用可能です。

変数

共通の変数だけ

機能

EtherCAT 通信の、パケットモニタ機能を停止します。
パケットモニタ機能とは、EtherCAT 通信のパケットを、直近の一定数だけ収集する機能のことです。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_PktMonStop	パケットモニタ停止中	BOOL	パケットモニタ機能が、停止しているかどうかを示します。 TRUE: 停止している FALSE: 停止していない
_EC_PktSaving	パケットデータファイル保存中	BOOL	パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに、保存中であるかどうかを示します。 TRUE: 保存中 FALSE: 保存中ではない

参考

- 収集したパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存する場合は、本命令でパケットモニタ機能を停止させておいてから、EC_SaveMon 命令で保存を実行します。
- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。

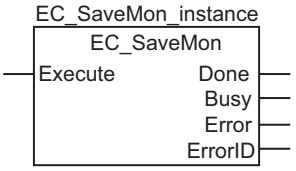
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a) パケットモニタ機能が、すでに停止状態の場合。
  b) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.2-905)」を参照してください。

EC_SaveMon

EtherCAT 通信のパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_SaveMon	EtherCAT パケット保存	FB		EC_SaveMon_instance(Execute, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、Sysmac Studio のバージョンにより、以下のような制約があります。
- NY シリーズコントローラは、Sysmac Studio Ver.1.17 以降で使用可能です。

変数

共通の変数だけ

機能

EtherCAT 通信の、パケットモニタ機能を使って収集したパケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存します。
パケットモニタ機能とは、EtherCAT 通信のパケットを、直近の一定数だけ収集する機能のことです。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_PktMonStop	パケットモニタ停止中	BOOL	パケットモニタ機能が、停止しているかどうかを示します。 TRUE: 停止している FALSE: 停止していない
_EC_PktSaving	パケットデータファイル保存中	BOOL	パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに、保存中であるかどうかを示します。 TRUE: 保存中 FALSE: 保存中ではない

参考

- 本命令実行中は、パケットモニタ機能を実行できません。
- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。

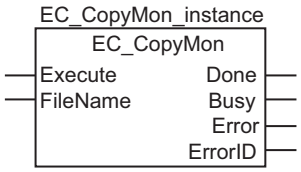
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- 本命令は、パケットモニタ機能実行中には使用できません。あらかじめ EC_StopMon 命令を使用して、パケットモニタ機能を停止させておいてください。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a) パケットモニタ機能が、実行中の場合。
  b) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.2-905)」を参照してください。

EC_CopyMon

EtherCAT 通信の、CPU ユニットのメインメモリ上の内部ファイルにあるパケットデータを、SD メモリカードに転送します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_CopyMon	EtherCAT パケット転送	FB		EC_CopyMon_instance(Execute, FileName, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、Sysmac Studio のバージョンにより、以下のような制約があります。
・ NY シリーズコントローラは、Sysmac Studio Ver.1.17 以降で使用可能です。

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	ファイル名	入力	SD メモリカード内のファイル名	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○

機能

EtherCAT 通信の、CPU ユニットのメインメモリ上の内部ファイルにあるパケットデータを、SD メモリカードに転送します。
SD メモリカード内のファイル名を、「FileName」で指定します。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_PktSaving	パケットデータファイル保存中	BOOL	パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに、保存中であるかどうかを示します。 TRUE: 保存中 FALSE: 保存中ではない

参考

EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

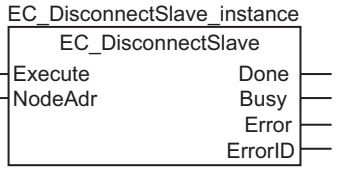
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- 本命令は、パケットモニタ保存実行中には使用できません。
- 本命令を使用する前に、あらかじめ EC_SaveMon 命令を使用して、パケットデータを、CPU ユニットのメインメモリ上の内部ファイルに保存しておいてください。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a) パケットデータファイル保存中の場合。
  b) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.2-905)」を参照してください。

EC_DisconnectSlave

EtherCAT ネットワーク上の、指定したスレーブを、ネットワークから離脱させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_DisconnectSlave	EtherCAT スレーブ離脱	FB		EC_DisconnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
NodeAdr	スレーブノードアドレス	入力	離脱させるスレーブのノードアドレス	1 ～ 512	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							○													

機能

スレーブノードアドレス「NodeAdr」で指定したスレーブを、EtherCAT ネットワークから離脱させます。
ネットワークからの離脱とは、スレーブが、ネットワーク上に存在しているけれど動作しない状態にすることを意味します。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_EntrySlavTbl[i] “i”は、ノードアドレス	ネットワーク加入スレーブテーブル	BOOL[]	スレーブが、ネットワークに加入（存在）しているかどうかを示します。 TRUE: 加入（存在）している FALSE: 加入（存在）していない
_EC_DisconnSlavTbl[i] “i”は、ノードアドレス	離脱指令中スレーブテーブル	BOOL[]	スレーブが、離脱指令中かどうかを示します。 TRUE: 離脱指令中 FALSE: 離脱指令中ではない
_EC_DisableSlavTbl[i] “i”は、ノードアドレス	無効設定スレーブテーブル	BOOL[]	スレーブが、無効設定されているかどうかを示します。 TRUE: 無効設定されている FALSE: 有効またはネットワークに加入（存在）していない

参考

EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- 離脱させたスレーブの後続に、ディジーチェーン接続（Out ポートに接続）されているスレーブが存在する場合、そのスレーブも EtherCAT ネットワークから離脱します。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a）「NodeAdr」で指定したスレーブが、EtherCAT ネットワークに加入（存在）していない場合。つまり、ネットワーク加入スレーブテーブル_EC_EntrySlavTbl[i]の値が FALSE の場合。
  b）「NodeAdr」で指定したスレーブが、無効設定されている場合。
  c）EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令のいずれかが実行中の場合。
  d）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

スレーブ 1 を、EtherCAT ネットワークから離脱させたり、再加入させたりします。
「Trigger1」が入力されたら EC_DisconnectSlave 命令を使って離脱、「Trigger2」が入力されたら EC_ConnectSlave 命令を使って再加入させます。

命令の排他制御

EC_DisconnectSlave 命令と EC_ConnectSlave 命令は、同時に実行することができません。また、どちらの命令も、複数のタスク周期にまたがって実行されます。
そこで、先に実行した方の命令の完了を確認してから、つぎの命令を実行します。
そのために、「ExclusiveFlg（命令排他フラグ）」という変数を使用します。
「ExclusiveFlg」の値が TRUE ならば、どちらかの命令が実行中であることを意味します。
「ExclusiveFlg」の値が TRUE である間は、つぎの命令を実行しません。

EC_DisconnectSlave 命令と EC_ConnectSlave 命令は、別のタスク内にあるそれらの命令とも、同時に実行することができません。
そこで、本サンプルプログラムでは「ExclusiveFlg」をグローバル変数にしています。
こうすることで、本サンプルプログラムとは別のタスク内の命令との排他制御が行えます。ただしその場合、別のタスク内でも、同じグローバル変数「ExclusiveFlg」を使って命令の排他制御を行う必要があります。
また、EC_DisconnectSlave 命令と EC_ConnectSlave 命令は、EC_ChangeEnableSetting 命令とも同時に実行することができません。
EC_ChangeEnableSetting 命令の「サンプルプログラム (P.2-927)」は、本サンプルプログラムと同じグローバル変数「ExclusiveFlg」を使用して、命令の排他制御を行う例になっています。

グローバル変数の定義

● グローバル変数

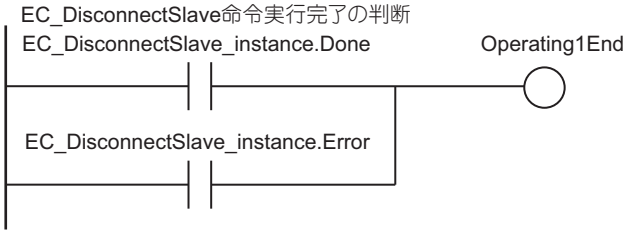
名称	データ型	初期値	コメント
ExclusiveFlg	BOOL	FALSE	命令排他フラグ

LD

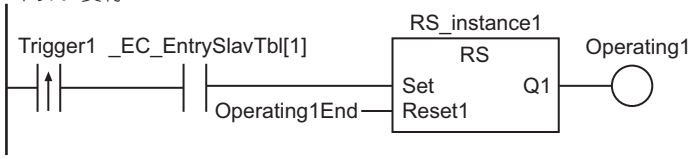
内部変数	名称	データ型	初期値	コメント
	Operating1End	BOOL	FALSE	処理終了 1
	Trigger1	BOOL	FALSE	実行条件 1
	Operating1	BOOL	FALSE	処理中 1
	RS_instance1	RS		
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Operating2End	BOOL	FALSE	処理終了 2
	Trigger2	BOOL	FALSE	実行条件 2
	Operating2	BOOL	FALSE	処理中 2
	RS_instance2	RS		
	EC_ConnectSlave_instance	EC_ConnectSlave		
	HMI_ConnectErrorID *1	WORD	16#0000	

*1. 「HMI_」で始まる名称はタッチパネル表示用の変数です。

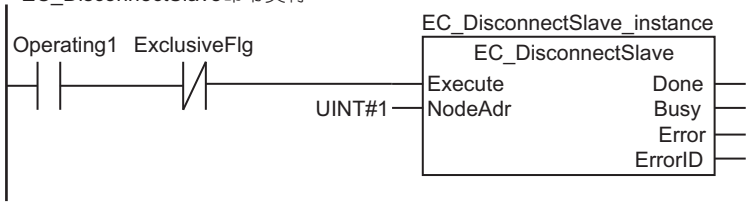
外部変数	名称	データ型	コンスタント	コメント
	_EC_EntrySlavTbl	ARRAY[1..512] OF BOOL	☑	ネットワーク加入スレーブテーブル
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL	☑	離脱指令中スレーブテーブル
	ExclusiveFlg	BOOL	☐	命令排他フラグ



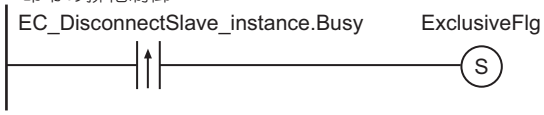
トリガ1受付

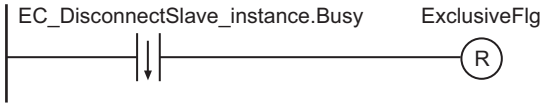


EC_DisconnectSlave命令実行

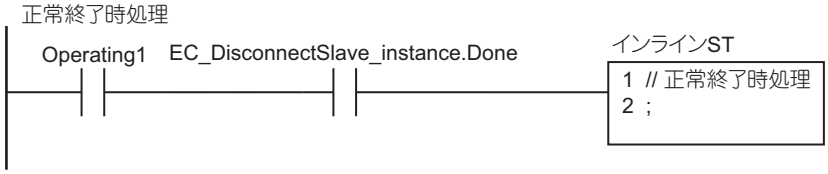


命令の排他制御

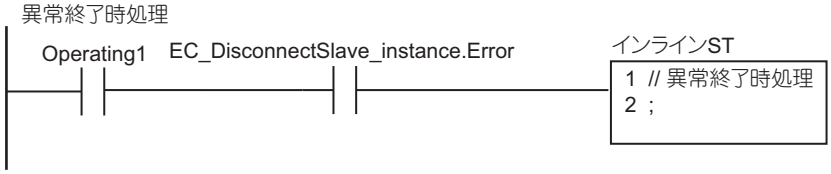




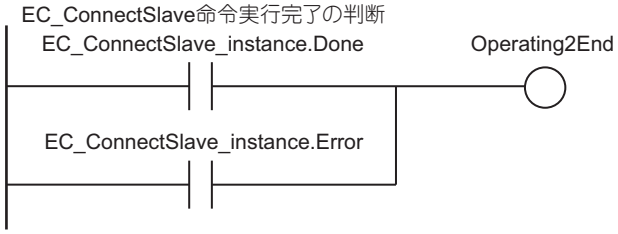
正常終了時処理



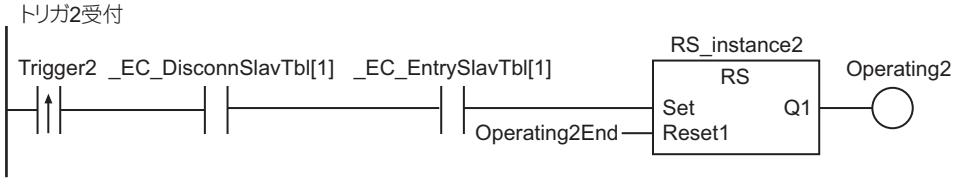
異常終了時処理

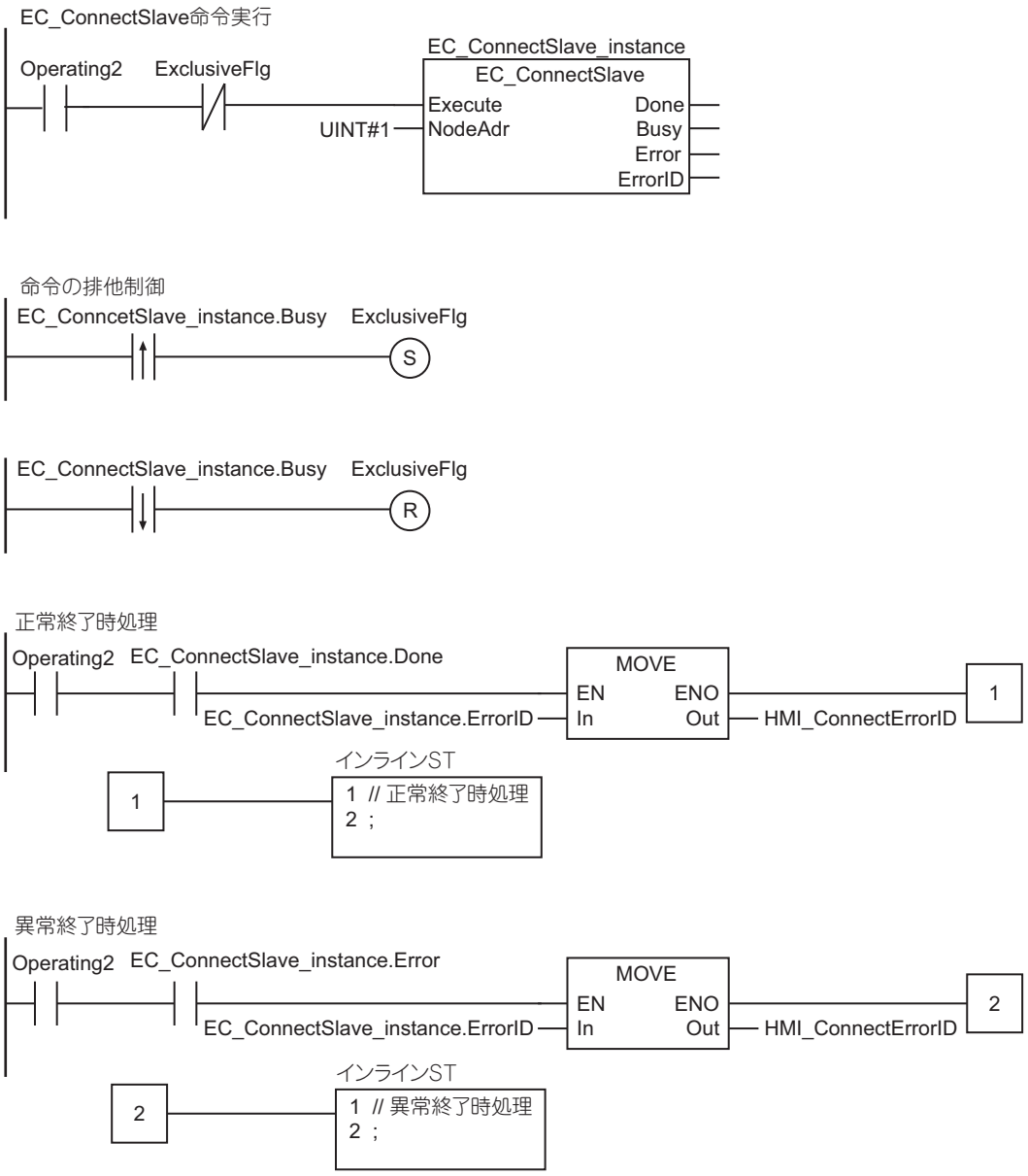


EC_ConnectSlave命令実行完了の判断



トリガ2受付





ST

内部変数	名称	データ型	初期値	コメント
	Trigger1	BOOL	FALSE	実行条件 1
	LastTrigger1	BOOL	FALSE	前タスク周期の「Trigger1」の値
	Operating1Start	BOOL	FALSE	処理開始 1
	Operating1	BOOL	FALSE	処理中 1
	DisconnectSet	BOOL	FALSE	EC_DisconnectSlave 命令実行中
	DisconnectReset	BOOL	FALSE	EC_DisconnectSlave 命令非実行中
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Trigger2	BOOL	FALSE	実行条件 2
	LastTrigger2	BOOL	FALSE	前タスク周期の「Trigger2」の値

内部変数	名称	データ型	初期値	コメント
	Operating2Start	BOOL	FALSE	処理開始 2
	Operating2	BOOL	FALSE	処理中 2
	ConnectSet	BOOL	FALSE	EC_ConnectSlave 命令実行中
	ConnectReset	BOOL	FALSE	EC_ConnectSlave 命令非実行中
	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. 「HMI_」で始まる名称はタッチパネル表示用の変数です。

外部変数	名称	データ型	コンスタント	コメント
	_EC_EntrySlavTbl	ARRAY[1..512] OF BOOL	☑	ネットワーク加入スレーブテーブル
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL	☑	離脱指令中スレーブテーブル
	ExclusiveFlg	BOOL		命令排他フラグ

```
// Trigger1 の立ち上がり検出
IF ( (Trigger1=TRUE) AND (LastTrigger1=FALSE) AND (_EC_EntrySlavTbl[1]=TRUE) ) THEN
  Operating1Start:=TRUE;
  Operating1 :=TRUE;
END_IF;
LastTrigger1:=Trigger1;

// EC_DisconnectSlave 命令初期化
IF (Operating1Start=TRUE) THEN
  EC_DisconnectSlave_instance(Execute:=FALSE);
  Operating1Start:=FALSE;
END_IF;

// EC_DisconnectSlave 命令実行
IF (Operating1=TRUE) THEN
  EC_DisconnectSlave_instance(
    Execute:=NOT(ExclusiveFlg),
    NodeAdr:=UINT#1);

  // 命令の排他制御
  R_TRIG_instance1(EC_DisconnectSlave_instance.Busy, DisconnectSet);
  F_TRIG_instance1(EC_DisconnectSlave_instance.Busy, DisconnectReset);
  RS_instance1(DisconnectSet, DisconnectReset, ExclusiveFlg);

  IF (EC_DisconnectSlave_instance.Done=TRUE) THEN
```

```
    // 正常終了時処理
    Operating1:=FALSE;
  END_IF;

  IF (EC_DisconnectSlave_instance.Error=TRUE) THEN
    // 異常終了時処理
    Operating1:=FALSE;
  END_IF;
END_IF;

// Trigger2 の立ち上がり検出
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;

// EC_ConnectSlave 命令初期化
IF (Operating2Start=TRUE) THEN
  EC_ConnectSlave_instance(Execute:=FALSE);
  Operating2Start:=FALSE;
END_IF;

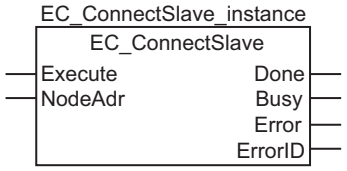
// EC_ConnectSlave 命令実行
IF (Operating2=TRUE) THEN
  EC_ConnectSlave_instance(
    Execute:=NOT(ExclusiveFlg),
  NodeAdr:=UINT#1);

  // 命令の排他制御
  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
    // 正常終了時処理
    HMI_ConnectErrorID:=EC_ConnectSlave_instance.ErrorID;
    Operating2:=FALSE;
  END_IF;

  IF (EC_ConnectSlave_instance.Error=TRUE) THEN
    // 異常終了時処理
    HMI_ConnectErrorID:=EC_ConnectSlave_instance.ErrorID;
    Operating2:=FALSE;
  END_IF;
END_IF;
```

EC_ConnectSlave

EtherCAT ネットワーク上の、指定したスレーブを、ネットワークに再加入させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_ConnectSlave	EtherCAT スレーブ再加入	FB		EC_ConnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
NodeAdr	スレーブノードアドレス	入力	登録するスレーブのノードアドレス	0 *1～ 512	−	−

*1. 「0」は、ネットワーク設定に登録されている全スレーブを意味します。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							○													

機能

スレーブノードアドレス「NodeAdr」で指定したスレーブを、EtherCAT ネットワークに再加入させます。
ネットワークへの再加入とは、スレーブがネットワーク上に存在し、かつ動作する状態にすることを意味します。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_EntrySlavTbl[i] “i”は、ノードアドレス	ネットワーク加入スレーブテーブル	BOOL[]	スレーブが、ネットワークに加入（存在）しているかどうかを示します。 TRUE: 加入（存在）している FALSE: 加入（存在）していない
_EC_DisconnSlavTbl[i] “i”は、ノードアドレス	離脱指令中スレーブテーブル	BOOL[]	スレーブが、離脱指令中かどうかを示します。 TRUE: 離脱指令中 FALSE: 離脱指令中ではない

参考

EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

使用上の注意

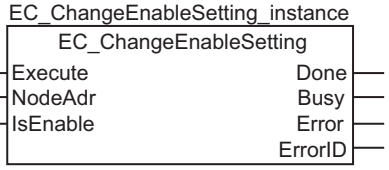
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a）「NodeAdr」で指定したスレーブが、EtherCAT ネットワークに加入（存在）していない場合。つまり、ネットワーク加入スレーブテーブル_EC_EntrySlavTbl[i]の値が FALSE の場合。
  b）EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令のいずれかが実行中の場合。
  c）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

EC_DisconnectSlave 命令の「サンプルプログラム (P.2-917)」を、参照してください。

EC_ChangeEnableSetting

EtherCAT スレーブの、有効／無効切替を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	FB		EC_ChangeEnableSetting_instance(Execute, NodeAdr, IsEnable, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
NodeAdr	スレーブノードアドレス	入力	有効／無効切替を行うEtherCAT スレーブのノードアドレス	1～512	–	1
IsEnable	有効／無効切替指示		指定した EtherCAT スレーブを有効にするか無効にするかの指示 TRUE: 有効 FALSE: 無効	データ型に従う		TRUE

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							○													
IsEnable	○																			

機能

スレーブノードアドレス「NodeAdr」で指定した EtherCAT スレーブに対して、有効／無効の切替を行います。
有効／無効切替指示「IsEnable」の値が TRUE なら有効になり、FALSE なら無効になります。
本命令が正常終了すると、「Done」の値が TRUE に変化します。
本命令が正常終了した時点で、有効／無効切替が完了しています。

本命令は、指定した EtherCAT スレーブの状態が、有効か無効か、加入中か離脱中か、EtherCAT ネットワーク内に存在しているかしていないかによって、正常終了したり異常終了したりします。

本命令実行前の EtherCAT スレーブの各状態に対する、本命令実行後の状態は、以下のとおりです。

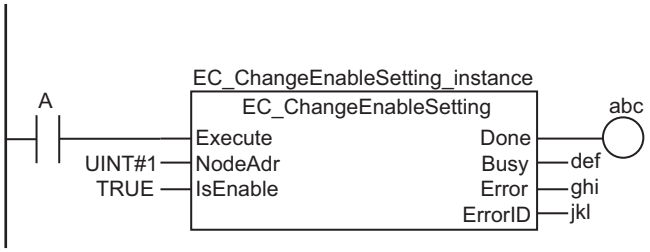
本命令実行前の状態			「IsEnable」の値	本命令実行後の状態	
有効／無効	加入中／離脱中	存在 *1		正常終了／異常終了	有効／無効
有効	加入中	○	TRUE（有効）	正常終了	有効
	離脱中	○		異常終了 *2	有効
		×			
無効	— *3	○		正常終了	有効
		×		異常終了 *4	無効 *4
有効	加入中	○	FALSE（無効）	正常終了	無効
	離脱中	○		異常終了 *2	有効
		×			
無効	— *3	○		正常終了	無効
		×		異常終了 *4	無効 *4

*1. 指定した EtherCAT スレーブが、EtherCAT ネットワーク内に物理的に取り付けられているか、取り外されているかを示します。
○: 取り付けられている。×: 取り外されている。
*2. エラーコード 1800 が返ります。
*3. 無効設定されている EtherCAT スレーブには、加入中／離脱中の区別はありません。
*4. 異常終了し、有効／無効の状態は実行前のままとなり、エラーコード 1801 が返ります。

● 使用例

ノードアドレス 1 の EtherCAT スレーブを有効にする場合の例を、以下に示します。
「NodeAdr」には UINT#1 を、「IsEnable」には TRUE を指定します。

LD



ST

```
EC_ChangeEnableSetting_instance(A, UINT#1,
                                TRUE, abc, def, ghi, jkl);
```

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_EntrySlavTbl[i] “i”は、ノードアドレス	ネットワーク加入スレーブテーブル	BOOL[]	スレーブが、ネットワークに加入（存在）しているかどうかを示します。 TRUE: 加入（存在）している FALSE: 加入（存在）していない
_EC_DisconnSlavTbl[i] “i”は、ノードアドレス	離脱指令中スレーブテーブル	BOOL[]	スレーブが、離脱指令中かどうかを示します。 TRUE: 離脱指令中 FALSE: 離脱指令中ではない
_EC_DisableSlavTbl[i] “i”は、ノードアドレス	無効設定スレーブテーブル	BOOL[]	スレーブが、無効設定されているかどうかを示します。 TRUE: 無効設定されている FALSE: 有効またはネットワークに加入（存在）していない

参考

- EtherCAT 通信の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherCAT ポート(SBCD-368)』を、参照してください。

- EtherCAT スレーブを、EtherCAT ネットワークに再加入させるには、「EC_ConnectSlave (P.2-923)」命令を使用してください。

使用上の注意

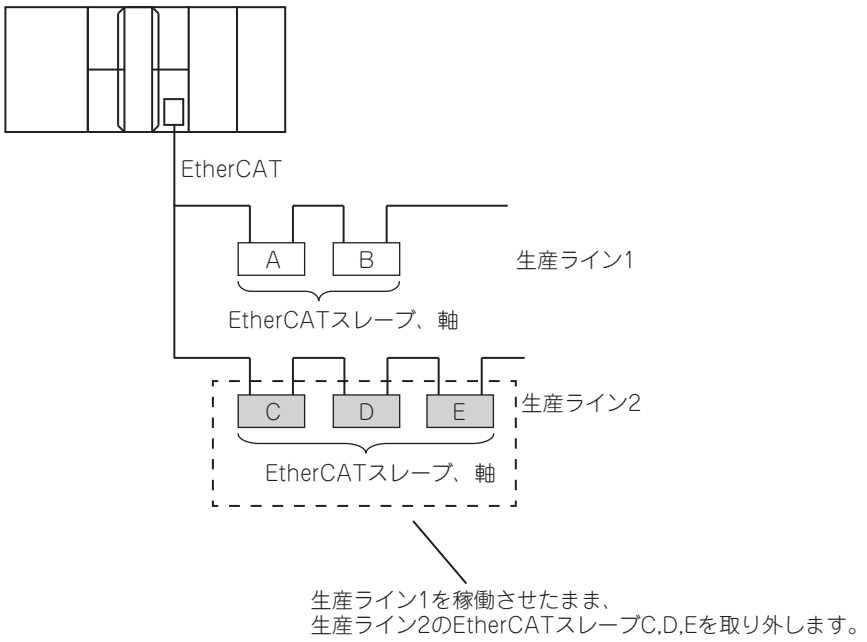
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令を実行中に、本命令を実行することはできません。
- 本命令の実行結果は、CPU ユニットの不揮発性メモリに保存されません。したがって、本命令実行後に CPU ユニットの電源を再投入したり、ユーザプログラムのダウンロードを実行したりすると、EtherCAT スレーブの有効／無効設定は、Sysmac Studio で設定したものに戻ります。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a）「NodeAdr」で指定したスレーブが、EtherCAT ネットワークに加入（存在）していない場合。つまり、ネットワーク加入スレーブテーブル_EC_EntrySlavTbl[i]の値が、FALSE の場合。
  b）「NodeAdr」の値が、有効範囲外の場合。
  c）EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、ResetECError 命令、RestartNXUnit 命令、NX_ChangeWriteMode 命令のいずれかが実行中の場合。
  d）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。

サンプルプログラム

ここでは、以下の 2 つのサンプルプログラムについて説明します。
- EtherCAT スレーブを EtherCAT ネットワークから取り外す例
- EtherCAT スレーブを EtherCAT ネットワークに取り付ける例

EtherCAT スレーブを EtherCAT ネットワークから取り外す例

下図に示すシステムの生産ライン 1 を稼働させたまま、生産ライン 2 の EtherCAT スレーブ C,D,E を取り外します。
EtherCAT スレーブ C,D,E には、モーション制御の軸も設定されているとします。したがって、EtherCAT スレーブを無効化するとともに、軸使用を未使用軸化する処理も行います。



● 操作手順

本サンプルプログラムの操作手順は、以下のとおりです。

1 **操作者が表示器のボタンを押すと、実行条件が ON になります。**

2 **コントローラが、EtherCAT スレーブ C,D,E を無効化します。また、それぞれの軸使用を未使用軸に設定します。**

3 **3 つの EtherCAT スレーブの無効化、未使用軸化がすべて完了したら、コントローラが取り外し可能ランプを点灯させます。**

4 **操作者が取り外し可能ランプの点灯を確認した後、3 つの EtherCAT スレーブを取り外します。**

● 軸使用を未使用軸化する命令

軸使用を未使用軸化するために、MC_ChangeAxisUse 命令を使用します。
MC_ChangeAxisUse 命令の仕様の詳細は、『NY シリーズ コマンドリファレンスマニュアル モーション編(SBCE-380)』を、参照してください。

● 命令の排他制御

EC_ChangeEnableSetting 命令は、複数同時に実行することができません。
また、EC_ChangeEnableSetting 命令は、複数のタスク周期にまたがって実行されます。
そこで、先に実行した EC_ChangeEnableSetting 命令の完了を確認してから、つぎの EC_ChangeEnableSetting 命令を実行します。
そのために、「ExclusiveFlg（命令排他フラグ）」という変数を使用します。
「ExclusiveFlg」の値が TRUE ならば、EC_ChangeEnableSetting 命令が実行中であることを意味します。
「ExclusiveFlg」の値が TRUE である間は、つぎの EC_ChangeEnableSetting 命令を実行しません。

EC_ChangeEnableSetting 命令は、別のタスク内の EC_ChangeEnableSetting 命令とも、同時に実行することができません。
そこで、本サンプルプログラムでは「ExclusiveFlg」をグローバル変数にしています。

こうすることで、本サンプルプログラムとは別のタスク内の EC_ChangeEnableSetting 命令との排他制御が行えます。
ただしその場合、別のタスク内でも、同じグローバル変数「ExclusiveFlg」を使って命令の排他制御を行う必要があります。
また、EC_ChangeEnableSetting 命令は、EC_DisconnectSlave 命令や EC_ConnectSlave 命令とも同時に実行することができません。
EC_DisconnectSlave 命令の「サンプルプログラム (P.2-917)」は、本サンプルプログラムと同じグローバル変数「ExclusiveFlg」を使用して、命令の排他制御を行う例になっています。

● 各 EtherCAT スレーブの軸変数とノードアドレス

EtherCAT スレーブ C,D,E の軸に割り付けられた軸変数と、ノードアドレスは、以下のとおりとします。

EtherCAT スレーブ	軸変数	ノードアドレス
C	MC_Axis000	1
D	MC_Axis001	2
E	MC_Axis002	3

● グローバル変数

名称	データ型	初期値	割付先	コンスタント	コメント
MC_Axis000	_sAXIS_REF		MC://_MC_AX[0]	☑	EtherCAT スレーブ C の軸変数
MC_Axis001	_sAXIS_REF		MC://_MC_AX[1]	☑	EtherCAT スレーブ D の軸変数
MC_Axis002	_sAXIS_REF		MC://_MC_AX[2]	☑	EtherCAT スレーブ E の軸変数
ExclusiveFlg	BOOL	FALSE		☐	命令排他フラグ

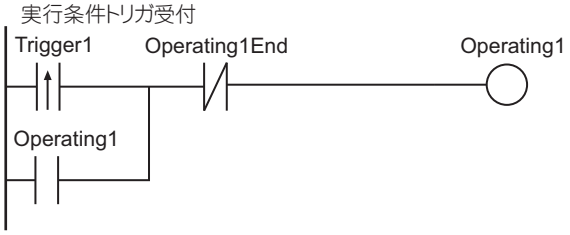
● LD

内部変数	名称	データ型	初期値	コメント
	Operating1End	BOOL	FALSE	処理終了
	Trigger1	BOOL	FALSE	実行条件
	Operating1	BOOL	FALSE	処理中
	AxisUnuseDone_DevC	BOOL	FALSE	EtherCAT スレーブ C の軸使用の未使用軸化完了
	SlaveDisableDone_DevC	BOOL	FALSE	EtherCAT スレーブ C の無効化完了
	DoneHold_DevC	BOOL	FALSE	EtherCAT スレーブ C の処理完了の保持
	AxisUnuseDone_DevD	BOOL	FALSE	EtherCAT スレーブ D の軸使用の未使用軸化完了
	SlaveDisableDone_DevD	BOOL	FALSE	EtherCAT スレーブ D の無効化完了
	DoneHold_DevD	BOOL	FALSE	EtherCAT スレーブ D の処理完了の保持
	AxisUnuseDone_DevE	BOOL	FALSE	EtherCAT スレーブ E の軸使用の未使用軸化完了
	SlaveDisableDone_DevE	BOOL	FALSE	EtherCAT スレーブ E の無効化完了
	DoneHold_DevE	BOOL	FALSE	EtherCAT スレーブ E の処理完了の保持

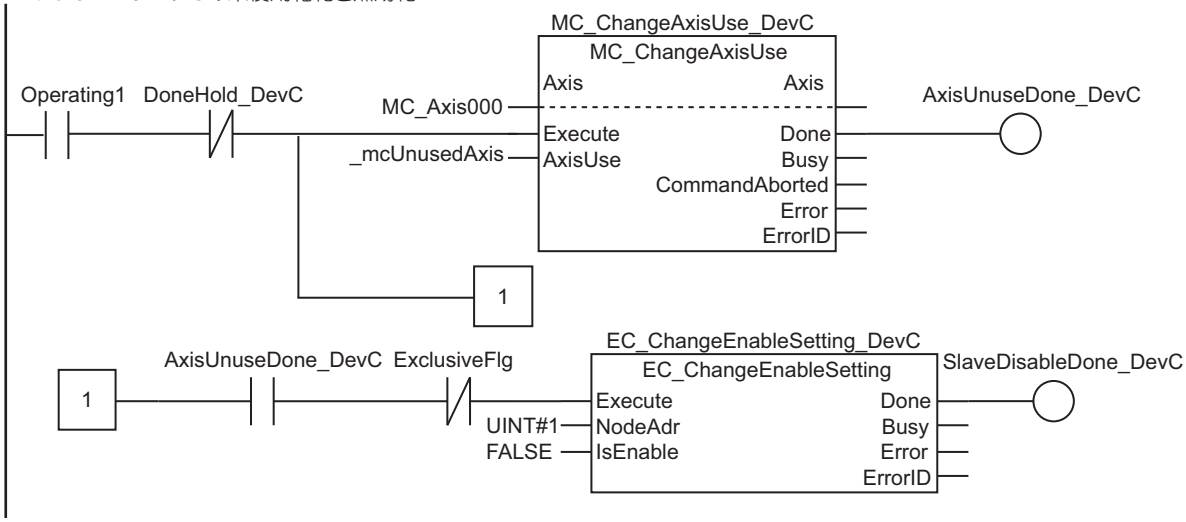
内部変数	名称	データ型	初期値	コメント
	Light1On	BOOL	FALSE	取り外し可能ランプ点灯
	MC_ChangeAxisUse_DevC	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevC	EC_ChangeEnableSetting		
	MC_ChangeAxisUse_DevD	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevD	EC_ChangeEnableSetting		
	MC_ChangeAxisUse_DevE	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevE	EC_ChangeEnableSetting		

外部変数	名称	データ型	コンスタント	コメント
	MC_Axis000	_sAXIS_REF	☑	EtherCAT スレーブ C の軸変数
	MC_Axis001	_sAXIS_REF	☑	EtherCAT スレーブ D の軸変数
	MC_Axis002	_sAXIS_REF	☑	EtherCAT スレーブ E の軸変数
	ExclusiveFlg	BOOL	☐	命令排他フラグ

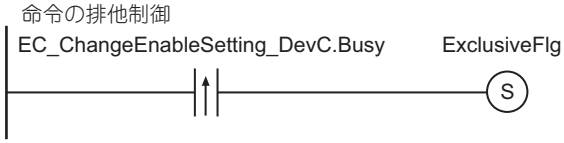
実行条件トリガ受付

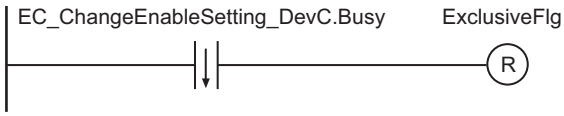


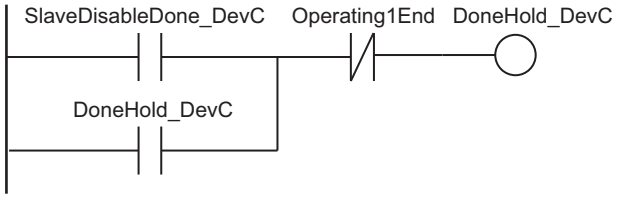
EtherCATスレーブCの未使用軸化と無効化



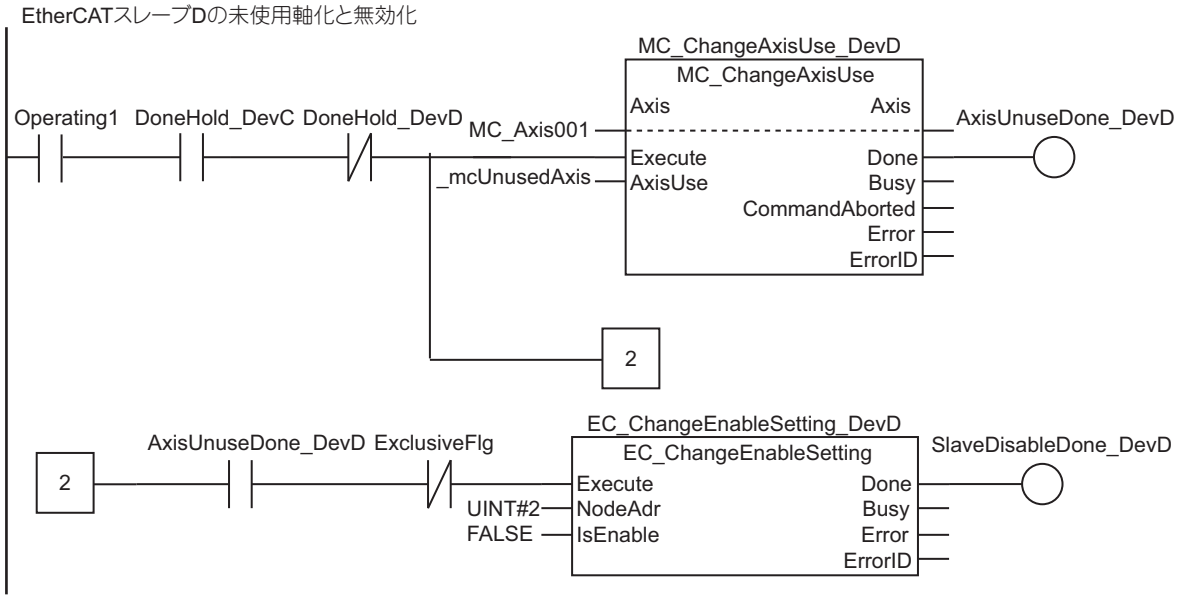
命令の排他制御



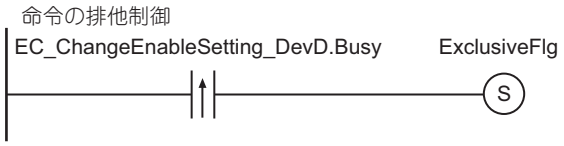


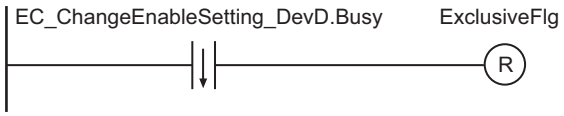


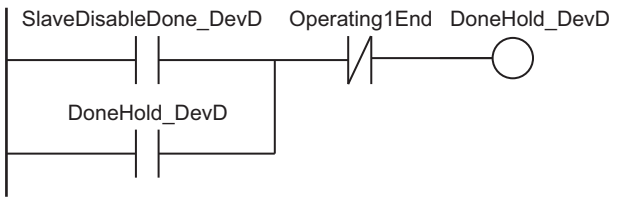
EtherCATスレーブDの未使用軸化と無効化

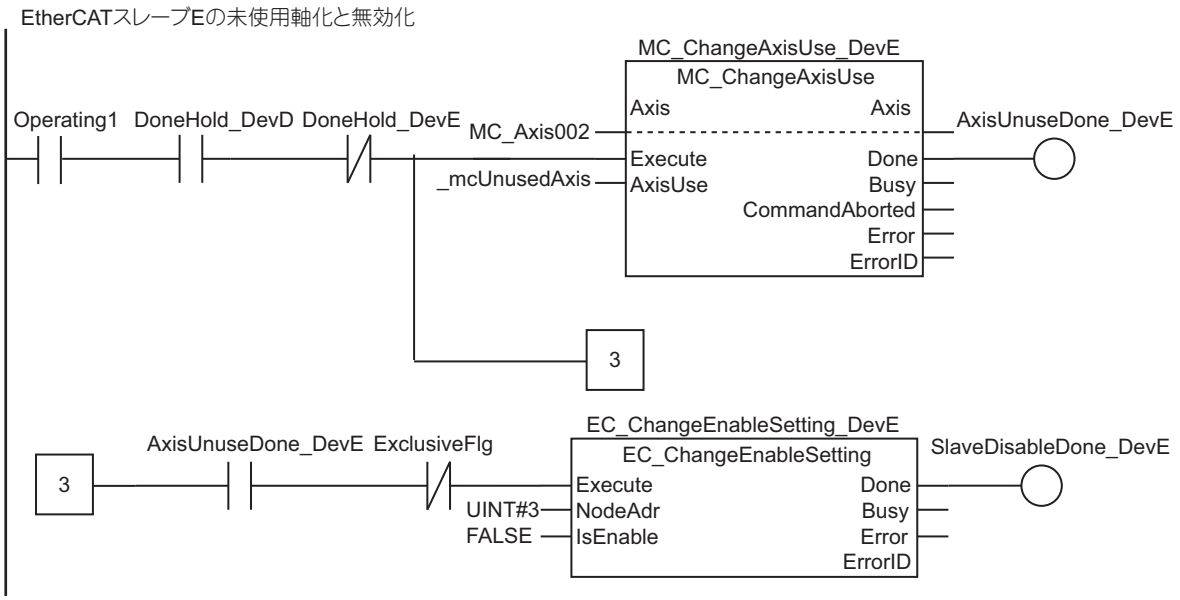


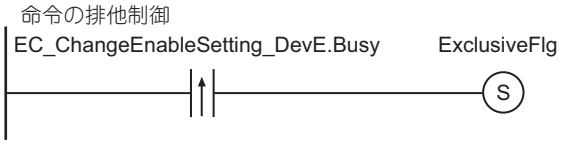
命令の排他制御

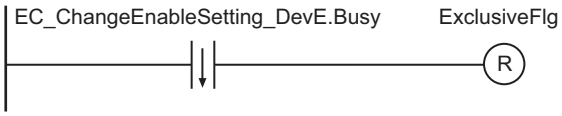


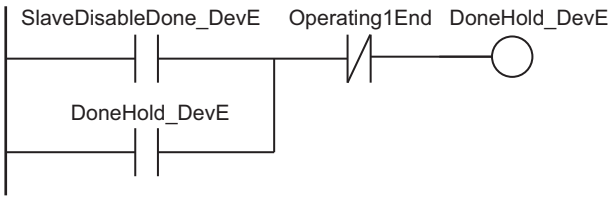




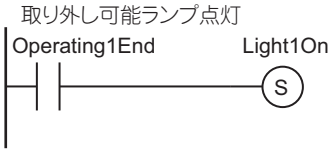












● ST

内部変数	名称	データ型	初期値	コメント
	Operating1End	BOOL	FALSE	処理終了
	Trigger1	BOOL	FALSE	実行条件
	Operating1	BOOL	FALSE	処理中

内部変数	名称	データ型	初期値	コメント
	Operating1Set	BOOL	FALSE	処理開始
	Light1On	BOOL	FALSE	取り外し可能ランプ点灯
	DoneHold_DevC	BOOL	FALSE	EtherCAT スレーブ C の処理完了の保持
	DoneHold_DevD	BOOL	FALSE	EtherCAT スレーブ D の処理完了の保持
	DoneHold_DevE	BOOL	FALSE	EtherCAT スレーブ E の処理完了の保持
	ExclusiveFlgSet	BOOL	FALSE	命令排他フラグ ON
	ExclusiveFlgReset	BOOL	FALSE	命令排他フラグ OFF
	R_TRIG_instance1	R_TRIG		
	RS_instance1	RS		
	SR_instance1	SR		
	MC_ChangeAxisUse_DevC	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevC	EC_ChangeEnableSetting		
	R_TRIG_DevC	R_TRIG		
	F_TRIG_DevC	F_TRIG		
	RS_ExFlg_DevC	RS		
	RS_DevC	RS		
	MC_ChangeAxisUse_DevD	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevD	EC_ChangeEnableSetting		
	R_TRIG_DevD	R_TRIG		
	F_TRIG_DevD	F_TRIG		
	RS_ExFlg_DevD	RS		
	RS_DevD	RS		
	MC_ChangeAxisUse_DevE	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevE	EC_ChangeEnableSetting		
	R_TRIG_DevE	R_TRIG		
	F_TRIG_DevE	F_TRIG		
	RS_ExFlg_DevE	RS		
	RS_DevE	RS		

外部変数	名称	データ型	コンスタント	コメント
	MC_Axis000	_sAXIS_REF	☑	EtherCAT スレーブ C の軸変数
	MC_Axis001	_sAXIS_REF	☑	EtherCAT スレーブ D の軸変数
	MC_Axis002	_sAXIS_REF	☑	EtherCAT スレーブ E の軸変数
	ExclusiveFlg	BOOL	☐	命令排他フラグ

```
// 実行条件トリガ受付
R_TRIG_instance1(Trigger1, Operating1Set);
RS_instance1(
  Set :=Operating1Set,
```

```
  Reset1:=Operating1End,
  Q1 =>Operating1);

// EtherCAT スレーブ C の軸使用の未使用軸化
MC_ChangeAxisUse_DevC(
  Axis :=MC_Axis000,
  Execute:=(Operating1 & NOT(DoneHold_DevC)),
  AxisUse:=_mcUnusedAxis);

// EtherCAT スレーブ C の無効化
EC_ChangeEnableSetting_DevC(
  Execute :=(Operating1 & MC_ChangeAxisUse_DevC.Done & NOT(ExclusiveFlg)),
  NodeAdr :=UINT#1,
  IsEnable:=FALSE);

// 命令の排他制御
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);

// EtherCAT スレーブ D の軸使用の未使用軸化
MC_ChangeAxisUse_DevD(
  Axis :=MC_Axis001,
  Execute:=(Operating1 & DoneHold_DevC & NOT(DoneHold_DevD)),
  AxisUse:=_mcUnusedAxis);

// EtherCAT スレーブ D の無効化
EC_ChangeEnableSetting_DevD(
  Execute :=(Operating1 & DoneHold_DevC & MC_ChangeAxisUse_DevD.Done & NOT(ExclusiveFlg)),
  NodeAdr :=UINT#2,
  IsEnable:=FALSE);

// 命令の排他制御
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);
```

```
// EtherCAT スレーブ E の軸使用の未使用軸化
MC_ChangeAxisUse_DevE(
  Axis :=MC_Axis002,
  Execute:=(Operating1 & DoneHold_DevD & NOT(DoneHold_DevE)),
  AxisUse:=_mcUnusedAxis);

// EtherCAT スレーブ E の無効化
EC_ChangeEnableSetting_DevE(
  Execute :=(Operating1 & DoneHold_DevD & MC_ChangeAxisUse_DevE.Done & NOT(ExclusiveFlg)),
  NodeAdr :=UINT#3,
  IsEnable:=FALSE);

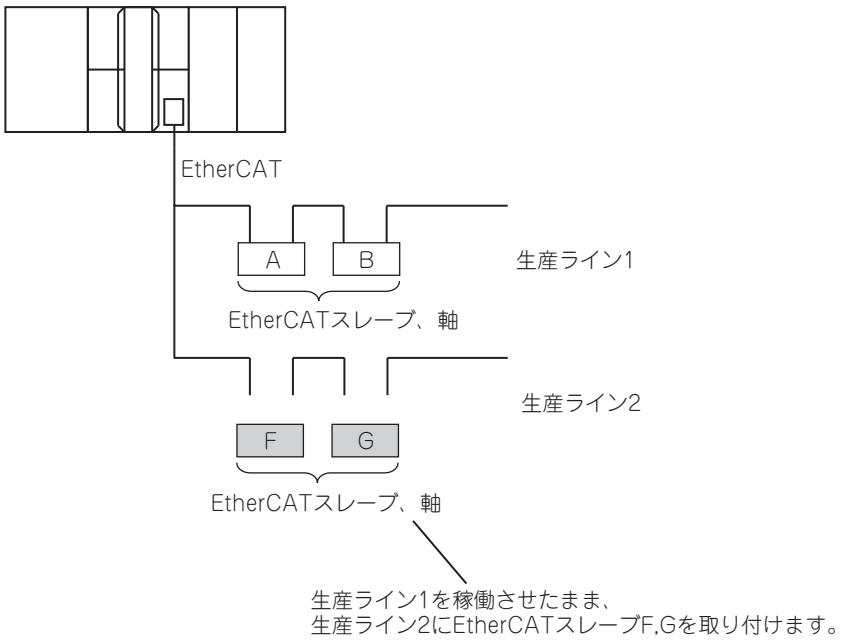
// 命令の排他制御
R_TRIG_DevE(EC_ChangeEnableSetting_DevE.Busy, ExclusiveFlgSet);
F_TRIG_DevE(EC_ChangeEnableSetting_DevE.Busy, ExclusiveFlgReset);
RS_ExFlg_DevE(
  Set :=ExclusiveFlgSet,
  Reset1:=ExclusiveFlgReset,
  Q1 =>ExclusiveFlg);
RS_DevE(
  Set :=EC_ChangeEnableSetting_DevE.Done,
  Reset1:=Operating1End,
  Q1 =>DoneHold_DevE);

// EtherCAT スレーブ E の未使用軸化と無効化確認
Operating1End:=(Operating1 & DoneHold_DevE);

// 取り外し可能ランプ点灯
SR_instance1(
  Set1:=Operating1End,
  Q1 =>Light1On);
```

EtherCAT スレーブを EtherCAT ネットワークに取り付ける例

先の例と同じシステムの生産ライン 1 を稼働させたまま、生産ライン 2 に EtherCAT スレーブ F,G を取り付けます。
EtherCAT スレーブ F,G には、モーション制御の軸も設定するとします。
したがって、EtherCAT スレーブを有効化するとともに、軸使用を使用軸化する処理も行います。



● 操作手順

本サンプルプログラムの操作手順は、以下のとおりです。

1 **操作者が、EtherCAT スレーブ F,G を取り付けます。**

2 **操作者が表示器のボタンを押すと、実行条件が ON になります。**

3 **コントローラが、EtherCAT スレーブ F,G を有効化します。また、それぞれの軸使用を使用軸に設定します。**

4 **2 つの EtherCAT スレーブの有効化、使用軸化がすべて完了したら、コントローラが取付完了ランプを点灯させます。**

● 軸使用を使用軸化する命令

軸使用を使用軸化するために、MC_ChangeAxisUse 命令を使用します。
MC_ChangeAxisUse 命令の仕様の詳細は、『NY シリーズ コマンドリファレンスマニュアル モーション編(SBCE-380)』を、参照してください。

● 命令の排他制御

EC_ChangeEnableSetting 命令は、複数同時に実行することができません。
また、EC_ChangeEnableSetting 命令は、複数のタスク周期にまたがって実行されます。
そこで、先に実行した EC_ChangeEnableSetting 命令の完了を確認してから、つぎの EC_ChangeEnableSetting 命令を実行します。
そのために、「ExclusiveFlg（命令排他フラグ）」という変数を使用します。
「ExclusiveFlg」の値が TRUE ならば、EC_ChangeEnableSetting 命令が実行中であることを意味します。
「ExclusiveFlg」の値が TRUE である間は、つぎの EC_ChangeEnableSetting 命令を実行しません。

EC_ChangeEnableSetting 命令は、別のタスク内の EC_ChangeEnableSetting 命令とも、同時に実行することができません。
そこで、本サンプルプログラムでは「ExclusiveFlg」をグローバル変数にしています。

こうすることで、本サンプルプログラムとは別のタスク内の EC_ChangeEnableSetting 命令との排他制御が行えます。
ただしその場合、別のタスク内でも、同じグローバル変数「ExclusiveFlg」を使って命令の排他制御を行う必要があります。
また、EC_ChangeEnableSetting 命令は、EC_DisconnectSlave 命令や EC_ConnectSlave 命令とも同時に実行することができません。
EC_DisconnectSlave 命令の「サンプルプログラム (P.2-917)」は、本サンプルプログラムと同じグローバル変数「ExclusiveFlg」を使用して、命令の排他制御を行う例になっています。

● 各 EtherCAT スレーブの軸変数とノードアドレス

EtherCAT スレーブ F,G の軸に割り付けられた軸変数と、ノードアドレスは、以下のとおりとします。

EtherCAT スレーブ	軸変数	ノードアドレス
F	MC_Axis003	4
G	MC_Axis004	5

● グローバル変数

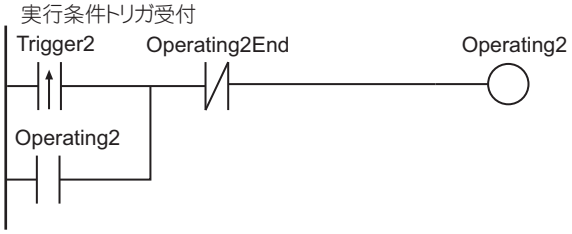
名称	データ型	初期値	割付先	コンスタント	コメント
MC_Axis003	_sAXIS_REF		MC://_MC_AX[3]	☑	EtherCAT スレーブ F の軸変数
MC_Axis004	_sAXIS_REF		MC://_MC_AX[4]	☑	EtherCAT スレーブ G の軸変数
ExclusiveFlg	BOOL	FALSE		☐	命令排他フラグ

● LD

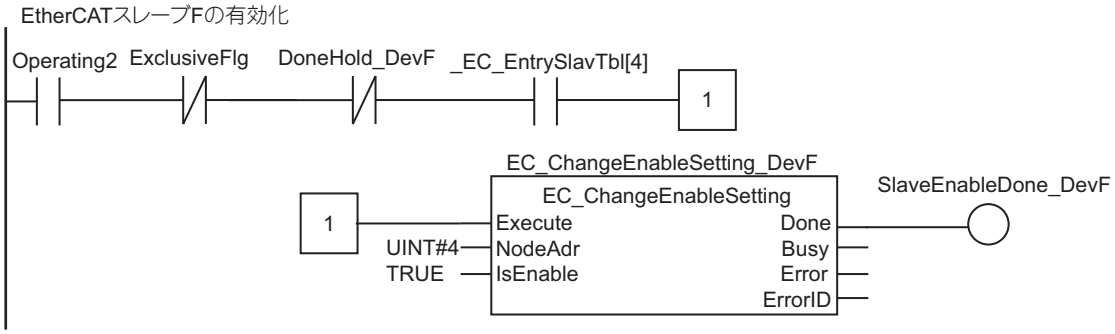
内部変数	名称	データ型	初期値	コメント
	Operating2End	BOOL	FALSE	処理終了
	Trigger2	BOOL	FALSE	実行条件
	Operating2	BOOL	FALSE	処理中
	AxisUseDone_DevF	BOOL	FALSE	EtherCAT スレーブ F の軸使用の使用軸化完了
	SlaveEnableDone_DevF	BOOL	FALSE	EtherCAT スレーブ F の有効化完了
	DoneHold_DevF	BOOL	FALSE	EtherCAT スレーブ F の処理完了の保持
	AxisUseDone_DevG	BOOL	FALSE	EtherCAT スレーブ G の軸使用の使用軸化完了
	SlaveEnableDone_DevG	BOOL	FALSE	EtherCAT スレーブ G の有効化完了
	DoneHold_DevG	BOOL	FALSE	EtherCAT スレーブ G の処理完了の保持
	Light2On	BOOL	FALSE	取付完了ランプ点灯
	MC_ChangeAxisUse_DevF	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevF	EC_ChangeEnableSetting		
	MC_ChangeAxisUse_DevG	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevG	EC_ChangeEnableSetting		

外部変数	名称	データ型	コンスタント	コメント
	MC_Axis003	_sAXIS_REF	☑	EtherCAT スレーブ F の軸変数
	MC_Axis004	_sAXIS_REF	☑	EtherCAT スレーブ G の軸変数
	ExclusiveFlg	BOOL	☐	命令排他フラグ

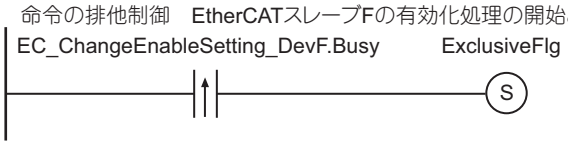
実行条件トリガ受付



EtherCATスレーブFの有効化

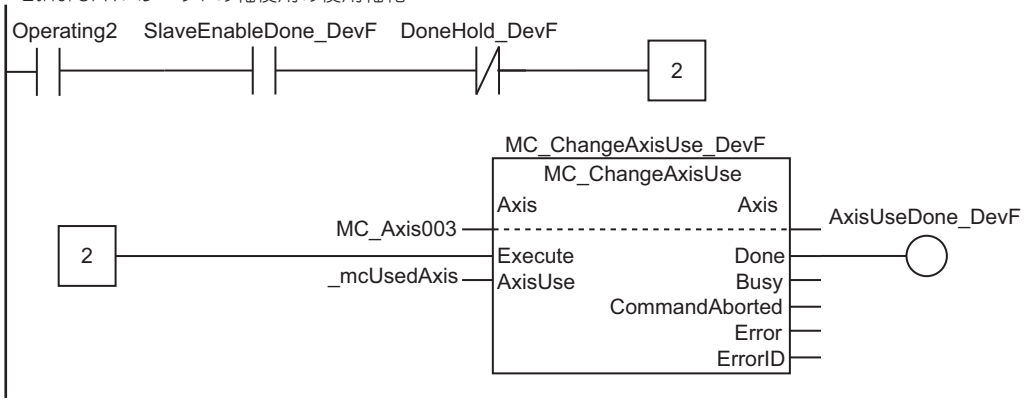


命令の排他制御　EtherCATスレーブFの有効化処理の開始と完了の確認

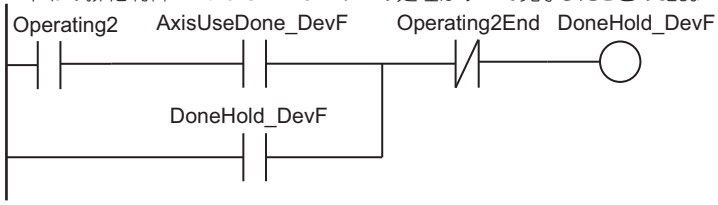




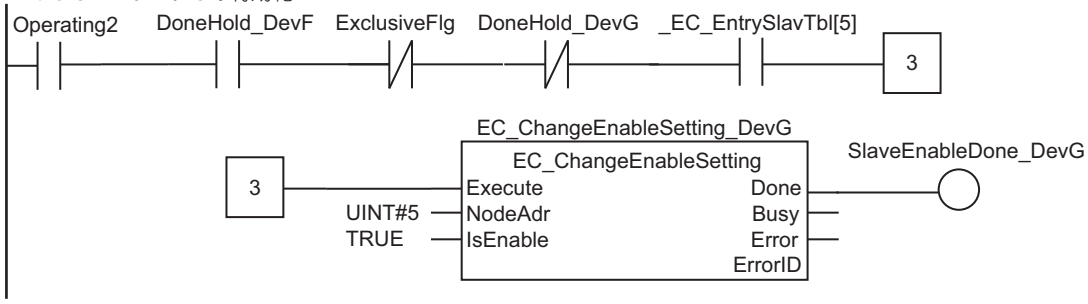
EtherCATスレーブFの軸使用の使用軸化



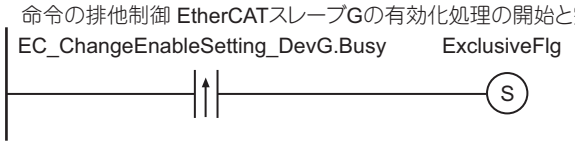
命令の排他制御　EtherCATスレーブFの処理がすべて完了したことの確認

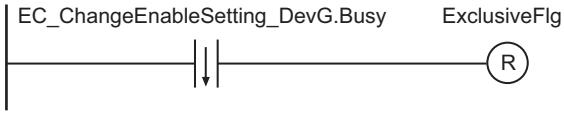


EtherCATスレーブGの有効化

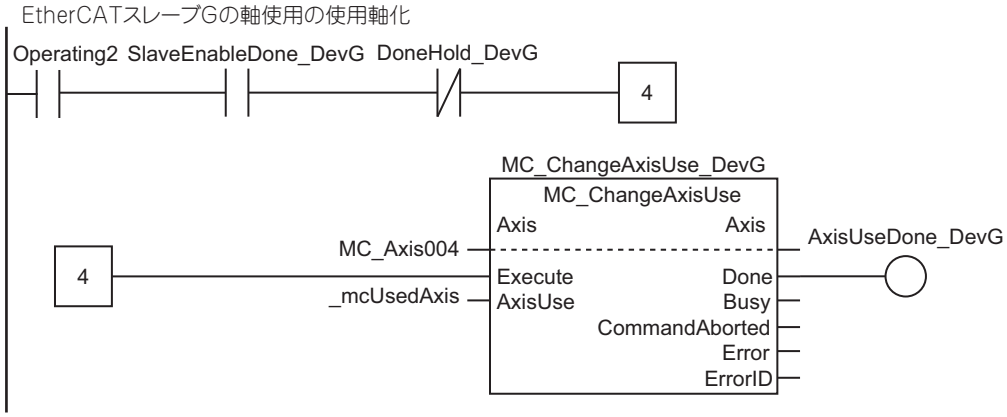


命令の排他制御 EtherCATスレーブGの有効化処理の開始と完了の確認

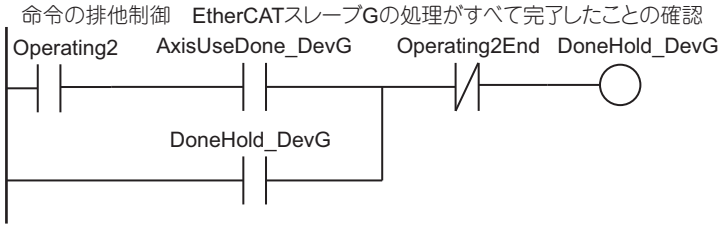


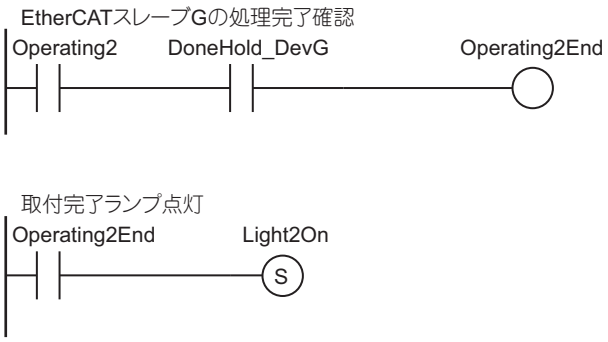


EtherCATスレーブGの軸使用の使用軸化



命令の排他制御　EtherCATスレーブGの処理がすべて完了したことの確認





● ST

内部変数	名称	データ型	初期値	コメント
	Operating2End	BOOL	FALSE	処理終了
	Trigger2	BOOL	FALSE	実行条件
	Operating2	BOOL	FALSE	処理中
	Operating2Set	BOOL	FALSE	処理開始
	Light2On	BOOL	FALSE	取付完了ランプ点灯
	DoneHold_DevF	BOOL	FALSE	EtherCAT スレーブ F の処理完了の保持
	DoneHold_DevG	BOOL	FALSE	EtherCAT スレーブ G の処理完了の保持
	ExclusiveFlgSet	BOOL	FALSE	命令排他フラグ ON
	ExclusiveFlgReset	BOOL	FALSE	命令排他フラグ 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		

外部変数	名称	データ型	コンスタント	コメント
	MC_Axis003	_sAXIS_REF	☑	EtherCAT スレーブ F の軸変数
	MC_Axis004	_sAXIS_REF	☑	EtherCAT スレーブ G の軸変数
	ExclusiveFlg	BOOL	☐	命令排他フラグ

```
// 実行条件トリガ受付
R_TRIG_instance2(Trigger2, Operating2Set);
RS_instance2(
  Set :=Operating2Set,
  Reset1:=Operating2End,
  Q1 =>Operating2);

// EtherCAT スレーブ F の有効化
EC_ChangeEnableSetting_DevF(
  Execute :=(Operating2 & NOT(ExclusiveFlg) & NOT(DoneHold_DevF) & _EC_EntrySlavTbl[4]),
  NodeAdr :=UINT#4,
  IsEnable:=TRUE);

// 命令の排他制御 EtherCAT スレーブ F の有効化処理の開始と完了の確認
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);

// EtherCAT スレーブ F の軸使用の使用軸化
MC_ChangeAxisUse_DevF(
  Axis :=MC_Axis003,
  Execute:=(Operating2 & EC_ChangeEnableSetting_DevF.Done & NOT(DoneHold_DevF)),
  AxisUse:=_mcUsedAxis);

// 命令の排他制御 EtherCAT スレーブ F の処理がすべて完了したことの確認
RS_DevF(
  Set :=(Operating2 & MC_ChangeAxisUse_DevF.Done),
  Reset1:=Operating2End,
  Q1 =>DoneHold_DevF);

// EtherCAT スレーブ G の有効化
EC_ChangeEnableSetting_DevG(
  Execute :=(Operating2 & DoneHold_DevF & NOT(ExclusiveFlg) & NOT(DoneHold_DevG) &
    _EC_EntrySlavTbl[5]),
  NodeAdr :=UINT#5,
  IsEnable:=TRUE);

// 命令の排他制御 EtherCAT スレーブ G の有効化処理の開始と完了の確認
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);

// EtherCAT スレーブ G の軸使用の使用軸化
MC_ChangeAxisUse_DevG(
```

```
  Axis :=MC_Axis004,
  Execute:=(Operating2 & EC_ChangeEnableSetting_DevG.Done & NOT(DoneHold_DevG)),
  AxisUse:=_mcUsedAxis);

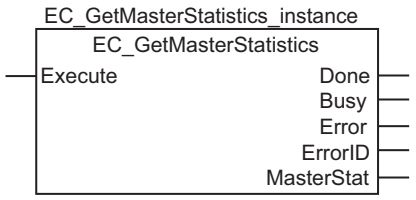
// 命令の排他制御 EtherCAT スレーブ G の処理がすべて完了したことの確認
RS_DevG(
  Set :=(Operating2 & MC_ChangeAxisUse_DevG.Done),
  Reset1:=Operating2End,
  Q1 =>DoneHold_DevG);

// EtherCAT スレーブ G の処理完了確認
Operating2End:=Operating2 & DoneHold_DevG;

// 取付完了ランプ点灯
SR_instance2(
  Set1:=Operating2End,
  Q1 =>Light20n);
```

EC_GetMasterStatistics

EtherCAT マスタの診断・統計情報を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_GetMasterStatistics	EtherCAT マスタ診断・統計情報読出	FB		EC_GetMasterStatistics_instance(Execute, Done, Busy, Error, ErrorID, MasterStat);

バージョン関連情報

変数

	名称	入／出力	内容	有効範囲	単位	初期値
MasterStat	マスタ診断・統計情報	出力	読み出した EtherCAT マスタの診断・統計情報を格納	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MasterStat	構造体_sECAT_MASTER_STAT の詳細は「機能 (P.2-943)」参照																			

機能

EtherCAT マスタの診断・統計情報を読み出します。
読み出した EtherCAT マスタの診断・統計情報は、マスタ診断・統計情報「MasterStat」に格納されます。

「MasterStat」のデータ型は、構造体_sECAT_MASTER_STAT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
MasterStat	マスタ診断・統計情報	読み出した EtherCAT マスタの診断･統計情報を格納	_sECAT_MASTER_STAT	−	−	−
TotalSentCnt	総送信フレーム数	マスタが送信した EtherCAT フレームの総数	UDINT	データ型に従う	−	−
TotalRecvCnt	総受信フレーム数	マスタが受信した EtherCAT フレームの総数	UDINT			
TimeoutCnt	フレーム受信タイムアウト回数	フレームの受信タイムアウトが発生した回数	UDINT			
DiscardPDOCnt	プロセスデータ受信破棄数	プロセスデータ受信処理にて破棄したプロセスデータ数	UDINT			
NetDelay	ネットワーク伝播遅延時間	CPU ユニットからフレームを送信してから､送信したフレームを受信するまでの時間	UDINT			
CRCErrCnt	CRC エラーフレーム受信数	CRC エラーとなったフレームを受信した回数	UDINT			

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_StatisticsLogBusy	診断統計情報ログ実行状態フラグ	BOOL	診断統計情報ログ機能が実行中かどうかを示します。 TRUE: 実行中 FALSE: 実行中ではない

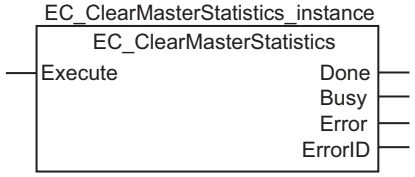
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- CPU ユニットの診断・統計情報ログ機能が実行中は、本命令を実行することはできません。
- EtherCAT マスタ診断・統計情報読出命令の別インスタンスが実行中、または EtherCAT マスタ診断・統計情報クリア命令が実行中は、本命令を実行することはできません。
- 診断・統計情報の読み出し中に、診断・統計情報のクリアを実行すると、0 クリアされた診断・統計情報が読み出されることがあります。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。また、EtherCAT マスタ診断・統計情報「MasterStat」には何も格納しません。

a）CPU ユニットの診断・統計情報ログ機能が実行中の場合。
b）EtherCAT マスタ診断・統計情報読出命令の別インスタンスが実行中、または EtherCAT マスタ診断・統計情報クリア命令が実行中の場合。
c）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
d）EtherCAT フレーム受信なし異常が発生中で、異常発生箇所特定中だった場合

EC_ClearMasterStatistics

EtherCAT マスタの診断・統計情報をクリアします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_ClearMasterStatistics	EtherCAT マスタ診断・統計情報クリア	FB		EC_ClearMasterStatistics_instance(Execute, Done, Busy, Error, ErrorID);

バージョン関連情報

変数

共通の変数だけ

機能

EtherCAT マスタの診断・統計情報をクリアします。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_StatisticsLogBusy	診断統計情報ログ実行状態フラグ	BOOL	診断統計情報ログ機能が実行中かどうかを示します。 TRUE: 実行中 FALSE: 実行中ではない

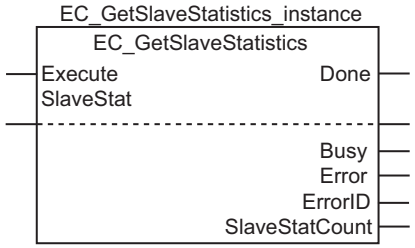
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- CPU ユニットの診断・統計情報ログ機能が実行中は、本命令を実行することはできません。
- EtherCAT マスタ診断・統計情報クリア命令の別インスタンスが実行中、または EtherCAT マスタ診断・統計情報読出命令が実行中は、本命令を実行することはできません。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。

a）CPU ユニットの診断・統計情報ログ機能が実行中の場合。
b）EtherCAT マスタ診断・統計情報クリア命令の別インスタンスが実行中、または EtherCAT マスタ診断・統計情報読出命令が実行中の場合。
c）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
d）EtherCAT フレーム受信なし異常が発生中で、異常発生箇所特定中だった場合

EC_GetSlaveStatistics

EtherCAT スレーブの診断・統計情報を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_GetSlaveStatistics	EtherCAT スレーブ診断・統計情報読出	FB		EC_GetSlaveStatistics_instance(Execute, SlaveStat, Done, Busy, Error, ErrorID, SlaveStatCount);

バージョン関連情報

変数

	名称	入／出力	内容	有効範囲	単位	初期値
SlaveStat[ ]配列	スレーブ診断・統計情報	入出力	読み出した EtherCAT スレーブの診断・統計情報を格納	データ型に従う	−	−
SlaveStatCount	読み出したスレーブ診断・統計情報の要素数	出力	EtherCAT マスタのネットワーク構成情報に登録されている EtherCAT スレーブの数	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SlaveStat[] 配列	構造体_sECAT_SLAVE_STAT の詳細は「機能 (P.2-948)」参照																			
SlaveStatCount							○													

機能

EtherCAT スレーブの診断・統計情報を読み出します。
EtherCAT マスタのネットワーク構成情報に登録されている EtherCAT スレーブの数を、読み出したスレーブ診断・統計情報の要素数「SlaveStatCount」に格納します。
EtherCAT スレーブから読み出した診断・統計情報を、スレーブ診断・統計情報「SlaveStat[]」に格納します。
なお、スレーブ診断・統計情報を格納する「SlaveStat[]」の配列の要素数を、ネットワーク構成情報のスレーブ数以上にしてください。ネットワーク構成情報のスレーブ数より少ない場合、命令実行時にエラーとなります。

「SlaveStat[]」のデータ型は、構造体_sECAT_SLAVE_STAT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SlaveStat	スレーブ診断・統計情報	スレーブ診断･統計情報	_sECAT_SLAVE_STAT	–	–	–
Result	取得結果	スレーブに対して診断･統計情報を取得した結果	_eEC_STAT_RESULT	データ型に従う	–	–
NodeAdr	ノードアドレス	スレーブのノードアドレス	UINT			
PortNum	ポート数	スレーブが持つポート数	USINT			
ErrCnt	ポート毎のエラーカウンタ	ポートごとのエラーカウンタ	ARRAY[0..3] OF USINT			

「SlaveStat.Result」のデータ型は、列挙型_eEC_STAT_RESULT です。列挙子の意味は、以下のとおりです。

列挙子	意味
_EC_STAT_SUCCESS	取得成功
_EC_STAT_FAIL	取得失敗
_EC_STAT_UNCONNECTED	未接続

スレーブ診断・統計情報は、「SlaveStat[]」の配列に EtherCAT マスタのネットワーク構成情報の接続順で格納します。
また、EtherCAT スレーブの診断・統計情報のポート毎のエラーカウンタは、以下のように、フレームを受信するポートの順番で格納します。

- 2 ポートスレーブの場合(IN/OUT のみ)
  ErrCnt[0]=IN、ErrCnt[1]=OUT
- 3 ポートスレーブの場合(IN/X2/X3 をもつ GX-JC03 など)
  ErrCnt[0]=IN、ErrCnt[1]=X2、ErrCnt[2]=X3
- 6 ポートスレーブの場合(IN/X2/X3/InternalPort をもつ GX-JC06 の MainDevice、InternalPort/X4/X5/X6 をもつ GX-JC06 の SubDevice など)
  MainDevice と SubDevice は別々のスレーブとして格納します。
  a）MainDevice
  ErrCnt[0]=IN、ErrCnt[1]=X2、ErrCnt[2]=X3、ErrCnt[3]=InternalPort
  b）SubDevice
  ErrCnt[0]=InternalPort、ErrCnt[1]=X4、ErrCnt[2]=X5、ErrCnt[3]=X6

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_StatisticsLogBusy	診断統計情報ログ実行状態フラグ	BOOL	診断統計情報ログ機能が実行中かどうかを示します。 TRUE: 実行中 FALSE: 実行中ではない

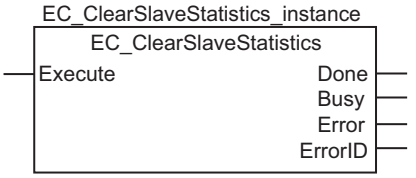
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。

- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 診断・統計情報を読み出す EtherCAT スレーブは、CPU ユニットにダウンロードした EtherCAT マスタのネットワーク構成情報に登録されている、かつ実ネットワーク構成と照合一致しているものだけです。
- すべての EtherCAT スレーブ診断・統計情報を複数のタスク周期にまたがって読み出すため、各 EtherCAT スレーブのデータの同時性はありません。
- CPU ユニットの診断・統計情報ログ機能が実行中は、本命令を実行することはできません。
- EtherCAT スレーブ診断・統計情報読出命令の別インスタンスが実行中、または EtherCAT スレーブ診断・統計情報クリア命令が実行中は、本命令を実行することはできません。
- 診断・統計情報の読み出し中に、診断・統計情報のクリアを実行すると、0 クリアされた診断・統計情報が読み出されることがあります。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a）CPU ユニットの診断・統計情報ログ機能が実行中の場合。
  b）EtherCAT スレーブ診断・統計情報読出命令の別インスタンスが実行中、または EtherCAT スレーブ診断・統計情報クリア命令が実行中の場合。
  c）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
  d）EtherCAT フレーム受信なし異常が発生中で、異常発生箇所特定中だった場合
  e）「SlaveStat[]」の配列の要素数が、ネットワーク構成情報のスレーブ数より少ない場合

EC_ClearSlaveStatistics

EtherCAT スレーブの診断・統計情報をクリアします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
EC_ClearSlaveStatistics	EtherCAT スレーブ診断・統計情報クリア	FB		EC_ClearSlaveStatistics_instance(Execute, Done, Busy, Error, ErrorID);

バージョン関連情報

変数

共通の変数だけ

機能

EtherCAT スレーブの診断・統計情報をクリアします。

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_StatisticsLogBusy	診断統計情報ログ実行状態フラグ	BOOL	診断統計情報ログ機能が実行中かどうかを示します。 TRUE: 実行中 FALSE: 実行中ではない

使用上の注意

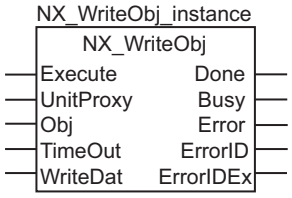
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズの EtherCAT ポートにだけ、使用可能です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 診断・統計情報をクリアする EtherCAT スレーブは、CPU ユニットにダウンロードした EtherCAT マスタのネットワーク構成情報に登録されている、かつ実ネットワーク構成と照合一致しているものだけです。
- CPU ユニットの診断・統計情報ログ機能が実行中は、本命令を実行することはできません。
- EtherCAT スレーブ診断・統計情報クリア命令の別インスタンスが実行中、または EtherCAT スレーブ診断・統計情報読出命令が実行中は、本命令を実行することはできません。
- EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命

令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令で同時実行可能な個数は、最大 32 です。

- つぎの場合、異常となります。「Error」は TRUE となり、「ErrorID」に異常 ID が代入されます。
  a）CPU ユニットの診断・統計情報ログ機能が実行中の場合。
  b）EtherCAT スレーブ診断・統計情報クリア命令の別インスタンスが実行中、または EtherCAT スレーブ診断・統計情報読出命令が実行中の場合。
  c）EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、EC_GetMasterStatistics 命令、EC_ClearMasterStatistics 命令、EC_GetSlaveStatistics 命令、EC_ClearSlaveStatistics 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
  d）EtherCAT フレーム受信なし異常が発生中で、異常発生箇所特定中だった場合

NX_WriteObj

EtherCAT カプラユニット、または NX ユニットの NX オブジェクトに、データを書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_WriteObj	NX オブジェクト書込	FB		NX_WriteObj_instance(Execute, UnitProxy, Obj, TimeOut, WriteDat, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	指定ユニット	入力	データを書き込むユニット	–	–	*1
Obj	オブジェクトパラメータ		オブジェクトパラメータ			–
TimeOut	タイムアウト時間		タイムアウト時間 0 の場合は、2.0s	0 ～ 60000	ms	2000 (2.0s)
WriteDat	書込データ		NX オブジェクトに書き込むデータ	データ型に従う	–	*1

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID　詳細は「機能 (P.2-953)」参照																			
Obj	構造体_sNXOBJ_ACCESS　詳細は「機能 (P.2-953)」参照																			
TimeOut							○													
WriteDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	配列全体も指定可能																			

機能

EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットの NX オブジェクトに、「WriteDat」の内容を書き込みます。
データを書き込むユニットを、「UnitProxy」で指定します。

タイムアウト時間は、「TimeOut」で指定します。
タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。その場合、データの書込が行われないことがあります。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

変数		名称	内容	データ型
UnitProxy		指定ユニット	指定するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定するユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定するユニットに割り付けられたデバイス変数を渡してください。

「Obj」のデータ型は、構造体_sNXOBJ_ACCESS です。仕様は、以下のとおりです。

変数		名称	内容	データ型	有効範囲	単位	初期値
Obj		オブジェクトパラメータ	オブジェクトパラメータ	_sNXOBJ_ACCESS	−	−	−
	Index	インデックス	インデックス	UINT	データ型に従う	−	0
	Subindex	サブインデックス	サブインデックス	USINT			
	IsCompleteAccess *1	Complete アクセス	Complete アクセス	BOOL	FALSE だけ		FALSE

*1. このメンバを、本命令で使用することはありません。値は、必ず FALSE にしてください。

関連命令との実行手順

本命令は、EtherCAT カプラユニット、または EtherCAT カプラユニット上の NX ユニットに書き込むデータの属性によって、他の関連命令と連携して実行する必要があります。

● 実行手順 1

以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性あり
- 値が反映されるのはユニットのリスタート時

1 **「NX_ChangeWriteMode (P.2-845)」命令を使って、対象ユニットをデータの書込ができるモードに変更します。**

2 **本命令を使って、対象ユニットにデータを書き込みます。**

3 **「NX_SaveParam (P.2-850)」命令を使って、書き込んだデータを保存します。**

4 **「RestartNXUnit (P.2-840)」命令を使って、対象ユニットをリスタートさせます。**

● 実行手順 2

以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性あり
- 値が反映されるのは書込直後

1 **本命令を使って、対象ユニットにデータを書き込みます。**

2 **「NX_SaveParam (P.2-850)」命令を使って、書き込んだデータを保存します。**

● **実行手順 3**

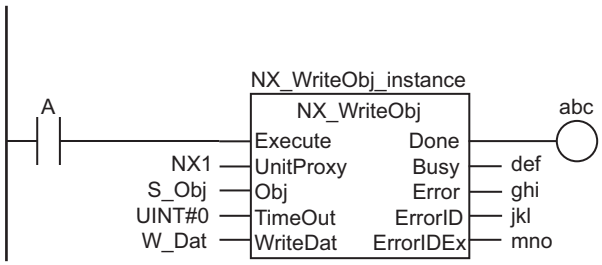
以下の属性をもつデータを書き込む場合の手順を示します。
- 電源断保持属性なし

1 **本命令を使って、対象ユニットにデータを書き込みます。**

記述例

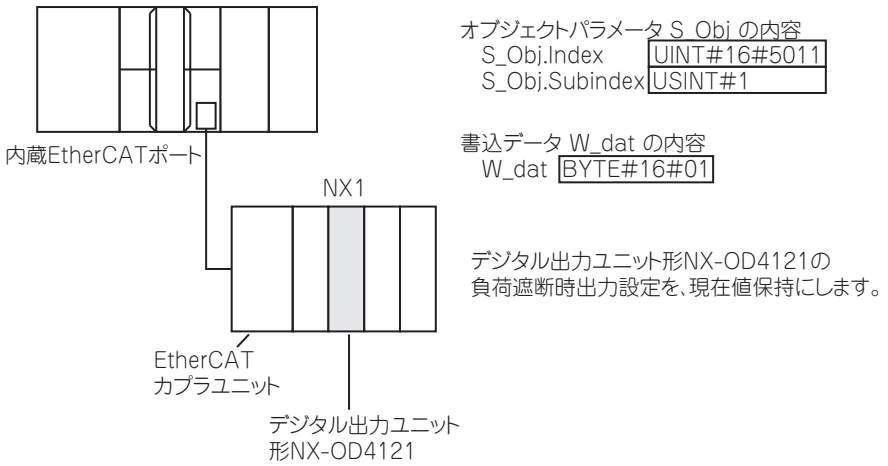
デジタル出力ユニット形 NX-OD4121 の負荷遮断時出力設定を、現在値保持にする場合の例を、以下に示します。
データを書き込むユニットには、_sNXUNIT_ID 型の「NX1」という変数が割り付けられているとします。
また、形 NX-OD4121 の負荷遮断時出力設定のインデックスは UINT#16#5011、サブインデックスは USINT#1 です。
負荷遮断時出力設定を現在値保持にするために書き込む値は、BYTE#16#01 です。

LD



ST

```
NX_WriteObj_instance(A, NX1, S_Obj, UINT#0,
                     W_Dat, abc, def, ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可
_NXB_UnitMsgActiveTbl[i]	NX ユニットメッセージ通信可能ステータス	BOOL	メッセージ通信が可能なスレーブの一覧です。該当スレーブと、通信が可能かどうかを確認する場合に使用します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「WriteDat」が配列全体の場合、配列全体のサイズを、指定したユニットの書込先の NX オブジェクトと同じサイズにしてください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニット、または EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 「WriteDat」に渡すパラメータは、必ず変数にしてください。定数を渡すと、ビルド時に異常になります。
- 電源断時の保持属性ありのデータを書き込み、保持させたい場合は、本命令の実行後に「NX_SaveParam (P.2-850)」命令を実行してください。NX_SaveParam 命令を実行せずに対象ユニットがリスタートされると、NX オブジェクトのデータは書込前の値に戻ります。
- 本命令は、NX メッセージ通信異常に関連する命令です。NX メッセージ通信異常に関連する命令を一度に多数実行すると、NX メッセージ通信異常が発生する可能性があります。NX メッセージ通信異常に関連する命令については、「A-3 NX メッセージ通信異常に関連する命令 (P.A-22)」を参照してください。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0400	16#00000000	• 「UnitProxy」の値が、有効範囲外です。 • 「TimeOut」の値が、有効範囲外です。
16#0419	16#00000000	• 「UnitProxy」のデータ型が、不正です。 • 「WriteDat」のデータ型が、不正です。
16#041B	16#00000000	「WriteDat」に 2048 バイトを超えるデータが指定されています。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。
16#2C01	16#00000000	同時に実行できる命令数を超えています。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。
16#2C03	16#00000000	送信メッセージのサイズが不正です。

サンプルプログラム

ここでは、以下の 2 つのサンプルプログラムについて説明します。

- NX ユニットに、電源断保持属性あり、かつ値が反映されるのがユニットリスタート時であるデータを書き込む例
- NX ユニットに、電源断保持属性あり、かつ値が反映されるのが書込直後であるデータを書き込む例

反映がユニットリスタート時となるデータの書き込み例

EtherCAT カプラユニットに接続されている、AC 入力ユニット形 NX-AD2203 のオブジェクトパラメータ［Ch1 入力移動平均時間］を、500μs に設定します。
EtherCAT カプラユニットのノードアドレスは 10 とします。

オブジェクトパラメータ［Ch1 入力移動平均時間］の仕様は、以下のとおりです。

項目	値
インデックス	16#5004
サブインデックス	16#01
500μs に対応する設定データ	2

オブジェクトパラメータ［Ch1 入力移動平均時間］は、電源断保持属性あり、かつ値が反映されるのがユニットリスタート時であるデータです。したがって、処理の手順は、以下のようになります。

1 **NX_ChangeWriteMode 命令を使って、ユニットを、データの書込ができるモードに変更します。**

2 **NX_WriteObj 命令を使って、ユニットにデータを書き込みます。**

3 **NX_SaveParam 命令を使って、書き込んだデータを保存します。**

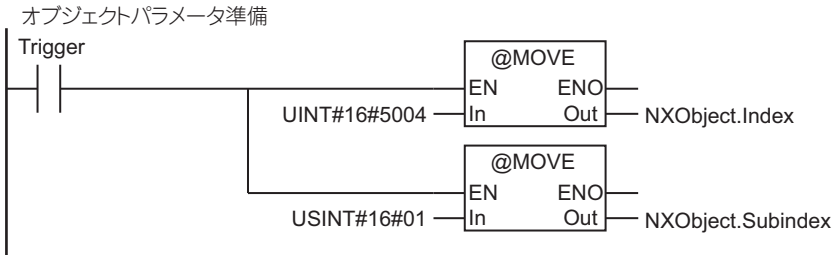
4 **RestartNXUnit 命令を使って、ユニットをリスタートさせます。**

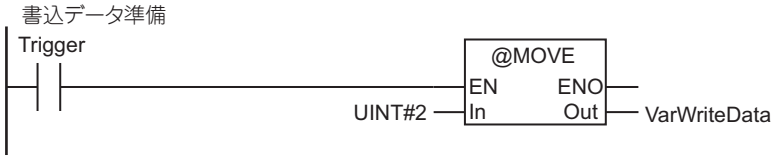
- **LD**

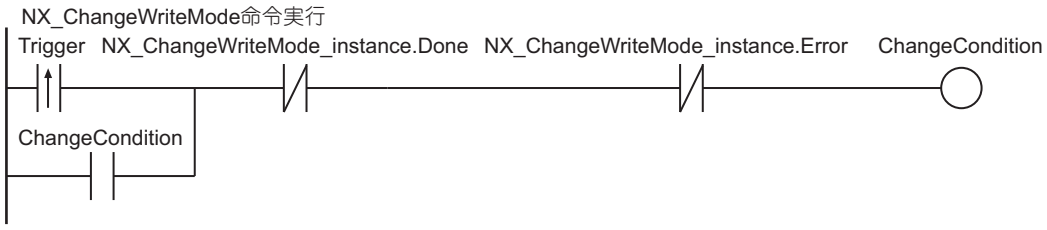
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ChangeCondition	BOOL	FALSE	書込モード変更実行条件
	WriteCondition	BOOL	FALSE	データ書込実行条件
	SaveCondition	BOOL	FALSE	データ保存実行条件
	RestartCondition	BOOL	FALSE	ユニットリスタート実行条件
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（DC 入力ユニット）
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）

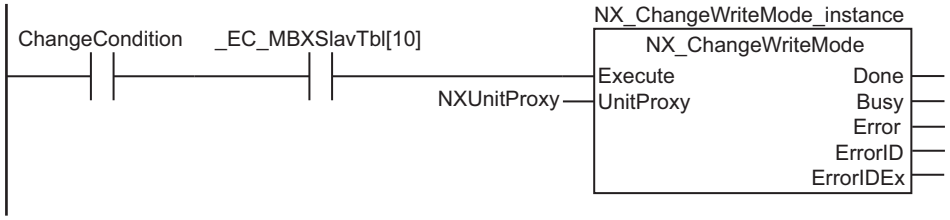
内部変数	名称	データ型	初期値	コメント
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	オブジェクトパラメータ
	VarWriteData	UINT	0	書込データ
	NX_ChangeWriteMode_instance	NX_ChangeWriteMode		
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		
	RestartNXUnit_instance	RestartNXUnit		

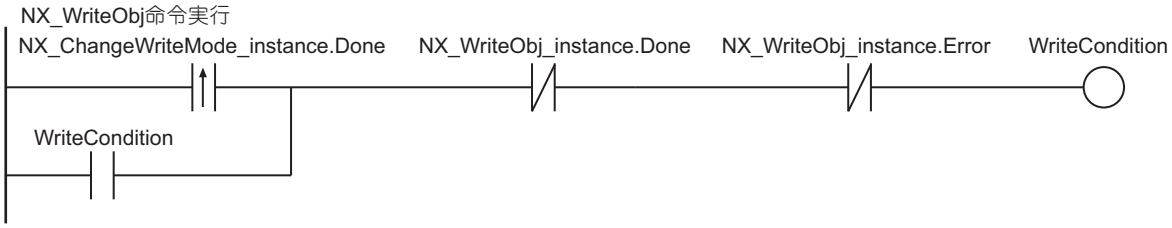
外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル

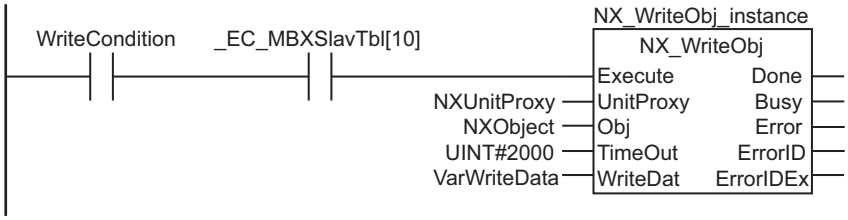


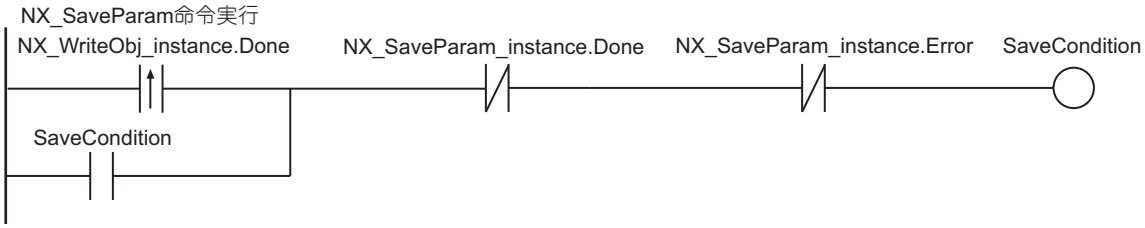


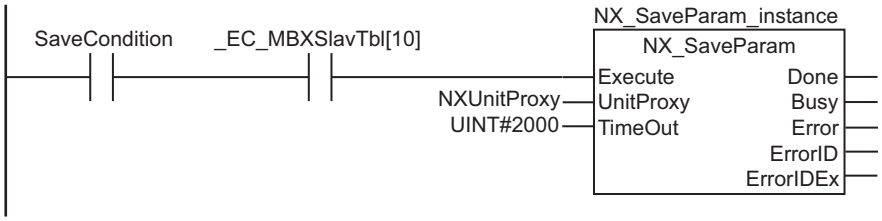


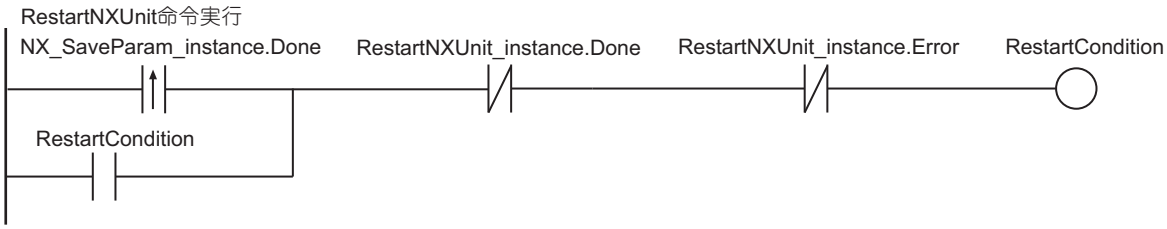


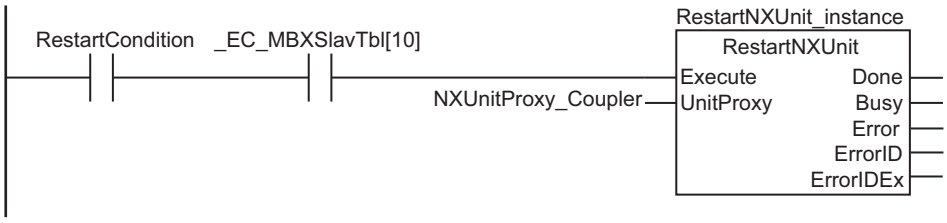

NX_WriteObj命令実行
NX_ChangeWriteMode_instance.Done
NX_WriteObj_instance.Done
NX_WriteObj_instance.Error
WriteCondition
WriteCondition

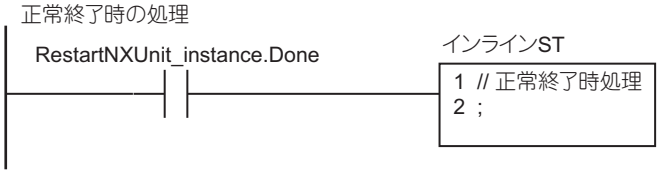

WriteCondition
_EC_MBXSlavTbl[10]
NX_WriteObj_instance
NX_WriteObj
Execute
Done
NXUnitProxy
UnitProxy
Busy
NXObject
Obj
Error
UINT#2000
TimeOut
ErrorID
VarWriteData
WriteDat
ErrorIDEx

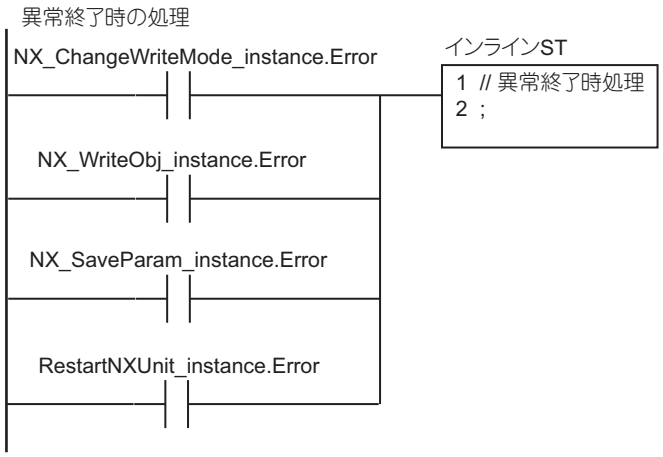

NX_SaveParam命令実行
NX_WriteObj_instance.Done
NX_SaveParam_instance.Done
NX_SaveParam_instance.Error
SaveCondition
SaveCondition


SaveCondition
_EC_MBXSlavTbl[10]
NX_SaveParam_instance
NX_SaveParam
Execute
Done
NXUnitProxy
UnitProxy
Busy
UINT#2000
TimeOut
Error
ErrorID
ErrorIDEx


RestartNXUnit命令実行
NX_SaveParam_instance.Done
RestartNXUnit_instance.Done
RestartNXUnit_instance.Error
RestartCondition
RestartCondition


RestartCondition
_EC_MBXSlavTbl[10]
RestartNXUnit_instance
RestartNXUnit
Execute
Done
NXUnitProxy_Coupler
UnitProxy
Busy
Error
ErrorID
ErrorIDEx


正常終了時の処理
RestartNXUnit_instance.Done
インラインST
1 // 正常終了時処理
2 ;



● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ChangeCondition	BOOL	FALSE	書込モード変更実行条件
	ChangeGo	BOOL	FALSE	書込モード変更実行
	WriteCondition	BOOL	FALSE	データ書込実行条件
	WriteGo	BOOL	FALSE	データ書込実行
	SaveCondition	BOOL	FALSE	データ保存実行条件
	SaveGo	BOOL	FALSE	データ保存実行
	RestartCondition	BOOL	FALSE	ユニットリスタート実行条件
	RestartGo	BOOL	FALSE	ユニットリスタート実行
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（DC 入力ユニット）
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	オブジェクトパラメータ
	VarWriteData	UINT	0	書込データ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	NX_ChangeWriteMode_instance	NX_ChangeWriteMode		
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		
	RestartNXUnit_instance	RestartNXUnit		
	R_Trig_instance	R_TRIG		

外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル

```
// オブジェクトパラメータと書込データの準備
```

```
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;

// NX_ChangeWriteMode 命令実行
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);

// NX_WriteObj 命令実行
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);

// NX_SaveParam 実行
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);

// RestartNXUnit 実行
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
  // 正常終了時処理
  NormalEnd := NormalEnd + UINT#1;
ELSIF ((NX_ChangeWriteMode_instance.Error=TRUE) OR (NX_WriteObj_instance.Error=TRUE)
  OR (NX_SaveParam_instance.Error=TRUE) OR (RestartNXUnit_instance.Error=TRUE)) THEN
  // 異常終了時処理
  ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

反映が書き込み直後となるデータの書き込み例

EtherCAT カプラユニットに接続されている、温度入力ユニット形 NX-TS2101 のオブジェクトパラメータ［Ch1 オフセット値（1 点補正)］を、+0.3℃ に設定します。
EtherCAT カプラユニットのノードアドレスは 10 とします。

オブジェクトパラメータ［Ch1 オフセット値（1 点補正)］の仕様は、以下のとおりです。

項目	値
インデックス	16#5010
サブインデックス	16#01
書き込む値	0.3

オブジェクトパラメータ［Ch1 オフセット値（1 点補正)］は、電源断保持属性あり、かつ値が反映されるのが書込直後であるデータです。したがって、処理の手順は、以下のようになります。

1 **NX_WriteObj 命令を使って、対象ユニットにデータを書き込みます。**

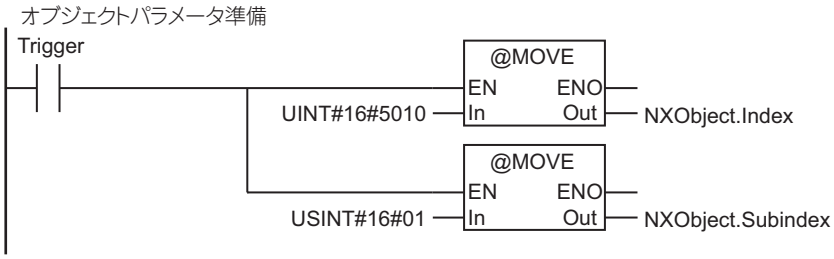
2 **NX_SaveParam 命令を使って、書き込んだデータを保存します。**

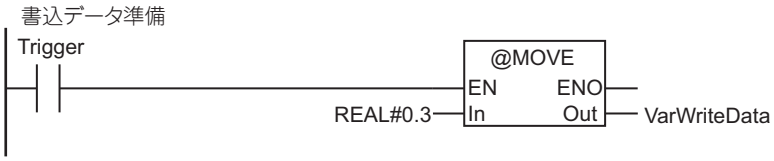
- **LD**

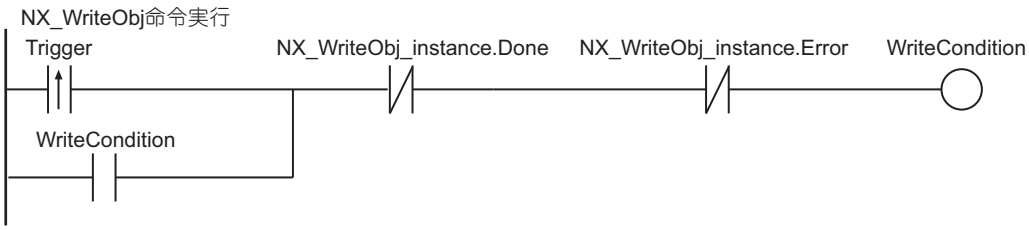
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteCondition	BOOL	FALSE	データ書込実行条件
	SaveCondition	BOOL	FALSE	データ保存実行条件
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（AC 入力ユニット）

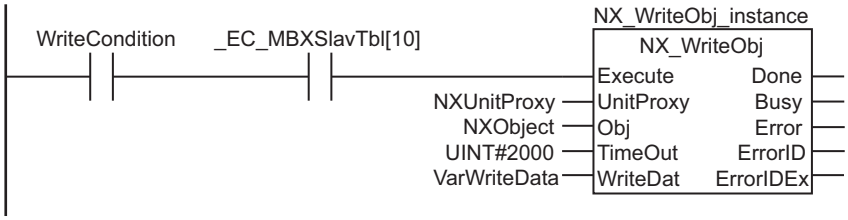
内部変数	名称	データ型	初期値	コメント
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	オブジェクトパラメータ
	VarWriteData	Real	0.0	書込データ
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		

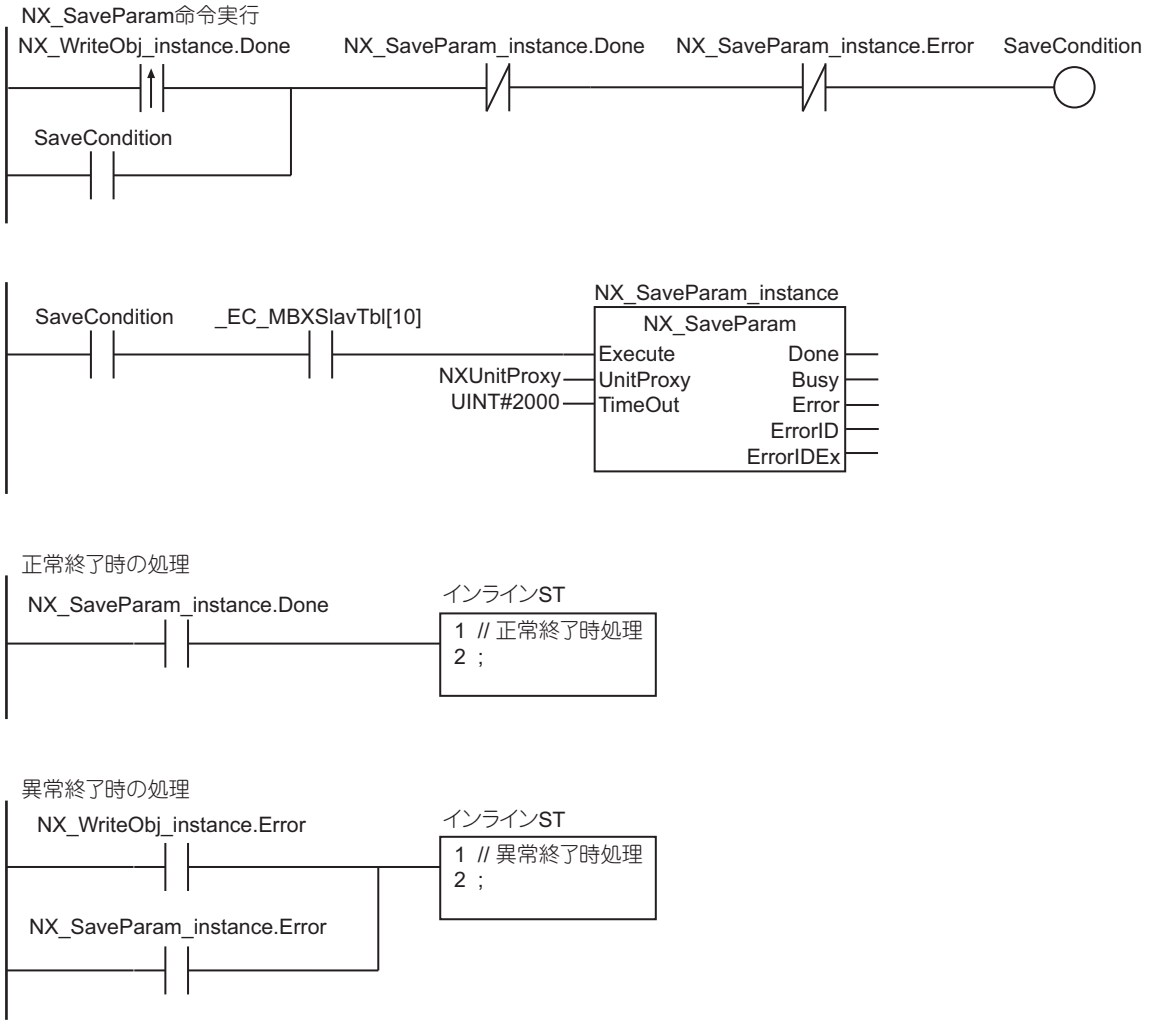
外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル











● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteCondition	BOOL	FALSE	データ書込実行条件
	WriteGo	BOOL	FALSE	データ書込実行
	SaveCondition	BOOL	FALSE	データ保存実行条件
	SaveGo	BOOL	FALSE	データ保存実行
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（温度入力ユニット）
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	オブジェクトパラメータ
	VarWriteData	REAL	0.0	書込データ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		

内部変数	名称	データ型	初期値	コメント
	R_Trig_instance	R_TRIG		

外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル

```
// オブジェクトパラメータと書込データの準備
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;

// NX_WriteObj 命令実行
IF (Trigger=TRUE) THEN
  WriteCondition := TRUE;
END_IF;

IF ((NX_WriteObj_instance.Done=TRUE) OR (NX_WriteObj_instance.Error=TRUE))THEN
  WriteCondition := FALSE;
END_IF;

WriteGo := WriteCondition & _EC_MBXSlavTbl[10];
NX_WriteObj_instance(
  Execute := WriteGo,
  UnitProxy := NXUnitProxy,
  Obj := NXObject,
  TimeOut := UINT#2000,
  WriteDat := VarWriteData);

// NX_SaveParam 実行
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;

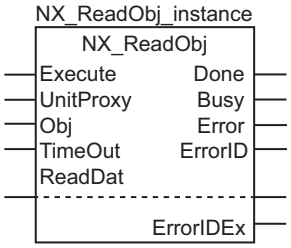
SaveGo := SaveCondition & _EC_MBXSlavTbl[10];
NX_SaveParam_instance(
  Execute := SaveGo,
  UnitProxy := NXUnitProxy,
  TimeOut := UINT#2000);

IF (NX_SaveParam_instance.Done=TRUE) THEN
  // 正常終了時処理
  NormalEnd := NormalEnd + UINT#1;
```

```
ELSIF ((NX_WriteObj_instance.Error=TRUE) OR (NX_SaveParam_instance.Error=TRUE)) THEN
  // 異常終了時処理
  ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

NX_ReadObj

EtherCAT カプラユニット、または NX ユニットの NX オブジェクトから、データを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ReadObj	NX オブジェクト読出	FB		NX_ReadObj_instance(Execute, UnitProxy, Obj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitProxy	指定ユニット	入力	データを読み出すユニット	－	－	*1
Obj	オブジェクトパラメータ		オブジェクトパラメータ			－
TimeOut	タイムアウト時間		タイムアウト時間 0 の場合は、2.0s	0 ～ 60000	ms	2000 (2.0s)
ReadDat	読出データ	入出力	NX オブジェクトから読み出したデータ	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	構造体_sNXUNIT_ID　詳細は「機能 (P.2-968)」参照																			
Obj	構造体_sNXOBJ_ACCESS　詳細は「機能 (P.2-968)」参照																			
TimeOut							○													
ReadDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	配列全体も指定可能																			

機能

EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットの NX オブジェクトから読み出し、「ReadDat」に格納します。
データを読み出すユニットを「UnitProxy」で指定します。

タイムアウト時間は、「TimeOut」で指定します。タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。その場合、データの読出は行われません。

「UnitProxy」のデータ型は、構造体_sNXUNIT_ID です。仕様は、以下のとおりです。

変数		名称	内容	データ型
UnitProxy		指定ユニット	指定するユニット	_sNXUNIT_ID
	NodeAdr	ノードアドレス	通信カプラユニットのノードアドレス	UINT
	IPAdr	IP アドレス	通信カプラユニットの IP アドレス	BYTE[5]
	UnitNo	ユニット番号	指定するユニットのユニット番号	UDINT
	Path	パス	指定するユニットまでの経路情報	BYTE[64]
	PathLength	「Path」の有効データ長	「Path」の有効データ長	USINT

「UnitProxy」には、指定するユニットに割り付けられたデバイス変数を渡してください。

「Obj」のデータ型は、構造体_sNXOBJ_ACCESS です。仕様は、以下のとおりです。

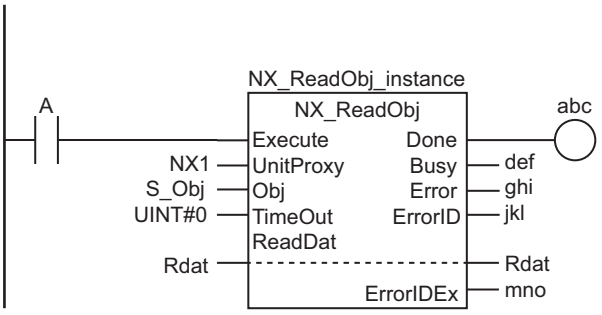
変数		名称	内容	データ型	有効範囲	単位	初期値
Obj		オブジェクトパラメータ	オブジェクトパラメータ	_sNXOBJ_ACCESS	−	−	−
	Index	インデックス	インデックス	UINT	データ型に従う	−	0
	Subindex	サブインデックス	サブインデックス	USINT			
	IsCompleteAccess *1	Complete アクセス	Complete アクセス	BOOL	FALSE だけ		FALSE

*1. このメンバを、本命令で使用することはありません。値は、必ず FALSE にしてください。

記述例

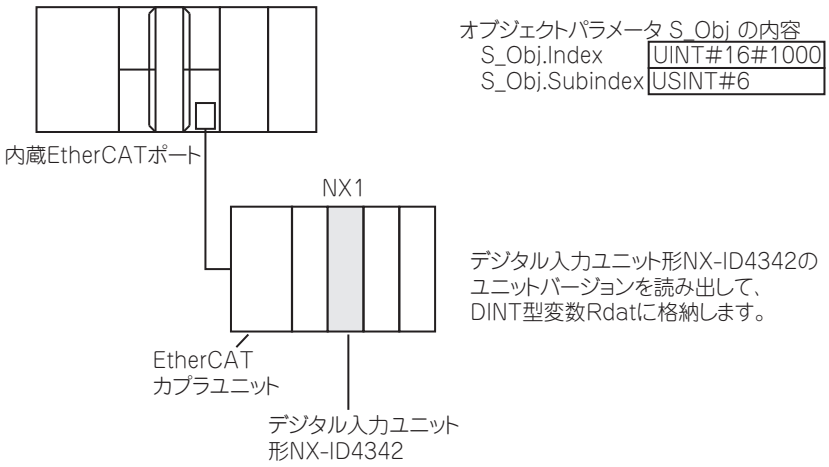
デジタル入力ユニット形 NX-ID4342 の、ユニットバージョンを読み出す場合の例を、以下に示します。
読み出したデータは、UDINT 型変数「Rdat」に格納します。
データを読み出すユニットには、_sNXUNIT_ID 型の「NX1」という変数が割り付けられているとします。
また、形 NX-ID4342 のユニットバージョンのインデックスは UINT#16#1000、サブインデックスは USINT#6 です。

LD



ST

```
NX_ReadObj_instance(A, NX1, S_Obj, UINT#0,
                    Rdat, abc, def, ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl[i] “i”は、ノードアドレス	メッセージ通信可能スレーブテーブル	BOOL	対応するスレーブが、通信可能かどうかを示します。 TRUE: 通信可 FALSE: 通信不可
_NXB_UnitMsgActiveTbl[i]	NX ユニットメッセージ通信可能ステータス	BOOL	メッセージ通信が可能なスレーブの一覧です。該当スレーブと、通信が可能かどうかを確認する場合に使用します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「ReadDat」が配列全体の場合、配列全体のサイズを、指定したユニットの読出元の NX オブジェクトと同じサイズにしてください。
- 「UnitProxy」には、Sysmac Studio の I/O マップで、EtherCAT カプラユニットまたは EtherCAT カプラユニット上の NX ユニットに割り付けたデバイス変数を指定します。
  デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル (SBCA-470)』を参照してください。
- 本命令は、NX メッセージ通信異常に関連する命令です。NX メッセージ通信異常に関連する命令を一度に多数実行すると、NX メッセージ通信異常が発生する可能性があります。NX メッセージ通信異常に関連する命令については、「A-3 NX メッセージ通信異常に関連する命令 (P.A-22)」を参照してください。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」、「ErrorIDEx」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#0400	16#00000000	• 「UnitProxy」の値が、有効範囲外です。 • 「TimeOut」の値が、有効範囲外です。
16#0410	16#00000000	「ReadDat」が STRING 型で、かつ NULL 文字で終端されていません。
16#0419	16#00000000	• 「UnitProxy」のデータ型が、不正です。 • 「ReadDat」のデータ型が、不正です。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
16#041C	16#00000000	「ReadDat」のサイズと、読出元 NX オブジェクトのサイズが不一致です。
16#2C00	16#00000401	指定したユニットが、本命令の対象外です。
	16#00001001 16#00001002 16#00170000 16#00200000 16#00210000	入力パラメータ、出力パラメータ、入出力パラメータが不正です。 入力パラメータ、出力パラメータ、入出力パラメータが、意図したものになっているか、確認してください。
	16#00001010	指定した NX オブジェクトのデータサイズと、「WriteDat」のデータサイズが合っていません。
	16#00001101	指定したユニットが、正しくありません。 ユニットを確認してください。
	16#0000110B	読み出すデータのサイズが大きすぎます。 読み出すデータの指定が正しいか、確認してください。
	16#00001110	「Obj.Index」の値に対応したオブジェクトが存在しません。
	16#00001111	「Obj.Subindex」の値に対応したオブジェクトが存在しません。
	16#00002101	書込先 NX オブジェクトが、書込不可です。
	16#00002110	「WriteDat」の値が、書込先 NX オブジェクトの値の範囲外です。
	16#00002210	指定したユニットが、データの書込ができるモードになっていません。
	16#00002213	指定したユニットが、I/O チェック機能実行中のため、本命令を実行できません。 I/O チェック機能を終了してから、本命令を実行してください。
	16#00002230	指定したユニットの状態と、読出元または書込先 NX オブジェクトの値が、合っていません。 「Obj.Index」の値が 0x6000 ～ 0x6FFF または 0x7000 ～ 0x7FFF の場合、以下の対応をしてください。 • 読出元または書込先 NX オブジェクトを、I/O 割付設定からはずしてください。 • 指定したユニットの異常を解除してください。 • 指定したユニットを、データの書込ができるモードから解除してください。
	16#00002231	指定したユニットが、初期処理中のため本命令を実行できません。 正常動作中になるのを待って、本命令を実行してください。
	16#0000250F	ハードウェアアクセスに失敗しました。 再度、本命令を実行してください。
	16#00002601 16#00002602 16#00100000	指定したユニットには、本命令が使用できません。 ユニットのバージョンを、確認してください。
	16#00002603	本命令の実行に失敗しました。 再度、本命令を実行してください。 使用チャネル選択で、最低 1 チャネルが［有効］に設定されているか、確認してください。
	16#00002621	NX ユニットが、本命令を受け付ける状態にありません。 しばらく待ってから、再度、本命令を実行してください。
	16#00010000	指定したユニットが存在しません。 ユニットの構成が正しいか、確認してください。
	16#00110000	指定したポート No.が存在しません。 ユニットの構成が正しいか、確認してください。
	16#00120000 16#00130000 16#00150000 16#00160000	「UnitProxy」の値が、不正です。 指定する EtherCAT カプラユニットを示す変数を、設定しなおしてください。
	16#00140000	指定したノードアドレスが、不正です。 ユニットの構成が正しいか、確認してください。
	16#00300000 16#80010000	指定したユニットが、ビジー状態です。 再度、本命令を実行してください。

「ErrorID」の値	「ErrorIDEx」の値	異常内容
	16#00310000	指定したユニットが、接続対象外です。 ユニットのバージョンを、確認してください。
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 ～ 16#8FFF0000 16#90010000 ～ 16#FFFE0000	通信ネットワーク上で、異常が発生しました。 再度、本命令を実行してください。
	16#80020000 16#80030000 16#81030000 16#82000000	通信ネットワーク上で、異常が発生しました。 通信負荷を下げてください。
	16#80040000 16#81000000 16#82010000 16#82040000 16#82050000 16#90000000	通信ネットワーク上で、異常が発生しました。 ユニットやケーブルの接続を確認してください。 ユニットの電源が入っているか、確認してください。
16#2C01	16#00000000	同時に実行できる命令数を超えています。
16#2C02	16#00000000	通信実行中にタイムアウトが発生しました。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC201 のオブジェクトパラメータ「I/O リフレッシュ方式 1」の値を、読み出します。
EtherCAT カプラユニットのノードアドレスは 10 とします。

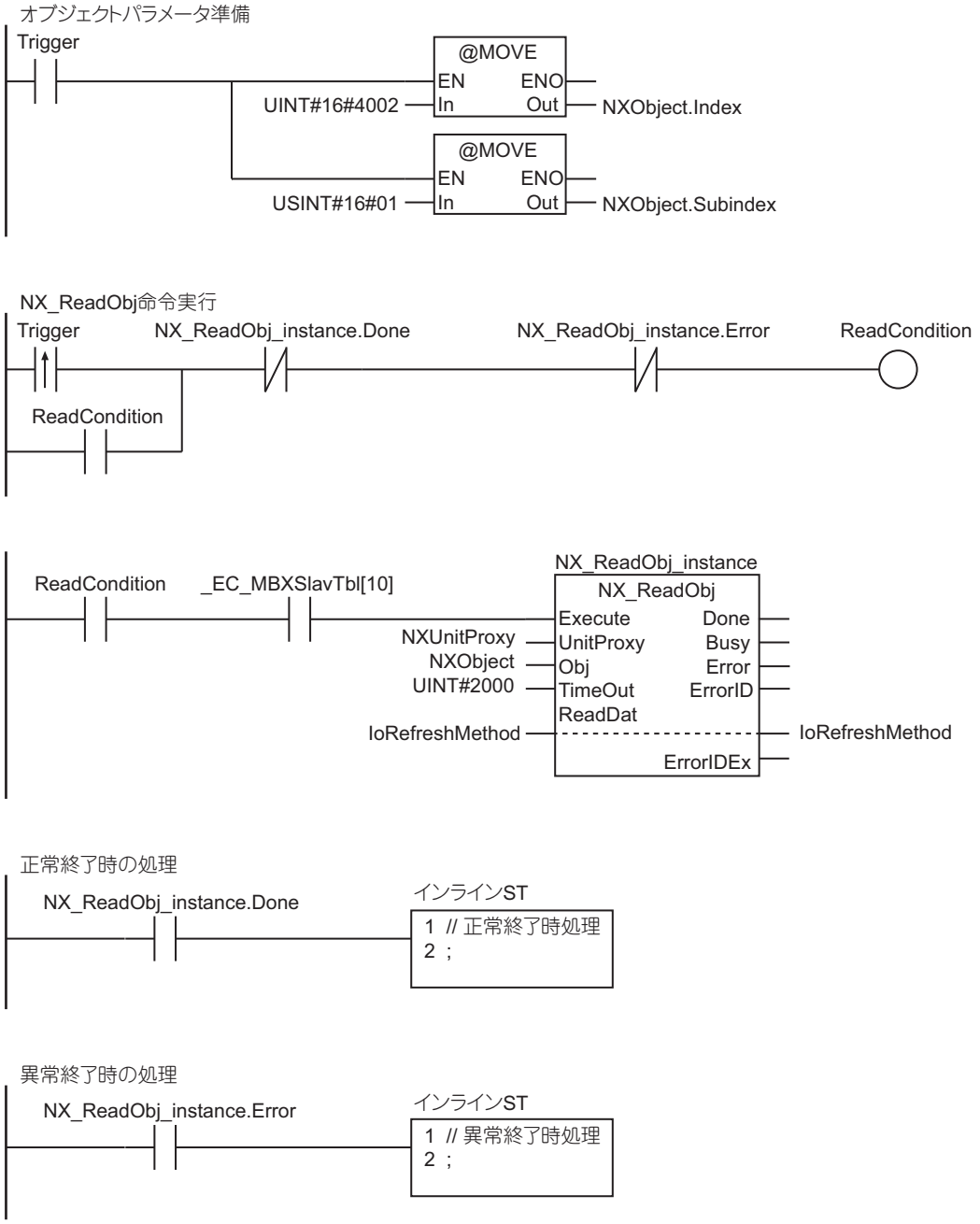
オブジェクトパラメータ「I/O リフレッシュ方式 1」のインデックスとサブインデックスの値は、以下のとおりです。

項目	値
インデックス	16#4002
サブインデックス	16#01

LD

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ReadCondition	BOOL	FALSE	データ読出実行条件
	NXUnitProxy	_sNXUNIT_ID		ユニット指定
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:= FALSE)	オブジェクトパラメータ
	IoRefreshMethod	USINT	0	読出データ
	NX_ReadObj_instance	NX_ReadObj		

外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ReadCondition	BOOL	FALSE	データ読出実行条件
	ReadGo	BOOL	FALSE	データ読出実行
	NXUnitProxy	_sNXUNIT_ID		ユニット指定

内部変数	名称	データ型	初期値	コメント
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:= FALSE)	オブジェクトパラメータ
	IoRefreshMethod	USINT	0	読出データ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	R_Trig_instance	R_Trig		
	NX_ReadObj_instance	NX_ReadObj		

外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル

```
// オブジェクトパラメータの準備
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
  NXObject.Index := UINT#16#4002;
  NXObject.Subindex := USINT#1;
END_IF;

// NX_ReadObj の実行
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);

// 命令実行後の処理
IF (NX_ReadObj_instance.Done=TRUE) THEN
  // 正常終了時処理
  NormalEnd := NormalEnd + UINT#1;
ELSIF (NX_ReadObj_instance.Error=TRUE) THEN
  // 異常終了時処理
  ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

IO-Link 通信命令

命令	名称	ページ
IOL_ReadObj	IO-Link デバイスオブジェクト読出	P.2-976
IOL_WriteObj	IO-Link デバイスオブジェクト書込	P.2-984

IOL_ReadObj

IO-Link デバイスのオブジェクトからデータを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
IOL_ReadObj	IO-Link デバイスオブジェクト読出	FB	IOL_ReadObj_instance IOL_ReadObj Execute / Done DevicePort / Busy DeviceObj / Error RetryCfg / ErrorID ReadDat ErrorType ReadSize	IOL_ReadObj_instance(Execute, DevicePort, DeviceObj, RetryCfg, ReadDat, Done, Busy, Error, ErrorID, ErrorType, ReadSize);

使用上の注意

本命令は、EtherNet/IP タイプの IO-Link マスタユニットでは使用できません。

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	–	–	–
DeviceObj	IO-Link デバイスのオブジェクトパラメータ		IO-Link デバイスのオブジェクト指定	–	–	–
RetryCfg	実行リトライ設定		命令実行リトライ設定	–	–	–
ReadDat	読出データ	入出力	IO-Link デバイスから読み出したデータ	データ型に従う	–	0
ErrorType	エラータイプ	出力	「ErrorID」が「4800Hex」時、IO-Link デバイスが返信したエラーコードを格納	16#0000 ～ 16#FFFF	–	–
ReadSize	読出データサイズ		「ReadDat」に格納したデータのサイズ	10#1 ～ 10#232	バイト	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-977)」参照																			
DeviceObj	構造体_sIOLOBJ_ACCESS　詳細は「機能 (P.2-977)」参照																			
RetryCfg	構造体_sIOL_RETRY_CFG　詳細は「機能 (P.2-977)」参照																			
ReadDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	配列全体も指定可能																			
ErrorType			○																	
ReadSize							○													

機能

IO-Link デバイスのオブジェクトのデータを読み出します。

読み出し対象の IO-Link デバイスを接続している IO-Link マスタユニットと、ポート番号を入力変数「DevicePort」で指定します。
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOAD_ID	−	−	−
Reserved	予約	予約	−	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2 3: ポート 3 4: ポート 4 5: ポート 5 6: ポート 6 7: ポート 7 8: ポート 8	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX タイプの IO-Link マスタユニットの場合は、_DeviceNXUnit を設定してください。EtherCAT タイプの IO-Link マスタユニットの場合は、_DeviceEcatSlave を設定してください。

指定した種別により、デバイス指定に使用する変数が決まっています。

この命令では、以下のとおりです。
NX タイプを指定する場合は、「NxUnit」を使用してデバイスを指定します。この場合、「EcatSlave」は使用しません。
「NxUnit」には、指定するデバイスに割り付けられたデバイス変数を渡してください。
EtherCAT タイプを指定する場合は、「EcatSlave」を使用してデバイスを指定します。この場合、「NxUnit」は使用しません。
「EcatSlave」には、指定するデバイスに割り付けられたデバイス変数を渡してください。

「PortNo」で IO-Link デバイスを接続しているポート番号を指定します。
IO-Link マスタユニットのタイプによりポート数が異なります。
NX タイプ: 1～4
EtherCAT タイプ: 1～8

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定

読み出し対象の IO-Link デバイスのオブジェクトパラメータを入力変数「DeviceObj」で指定します。
「DeviceObj」のデータ型は、構造体_sIOLOBJ_ACCESS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DeviceObj	IO-Link デバイスのオブジェクトパラメータ	IO-Link デバイスのオブジェクト指定	_sIOLOBJ_ACCESS	–	–	–
Index	インデックス	インデックス	UINT	データ型に従う	–	–
Subindex	サブインデックス	インデックス全体を読み出す場合は「0」を指定	USINT	データ型に従う	–	–

命令実行のリトライの設定を、入力変数「RetryCfg」で指定します。
「RetryCfg」のデータ型は、構造体_sIOL_RETRY_CFG です。仕様は、以下のとおりです。

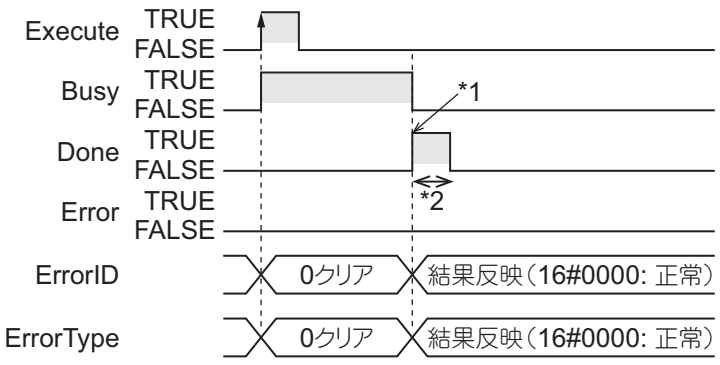
変数	名称	内容	データ型	有効範囲	単位	初期値
RetryCfg	実行リトライ設定	命令実行リトライ設定	_sIOL_RETRY_CFG	–	–	–
TimeOut	タイムアウト時間	タイムアウト時間 「0」の場合は、2.0s	TIME	0 ～ 300s	–	T#2.0s
RetryNum	リトライ回数	タイムアウト発生時のリトライ回数指定 「0」の場合は、3 回	UINT	データ型に従う	回	3

IO-Link デバイスから読み出したデータは、入出力変数「ReadDat」に格納します。

タイミングチャート

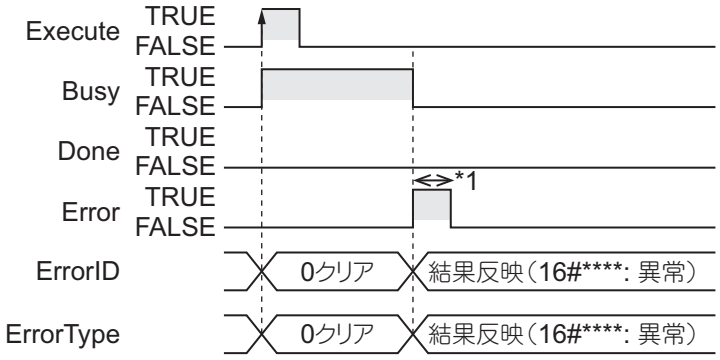
タイミングチャートを、以下に示します。

● 正常終了時



*1. 読み出し完了
*2. タスク周期

● **異常終了時**



*1. タスク周期

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl	メッセージ通信可能スレーブテーブル	ARRAY[1..512] OF BOOL	メッセージ通信が可能なスレーブ一覧です。 スレーブノードアドレス順のテーブルで表します。 TRUE: 通信可 FALSE: 通信不可

使用上の注意

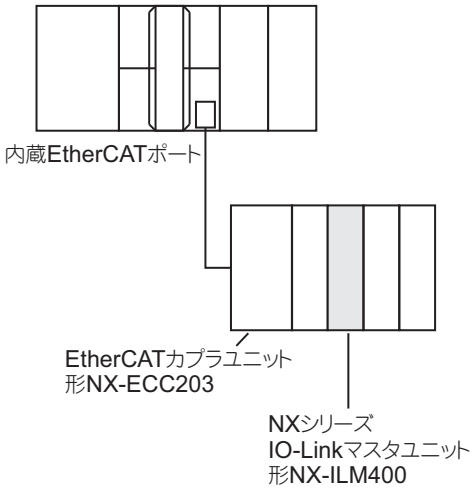
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「DevicePort.NxUnit」、「DevicePort.EcatSlave」には、Sysmac Studio の I/O マップで、IO-Link マスタユニットに割り付けたデバイス変数を指定します。デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。
- 「ReadDat」のサイズは、実際に読み出すオブジェクトのサイズよりも大きいサイズの変数を指定してください。
- 「ReadDat」が STRING 型の場合、実際に読み出す文字列に NULL 文字を加えたサイズの変数を指定してください。
- 「ReadDat」が STRING 型の場合、「ReadSize」には NULL 文字を含まないサイズを出力します。
- 「ReadDat」に渡すパラメータは、必ず変数にしてください。定数を渡すと、ビルド時に異常になります。
- 1 つの IO-Link マスタユニットに同時に実行できる命令数は、NX タイプの IO-Link マスタユニット、EtherCAT タイプの IO-Link マスタユニットいずれも 1 個です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 本命令は、「Execute」の立ち上がりで動作します。「Execute」が常に TRUE の場合は実行しません。
- IOL_ReadObj 命令、および IOL_WriteObj 命令の定義できるインスタンス数は、64 個までです。
- つぎの場合、異常となります。
  a）「DevicePort.NxUnit」、「DevicePort.EcatSlave」に範囲外の値を指定した場合。
  b）「ReadDat」 のサイズよりも、読み出し元の IO-Link デバイスのオブジェクトのサイズが大きい場合。本異常時は、「ReadDat」に読み出したデータを格納しません。
  c）IO-Link デバイスからのエラーレスポンスを受信した場合。
  上位 8 ビットが「ErrorCode」、下位 8 ビットが「AdditionalCode」です。

「ErrorCode」、「AdditionalCode」については、IO-Link Communication Specification の Error type 仕様を確認してください。Error type 仕様については、IO-Link Consortium から入手できます。
http://www.io-link.com/

d) 指定した IO-Link マスタユニットが存在しない場合。
e) IO-Link マスタのメッセージ処理数オーバーの場合。IO-Link マスタが他のアプリケーションからのメッセージを処理中のため実行できません。
f) 指定した IO-Link マスタユニットが、メッセージを受け付ける状態にない場合。
g) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
h) 通信実行中にタイムアウトが発生した場合。
i) IO-Link マスタユニットの指定したポートが IO-Link モードでない場合。ポートが無効、または SIO モードの場合。
j) IO-Link マスタユニットの指定したポートに IO-Link デバイスが接続されていない場合。
k) IO-Link マスタユニットの指定したポートの IO 電源が入っていない場合。
l) IO-Link マスタユニットの指定したポートに照合異常、または通信異常が発生している場合。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に IO-Link マスタユニット形 NX-ILM400 を接続する構成とします。



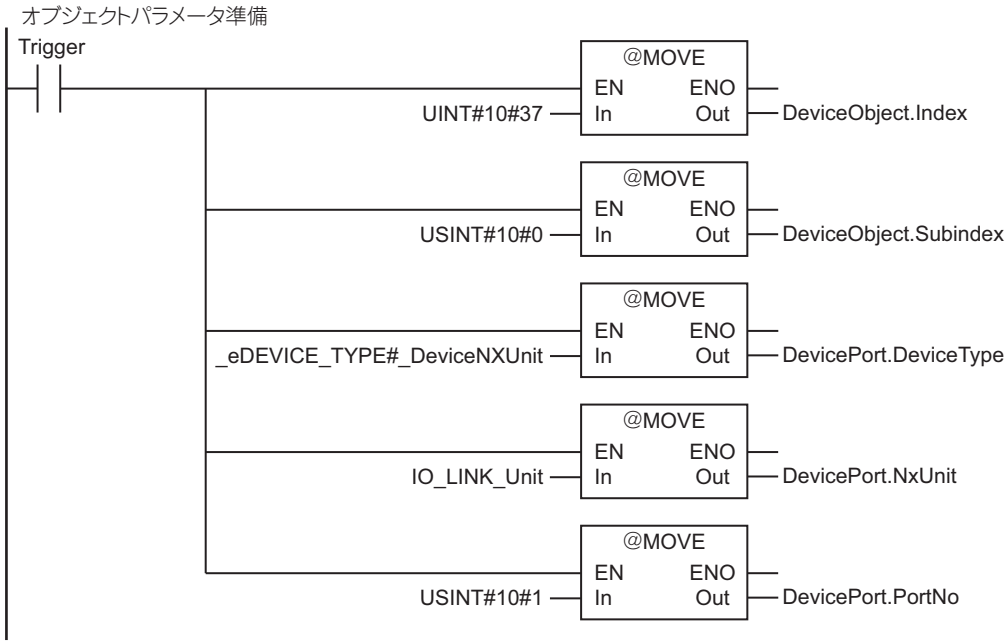
形 NX-ILM400 のポート 1 に接続の光電センサ（形 E3Z）のエラーログ(Index:37/Subindex:0)30byte を読み出します。読み出しデータは、「DeviceErrorLog」に格納します。
形 NX-ECC203 のノードアドレスは 10 とします。

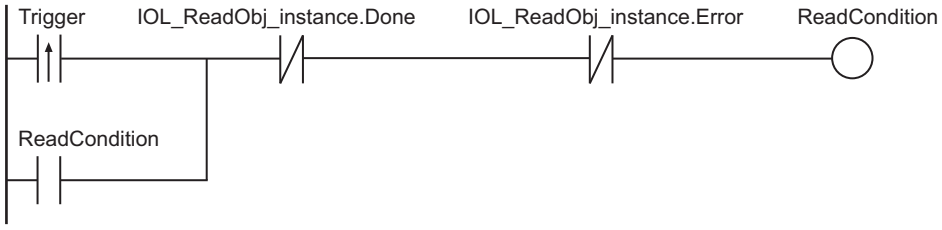
LD

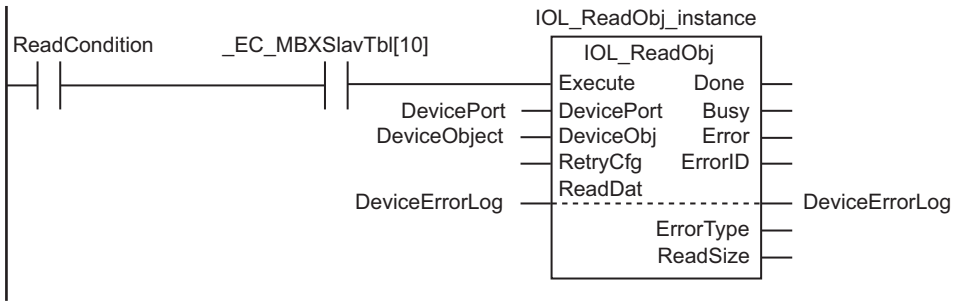
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ReadCondition	BOOL	FALSE	データ読み出し実行条件
	DevicePort	_sDEVICE_PORT		

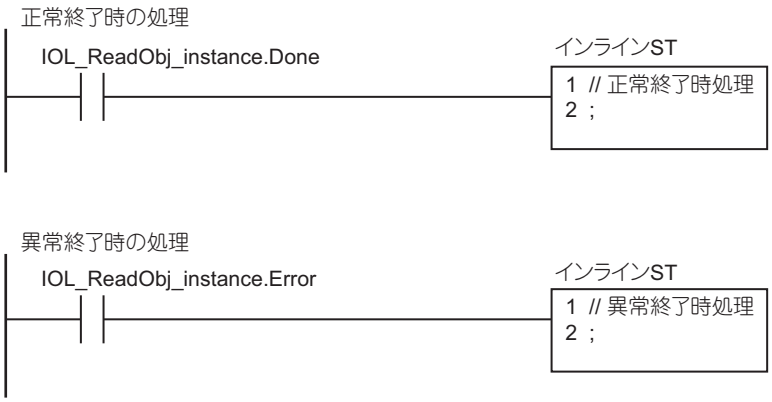
内部変数	名称	データ型	初期値	コメント
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	DeviceErrorLog	ARRAY[1..30] OF BYTE		読み出しデータ
	IOL_ReadObj_instance	IOL_ReadObj		

外部変数	名称	コンスタント	初期値	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル
	IO_LINK_Unit	☑	構造体メンバ「NxUnit」に初期値として形 NX-ILM400 を指定するデバイス変数を設定しておく	









ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ReadGo	BOOL	FALSE	データ読み出し実行
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	DeviceErrorLog	ARRAY[1..30] OF BYTE		読み出しデータ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	R_Trig_instance	R_Trig		
	IOL_ReadObj_instance	IOL_ReadObj		

外部変数	名称	コンスタント	初期値	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル
	IO_LINK_Unit	☑	構造体メンバ「NxUnit」に初期値として形 NX-ILM400 を指定するデバイス変数を設定しておく	

```
// オブジェクトパラメータの準備
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;

//IOL_ReadObj 命令実行
IOL_ReadObj_instance(
    Execute := ReadGo,
    DevicePort:= DevicePort,
    DeviceObj := DeviceObject,
    ReadDat :=DeviceErrorLog);

// 命令実行後の処理
IF (IOL_ReadObj_instance.Done=TRUE) THEN
    // 正常終了時処理
    NormalEnd := NormalEnd + UINT#1;
ELSIF (IOL_ReadObj_instance.Error=TRUE) THEN
    // 異常終了時処理
    ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

IOL_WriteObj

IO-Link デバイスのオブジェクトにデータを書き込みます。

命令	名称	FB /FUN	グラフィック表現	ST 表現
IOL_WriteObj	IO-Link デバイスオブジェクト書込	FB	IOL_WriteObj_instance IOL_WriteObj Execute / Done DevicePort / Busy DeviceObj / Error RetryCfg / ErrorID WriteDat / ErrorType WriteSize	IOL_WriteObj_instance(Execute, DevicePort, DeviceObj, RetryCfg, WriteDat, WriteSize, Done, Busy, Error, ErrorID, ErrorType);

使用上の注意

本命令は、EtherNet/IP タイプの IO-Link マスタユニットでは使用できません。

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	−	−	−
DeviceObj	IO-Link デバイスのオブジェクトパラメータ		IO-Link デバイスのオブジェクト指定	−	−	−
RetryCfg	実行リトライ設定		命令実行リトライ設定	−	−	−
WriteDat	書込データ		IO-Link デバイスに書き込むデータ	データ型に従う	−	−
WriteSize	書込データサイズ		書き込みデータサイズ *1	10#1 ～ 10#232	バイト	−
ErrorType	エラータイプ	出力	「ErrorID」が「4800Hex」時、IO-Link デバイスが返信したエラーコードを格納	16#0000 ～ 16#FFFF	−	−

*1. 書き込みデータが BOOL 型の場合は「1」を、BOOL 型配列の場合は、配列の要素数を入力してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-985)」参照																			
DeviceObj	構造体_sIOLOBJ_ACCESS　詳細は「機能 (P.2-985)」参照																			
RetryCfg	構造体_sIOL_RETRY_CFG　詳細は「機能 (P.2-985)」参照																			
WriteDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	配列全体も指定可能																			
WriteSize							○													
ErrorType			○																	

機能

IO-Link デバイスのオブジェクトのデータを書き込みます。

書き込み対象の IO-Link デバイスを接続している IO-Link マスタユニットと、ポート番号を入力変数「DevicePort」で指定します。
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOAD_ID	−	−	−
Reserved	予約	予約	−	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2 3: ポート 3 4: ポート 4 5: ポート 5 6: ポート 6 7: ポート 7 8: ポート 8	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX タイプの IO-Link マスタユニットの場合は、_DeviceNXUnit を設定してください。EtherCAT タイプの IO-Link マスタユニットの場合は、_DeviceEcatSlave を設定してください。

指定した種別により、デバイス指定に使用する変数が決まっています。

この命令では、以下のとおりです。
NX タイプを指定する場合は、「NxUnit」を使用してデバイスを指定します。この場合、「EcatSlave」は使用しません。
「NxUnit」には、指定するデバイスに割り付けられたデバイス変数を渡してください。
EtherCAT タイプを指定する場合は、「EcatSlave」を使用してデバイスを指定します。この場合、「NxUnit」は使用しません。
「EcatSlave」には、指定するデバイスに割り付けられたデバイス変数を渡してください。

「PortNo」で IO-Link デバイスを接続しているポート番号を指定します。
IO-Link マスタユニットのタイプによりポート数が異なります。
NX タイプ: 1～4
EtherCAT タイプ: 1～8

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定

書き込み対象の IO-Link デバイスのオブジェクトパラメータを入力変数「DeviceObj」で指定します。

「DeviceObj」のデータ型は、構造体_sIOLOBJ_ACCESS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DeviceObj	IO-Link デバイスのオブジェクトパラメータ	IO-Link デバイスのオブジェクト指定	_sIOLOBJ_ACCESS	–	–	–
Index	インデックス	インデックス	UINT	データ型に従う	–	–
Subindex	サブインデックス	インデックス全体を読み出す場合は「0」を指定	USINT	データ型に従う	–	–

命令実行のリトライの設定は、入力変数「RetryCfg」で指定します。
「RetryCfg」のデータ型は、構造体_sIOL_RETRY_CFG です。仕様は、以下のとおりです。

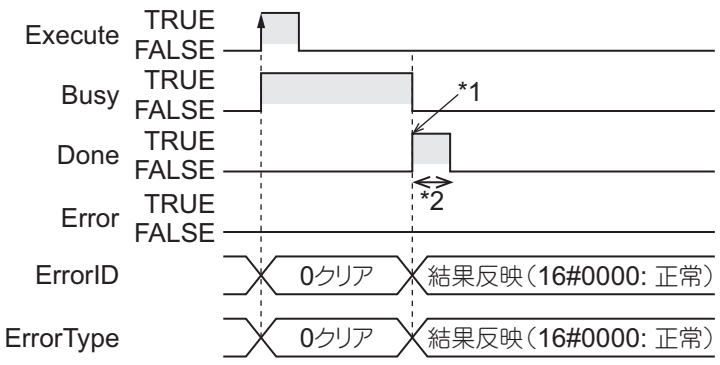
変数	名称	内容	データ型	有効範囲	単位	初期値
RetryCfg	実行リトライ設定	命令実行リトライ設定	_sIOL_RETRY_CFG	–	–	–
TimeOut	タイムアウト時間	タイムアウト時間 「0」の場合は、2.0s	TIME	0 ～ 300s	–	T#2.0s
RetryNum	リトライ回数	タイムアウト発生時のリトライ回数指定 「0」の場合は、3 回	UINT	データ型に従う	回	3

IO-Link デバイスに書き込むデータは、入力変数「WriteDat」で指定します。

タイミングチャート

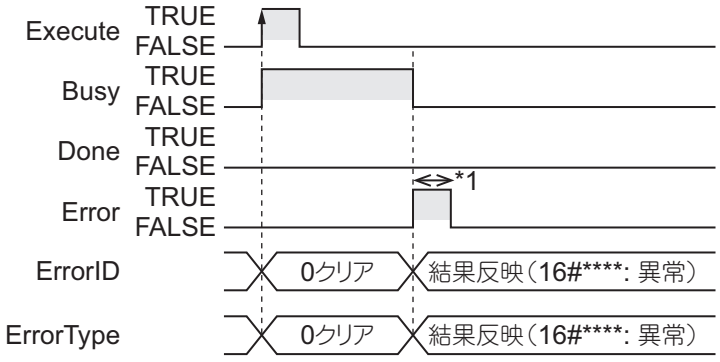
タイミングチャートを、以下に示します。

● 正常終了時



*1. 書き込み完了
*2. タスク周期

● **異常終了時**



*1. タスク周期

関連するシステム定義変数

変数名	名称	データ型	内容
_EC_MBXSlavTbl	メッセージ通信可能スレーブテーブル	ARRAY[1..512] OF BOOL	メッセージ通信が可能なスレーブ一覧です。 スレーブノードアドレス順のテーブルで表します。 TRUE: 通信可 FALSE: 通信不可

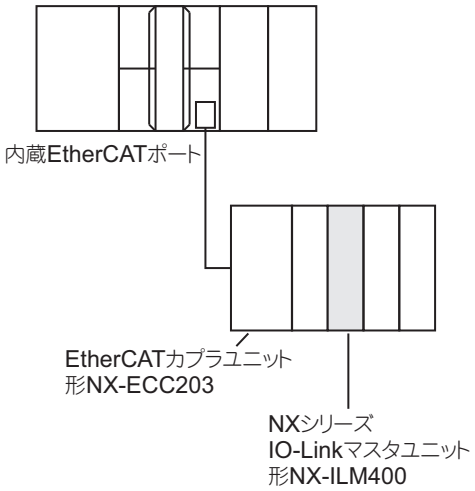
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「DevicePort.NxUnit」、「DevicePort.EcatSlave」には、Sysmac Studio の I/O マップで、IO-Link マスタユニットに割り付けたデバイス変数を指定します。デバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。
- 「WriteDat」に渡すパラメータは、必ず変数にしてください。定数を渡すと、ビルド時に異常になります。
- 1 つの IO-Link マスタユニットに同時に実行できる命令数は、NX タイプの IO-Link マスタユニット、EtherCAT タイプの IO-Link マスタユニットいずれも 1 個です。
- イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 本命令は、「Execute」の立ち上がりで動作します。「Execute」が常に TRUE の場合は実行しません。
- IOL_ReadObj 命令、および IOL_WriteObj 命令の定義できるインスタンス数は、64 個までです。
- つぎの場合、異常となります。
  a)「DevicePort.NxUnit」、「DevicePort.EcatSlave」に範囲外の値を指定した場合。
  b)「TimeOut」が有効範囲外の場合。
  c)「DevicePort」のデータ型が不正の場合。
  d)「WriteDat」に 232 バイトを超えるデータが指定されている場合。
  e) IO-Link デバイスからのエラーレスポンスを受信した場合。
  上位 8 ビットが「ErrorCode」、下位 8 ビットが「AdditionalCode」です。
  「ErrorCode」、「AdditionalCode」については、IO-Link Communication Specification の Error type 仕様を確認してください。Error type 仕様については、IO-Link Consortium から入手できます。
  http://www.io-link.com/
  f) 指定した IO-Link マスタユニットが存在しない場合。

g) IO-Link マスタのメッセージ処理数オーバーの場合。IO-Link マスタが他のアプリケーションからのメッセージを処理中のため実行できません。
h) 指定した IO-Link マスタユニットが、メッセージを受け付ける状態にない場合。
i) EC_CoESDOWrite 命令、EC_CoESDORead 命令、EC_StartMon 命令、EC_StopMon 命令、EC_SaveMon 命令、EC_CopyMon 命令、EC_DisconnectSlave 命令、EC_ConnectSlave 命令、EC_ChangeEnableSetting 命令、IOL_ReadObj 命令、IOL_WriteObj 命令を、同時に 32 個を超えて実行した場合。
j) 通信実行中にタイムアウトが発生した場合。
k) IO-Link マスタユニットの指定したポートが IO-Link モードでない場合。ポートが無効、または SIO モードの場合。
l) IO-Link マスタユニットの指定したポートに IO-Link デバイスが接続されていない場合。
m) IO-Link マスタユニットの指定したポートの IO 電源が入っていない場合。
n) IO-Link マスタユニットの指定したポートに照合異常、または通信異常が発生している場合。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に IO-Link マスタユニット形 NX-ILM400 を接続する構成とします。

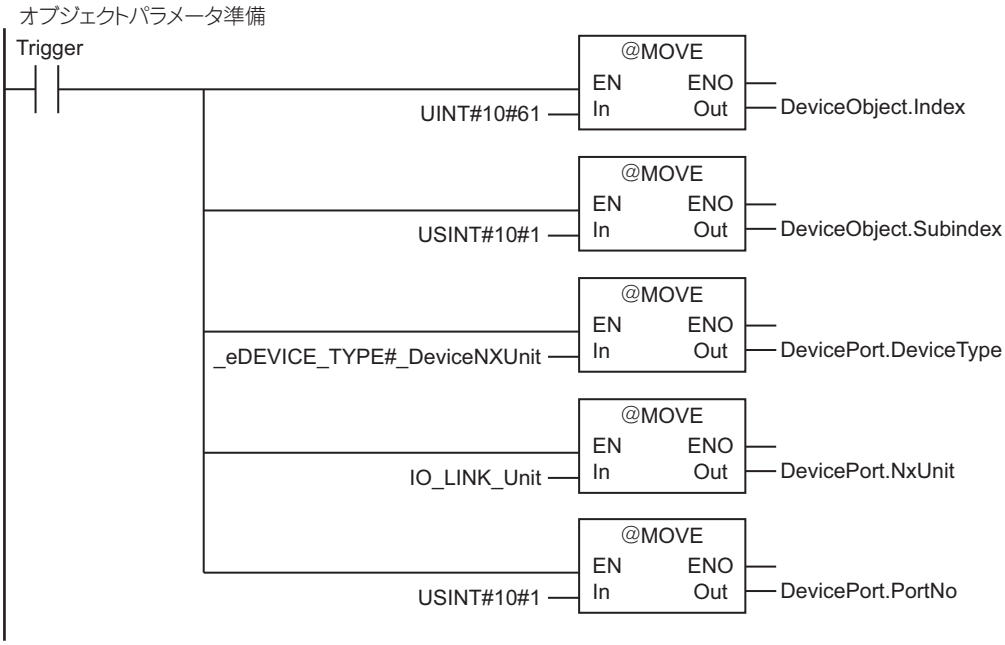


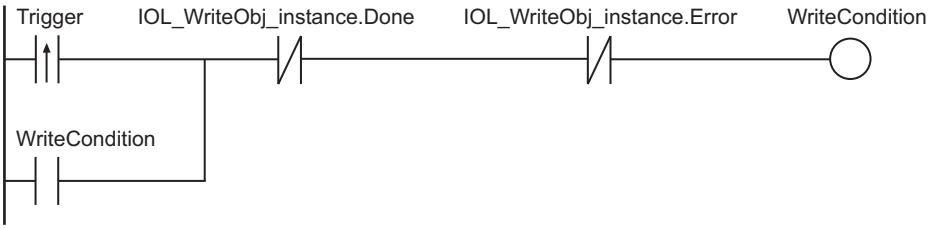
形 NX-ILM400 のポート 1 に接続している光電センサ（形 E3Z）の制御出力 1 動作モード切替 (SwitchPoint Logic Output1)(Index:61/Subindex:1)1byte に「01」を書き込みます。書き込みデータは、「SwitchPoint」に格納します。
形 NX-ECC203 のノードアドレスは 10 とします。

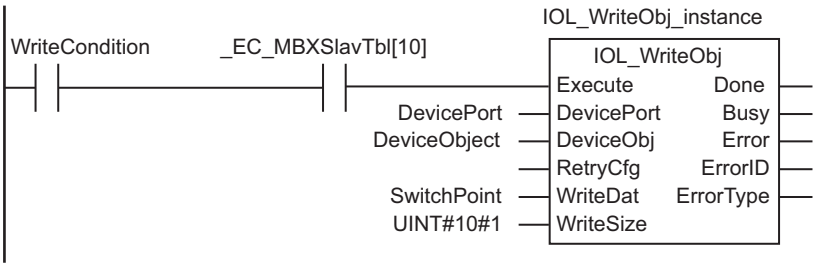
LD

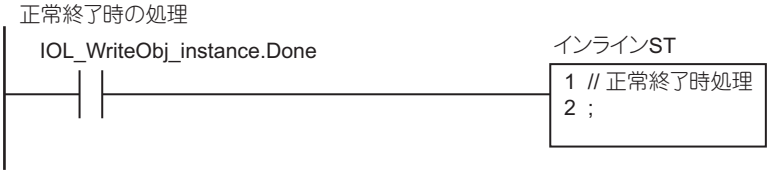
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteCondition	BOOL	FALSE	データ書き込み実行条件
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	SwitchPoint	USINT	USINT#01	書き込みデータ
	IOL_WriteObj_instance	IOL_WriteObj		

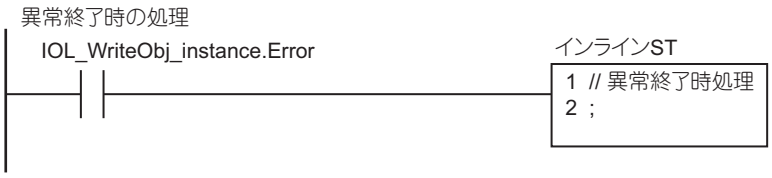
外部変数	名称	コンスタント	初期値	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル
	IO_LINK_Unit	☑	構造体メンバ「NxUnit」に初期値として形 NX-ILM400 を指定するデバイス変数を設定しておく	











ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteGo	BOOL	FALSE	データ書き込み実行
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	SwitchPoint	USINT	USINT#01	書き込みデータ
	NormalEnd	UINT	0	正常終了
	ErrorEnd	UINT	0	異常終了
	R_Trig_instance	R_Trig		
	IOL_WriteObj_instance	IOL_WriteObj		

外部変数	名称	コンスタント	初期値	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL	メッセージ通信可能スレーブテーブル
	IO_LINK_Unit	☑	構造体メンバ「NxUnit」に初期値として形 NX-ILM400 を指定するデバイス変数を設定しておく	

```
// オブジェクトパラメータの準備
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;

//IOL_WriteObj 命令実行
IOL_WriteObj_instance(
    Execute := WriteGo,
    DevicePort:= DevicePort,
```

```
    DeviceObj := DeviceObject,
    WriteDat := SwitchPoint,
    WriteSize := UINT#10#1);

// 命令実行後の処理
IF (IOL_WriteObj_instance.Done=TRUE) THEN
    // 正常終了時処理
    NormalEnd := NormalEnd + UINT#1;
ELSIF (IOL_WriteObj_instance.Error=TRUE) THEN
    // 異常終了時処理
    ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

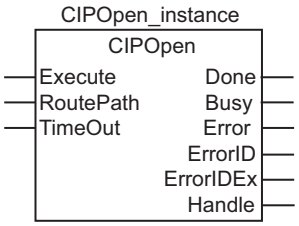
EtherNet/IP 通信命令

命令	名称	ページ
CIPOpen	CIPClass3(Large_Forward_Open)コネクション確立	P.2-995
CIPOpenWithDataSize	CIPClass3 コネクション確立（サイズ指定あり）	P.2-1004
CIPRead	変数読出（Class3 Explicit メッセージ）	P.2-1007
CIPWrite	変数書込（Class3 Explicit メッセージ）	P.2-1012
CIPSend	任意 Explicit メッセージ送信(Class3)	P.2-1018
CIPClose	CIP Class3 コネクションの切断	P.2-1023
CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	P.2-1025
CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	P.2-1030
CIPUCMMSend	任意 Explicit メッセージ送信(UCMM)	P.2-1036
SktUDPCreate	UDP ソケットクリエイト	P.2-1047
SktUDPRcv	UDP ソケット受信	P.2-1055
SktUDPSend	UDP ソケット送信	P.2-1058
SktTCPAccept	TCP ソケットアクセプト	P.2-1061
SktTCPConnect	TCP ソケットコネクト	P.2-1064
SktTCPRcv	TCP ソケット受信	P.2-1072
SktTCPSend	TCP ソケット送信	P.2-1075
SktGetTCPStatus	TCP ソケットのステータス読出	P.2-1078
SktClose	TCP/UDP ソケットクローズ	P.2-1081
SktClearBuf	TCP/UDP ソケット受信バッファクリア	P.2-1084
SktSetOption	TCP ソケットオプション設定	P.2-1087
ChangeIPAdr	IP アドレス変更	P.2-1092
ChangeFTPAccount	FTP アカウント変更	P.2-1100
FTPGetFileList	FTP サーバのファイルリスト取得	P.2-1104

命令	名称	ページ
FTPGetFile	FTP サーバからのファイルダウンロード	P.2-1117
FTPPutFile	FTP サーバへのファイルアップロード	P.2-1125
FTPRemoveFile	FTP サーバのファイル削除	P.2-1135
FTPRemoveDir	FTP サーバのディレクトリ削除	P.2-1144

CIPOpen

指定した相手ノードに対して、データ長 1994 バイトで、CIP の Class3(Large_Forward_Open)コネクションを確立します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPOpen	CIP Class3(Large_Forward_Open)コネクション確立	FB		CIPOpen_instance(Execute, RoutePath, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, Handle);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
RoutePath	ルートパス	入力	ルートパス	データ型に従う	−	−
TimeOut	タイムアウト時間		コネクションタイムアウト時間	1 ～ 65535	0.1s	20 (2s)
Handle	ハンドル	出力	ハンドル	−	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				○
TimeOut							○													
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-995)」参照																			

機能

ルートパス「RoutePath」で指定した、CIP ネットワーク上の相手ノードに対して、データ長 1994 バイトで、CIP の Class3(Large_Forward_Open)コネクションを確立します。
コネクションが確立すると、ハンドル「Handle」が出力されます。

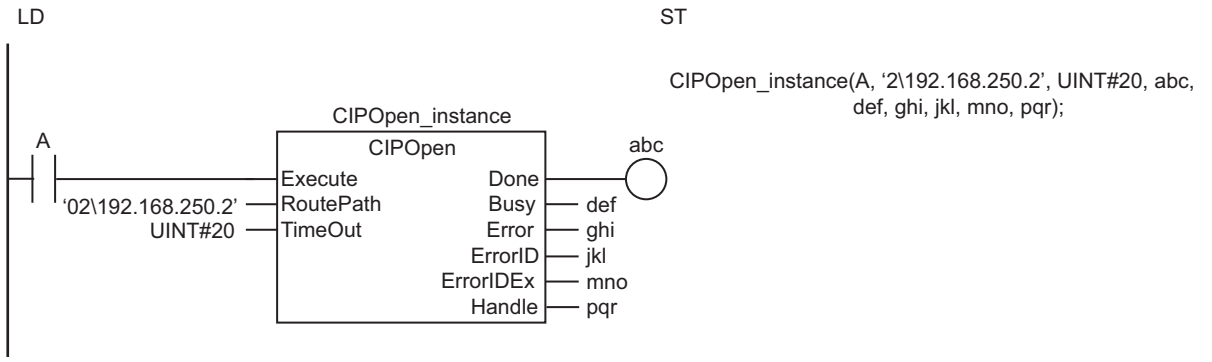
コネクションタイムアウト時間は、「TimeOut」で指定します。
CIPSend 命令、CIPWrite 命令、CIPRead 命令のいずれかを実行した後、コネクションタイムアウト時間以内に相手ノードからのレスポンスが返ってこないと、通信失敗と判断されます。
コネクションタイムアウト時間は、CIPRead 命令、CIPWrite 命令、CIPSend 命令が実行して、相手ノードからのレスポンスが返ってくると、リセットされます。

「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

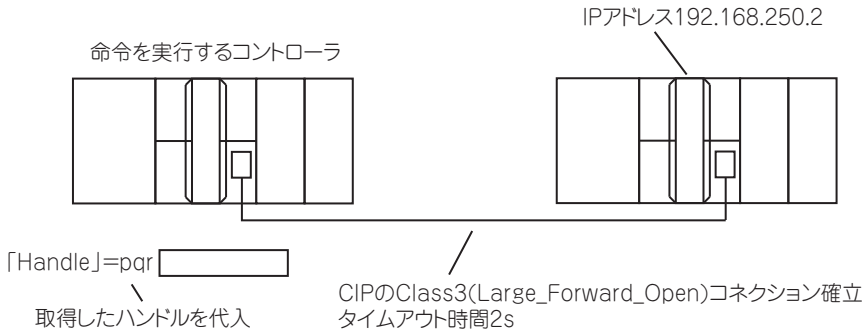
変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	−	−	−
Handle	ハンドル	ハンドル	UDINT	データ型に従う	−	−

「RoutePath」='02\192.168.250.2'、「TimeOut」=UINT#20 の場合の記述例を、以下に示します。

IP アドレス'192.168.250.2'の相手ノードに対して、CIP の Class3(Large_Forward_Open)コネクションを確立します。タイムアウト時間は 2s です。変数「pqr」にハンドルが代入されます。



「RoutePath」で指定したCIPネットワーク上の相手ノードに対して、CIPの Class3(Large_Forward_Open)コネクションを確立します。



「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。「ErrorIDEx」の値と意味は、相手ノードによって規定されます。相手ノードのマニュアルを参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

- CIP 通信の詳細は、以下のマニュアルを参照してください。
  a)『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』
- Forward Open でコネクションを確立する場合、または任意のデータ長としてコネクションを確立する場合は、「CIPOpenWithDataSize (P.2-1004)」命令を使用してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令または CIPOpenWithDataSize 命令を、CIPRead 命令、CIPWrite 命令、CIPSend 命令の前に実行しておく必要があります。
- 本命令でコネクション確立後、最初のタイムアウト時間は、「TimeOut」の値が 100(10s)以下であっても、10s となります。
- 本命令で確立したコネクションは、CIPClose 命令を使用してクローズしてください。
- コネクションがタイムアウトになっても、本命令で生成したハンドルは残ります。必ず、CIPClose 命令を使用して、コネクションをクローズしてください。
- 本命令で生成したハンドルは、プログラムモード変更時に無効になります。
- 同時に生成できるハンドルは、最大 32 個です。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「TimeOut」の値が、有効範囲外の場合。
  b)「RoutePath」の文字列が、不正の場合。
  c) CIP 関連命令が、同時に 32 個を超えて実行された場合。
  d) CIPClass コネクションのリソース（32 個）を超えて、コネクション開設をしようとした場合。
  e) コネクション確立の応答がない場合。
  f) コネクションを確立する相手ノードが、Large_Forward_Open に対応していない場合。
  g) 自 IP アドレスの、設定異常が発生している場合。
  h) IP 重複異常が発生している場合。
  i) TCP コネクションがすべて使用されている場合。
  j) BOOTP サーバ異常が発生している状態で、本命令が実行された場合。



参考

「Error」が TRUE となっても「Handle」の値を変更しません。

サンプルプログラム

CIP の Class3 メッセージを使用して、変数書込、変数読出、メッセージ送信を行います。
コントローラどうしは、EtherNet/IP ネットワークで接続しており、相手ノードの IP アドレスは 192.168.250.2 とします。
処理の手順は、以下のようになります。

1 **CIPOpen 命令を使用して、Class3(Large_Forward_Open)コネクションを確立します。タイムアウト時間は、2s とします。**

2 **CIPWrite 命令を使用して、相手ノードの変数に値を書き込みます。他局変数名は「WritingDat」、書き込む値は、変数「WriteDat」の中身とします。**
「WritingDat」は、相手ノードでグローバル変数として定義されており、かつ、ネットワーク公開されていることが必要です。

3 **CIPRead 命令を使用して、相手ノードの変数の値を読み出します。他局変数名は「OriginalDat」、読み出した値の格納先は、変数「ReadDat」とします。**

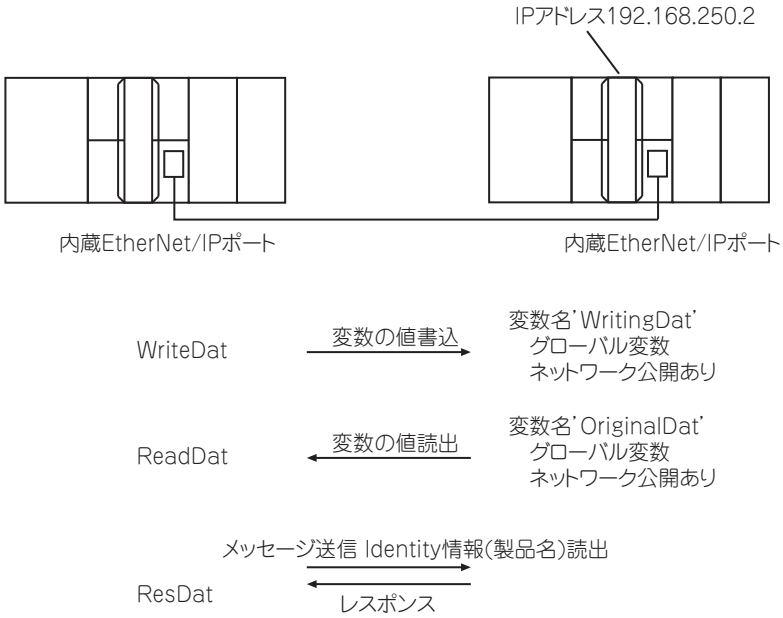
「OriginalDat」は、相手ノードでグローバル変数として定義されており、かつ、ネットワーク公開されていることが必要です。

4 CIPSend 命令を使用して、相手ノードに Explicit メッセージを送信します。メッセージの内容は、Identity 情報（製品名）読出です。

そのときのクラス ID、インスタンス ID、アトリビュート ID、サービスコードは、以下のとおりです。レスポンスデータは、変数「ResDat」に格納します。

項目	値
クラス ID	1
インスタンス ID	1
アトリビュート ID	7
サービスコード	16#0E

5 CIPClose 命令を使用して、Class3 コネクションを切断します。

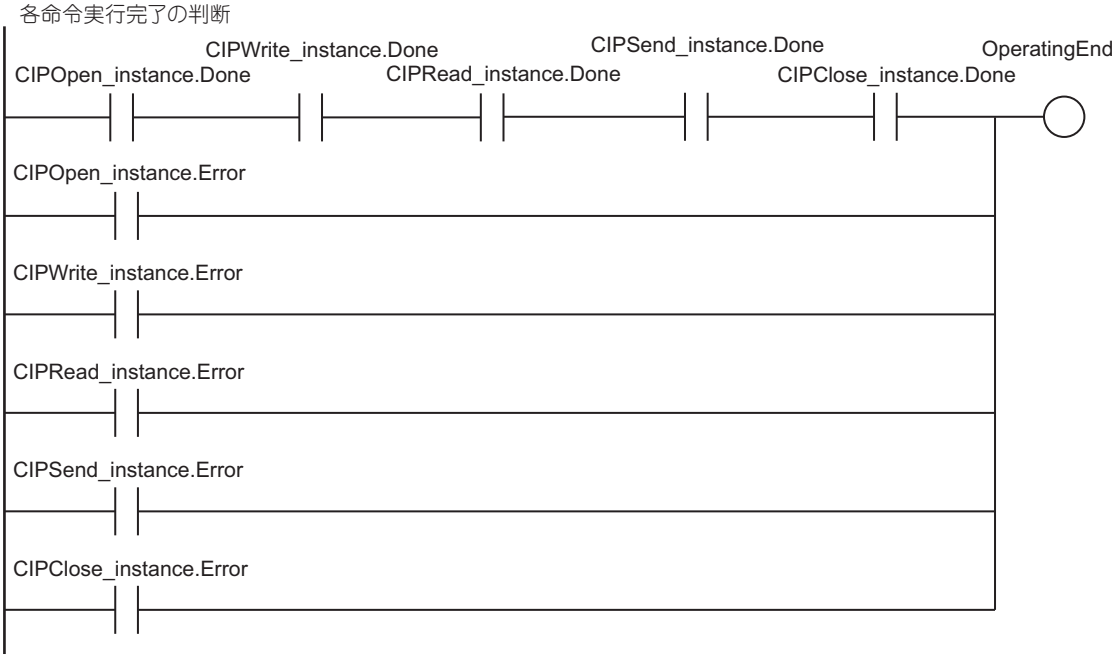


LD

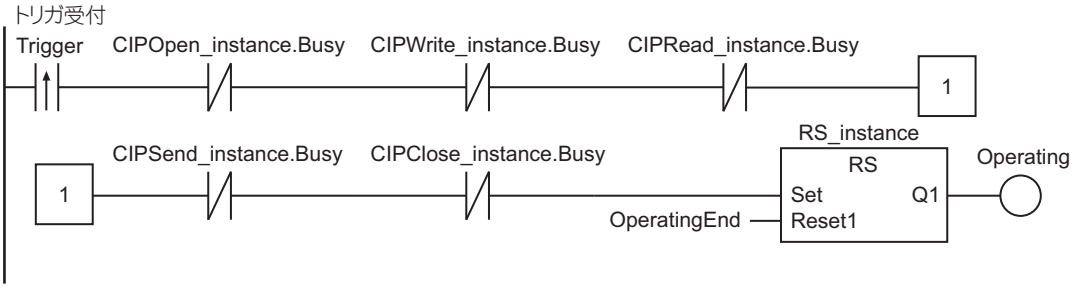
名称	データ型	初期値	コメント
OperatingEnd	BOOL	FALSE	処理終了
Trigger	BOOL	FALSE	実行条件
Operating	BOOL	FALSE	処理中
WriteDat	INT	1234	書込データ
ReadDat	INT	0	読出データ
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	リクエストパス
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	レスポンスデータ
Dummy	BYTE	16#0	ダミー
RS_instance	RS		
CIPOpen_instance	CIPOpen		
CIPWrite_instance	CIPWrite		
CIPRead_instance	CIPRead		

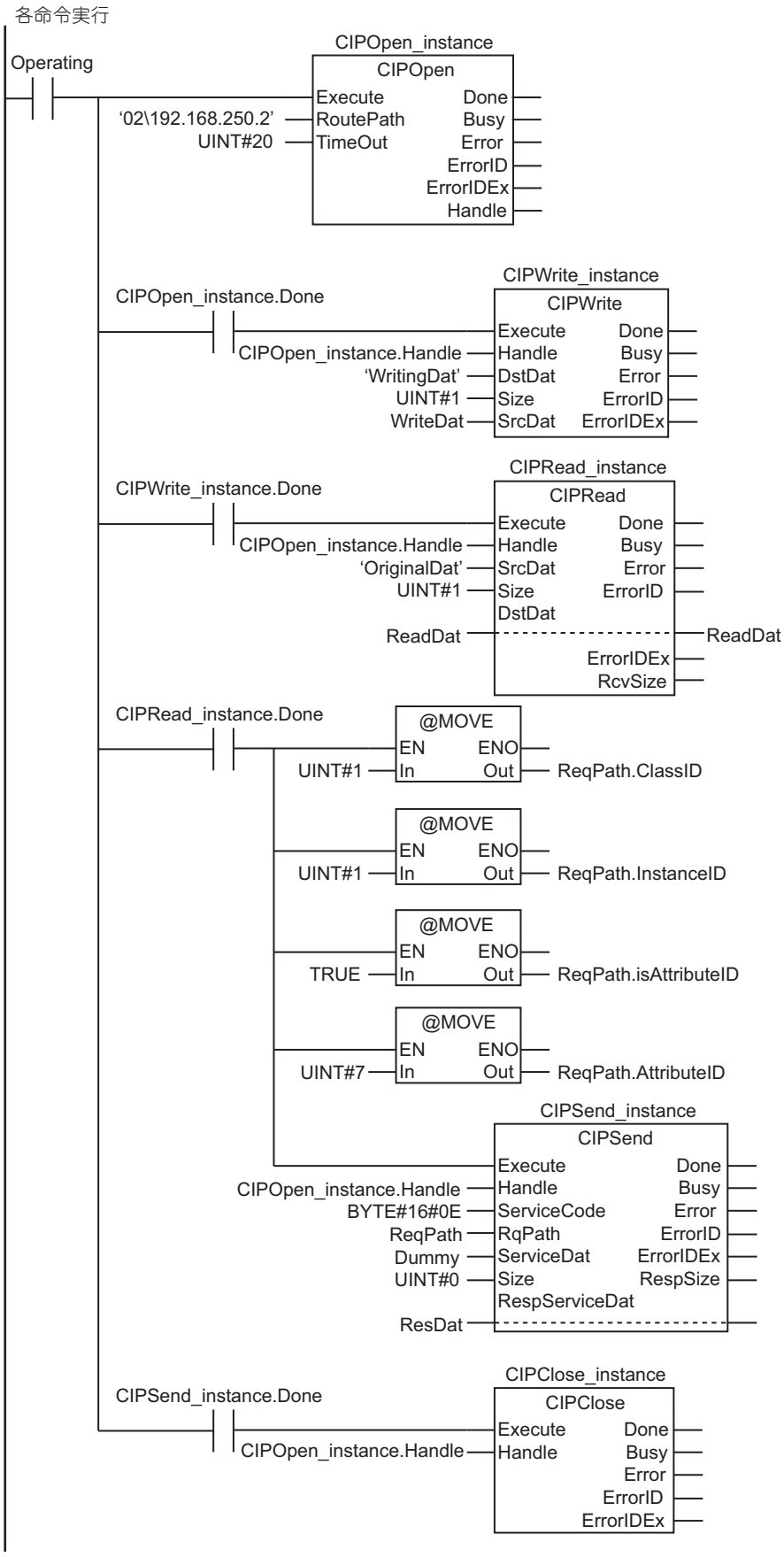
名称	データ型	初期値	コメント
CIPSend_instance	CIPSend		
CIPClose_instance	CIPClose		

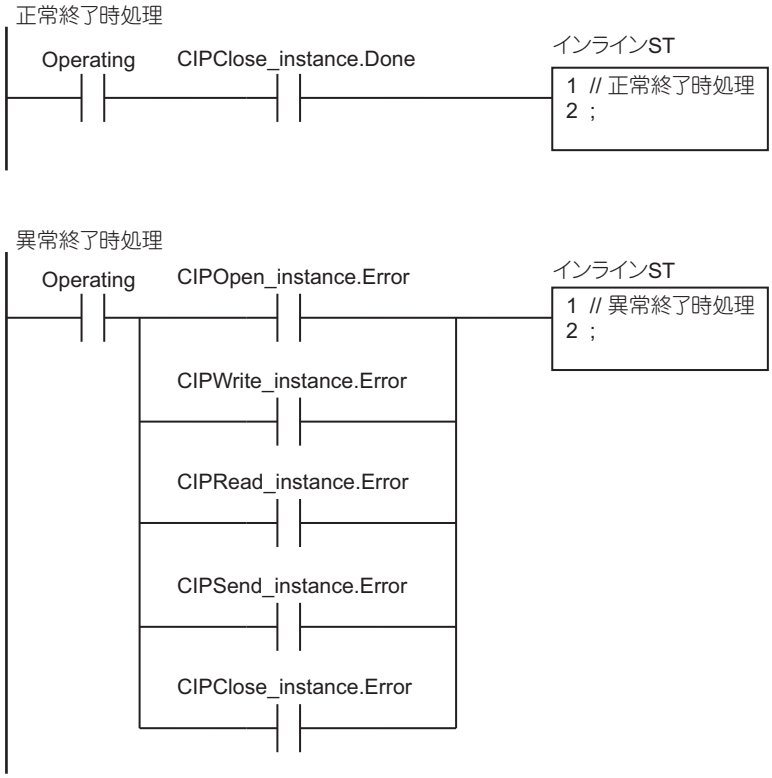
各命令実行完了の判断



トリガ受付




各命令実行
Operating
CIPOpen_instance
CIPOpen
Execute
RoutePath
TimeOut
Done
Busy
Error
ErrorID
ErrorIDEx
Handle
'02\192.168.250.2'
UINT#20
CIPOpen_instance.Done
CIPWrite_instance
CIPWrite
Execute
Handle
DstDat
Size
SrcDat
Done
Busy
Error
ErrorID
ErrorIDEx
CIPOpen_instance.Handle
'WritingDat'
UINT#1
WriteDat
CIPWrite_instance.Done
CIPRead_instance
CIPRead
Execute
Handle
SrcDat
Size
DstDat
Done
Busy
Error
ErrorID
ErrorIDEx
RcvSize
CIPOpen_instance.Handle
'OriginalDat'
UINT#1
ReadDat
ReadDat
CIPRead_instance.Done
@MOVE
EN
In
ENO
Out
UINT#1
ReqPath.ClassID
ReqPath.InstanceID
TRUE
ReqPath.isAttributeID
UINT#7
ReqPath.AttributeID
CIPSend_instance
CIPSend
Execute
Handle
ServiceCode
RqPath
ServiceDat
Size
RespServiceDat
Done
Busy
Error
ErrorID
ErrorIDEx
RespSize
CIPOpen_instance.Handle
BYTE#16#0E
ReqPath
Dummy
UINT#0
ResDat
CIPSend_instance.Done
CIPClose_instance
CIPClose
Execute
Handle
Done
Busy
Error
ErrorID
ErrorIDEx
CIPOpen_instance.Handle



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoCIPTrigger	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	WriteDat	INT	0	書込データ
	ReadDat	INT	0	読出データ
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	リクエストパス
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	レスポンスデータ
	Dummy	BYTE	16#0	ダミー
	CIPOpen_instance	CIPOpen		
	CIPWrite_instance	CIPWrite		
	CIPRead_instance	CIPRead		
	CIPSend_instance	CIPSend		
	CIPClose_instance	CIPClose		

外部変数	名称	コンスタント	データ型	コメント
	_EIP_EtnOnlineSta	☑	BOOL	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoCIPTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoCIPTrigger  :=TRUE;
```

```
    Stage  :=INT#1;
    CIPOpen_instance(Execute:=FALSE);  // インスタンス初期化
    CIPWrite_instance(
        Execute  :=FALSE,  // インスタンス初期化
        SrcDat  :=WriteDat);  // ダミー
    CIPRead_instance(  // インスタンス初期化
        Execute  :=FALSE,  // ダミー
        DstDat  :=ReadDat);  // ダミー
    CIPSend_instance(
        Execute  :=FALSE,  // インスタンス初期化
        ServiceDat  := Dummy,  // ダミー
        RespServiceDat  :=ResDat);  // ダミー
    CIPClose_instance(Execute:=FALSE);  // インスタンス初期化
END_IF;

IF (DoCIPTrigger=TRUE) THEN
    CASE Stage OF
    1 :  // CIP Class3(Large_Forward_Open)コネクション確立
        CIPOpen_instance(
            Execute  :=TRUE,
            TimeOut  :=UINT#20,  // タイムアウト時間 2.0s
            RoutePath  :='02¥192.168.250.2');  // ルートパス

        IF (CIPOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;  // 正常終了
        ELSIF (CIPOpen_instance.Error=TRUE) THEN
            Stage:=INT#10;  // 異常終了
        END_IF;

    2 :  // 変数書込要求
        CIPWrite_instance(
            Execute  :=TRUE,
            Handle  :=CIPOpen_instance.Handle,  // ハンドル
            DstDat  :='WritingDat',  // 他局変数名
            Size  :=UINT#1,  // 書込要素数
            SrcDat  :=WriteDat);  // 書込データ
        IF (CIPWrite_instance.Done=TRUE) THEN
            Stage:=INT#3;  // 正常終了
        ELSIF (CIPWrite_instance.Error=TRUE) THEN
            Stage:=INT#20;  // 異常終了
        END_IF;

    3 :  // 変数読出要求
            CIPRead_instance(
                Execute  :=TRUE,
                Handle  :=CIPOpen_instance.Handle,  // ハンドル
                SrcDat  :='OriginalDat',  // 他局変数名
                Size  :=UINT#1,  // 読出要素数
                DstDat  :=ReadDat);  // 読出データ

            IF (CIPRead_instance.Done=TRUE) THEN
```

```
            Stage:=INT#4;  // 正常終了
        ELSIF (CIPRead_instance.Error=TRUE) THEN
            Stage:=INT#30;  // 異常終了
        END_IF;

    4 :  // 任意メッセージ送信
        ReqPath.ClassID  :=UINT#01;
        ReqPath.InstanceID  :=UINT#01;
        ReqPath.isAttributeID  :=TRUE;
        ReqPath.AttributeID  :=UINT#07;
        CIPSend_instance(
            Execute  :=TRUE,
            Handle  :=CIPOpen_instance.Handle,  // ハンドル
            ServiceCode  :=BYTE#16#0E,  // サービスコード
            RqPath  :=ReqPath,  // リクエストパス
            ServiceDat  :=Dummy,  // サービスデータ
            Size  :=UINT#0,  // 要素数
            RespServiceDat:=ResDat);  // レスポンスデータ

        IF (CIPSend_instance.Done=TRUE) THEN
            Stage:=INT#5;  // 正常終了
        ELSIF (CIPSend_instance.Error=TRUE) THEN
            Stage:=INT#40;  // 異常終了
        END_IF;

    5 :  // CIPClass3 コネクション切断要求
        CIPClose_instance(
            Execute  :=TRUE,
            Handle  :=CIPOpen_instance.Handle);  // ハンドル

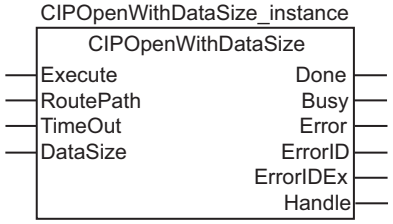
        IF (CIPClose_instance.Done=TRUE) THEN
            Stage:=INT#0;
        ELSIF (CIPClose_instance.Error=TRUE) THEN
            Stage:=INT#50;
        END_IF;

    0:  // 正常終了処理
        DoCIPTrigger  :=FALSE;
        Trigger  :=FALSE;

    ELSE  // 異常終了処理
        DoCIPTrigger  :=FALSE;
        Trigger  :=FALSE;
    END_CASE;
END_IF;
```

CIPOpenWithDataSize

指定した相手ノードに対して、指定したデータ長以下の Class3 Explicit メッセージを送受信可能な CIP の Class3 コネクションを確立します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPOpenWithDataSize	CIP Class3 コネクション確立（サイズ指定あり）	FB		CIPOpen_instance(Execute, RoutePath, TimeOut, DataSize, Done, Busy, Error, ErrorID, ErrorIDEx, Handle);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
RoutePath	ルートパス	入力	ルートパス	データ型に従う	－	－
TimeOut	タイムアウト時間		タイムアウト時間	1 ～ 65535	0.1s	20 (2s)
DataSize	データ長		データ長	6 ～ 8192	バイト	1994
Handle	ハンドル	出力	ハンドル	－	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				○
TimeOut							○													
DataSize							○													
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-1004)」参照																			

機能

ルートパス「RoutePath」で指定した、CIP ネットワーク上の相手ノードに対して、CIP の Class3 コネクションを確立します。データ長「DataSize」で、送受信する Class3Explicit メッセージのデータ長を指定します。
Class3 コネクションのサービスは、「DataSize」の値により、以下のように決まります。

「DataSize」の値[バイト]	サービス
509 以下	Forward_Open
510 ～ 8192	Large_Forward_Open

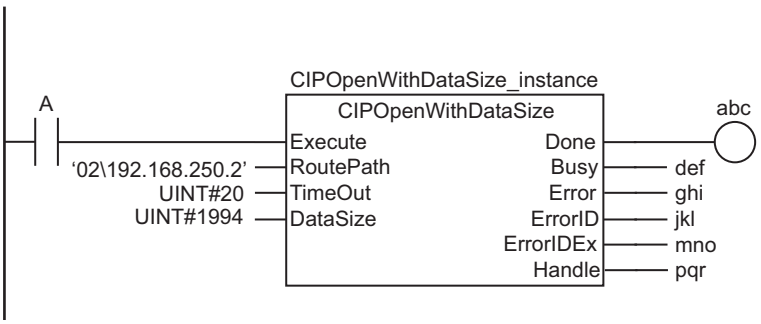
コネクションが確立すると、ハンドル「Handle」が出力されます。
コネクションタイムアウト時間は、「TimeOut」で指定します。CIPSend 命令、CIPWrite 命令、CIPRead 命令のいずれかを実行した後、コネクションタイムアウト時間以内に相手ノードからのレスポンスが返ってこないと、通信失敗と判断されます。
コネクションタイムアウト時間は、CIPRead 命令、CIPWrite 命令、CIPSend 命令が実行して、相手ノードからのレスポンスが返ってくると、リセットされます。

「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	–	–	–
Handle	ハンドル	ハンドル	UDINT	データ型に従う	–	–

「RoutePath」='02\192.168.250.2'、「TimeOut」=UINT#20 の場合の記述例を、以下に示します。IP アドレス'192.168.250.2'の相手ノードに対して、CIP の Class3 コネクションを確立します。データ長は 1994 バイト、タイムアウト時間は 2s です。変数「pqr」にハンドルが代入されます。

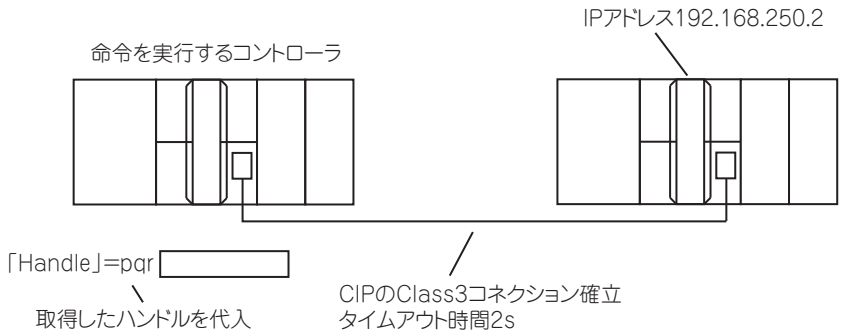
LD



ST

```
CIPOpenWithDataSize_instance(A,
                '2\192.168.250.2',
                UINT#20, UINT#1994,
                abc, def, ghi, jkl, mno,
                pqr);
```

「RoutePath」で指定したCIPネットワーク上の相手ノードに対して、CIPのClass3コネクションを確立します。



「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。「ErrorIDEx」の値と意味は、相手ノードによって規定されます。相手ノードのマニュアルを参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

- CIP 通信の詳細は、以下のマニュアルを参照してください。

a)『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

- Class3 コネクションのサービスが、Large_Forward_Open の場合は、「CIPOpen (P.2-995)」命令を使用することも可能です。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令または CIPOpen 命令を、CIPRead 命令、CIPWrite 命令、CIPSend 命令の前に実行しておく必要があります。
- 本命令でコネクション確立後、最初のタイムアウト時間は、「TimeOut」の値が 100(10s)以下であっても、10s となります。
- 本命令で確立したコネクションは、CIPClose 命令を使用してクローズしてください。
- コネクションがタイムアウトになっても、本命令で生成したハンドルは残ります。必ず、CIPClose 命令を使用して、コネクションをクローズしてください。
- 本命令で生成したハンドルは、プログラムモード変更時に無効になります。
- 同時に生成できるハンドルは、最大 32 個です。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「TimeOut」の値が、有効範囲外の場合。
  b)「RoutePath」の文字列が、不正の場合。
  c) CIP 関連命令が、同時に 32 個を超えて実行された場合。
  d) CIPClass コネクションのリソース（32 個）を超えて、コネクション開設をしようとした場合。
  e) コネクション確立の応答がない場合。
  f)「DataSize」の値が 510～1994 で、かつコネクションを確立する相手ノードが Large_Forward_Open に対応していない場合。
  g) 自 IP アドレスの、設定異常が発生している場合。
  h) IP 重複異常が発生している場合。
  i) TCP コネクションがすべて使用されている場合。
  j) BOOTP サーバ異常が発生している状態で、本命令が実行された場合。

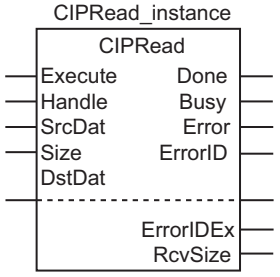


参考

「Error」が TRUE となっても「Handle」の値を変更しません。

CIPRead

CIP ネットワーク上の、他局コントローラの変数の値を読み出します（Class3 Explicit メッセージ）。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPRead	変数読出（Class3 Explicit メッセージ）	FB		CIPRead_instance(Execute, Handle, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Handle	ハンドル	入力	CIPOpen または CIPOpenWithDataSize 命令で取得したハンドル	－	－	－
SrcDat	他局変数名		読み出す他局コントローラの変数名	データ型に従う		"
Size	読出要素数		読み出す要素数	0 ～ 8186		1
DstDat	読出データ	入出力	読み出したデータの値	データ型に従う	－	－
RcvSize	読出データサイズ	出力	読み出したデータのサイズ	0 ～ 8186	バイト	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-1007)」参照																			
SrcDat																				○
Size							○													
DstDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列、構造体全体、構造体の 1 メンバ、共用体の 1 メンバも指定可能 *1																			
RcvSize							○													

*1.　STRING 型の配列は、指定できません。

機能

「Handle」で指定した、CIP ネットワーク上の他局コントローラ内の、他局変数名「SrcDat」で指定したネットワーク変数の値を読み出します。
読み出した値は、読出データ「DstDat」に格納します。

読み出す要素数は、「Size」で指定します。

「SrcDat」が配列の場合は、読み出す要素数を指定します。「SrcDat」が配列ではない場合は、必ず「1」を指定します。

「Size」の値が「0」の場合、「SrcDat」が配列かどうかに関係なく、読出を行いません。
読出完了後、読出データサイズ「RcvSize」に、読み出したデータのサイズが、バイト単位で代入されます。
読み出せるデータの最大サイズは、コネクション確立を行った命令、および読み出すデータ型によって以下のように変わります。

コネクション確立を行った命令	読み出すデータ型	読み出せるデータの最大サイズ[バイト]
CIPOpen	構造体	1984
	STRING 型	1986
	上記以外のデータ型	1988
CIPOpenWithDataSize	構造体	CIPOpenWithDataSize 命令の「DataSize」 - 10
	STRING 型	CIPOpenWithDataSize 命令の「DataSize」 - 8
	上記以外のデータ型	CIPOpenWithDataSize 命令の「DataSize」 - 6

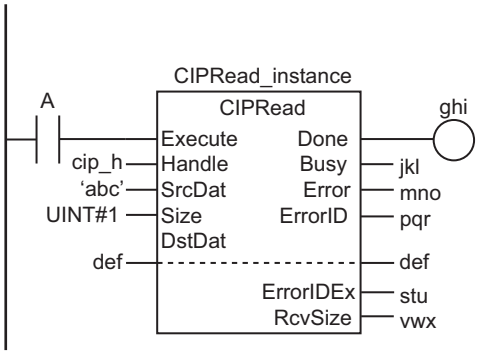
「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	–	–	–
Handle	ハンドル	ハンドル	UDINT	データ型に従う	–	–

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。

他局の変数「abc」の値を読み出し、自局の変数「def」に格納する場合の例を、以下に示します。読出要素数「Size」の値は UINT#1 です。
「abc」、「def」のデータ型は SINT とします。
SINT 型のデータサイズは 1 バイトなので、読出データサイズ「vwx」の値は UINT#1 になります。

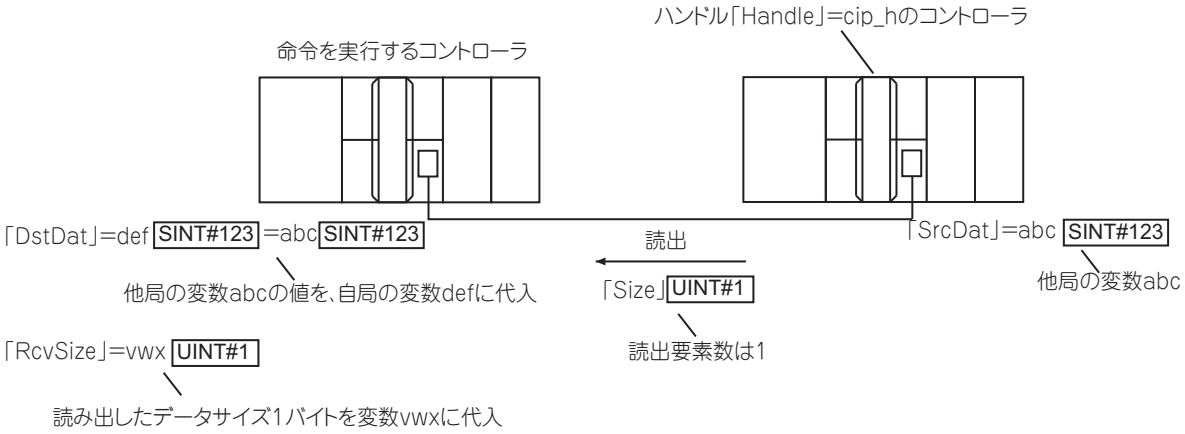
LD



ST

```
CIPRead_instance(A, cip_h, 'abc', UINT#1,
                 def, ghi, jkl, mno, pqr,
                 stu, vwx);
```

ハンドル「Handle」で指定したCIPネットワーク上の、他局の変数「SrcDat」の値を、
自局の変数「DstDat」に代入します。
読出要素数は「Size」で指定します。読み出したデータサイズは、「RcvSize」に代入されます。

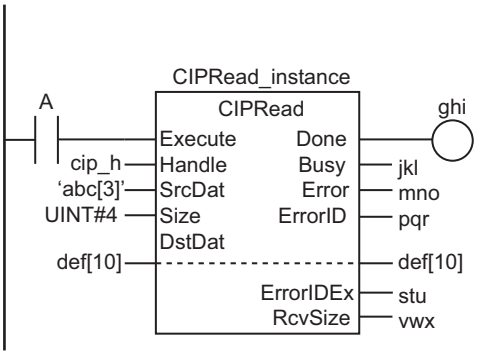


配列を読み出す場合

読み出すデータが配列の場合、「SrcDat」には、必ず添え字付きの配列の要素をパラメータとして渡します。また、「DstDat」にも、添え字付きの配列の要素をパラメータとして渡します。

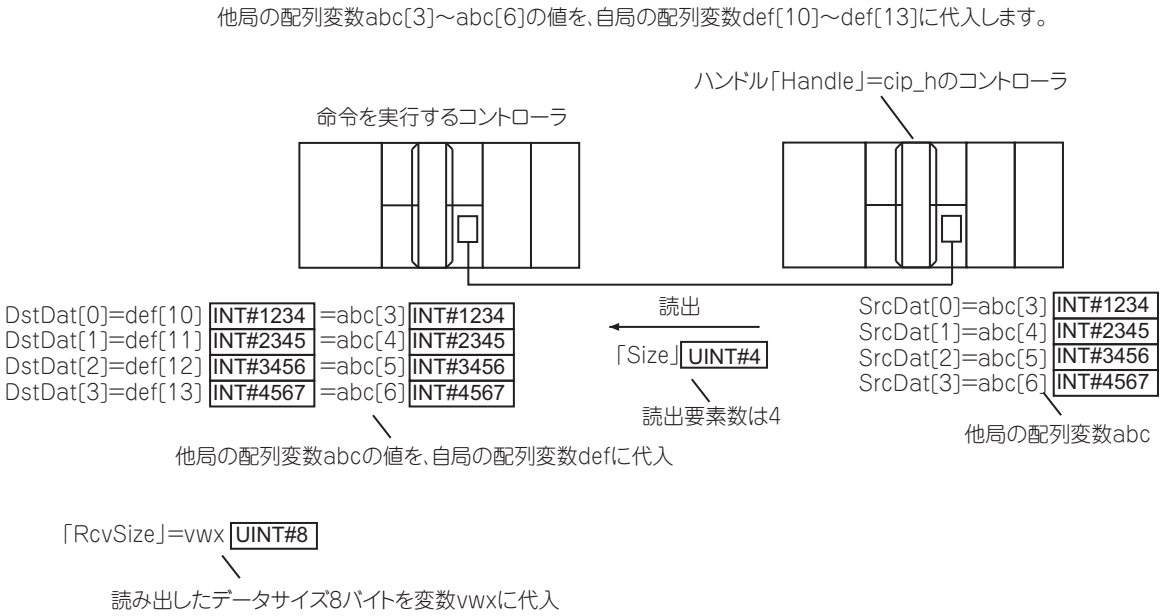
他局の配列変数 abc[3]～abc[6]の 4 つの要素を読み出し、自局の配列変数 def[10]～def[13]に格納する場合の例を、以下に示します。
「abc」、「def」のデータ型は INT とします。
INT 型のデータサイズは 2 バイトなので、読出データサイズ「vwx」の値は UINT#8 になります。

LD



ST

```
CIPRead_instance(A, cip_h, 'abc[3]', UINT#4,
                 def[10], ghi, jkl, mno, pqr,
                 stu, vwx);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。

- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令を実行する前に、必ず CIPOpen 命令または CIPOpenWithDataSize 命令を実行して、「Handle」を取得してください。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 他局コントローラがオムロン製の場合、読み出せる変数は、ネットワーク公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
- 「SrcDat」として使用できる文字の条件は、以下のとおりです。

項目	仕様
最大バイト数	127 バイト

項目	仕様
文字コード	UTF-8
使用可能な文字	英数字（大文字小文字の区別なし）、半角カタカナ、マルチバイト文字、'_'（アンダーバー）
組み合わせで使用できない文字	• ASCII コードの、数字 0～9（文字コード 16#30～16#39）で始まる文字列。 • ASCII コードの、'_'（アンダーバー）が 1 文字だけの文字列。 • 2 個以上連続した、ASCII コードの'_'（アンダーバー）を含む文字列。 • 最初の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最後の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最初の 2 文字が、'P_'である文字列。

- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「Size」の値が、有効範囲外の場合。
  b)「SrcDat」の文字列が不正の場合。
  c) 読み出した値のデータ型と、「DstDat」のデータ型が一致しない場合。
  d) 読み出した値のデータサイズが、「DstDat」の領域を超える場合。
  e)「DstDat」に、サポートしていないデータ型を指定した場合。
  f) CIP で規定されるエラーレスポンスが返った場合。
  g)「Handle.Handle」の値が、有効範囲外の場合。
  h) CIP 関連命令が、同時に 32 個を超えて実行された場合。
  i) CIPOpen 命令または CIPOpenWithDataSize 命令で生成したコネクションが、タイムアウトしている場合。
  j)「SrcDat」のサイズが、コネクション確立を行った命令、および読み出すデータ型で決まるデータサイズを超えている場合。
- 拡張エラーコード「ErrorIDEx」は、本命令では CIP メッセージ異常コードの意味を持ちます。その内容は、以下のとおりです。

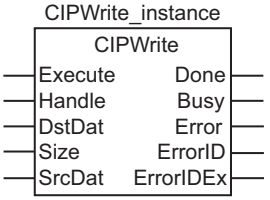
値	異常内容
16#02000000	通信先のノードが高負荷状態で、正常な通信が行えません。
16#04000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 基本データ型 • 列挙型全体 • 構造体全体 • 共用体全体 • 配列全体
16#05000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 列挙型の列挙子 • 構造体の 1 メンバ • 共用体の 1 メンバ • 配列の要素
16#08000000	要求されたサービスが、未サポートです。
16#0C008010	指定された他局変数がダウンロード中です。
16#0C008011	
16#11000000	「Size」の値が、現在、読出可能なデータサイズを超えています。
16#1F000102	読出対象が、読出不可の変数です。
16#1F008007	アクセス不可の変数が指定されました。
16#20008017	指定された他局変数が配列ではないのに、読出要素数が 1 以外になっています。
16#20008018	指定された他局変数が配列で、読出要素数が、その配列の要素数を超えています。
16#26000000	指定された他局変数が、NULL 文字だけです。

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.2-997)」を、参照してください。

CIPWrite

CIP ネットワーク上の、他局コントローラの変数に、値を書き込みます（Class3 Explicit メッセージ）。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPWrite	変数書込（Class3 Explicit メッセージ）	FB		CIPWrite_instance(Execute, Handle, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Handle	ハンドル	入力	CIPOpen 命令または CIPOpenWithDataSize 命令で取得したハンドル	−	−	−
DstDat	他局変数名		書き込む他局コントローラの変数名	データ型に従う		"
Size	書込要素数		書き込む要素数	0 ～ 8178		1
SrcDat	書込データ		書き込むデータの値	データ型に従う		*1

*1.　入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-1012)」参照																			
DstDat																				○
Size							○													
SrcDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列 *1、構造体全体、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			

*1.　STRING 型の配列は、指定できません。

機能

「Handle」で指定した、CIP ネットワーク上の他局コントローラ内の、他局変数名「DstDat」で指定したネットワーク変数に、値を書き込みます。
書き込む値は、書込データ「SrcDat」の中身です。
書き込む要素数は、「Size」で指定します。
「DstDat」が配列の場合は、書き込む要素数を指定します。
「DstDat」が配列ではない場合は、必ず「1」を指定します。
「Size」の値が「0」の場合、「DstDat」が配列かどうかに関係なく、書込を行いません。

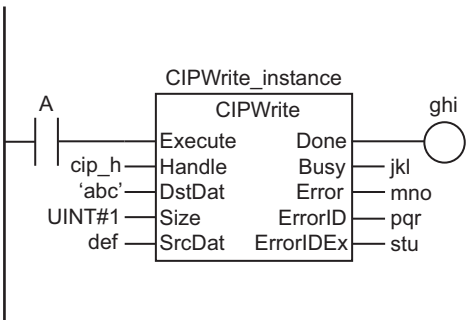
「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	–	–	–
Handle	ハンドル	ハンドル	UDINT	データ型に従う	–	–

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。

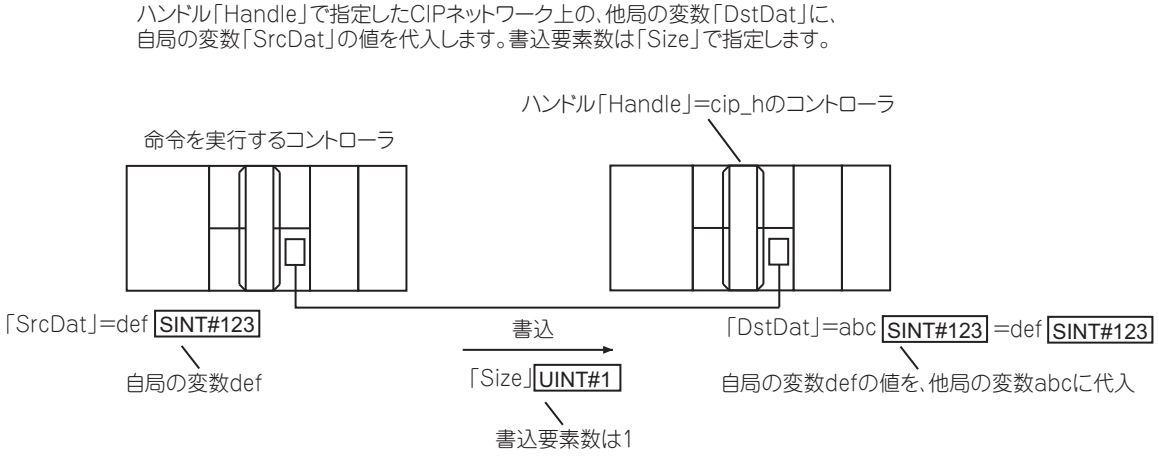
他局の変数「abc」に、自局の変数「def」の値を書き込む場合の例を、以下に示します。書込要素数「Size」の値は UINT#1 です。

LD



ST

```
CIPWrite_instance(A, cip_h, 'abc', UINT#1,
                  def, ghi, jkl, mno, pqr, stu);
```

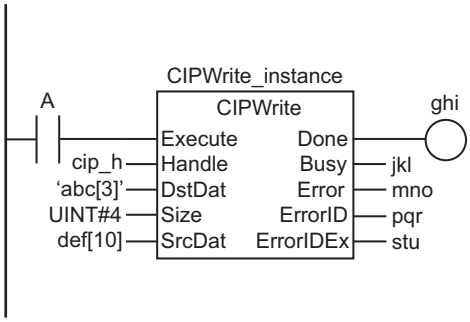


配列に書き込む場合

書き込むデータが配列の場合、「DstDat」には、必ず添え字付きの配列の要素をパラメータとして渡します。また、「SrcDat」にも、添え字付きの配列の要素をパラメータとして渡します。

他局の配列変数 abc[3]～abc[6]の 4 つの要素に、自局の配列変数 def[10]～def[13]の値を書き込む場合の例を、以下に示します。

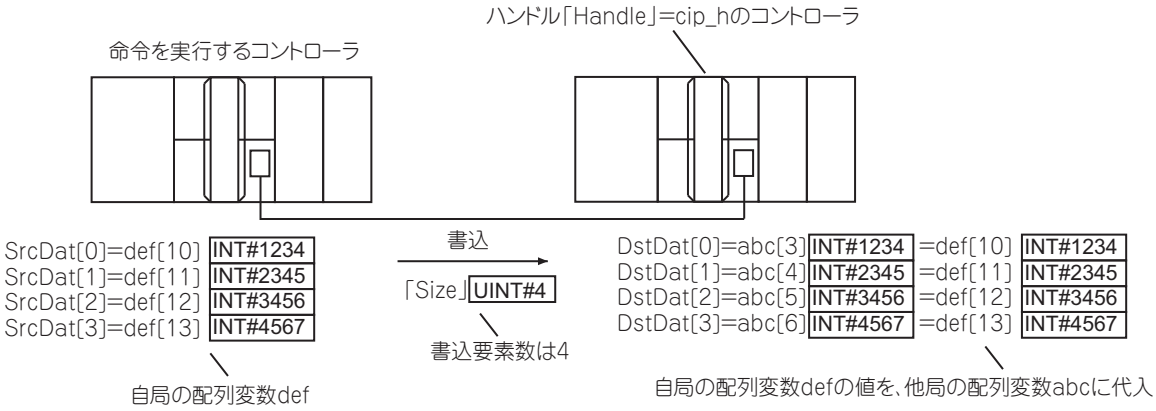
LD



ST

```
CIPWrite_instance(A, cip_h, 'abc[3]', UINT#4, def,
                  ghi[10], jkl, mno, pqr, stu);
```

他局の配列変数abc[3]～abc[6]に、自局の配列変数def[10]～def[13]の値を格納します。



書き込めるデータサイズ

書き込めるデータの最大サイズは、「DstDat」で指定した変数のデータ型や変数名によって、以下のように変わります。

書き込めるデータサイズ[バイト]＝基本サイズ－「DstDat」の変数名サイズ

上式の項目	意味
基本サイズ	CIPOpen 命令でコネクション確立を行った場合 • 「DstDat」で指定した変数のデータ型が、構造体の場合、1984 バイトです。 • 「DstDat」で指定した変数のデータ型が、STRING 型の場合、1986 バイトです。 • 上記以外のデータ型の場合、1988 バイトです。
	CIPOpenWithDataSize 命令でコネクション確立を行った場合 • 「DstDat」で指定した変数のデータ型が、構造体型の場合、以下の式で求められます。 基本サイズ［バイト］ ＝ CIPOpenWithDataSize 命令の「DataSize」 - 10 • 「DstDat」で指定した変数のデータ型が、STRING 型の場合、以下の式で求められます。 基本サイズ［バイト］ ＝ CIPOpenWithDataSize 命令の「DataSize」 - 8 • 上記以外のデータ型の場合、以下の式で求められます。 基本サイズ［バイト］ ＝ CIPOpenWithDataSize 命令の「DataSize」 - 6

上式の項目	意味
「DstDat」の変数名サイズ	• 変数名の、構造体の各階層の ASCII コードバイト数の合計＋（階層数*2）で計算します。 • 各階層の文字列の ASCII コードバイト数が奇数の場合は、それに 1 を加えます。 • ある階層が配列の場合は、さらに（配列の次元数*4）を加えます。 • 構造体の階層の区切りを示す"."、配列の"["、"]"、","は、変数名サイズに含みません。 （例 1）「DstDat」の変数名が、'aaa.bbbbb[1,2,3].cc'の場合 • 第 1 階層“aaa”の文字列は 3 バイトです。奇数なので、1 を加えて 4 バイトとします。 • 第 2 階層“bbbbb[1,2,3]”の、“bbbbb”の文字列は 5 バイトです。奇数なので、1 を加えて 6 バイトとします。 • さらに、第 2 階層“bbbbb[1,2,3]”は 3 次元配列なので、3*4=12 を加えて 18 バイトとします。 • 第 3 階層“cc”の文字列は 2 バイトです。偶数なので、2 バイトのままとします。 • 第 1 階層 4 バイト、第 2 階層 18 バイト、第 3 階層 2 バイトに、階層数 3*2=6 を加えて、変数名サイズは 30 バイトとなります。 （例 2）「DstDat」の変数名が、'val'の場合 • 第 1 階層“val”の文字列は 3 バイトです。奇数なので、1 を加えて 4 バイトとします。 • それに、階層数 1*2=2 を加えて、変数名サイズは 6 バイトとなります。 （例 3）「DstDat」の変数名が、'array[8]'の場合 • 第 1 階層“array”の文字列は 5 バイトです。奇数なので、1 を加えて 6 バイトとします。 • 配列の次元数は 1 です。したがって、1*4=4 を加えます。 • それに、階層数 1*2=2 を加えて、変数名サイズは 12 バイトとなります。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。
• 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

• 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
• 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
• 本命令を実行する前に、必ず CIPOpen 命令または CIPOpenWithDataSize 命令を実行して、「Handle」を取得してください。
• 「SrcDat」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
• 「SrcDat」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 他局コントローラがオムロン製の場合、書き込める変数は、ネットワークに公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
- 「DstDat」として使用できる文字の条件は、以下のとおりです。

項目	仕様
最大バイト数	127 バイト
文字コード	UTF-8
使用可能な文字	英数字（大文字小文字の区別なし）、半角カタカナ、マルチバイト文字、'_'（アンダーバー）
組み合わせで使用できない文字	• ASCII コードの、数字 0～9（文字コード 16#30～16#39）で始まる文字列。 • ASCII コードの、'_'（アンダーバー）が 1 文字だけの文字列。 • 2 個以上連続した、ASCII コードの'_'（アンダーバー）を含む文字列。 • 最初の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最後の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最初の 2 文字が、'P_'である文字列。

- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「Size」の値が、有効範囲外の場合。
  b）「DstDat」の文字列が不正の場合。
  c）「Size」の値が、「SrcDat」の領域を超える場合。
  d）「SrcDat」に、サポートしていないデータ型を指定した場合。
  e）CIP で規定されるエラーレスポンスが返った場合。
  f）「Handle.Handle」の値が、有効範囲外の場合。
  g）CIP 関連命令が、同時に 32 個を超えて実行された場合。
  h）CIPOpen 命令または CIPOpenWithDataSize 命令で生成したコネクションが、タイムアウトしている場合。
  i）「DstDat」のサイズと「SrcDat」の値の合計が、コネクション確立を行った命令、および書き込むデータ型で決まるデータサイズを超えている場合。
- 拡張エラーコード「ErrorIDEx」は、本命令では CIP メッセージ異常コードの意味を持ちます。その内容は、以下のとおりです。

値	異常内容
16#02000000	通信先のノードが高負荷状態で、正常な通信が行えません。
16#04000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 基本データ型 • 列挙型全体 • 構造体全体 • 共用体全体 • 配列全体
16#05000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 列挙型の列挙子 • 構造体の 1 メンバ • 共用体の 1 メンバ • 配列の要素
16#08000000	要求されたサービスが、未サポートです。
16#0C008010	指定された他局変数がダウンロード中です。
16#0C008011	
16#1F000102	• 指定された他局変数にコンスタント属性が設定されているため、書込できません。 • 書込データと、書込要素数が合っていません。
16#1F008007	アクセス不可の変数が指定されました。
16#20008017	指定された他局変数が配列ではないのに、書込要素数が 1 以外になっています。
16#20008018	指定された他局変数が配列で、書込要素数が、その配列の要素数を超えています。
16#20008022	サーバ側の変数のデータ型と、クライアント側の変数のデータ型が一致していません。

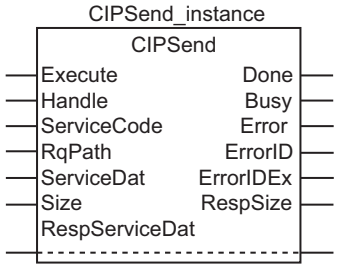
値	異常内容
16#20008028	• 指定された他局変数が列挙型で、書込データが、列挙子にない値です。 • 指定された他局変数が範囲型指定属性を持ち、書込データが、範囲外の値です。
16#26000000	指定された他局変数名が、NULL 文字だけです。

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.2-997)」を、参照してください。

CIPSend

CIP ネットワーク上の指定したデバイスに、Class3 の CIP メッセージを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPSend	任意 Explicit メッセージ送信（Class3）	FB		CIPSend_instance(Execute, Handle, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Handle	ハンドル	入力	CIPOpen 命令または CIPOpenWithDataSize 命令で取得したハンドル	–	–	–
ServiceCode	サービスコード		サービスコード	データ型に従う		
RqPath	リクエストパス		リクエストパス	–		
ServiceDat	サービスデータ		送信するサービスデータ	データ型に従う		*1
Size	送信要素数		送信する要素数			1
RespServiceDat	レスポンスデータ	入出力	レスポンスデータ	データ型に従う	–	–
RespSize	レスポンスデータサイズ	出力	レスポンスデータのサイズ	データ型に従う	バイト	–

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-1019)」参照																			
ServiceCode		○																		
RqPath	構造体_sREQUEST_PATH または_sREQUEST_PATH_EX 詳細は「「RqPath」のデータ型 (P.2-1019)」参照																			
ServiceDat		○	○	○	○	○	○	○	○	○	○	○	○	○	○					
	配列、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			
Size							○													
RespServiceDat		○	○	○	○	○	○	○	○	○	○	○	○	○	○					
	配列、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			
RespSize							○													

機能

サービスコード「ServiceCode」で指定したサービスに対応して、サービスデータ「ServiceDat」を、Class3Explicit メッセージとして、送信します。
送信先は、ハンドル「Handle」で指定します。
リクエストパスは、「RqPath」で指定します。

送信する要素数は、「Size」で指定します。
「ServiceDat」が配列の場合は、送信する要素数を指定します。
「ServiceDat」が配列ではない場合は、必ず「1」を指定します。
サービスデータが不要の場合は、「Size」の値を「0」にします。

送信後、レスポンスデータが、「RespServiceDat」に格納されます。また、レスポンスデータのサイズが、バイト単位で「RespSize」に格納されます。

「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	–	–	–
Handle	ハンドル	ハンドル	UDINT	データ型に従う	–	–

「ClassIDLogicalFormat」、「InstanceIDLogicalFormat」、「AttributeIDLogicalFormat」のデータ型は、列挙型_eCIP_LOGICAL_FORMAT です。
列挙型_eCIP_LOGICAL_FORMAT の、列挙子の意味は以下のとおりです。

列挙子	意味
_8BIT	8 ビット
_16BIT	16 ビット
_32BIT	32 ビット

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。
「ErrorIDEx」の値と意味は、相手ノードによって規定されます。相手ノードのマニュアルを参照してください。

「RqPath」のデータ型

「RqPath」のデータ型は、構造体_sREQUEST_PATH または_sREQUEST_PATH_EX です。
通常は_sREQUEST_PATH を使用してください。
任意の論理フォーマットのサイズを設定して使用する場合は_sREQUEST_PATH_EX を使用してください。

● **_sREQUEST_PATH 型**

変数	名称	内容	データ型	有効範囲	単位	初期値
RqPath	リクエストパス	リクエストパス	_sREQUEST_PATH	–	–	–
ClassID	クラス ID	クラス ID	UINT	データ型に従う	–	0
InstanceID	インスタンス ID	インスタンス ID	UINT			
isAttributeID	アトリビュート有無	TRUE: アトリビュート ID 使用 FALSE: アトリビュート ID 不使用	BOOL			FALSE
AttributeID	アトリビュート ID	アトリビュート ID	UINT			0

（注）_sREQUEST_PATH 型の場合、各 ID の論理フォーマットサイズは 16 ビットです。

● **_sREQUEST_PATH_EX 型**

変数	名称	内容	データ型	有効範囲	単位	初期値
RqPath	リクエストパス	リクエストパス	_sREQUEST_PATH_EX	–	–	–
ClassIDLogicalFormat	クラス ID 論理フォーマット	クラス ID のデータサイズ	_eCIP_LOGICAL_FORMAT	データ型に従う	–	_8BIT
ClassID	クラス ID	クラス ID	UDINT			0
InstanceIDLogicalFormat	インスタンス ID 論理フォーマット	インスタンス ID のデータサイズ	_eCIP_LOGICAL_FORMAT			_8BIT
InstanceID	インスタンス ID	インスタンス ID	UDINT			0
isAttributeID	アトリビュート有無	TRUE: アトリビュート ID 使用 FALSE: アトリビュート ID 不使用	BOOL			FALSE
AttributeIDLogicalFormat	アトリビュート ID 論理フォーマット	アトリビュート ID のデータサイズ	_eCIP_LOGICAL_FORMAT			_8BIT
AttributeID	アトリビュート ID	アトリビュート ID	UDINT			0

配列を送受信する場合

「ServiceDat」または「RespServiceDat」が配列の場合、それらの変数には、必ず添え字付きの配列の要素をパラメータとして渡します。

読み書きできるデータサイズ

読み出せるデータの最大サイズは、コネクション確立を CIPOpen 命令で行ったか、CIPOpenWithDataSize 命令で行ったかによって、以下のように変わります。

コネクション確立を行った命令	読み出せるデータの最大サイズ
CIPOpen	1990 バイト

コネクション確立を行った命令	読み出せるデータの最大サイズ
CIPOpenWithDataSize	8188 バイトを上限として、サーバが返すレスポンスを受信可能

書き込めるデータの最大サイズは、リクエストパスのアトリビュート有無と、コネクション確立を行った命令によって、以下のように変わります。

書き込めるデータサイズ[バイト]＝基本サイズ－アトリビュート有無

上式の項目	意味
基本サイズ	• CIPOpen 命令でコネクション確立を行った場合、1992 バイト • CIPOpenWithDataSize 命令でコネクション確立を行った場合、CIPOpenWithDataSize 命令の「DataSize」-2
アトリビュート有無	• アトリビュート ID 使用の場合、14 バイト • アトリビュート ID 不使用の場合、10 バイト

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。
• 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

• 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
• 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
• 本命令を実行する前に、必ず CIPOpen 命令または CIPOpenWithDataSize 命令を実行して、「Handle」を取得してください。
• 「ServiceDat」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
• 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
• 他局コントローラがオムロン製の場合、書き込める変数は、ネットワークに公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
• つぎの場合、異常となります。「Error」は TRUE となります。
a)「RqPath.ClassIDLogicalFormat」、「RqPath.AttributeIDLogicalFormat」の値が有効範囲外の場合。
b)「RqPath.ClassIDLogicalFormat」で指定したサイズと「RqPath.ClassID」のデータサイズ、または「RqPath.InstanceIDLogicalFormat」で指定したサイズと「RqPath.InstanceID」のデータサイズ、または「RqPath.AttributeIDLogicalFormat」で指定したサイズと「RqPath.AttributeID」のデータサイズが、不一致の場合。

c)「Size」の値が、書込データ領域を超える場合。
d)「Size」の値が、「ServiceDat」の領域を超える場合。
e)「RespSize」の値が、「RespServiceDat」の領域を超える場合。
f)「ServiceDat」に、サポートしていないデータ型を指定した場合。
g)「RespServiceDat」に、サポートしていないデータ型を指定した場合。
h)「RqPath」に、_sREQUEST_PATH または_sREQUEST_PATH_EX 以外のデータ型の変数を指定した場合。
i) CIP で規定されるエラーレスポンスが返った場合。
j)「Handle.Handle」の値が、有効範囲外の場合。
k) CIP 関連命令が、同時に 32 個を超えて実行された場合。
l) CIPOpen 命令または CIPOpenWithDataSize 命令で生成したコネクションが、タイムアウトしている場合。
m)「RqPath」と「ServiceDat」のサイズの合計が、コネクション確立を行った命令で決まるデータサイズを超えている場合。

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.2-997)」を、参照してください。

CIPClose

指定したハンドルの、CIP Class3 のコネクションを切断します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPClose	CIP Class3 コネクションの切断	FB	CIPClose_instance CIPClose Execute / Handle Done / Busy / Error / ErrorID / ErrorIDEx	CIPClose_instance(Execute, Handle, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Handle	ハンドル	入力	CIPOpen 命令または CIPOpenWithDataSize 命令で取得したハンドル	–	–	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	構造体_sCIP_HANDLE　詳細は「機能 (P.2-1023)」参照																			

機能

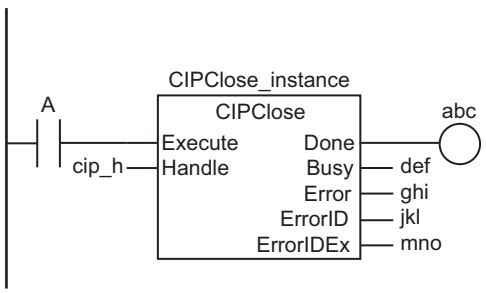
ハンドル「Handle」で指定した、CIP Class3 コネクションを切断します。

「Handle」のデータ型は、構造体_sCIP_HANDLE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Handle	ハンドル	ハンドル	_sCIP_HANDLE	–	–	–
Handle	ハンドル	ハンドル	UDINT	データ型に従う	–	–

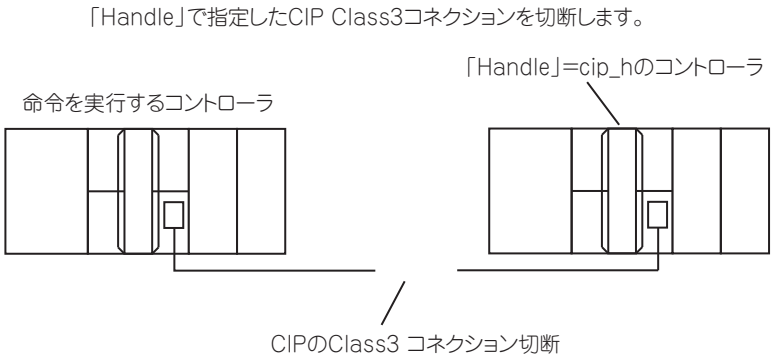
記述例を、以下に示します。「Handle」=cip_h で指定した CIP Class3 コネクションを切断します。

LD



ST

```
CIPClose_instance(A, cip_h, abc, def,
                  ghi, jkl, mno);
```



関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPln1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。
- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

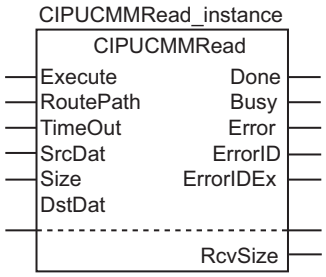
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「Handle」は、CIPOpen 命令または CIPOpenWithDataSize 命令で取得したものを指定してください。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令では、「ErrorIDEx」は使用しません。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「Handle.Handle」の値が、有効範囲外の場合。
  b) CIP 関連命令が、同時に 32 個を超えて実行された場合。

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.2-997)」を、参照してください。

CIPUCMMRead

指定した CIP ネットワーク上の、他局コントローラの変数の値を読み出します（UCMM Explicit メッセージ）。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	FB		CIPUCMMRead_instance(Execute, RoutePath, TimeOut, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
RoutePath	ルートパス	入力	ルートパス	データ型に従う	−	−
TimeOut	タイムアウト時間		タイムアウト時間	1 ～ 65535	0.1s	20（2s）
SrcDat	他局変数名		読み出す他局コントローラの変数名	データ型に従う	−	"
Size	読出要素数		読み出す要素数	0 ～ 496		1
DstDat	読出データ	入出力	読み出したデータの値	データ型に従う	−	−
RcvSize	読出データサイズ	出力	読み出したデータのサイズ	0 ～ 496	バイト	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				○
TimeOut							○													
SrcDat																				○
Size							○													
DstDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列、構造体全体、構造体の 1 メンバ、共用体の 1 メンバも指定可能 *1																			
RcvSize							○													

*1. STRING 型の配列は、指定できません。

機能

ルートパス「RoutePath」で指定した、CIP ネットワーク上の他局コントローラ内の、他局変数名「SrcDat」で指定したネットワーク変数の値を読み出します。
読み出した値は、読出データ「DstDat」に格納します。

読み出す要素数は、「Size」で指定します。

「SrcDat」が配列の場合は、読み出す要素数を指定します。
「SrcDat」が配列ではない場合は、必ず「1」を指定します。
「Size」の値が「0」の場合、「SrcDat」が配列かどうかに関係なく、読出を行いません。

読出完了後、読出データサイズ「RcvSize」に、読み出したデータのサイズが、バイト単位で代入されます。読み出せるデータの最大サイズは、変数のデータ型によって以下のように変わります。

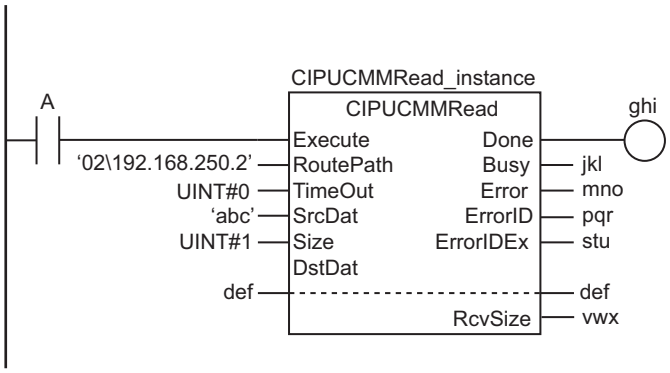
- 構造体の場合、492 バイトです。
- STRING 型の場合、494 バイトです。
- 上記以外のデータ型の場合、496 バイトです。

タイムアウト時間は、「TimeOut」で指定します。タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。

他局の変数「abc」の値を読み出し、自局の変数「def」に格納する場合の例を、以下に示します。
読出要素数「Size」の値は UINT#1 です。
「abc」、「def」のデータ型は SINT とします。
SINT 型のデータサイズは 1 バイトなので、読出データサイズ「vwx」の値は UINT#1 になります。

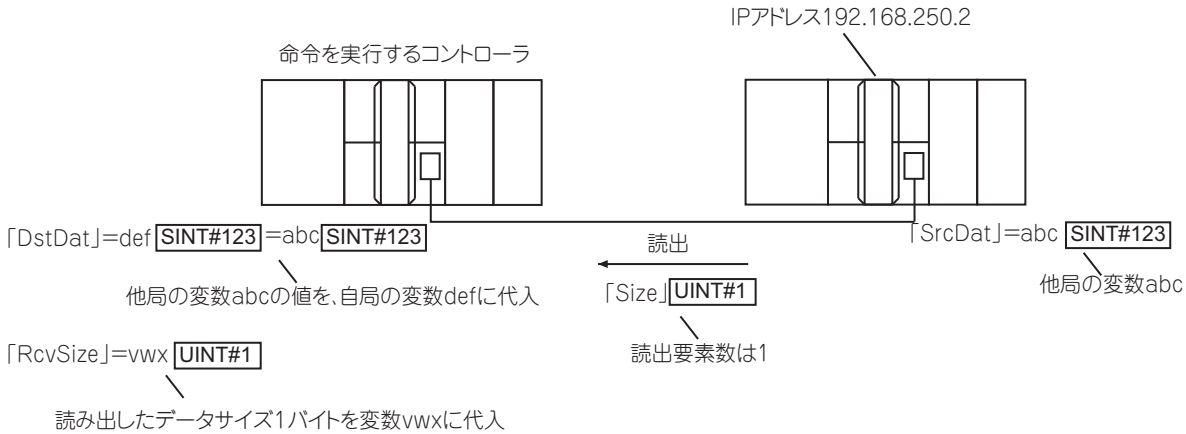
LD



ST

```
CIPUCMMRead_instance(A, '2\192.168.250.2',
                     UINT#0, 'abc', UINT#1,
                     def, ghi, jkl, mno, pqr,
                     stu, vwx);
```

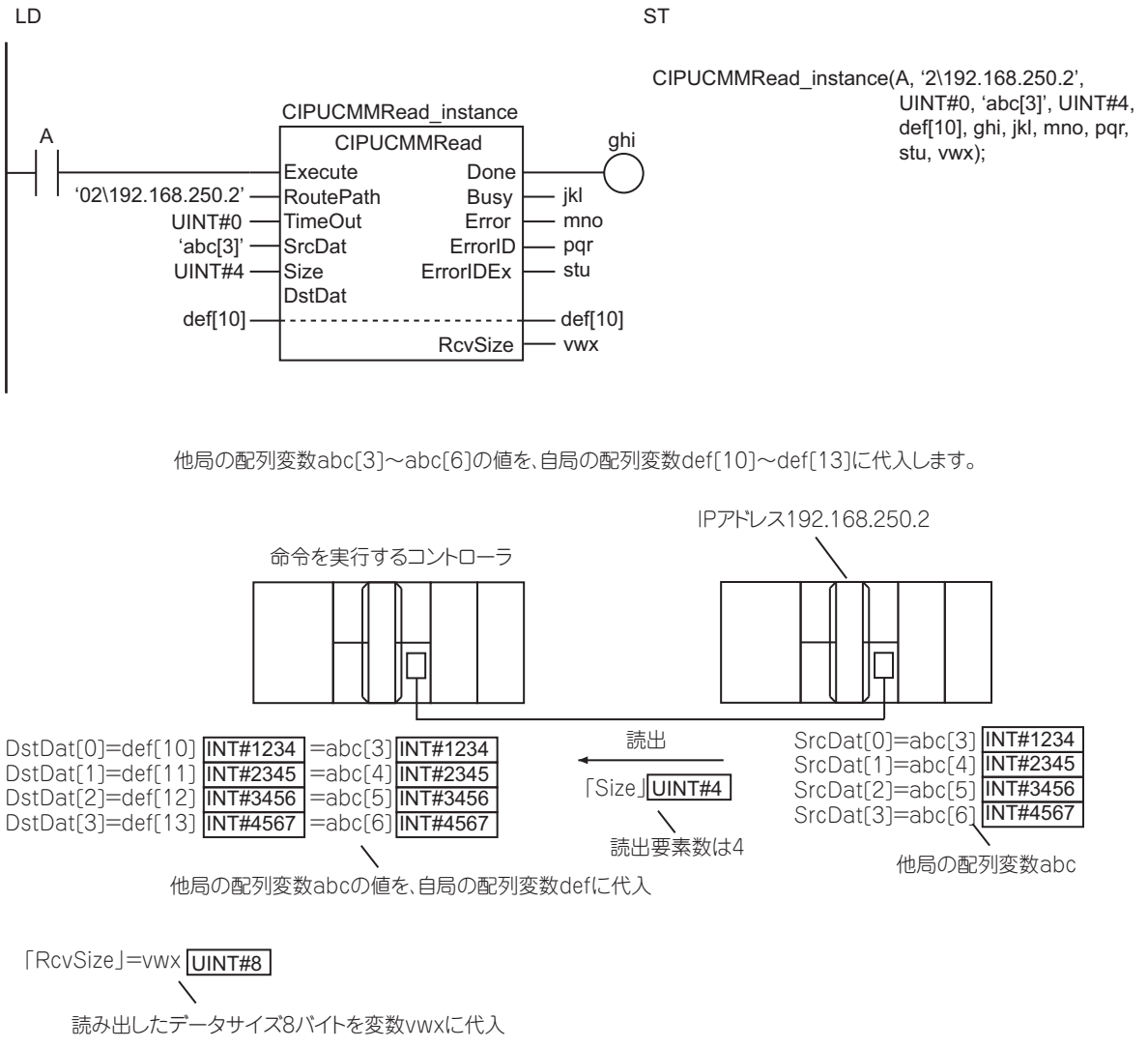
ルートパス「RoutePath」で指定したCIPネットワーク上の他局の変数「SrcDat」の値を、
自局の変数「DstDat」に代入します。
読出要素数は「Size」で指定します。読み出したデータサイズは、「RcvSize」に代入されます。



配列を読み出す場合

読み出すデータが配列の場合、「SrcDat」には、必ず添え字付きの配列の要素をパラメータとして渡します。また、「DstDat」にも、添え字付きの配列の要素をパラメータとして渡します。

他局の配列変数 abc[3]～abc[6]の 4 つの要素を読み出し、自局の配列変数 def[10]～def[13]に格納する場合の例を、以下に示します。
「abc」、「def」のデータ型は INT とします。
INT 型のデータサイズは 2 バイトなので、読出データサイズ「vwx」の値は UINT#8 になります。



関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPln1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。
- 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 他局コントローラがオムロン製の場合、読み出せる変数は、ネットワーク公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
- 「SrcDat」として使用できる文字の条件は、以下のとおりです。

項目	仕様
最大バイト数	127 バイト
文字コード	UTF-8
使用可能な文字	英数字（大文字小文字の区別なし）、半角カタカナ、マルチバイト文字、'_'（アンダーバー）
組み合わせで使用できない文字	• ASCII コードの、数字 0～9（文字コード 16#30～16#39）で始まる文字列。 • ASCII コードの、'_'（アンダーバー）が 1 文字だけの文字列。 • 2 個以上連続した、ASCII コードの'_'（アンダーバー）を含む文字列。 • 最初の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最後の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最初の 2 文字が、'P_'である文字列。

- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「TimeOut」の値が、有効範囲外の場合。
  b）「Size」の値が、有効範囲外の場合。
  c）「SrcDat」の文字列が不正の場合。
  d）読み出した値のデータ型と、「DstDat」のデータ型が一致しない場合。
  e）読み出した値のデータサイズが、「DstDat」の領域を超える場合。
  f）「DstDat」に、サポートしていないデータ型を指定した場合。
  g）CIP で規定されるエラーレスポンスが返った場合。
  h）「RoutePath」の文字列が、不正の場合。
  i） CIP 関連命令が、同時に 32 個を超えて実行された場合。
  j） タイムアウト時間を超えても応答がない場合。
  k）自 IP アドレスの、設定異常が発生している場合。
  l） BOOTP サーバ異常が発生している状態で、本命令が実行された場合。
  m）IP 重複異常が発生している場合。
- 拡張エラーコード「ErrorIDEx」は、本命令では CIP メッセージ異常コードの意味を持ちます。その内容は、以下のとおりです。

値	異常内容
16#02000000	通信先のノードが高負荷状態で、正常な通信が行えません。

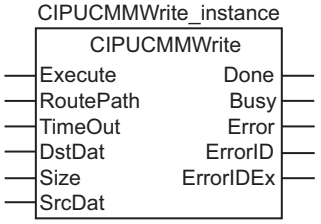
値	異常内容
16#04000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 基本データ型 • 列挙型全体 • 構造体全体 • 共用体全体 • 配列全体
16#05000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 列挙型の列挙子 • 構造体の 1 メンバ • 共用体の 1 メンバ • 配列の要素
16#08000000	要求されたサービスが、未サポートです。
16#0C008010	指定された他局変数がダウンロード中です。
16#0C008011	
16#11000000	「Size」の値が、現在、読出可能なデータサイズを超えています。
16#1F000102	読出対象が、読出不可の変数 I/O です。
16#1F008007	アクセス不可の変数が指定されました。
16#20008017	指定された他局変数が配列ではないのに、読出要素数が 1 以外になっています。
16#20008018	指定された他局変数が配列で、読出要素数が、その配列の要素数を超えています。
16#26000000	指定された他局変数が、NULL 文字だけです。

サンプルプログラム

CIPUCMMSend 命令の「サンプルプログラム (P.2-1040)」を、参照してください。

CIPUCMMWrite

CIP ネットワーク上の、他局コントローラの変数に、値を書き込みます（UCMM Explicit メッセージ）。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	FB		CIPUCMMWrite_instance(Execute, RoutePath, TimeOut, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
RoutePath	ルートパス	入力	ルートパス	データ型に従う	−	−
TimeOut	タイムアウト時間		タイムアウト時間	1 ～ 65535	0.1s	20 (2s)
DstDat	他局変数名		書き込む他局コントローラの変数名	データ型に従う	−	"
Size	書込要素数		書き込む要素数	0 ～ 488		1
SrcDat	書込データ		書き込むデータの値	データ型に従う		*1

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				○
TimeOut							○													
DstDat																				○
Size							○													
SrcDat	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列 *1、構造体全体、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			

*1. STRING 型の配列は、指定できません。

機能

ルートパス「RoutePath」で指定した、CIP ネットワーク上の他局コントローラ内の、他局変数名「DstDat」で指定したネットワーク変数に、値を書き込みます。
書き込む値は、書込データ「SrcDat」の中身です。

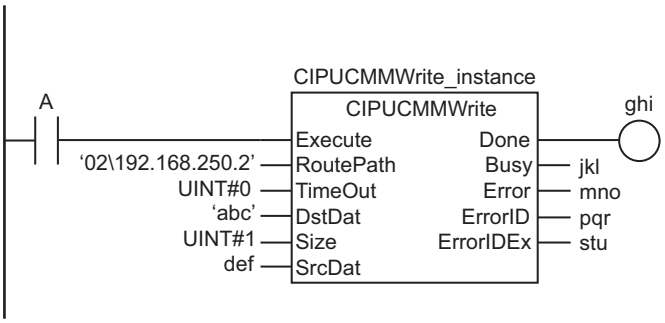
書き込む要素数は、「Size」で指定します。
「DstDat」が配列の場合は、書き込む要素数を指定します。
「DstDat」が配列ではない場合は、必ず「1」を指定します。
「Size」の値が「0」の場合、「DstDat」が配列かどうかに関係なく、書込を行いません。

タイムアウト時間は、「TimeOut」で指定します。タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。

他局の変数「abc」に、自局の変数「def」の値を書き込む場合の例を、以下に示します。書込要素数「Size」の値は UINT#1 です。

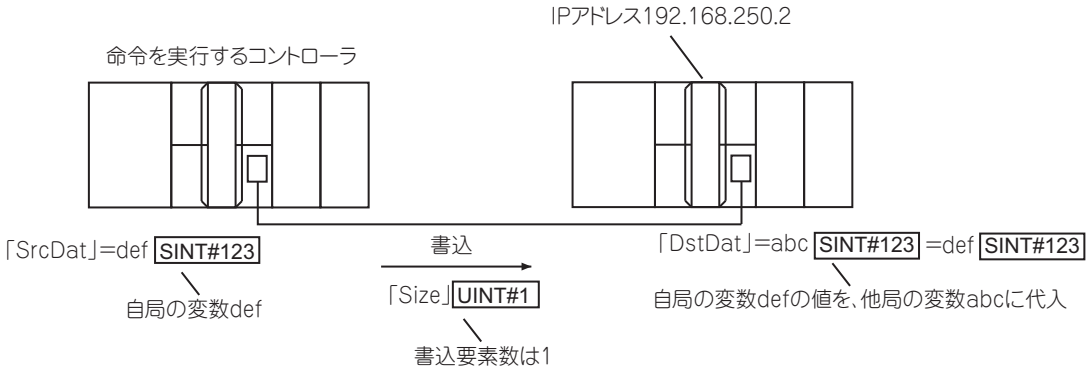
LD



ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc', UINT#1,
                      def, ghi, jkl, mno, pqr,
                      stu);
```

ルートパス「RoutePath」で指定したCIPネットワーク上の他局の変数「DstDat」に、
自局の変数「SrcDat」の値を代入します。書込要素数は「Size」で指定します。

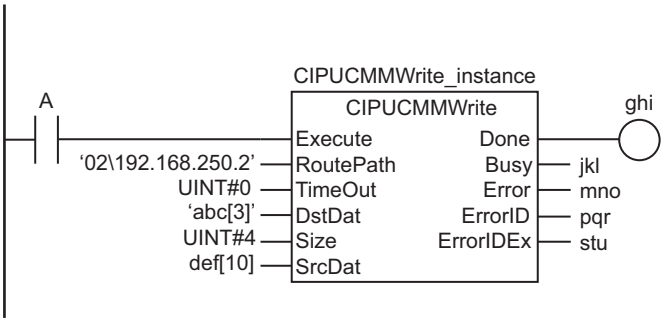


配列に書き込む場合

書き込むデータが配列の場合、「DstDat」には、必ず添え字付きの配列の要素をパラメータとして渡します。また、「SrcDat」にも、添え字付きの配列の要素をパラメータとして渡します。

他局の配列変数 abc[3]～abc[6]の 4 つの要素に、自局の配列変数 def[10]～def[13]の値を書き込む場合の例を、以下に示します。

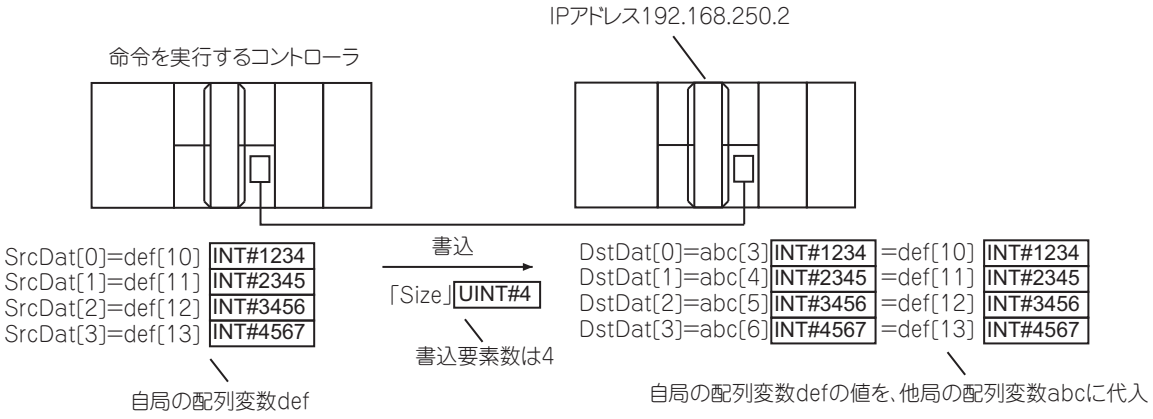
LD



ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc[3]', UINT#4,
                      def[10], ghi, jkl, mno, pqr,
                      stu);
```

他局の配列変数abc[3]～abc[6]に、自局の配列変数def[10]～def[13]の値を格納します。



書き込めるデータサイズ

書き込めるデータの最大サイズは、「DstDat」で指定した変数のデータ型や変数名、およびルートパスによって、以下のように変わります。

書き込めるデータサイズ[バイト]＝基本サイズー「DstDat」の変数名サイズー経路情報サイズ

上式の項目	意味
基本サイズ	• 「DstDat」で指定した変数のデータ型が、構造体の場合、492 バイトです。 • 「DstDat」で指定した変数のデータ型が、STRING 型の場合、494 バイトです。 • 上記以外のデータ型の場合、496 バイトです。
「DstDat」の変数名サイズ	• 変数名の、構造体の各階層の ASCII コードバイト数の合計＋（階層数*2）で計算します。 • 各階層の文字列の ASCII コードバイト数が奇数の場合は、それに 1 を加えます。 • ある階層が配列の場合は、さらに（配列の次元数*4）を加えます。 • 構造体の階層の区切りを示す"."、配列の"["、"]"、","は、変数名サイズに含みません。 （例 1）「DstDat」の変数名が、'aaa.bbbbb[1,2,3].cc'の場合 • 第 1 階層“aaa”の文字列は 3 バイトです。奇数なので、1 を加えて 4 バイトとします。 • 第 2 階層“bbbbb[1,2,3]”の、“bbbbb”の文字列は 5 バイトです。奇数なので、1 を加えて 6 バイトとします。 • さらに、第 2 階層“bbbbb[1,2,3]”は 3 次元配列なので、3*4=12 を加えて 18 バイトとします。 • 第 3 階層“cc”の文字列は 2 バイトです。偶数なので、2 バイトのままとします。 • 第 1 階層 4 バイト、第 2 階層 18 バイト、第 3 階層 2 バイトに、階層数 3*2=6 を加えて、変数名サイズは 30 バイトとなります。 （例 2）「DstDat」の変数名が、'val'の場合 • 第 1 階層“val”の文字列は 3 バイトです。奇数なので、1 を加えて 4 バイトとします。 • それに、階層数 1*2=2 を加えて、変数名サイズは 6 バイトとなります。 （例 3）「DstDat」の変数名が、'array[8]'の場合 • 第 1 階層“array”の文字列は 5 バイトです。奇数なので、1 を加えて 6 バイトとします。 • 配列の次元数は 1 です。したがって、1*4=4 を加えます。 • それに、階層数 1*2=2 を加えて、変数名サイズは 12 バイトとなります。

上式の項目	意味
経路情報サイズ	• ホップがない場合、経路情報サイズは 0 バイトです。*1 • ホップがある場合、経路情報サイズは、ルートパスサイズ+12 バイトとなります。 • ルートパスサイズとは、ルートパスの文字列の ASCII コードのバイト数のことです。 • ただし、以下のことに注意してください。 a）アドレス部の先頭に“#”が付く場合、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 b）アドレス部の先頭に“#”が付かない場合、ネットワーク部は 2 バイトとして計算します。 c）アドレス部の先頭に“#”が付かず、かつアドレス部の ASCII コードのバイト数が奇数の場合、その値に 1 バイトを加算します。 d）ルートパスの階層の区切りを示す“\”は、ルートパスサイズに含みません。 e）最初のホップは、ルートパスサイズに含みません。 （例 1）ルートパスが、'01\#11\02\192.168.250.2\01\#01'の場合 • 最初のホップはルートパスサイズに含まないので、先頭の“01\#11”は無視します。 • そのつぎの、ネットワーク種別“02”は、2 バイトです。 • そのつぎの、アドレス部“192.168.250.2”は、13 バイトです。奇数なので、1 を加えて 14 バイトとします。 • そのつぎの、“01\#01”は、アドレス部に“#”が付いているので、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 • 上記をすべて加えて、ルートパスサイズは、18 バイトとなります。 • ルートパスサイズに 12 バイトを加えて、経路情報サイズは 30 バイトとなります。 （例 2）ルートパスが、'02\192.168.250.2\01\#00'の場合 • 最初のホップはルートパスサイズに含まないので、先頭の“02\192.168.250.2”は無視します。 • そのつぎの、“01\#00”は、アドレス部に“#”が付いているので、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 • したがって、ルートパスサイズは 2 バイトとなります。 • ルートパスサイズに 12 バイトを加えて、経路情報サイズは 14 バイトとなります。 （例 3）ルートパスが、'02\192.168.250.2'の場合 • ホップがないので、経路情報サイズは 0 バイトです。

*1. ホップとは、送信ノードと受信ノードの間で、ルーチングを行うことです。たとえば、ルートパスが、'02\192.168.250.2\01\#00'の場合、いったん IP アドレス 192.168.250.2 のノードにルーチングして、00 号機にメッセージを送るので、ホップがひとつあることになります。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。

• 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「SrcDat」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「SrcDat」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 他局コントローラがオムロン製の場合、書き込める変数は、ネットワークに公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
- 「DstDat」として使用できる文字の条件は、以下のとおりです。

項目	仕様
最大バイト数	127 バイト
文字コード	UTF-8
使用可能な文字	英数字（大文字小文字の区別なし）、半角カタカナ、マルチバイト文字、'_'（アンダーバー）
組み合わせで使用できない文字	• ASCII コードの、数字 0～9（文字コード 16#30～16#39）で始まる文字列。 • ASCII コードの、'_'（アンダーバー）が 1 文字だけの文字列。 • 2 個以上連続した、ASCII コードの'_'（アンダーバー）を含む文字列。 • 最初の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最後の 1 文字が、ASCII コードの'_'（アンダーバー）である文字列。 • 最初の 2 文字が、'P_'である文字列。

- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「TimeOut」の値が、有効範囲外の場合。
  b）「Size」の値が、有効範囲外の場合。
  c）「DstDat」の文字列が不正の場合。
  d）「Size」の値が、「SrcDat」の領域を超える場合。
  e）「SrcDat」に、サポートしていないデータ型を指定した場合。
  f） CIP で規定されるエラーレスポンスが返った場合。
  g）「RoutePath」の文字列が、不正の場合。
  h） CIP 関連命令が、同時に 32 個を超えて実行された場合。
  i） タイムアウト時間を超えても応答がない場合。
  j） 自 IP アドレスの、設定異常が発生している場合。
  k） IP 重複異常が発生している場合。
- 拡張エラーコード「ErrorIDEx」は、本命令では CIP メッセージ異常コードの意味を持ちます。その内容は、以下のとおりです。

値	異常内容
16#02000000	通信先のノードが高負荷状態で、正常な通信が行えません。
16#04000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 基本データ型 • 列挙型全体 • 構造体全体 • 共用体全体 • 配列全体

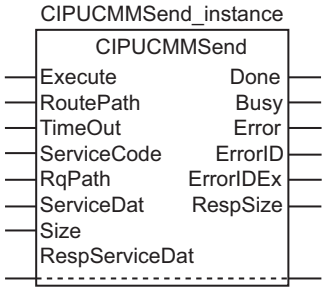
値	異常内容
16#05000000	指定された他局変数が以下のデータ型であり、かつ他局に存在しません。 • 列挙型の列挙子 • 構造体の 1 メンバ • 共用体の 1 メンバ • 配列の要素
16#08000000	要求されたサービスが、未サポートです。
16#0C008010	指定された他局変数がダウンロード中です。
16#0C008011	
16#1F000102	• 指定された他局変数にコンスタント属性が設定されているため、書込できません。 • 書込データと、書込要素数が合っていません。
16#1F008007	アクセス不可の変数が指定されました。
16#20008017	指定された他局変数が配列ではないのに、書込要素数が 1 以外になっています。
16#20008018	指定された他局変数が配列で、書込要素数が、その配列の要素数を超えています。
16#20008028	• 指定された他局変数が列挙型で、書込データが、列挙子にない値です。 •
16#26000000	指定された他局変数名が、NULL 文字だけです。

サンプルプログラム

CIPUCMMSend 命令の「サンプルプログラム (P.2-1040)」を、参照してください。

CIPUCMMSend

CIP ネットワーク上の指定したデバイスに、UCMM の CIP メッセージを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPUCMMSend	任意 Explicit メッセージ送信 (UCMM)	FB		CIPUCMMSend_instance(Execute, RoutePath, TimeOut, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
RoutePath	ルートパス	入力	ルートパス	データ型に従う	−	−
TimeOut	タイムアウト時間		タイムアウト時間	1 ～ 65535	0.1s	20 (2.0s)
ServiceCode	サービスコード		サービスコード	データ型に従う	−	−
RqPath	リクエストパス		リクエストパス	−		
ServiceDat	サービスデータ		送信するデータ	データ型に従う		*1
Size	送信要素数		送信する要素数			1
RespServiceDat	レスポンスデータ	入出力	レスポンスデータ	データ型に従う	−	−
RespSize	レスポンスデータサイズ	出力	レスポンスデータのサイズ	データ型に従う	バイト	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				○
TimeOut							○													
ServiceCode		○																		
RqPath	構造体_sREQUEST_PATH または_sREQUEST_PATH_EX 詳細は「「RqPath」のデータ型 (P.2-1019)」参照																			
ServiceDat		○	○	○	○	○	○	○	○	○	○	○	○	○	○					
	配列、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			
Size							○													
RespServiceDat		○	○	○	○	○	○	○	○	○	○	○	○	○	○					
	配列、構造体の 1 メンバ、共用体の 1 メンバも指定可能																			
RespSize							○													

機能

サービスコード「ServiceCode」で指定したサービスに対応して、サービスデータ「ServiceDat」を、UCMM メッセージとして、送信します。
送信先は、ルートパス「RoutePath」で指定します。
リクエストパスは、「RqPath」で指定します。

送信する要素数は、「Size」で指定します。
「ServiceDat」が配列の場合は、送信する要素数を指定します。
「ServiceDat」が配列ではない場合は、必ず「1」を指定します。
サービスデータが不要の場合は、「Size」の値を「0」にします。

送信後、レスポンスデータが、「RespServiceDat」に格納されます。また、レスポンスデータのサイズが、バイト単位で「RespSize」に格納されます。

タイムアウト時間は、「TimeOut」で指定します。タイムアウト時間以内にレスポンスが返ってこないと、通信失敗と判断されます。

「ClassIDLogicalFormat」、「InstanceIDLogicalFormat」、「AttributeIDLogicalFormat」のデータ型は、列挙型_eCIP_LOGICAL_FORMAT です。
列挙型_eCIP_LOGICAL_FORMAT の、列挙子の意味は以下のとおりです。

列挙子	意味
_8BIT	8 ビット
_16BIT	16 ビット
_32BIT	32 ビット

「ErrorID」の値が WORD#16#1C00 の場合に、「ErrorIDEx」に CIP メッセージ異常コードが格納されます。
「ErrorIDEx」の値と意味は、相手ノードによって規定されます。相手ノードのマニュアルを参照してください。

「RqPath」のデータ型

「RqPath」のデータ型は、構造体_sREQUEST_PATH または_sREQUEST_PATH_EX です。
通常は_sREQUEST_PATH を使用してください。
任意の論理フォーマットのサイズを設定して使用する場合は_sREQUEST_PATH_EX を使用してください。

● _sREQUEST_PATH 型

変数	名称	内容	データ型	有効範囲	単位	初期値
RqPath	リクエストパス	リクエストパス	_sREQUEST_PATH	–	–	–
ClassID	クラス ID	クラス ID	UINT	データ型に従う	–	0
InstanceID	インスタンス ID	インスタンス ID	UINT			
isAttributeID	アトリビュート有無	TRUE: アトリビュート ID 使用 FALSE: アトリビュート ID 不使用	BOOL			FALSE
AttributeID	アトリビュート ID	アトリビュート ID	UINT			0

（注）_sREQUEST_PATH 型の場合、各 ID の論理フォーマットサイズは 16 ビットです。

● **_sREQUEST_PATH_EX 型**

変数	名称	内容	データ型	有効範囲	単位	初期値
RqPath	リクエストパス	リクエストパス	_sREQUEST_PATH_EX	–	–	–
ClassIDLogicalFormat	クラス ID 論理フォーマット	クラス ID のデータサイズ	_eCIP_LOGICAL_FORMAT	データ型に従う	–	_8BIT
ClassID	クラス ID	クラス ID	UDINT			0
InstanceIDLogicalFormat	インスタンス ID 論理フォーマット	インスタンス ID のデータサイズ	_eCIP_LOGICAL_FORMAT			_8BIT
InstanceID	インスタンス ID	インスタンス ID	UDINT			0
isAttributeID	アトリビュート有無	TRUE: アトリビュート ID 使用 FALSE: アトリビュート ID 不使用	BOOL			FALSE
AttributeIDLogicalFormat	アトリビュート ID 論理フォーマット	アトリビュート ID のデータサイズ	_eCIP_LOGICAL_FORMAT			_8BIT
AttributeID	アトリビュート ID	アトリビュート ID	UDINT			0

配列を送受信する場合

「ServiceDat」または「RespServiceDat」が配列の場合、それらの変数には、必ず添え字付きの配列の要素をパラメータとして渡します。

読み書きできるデータサイズ

読み出せるデータの最大サイズは、492 バイトです。
書き込めるデータの最大サイズは、リクエストパスのアトリビュート有無、およびルートパスによって、以下のように変わります。

書き込めるデータサイズ[バイト]＝基本サイズ－アトリビュート有無－経路情報サイズ

上式の項目	意味
基本サイズ	500 バイト
アトリビュート有無	アトリビュート ID 使用の場合、14 バイト アトリビュート ID 不使用の場合、10 バイト

上式の項目	意味
経路情報サイズ	• ホップがない場合、経路情報サイズは 0 バイトです。*1 • ホップがある場合、経路情報サイズは、ルートパスサイズ+12 バイトとなります。 • ルートパスサイズとは、ルートパスの文字列の ASCII コードのバイト数のことです。 • ただし、以下のことに注意してください。 a）アドレス部の先頭に“#”が付く場合、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 b）アドレス部の先頭に“#”が付かない場合、ネットワーク部は 2 バイトとして計算します。 c）アドレス部の先頭に“#”が付かず、かつアドレス部の ASCII コードのバイト数が奇数の場合、その値に 1 バイトを加算します。 d）ルートパスの階層の区切りを示す“\”は、ルートパスサイズに含みません。 e）最初のホップは、ルートパスサイズに含みません。 （例 1）ルートパスが、'01\#11\02\192.168.250.2\01\#01'の場合 • 最初のホップはルートパスサイズに含まないので、先頭の“01\#11”は無視します。 • そのつぎの、ネットワーク種別“02”は、2 バイトです。 • そのつぎの、アドレス部“192.168.250.2”は、13 バイトです。奇数なので、1 を加えて 14 バイトとします。 • そのつぎの、“01\#01”は、アドレス部に“#”が付いているので、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 • 上記をすべて加えて、ルートパスサイズは、18 バイトとなります。 • ルートパスサイズに 12 バイトを加えて、経路情報サイズは 30 バイトとなります。 （例 2）ルートパスが、'02\192.168.250.2\01\#00'の場合 • 最初のホップはルートパスサイズに含まないので、先頭の“02\192.168.250.2”は無視します。 • そのつぎの、“01\#00”は、アドレス部に“#”が付いているので、ネットワーク部とアドレス部を合わせて 2 バイトとして計算します。 • したがって、ルートパスサイズは 2 バイトとなります。 • ルートパスサイズに 12 バイトを加えて、経路情報サイズは 14 バイトとなります。 （例 3）ルートパスが、'02\192.168.250.2'の場合 • ホップがないので、経路情報サイズは 0 バイトです。

*1. ホップとは、送信ノードと受信ノードの間で、ルーチングを行うことです。たとえば、ルートパスが、'02\192.168.250.2\01\#00'の場合、いったん IP アドレス 192.168.250.2 のノードにルーチングして、00 号機にメッセージを送るので、ホップがひとつあることになります。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

CIP 通信の詳細は、以下のマニュアルを参照してください。

• 『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「ServiceDat」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 他局コントローラがオムロン製の場合、書き込める変数は、ネットワークに公開されているものだけです。事前に、その変数をネットワーク公開指定しておいてください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「RqPath.ClassIDLogicalFormat」、「RqPath.AttributeIDLogicalFormat」の値が有効範囲外の場合。
  b)「RqPath.ClassIDLogicalFormat」で指定したサイズと「RqPath.ClassID」のデータサイズ、または「RqPath.InstanceIDLogicalFormat」で指定したサイズと「RqPath.InstanceID」のデータサイズ、または「RqPath.AttributeIDLogicalFormat」で指定したサイズと「RqPath.AttributeID」のデータサイズが、不一致の場合。
  c)「TimeOut」の値が、有効範囲外の場合。
  d)「Size」の値が、書込データ領域を超える場合。
  e)「Size」の値が、「ServiceDat」の領域を超える場合。
  f)「RespSize」の値が、「RespServiceDat」の領域を超える場合。
  g)「ServiceDat」に、サポートしていないデータ型を指定した場合。
  h)「RespServiceDat」に、サポートしていないデータ型を指定した場合。
  i)「RqPath」に、_sREQUEST_PATH または_sREQUEST_PATH_EX 以外のデータ型の変数を指定した場合。
  j) 自 IP アドレスの、設定異常が発生している場合。
  k) IP 重複異常が発生している場合。
  l) BOOTP サーバ異常が発生している状態で、本命令が実行された場合。
  m) CIP で規定されるエラーレスポンスが返った場合。
  n)「RoutePath」の文字列が、不正の場合。
  o) CIP 関連命令が、同時に 32 個を超えて実行された場合。
  p) タイムアウト時間を超えても応答がない場合。

サンプルプログラム

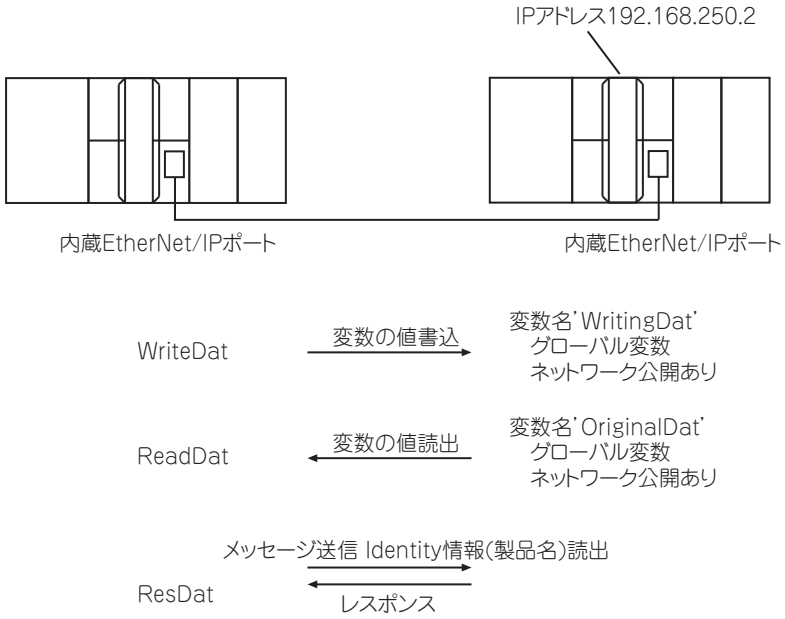
CIP の UCMM メッセージを使用して、変数書込、変数読出、メッセージ送信を行います。
コントローラどうしは、EtherNet/IP ネットワークで接続しており、相手ノードの IP アドレスは 192.168.250.2 とします。
処理の手順は、以下のようになります。

1 **CIPUCMMWrite 命令を使用して、相手ノードの変数に値を書き込みます。他局変数名は「WritingDat」、書き込む値は、変数「WriteDat」の中身とします。**
「WritingDat」は、相手ノードでグローバル変数として定義されており、かつ、ネットワーク公開されていることが必要です。

2 **CIPUCMMRead 命令を使用して、相手ノードの変数の値を読み出します。他局変数名は「OriginalDat」、読み出した値の格納先は、変数「ReadDat」とします。**
「OriginalDat」は、相手ノードでグローバル変数として定義されており、かつ、ネットワーク公開されていることが必要です。

3 **CIPUCMMSend 命令を使用して、相手ノードに Explicit メッセージを送信します。**
メッセージの内容は、Identity 情報（製品名）読出です。
その際のクラス ID、インスタンス ID、アトリビュート ID、サービスコードは、以下のとおりです。レスポンスデータは、変数「ResDat」に格納します。

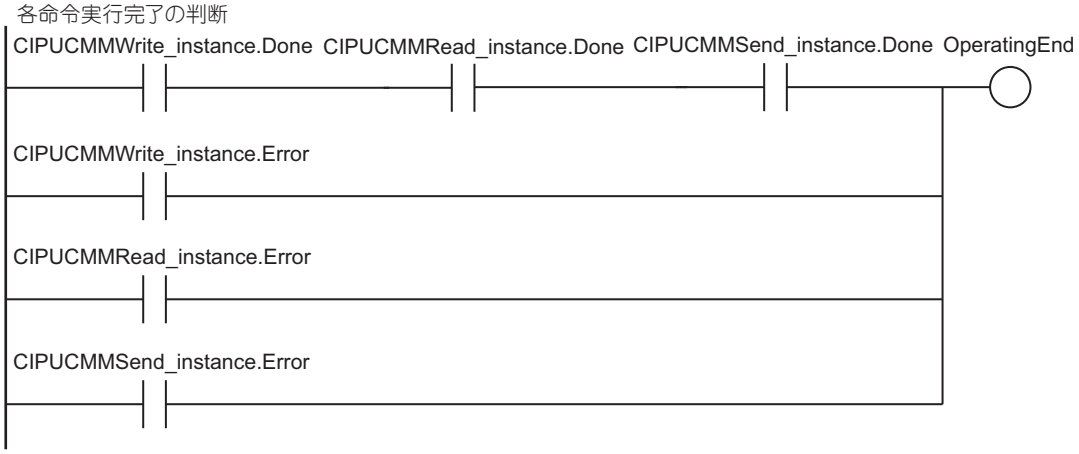
項目	値
クラス ID	1
インスタンス ID	1
アトリビュート ID	7
サービスコード	16#0E



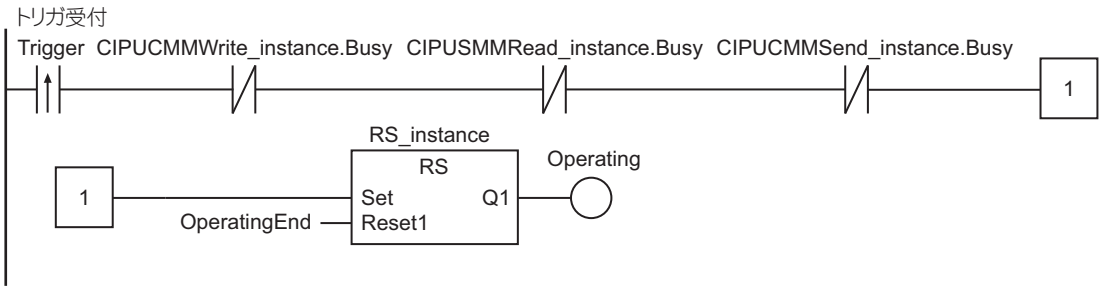
LD

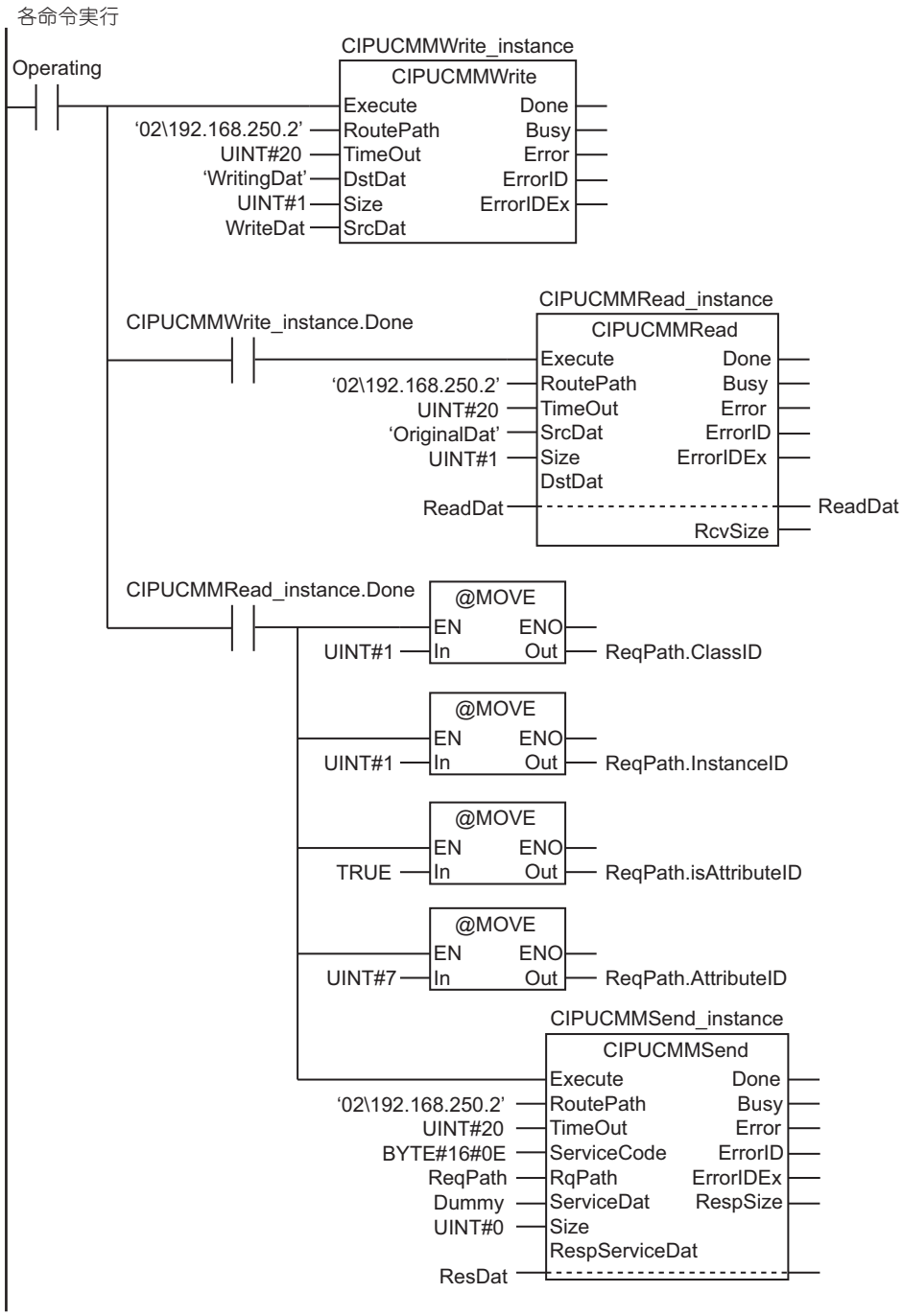
名称	データ型	初期値	コメント
OperatingEnd	BOOL	FALSE	処理終了
Trigger	BOOL	FALSE	実行条件
Operating	BOOL	FALSE	処理中
WriteDat	INT	1234	書込データ
ReadDat	INT	0	読出データ
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	リクエストパス
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	レスポンスデータ
Dummy	BYTE	16#0	ダミー
RS_instance	RS		
CIPUCMMWrite_instance	CIPUCMMWrite		
CIPUCMMRead_instance	CIPUCMMRead		
CIPUCMMSend_instance	CIPUCMMSend		

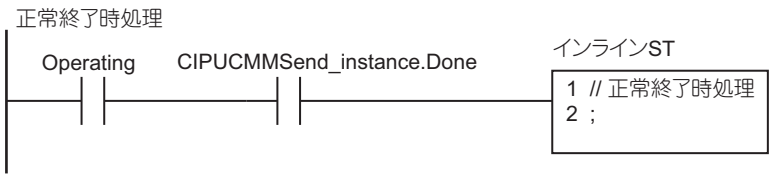
各命令実行完了の判断

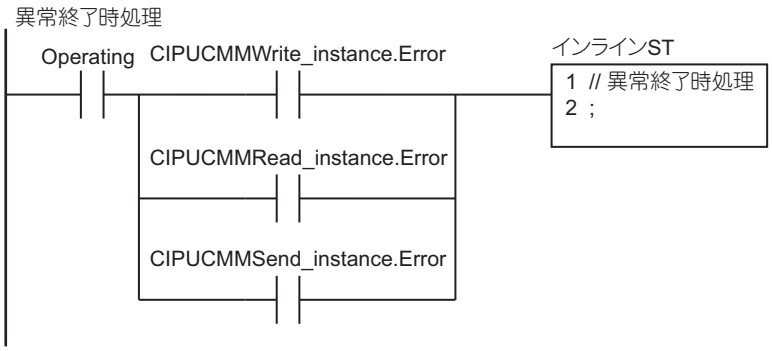


トリガ受付




各命令実行
Operating
CIPUCMMWrite_instance
CIPUCMMWrite
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
'WritingDat'
DstDat
ErrorID
UINT#1
Size
ErrorIDEx
WriteDat
SrcDat
CIPUCMMWrite_instance.Done
CIPUCMMRead_instance
CIPUCMMRead
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
'OriginalDat'
SrcDat
ErrorID
UINT#1
Size
ErrorIDEx
DstDat
ReadDat
ReadDat
RcvSize
CIPUCMMRead_instance.Done
@MOVE
EN
ENO
UINT#1
In
Out
ReqPath.ClassID
@MOVE
EN
ENO
UINT#1
In
Out
ReqPath.InstanceID
@MOVE
EN
ENO
TRUE
In
Out
ReqPath.isAttributeID
@MOVE
EN
ENO
UINT#7
In
Out
ReqPath.AttributeID
CIPUCMMSend_instance
CIPUCMMSend
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
BYTE#16#0E
ServiceCode
ErrorID
ReqPath
RqPath
ErrorIDEx
Dummy
ServiceDat
RespSize
UINT#0
Size
RespServiceDat
ResDat


正常終了時処理
Operating
CIPUCMMSend_instance.Done
インラインST
1 // 正常終了時処理
2 ;



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoUCMMTrigger	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	WriteDat	INT	0	書込データ
	ReadDat	INT	0	読出データ
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	リクエストパス
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	レスポンスデータ
	Dummy	BYTE	16#0	ダミー
	CIPUCMMWrite_instance	CIPUCMMWrite		
	CIPUCMMRead_instance	CIPUCMMRead		
	CIPUCMMSend_instance	CIPUCMMSend		

外部変数	名称	コンスタント	データ型	コメント
	_EIP_EtnOnlineSta	☑	BOOL	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoUCMMTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoUCMMTrigger          :=TRUE;
    Stage                  :=INT#1;
    CIPUCMMWrite_instance(
        Execute            :=FALSE,                // インスタンス初期化
        SrcDat             :=WriteDat);            // ダミー
    CIPUCMMRead_instance(                          // インスタンス初期化
        Execute            :=FALSE,                // ダミー
        DstDat             :=ReadDat);             // ダミー
    CIPUCMMSend_instance(
        Execute            :=FALSE,                // インスタンス初期化
        ServiceDat         := Dummy,               // ダミー
        RespServiceDat:=ResDat);                   // ダミー
END_IF;
```

```
IF (DoUCMMTrigger=TRUE) THEN
    CASE Stage OF
    1 :                                     // 変数書込要求
        CIPUCMMWrite_instance(
            Execute  :=TRUE,
            RoutePath:='02¥192.168.250.2',      // ルートパス
            TimeOut  :=UINT#20,                 // タイムアウト時間
            DstDat   :='WritingDat',            // 他局変数名
            Size     :=UINT#1,                  // 書込要素数
            SrcDat   :=WriteDat);               // 書込データ

        IF (CIPUCMMWrite_instance.Done=TRUE) THEN
            Stage:=INT#2;                               // 正常終了
        ELSIF (CIPUCMMWrite_instance.Error=TRUE) THEN
            Stage:=INT#10;                              // 異常終了
        END_IF;
    2 :                                     // 変数読出要求
        CIPUCMMRead_instance(
            Execute   :=TRUE,
            RoutePath :='02¥192.168.250.2',             // ルートパス
            TimeOut   :=UINT#20,                        // タイムアウト時間
            SrcDat    :='OriginalDat',                  // 他局変数名
            Size      :=UINT#1,                         // 読出要素数
            DstDat    :=ReadDat);                       // 読出データ

        IF (CIPUCMMRead_instance.Done=TRUE) THEN
            Stage:=INT#3;                               // 正常終了
        ELSIF (CIPUCMMRead_instance.Error=TRUE) THEN
            Stage:=INT#40;                              // 異常終了
        END_IF;

    3 :                                     // 任意メッセージ送信
        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',        // ルートパス
            TimeOut        :=UINT#20,                   // タイムアウト値
            ServiceCode    :=BYTE#16#0E,                // サービスコード
            RqPath         :=ReqPath,                   // リクエストパス
            ServiceDat     := Dummy,                    // サービスデータ
            Size           :=UINT#0,                    // 要素数
            RespServiceDat :=ResDat);                   // レスポンスデータ

        IF (CIPUCMMSend_instance.Done=TRUE) THEN
            Stage:=INT#0;                               // 正常終了
        ELSIF (CIPUCMMSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                              // 異常終了
```

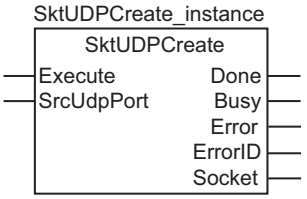
```
        END_IF;

    0 :                                 // 正常終了処理
        DoUCMMTrigger:=FALSE;
        Trigger      :=FALSE;

    ELSE                                    // 異常終了処理
        DoUCMMTrigger:=FALSE;
        Trigger      :=FALSE;
    END_CASE;
END_IF;
```

SktUDPCreate

EtherNet/IP の、UDP ソケットのクリエイト要求（サーバポートのオープン）を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktUDPCreate	UDP ソケットクリエイト	FB		SktUDPCreate_instance(Execute, SrcUdpPort, Done, Busy, Error, ErrorID, Socket);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
SrcUdpPort	自 UDP ポート番号	入力	自 UDP ポート番号	1 ～ 65535	−	1
Socket	ソケット	出力	ソケット	−	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcUdpPort							○													
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1047)」参照																			

機能

自 UDP ポート番号「SrcUdpPort」で指定したポートを、オープンします（ソケット関数 Socket()、Bind()の実行）。
オープンしたソケットの情報が、「Socket」に格納されます。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、UDP ポートがオープンしています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. メンバは、「0」または、NULL 出力されます。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令で生成したハンドルは、SktClose 命令を使用してクローズしてください。
- 本命令で生成したハンドルは、プログラムモード変更時に無効になります。

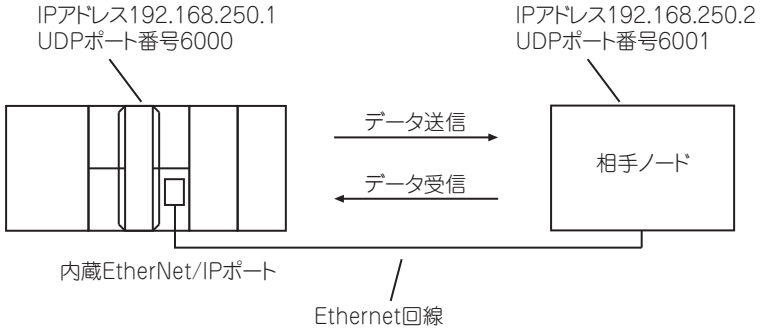
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「SrcUdpPort」の値が、有効範囲外の場合。
  c）「SrcUdpPort」で指定したポートが、すでにオープンしている、または、クローズ処理中の場合。
  d）「SrcUdpPort」で指定したポートが、すでに使用されている場合。

参考

- 同時にオープンできるソケット数は、最大 30 です。ソケット数は、UDP ソケットと TCP ソケットを合わせた数です。
- 「Error」が TRUE となっても「Socket」の値を変更しません。

サンプルプログラム

NY シリーズ コントローラと、相手ノードとの間で、ソケットサービス機能(UDP)を使用して、データの送受信を行います。



NY シリーズ コントローラのユーザプログラム

処理の手順は、以下のとおりです。

1 **SktUDPCreate 命令で、UDP ソケットのクリエイト要求を行います。**

2 **SktUDPSend 命令で、送信要求を行います。送信するデータは、SendSocketDat[]の中身とします。**

3 **SktUDPRcv 命令で、受信要求を行います。受信したデータは、RcvSocketDat[]に格納します。**

4 **SktClose 命令で、ソケットをクローズします。**

● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoSendAndRcv	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移

内部変数	名称	データ型	初期値	コメント
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	受信データ
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	ソケット
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	送信データ
	SktUDPCreate_instance	SktUDPCreate		
	SktUDPSend_instance	SktUDPSend		
	SktUDPRcv_instance	SktUDPRcv		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP_EtnOnlineSta	BOOL	☑	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage       :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);     // インスタンス初期化
    SktUDPSend_instance(                       // インスタンス初期化
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);            // ダミー
    SktUDPRcv_instance(                        // インスタンス初期化
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);             // ダミー
    SktClose_instance(Execute:=FALSE);         // インスタンス初期化
END_IF;

IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
    1 :                                  // クリエイト要求
        SktUDPCreate_instance(
            Execute   :=TRUE,
            SrcUdpPort:=UINT#6000,               // 自 UDP ポート番号
            Socket    =>WkSocket);               // ソケット

    IF (SktUDPCreate_instance.Done=TRUE) THEN
        Stage:=INT#2;                        // 正常終了
    ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
        Stage:=INT#10;                       // 異常終了
    END_IF;

    2 :                                  // 送信要求
        WkSocket.DstAdr.PortNo:=UINT#6001;
        WkSocket.DstAdr.IpAdr :='192.168.250.2';
        SktUDPSend_instance(
            Execute:=TRUE,
```

```
            Socket :=WkSocket,                  // ソケット
            SendDat:=SendSocketDat[0],          // 送信データ
            Size   :=UINT#2000);                // 送信データサイズ

        IF (SktUDPSend_instance.Done=TRUE) THEN
                Stage:=INT#3;                       // 正常終了
        ELSIF (SktUDPSend_instance.Error=TRUE) THEN
                Stage:=INT#20;                      // 異常終了
        END_IF;

    3 :                                 // 受信要求
        SktUDPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,                  // ソケット
            TimeOut:=UINT#0,                    // タイムアウト時間
            Size   :=UINT#2000,                 // 受信データサイズ
            RcvDat :=RcvSocketDat[0]);          // 受信データ

        IF (SktUDPRcv_instance.Done=TRUE) THEN
            Stage:=INT#4;                       // 正常終了
        ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
            Stage:=INT#30;                      // 異常終了
        END_IF;

    4 :                                 // クローズ要求
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);                 // ソケット

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                       // 正常終了
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                      // 異常終了
        END_IF;

    0 :                                 // 正常終了
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;

    ELSE                                // 異常による中断
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;
    END_CASE;

END_IF;
```

相手ノードのユーザプログラム

この例では、相手ノードにもユーザプログラムが必要です。上記の処理手順とは、送信と受信の順序が逆になります。

1 **SktUDPCreate 命令で、UDP ソケットのクリエイト要求を行います。**

2 **SktUDPRcv 命令で、受信要求を行います。受信したデータは、RcvSocketDat[]に格納します。**

3 **SktUDPSend 命令で、送信要求を行います。送信するデータは、SendSocketDat[]の中身とします。**

4 **SktClose 命令で、ソケットをクローズします。**

● **ST**

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoSendAndRcv	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	受信データ
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	ソケット
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	送信データ
	SktUDPCreate_instance	SktUDPCreate		
	SktUDPSend_instance	SktUDPSend		
	SktUDPRcv_instance	SktUDPRcv		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP_EtnOnlineSta	BOOL	☑	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage       :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);  // インスタンス初期化
    SktUDPSend_instance(                    // インスタンス初期化
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);         // ダミー
    SktUDPRcv_instance(                     // インスタンス初期化
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);          // ダミー
    SktClose_instance(Execute:=FALSE);      // インスタンス初期化
END_IF;

IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
```

```
1 :                                 // クリエイト要求
    SktUDPCreate_instance(
        Execute    :=TRUE,
        SrcUdpPort:=UINT#6001,              // 自 UDP ポート番号
        Socket     =>WkSocket);             // ソケット

    IF (SktUDPCreate_instance.Done=TRUE) THEN
        Stage:=INT#2;                       // 正常終了
    ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
        Stage:=INT#10;                      // 異常終了
    END_IF;
2 :                                 // 受信要求
    SktUDPRcv_instance(
        Execute:=TRUE,
        Socket :=WkSocket,                  // ソケット
        TimeOut:=UINT#0,                    // タイムアウト時間
        Size   :=UINT#2000,                 // 受信データサイズ
        RcvDat :=RcvSocketDat[0]);          // 受信データ

    IF (SktUDPRcv_instance.Done=TRUE) THEN
        Stage:=INT#3;                       // 正常終了
    ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
        Stage:=INT#20;                      // 異常終了
    END_IF;

3 :                                 // 送信要求
    WkSocket.DstAdr.PortNo:=UINT#6000;
    WkSocket.DstAdr.IpAdr :='192.168.250.1';
    SktUDPSend_instance(
        Execute:=TRUE,
    Socket :=WkSocket,                  // ソケット
    SendDat:=SendSocketDat[0],          // 送信データ
    Size   :=UINT#2000);                // 送信データサイズ

IF (SktUDPSend_instance.Done=TRUE) THEN
    Stage:=INT#4;                       // 正常終了
ELSIF (SktUDPSend_instance.Error=TRUE) THEN
    Stage:=INT#30;                      // 異常終了
END_IF;

4 :                                 // クローズ要求
    SktClose_instance(
        Execute:=TRUE,
        Socket :=WkSocket);                 // ソケット

    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=INT#0;                       // 正常終了
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=INT#40;                      // 異常終了
    END_IF;
```

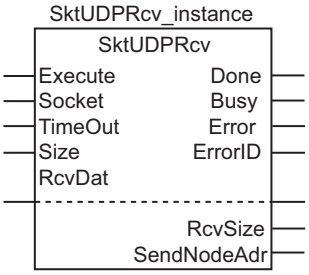
```
        0 :                             // 正常終了
            DoSendAndRcv:=FALSE;
            Trigger     :=FALSE;

        ELSE                            // 異常による中断
            DoSendAndRcv:=FALSE;
            Trigger     :=FALSE;
        END_CASE;

    END_IF;
```

SktUDPRcv

EtherNet/IP の、UDP ソケットの受信バッファ内のデータを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktUDPRcv	UDP ソケット受信	FB		SktUDPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize, SendNodeAdr);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	−	−	−
TimeOut	タイムアウト時間		0: タイムアウトなし 1 ～ 65535: 0.1 ～ 6553.5s	データ型に従う	0.1s	0
Size	格納サイズ		受信バッファから読み出すデータサイズ	0 ～ 2000	バイト	1
RcvDat[] 配列	受信データ	入出力	受信データ	データ型に従う	−	−
RcvSize	受信データサイズ	出力	実際に RcvDat[]に格納したデータサイズ	0 ～ 2000	バイト	−
SendNode Adr	送信元ノードアドレス		送信元ノードアドレス	−	−	

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1055)」参照																			
TimeOut							○													
Size							○													
RcvDat[] 配列		○																		
RcvSize							○													
SendNode Adr	構造体_sSOCKET_ADDRESS　詳細は「機能 (P.2-1055)」参照																			

機能

「Socket」で指定したソケットの、受信バッファ内のデータを、受信データ RcvDat[]に格納します。
格納するデータのサイズは、「Size」で指定します。
データ格納後に、実際に格納したデータサイズを、「RcvSize」に代入します。
送信元のノードアドレスは、「SendNodeAdr」に格納します。

もし、受信バッファにデータがなかった場合は、タイムアウト時間「TimeOut」に設定された時間だけ、受信データを待ちます。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、RcvDat[]へのデータ格納が完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. このメンバを、本命令で使用することはありません。

「SendNodeAdr」のデータ型は、構造体_sSOCKET_ADDRESS です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SendNodeAdr	送信元ノードアドレス	送信元ノードアドレス	_sSOCKET_ADDRESS	—	—	—
PortNo	ポート番号	送信元ノードの UDP ポート番号	UINT	1 ～ 65535	—	—
IpAdr	IP アドレス	送信元ノードの IP アドレス	STRING	データ型に従う		

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

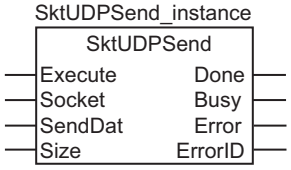
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 一度に受信できるデータサイズは、最大 2000 バイトです。
- 指定したソケットが受信しているデータサイズが「Size」の値より小さい場合、受信しているデータだけが RcvDat[]に格納されます。そして、格納したデータサイズを「RcvSize」に代入します。
- 指定したソケットが受信しているデータサイズが「Size」の値より大きい場合、「Size」分のデータだけが RcvDat[]に格納されます。
- 「Size」の値が 0 の場合、受信データの読出を行いません。
- 受信バッファにデータがなかった場合、ソケットを SktClose 命令でクローズすると、タイムアウト時間未満でも、受信データを待つのをやめて正常終了します。この場合、「RcvSize」の値は 0 になります。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「Socket」で指定したソケットが、受信処理中の場合。
  c）「Socket」で指定したソケットが、オープンしていない場合。
  d）「Socket.Handle」で指定したハンドルが、存在しない場合。

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.2-1049)」を参照してください。

SktUDPSend

EtherNet/IP の UDP ポートから、データを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktUDPSend	UDP ソケット送信	FB		SktUDPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	－	－	－
SendDat[]配列	送信データ		送信データ	データ型に従う		
Size	送信データサイズ		送信データサイズ	0 ～ 2000	バイト	1

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1058)」参照																			
SendDat[]配列		○																		
Size							○													

機能

「Socket」で指定したソケットから、送信データ SendDat[]を送信します。
送信するデータサイズは、「Size」で指定します。
相手ノードは、「Socket.DstAdr」で指定します。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、SendDat[]の、送信バッファへの格納が完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"
DstAdr	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"

*1. このメンバを、本命令で使用することはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 一度に送信できるデータサイズは、最大 2000 バイトです。SendDat[]の配列の大きさが 2000 バイトより大きくても、2000 バイトまでしか送信できません。さらに、ブロードキャストアドレス指定時は、1472 バイトまでしか送信できません。
- 「Size」の値が 0 の場合、0 バイトの送信データを回線に送信します。

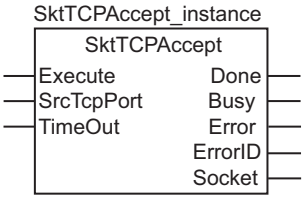
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「Socket」のメンバのいずれかが、有効範囲外の場合。
  c）「Socket」で指定したソケットが、送信処理中の場合。
  d）「Socket」で指定したソケットが、オープンしていない場合。
  e）「Socket」で相手ノードをドメイン名指定し、アドレス解決に失敗した場合。
  f）「Socket.Handle」で指定したハンドルが、存在しない場合。
  g）「Size」の値が、SendDat[]の要素数を超える場合。

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.2-1049)」を参照してください。

SktTCPAccept

EtherNet/IP の、TCP ソケットのアクセプト要求を行います。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPAccept	TCP ソケットアクセプト	FB		SktTCPAccept_instance(Execute, SrcTcpPort, TimeOut, Done, Busy, Error, ErrorID, Socket);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
SrcTcpPort	自 TCP ポート番号	入力	自 TCP ポート番号	1 ～ 65535	−	1
TimeOut	タイムアウト時間		0: タイムアウトなし 1 ～ 65535: 0.1 ～ 6553.5s	データ型に従う	0.1s	0
Socket	ソケット	出力	ソケット	−	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							O													
TimeOut							O													
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1061)」参照																			

機能

自 TCP ポート番号「SrcTcpPort」で指定したポートの、アクセプト要求を行います（ソケット関数 Socket()、Bind()、Listen()、Accept()の実行)。
通信相手からのコネクション確立を、タイムアウト時間「TimeOut」に設定された時間だけ待ちます。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、コネクションが確立しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	−	−	−
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	−	0
SrcAdr	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	−	−	−
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	−	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	−	−	−
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	−	0
IpAdr	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. メンバは、NULL 出力されます。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPln1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

- ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。
- 本命令を複数回実行することにより、ひとつの自ポート番号で複数のクライアントからのコネクションを開設することができます。その場合、コネクションごとに異なるソケットを返します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令で生成したハンドルは、SktClose 命令を使用してクローズしてください。
- 本命令で生成したハンドルは、プログラムモード変更時に無効になります。

- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「SrcTcpPort」の値が、有効範囲外の場合。
  c）「SrcTcpPort」で指定した TCP ポートが、オープン処理中の場合。
  d）「SrcTcpPort」で指定した TCP ポートが、クローズ処理中の場合。
  e）「TimeOut」で指定した時間だけ待っても、コネクションが確立しない場合。



参考

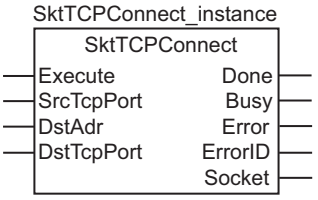
- 同時にオープンできるソケット数は、最大 30 です。ソケット数は、UDP ソケットと TCP ソケットを合わせた数です。
- 「Error」が TRUE となっても「Socket」の値を変更しません。

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktTCPConnect

EtherNet/IP から、相手 TCP ポートにコネクト（接続）します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPConnect	TCP ソケットコネクト	FB		SktTCPConnect_instance(Execute, SrcTcpPort, DstAdr, DstTcpPort, Done, Busy, Error, ErrorID, Socket);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
SrcTcpPort	自 TCP ポート番号	入力	自 TCP ポート番号 ただし、0 の場合は、Well-Known ポート以外の 1024 から空いている TCP ポートを自動割当	データ型に従う	－	0
DstAdr	相手アドレス		相手の IP アドレスまたはホスト名	最大 200 バイト		－
DstTcpPort	相手 TCP ポート番号		相手 TCP ポート番号	1 ～ 65535		1
Socket	ソケット	出力	ソケット	－	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							○													
DstAdr																				○
DstTcpPort							○													
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1064)」参照																			

機能

自 TCP ポート番号「SrcTcpPort」と、相手アドレス「DstAdr」の相手 TCP ポート番号「DstTcpPort」を接続するためのコネクト要求を行います（ソケット関数 Connect()の実行）。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、コネクションが確立しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	－	－	－
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	－	0
SrcAdr	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	－	－	－
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	－	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	－	－	－
PortNo	ポート番号	ポート番号	UINT	1 ～ 65535	－	0
IpAdr	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. メンバは、NULL 出力されます。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令で生成したハンドルは、SktClose 命令を使用してクローズしてください。
- 本命令で生成したハンドルは、プログラムモード変更時に無効になります。

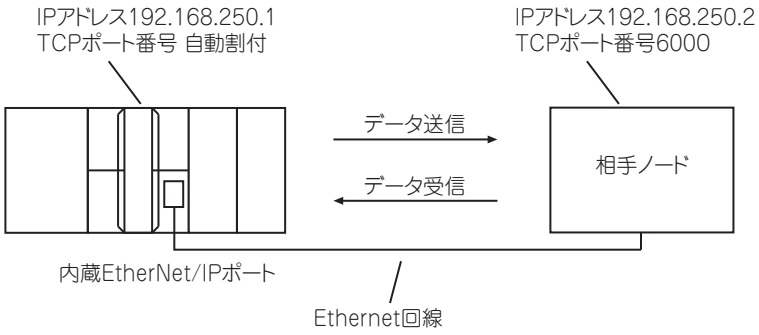
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「DstAdr」の値が、有効範囲外の場合。
  c）「DstTcpPort」の値が、有効範囲外の場合。
  d）「SrcTcpPort」で指定した TCP ポートが、すでにオープンしている場合。
  e）「DstAdr」で指定した相手ノードが、存在しない場合。
  f）「DstAdr」、「DstTcpPort」で指定した相手ノードが、コネクト待ち状態ではない場合。
  g）「DstAdr」で指定したホスト名の、アドレス解決に失敗した場合。
  h）同一クライアント（IP アドレス、TCP ポート）とのコネクションが、すでに確立している場合。

参考

- 同時にオープンできるソケット数は、最大 30 です。ソケット数は、UDP ソケットと TCP ソケットを合わせた数です。
- 「Error」が TRUE となっても「Socket」の値を変更しません。

サンプルプログラム

NY シリーズ コントローラと、相手ノードとの間で、ソケットサービス機能(TCP)を使用して、データの送受信を行います。



NY シリーズ コントローラのユーザプログラム

処理の手順は、以下のとおりです。

1 **SktTCPConnect 命令で、相手ノードの TCP ポートにコネクト要求を行います。**

2 **SktClearBuf 命令で、TCP ソケットの受信バッファをクリアします。**

3 **SktGetTCPStatus 命令で、TCP ソケットのステータスを読み出します。**

4 **SktTCPSend 命令で、送信要求を行います。送信するデータは、SendSocketDat[]の中身とします。**

5 **SktTCPRcv 命令で、受信要求を行います。受信したデータは、RcvSocketDat[]に格納します。**

6 **SktClose 命令で、ソケットをクローズします。**

● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoTCP	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	受信データ
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	ソケット
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	送信データ
	SktTCPConnect_instance	SktTCPConnect		
	SktClearBuf_instance	SktClearBuf		
	SktGetTCPStatus_instance	SktGetTCPStatus		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP_EtnOnlineSta	BOOL	✉	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage:=INT#1;
    SktTCPConnect_instance(Execute:=FALSE);  // インスタンス初期化
    SktClearBuf_instance(Execute:=FALSE);  // インスタンス初期化
    SktGetTCPStatus_instance(Execute:=FALSE);  // インスタンス初期化
    SktTCPSend_instance(  // インスタンス初期化
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);  // ダミー
    SktTCPRcv_instance(  // インスタンス初期化
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);  // ダミー
    SktClose_instance(Execute:=FALSE);  // インスタンス初期化
END_IF;

IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :  // コネクト要求
        SktTCPConnect_instance(
            Execute    :=TRUE,
            SrcTcpPort:=UINT#0,   // 自 TCP ポート番号　自動割付
            DstAdr     :='192.168.250.2',   // 相手 IP アドレス
            DstTcpPort:=UINT#6000,   // 相手 TCP ポート番号
            Socket     =>WkSocket);  // ソケット
```

```
    IF (SktTCPConnect_instance.Done=TRUE) THEN
        Stage:=INT#2;  // 正常終了
    ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
        Stage:=INT#10;  // 異常終了
    END_IF;

2 :  // 受信バッファクリア
    SktClearBuf_instance(
        Execute:=TRUE,
        Socket :=WkSocket);  // ソケット

    IF (SktClearBuf_instance.Done=TRUE) THEN
        Stage:=INT#3;  // 正常終了
    ELSIF (SktClearBuf_instance.Error=TRUE) THEN
        Stage:=INT#20;  // 異常終了
    END_IF;

3 :  // ステータス読出要求
    SktGetTCPStatus_instance(
        Execute:=TRUE,
        Socket :=WkSocket);  // ソケット

    IF (SktGetTCPStatus_instance.Done=TRUE) THEN
        Stage:=INT#4;  // 正常終了
    ELSIF (SktGetTCPStatus_instance.Error=TRUE) THEN
        Stage:=INT#30;  // 異常終了
    END_IF;

4 :  // 送信要求
    SktTCPSend_instance(
        Execute:=TRUE,
        Socket :=WkSocket,  // ソケット
        SendDat:=SendSocketDat[0],  // 送信データ
        Size   :=UINT#2000);  // 送信データサイズ

    IF (SktTCPSend_instance.Done=TRUE) THEN
        Stage:=INT#5;  // 正常終了
    ELSIF (SktTCPSend_instance.Error=TRUE) THEN
        Stage:=INT#40;  // 異常終了
    END_IF;

5 :  // 受信要求
    SktTCPRcv_instance(
        Execute:=TRUE,
        Socket :=WkSocket,  // ソケット
        TimeOut:=UINT#0,  // タイムアウト時間
        Size   :=UINT#2000,  // 受信データサイズ
        RcvDat :=RcvSocketDat[0]);  // 受信データ

    IF (SktTCPRcv_instance.Done=TRUE) THEN
```

```
            Stage:=INT#6;  // 正常終了
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#50;  // 異常終了
        END_IF;

    6 :  // クローズ要求
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);  // ソケット

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;  // 正常終了
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;  // 異常終了
        END_IF;

    0 :  // 正常終了
        DoTCP  :=FALSE;
        Trigger:=FALSE;

    ELSE  // 異常による中断
        DoTCP  :=FALSE;
        Trigger:=FALSE;
    END_CASE;

END_IF;
```

相手ノードのユーザプログラム

この例では、相手ノードにもユーザプログラムが必要です。上記の処理手順とは、送信と受信の順序が逆になります。

1 **SktTCPAccept 命令で、TCP ソケットのアクセプト要求を行います。**

2 **SktTCPRcv 命令で、受信要求を行います。受信したデータは、RcvSocketDat[]に格納します。**

3 **SktTCPSend 命令で、送信要求を行います。送信するデータは、SendSocketDat[]の中身とします。**

4 **SktClose 命令で、ソケットをクローズします。**

● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoTCP	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	受信データ

内部変数	名称	データ型	初期値	コメント
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	ソケット
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	送信データ
	SktTCPAccept_instance	SktTCPAccept		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP_EtnOnlineSta	BOOL	☑	オンライン

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage:=INT#1;
    SktTCPAccept_instance(Execute:=FALSE);   // インスタンス初期化
    SktTCPSend_instance(                     // インスタンス初期化
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);          // ダミー
    SktTCPRcv_instance(                      // インスタンス初期化
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);           // ダミー
    SktClose_instance(Execute:=FALSE);       // インスタンス初期化
END_IF;

IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :                                  // アクセプト要求
        SktTCPAccept_instance(
            Execute   :=TRUE,
            SrcTcpPort:=UINT#6000,           // 自 TCP ポート番号
            TimeOut   :=UINT#0,              // タイムアウト時間
            Socket    =>WkSocket);           // ソケット

        IF (SktTCPAccept_instance.Done=TRUE) THEN
            Stage:=INT#2;                     // 正常終了
        ELSIF (SktTCPAccept_instance.Error=TRUE) THEN
            Stage:=INT#10;                    // 異常終了
        END_IF;
    2 :                                  // 受信要求
        SktTCPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,               // ソケット
            TimeOut:=UINT#0,                 // タイムアウト時間
            Size   :=UINT#2000,              // 受信データサイズ
```

```
            RcvDat :=RcvSocketDat[0]);          // 受信データ

        IF (SktTCPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;                       // 正常終了
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;                      // 異常終了
        END_IF;

    3 :                                 // 送信要求
        SendSocketDat:=RcvSocketDat;
        SktTCPSend_instance(
            Execute:=TRUE,
            Socket :=WkSocket,                  // ソケット
            SendDat:=SendSocketDat[0],          // 送信データ
            Size   :=UINT#2000);                // 送信データサイズ

        IF (SktTCPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;                       // 正常終了
        ELSIF (SktTCPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                      // 異常終了
        END_IF;

    4 :                                 // クローズ要求
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);                 // ソケット

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                       // 正常終了
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                      // 異常終了
        END_IF;

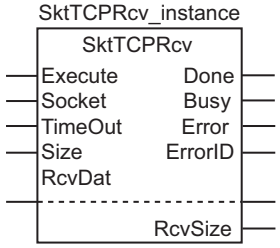
    0 :                                 // 正常終了
        DoTCP  :=FALSE;
        Trigger:=FALSE;

    ELSE                                // 異常による中断
        DoTCP  :=FALSE;
        Trigger:=FALSE;
    END_CASE;

END_IF;
```

SktTCPRcv

EtherNet/IP の、指定した TCP ソケットの受信バッファ内のデータを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPRcv	TCP ソケット受信	FB		SktTCPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID,RcvSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	−	−	−
TimeOut	タイムアウト時間		0: タイムアウトなし 1 ～ 65535: 0.1 ～ 6553.5s	データ型に従う	0.1s	0
Size	格納サイズ		受信バッファから読み出すデータサイズ	0 ～ 2000	バイト	1
RcvDat[] 配列	受信データ	入出力	受信データ	データ型に従う	−	−
RcvSize	受信データサイズ	出力	実際に RcvDat[]に格納したデータサイズ	0 ～ 2000	バイト	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1072)」参照																			
TimeOut							○													
Size							○													
RcvDat[] 配列		○																		
RcvSize							○													

機能

「Socket」で指定したソケットの、受信バッファ内のデータを、受信データ RcvDat[]に格納します。
格納するデータのサイズは、「Size」で指定します。
データ格納後に、実際に格納したデータサイズを、「RcvSize」に代入します。
もし、受信バッファにデータがない場合は、タイムアウト時間「TimeOut」に設定された時間だけ、受信データを待ちます。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、RcvDat[]へのデータ格納が完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"

*1. このメンバを、本命令で使用することはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 一度に読み出せるデータサイズは、最大 2000 バイトです。RcvDat[]の配列の大きさが 2000 バイトより大きくても、2000 バイトまでしか読み出せません。

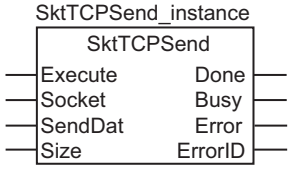
- 指定したソケットが受信しているデータサイズが「Size」の値より小さい場合、受信しているデータだけが RcvDat[]に格納されます。そして、格納したデータサイズを「RcvSize」に代入します。
- 指定したソケットが受信しているデータサイズが「Size」の値より大きい場合、「Size」分のデータだけが RcvDat[]に格納されます。
- 「Size」の値が 0 の場合、受信データの読出を行いません。
- 受信バッファにデータがなかった場合、ソケットを SktClose 命令でクローズすると、タイムアウト時間未満でも異常終了します。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「Socket」のメンバのいずれかが、有効範囲外の場合。
  c）「Socket」で指定したソケットが、受信処理中の場合。
  d）「Socket」で指定したソケットの、コネクションが確立していない場合。
  e）「Socket.Handle」で指定したハンドルが、存在しない場合。
  f）「TimeOut」で指定した時間だけ待っても、データを受信しなかった場合。
  g）ソケットを、SktClose 命令でクローズした場合。
  h）「Socket.Handle」で指定したハンドルが、すでにセキュアソケット通信に使用されている場合。

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktTCPSend

EtherNet/IP の、指定した TCP ポートから、データを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPSend	TCP ソケット送信	FB		SktTCPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	－	－	－
SendDat[] 配列	送信データ		送信データ	データ型に従う		
Size	送信データサイズ		送信データサイズ	0 ～ 2000	バイト	1

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1075)」参照																			
SendDat[] 配列		○																		
Size							○													

機能

「Socket」で指定したソケットから、送信データ SendDat[]を送信します。
送信するデータサイズは、「Size」で指定します。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. このメンバを、本命令で使用することはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 一度に送信できるデータサイズは、最大 2000 バイトです。SendDat[]の配列の大きさが 2000 バイトより大きくても、2000 バイトまでしか送信できません。
- 「Size」の値が 0 の場合、データ送信を行いません。

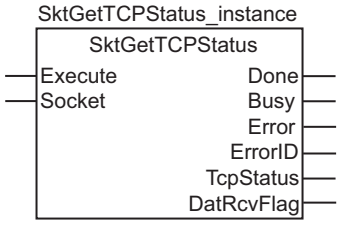
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「Socket」のメンバのいずれかが、有効範囲外の場合。
  c）「Socket」で指定したソケットが、送信処理中の場合。
  d）「Socket」で指定したソケットの、コネクションが確立していない場合。
  e）「Socket.Handle」で指定したハンドルが、存在しない場合。
  f）「Socket.Handle」で指定したハンドルが、すでにセキュアソケット通信に使用されている場合。

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktGetTCPStatus

TCP ソケットの、ステータスを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktGetTCPStatus	TCP ソケットのステータス読出	FB		SktGetTCPStatus_instance(Execute, Socket, Done, Busy, Error, ErrorID, TcpStatus, DatRcvFlag);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	－	－	－
TcpStatus	TCP コネクションステータス	出力	TCP コネクションステータス	*1	－	－
DatRcvFlag	受信データ有無フラグ		TRUE: 受信データあり FALSE: 受信データなし	データ型に従う		

*1. _CLOSED, _LISTEN, _SYN_SENT, _SYN_RECEIVED, _ESTABLISHED, _CLOSE_WAIT, _FIN_WAIT1, _CLOSING, _LAST_ACK, _FIN_WAIT2,_TIME_WAIT

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1078)」参照																			
TcpStatus	列挙型_eCONNECTION_STATE　列挙子は「機能 (P.2-1078)」参照																			
DatRcvFlag	○																			

機能

「Socket」で指定したソケットの、TCP コネクションステータス「TcpStatus」を取得します。
また、受信バッファに受信データが存在する場合は、受信データ有無フラグ「DatRcvFlag」の値を TRUE にします。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、「TcpStatus」と「DatRcvFlag」へのデータ格納が完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	データ送受信のハンドル	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"

*1. このメンバを、本命令で使用することはありません。

「TcpStatus」のデータ型は、列挙型_eCONNECTION_STATE です。
列挙子は、TCP の各状態を示します。それぞれの意味は、以下のとおりです。

列挙子	TCP の状態	意味
_CLOSED	CLOSED	コネクションがクローズされている。
_LISTEN	LISTEN	サーバ側が、パッシブオープンでコネクション確立要求(SYN)を待っている。
_SYN_SENT	SYN SENT	クライアント側が、アクティブオープンでコネクション確立要求(SYN)を送信し、それに対する確認応答(SYN+ACK)を待っている。
_SYN_RECEIVED	SYN RECEIVED	サーバ側が、コネクション確立要求(SYN)に対する確認応答(SYN+ACK)を送信し、それに対する確認応答(ACK)を待っている。
_ESTABLISHED	ESTABLISHED	コネクションが確立されている。
_CLOSE_WAIT	CLOSE WAIT	サーバ側が、コネクション切断要求(FIN)に対する確認応答(ACK)を送信し、サーバのアプリケーションが切断可能になるのを待っている。
_FIN_WAIT1	FIN WAIT-1	クライアント側が、コネクション切断要求(FIN)を送信し、それに対する確認応答(ACK)を待っている。
_CLOSING	CLOSING	クライアント側とサーバ側が、同時にコネクション切断要求(FIN)受信し、それに対する確認応答(ACK)を待っている。
_LAST_ACK	LAST-ACK	サーバ側が、コネクション切断要求(FIN)を送信し、それに対する確認応答(ACK)を待っている。
_FIN_WAIT2	FIN WAIT-2	クライアント側が、コネクション切断要求(FIN)を待っている。
_TIME_WAIT	TIME WAIT	クライアント側が、コネクション切断要求(FIN)に対する確認応答(ACK)を受信し、サーバに届いて処理されるのを待っている。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

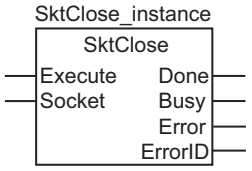
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「Socket」のメンバのいずれかが、有効範囲外の場合。
  b)「Socket.Handle」で指定したハンドルが、存在しない場合。

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktClose

EtherNet/IP の指定した、TCP ソケットまたは UDP ソケットを、クローズします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktClose	TCP/UDP ソケットクローズ	FB		SktClose_instance(Execute, Socket, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	－	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1081)」参照																			

機能

「Socket」で指定したソケットをクローズします。
TCP ソケットを指定した場合は、コネクション切断を行ってから、クローズします。
ソケットのハンドル「Socket.Handle」の値が 0 の場合は、ソケットサービス機能で使用しているすべての TCP/UDP ポートをクローズします。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、TCP/UDP ソケットのクローズが完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	—	—	—
Handle	ハンドル	クローズするコネクションのハンドル 0: ソケットサービス機能で使用しているすべてのTCP/UDP コネクションをクローズ	UDINT	データ型に従う	—	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	—	—	—
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	—	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. このメンバを、本命令で使用することはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。

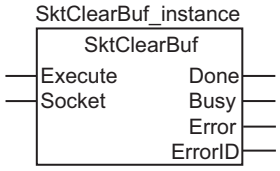
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- SktUDPRcv 命令または SktTCPRcv 命令を実行し、指定したハンドルのソケットで受信を待っている最中に本命令を実行した場合、受信待ちをキャンセルします。
- ひとつの自ポート番号で複数のコネクションを開設している場合、指定したソケットのコネクションだけをクローズします。
- ソケットのハンドル「Socket.Handle」の値が 0 の場合は、SktTCPAccept 命令でのコネクト待ちもキャンセルされます。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）自 IP アドレスの設定異常が、発生している場合。
  b）「Socket」のメンバのいずれかが、有効範囲外の場合。
  c）「Socket.Handle」で指定したハンドルが、存在しない場合。

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.2-1049)」、または SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktClearBuf

EtherNet/IP の指定した、TCP ソケットまたは UDP ソケットの、受信バッファをクリアします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktClearBuf	TCP/UDP ソケット受信バッファクリア	FB		SktClearBuf_instance(Execute, Socket, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	−	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1084)」参照																			

機能

「Socket」で指定したソケットの、受信バッファをクリアします。
命令が正常終了すると、「Done」の値が TRUE に変化します。
命令が正常終了した時点で、受信バッファのクリアが完了しています。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	−	−	−
Handle	ハンドル	受信バッファをクリアするソケットのハンドル	UDINT	データ型に従う	−	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	−	−	−
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	−	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	−	−	−
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	−	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		"

*1. このメンバを、本命令で使用することはありません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。

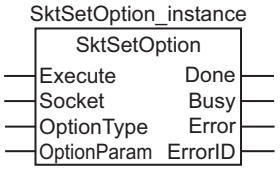
・つぎの場合、異常となります。「Error」は TRUE となります。
a)「Socket」のメンバのいずれかが、有効範囲外の場合。
b)「Socket」で指定したソケットが、存在しない場合。
c)「Socket.Handle」で指定したハンドルが、存在しない場合。
d)「Socket.Handle」で指定したハンドルが、すでにセキュアソケット通信に使用されている場合。

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.2-1066)」を、参照してください。

SktSetOption

EtherNet/IP の、指定した TCP ソケットのソケットオプションを設定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktSetOption	TCP ソケットオプション設定	FB		SktSetOption_instance(Execute, Socket, OptionType, OptionParam, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Socket	ソケット	入力	ソケット	–	–	–
OptionType	オプションタイプ		ソケットオプションのタイプ	–	–	–
OptionParam	オプションパラメータ		指定したソケットオプションのタイプに応じたパラメータの指定	–	–	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	構造体_sSOCKET　詳細は「機能 (P.2-1087)」参照																			
OptionType	列挙型_eSKT_OPTION_TYPE　列挙子は「機能 (P.2-1087)」参照																			
OptionParam	○*1																			

*1.　定数（リテラル）の入力はできません。変数を指定してください。

機能

「Socket」で指定したソケットの、ソケットオプションを指定します。
命令が正常終了すると、「Done」が TRUE に変化します。
命令が正常終了した時点で、ソケットオプションの設定が完了します。

「Socket」のデータ型は、構造体_sSOCKET です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Socket	ソケット	ソケット	_sSOCKET	－	－	－
Handle	ハンドル	受信バッファをクリアするソケットのハンドル	UDINT	データ型に従う	－	0
SrcAdr *1	自アドレス	自身の IP アドレス、ポート番号	_sSOCKET_ADDRESS	－	－	－
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	－	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''
DstAdr *1	相手アドレス	相手の IP アドレス、ポート番号	_sSOCKET_ADDRESS	－	－	－
PortNo *1	ポート番号	ポート番号	UINT	1 ～ 65535	－	0
IpAdr *1	IP アドレス	IP アドレスまたはホスト名。 ただし、ホスト名の場合は、DNS 設定または Hosts 設定が必須。	STRING	データ型に従う		''

*1. このメンバを、本命令で使用することはありません。

指定可能な「OptionType」の内容、および指定した「OptionType」で使用できる「OptionParam」のデータ型は、以下のとおりです。また、本命令を使用しなかった場合のデフォルト動作は、以下の初期値の動作となります。

OptionType		OptionParam		
列挙子	意味	使用可能なデータ型	値の意味	初期値
_TCP_NODELAY	TCP_NODELAY オプションを指定します。TCP ソケットだけに使用可能です。	BOOL	TRUE *1: TCP_NODELAY オプション有効 FALSE: TCP_NODELAY オプション無効	FALSE

*1. TRUE にすると Nagle アルゴリズムを無効にできます。データ量が少ない場合でもまとめて送信しません。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

ソケットサービス機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーになります。
- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、SktTCPAccept 命令、SktTCPConnect 命令によりソケットのハンドルをオープンしたあとに、SktTCPRcv 命令、SktTCPSend 命令、SktClearBuf 命令でデータ送受信を開始する前に使用することができます。データ送受信を開始したあとに本命令を使用した場合は、異常となります。
- ソケットオプションの設定は「Socket」で指定したハンドルごとに必要です。設定したソケットオプションは、ハンドルがオープンしている間、有効となります。SktClose 命令を実行してハンドルをクローズ後、再度 SktTCPAccept 命令、SktTCPConnect 命令を実行してハンドルをオープンし直した場合、本命令を実行してソケットオプションを設定し直してください。
- SktUDPCreate 命令、SktUDPRcv 命令、SktUDPSend 命令、SktTCPAccept 命令、SktTCPConnect 命令、SktTCPRcv 命令、SktTCPSend 命令、SktGetTCPStatus 命令、SktClose 命令、SktClearBuf 命令、SktSetOption 命令で同時実行可能な個数は、最大 32 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「Socket」のメンバのいずれかが、有効範囲外の場合。
  b)「OptionType」が対応していないデータ型を「OptionParam」に指定した場合。
  c) 指定したハンドルのソケットがすでに送受信を開始している場合。
  d) TCP_NODELAY を UDP ソケットに実行するなど、指定したハンドルのソケットが対応していないソケットタイプを指定した場合。
  e)「Socket.Handle」で指定したハンドルが、存在しない場合。

サンプルプログラム

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	DoTCP	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	ソケット
	SendSocketDat	ARRAY[0..1999] OF BYTE		送信データ
	Nodelay	BOOL	TRUE	NoDelay 設定
	SktTCPConnect_instance	SktTCPConnect		
	SktSetOption_instance	SktSetOption		
	SktTCPSend_instance	SktTCPSend		
	SktClose_instance	SktClose		

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_EIP_EtnOnlineSta=TRUE) ) THEN
        DoTCP:=TRUE;
        Nodelay:=TRUE;
        Stage:=INT#1;
        SktTCPConnect_instance(Execute:=FALSE); // インスタンス初期化
        SktSetOption_instance( // インスタンス初期化
            Execute:=FALSE,
            OptionType:=_TCP_NODELAY,
            OptionParam:= Nodelay);
        SktTCPSend_instance( // インスタンス初期化
            Execute:=FALSE,
            SendDat:=SendSocketDat[0]); // ダミー
        SktClose_instance(Execute:=FALSE); // インスタンス初期化
END_IF;

IF (DoTCP=TRUE) THEN
        CASE Stage OF
        1: // コネクト要求
            SktTCPConnect_instance(
                Execute:=TRUE,
                SrcTcpPort:=UINT#0, // 自 UDP ポート番号　自動割付
                DstAdr:='192.168.250.2', // 相手 IP アドレス
                DstTcpPort:=UINT#6000, // 相手 TCP ポート番号
                Socket =>WkSocket); // ソケット
            IF (SktTCPConnect_instance.Done=TRUE) THEN
                Stage:=INT#2; // 正常終了
            ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
                Stage:=INT#10; // 異常終了
            END_IF;

        2: // ソケットオプションの設定
            SktSetOption_instance(
                Execute:=TRUE,
                Socket:=WkSocket, // ソケット
                OptionType:=_TCP_NODELAY, // オプションタイプ
                OptionParam:= Nodelay); // NODELAY 有効
            IF (SktSetOption_instance.Done=TRUE) THEN
                Stage:=INT#3; // 正常終了
            ELSIF (SktSetOption_instance.Error=TRUE) THEN
                Stage:=INT#20; // 異常終了
            END_IF;

        3: // 送信要求
            SktTCPSend_instance(
                Execute:=TRUE,
                Socket:=WkSocket, // ソケット
                SendDat:=SendSocketDat[0], // 送信データ
                Size:=UINT#2000); // 送信データサイズ
            IF (SktTCPSend_instance.Done=TRUE) THEN
```

```
                Stage:=INT#4; // 正常終了
            ELSIF (SktTCPSend_instance.Error=TRUE) THEN
                Stage:=INT#30; // 異常終了
            END_IF;

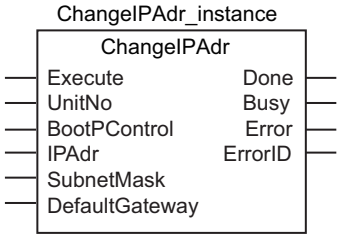
        4: // クローズ要求
            SktClose_instance(
                Execute:=TRUE,
                Socket:=WkSocket); // ソケット
            IF (SktClose_instance.Done=TRUE) THEN
                Stage:=INT#0; // 正常終了
            ELSIF (SktClose_instance.Error=TRUE) THEN
                Stage:=INT#40; // 異常終了
            END_IF;

        0: // 正常終了
            DoTCP:=FALSE;
            Trigger:=FALSE;

        ELSE // 異常による中断
            DoTCP:=FALSE;
            Trigger:=FALSE;
    END_CASE;
END_IF;
```

ChangeIPAdr

CPU ユニットの内蔵 EtherNet/IP ポート、または内部通信ポートの IP アドレスを変更します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ChangeIPAdr	IP アドレス変更	FB		ChangeIPAdr_instance(Execute, UnitNo, BootPControl, IPAdr, SubnetMask, DefaultGateway, Done, Busy, Error, ErrorID);

変数_ChangeIPAdr

	名称	入／出力	内容	有効範囲	単位	初期値
UnitNo	ユニット番号	入力	IP アドレスを変更するユニット番号	_CBU_CPU_Port1 *1, _CBU_CPU_InPort1	–	_CBU_No00
BootPControl	IP アドレス取得方法+設定タイミング		IP アドレスの取得方法と設定タイミング	0 ～ 2 *2		0
IPAdr[] 配列 *3	IP アドレス		IP アドレス	*4		–
SubnetMask[] 配列 *3	サブネットマスク		サブネットマスク			
DefaultGateway[] 配列 *3	デフォルトゲートウェイ		デフォルトゲートウェイ			

*1. _CBU_CPU_Port1 の代わりに、_CBU_CPU を指定することも可能です。
*2. 内部通信ポートを使用する場合は、「0」となります。
*3. 要素番号は 0～3、要素数は 4 です。
*4. 詳細は「機能_ChangeIPAdr (P.2-1093)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	列挙型_eUnitNo　列挙子は「機能_ChangeIPAdr (P.2-1093)」参照																			
BootPControl							○													
IPAdr[] 配列		○																		
	配列全体を指定																			
SubnetMask[] 配列		○																		
	配列全体を指定																			
DefaultGateway[] 配列		○																		
	配列全体を指定																			

機能_ChangeIPAdr

ユニット番号「UnitNo」で指定した、内蔵 EtherNet/IP ポート、または内部通信ポートの IP アドレスを、IP アドレス取得方法+設定タイミング「BootPControl」に従って変更します。

「UnitNo」で内蔵 EtherNet/IP ポートを指定した場合、本命令の実行が完了すると、いったんリンクオフになり、その後、変更した IP アドレスでリンクオンとなります。
「UnitNo」で内部通信ポートを指定した場合、本命令の実行が完了すると、システム変数「_EIPIn1_EtnOnlineSta」が FALSE になります。TRUE になると変更した IP アドレスで通信可能となります。

本命令を使用して、内蔵 EtherNet/IP ポート、または内部通信ポートの IP アドレスを、表示器などから変更することができます。

「UnitNo」のデータ型は、列挙型_eUnitNo です。列挙子の意味は、以下のとおりです。

列挙子	意味
_CBU_CPU	内蔵 EtherNet/IP ポート
_CBU_CPU_Port1 *1	内蔵 EtherNet/IP の通信ポート 1
_CBU_CPU_InPort1	内部通信ポート 1

*1. _CBU_CPU_Port1 の代わりに、_CBU_CPU を指定することも可能です。

「BootPControl」の値によって、新しい IP アドレスの取得方法と、それを設定するタイミングが以下のように決まります。
形 NX502 CPU ユニットのポート 1 を使用する場合、「BootPControl」の値は、「0～2, 4」が指定可能です。ポート 2 を使用する場合は、「0～4」が指定可能です。
「BootPControl」の値は、「0～2」が指定可能です。
NY シリーズコントローラの内部通信ポートを使用する場合は、「0」だけが指定可能です。

「BootPControl」の値	IP アドレスの取得方法	IP アドレスの設定タイミング
0	IP アドレス IPAdr[]、サブネットマスク SubnetMask[]、デフォルトゲートウェイ DefaultGateway[]の値が示す IP アドレスを取得	本命令実行時に 1 回だけ設定（固定設定）
1	BOOTP サーバから IP アドレスを取得	本命令実行時に一度設定し、その後はコントローラの電源を ON するたびに設定
2	BOOTP サーバから IP アドレスを取得	本命令実行時に 1 回だけ設定（固定設定）

IPAdr[]、SubnetMask[]、DefaultGateway[]の要素番号 0～3 には、IP アドレス、サブネットマスク、デフォルトゲートウェイの値を上位から順に設定します。たとえば、IP アドレスを 130.58.17.32 に変更する場合、IPAdr[0]=BYTE#16#82、IPAdr[1]=BYTE#16#3A、IPAdr[2]=BYTE#16#11、IPAdr[3]=BYTE#16#20 とします。

「UnitNo」で内蔵 EtherNet/IP ポート、または内部通信ポートを指定すると、IPAdr[]、SubnetMask[]、DefaultGateway[]の値の有効範囲は以下のようになります。これらの値の有効範囲は、「BootPControl」の値が「0」の場合にだけ有効です。

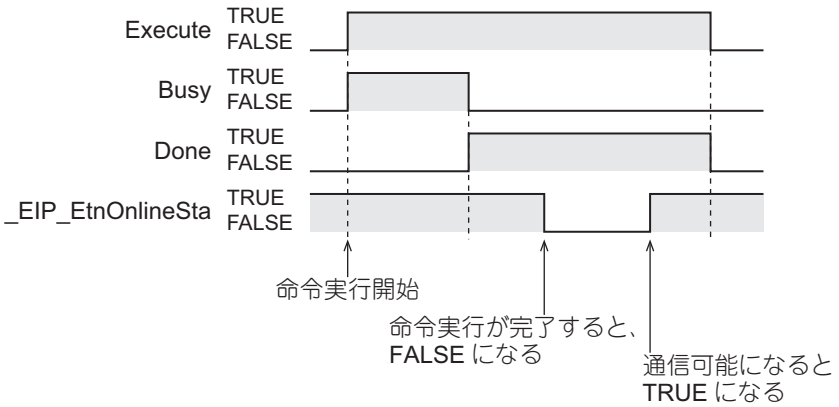
「UnitNo」の指定	入力変数	有効範囲
内蔵 EtherNet/IP ポート 内部通信ポート	IPAdr[]配列	以下に示す IP アドレス以外が有効 • 127. 0. 255 で始まる IP アドレス • ホスト ID の全ビットが 0 または全ビットが 1 の IP アドレス • クラス D の IP アドレス(224.0.0.0 ～ 239.255.255.255) • クラス E の IP アドレス(240.0.0.0 ～ 255.255.255.255) • AutoIP *1 用に予約された IP アドレス(169.254.0.0 ～ 169.254.255.255)
	SubnetMask[] 配列	192.0.0.0 ～ 255.255.255.252
	DefaultGateway[] 配列	以下に示す IP アドレス以外が有効 • 127. 0. 255 で始まる IP アドレス • 全ビットが 1 の IP アドレス • クラス D の IP アドレス(224.0.0.0 ～ 239.255.255.255) • クラス E の IP アドレス(240.0.0.0 ～ 255.255.255.255) • AutoIP *1 用に予約された IP アドレス(169.254.0.0 ～ 169.254.255.255)

*1. AutoIP とは、Windows 98 以降の OS が搭載する IP アドレス自動割り当て機能のことです。

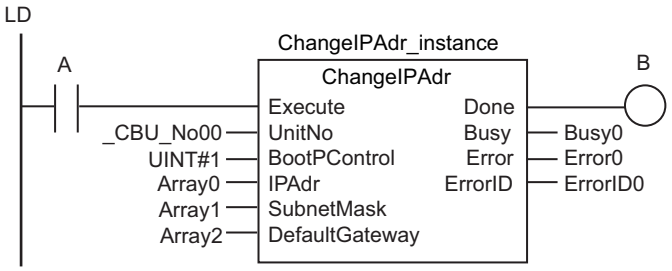
「BootPControl」の値が 1 または 2 の場合は、IPAdr[]、SubnetMask[]、DefaultGateway[]の値は無視します。IPAdr[]、SubnetMask[]、DefaultGateway[]の値が、有効範囲外でもかまいません。

「UnitNo」で内蔵 EtherNet/IP ポートを指定した場合は、システム定義変数「_EIP_EtnOnlineSta」、「_EIP1_EtnOnlineSta」、「_EIPln1_EtnOnlineSta」により、通信可能であるかどうかを確認できます。
ここでは、「_EIP_EtnOnlineSta」を例にして説明しますが、「_EIP1_EtnOnlineSta」、「_EIPln1_EtnOnlineSta」についても同様になります。

「Busy」が TRUE から FALSE になると、「_EIP_EtnOnlineSta」は FALSE になります。その後、新しい IP アドレスで通信可能になると、「_EIP_EtnOnlineSta」は FALSE から TRUE になります。



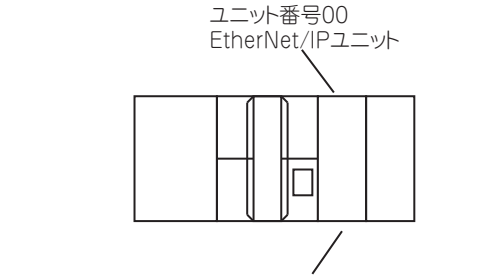
ユニット番号 00 の EtherNet/IP ユニットの IP アドレスを、BOOTP サーバから都度取得して変更する場合の記述例を以下に示します。
「Execute」=A を表示器などから ON にすることで、BOOTP サーバから取得した IP アドレスに変更します。その後は、電源を ON するたびに BOOTP サーバから IP アドレスを取得し、変更します。



ST

```
ChangeIPAdr_instance(A,_CBU_No00,UINT#1,Array0,Array1,Array2,B,Busy0,Error0,ErrorID0);
```

ユニット番号00のEtherNet/IPユニットに、BOOTPサーバから取得したIPアドレスを設定します。
その後は、電源をONするたびにBOOTPサーバから取得したIPアドレスを設定し直します。



関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

- 「UnitNo」で内蔵 EtherNet/IP ポートを指定した場合、本命令を実行すると、以下の項目がイベントログに記録されます。
  a）リンクオフ検知
  b）IP アドレス決定
- 「UnitNo」で内部通信ポートを指定した場合、本命令を実行すると、以下の項目がイベントログに記録されます。
  a）IP アドレス決定
- CPU ユニットの書込プロテクトが有効である場合も、本命令で IP アドレスを変更できます。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「UnitNo」で内蔵 EtherNet/IP ポートを指定した場合、本命令の実行が完了すると、内蔵 EtherNet/IP ポートはいったん通信不可になります。リンクオフになっても設備に支障がないことを確認してから、本命令を実行してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 本命令実行中（「Busy」の値が TRUE）にコントローラの電源を OFF すると、次回電源投入時に全停止フォールトが発生する場合があります。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	異常名称	異常内容
16#0400	入力値範囲外	入力変数のいずれかの値が、有効範囲外
16#2400	実行権限なし	変更ができない状態のときに命令を実行しました。 • 設定を変更中 • 内蔵 EtherNet/IP ポートがリスタート中 • NetworkConfigrator からタグデータリンク設定をダウンロード中
16#2402	命令同時実行数オーバー	ChangeIPAdr 命令、ChangeXBUnitIPAdr 命令、ChangeFTPAccount 命令、ChangeNTPServerAdr 命令の同時実行数がオーバーしています。
16#240D	IP アドレス設定不正	指定したポートと別のポートのネットワークアドレスが同じです。

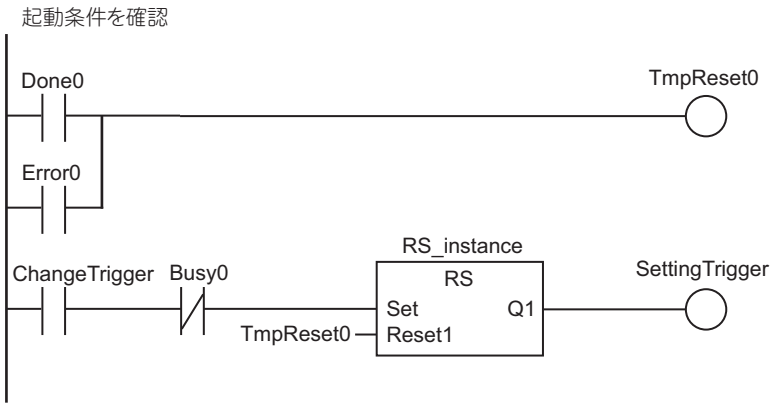
サンプルプログラム

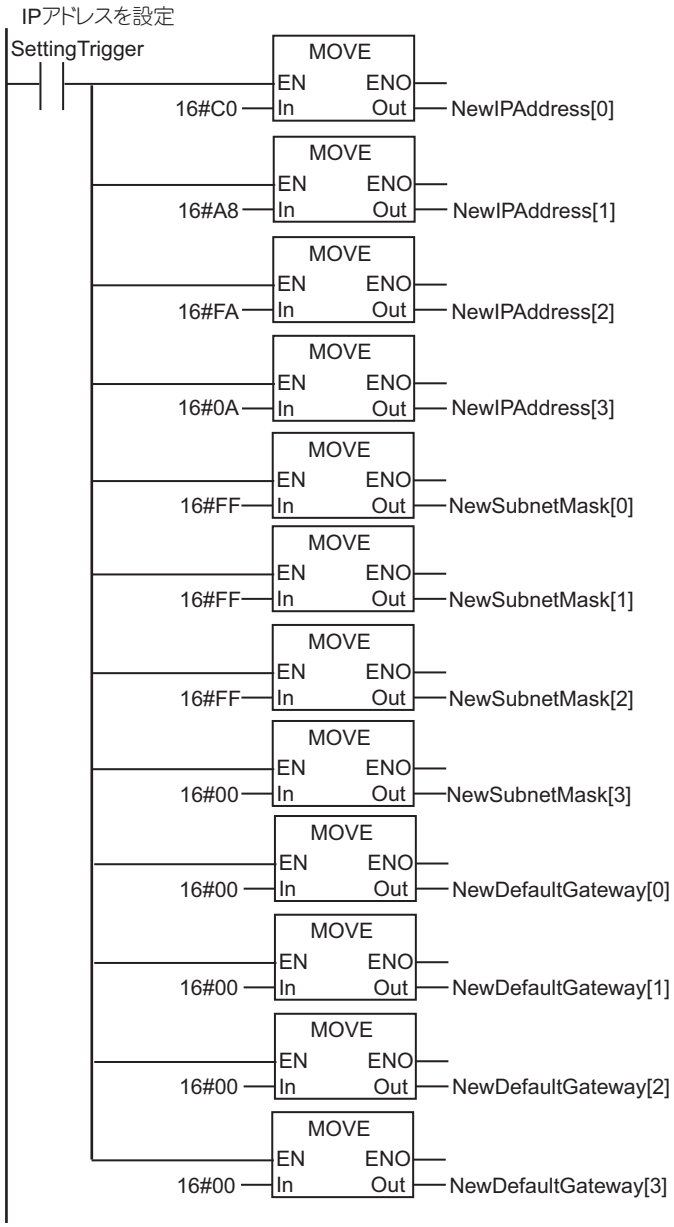
内蔵 EtherNet/IP ポートの IP アドレスを、以下の IP アドレスの固定設定に変更します。

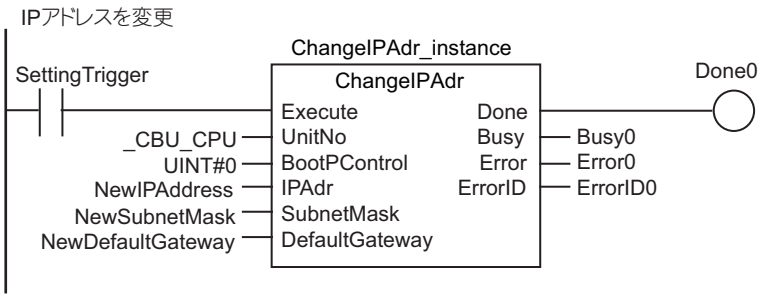
項目	内容
IP アドレス	192.168.250.10
サブネットマスク	255.255.255.0
デフォルトゲートウェイ	0.0.0.0

LD

名称	データ型	初期値	コメント
ChangeTrigger	BOOL	False	変更フラグ
SettingTrigger	BOOL	False	IP アドレス変更中フラグ
Done0	BOOL	False	IP アドレス変更完了
Error0	BOOL	False	IP アドレス変更異常
Busy0	BOOL	False	IP アドレス変更中
ErrorID0	WORD	16#0	IP アドレス変更異常 ID
TmpReset0	BOOL	False	テンポラリ変数
NewIPAddress	ARRAY[0..3] OF BYTE	[4(16#0)]	IP アドレス
NewSubnetMask	ARRAY[0..3] OF BYTE	[4(16#0)]	サブネットマスク
NewDefaultGateway	ARRAY[0..3] OF BYTE	[4(16#0)]	デフォルトゲートウェイ
RS_instance	RS		
ChangeIPAdr_instance	ChangeIPAdr		


起動条件を確認
Done0
TmpReset0
Error0
RS_instance
RS
ChangeTrigger
Busy0
Set
Q1
SettingTrigger
TmpReset0
Reset1


IPアドレスを設定
SettingTrigger
MOVE
EN
ENO
16#C0
In
Out
NewIPAddress[0]
16#A8
NewIPAddress[1]
16#FA
NewIPAddress[2]
16#0A
NewIPAddress[3]
16#FF
NewSubnetMask[0]
NewSubnetMask[1]
NewSubnetMask[2]
16#00
NewSubnetMask[3]
NewDefaultGateway[0]
NewDefaultGateway[1]
NewDefaultGateway[2]
NewDefaultGateway[3]



ST

名称	データ型	初期値	コメント
ChangeTrigger	BOOL	False	変更フラグ
SettingTrigger	BOOL	False	IP アドレス変更中フラグ
Done0	BOOL	False	IP アドレス変更完了
Error0	BOOL	False	IP アドレス変更異常
Busy0	BOOL	False	IP アドレス変更中
ErrorID0	WORD	16#0	IP アドレス変更異常 ID
NewIPAddress	ARRAY[0..3] OF BYTE	[4(16#0)]	IP アドレス
NewSubnetMask	ARRAY[0..3] OF BYTE	[4(16#0)]	サブネットマスク
NewDefaultGateway	ARRAY[0..3] OF BYTE	[4(16#0)]	デフォルトゲートウェイ
RS_instance	RS		
ChangeIPAdr_instance	ChangeIPAdr		

```
//起動条件を確認
IF((ChangeTrigger=TRUE)AND(Busy0=FALSE))THEN
   SettingTrigger:= TRUE;
END_IF;

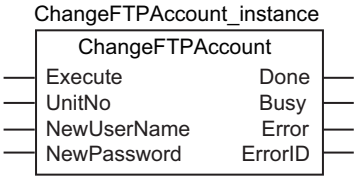
IF((Done0=TRUE)OR(Error0=TRUE))THEN
   SettingTrigger:= FALSE;
END_IF;

//IP アドレスを設定
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;
```

```
//IP アドレスを変更
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

CPU ユニットの EtherNet/IP ポートの FTP のログイン名とパスワードを変更します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ChangeFTPAccount	FTP アカウント変更	FB		ChangeFTPAccount_instance(Execute, UnitNo, NewUserName, NewPassword, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
UnitNo	ユニット番号	入力	FTP のログイン名とパスワードを変更するユニット番号	_CBU_CPU	－	_CBU_No00
NewUserName	ログイン名		ログイン名	半角英数字 1 ～ 12 文字（大文字小文字の区別あり）		－
NewPassword	パスワード		パスワード	*1		

*1. 詳細は「「NewPassword」の有効範囲 (P.2-1101)」を参照してください。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	列挙型_eUnitNo　列挙子は「機能 (P.2-1100)」参照																			
NewUserName																				○
NewPassword																				○

機能

ユニット番号「UnitNo」で指定した、内蔵 EtherNet/IP ポートの FTP ログイン名とパスワードを、ログイン名「NewUserName」とパスワード「NewPassword」で指定した値に変更します。
「Execute」が FALSE から TRUE に変化したときに、「NewUserName」、「NewPassword」の値を内蔵 EtherNet/IP ポートの FTP ログイン名とパスワードとして書き込みます。
実行中は「Busy」が TRUE になり、設定変更要求の受け付け完了時に「Done」が TRUE となります。
「Done」が TRUE になった時点では設定は反映されていません。

本命令を使用して、内蔵 EtherNet/IP ポートの FTP のログイン名とパスワードを、表示器などから変更することができます。

「UnitNo」のデータ型は、列挙型_eUnitNo です。列挙子の意味は、以下のとおりです。

列挙子	意味
_CBU_CPU	内蔵 EtherNet/IP ポート
_CBU_No00～_CBU_No15 *1	EtherNet/IP ユニットのユニット番号 00～15

*1. NJ シリーズ CPU ユニットの場合だけ設定ができます。

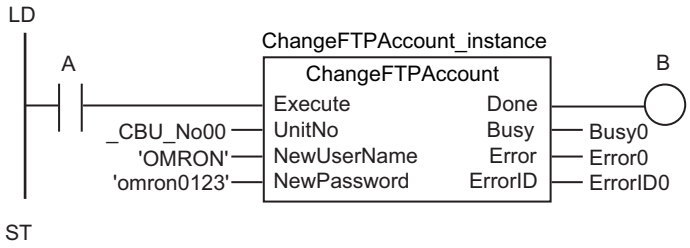
「NewPassword」の有効範囲

「NewPassword」の値の有効範囲は、以下のようになります。

「UnitNo」の指定	有効範囲
内蔵 EtherNet/IP ポート	半角英数字 8～32 文字（大文字小文字の区別あり）

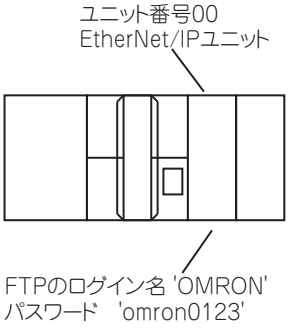
記述例

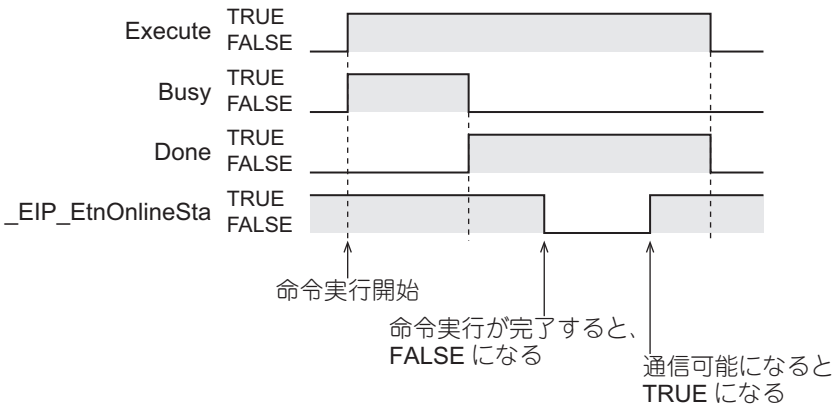
ユニット番号 00 の EtherNet/IP ユニットの、FTP のログイン名とパスワードを変更する場合の記述例を以下に示します。
「Execute」=A を表示器などから ON にすることで、FTP のログイン名を'OMRON'に、パスワードを'omron0123'に変更します。




```
ChangeFTPAccount_instance(A,_CBU_No00,'OMRON','omron0123',B,Busy0,Error0,ErrorID0);
```

ユニット番号00のEtherNet/IPユニットの、FTPのログイン名を'OMRON'に、
パスワードを'omron0123'に変更します。





関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

参考

- 本命令を使用すると、CPU ユニットの書込プロテクトが有効でも、FTP のログイン名とパスワードが変更できます。
- FTP サーバに接続し、ファイルを転送しているときに、本命令により FTP のログイン名とパスワードを変更しても、ファイルの転送は続行します。
- FTP でログインしている状態で、本命令により設定を変更した場合、すでに確立している FTP のセッションは、本命令で設定を変更したあとも継続して維持されます。

使用上の注意

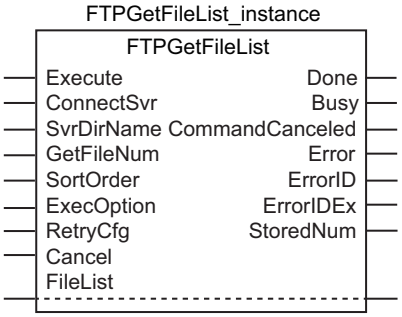
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、イベントタスク上で使うことができません。コンパイル時にエラーとなります。
- 本命令実行中（「Busy」の値が TRUE）にコントローラの電源を OFF すると、次回電源投入時に全停止フォールトが発生する場合があります。
- 異常が発生した場合、「Error」は TRUE となります。「ErrorID」の値と、それに対応する異常内容は、以下のとおりです。

「ErrorID」の値	異常名称	異常内容
16#0400	入力値範囲外	• 入力変数のいずれかの値が、有効範囲外 • 入力変数のいずれかの値が、不正

「ErrorID」の値	異常名称	異常内容
16#2400	実行権限なし	変更ができない状態のときに命令を実行しました。 • 設定を変更中 • 内蔵 EtherNet/IP ポートがリスタート中 • NetworkConfigrator からタグデータリンク設定をダウンロード中
16#2402	命令同時実行数オーバー	ChangeIPAdr 命令、ChangeXBUnitIPAdr 命令、ChangeFTPAccount 命令、ChangeNTPServerAdr 命令の同時実行数がオーバーしています。

FTPGetFileList

FTP サーバ内のファイルのリストを取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPGetFileList	FTP サーバのファイルリスト取得	FB		FTPGetFileList_instance(Execute, ConnectSvr, SvrDirName, GetFileNum, SortOrder, ExecOption, RetryCfg, Cancel, FileList, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, StoredNum);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	入力	接続先 FTP サーバの各種設定パラメータ	–	–	*1
SvrDirName	FTP サーバディレクトリ名		ファイルリストを取得する FTP サーバのディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*2*3		" *4
GetFileNum	取得ファイル数		リストを取得するファイルの数	1 ～ 1000		1
SortOrder *5	並び替え順序		ファイルリストの並び順	_NAME_ASC, _NAME_DESC, _DATE_ASC, _DATE_DESC		_NAME_ASC
ExecOption	FTP 実行オプション		FTP 実行に関するオプション	–		–
RetryCfg	実行リトライ設定		命令実行リトライ設定			
Cancel	取り消し		TRUE: 命令実行を取り消す FALSE: 命令実行を取り消さない	データ型に従う		FALSE
FileList[] 配列 *6 *7 *8	ファイル詳細	入出力	取得したファイルの詳細情報	–	–	*1
CommandCanceled	取り消し完了	出力	TRUE: 取り消し完了 FALSE: 取り消し未完了	データ型に従う	–	–
StoredNum	取得済みファイル数		詳細情報を取得したファイルの数	0 ～ 1000		

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. FTP サーバディレクトリ名として、以下の文字は使用できません。
"*"、"?"、"<"、">"、"|"、"""（ダブルクォーテーション）
*3. 半角スペースは、FTP サーバにより動作が変わることがありますので使用を推奨しません。
半角スペースを使用する場合は、2 文字として扱います。

*4. FTP サーバの、ログイン時のホームディレクトリです。
*5. FTP サーバが並び替え順序をサポートしていない場合、並び替え順序は、「SortOrder」の値に関係なく名前昇順になります。
*6. 配列の最大要素数は、1000 です。
*7. 1 次元の配列です。2 次元以上の配列を指定した場合は、ビルド時に異常になります。
*8. 先頭の要素番号は 0 です。先頭の要素番号が 0 以外の配列を指定した場合は、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	構造体_sFTP_CONNECT_SVR　詳細は「機能 (P.2-1105)」参照																			
SvrDirName																				○
GetFileNum							○													
SortOrder	列挙型_eFILE_SORT_ORDER　列挙子は「機能 (P.2-1105)」参照																			
ExecOption	構造体_sFTP_EXEC_OPTION　詳細は「FTP サーバとの処理に関するオプション指定 (P.2-1106)」参照																			
RetryCfg	構造体_sFTP_RETRY_CFG　詳細は「FTP サーバとの接続処理のリトライ指定 (P.2-1106)」参照																			
Cancel	○																			
FileList[] 配列	構造体_sFTP_FILE_DETAIL　詳細は「機能 (P.2-1105)」参照																			
CommandCanceled	○																			
StoredNum							○													

機能

接続先 FTP サーバ「ConnectSvr」の指定ディレクトリ「SvrDirName」に存在する複数のファイルのリストを、各ファイルの詳細情報とともに取得します。
リストを取得するファイルの数は、取得ファイル数「GetFileNum」で指定します。取得したファイル情報の並び順は、並び替え順序「SortOrder」で指定します。

「ConnectSvr」のデータ型は、構造体_sFTP_CONNECT_SVR です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	接続先 FTP サーバの各種設定パラメータ	_sFTP_CONNECT_SVR	–	–	–
Adr	アドレス	IP アドレスまたは、ホスト名 *1	STRING	1 ～ 200 バイト *2	–	–
PortNo	ポート番号	FTP サーバのコントロールコネクションの TCP ポート番号	UINT	0 ～ 65535 *3		
UserName	ユーザ名	FTP サーバのユーザ名	STRING	最大 33 バイト *4 *5		
Password	パスワード	FTP サーバのパスワード	STRING	最大 33 バイト *4 *5		

*1. ホスト名を指定時は、別途 DNS 設定または Hosts 設定が必要です。
*2. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（アンダーバー）です。
*3. 0 の場合、TCP ポート番号は 21 になります。
*4. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（ アンダーバー）です。CPU ユニット Ver.1.16 以降では、“\”（バックスラッシュ）、“@”の入力も可能です。
*5. バイト数には、終端 NULL 文字を含みます。

「SortOrder」のデータ型は、列挙型_eFILE_SORT_ORDER です。列挙子の意味は、以下のとおりです。

列挙子	意味
_NAME_ASC	名前昇順
_NAME_DESC	名前降順
_DATE_ASC	更新日時昇順
_DATE_DESC	更新日時降順

各ファイルの詳細情報は、ファイル詳細 FileList[]に格納されます。取得したファイルの数は、取得済みファイル数「StoredNum」に格納されます。

FileList[]のデータ型は、構造体_sFTP_FILE_DETAIL です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
FileList	ファイル詳細	取得したファイルの詳細情報	_sFTP_FILE_DETAIL	–	–	–
Name	フォルダ名・ファイル名	フォルダ名またはファイル名	STRING	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）	–	–
ModifiedDate	更新日時	フォルダまたはファイルの更新日時	DATE_AND_TIME	–	–	–
Size	ファイルサイズ	ファイルのサイズ *1	ULINT	–	バイト	–
ReadOnly	属性-読取専用	フォルダおよびファイルの読取専用属性 TRUE: 読取専用である FALSE: 読取専用ではない	BOOL	データ型に従う	–	–
Folder	フォルダ	TRUE: フォルダである FALSE: フォルダではない	BOOL	データ型に従う	–	–

*1. フォルダの場合は 0 を格納します。

FTP サーバとの処理に関するオプション指定

FTP サーバからファイルリストを取得する際に、FTP 実行オプション「ExecOption」で指定した動作を行います。
オプション設定内容は、FTPGetFile 命令と同じ仕様です。「FTPGetFile (P.2-1117)」命令を参照してください。
ただし、本命令で有効なオプションは、パッシブモード指定「ExecOption.PassiveMode」だけです。

FTP サーバとの接続処理のリトライ指定

FTP サーバとの接続処理は、指定したタイムアウト時間「RetryCfg.TimeOut」内に成功しない場合、タイムアウトとして終了します。また、FTP サーバから接続を拒否された場合は、タイムアウト時間を待たずに終了します。
接続失敗の後、指定したリトライ間隔「RetryInterval」秒後に、接続処理をリトライします。
指定したリトライ回数「RetryCfg.RetryNum」のリトライで、FTP サーバと接続できなかった場合、命令は実行エラーとなります。

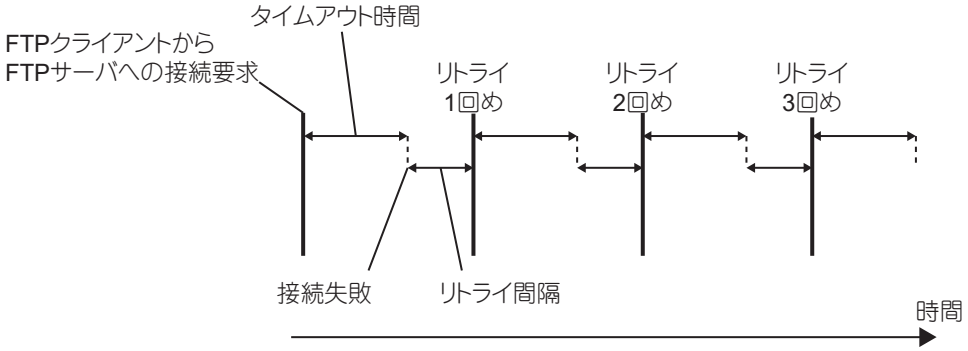
FTP サーバとの接続処理には成功した後、ネットワークにおける障害などの原因でファイル転送が中断した場合、「RetryCfg.TimeOut」で指定した時間でタイムアウトし、リトライは行いません。

「RetryCfg」のデータ型は、構造体_sFTP_RETRY_CFG です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
RetryCfg	実行リトライ設定	命令実行リトライ設定	_sFTP_RETRY_CFG	−	−	−
TimeOut	タイムアウト時間	FTP サーバへの接続タイムアウト時間	UINT	0 ～ 60 *1	秒	20
RetryNum	リトライ回数	接続失敗時のリトライ回数	UINT	0 ～ 3	回	0
RetryInterval	リトライ間隔	接続失敗時のリトライ間隔	UINT	0 ～ 65535 *2	秒	1

*1. 「0」を設定した場合は、タイムアウト時間が 20 秒になります。
*2. 「0」を設定した場合は、リトライ間隔が 1 秒になります。

以下に、FTP クライアントから FTP サーバへの接続処理におけるタイムアウト時間、リトライ回数、リトライ間隔の関係を示します。

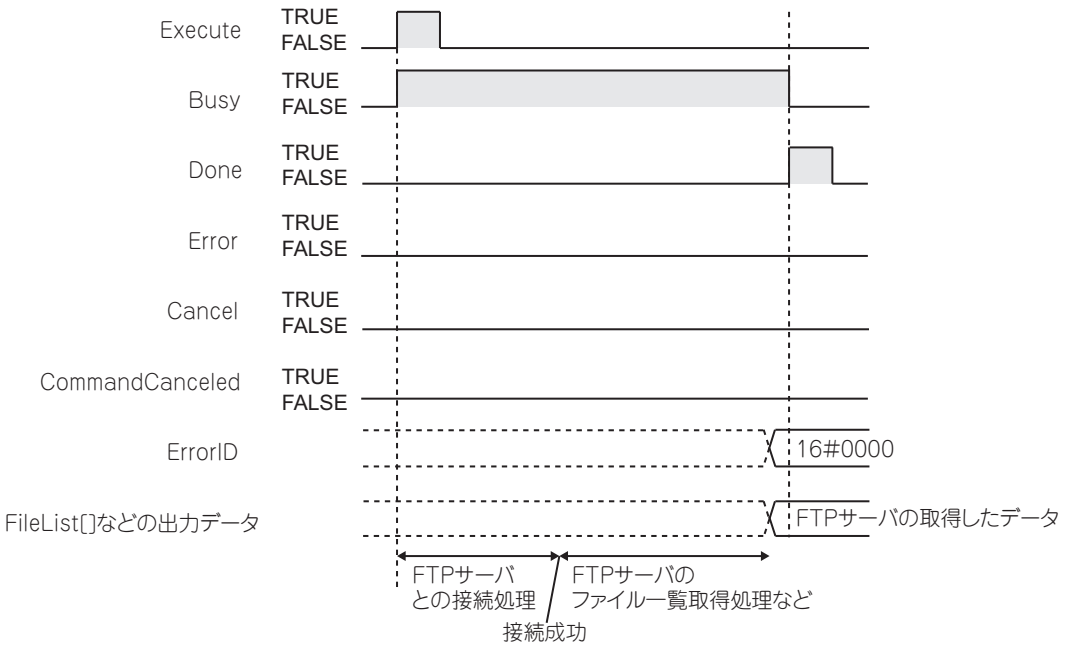


● FTP サーバとの接続に成功した場合

FTP サーバとの接続処理に成功した場合、FTP サーバからファイルリストを取得した後、以下の処理が行われます。

- 「ErrorID」に 16#0000 が格納されます。
- FileList[]などの出力データに取得データが格納されます。
- 正常終了「Done」の値が、TRUE になります。

以下に、FTP サーバとの接続処理に成功した場合のタイミングチャートの例を示します。

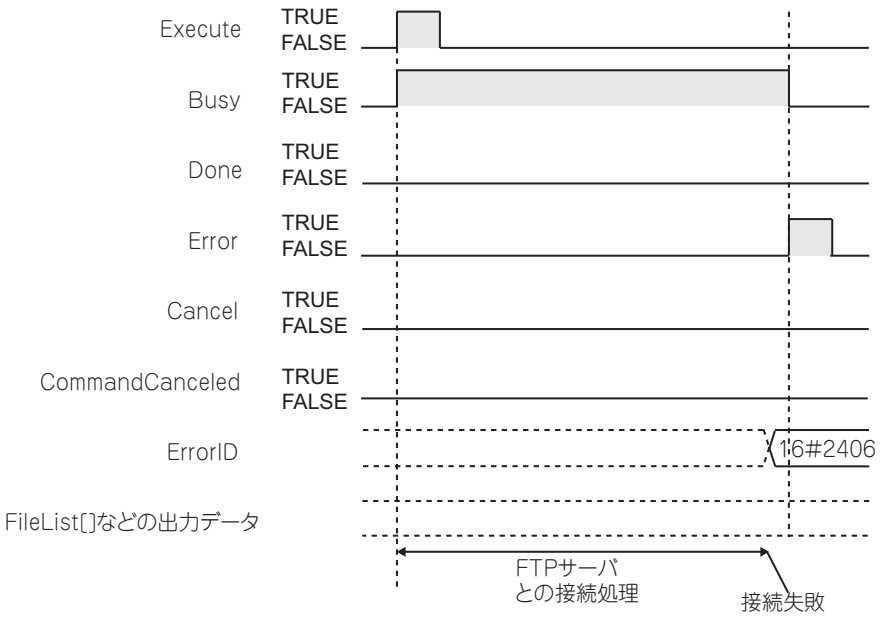


● FTP サーバとの接続に失敗した場合

FTP サーバとの接続処理に失敗した場合、以下の処理が行われます。

- 「ErrorID」にエラーコードが格納されます。
- 異常終了「Error」の値が、TRUE になります。

以下に、FTP サーバとの接続処理が異常終了した場合のタイミングチャートの例を示します。

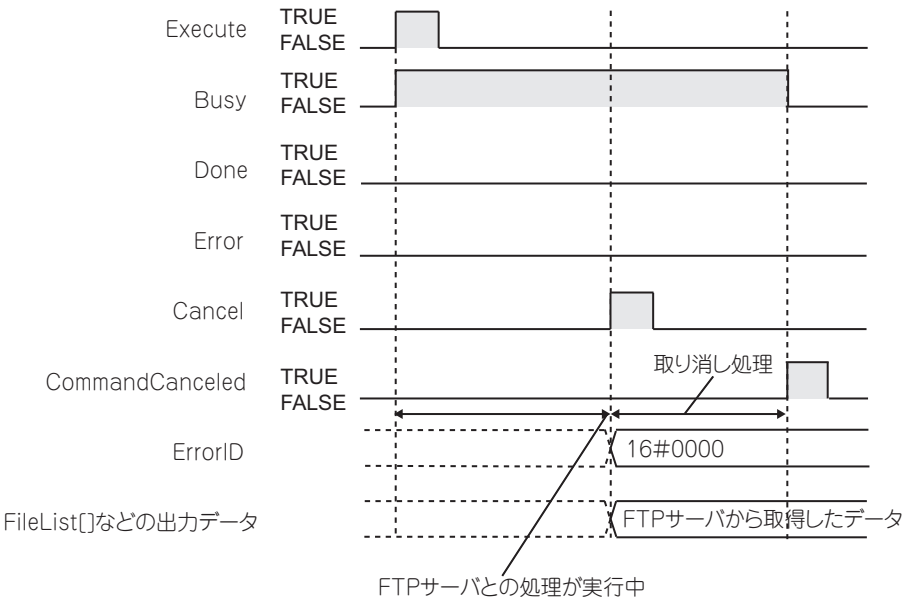


命令実行の取り消し

命令実行中に、取り消し「Cancel」を TRUE にすることで、FTP サーバとの処理を強制終了します。ファイルリストの取得や FTP サーバへの接続処理に時間がかかって、処理を終了させたい場合に使用します。

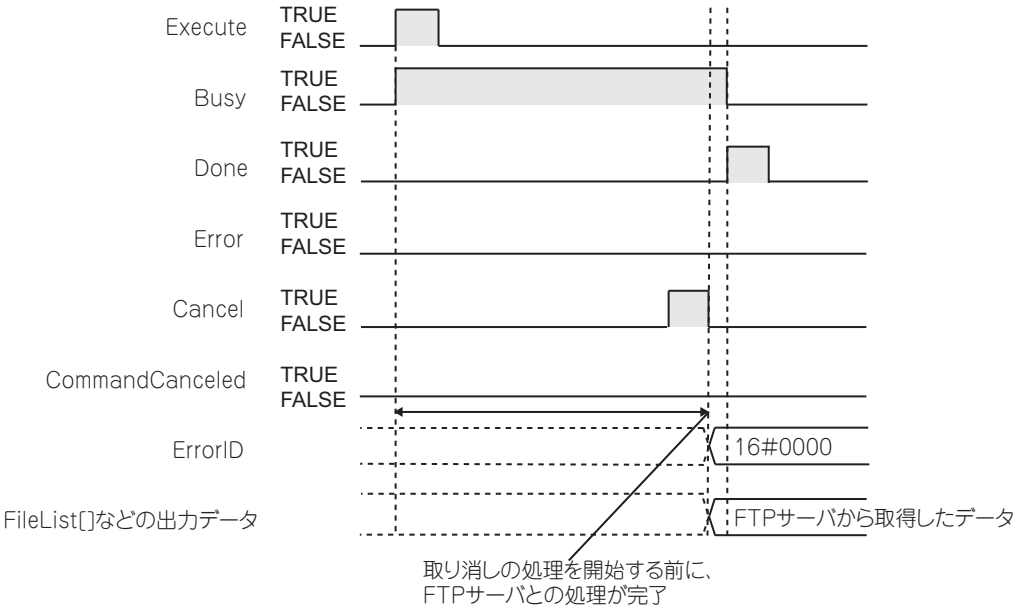
● FTP サーバとの処理が実行中に取り消し「Cancel」を TRUE にした場合

FTPGetFileList 命令で、FTP サーバからファイルリストを取得中に、取り消し「Cancel」を TRUE にすると、以下のようになります。
ファイル詳細 FileList[]には、FTP サーバからファイル詳細情報を取得できたものだけを格納します。
取得ファイル数「StoredNum」には、正常にファイル詳細情報を取得できたファイルの数を格納します。
正常終了「Done」は TRUE になりません。
取り消しが、正常終了したかどうかは「CommandCanceled」の値が TRUE になることで確認してください。



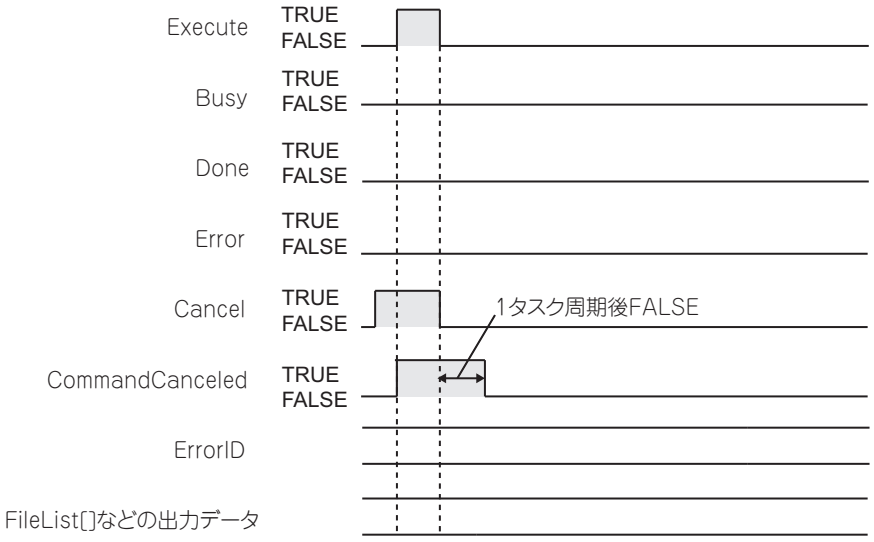
● 取り消しの処理を開始する前に、FTP サーバとの処理が完了した場合

取り消し「Cancel」が TRUE になっても、取り消しの処理を開始する前に、FTP サーバとの処理が完了した場合は、正常終了「Done」を通知します。また、「CommandCanceled」の値が TRUE になりません。



● 取り消し「Cancel」と起動「Execute」の両方を TRUE にした場合

取り消し「Cancel」と起動「Execute」の両方が TRUE になった場合は、取り消しを優先し、FTP サーバとの処理は行いません。「CommandCanceled」が TRUE になります。



関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

使用上の注意

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 入力変数「SvrDirName」で指定したディレクトリにファイルまたはサブディレクトリがない場合は、正常終了とし「Done」が TRUE となります。「StoredNum」には 0 を格納し、FileList[]には何も格納しません。
- 入力変数「GetFileNum」で指定した取得ファイル数より FileList[]の配列要素数が小さい場合、ファイル情報は FileList[]の配列要素数分だけ格納し、超えた分の情報を格納しません。この場合「Error」は TRUE になりません。
- 255 文字を超えるファイル名については先頭から 255 文字分を FileList[]の「Name」に格納します。この場合「Error」は TRUE になりません。
- FTP サーバの仕様によっては、指定した一部またはすべてのファイルの詳細情報が取得できない場合があります。詳細情報が取得できなかったファイルは、FileList[]のメンバの値が以下のようになります。また、この場合、「Error」の値は FALSE になります。

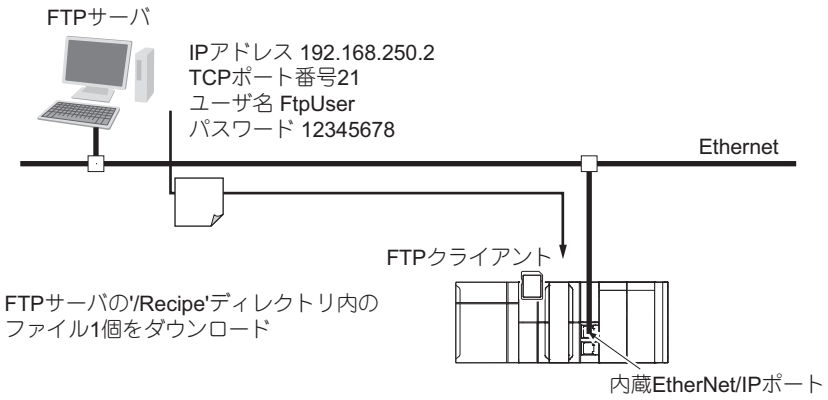
メンバ	値
ModifiedDate	DT#1970-01-01-00:00:00.000000000
Size	0
ReadOnly	FALSE
Folder	FALSE

- FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令で同時実行可能な個数は、最大 3 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）入力パラメータのいずれかがの値が、有効範囲外の場合
  b）「SvrDirName」に、1 階層上のディレクトリ“..”を指定した場合
  c）「SvrDirName」に、“//”などの不正なパスを指定した場合
  d）「SvrDirName」で指定したディレクトリが FTP サーバに存在しない場合
  e）「ConnectSvr」で指定した FTP サーバがネットワーク上に存在しない、またはサービスを停止している場合
  f）FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令を、同時に 3 個を超えて実行した場合
  g）FTP サーバとの接続処理中に、ネットワークにおける障害によりファイル転送処理が中断した場合
- 拡張エラーコード「ErrorIDEx」は、本命令では、FTP サーバより返信された FTP レスポンスコードを表示します。代表的な「ErrorIDEx」の値、異常内容および対処方法は、以下のとおりです。詳細については、FTP サーバの仕様を確認してください。「ErrorIDEx」は、エラーコード「ErrorID」の値が WORD#16#2407 の場合に出力されます。

「ErrorIDEx」の値	異常内容	対処方法
16#000001A9	データコネクションを確立できませんでした。	インターネット上の FTP サーバとの FTP 通信の場合、FTP のオープンモードが Active になっていないか確認してください。
16#000001AA	コネクションがクローズされています。データ転送を中止しました。	FTP サーバとの接続を確認してください。FTP サーバが停止していないか確認してください。
16#000001C2	要求されたファイル操作を行えませんでした。ファイルが、すでにオープンされている場合など、ファイルが使用できません。	対象のファイルが他アプリケーションによりオープンされていないか確認してください。
16#00000212	ユーザはログインできませんでした。	FTP ユーザ名またはパスワードを確認してください。
16#00000214	ファイル保存用のアカウントが必要です。	FTP ユーザのアクセス権を確認してください。
16#00000226	ファイルが見つからないためにアクセスできない場合など、ファイルが使用できないため、要求されたファイル操作を実行しませんでした。	指定したファイル名に該当するファイルが FTP サーバのディレクトリに存在するか確認してください。 指定したファイルのアクセス権を確認してください。
16#00000229	ファイル名が不正なため実行できませんでした。	指定したディレクトリのアクセス権を確認してください。

サンプルプログラム

FTP サーバの'/Recipe'ディレクトリ内にあるファイルを 1 個ダウンロードして、SD メモリカードのルートディレクトリに格納します。
ダウンロードするファイルは、FTP サーバの'/Recipe'ディレクトリ内のファイルを名前昇順に並べたうちの、最後のファイルです。



コントローラと FTP サーバは、EtherNet/IP ネットワークで接続しています。FTP サーバと接続するための設定パラメータの値は、以下のとおりとします。

設定パラメータ	値
IP アドレス	192.168.250.2
TCP ポート番号	21
ユーザ名	FtpUser
パスワード	12345678

処理の手順は、以下のようになります。

1 **FTPGetFileList 命令を使用して、FTP サーバのファイルリストを取得します。**
FTP サーバのディレクトリ名、取得ファイル数、並び替え順序、ファイル詳細情報を格納する変数は、以下のとおりです。

設定項目	仕様
FTP サーバのディレクトリ名	'/Recipe'
取得ファイル数	1000
並び替え順序	名前昇順
ファイル詳細情報を格納する変数	FTPFileList[]

2 **FTPGetFile 命令を使用して、1 で取得したファイルリストの中から、名前昇順に並べたうちの最後のファイルをダウンロードします。**
ファイルの格納先は、SD メモリカードのルートディレクトリです。

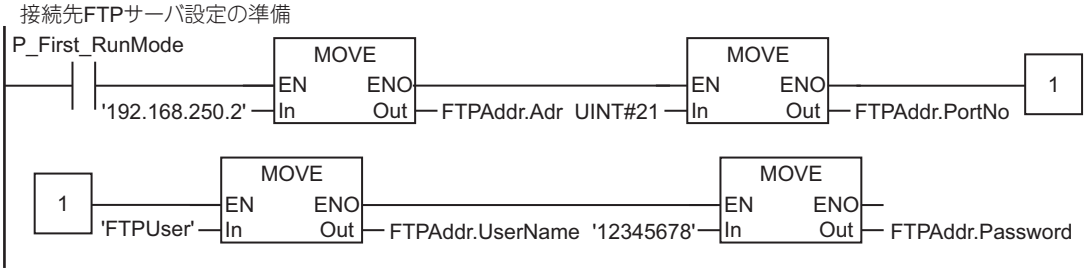
3 **すべての処理が正常に終了した場合は正常終了時処理を実行します。**
異常が発生した場合は異常終了時処理を実行します。

LD

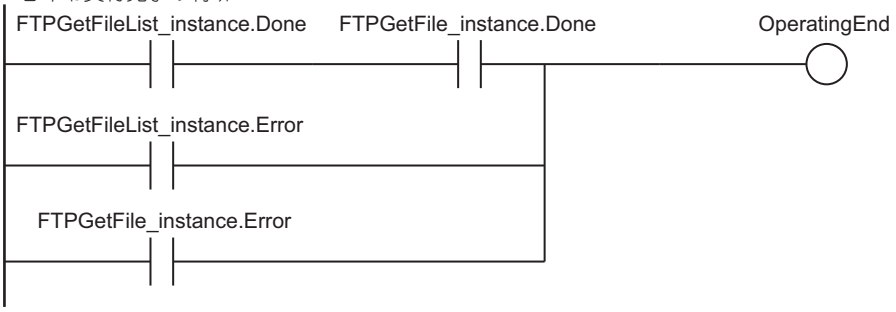
内部変数	名称	データ型	初期値	コメント
	FTPGetFileList_instance	FTPGetFileList		FTPGetFileList 命令のインスタンス
	FTPGetFile_instance	FTPGetFile		FTPGetFile 命令のインスタンス
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定

内部変数	名称	データ型	初期値	コメント
	FTPFileList	ARRAY[0..999] OF _sFTP_FILE_DETAIL	[1000((Name := '', ModifiedDate := DT#1970-01-01-00:00:00, Size := 0, ReadOnly := False, Folder := False))]	ファイル詳細情報
	GetResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	ダウンロード済みファイル結果
	FTPStoredNum	UINT	0	ファイルリスト取得済みファイル数
	LastFileIndex	UINT	0	名前昇順に並べたうちの、最後のファイルのインデックス
	RS_instance	RS		RS 命令のインスタンス
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中

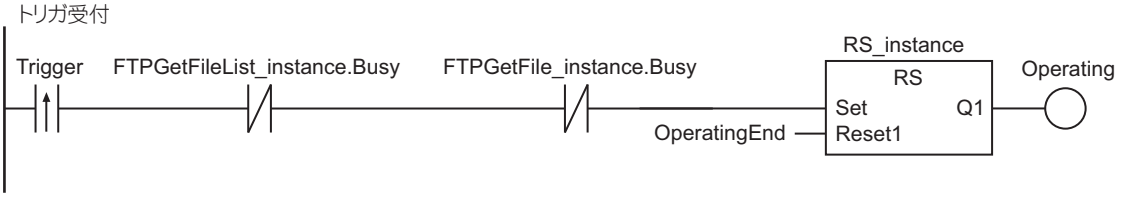
接続先FTPサーバ設定の準備

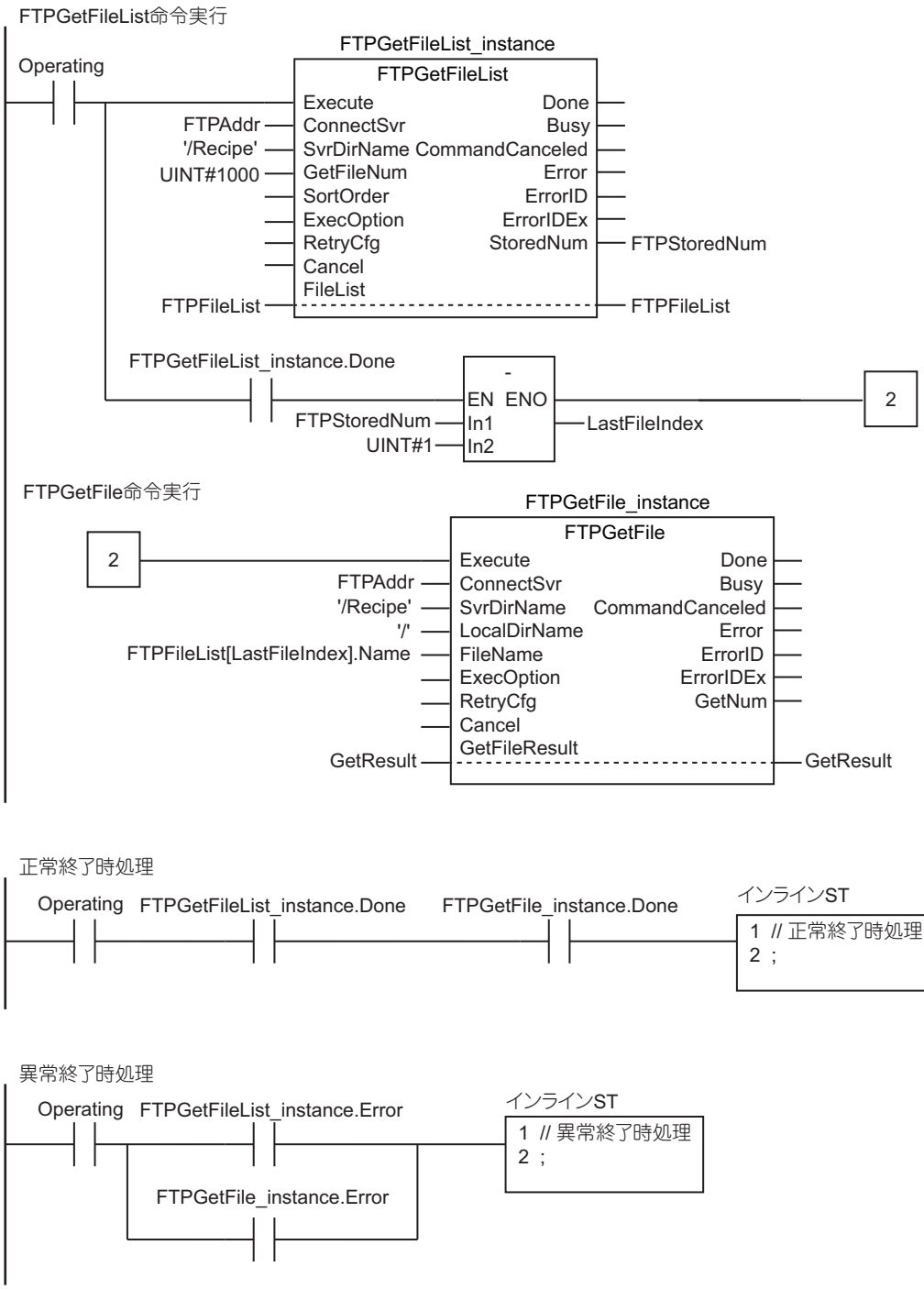


各命令実行完了の判断



トリガ受付





ST

内部変数	名称	データ型	初期値	コメント
	R_TRIG_instance	R_TRIG		R_TRIG 命令のインスタンス
	UP_Q	BOOL	FALSE	トリガ出力
	FTPGetFile_instance	FTPGetFile		FTPGetFile 命令のインスタンス
	FTPGetFileList_instance	FTPGetFileList		FTPGetFileList 命令のインスタンス

内部変数	名称	データ型	初期値	コメント
	FTPFileList	ARRAY[0..999] OF _sFTP_FILE_DETAIL	[1000((Name := '', ModifiedDate := DT#1970-01-01-00:00:00, Size := 0, ReadOnly := False, Folder := False))]	ファイル詳細情報
	FTPStoredNum	UINT	0	ファイルリスト取得済みファイル数
	DoFTPTrigger	BOOL	FALSE	FTPGetFileList、FTPGetFIle 実行条件
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定
	GetResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	ダウンロード済みファイル結果
	Stage	UINT	0	命令実行手順
	Trigger	BOOL	FALSE	実行条件

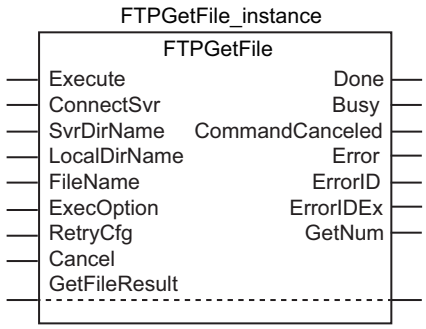
```
// 接続先 FTP サーバ設定の準備
IF P_First_RunMode THEN
   FTPAddr.Adr := '192.168.250.2';  // アドレス
   FTPAddr.PortNo := UINT#21;  // ポート番号
   FTPAddr.UserName := 'FtpUser';  // ユーザー名
   FTPAddr.Password := '12345678';  // パスワード
END_IF;

// トリガ受付
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(  // インスタンス初期化
      Execute := FALSE,
      ConnectSvr := FTPAddr,
      SvrDirName := '/Recipe',
      GetFileNum := UINT#1000,
      FileList := FTPFileList,
      StoredNum => FTPStoredNum) ;
   FTPGetFile_instance(  // インスタンス初期化
      Execute := FALSE,
      ConnectSvr := FTPAddr,
      SvrDirName := '/Recipe',
      LocalDirName := '/',
      FileName := '',
      GetFileResult := GetResult) ;
END_IF;
IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1 :  // FTPGetFileList 命令実行
```

```
            FTPGetFileList_instance(
               Execute := TRUE,  // 起動
               ConnectSvr := FTPAddr,  // 接続先FTPサーバ
               SvrDirName := '/Recipe',  // FTPサーバディレクトリ名
               GetFileNum := UINT#1000,  // 取得ファイル数
               FileList := FTPFileList,  // ファイル詳細
               StoredNum => FTPStoredNum);  // 取得済みファイル数
            IF (FTPGetFileList_instance.Done = TRUE) THEN
               Stage := INT#2;  // 次へ
            ELSIF (FTPGetFileList_instance.Error = TRUE) THEN
               Stage := INT#10;  // 異常終了
            END_IF;
         2 : // FTPGetFile命令実行
            FTPGetFile_instance(
               Execute := TRUE,  // 起動
               ConnectSvr := FTPAddr,  // 接続先FTPサーバ
               SvrDirName := '/Recipe',  // FTPサーバディレクトリ名
               LocalDirName := '/',  // ローカルディレクトリ名
               FileName := FTPFileList[FTPStoredNum - 1].Name,  // ファイル名
               GetFileResult := GetResult);  // ダウンロードファイル結果
            IF (FTPGetFile_instance.Done = TRUE) THEN
               Stage := INT#0;  // 正常終了
            ELSIF (FTPGetFile_instance.Error = TRUE) THEN
               Stage := INT#20;  // 異常終了
            END_IF;
         0 :  // 正常終了処理
            DoFTPTrigger := FALSE;
            Trigger := FALSE;
         ELSE  // 異常終了処理
            DoFTPTrigger := FALSE;
            Trigger := FALSE;
      END_CASE;
   END_IF;
```

FTPGetFile

FTP サーバからファイルをダウンロードします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPGetFile	FTP サーバからのファイルダウンロード	FB		FTPGetFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, GetFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx,GetNum);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	入力	接続先 FTP サーバの各種設定パラメータ	–	–	*1
SvrDirName	FTP サーバディレクトリ名		ファイルのダウンロード元となる FTP サーバのディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*2*3		" *4
LocalDirName	ローカルディレクトリ名		FTP サーバからダウンロードしたファイルを格納するディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*3		'/'
FileName	ファイル名		ダウンロードするファイル名 *5	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*3*6		*1
ExecOption	FTP 実行オプション		FTP 実行に関するオプション	–		–
RetryCfg	実行リトライ設定		命令実行リトライ設定			
Cancel	取り消し		TRUE: 命令実行を取り消す FALSE: 命令実行を取り消さない	データ型に従う		FALSE
GetFileResult[] 配列 *7 *8 *9	ダウンロード済ファイル結果	入出力	ダウンロードしたファイル結果	–	–	*1
CommandCanceled	取り消し完了	出力	TRUE: 取り消し完了 FALSE: 取り消し未完了	データ型に従う	–	–
GetNum	ダウンロードファイル数		ダウンロード対象のファイル数	–		

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. FTP サーバディレクトリ名として、つぎの文字は使用できません。

"*"、"?"、"<"、">"、"¦"、"""（ダブルクォーテーション）
*3. 半角スペースは、FTP サーバにより動作が変わることがありますので使用を推奨しません。
半角スペースを使用する場合は、2 文字として扱います。
*4. FTP サーバの、ログイン時のホームディレクトリです。
*5. ファイル名にワイルドカード指定が可能です。
*6. ファイル名として、つぎの文字は使用できません。
"¦"
*7. 配列の最大要素数は、1000 です。
*8. 1 次元の配列です。2 次元以上の配列を指定した場合は、ビルド時に異常になります。
*9. 先頭の要素番号は 0 です。先頭の要素番号が 0 以外の配列を指定した場合は、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	構造体_sFTP_CONNECT_SVR　詳細は「機能 (P.2-1118)」参照																			
SvrDirName																				○
LocalDirName																				○
FileName																				○
ExecOption	構造体_sFTP_EXEC_OPTION　詳細は「FTP サーバとの処理に関するオプション指定 (P.2-1120)」参照																			
RetryCfg	構造体_sFTP_RETRY_CFG　詳細は「FTP サーバとの接続処理のリトライ指定 (P.2-1122)」参照																			
Cancel	○																			
GetFileResult[] 配列	構造体_sFTP_FILE_RESULT　詳細は「機能 (P.2-1118)」参照																			
CommandCanceled	○																			
GetNum							○													

機能

接続先 FTP サーバ「ConnectSvr」上の指定ディレクトリ「SvrDirName」にある指定ファイル「FileName」を、SD メモリカード内の指定ディレクトリ「LocalDirName」へダウンロードします。指定ディレクトリ「LocalDirName」が SD メモリカード内に存在しない場合は、ディレクトリを作成してファイルをダウンロードします。
「FileName」には、ワイルドカードの指定が可能です。したがって、複数のファイルを一度にダウンロードすることができます。

ダウンロードした結果は、ファイル単位でダウンロードファイル結果 GetFileResult[]に格納します。
ダウンロードファイル数「GetNum」には、ダウンロードの対象となるファイル数を格納します。
「FileName」にワイルドカードの指定をした場合は、ワイルドカードに合致するファイル数を格納します。

ファイル転送時に異常があった場合は、「GetFileResult[].TxError」の値が TRUE になります。
ダウンロード後、転送元ファイル削除時に異常があった場合は、「GetFileResult[].RemoveError」の値が TRUE になります。

NY シリーズ コントローラでは、共有フォルダ（仮想 SD メモリカード）内にダウンロードします。ダウンロードするには、事前に仮想 SD メモリカードの設定が必要です。仮想 SD メモリカードの詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。

「ConnectSvr」のデータ型は、構造体_sFTP_CONNECT_SVR です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	接続先 FTP サーバの各種設定パラメータ	_sFTP_CONNECT_SVR	–	–	–
Adr	アドレス	IP アドレスまたは、ホスト名 *1	STRING	1 ～ 200 バイト *2	–	–
PortNo	ポート番号	FTP サーバのコントロールコネクションの TCP ポート番号	UINT	0 ～ 65535 *3		
UserName	ユーザ名	FTP サーバのユーザ名	STRING	最大 33 バイト *4 *5		
Password	パスワード	FTP サーバのパスワード	STRING	最大 33 バイト *4 *5		

*1. ホスト名を指定時は、別途 DNS 設定または Hosts 設定が必要です。
*2. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（アンダーバー）です。
*3. 「0」の場合、TCP ポート番号は 21 になります。
*4. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（ アンダーバー）です。CPU ユニット Ver.1.16 以降では、“\”（バックスラッシュ）、“@”の入力も可能です。
*5. バイト数には、終端 NULL 文字を含みます。

GetFileResult[]のデータ型は、構造体_sFTP_FILE_RESULT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
GetFileResult	ダウンロード済ファイル結果	転送ファイルの結果	_sFTP_FILE_RESULT	–	–	–
Name	ファイル名 *1	転送済ファイル名	STRING	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）	–	–
TxError	転送異常	TRUE: 転送異常終了 FALSE: 転送正常終了	BOOL	データ型に従う		
RemoveError	削除異常	TRUE: 削除異常終了 FALSE: 削除正常終了	BOOL			
Reserved	予約	システム予約	ARRAY[0..3] Of Byte	–		0

*1. 拡張子付きです。

ワイルドカードでのファイル名指定

ダウンロードするファイル名「FileName」には、ワイルドカード指定が可能です。
ワイルドカード指定する文字には、“*”と“?”を使用します。
“*”は複数文字、“?”は 1 文字を示します。
以下にワイルドカードの指定方法の例を示します。

FTP サーバのディレクトリのファイル構成を以下とします。

```
┣DataFiles(対象ディレクトリ)
┃  LogA01.log
┃  LogA02.txt
┃  LogB.log
┃  LogC.txt
┃  ControlDataA1.csv
┃  ControlDataA10.csv
┃  ControlDataA100.csv
┃  ControlDataB10.csv
┃  ControlDataC100.csv
┃
┃  ┗ControlSubDataFiles(サブディレクトリ)
┃      SubData_A001.txt
┃      SubData_A002.txt
```

下記の表のように、ワイルドカードの指定方法によって、対象ファイルが決まります。

ワイルドカード指定	対象ファイル
Log*.log	LogA01.log, LogB.log
Log?.log	LogB.log
Log?.*	LogB.log, LogC.txt
Data	ControlDataA1.csv, ControlDataA10.csv, ControlDataA100.csv, ControlDataB10.csv, ControlDataC100.csv, (ControlSubDataFiles)*1
*	サブディレクトリ内のファイルを除く全ファイル
.	サブディレクトリ内のファイルを除く全ファイル
?.?	対象ファイルなし
????.???	LogB.log, LogC.txt

*1. FTP サーバの仕様により、サブディレクトリも対象に含まれる場合があります。

ワイルドカード指定した場合、ダウンロードするファイルの最大数は 1000 個となります。
ダウンロードした結果、いずれかのファイルで「GetFileResult[].TxError」または「GetFileResult[].RemoveError」が TRUE になった場合、「Error」が TRUE になり、最初に異常になった「ErrorID」に該当エラーコードを格納し、FTP サーバからのエラーレスポンスを「ErrorIDEx」に格納します。

FTP サーバとの処理に関するオプション指定

FTP サーバからファイルをダウンロードする際に、FTP 実行オプション「ExecOption」で指定した動作を行います。

「ExecOption」のデータ型は、構造体_sFTP_EXEC_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ExecOption	FTP 実行オプション	FTP 実行に関するオプション	_sFTP_EXEC_OPTION	–	–	–
PassiveMode	パッシブモード指定	TRUE: Passive モード FALSE: Active モード	BOOL	データ型に従う	–	FALSE
ASCIIMode	ASCII モード指定	TRUE: ASCII モード FALSE: Binary モード	BOOL			
FileRemove	転送後ファイル削除指定 *1	TRUE: 転送後ファイルを削除する FALSE: 転送後ファイルを削除しない	BOOL			
OverWrite	上書き指定	TRUE: 転送先のファイルを上書きする FALSE: 転送先のファイルを上書きしない	BOOL			
Reserved	予約	システム予約	ARRAY[0..7] Of Byte	–		0

*1. 転送失敗時は、転送元ファイルを削除しません。

● PassiveMode（パッシブモード指定）

パッシブモード指定は、FTP サーバとのデータコネクションの接続要求を、パッシブモードで行うかどうかの指定です。
パッシブモードを指定しない場合は、FTP サーバとのデータコネクションの接続要求は、アクティブモードで行われます。
コネクションの接続要求の方法の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル 内蔵 EtherNet/IP ポート(SBCD-369)』を参照してください。
「PassiveMode」の値と意味は、以下のとおりです。

設定値	意味
TRUE	FTP サーバとのデータコネクションの接続要求が、パッシブモードで行われます。 FTP クライアント側からデータコネクションの接続要求を行います。
FALSE	FTP サーバとのデータコネクションの接続要求が、アクティブモードで行われます。 FTP サーバ側からデータコネクションの接続要求を行います。

● ASCIIMode（ASCII モード指定）

ASCII モード指定は、転送元システムから転送先システムへの転送モードを、ASCII モードにするかどうかの指定です。
ASCII モードを指定しない場合は、転送元システムから転送先システムへの転送モードが、Binary モードになります。
「ASCIIMode」の値と意味は、以下のとおりです。

設定値	意味
TRUE	転送元システムから転送先システムへの転送モードを、ASCII モードにします。 テキストの改行コードを転送元のシステムのものから転送先のシステムものに変換して転送します。
FALSE	転送元システムから転送先システムへの転送モードを、Binary モードにします。 テキストの改行コードを転送元のシステムのまま変換せず、そのまま転送します。

● FileRemove（転送後ファイル削除指定）

転送後ファイル削除指定は、転送元のファイルを転送先にダウンロード後、転送元のファイルを削除するかどうかの指定です。
「FileRemove」の値と意味は、以下のとおりです。

設定値	意味
TRUE	転送元のファイルを削除します。
FALSE	転送元のファイルを削除しません。

- **OverWrite（上書き指定）**

上書き指定は、転送先にファイルをダウンロード時に、転送先に同名のファイルがあった場合、上書きするかどうかの指定です。
上書きしない場合は、転送先に同名のファイルがあった場合に、ファイルを転送しません。
ファイル名の大文字と小文字は区別しません。
「OverWrite」の値と意味は、以下のとおりです。

設定値	意味
TRUE	転送先のファイルを上書きします。
FALSE	転送先のファイルを上書きしません。転送先にファイルを転送しません。

FTP サーバとの接続処理のリトライ指定

FTP サーバとの接続処理のリトライが可能です。
リトライ設定内容および動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「FTP サーバとの接続処理のリトライ指定 (P.2-1106)」を参照してください。

命令実行の取り消し

FTPGetFile 命令実行中に、命令実行の取り消しが可能です。
取り消し前までに、FTP サーバからファイルをダウンロードした結果は、「GetNum」およびダウンロード済ファイル結果 GetFileResult[]に格納されます。
取り消し動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「命令実行の取り消し (P.2-1108)」を参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			
_Card1Ready	SD メモリカード使用可フラグ	BOOL	SD メモリカードの認識ができ、使用可能なことを示します。 TRUE: 使用可 FALSE: 使用不可

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

使用上の注意

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。

- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- ダウンロードファイル済結果に格納する結果の数が GetFileResult[]の配列要素数を超える場合は、超えた分の結果情報は格納しません。この場合「Error」は TRUE になりません。
- 255 文字を超えるファイル名については先頭から 255 文字分を GetFileResult[]の「Name」に格納します。この場合「Error」は TRUE になりません。
- FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令で同時実行可能な個数は、最大 3 です。
- ワイルドカードでファイル名を指定し、複数のファイルで異常が発生した場合、「ErrorID」、「ErrorIDEx」には、GetFileResult[]に格納されたファイルのうち、「GetFileResult[].TxError」の値が TRUE である先頭のファイルの結果が格納されます。
- ファイル名の大文字と小文字は区別しません。したがって、転送先に、転送ファイルと大文字小文字の違いしかない名前のファイルが存在すると、同名ファイルと認識されます。その場合の処理は、以下のとおりです。

「OverWrite」の値	上書き指定	処理
TRUE	上書きする	上書きします。
FALSE	上書きしない	転送先のファイルを上書きしません。転送先にファイルを転送しません。

- 「FileName」で指定したファイルが、FTP サーバ内の指定ディレクトリに存在しない場合は転送異常となり、「GetFileResult[].TxError」の値が TRUE になります。
- 「FileName」で指定したファイルが、ディレクトリの場合は転送異常となり、「GetFileResult[].TxError」の値が TRUE になります。
- FTP 実行オプションの転送後ファイル削除指定「ExecOption.FileRemove」が TRUE で、「FileName」で指定したファイルの属性が読み取り専用になっていた場合は削除異常となり、「GetFileResult[].RemoveError」が TRUE になります。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) 入力パラメータのいずれかがの値が、有効範囲外の場合
  b)「SvrDirName」、「LocalDirName」に、1 階層上のディレクトリ“..”を指定した場合
  c)「SvrDirName」、「LocalDirName」に、“//”などの不正なパスを指定した場合
  d)「SvrDirName」で指定したディレクトリが FTP サーバに存在しない場合
  e)「SvrDirName」で指定した FTP サーバのディレクトリに、1000 個以上のダウンロード対象となるファイルが存在する場合
  f)「FileName」で指定したファイルが FTP サーバのダウンロード元となるディレクトリに存在しない場合
  g)「ExecOption.OverWrite」が FALSE、かつ指定ディレクトリ「SvrDirName」に、指定ファイル名「FileName」と同じファイルがすでに存在する場合
  h)「ExecOption.FileRemove」が TRUE、かつ指定ファイル名「FileName」に合致するファイルの属性が読み取り専用である場合
  i)「ConnectSvr」で指定した FTP サーバがネットワーク上に存在しない、またはサービスを停止している場合
  j) ファイルのアクセス権がない、ファイルが壊れているなどの理由で、「FileName」で指定したファイルへのアクセスに失敗した場合
  k) FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令を、同時に 3 個を超えて実行した場合
  l) SD メモリカードが、使用可能状態ではない場合
  m) SD メモリカードが、ライトプロテクト中の場合
  n) SD メモリカードが、空き容量不足の場合
  o) SD メモリカードに、作成可能なファイル数、ディレクトリ数を超える場合
- 拡張エラーコード「ErrorIDEx」は、本命令では、FTP サーバより返信された FTP レスポンスコードを表示します。代表的な「ErrorIDEx」の値、異常内容および対処方法は、以下のとおりです。詳

細については、FTP サーバの仕様を確認してください。「ErrorIDEx」は、エラーコード「ErrorID」の値が WORD#16#2407 の場合に出力されます。

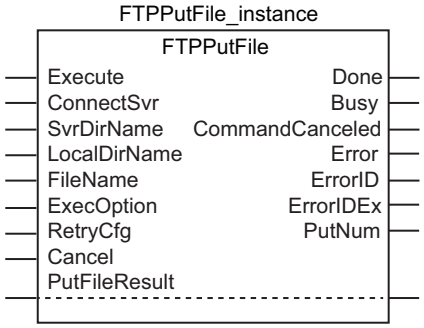
「ErrorIDEx」の値	異常内容	対処方法
16#000001A9	データコネクションを確立できませんでした。	インターネット上の FTP サーバとの FTP 通信の場合、FTP のオープンモードが Active になっていないか確認してください。
16#000001AA	コネクションがクローズされています。データ転送を中止しました。	FTP サーバとの接続を確認してください。 FTP サーバが停止していないか確認してください。
16#000001C2	要求されたファイル操作を行えませんでした。ファイルが、すでにオープンされている場合など、ファイルが使用できません。	対象のファイルが他アプリケーションによりオープンされていないか確認してください。
16#00000212	ユーザはログインできませんでした。	FTP ユーザ名またはパスワードを確認してください。
16#00000214	ファイル保存用のアカウントが必要です。	FTP ユーザのアクセス権を確認してください。
16#00000226	ファイルが見つからないためにアクセスできない場合など、ファイルが使用できないため、要求されたファイル操作を実行しませんでした。	指定したファイル名に該当するファイルが FTP サーバのディレクトリに存在するか確認してください。 指定したファイルのアクセス権を確認してください。
16#00000229	ファイル名が不正なため実行できませんでした。	指定したディレクトリのアクセス権を確認してください。

サンプルプログラム

FTPGetFileList 命令の、「サンプルプログラム (P.2-1111)」を参照してください。

FTPPutFile

FTP サーバへファイルをアップロードします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPPutFile	FTP サーバへのファイルアップロード	FB		FTPPutFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, PutFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, PutNum);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	入力	接続先 FTP サーバの各種設定パラメータ	－	－	*1
SvrDirName	FTP サーバディレクトリ名		ファイルのアップロード先となる FTP サーバのディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*2*3		" *4
LocalDirName	ローカルディレクトリ名		FTP サーバにアップロードするファイルが格納されているディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*3		'/'
FileName	ファイル名		アップロードするファイル名 *5			*1
ExecOption	FTP 実行オプション		FTP 実行に関するオプション	－		－
RetryCfg	実行リトライ設定		命令実行リトライ設定			
Cancel	取り消し		TRUE: 命令実行を取り消す FALSE: 命令実行を取り消さない	データ型に従う		FALSE
PutFileResult[] 配列 *6 *7 *8	アップロード済ファイル結果	入出力	アップロードファイル結果	－	－	*1
CommandCanceled	取り消し完了	出力	TRUE: 取り消し完了 FALSE: 取り消し未完了	データ型に従う	－	－
PutNum	アップロードファイル数		アップロード対象のファイル数	－		

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. FTP サーバディレクトリ名として、以下の文字は使用できません。
"*"、"?"、"<"、">"、"|"、"""（ダブルクォーテーション）
*3. 半角スペースは、FTP サーバにより動作が変わることがありますので使用を推奨しません。
半角スペースを使用する場合は、2 文字として扱います。

*4. FTP サーバの、ログイン時のホームディレクトリです。
*5. ファイル名にワイルドカード指定が可能です。
*6. 配列の最大要素数は、1000 です。
*7. 1 次元の配列です。2 次元以上の配列を指定した場合は、ビルド時に異常になります。
*8. 先頭の要素番号は 0 です。先頭の要素番号が 0 以外の配列を指定した場合は、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	構造体_sFTP_CONNECT_SVR　詳細は「機能 (P.2-1126)」参照																			
SvrDirName																				○
LocalDirName																				○
FileName																				○
ExecOption	構造体_sFTP_EXEC_OPTION　詳細は「FTP サーバとの処理に関するオプション指定 (P.2-1127)」参照																			
RetryCfg	構造体_sFTP_RETRY_CFG　詳細は「FTP サーバとの接続処理のリトライ指定 (P.2-1122)」参照																			
Cancel	○																			
PutFileResult[] 配列	構造体_sFTP_FILE_RESULT　詳細は「機能 (P.2-1126)」参照																			
CommandCanceled	○																			
PutNum							○													

機能

接続先 FTP サーバ「ConnectSvr」の指定ディレクトリ「SvrDirName」へ、SD メモリカード内の指定ディレクトリ「LocalDirName」にある指定ファイル「FileName」をアップロードします。
指定ディレクトリ「SvrDirName」が FTP サーバに存在しない場合は、ディレクトリを作成してファイルをアップロードします。
「FileName」には、ワイルドカードの指定が可能です。したがって、複数のファイルを一度にアップロードすることができます。

アップロードした結果は、ファイル単位でアップロードファイル結果 PutFileResult[]に格納します。
アップロードファイル数「PutNum」には、アップロードの対象となるファイル数を格納します。
「FileName」にワイルドカードの指定をした場合は、ワイルドカードに合致するファイル数を格納します。

ファイル転送時に異常があった場合は、「PutFileResult[].TxError」の値が TRUE になります。
アップロード後、転送元ファイル削除時に異常があった場合は、「PutFileResult[].RemoveError」の値が TRUE になります。

NY シリーズ コントローラでは、共有フォルダ（仮想 SD メモリカード）内にダウンロードします。ダウンロードするには、事前に仮想 SD メモリカードの設定が必要です。共有フォルダ（仮想 SD メモリカード）の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。

「ConnectSvr」のデータ型は、構造体_sFTP_CONNECT_SVR です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	接続先 FTP サーバの各種設定パラメータ	_sFTP_CONNECT_SVR	–	–	–
Adr	アドレス	IP アドレスまたは、ホスト名 *1	STRING	1 ～ 200 バイト *2	–	–
PortNo	ポート番号	FTP サーバのコントロールコネクションの TCP ポート番号	UINT	0 ～ 65535 *3		
UserName	ユーザ名	FTP サーバのユーザ名	STRING	最大 33 バイト *4 *5		
Password	パスワード	FTP サーバのパスワード	STRING	最大 33 バイト *4 *5		

*1. ホスト名を指定時は、別途 DNS 設定または Hosts 設定が必要です。
*2. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（アンダーバー）です。
*3. 0 の場合、TCP ポート番号は 21 になります。
*4. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（ アンダーバー）です。CPU ユニット Ver.1.16 以降では、“\”（バックスラッシュ）、“@” の入力も可能です。
*5. バイト数には、終端 NULL 文字を含みます。

PutFileResult[]のデータ型は、構造体_sFTP_FILE_RESULT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
PutFileResult	アップロード済ファイル結果	転送ファイルの結果	_sFTP_FILE_RESULT	–	–	–
Name	ファイル名 *1	転送済ファイル名	STRING	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）	–	–
TxError	転送異常	TRUE: 転送異常終了 FALSE: 転送正常終了	BOOL	データ型に従う		
RemoveError	削除異常	TRUE: 削除異常終了 FALSE: 削除正常終了	BOOL			
Reserved	予約	システム予約	ARRAY[0..3] Of Byte	–		0

*1. 拡張子付きです。

ワイルドカードでのファイル名を指定

アップロードするファイル名には、ワイルドカードの指定が可能です。
ワイルドカードの指定は、FTPGetFile 命令と同じ仕様です。FTPGetFile 命令の「ワイルドカードでのファイル名指定 (P.2-1119)」を参照してください。

FTP サーバとの処理に関するオプション指定

アップロード時に FTP サーバとの処理のオプションの指定が可能です。
オプション設定内容は、FTPGetFile 命令と同じ仕様です。FTPGetFile 命令の「FTP サーバとの処理に関するオプション指定 (P.2-1120)」参照してください。

FTP サーバとの接続処理のリトライ指定

FTP サーバとの接続処理のリトライが可能です。
リトライ設定内容および動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「FTP サーバとの接続処理のリトライ指定 (P.2-1106)」を参照してください。

命令実行の取り消し

FTPPutFile 命令実行中に、命令実行の取り消しが可能です。
取り消し前に、FTP サーバからファイルをアップロードした結果は、「PutNum」およびアップロード済ファイル結果 PutFileResult[]に格納されます。
取り消し動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「命令実行の取り消し (P.2-1108)」参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			
_Card1Ready	SD メモリカード使用可フラグ	BOOL	SD メモリカードの認識ができ、使用可能なことを示します。 TRUE: 使用可 FALSE: 使用不可

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

使用上の注意

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- アップロード済ファイル結果に格納する結果の数が PutFileResult[]の配列要素数を超える場合は、超えた分の結果情報は格納しません。この場合「Error」は TRUE になりません。
- 255 文字を超えるファイル名については先頭から 255 文字分を PutFileResult[]の「Name」に格納します。この場合「Error」は TRUE になりません。
- FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令で同時実行可能な個数は、最大 3 です。
- ワイルドカードでファイル名を指定し、複数のファイルで異常が発生した場合、「ErrorID」、「ErrorIDEx」には、PutFileResult[]に格納されたファイルのうち、「PutFileResult[].TxError」の値が TRUE である先頭のファイルの結果が格納されます。
- ファイル名の大文字と小文字は区別しません。したがって、転送先に、転送ファイルと大文字小文字の違いしかない名前のファイルが存在すると、同名ファイルと認識されます。その場合の処理は、以下のとおりです。

「OverWrite」の値	上書き指定	処理
TRUE	上書きする	上書きするかしないかは、FTP サーバの仕様によります。
FALSE	上書きしない	転送先のファイルを上書きしません。転送先にファイルを転送しません。

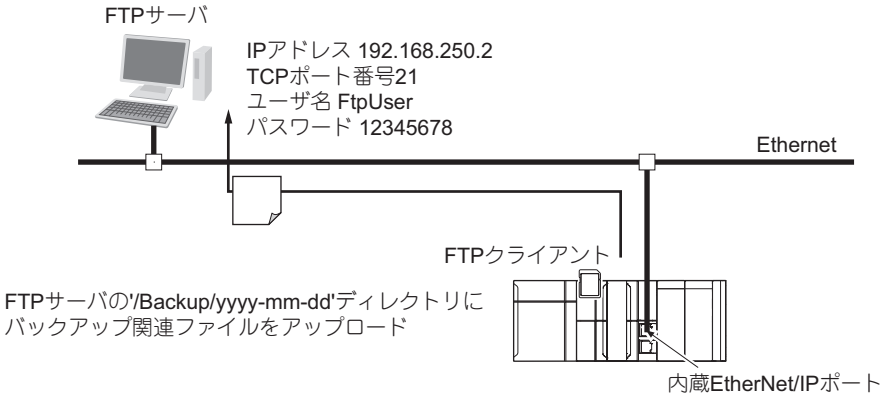
- 「FileName」で指定したファイルが、SD メモリカード内の指定ディレクトリに存在しない場合は転送異常になり、「PutFileResult[].TxError」の値が TRUE になります。
- 「FileName」で指定したファイルが、ディレクトリの場合は転送異常となり、「PutFileResult[].TxError」の値が TRUE になります。
- 転送後ファイル削除指定「ExecOption.FileRemove」が TRUE で、「FileName」で指定したファイルの属性が読み取り専用になっていた場合は削除異常になり、「PutFileResult[].RemoveError」の値が TRUE になります。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) 入力パラメータのいずれかがの値が、有効範囲外の場合
  b)「SvrDirName」、「LocalDirName」に、1 階層上のディレクトリ“..”を指定した場合
  c)「SvrDirName」、「LocalDirName」に、“//”などの不正なパスを指定した場合
  d)「SvrDirName」で指定したディレクトリが FTP サーバに存在しない場合
  e)「LocalDirName」で指定したディレクトリが、FTP クライアントに存在しない場合
  f)「LocalDirName」で指定したディレクトリに、1000 個以上のアップロード対象となるファイルが存在する場合
  g)「FileName」で指定したファイルが、SD メモリカードのアップロード元となるディレクトリに存在しない場合
  h)「ExecOption.OverWrite」が FALSE、かつ指定ディレクトリ「SvrDirName」に、指定ファイル名「FileName」と同じファイルがすでに存在する場合
  i)「ExecOption.FileRemove」が TRUE、かつ指定ファイル名「FileName」に合致するファイルの属性が読み取り専用である場合
  j)「ConnectSvr」で指定した FTP サーバがネットワーク上に存在しない、またはサービスを停止している場合
  k) ファイルのアクセス権がない、ファイルが壊れているなどの理由で、「FileName」で指定したファイルへのアクセスに失敗した場合
  l) FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令を、同時に 3 個を超えて実行した場合
  m) SD メモリカードが、使用可能状態ではない場合
- 拡張エラーコード「ErrorIDEx」は、本命令では、FTP サーバより返信された FTP レスポンスコードを表示します。代表的な「ErrorIDEx」の値、異常内容および対処方法は、以下のとおりです。詳細については、FTP サーバの仕様を確認してください。「ErrorIDEx」は、エラーコード「ErrorID」の値が WORD#16#2407 の場合に出力されます。

「ErrorIDEx」の値	異常内容	対処方法
16#000001A9	データコネクションを確立できませんでした。	インターネット上の FTP サーバとの FTP 通信の場合、FTP のオープンモードが Active になっていないか確認してください。
16#000001AA	コネクションがクローズされています。データ転送を中止しました。	FTP サーバとの接続を確認してください。 FTP サーバが停止していないか確認してください。
16#000001C2	要求されたファイル操作を行えませんでした。ファイルが、すでにオープンされている場合など、ファイルが使用できません。	対象のファイルが他アプリケーションによりオープンされていないか確認してください。
16#00000212	ユーザはログインできませんでした。	FTP ユーザ名またはパスワードを確認してください。
16#00000214	ファイル保存用のアカウントが必要です。	FTP ユーザのアクセス権を確認してください。

「ErrorIDEx」の値	異常内容	対処方法
16#00000226	ファイルが見つからないためにアクセスできない場合など、ファイルが使用できないため、要求されたファイル操作を実行しませんでした。	指定したファイル名に該当するファイルがFTPサーバのディレクトリに存在するか確認してください。 指定したファイルのアクセス権を確認してください。
16#00000229	ファイル名が不正なため実行できませんでした。	指定したディレクトリのアクセス権を確認してください。

サンプルプログラム

SD メモリカードバックアップを実行し、すべてのバックアップ関連ファイルを、FTP サーバの'/Backup/yyyy-mm-dd'ディレクトリにアップロードします。



コントローラと FTP サーバは、EtherNet/IP ネットワークで接続しています。FTP サーバと接続するための設定パラメータの値は、以下のとおりとします。

設定パラメータ	値
IP アドレス	192.168.250.2
TCP ポート番号	21
ユーザ名	FtpUser
パスワード	12345678

処理の手順は、以下のようになります。

1 **BackupToMemoryCard 命令を使用して、コントローラのバックアップ関連ファイルを、SD メモリカードのルートディレクトリに保存します。**

2 **FTPPutFile 命令を使用して、バックアップ関連ファイルを FTP サーバの'/Backup/yyyy-mm-dd'ディレクトリにアップロードします。転送ファイル名は、ワイルドカード指定で'*.*'とします。**

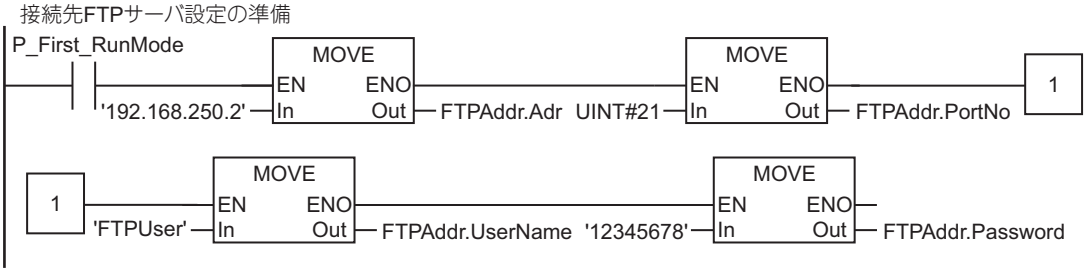
3 **すべての処理が正常に終了した場合は正常終了時処理を実行します。**
異常が発生した場合は、異常終了時処理を実行します。

LD

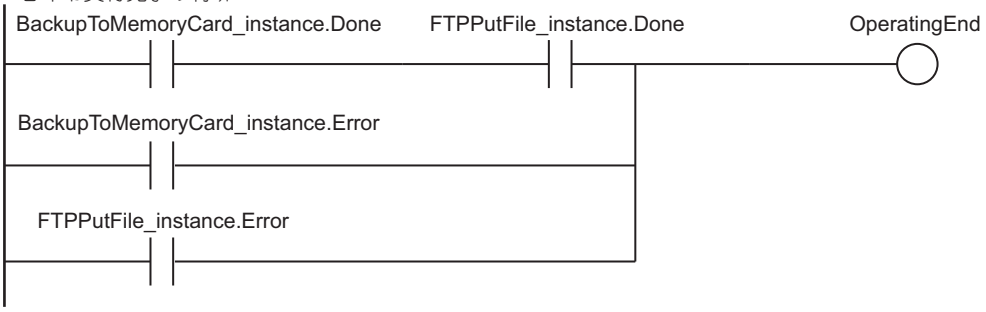
内部変数	名称	データ型	初期値	コメント
	FTPPutFile_instance	FTPPutFile		FTPPutFile 命令のインスタンス

内部変数	名称	データ型	初期値	コメント
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定
	PutResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	アップロード済みファイル結果
	RS_instance	RS		RS 命令のインスタンス
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	BackupToMemoryCard_instance	BackupToMemoryCard		BackupToMemoryCard 命令のインスタンス

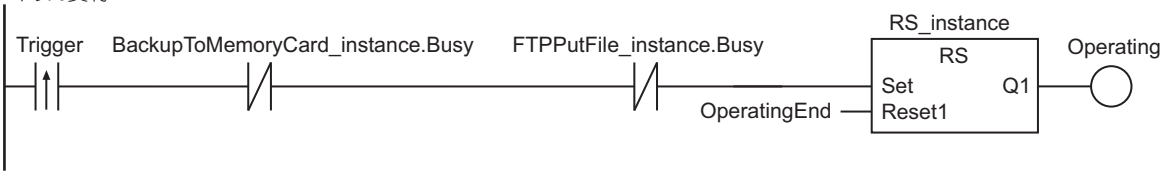
接続先FTPサーバ設定の準備

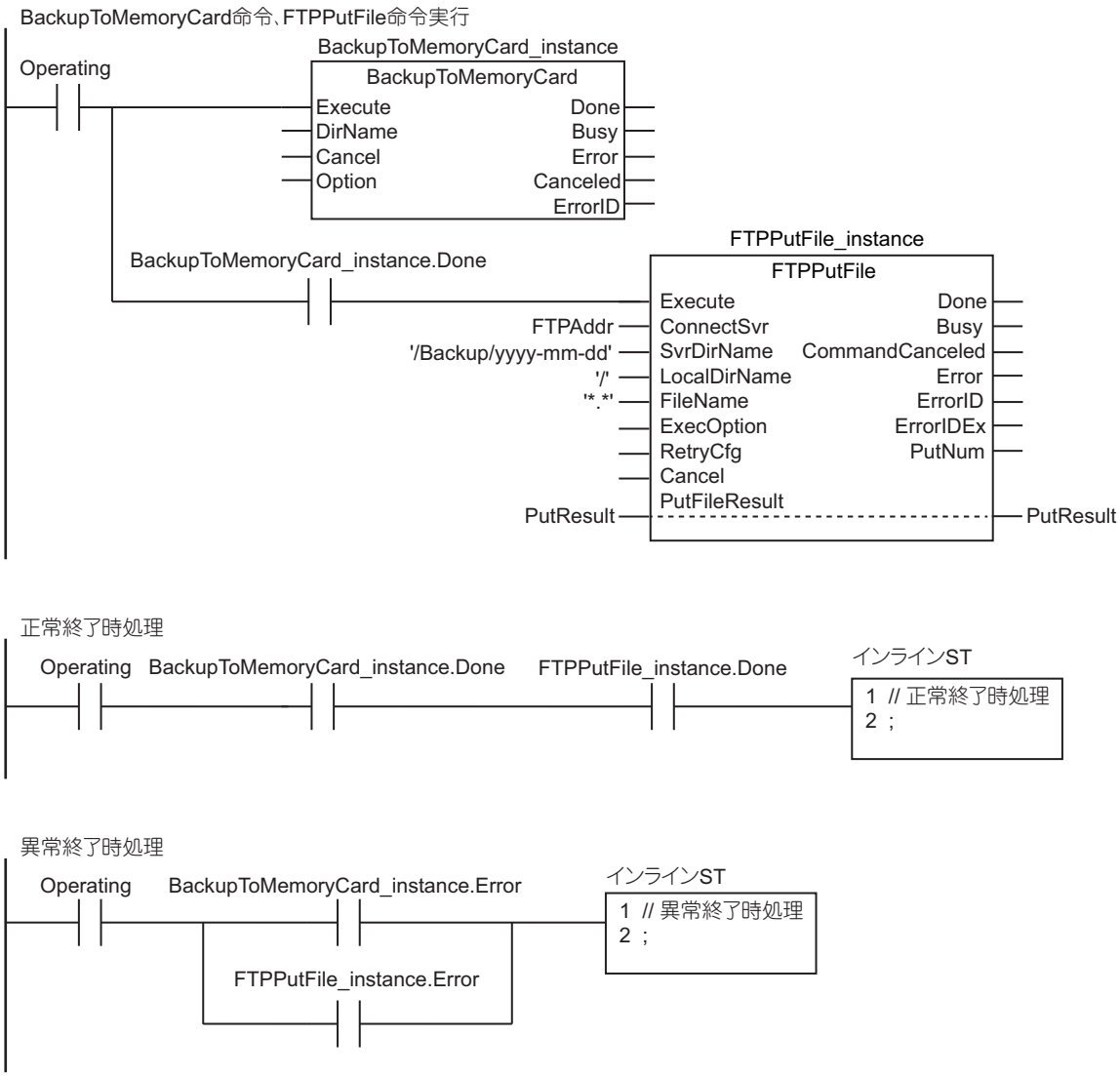


各命令実行完了の判断



トリガ受付





ST

内部変数	名称	データ型	初期値	コメント
	R_TRIG_instance	R_TRIG		R_TRIG 命令のインスタンス
	UP_Q	BOOL	FALSE	トリガ出力
	FTPPutFile_instance	FTPPutFile		FTPPutFile 命令のインスタンス
	DoFTPTrigger	BOOL	FALSE	BackupToMemoryCard、FTPPutFIle 実行条件
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定
	PutResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	アップロード済みファイル結果
	Stage	UINT	0	命令実行手順
	Trigger	BOOL	FALSE	実行条件

内部変数	名称	データ型	初期値	コメント
	BackupToMemoryCard_instance	BackupToMemoryCard		BackupToMemoryCard 命令のインスタンス


```
// 接続先 FTP サーバ設定の準備
IF P_First_RunMode THEN
   FTPAddr.Adr      := '192.168.250.2';// アドレス
   FTPAddr.PortNo   := UINT#21;        // ポート番号
   FTPAddr.UserName := 'FtpUser';      // ユーザー名
   FTPAddr.Password := '12345678';     // パスワード
END_IF;

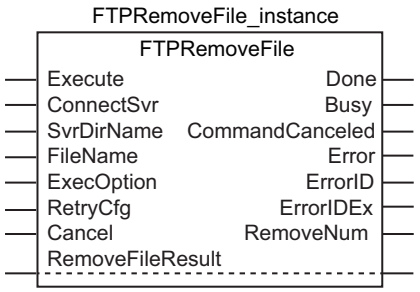
// トリガ受付
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (BackupToMemoryCard_instance.Busy = FALSE) AND
     (FTPPutFile_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   BackupToMemoryCard_instance(  // インスタンス初期化
      Execute := FALSE) ;
   FTPPutFile_instance(  // インスタンス初期化
      Execute       := FALSE,
      ConnectSvr    := FTPAddr,
      SvrDirName    := '/Backup/yyyy-mm-dd',
      LocalDirName  := '/',
      FileName      := '*.*',
      PutFileResult := PutResult) ;
END_IF;

IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1 :// BackupToMemoryCard 命令実行
         BackupToMemoryCard_instance(
            Execute := TRUE) ;// 起動
         IF (BackupToMemoryCard_instance.Done = TRUE) THEN
            Stage := INT#2;  // 次へ
         ELSIF (BackupToMemoryCard_instance.Error = TRUE) THEN
            Stage := INT#10; // 異常終了
         END_IF;
      2 : // FTPPutFile 命令実行
         FTPPutFile_instance(
            Execute       := TRUE,                // 起動
            ConnectSvr    := FTPAddr,             // 接続先 FTP サーバ
            SvrDirName    := '/Backup/yyyy-mm-dd',// FTP サーバディレクトリ名
            LocalDirName  := '/',                 // ローカルディレクトリ名
            FileName      := '*.*',               // ファイル名
            PutFileResult := PutResult) ;         // アップロードファイル結果
         IF (FTPPutFile_instance.Done = TRUE) THEN
            Stage := INT#0;  // 正常終了
         ELSIF (FTPPutFile_instance.Error = TRUE) THEN
```

```
            Stage := INT#20; // 異常終了
         END_IF;
      0 :  // 正常終了処理
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
      ELSE // 異常終了処理
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
   END_CASE;
END_IF;
```

FTPRemoveFile

FTP サーバのファイルを削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPRemoveFile	FTP サーバのファイル削除	FB		FTPRemoveFile_instance(Execute, ConnectSvr, SvrDirName, FileName, ExecOption, RetryCfg, Cancel, RemoveFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, RemoveNum);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	入力	接続先 FTP サーバの各種設定パラメータ	－	－	*1
SvrDirName	FTP サーバディレクトリ名		削除するファイルが存在する FTP サーバのディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*2*3		" *4
FileName	ファイル名		削除するファイル名 *5	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*3*6		*1
ExecOption	FTP 実行オプション		FTP 実行に関するオプション	－		－
RetryCfg	実行リトライ設定		命令実行リトライ設定			
Cancel	取り消し		TRUE: 命令実行を取り消す FALSE: 命令実行を取り消さない	データ型に従う		
RemoveFileResult[] 配列 *7 *8 *9	削除済ファイル結果	入出力	削除したファイル結果	－	－	*1
CommandCanceled	取り消し完了	出力	TRUE: 取り消し完了 FALSE: 取り消し未完了	データ型に従う	－	－
RemoveNum	削除ファイル数		削除対象のファイル数	－		

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*2. FTP サーバディレクトリ名として、以下の文字は使用できません。
"*"、"?"、"<"、">"、"|"、"""（ダブルクォーテーション）
*3. 半角スペースは、FTP サーバにより動作が変わることがありますので使用を推奨しません。
半角スペースを使用する場合は、2 文字として扱います。
*4. FTP サーバの、ログイン時のホームディレクトリです。

*5. ファイル名にワイルドカード指定が可能です。
*6. ファイル名として、つぎの文字は使用できません。
"|"
*7. 配列の最大要素数は、1000 です。
*8. 1 次元の配列です。2 次元以上の配列を指定した場合は、ビルド時に異常になります。
*9. 先頭の要素番号は 0 です。先頭の要素番号が 0 以外の配列を指定した場合は、ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	構造体_sFTP_CONNECT_SVR　詳細は「機能 (P.2-1136)」参照																			
SvrDirName																				○
FileName																				○
ExecOption	構造体_sFTP_EXEC_OPTION　詳細は「FTP サーバとの処理に関するオプション指定 (P.2-1137)」参照																			
RetryCfg	構造体_sFTP_RETRY_CFG　詳細は「FTP サーバとの接続処理のリトライ指定 (P.2-1122)」参照																			
Cancel	○																			
RemoveFileResult[] 配列	構造体_sFTP_FILE_RESULT　詳細は「機能 (P.2-1136)」参照																			
CommandCanceled	○																			
RemoveNum							○													

機能

接続先 FTP サーバ「ConnectSvr」の指定ディレクトリ「SvrDirName」にある指定ファイル「FileName」を削除します。
「FileName」には、ワイルドカードの指定が可能です。したがって、複数のファイルを一度に削除することができます。

削除した結果は、ファイル単位で、削除済ファイル結果 RemoveFileResult[]に格納します。削除ファイル数「RemoveNum」には、削除の対象となるファイル数を格納します。
「FileName」にワイルドカードの指定をした場合は、ワイルドカードに合致するファイル数を格納します。

ファイル削除時に異常があった場合は、「RemoveFileResult[].RemoveError」の値が TRUE になります。

「ConnectSvr」のデータ型は、構造体_sFTP_CONNECT_SVR です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	接続先 FTP サーバの各種設定パラメータ	_sFTP_CONNECT_SVR	–	–	–
Adr	アドレス	IP アドレスまたは、ホスト名 *1	STRING	1 ～ 200 バイト *2	–	–
PortNo	ポート番号	FTP サーバのコントロールコネクションの TCP ポート番号	UINT	0 ～ 65535 *3		
UserName	ユーザ名	FTP サーバのユーザ名	STRING	最大 33 バイト *4 *5		
Password	パスワード	FTP サーバのパスワード	STRING	最大 33 バイト *4 *5		

*1. ホスト名を指定時は、別途 DNS 設定または Hosts 設定が必要です。
*2. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（アンダーバー）です。
*3. 0 の場合、TCP ポート番号は 21 になります。
*4. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（ アンダーバー）です。CPU ユニット Ver.1.16 以降では、“\”（バックスラッシュ）、“@”の入力も可能です。
*5. バイト数には、終端 NULL 文字を含みます。

RemoveFileResult[]のデータ型は、構造体_sFTP_FILE_RESULT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
RemoveFileResult	削除済ファイル結果	転送ファイルの結果	_sFTP_FILE_RESULT	–	–	–
Name	ファイル名 *1	転送済ファイル名	STRING	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）	–	–
TxError	転送異常	TRUE: 転送異常終了 FALSE: 転送正常終了	BOOL	データ型に従う		
RemoveError	削除異常	TRUE: 削除異常終了 FALSE: 削除正常終了	BOOL			
Reserved	予約	システム予約	ARRAY[0..3] Of Byte	–		0

*1. 拡張子付きです。

ワイルドカードでのファイル名を指定

削除するファイル名には、ワイルドカードの指定が可能です。
ワイルドカードの指定方法は、FTPGetFile 命令と同じ仕様です。FTPGetFile 命令の「ワイルドカードでのファイル名指定 (P.2-1119)」を参照してください。

FTP サーバとの処理に関するオプション指定

FTP サーバのファイルを削除するときに、FTP 実行オプション「ExecOption」で指定した動作を行います。
オプション設定内容は、FTPGetFile 命令と同じ仕様です。FTPGetFile 命令の「FTP サーバとの処理に関するオプション指定 (P.2-1120)」を参照してください。
ただし、本命令で有効なオプションは、パッシブモード指定「ExecOption.PassiveMode」だけです。

FTP サーバとの接続処理のリトライ指定

FTP サーバとの接続処理のリトライが可能です。
リトライ設定内容および動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「FTP サーバとの接続処理のリトライ指定 (P.2-1106)」を参照してください。

命令実行の取り消し

FTPRemoveFile 命令実行中に、命令実行の取り消しが可能です。
取り消し前までに、FTP サーバからファイルを削除した結果は、「RemoveNum」および削除済ファイル結果 RemoveFileResult[]に格納されます。
取り消し動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「命令実行の取り消し (P.2-1108)」を参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

使用上の注意

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 削除済ファイルの数が、削除済ファイル結果 RemoveFileResult[]の配列要素数を超えた場合、超えた分の結果は格納しません。この場合「Error」は TRUE になりません。
- 255 文字を超えるファイル名については先頭から 255 文字分を「RemoveFileResult[].Name」に格納します。この場合「Error」は TRUE になりません。
- FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令で同時実行可能な個数は、最大 3 です。
- ワイルドカードでファイル名を指定し、複数のファイルで異常が発生した場合、「ErrorID」、「ErrorIDEx」には、RemoveFileResult[]に格納されたファイルのうち、「RemoveFileResult[].TxError」の値が TRUE である先頭のファイルの結果が格納されます。
- 次の場合、削除異常になり、「RemoveFileResult[].RemoveError」の値が TRUE になります。
  a)「FileName」で指定したファイルが、FTP サーバのディレクトリに存在しない場合
  b)「FileName」で指定したファイルの属性が読み取り専用になっていた場合
  c)「FileName」で指定したファイルが、ディレクトリだった場合
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) 入力パラメータのいずれかがの値が、有効範囲外の場合
  b)「SvrDirName」に、1 階層上のディレクトリ“..”を指定した場合
  c)「SvrDirName」に、“//”などの不正なパスを指定した場合

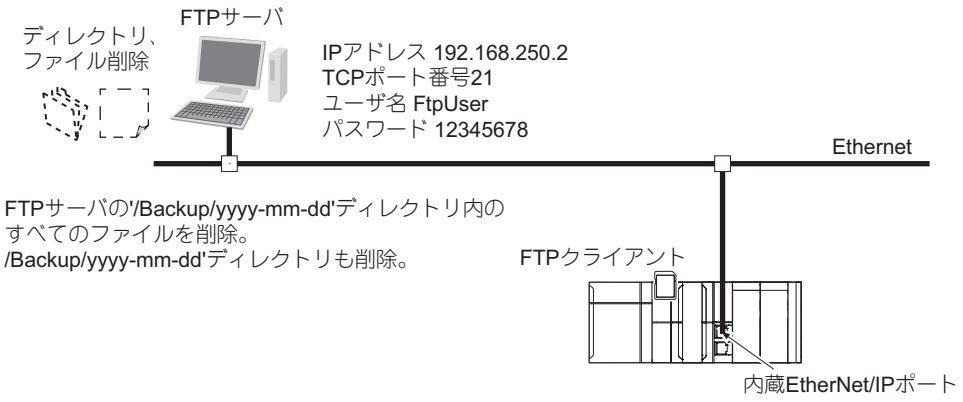
d)「SvrDirName」で指定したディレクトリが FTP サーバに存在しない場合
e)「SvrDirName」で指定したディレクトリに、1000 個以上の削除対象となるファイルが存在する場合
f)「FileName」で指定したワイルドカード含むファイル名に該当するファイルが、FTP サーバのディレクトリに存在しない場合
g)「FileName」で指定したファイルの属性が読み取り専用になっていた場合
h)「ConnectSvr」で指定した FTP サーバがネットワーク上に存在しない、またはサービスを停止している場合
i) FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令を、同時に 3 個を超えて実行した場合

- 拡張エラーコード「ErrorIDEx」は、本命令では、FTP サーバより返信された FTP レスポンスコードを表示します。代表的な「ErrorIDEx」の値、異常内容および対処方法は、以下のとおりです。詳細については、FTP サーバの仕様を確認してください。「ErrorIDEx」は、エラーコード「ErrorID」の値が WORD#16#2407 の場合に出力されます。

「ErrorIDEx」の値	異常内容	対処方法
16#000001A9	データコネクションを確立できませんでした。	インターネット上の FTP サーバとの FTP 通信の場合、FTP のオープンモードが Active になっていないか確認してください。
16#000001AA	コネクションがクローズされています。データ転送を中止しました。	FTP サーバとの接続を確認してください。 FTP サーバが停止していないか確認してください。
16#000001C2	要求されたファイル操作を行えませんでした。ファイルが、すでにオープンされている場合など、ファイルが使用できません。	対象のファイルが他アプリケーションによりオープンされていないか確認してください。
16#00000212	ユーザはログインできませんでした。	FTP ユーザ名またはパスワードを確認してください。
16#00000214	ファイル保存用のアカウントが必要です。	FTP ユーザのアクセス権を確認してください。
16#00000226	ファイルが見つからないためにアクセスできない場合など、ファイルが使用できないため、要求されたファイル操作を実行しませんでした。	指定したファイル名に該当するファイルが FTP サーバのディレクトリに存在するか確認してください。 指定したファイルのアクセス権を確認してください。
16#00000229	ファイル名が不正なため実行できませんでした。	指定したディレクトリのアクセス権を確認してください。

サンプルプログラム

FTP サーバの'/Backup/yyyy-mm-dd'ディレクトリ内のファイルをすべて削除します。その後、'/Backup/yyyy-mm-dd'ディレクトリも削除します。



コントローラと FTP サーバは、EtherNet/IP ネットワークで接続しています。FTP サーバと接続するための設定パラメータの値は、以下のとおりとします。

設定パラメータ	値
IP アドレス	192.168.250.2
TCP ポート番号	21
ユーザ名	FtpUser
パスワード	12345678

処理の手順は、以下のようになります。

1 **FTPRemoveFile 命令を使用して、FTP サーバの'/Backup/yyyy-mm-dd' ディレクトリ内のすべてのファイルを削除します。削除ファイル名は、ワイルドカード指定で'*.*'とします。**

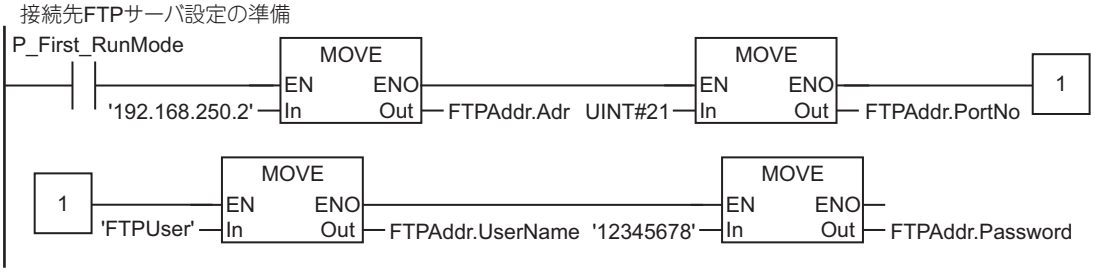
2 **FTPRemoveDir 命令を使用して、FTP サーバの'/Backup/yyyy-mm-dd' ディレクトリを削除します。**

3 **すべての処理が正常に終了した場合は正常終了時処理を実行します。異常が発生した場合は異常終了時処理を実行します。**

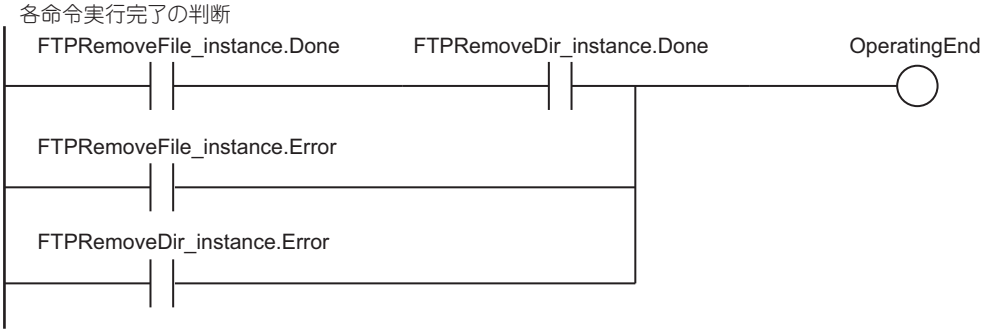
LD

内部変数	名称	データ型	初期値	コメント
	FTPRemoveFile_instance	FTPRemoveFile		FTPRemoveFile 命令のインスタンス
	FTPRemoveDir_instance	FTPRemoveDir		FTPRemoveDir 命令のインスタンス
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定
	RemoveResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	削除済みファイル結果
	RS_instance	RS		RS 命令のインスタンス
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中

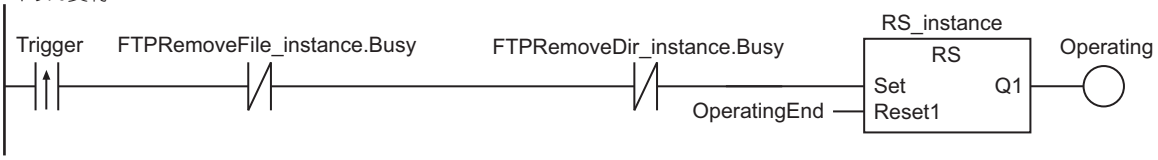
接続先FTPサーバ設定の準備



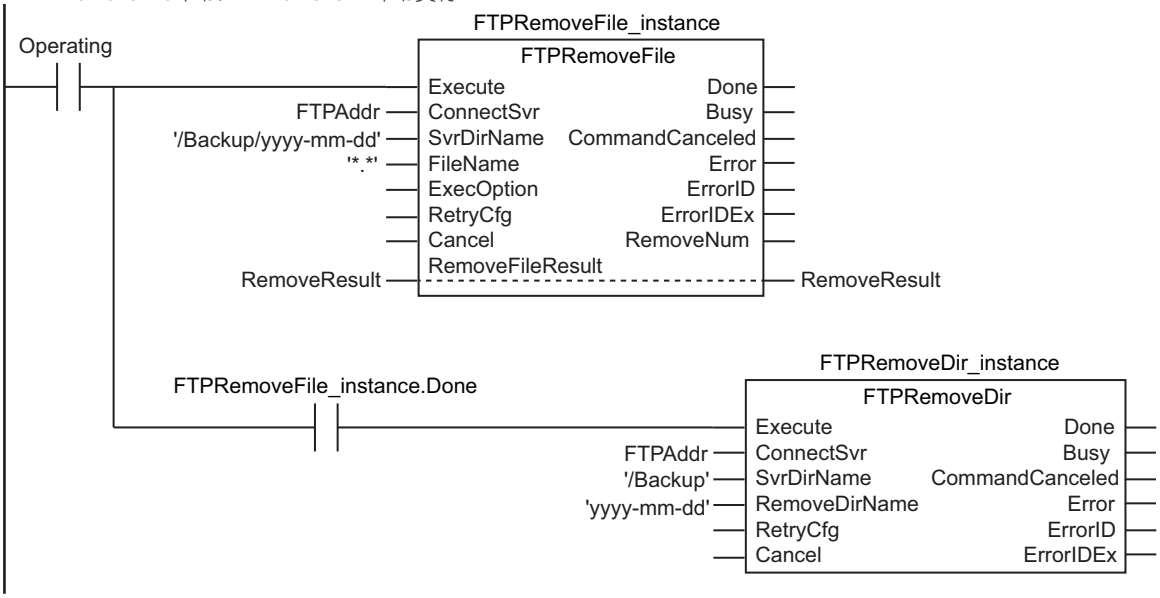
各命令実行完了の判断



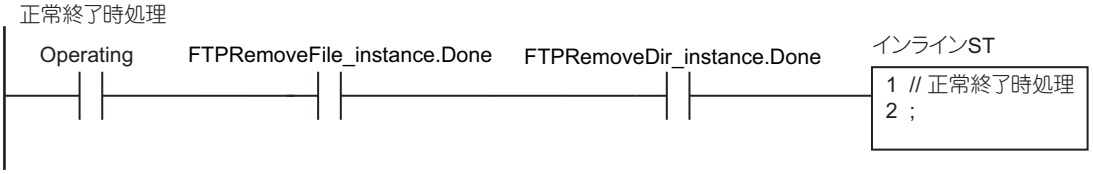
トリガ受付

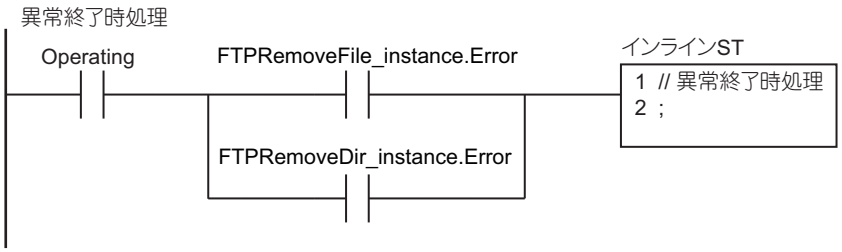


FTPRemoveFile命令、FTPRemoveDir命令実行



正常終了時処理





ST

内部変数	名称	データ型	初期値	コメント
	R_TRIG_instance	R_TRIG		R_TRIG 命令のインスタンス
	UP_Q	BOOL	FALSE	トリガ出力
	FTPRemoveFile_instance	FTPRemoveFile		FTPRemoveFile 命令のインスタンス
	FTPRemoveDir_instance	FTPRemoveDir		FTPRemoveDir 命令のインスタンス
	DoFTPTrigger	BOOL	FALSE	FTPRemoveFile、FTPRemoveDir 実行条件
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	接続先 FTP サーバ設定
	RemoveResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	削除済みファイル結果
	Stage	UINT	0	命令実行手順
	Trigger	BOOL	FALSE	実行条件

```
// 接続先 FTP サーバ設定の準備
IF P_First_RunMode THEN
   FTPAddr.Adr      := '192.168.250.2';// アドレス
   FTPAddr.PortNo   := UINT#21;        // ポート番号
   FTPAddr.UserName := 'FtpUser';      // ユーザー名
   FTPAddr.Password := '12345678';     // パスワード
END_IF;

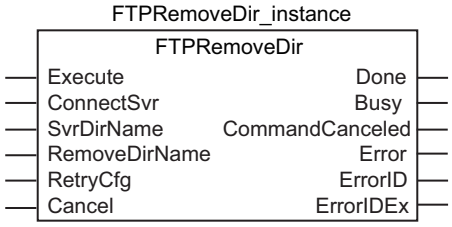
// トリガ受付
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (FTPRemoveFile_instance.Busy = FALSE) AND
      (FTPRemoveDir_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   FTPRemoveFile_instance( // インスタンス初期化
      Execute          := FALSE,
      ConnectSvr       := FTPAddr,
      SvrDirName       := '/Backup/yyyy-mm-dd',
      FileName         := '*.*',
      RemoveFileResult := RemoveResult) ;
   FTPRemoveDir_instance( // インスタンス初期化
```

```
      Execute       := FALSE,
      ConnectSvr    := FTPAddr,
      SvrDirName    := '/Backup',
      RemoveDirName := 'yyyy-mm-dd') ;
END_IF;

IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1 : // FTPRemoveFile 命令実行
         FTPRemoveFile_instance(
            Execute          := TRUE,                  // 起動
            ConnectSvr       := FTPAddr,               // 接続先 FTP サーバ
            SvrDirName       := '/Backup/yyyy-mm-dd', // FTP サーバディレクトリ名
            FileName         := '*.*',                 // ファイル名
            RemoveFileResult := RemoveResult) ;        // 削除ファイル結果
         IF (FTPRemoveFile_instance.Done = TRUE) THEN
            Stage := INT#2;  // 次へ
         ELSIF (FTPRemoveFile_instance.Error = TRUE) THEN
            Stage := INT#10; // 異常終了
         END_IF;
      2 : // FTPRemoveDir 命令実行
         FTPRemoveDir_instance(
            Execute       := TRUE,           // 起動
            ConnectSvr    := FTPAddr,        // 接続先 FTP サーバ
            SvrDirName    := '/Backup',      // FTP サーバディレクトリ名
            RemoveDirName := 'yyyy-mm-dd') ;// 削除ディレクトリ名
         IF (FTPRemoveDir_instance.Done = TRUE) THEN
            Stage:=INT#0;  // 正常終了
         ELSIF (FTPRemoveDir_instance.Error = TRUE) THEN
            Stage:=INT#20; // 異常終了
         END_IF;
      0 :  // 正常終了処理
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
      ELSE // 異常終了処理
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
   END_CASE;
END_IF;
```

FTPRemoveDir

FTP サーバのディレクトリを削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPRemoveDir	FTP サーバのディレクトリ削除	FB		FTPRemoveDir_instance(Execute, ConnectSvr, SvrDirName, RemoveDirName, Cancel, RetryCfg, Done,Busy, CommandCanceled, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	入力	接続先 FTP サーバの各種設定パラメータ	－	－	*1
SvrDirName	FTP サーバディレクトリ名		削除するディレクトリが存在する FTP サーバのディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*2*3		" *4
RemoveDirName	削除ディレクトリ名		削除するディレクトリ名	最大 256 バイト（半角英数字 255 文字+終端 NULL 文字）*3		*1
RetryCfg	実行リトライ設定		命令実行リトライ設定	－		－
Cancel	取り消し		TRUE: 命令実行を取り消す FALSE: 命令実行を取り消さない	データ型に従う		FALSE
CommandCanceled	取り消し完了	出力	TRUE: 取り消し完了 FALSE: 取り消し未完了	－	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
2. FTP サーバディレクトリ名として、以下の文字は使用できません。""、"?"、"<"、">"、"¦"、"""（ダブルクォーテーション）
*3. 半角スペースは、FTP サーバにより動作が変わることがありますので使用を推奨しません。
半角スペースを使用する場合は、2 文字として扱います。
*4. FTP サーバの、ログイン時のホームディレクトリです。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	構造体_sFTP_CONNECT_SVR　詳細は「機能 (P.2-1145)」参照																			
SvrDirName																				○
RemoveDirName																				○
RetryCfg	構造体_sFTP_RETRY_CFG　詳細は「FTP サーバとの接続処理のリトライ指定 (P.2-1122)」参照																			
Cancel	○																			
CommandCanceled	○																			

機能

接続先 FTP サーバアドレス「ConnectSvr」の削除するディレクトリが存在するディレクトリ「SvrDirName」にある指定ディレクトリ「RemoveDirName」を削除します。

命令が正常終了「Done」の値が TRUE になった時点で、指定ディレクトリの削除が完了しています。ディレクトリ削除に失敗すると、異常終了「Error」の値が TRUE になります。

「ConnectSvr」のデータ型は、構造体_sFTP_CONNECT_SVR です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ConnectSvr	接続先 FTP サーバ設定	接続先 FTP サーバの各種設定パラメータ	_sFTP_CONNECT_SVR	–	–	–
Adr	アドレス	IP アドレスまたは、ホスト名 *1	STRING	1 ～ 200 バイト *2	–	–
PortNo	ポート番号	FTP サーバのコントロールコネクションの TCP ポート番号	UINT	0 ～ 65535 *3		
UserName	ユーザ名	FTP サーバのユーザ名	STRING	最大 33 バイト *4 *5		
Password	パスワード	FTP サーバのパスワード	STRING	最大 33 バイト *4 *5		

*1. ホスト名を指定時は、別途 DNS 設定または Hosts 設定が必要です。
*2. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（アンダーバー）です。
*3. 0 の場合、TCP ポート番号は 21 になります。
*4. 使用できる文字は、半角で、“A-Z”、“a-z”、“0-9”、“-”（ハイフン）、“.”（ピリオド）、“_”（ アンダーバー）です。CPU ユニット Ver.1.16 以降では、“\”（バックスラッシュ）、“@”の入力も可能です。
*5. バイト数には、終端 NULL 文字を含みます。

FTP サーバとの接続処理のリトライ指定

FTP サーバとの接続処理のリトライが可能です。
リトライ設定内容および動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「FTP サーバとの接続処理のリトライ指定 (P.2-1106)」を参照してください。

命令実行の取り消し

FTPRemoveDir 命令実行中に、命令実行の取り消しが可能です。
取り消し動作は、FTPGetFileList 命令と同じ仕様です。FTPGetFileList 命令の「命令実行の取り消し (P.2-1108)」を参照してください。

関連するシステム定義変数

変数名	名称	データ型	内容
_EIP1_EtnOnlineSta *1	オンライン	BOOL	内蔵 EtherNet/IP ポートの通信機能が利用可能になっていることを示します。 TRUE: 通信可 FALSE: 通信不可
_EIPIn1_EtnOnlineSta *2			

*1. NY シリーズ コントローラを使用する場合の変数名です。
_EIP1_EtnOnlineSta の代わりに_EIP_EtnOnlineSta を指定することも可能です。
*2. NY シリーズ コントローラの内部通信ポートを使用する場合の変数名です。

使用上の注意

- 本命令は、NY シリーズ コントローラの内蔵 EtherNet/IP ポートで、使用可能です。
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令を「Cancel」によって命令実行を取り消しさせた場合、「Cancel」のタイミングによっては、FTP サーバのディレクトリが削除されていることがあります。FTP サーバのディレクトリを確認してください。
- FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令で同時実行可能な個数は、最大 3 です。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) 入力パラメータのいずれかがの値が、有効範囲外の場合
  b) 「SvrDirName」で指定したディレクトリが FTP サーバに存在しない場合
  c) 「SvrDirName」、「RemoveDirName」に、1 階層上のディレクトリ“..”を指定した場合
  d) 「SvrDirName」、「RemoveDirName」に、“//”などの不正なパスを指定した場合
  e) 「RemoveDirName」で指定したディレクトリが、FTP サーバに存在しない場合
  f) 「RemoveDirName」で指定したディレクトリに、ファイルまたはサブディレクトリが存在する場合
  g) 「RemoveDirName」で指定したディレクトリの属性が、読み取り専用になっていた場合
  h) 「ConnectSvr」で指定した FTP サーバがネットワーク上に存在しない、またはサービスを停止している場合
  i) FTPGetFileList 命令、FTPGetFile 命令、FTPPutFile 命令、FTPRemoveFile 命令、FTPRemoveDir 命令を、同時に 3 個を超えて実行した場合
- 拡張エラーコード「ErrorIDEx」は、本命令では、FTP サーバより返信された FTP レスポンスコードを表示します。代表的な「ErrorIDEx」の値、異常内容および対処方法は、以下のとおりです。詳細については、FTP サーバの仕様を確認してください。「ErrorIDEx」は、エラーコード「ErrorID」の値が WORD#16#2407 の場合に出力されます。

「ErrorIDEx」の値	異常内容	対処方法
16#000001A9	データコネクションを確立できませんでした。	インターネット上の FTP サーバとの FTP 通信の場合、FTP のオープンモードが Active になっていないか確認してください。
16#000001AA	コネクションがクローズされています。データ転送を中止しました。	FTP サーバとの接続を確認してください。 FTP サーバが停止していないか確認してください。
16#000001C2	要求されたファイル操作を行えませんでした。ファイルが、すでにオープンされている場合など、ファイルが使用できません。	対象のファイルが他アプリケーションによりオープンされていないか確認してください。
16#00000212	ユーザはログインできませんでした。	FTP ユーザ名またはパスワードを確認してください。
16#00000214	ファイル保存用のアカウントが必要です。	FTP ユーザのアクセス権を確認してください。
16#00000226	ファイルが見つからないためにアクセスできない場合など、ファイルが使用できないため、要求されたファイル操作を実行しませんでした。	指定したファイル名に該当するファイルが FTP サーバのディレクトリに存在するか確認してください。 指定したファイルのアクセス権を確認してください。
16#00000229	ファイル名が不正なため実行できませんでした。	指定したディレクトリのアクセス権を確認してください。

サンプルプログラム

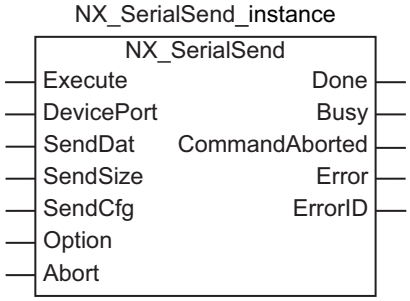
FTPRemoveFile 命令の、「サンプルプログラム (P.2-1139)」を参照してください。

シリアル通信命令

命令	名称	ページ
NX_SerialSend	無手順データの送信	P.2-1150
NX_SerialRcv	無手順データの受信	P.2-1161
NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	P.2-1174
NX_ModbusRtuRead	Modbus RTU Read コマンド送信	P.2-1184
NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	P.2-1195
NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	P.2-1206
NX_SerialBufClear	バッファクリア	P.2-1213
NX_SerialStartMon	シリアルラインモニタの開始	P.2-1222
NX_SerialStopMon	シリアルラインモニタの停止	P.2-1226

NX_SerialSend

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、無手順でデータを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialSend	無手順データの送信	FB		NX_SerialSend_instance(Execute, DevicePort, SendDat, SendSize, SendCfg, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	－	－	－
SendDat[] 配列	送信データ配列		送信データ配列	データ型に従う	－	*1
SendSize	送信データサイズ		送信データサイズ	0 ～ 4096	バイト	0
SendCfg	送信データ付加条件		送信データ付加条件	－	－	－
Option	オプション		オプション	－	－	－
Abort	中断		命令実行を中断	データ型に従う	－	FALSE
Command Aborted	中断完了	出力	中断完了	データ型に従う	－	－

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1151)」参照																			
SendDat[] 配列		○																		
SendSize							○													
SendCfg	構造体_sSERIAL_CFG　詳細は「機能 (P.2-1151)」参照																			
Option	構造体_sSERIAL_SEND_OPTION　詳細は「機能 (P.2-1151)」参照																			
Abort	○																			
CommandAborted	○																			

機能

NX シリーズ コミュニケーションインタフェースユニットの指定したポートから、無手順でデータを送信します。

入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	－	－	－
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	－	－
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	－	－	－
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	－	－	－
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	－	－	－
Reserved	予約	予約	予約	－	－	－
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	－	－

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2
NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

入力変数「SendDat」で指定したデータのうち、入力変数「SendSize」で指定したサイズを送信します。
このとき、「SendSize」の値が「0」の場合は送信しません。命令実行時に「Busy」は TRUE にならず、「Done」の値が TRUE になります。

送信データに、スタートコードおよびエンドコードを付加する場合は、入力変数「SendCfg」で設定します。
入力変数「SendCfg」のデータ型は、構造体_sSERIAL_CFG です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SendCfg	送信データ付加条件	送信データ付加条件	_sSERIAL_CFG	－	－	－
StartTrig	スタートコード有無	スタートコード有無	_eSERIAL_START	_SERIAL_START_NONE _SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	－	_SERIAL_START_NONE
StartCode	スタートコード	スタートコード	BYTE[2]	データ型に従う	－	[2(16#0)]
EndTrig	エンドコード有無	エンドコード有無	_eSERIAL_END	_SERIAL_END_NONE _SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2 _SERIAL_END_TERMINATION_CHAR _SERIAL_END_RCV_SIZE	－	_SERIAL_END_NONE
EndCode	エンドコード	エンドコード	BYTE[2]	データ型に従う	－	[2(16#0)]
RcvSizeCfg	受信サイズ	本命令では不使用	UINT	0 ～ 4096	バイト	0

「StartTrig」のデータ型は、列挙型_eSERIAL_START です。
列挙型_eSERIAL_START の、列挙子の意味は以下のとおりです。

列挙子	意味
_SERIAL_START_NONE	なし
_SERIAL_START_STARTCODE1	1 バイトコード
_SERIAL_START_STARTCODE2	2 バイトコード

「EndTrig」のデータ型は、列挙型_eSERIAL_END です。
列挙型_eSERIAL_END の、列挙子の意味は以下のとおりです。

列挙子	意味
_SERIAL_END_NONE	なし
_SERIAL_END_ENDCODE1	1 バイトコード
_SERIAL_END_ENDCODE2	2 バイトコード
_SERIAL_END_TERMINATION_CHAR	休止条件
_SERIAL_END_RCV_SIZE	受信サイズ

スタートコードとエンドコードの動作の詳細は、「スタートコードとエンドコードの動作 (P.2-1164)」を参照してください。

コントローラから NX シリーズ コミュニケーションインタフェースユニットへの送信を遅らせる場合は、入力変数「Option.SendDelay」で遅らせる時間を 0.01s 単位で指定します。
入力変数「Option」のデータ型は、構造体_sSERIAL_SEND_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sSERIAL_SEND_OPTION	–	–	–
SendDelay	送信ディレー時間	送信ディレー時間の指定	UINT	データ型に従う	0.01s	0

使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

スタートコードとエンドコードの動作

「SendCfg.StartTrig」と「SendCfg.EndTrig」で送信データに付加するスタートコードとエンドコードの条件を指定します。
送信データにスタートコード、エンドコードを付加する場合、入力変数「SendSize」には、スタートコード、エンドコードを含まない値を設定してください。
「StartTrig」と「EndTrig」の動作を次に示します。

「StartTrig」の値	動作
_SERIAL_START_NONE	–
_SERIAL_START_STARTCODE1	「SendDat」の先頭にスタートコードを付加して送信 例: STX
_SERIAL_START_STARTCODE2	

「EndTrig」の値	動作
_SERIAL_END_NONE	–
_SERIAL_END_ENDCODE1	「SendDat」の終端にエンドコードを付加して送信 例: ETX
_SERIAL_END_ENDCODE2	
_SERIAL_END_TERMINATION_CHAR	異常
_SERIAL_END_RCV_SIZE	異常

命令実行の中断

命令実行中に「Abort」を TRUE にすると、命令実行を中断します。
命令実行を中断すると「CommandAborted」が TRUE となります。送信の途中でも命令を中断します。
命令実行の中断が間に合わなかった場合は、「Done」が TRUE となり、命令が正常終了します。

「Abort」と「Execute」の両方を TRUE にした場合は、「CommandAborted」が TRUE となります。

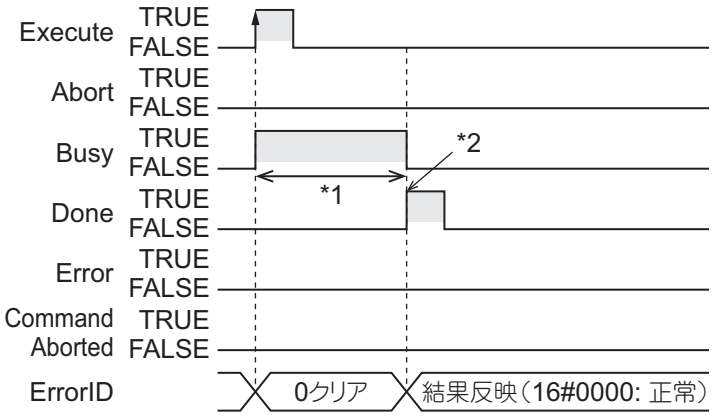
中断の動作は、「Busy」処理を終了させるだけで、送信バッファをクリアしません。バッファのクリアが必要な場合、「NX_SerialBufClear (P.2-1213)」命令を使用してください。

タイミングチャート

タイミングチャートを、以下に示します。

● 正常終了時（「SendDelay」が「0」(0s)の場合）

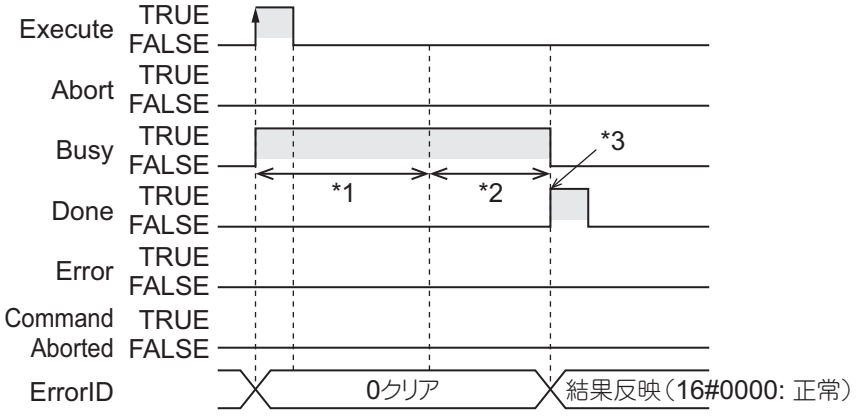
「SendDelay」が「0」(0s)の場合は、次の動作になります。



*1. 送信処理
*2. 送信完了

● 正常終了時（「SendDelay」が「100」(1s)の場合）

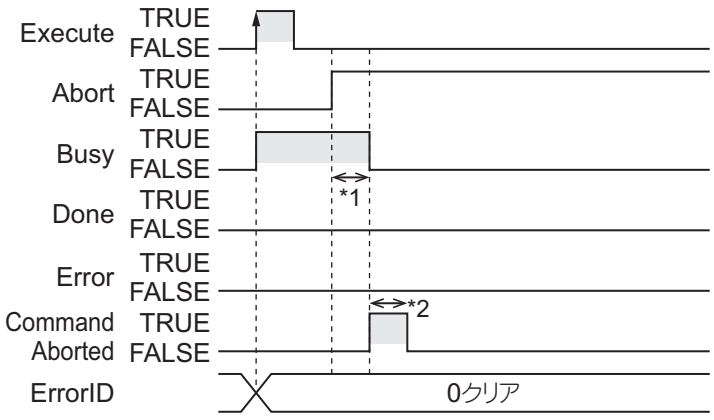
「SendDelay」が「100」(1s)の場合は、次の動作になります。



*1. 送信ディレー時間 1s
*2. 送信処理
*3. 送信完了

● 中断実行時（「Busy」が TRUE の場合）

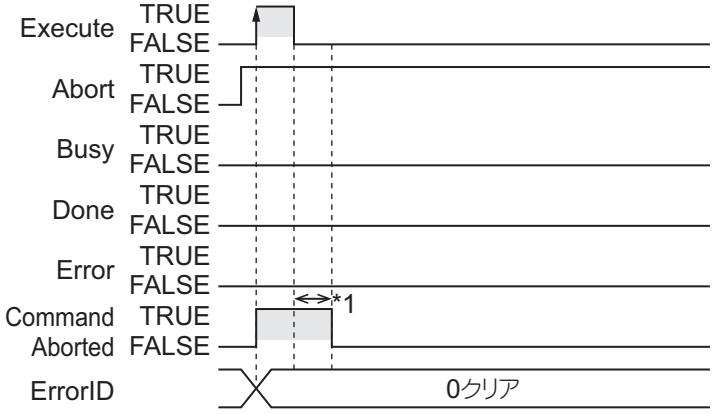
「Busy」が TRUE の状態で、「Abort」を TRUE にする場合は、次の動作になります。



*1. 中断処理
*2. 1 タスク周期後に FALSE

● 中断実行時（「Execute」が TRUE の場合）

「Abort」と「Execute」の両方を TRUE にする場合は、次の動作になります。



*1. 1 タスク周期後に FALSE

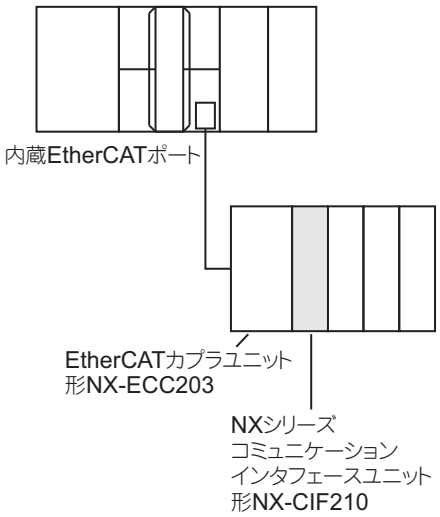
使用上の注意

- 「Abort」が FALSE のままの場合、本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。
  処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。命令実行中に「Abort」が TRUE になった場合は、「CommandAborted」または「Done」が TRUE になります。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「SendSize」、「SendCfg.StartTrig」、「SendCfg.EndTrig」、「DevicePort.DevicePortType」、「DevicePort.PortNo」の値が、有効範囲外の場合。
  b)「SendDat」に指定した配列変数が、「SendSize」で指定したサイズより小さい場合。

c）「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
d）「DevicePort」でデータ型が不正な場合。
e）NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
f）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
対象となる命令は、NX_SerialSend 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusWrite 命令です。
g）NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

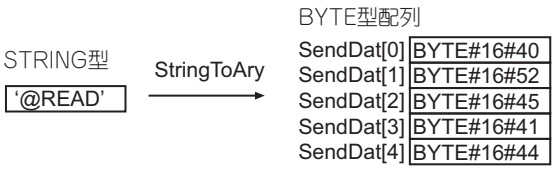
サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。



形 NX-CIF210 のシリアルポート 2 に接続したバーコードリーダに、無手順コマンドを送信します。
送信するコマンドは、シーン番号取得コマンド'@READ'です。

送信するコマンドは、StringToAry 命令を使用し、文字列'@READ'を 1 文字ずつの文字コードにして SendDat[]の各配列要素に格納しておきます。



スタートコードはなし、エンドコードは 16#0D(CR)とします。

形 NX-CIF210 の設定は、以下のとおりとします。

項目名	設定値
ポート 2: 伝送速度	38400bps
ポート 2: データ長	8 ビット
ポート 2: パリティ	なし

項目名	設定値
ポート 2: ストップビット	1 ビット
ポート 2: フロー制御	なし

グローバル変数の定義

● グローバル変数

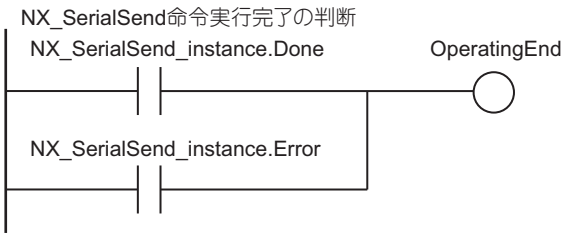
名称	データ型	割付先	コメント
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT:// node#1/NX Unit I/O Data Active Status 125	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

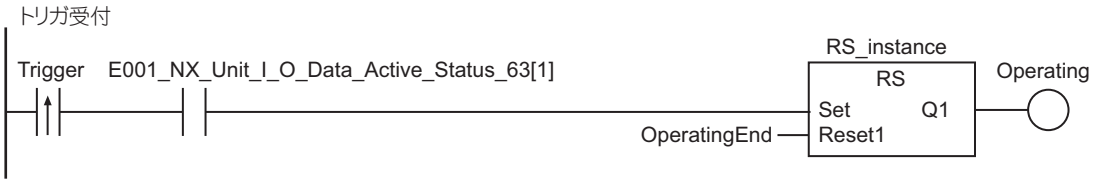
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

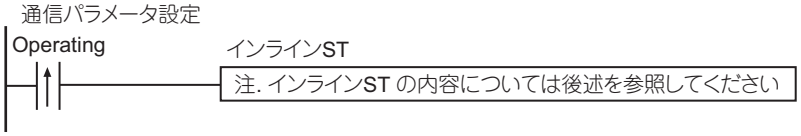
LD

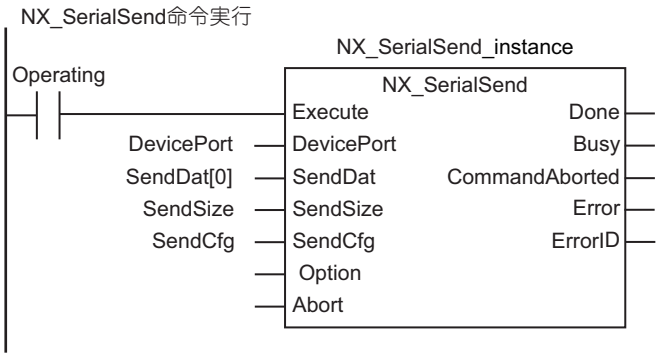
内部変数	名称	データ型	初期値	コメント
	OperationEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	SendDat	ARRAY [0..5] OF BYTE	[6(16#0)]	送信データ
	SendSize	UINT	0	送信データサイズ
	RS_instance	RS		
	NX_SerialSend_instance	NX_SerialSend		
	SendCfg	_sSERIAL_CFG		
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	スタートコードなし
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	エンドコードあり
	EndCode	BYTE[2]	[16#0D,16#00]	16#0D(CR)

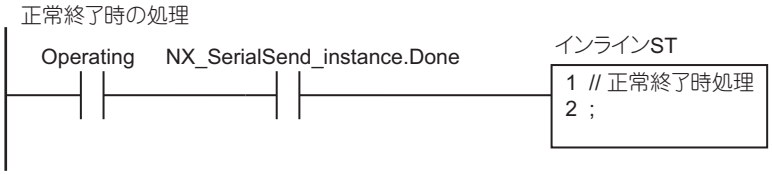
外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

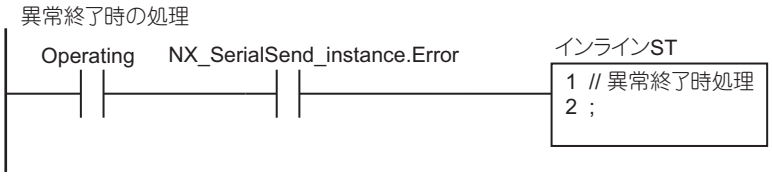












● インライン ST の内容

```
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

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	SendDat	ARRAY [0..5] of BYTE	[6(16#0)]	送信データ
	SendSize	UINT	0	送信データサイズ
	NX_SerialSend_instance	NX_SerialSend		
	SendCfg	_sSERIAL_CFG		
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	スタートコードなし
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	エンドコードあり
	EndCode	BYTE[2]	[16#0D, 16#00]	16#0D(CR)

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

```
// Trigger の立ち上がり検出
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;

// 通信パラメータ設定と NX_SerialSend 命令初期化
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;

// NX_SerialSend 命令実行
IF (Operating=TRUE) THEN
        NX_SerialSend_instance(
                        Execute:=TRUE,
                        DevicePort:=DevicePort, // ポート指定
                        SendDat:=SendDat[0],    // 送信データ
                        SendSize:=SendSize,     // 送信データサイズ
                        SendCfg:=SendCfg);      // エンドコード指定
        IF (NX_SerialSend_instance.Done=TRUE) THEN
                // 正常終了時処理

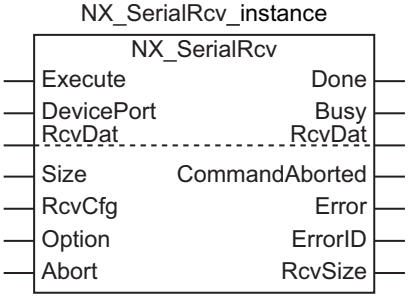
                Operating:=FALSE;
        END_IF;

        IF (NX_SerialSend_instance.Error=TRUE) THEN
                // 異常終了時処理

                Operating:=FALSE;
        END_IF;
END_IF;
```

NX_SerialRcv

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、無手順でデータを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialRcv	無手順データの受信	FB		NX_SerialRcv_instance(Execute, DevicePort, RcvDat, Size, RcvCfg, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, RcvSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	－	－	－
Size	格納サイズ		「RcvDat」のサイズをバイト数で指定	1 ～ 4096	バイト	1
RcvCfg	受信完了設定		受信完了設定	－	－	－
Option	オプション		オプション	－	－	－
Abort	中断		命令実行の中断	－	－	FALSE
RcvDat[]配列	受信データ	入出力	受信バッファから受信するデータを格納する変数	データ型に従う	－	－
CommandAborted	中断完了	出力	中断完了	データ型に従う	－	－
RcvSize	受信サイズ		受信バッファから実際に受信したデータのサイズ	0 ～ 4096	バイト	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1162)」参照																			
Size							○													
RcvCfg	構造体_sSERIAL_CFG　詳細は「機能 (P.2-1162)」参照																			
Option	構造体_sSERIAL_RCV_OPTION　詳細は「機能 (P.2-1162)」参照																			
Abort	○																			
RcvDat[]配列		○																		
CommandAborted	○																			
RcvSize							○													

機能

NX シリーズ コミュニケーションインタフェースユニットの指定したポートから、無手順でデータを読み出します。

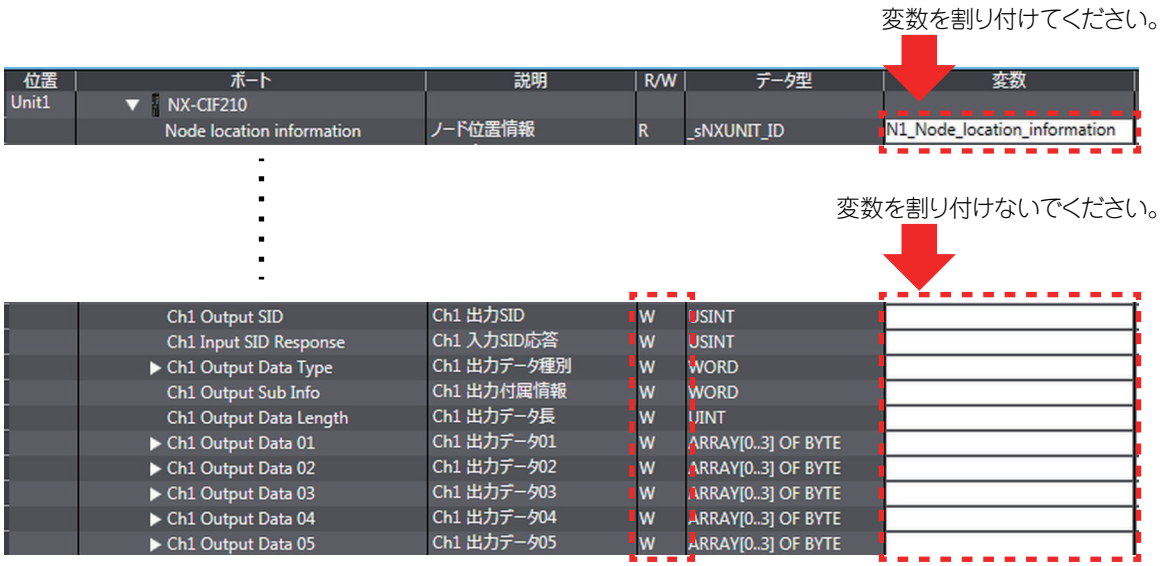
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	–	–	–
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	–	–
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	–	–	–
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	–	–	–
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	–	–	–
Reserved	予約	予約	予約	–	–	–
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	–	–

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2
NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

ユニットに受信したデータは、まず受信バッファに格納されます。
入出力変数「RcvDat」で、受信バッファから受信するデータを格納する変数を指定します。
入力変数「Size」で「RcvDat」のサイズをバイト数で指定します。
受信バッファから実際に受信したデータのサイズは、出力変数「RcvSize」で表します。

受信データが、スタートコードおよびエンドコードを含んでいる場合は、入力変数「RcvCfg」で設定します。
入力変数「RcvCfg」のデータ型は、構造体_sSERIAL_CFG です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
RcvCfg	受信完了設定	受信完了設定	_sSERIAL_CFG	–	–	–
StartTrig	スタートコード有無	スタートコード有無	_eSERIAL_START	_SERIAL_START_NONE _SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	–	_SERIAL_START_NONE
StartCode	スタートコード	スタートコード	BYTE[2]	データ型に従う	–	[2(16#0)]
EndTrig	エンドコード有無	エンドコード有無	_eSERIAL_END	_SERIAL_END_NONE _SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2 _SERIAL_END_TERMINATION_CHAR _SERIAL_END_RCV_SIZE	–	_SERIAL_END_NONE
EndCode	エンドコード	エンドコード	BYTE[2]	データ型に従う	–	[2(16#0)]
RcvSizeCfg	受信サイズ	エンドコードが_SERIAL_END_RCV_SIZE の場合、受信サイズを指定	UINT	0 ～ 4096	バイト	0

「StartTrig」のデータ型は、列挙型_eSERIAL_START です。
列挙型_eSERIAL_START の、列挙子の意味は以下のとおりです。

列挙子	意味
_SERIAL_START_NONE	なし

列挙子	意味
_SERIAL_START_STARTCODE1	1 バイトコード
_SERIAL_START_STARTCODE2	2 バイトコード

「EndTrig」のデータ型は、列挙型_eSERIAL_END です。
列挙型_eSERIAL_END の、列挙子の意味は以下のとおりです。

列挙子	意味
_SERIAL_END_NONE	なし
_SERIAL_END_ENDCODE1	1 バイトコード
_SERIAL_END_ENDCODE2	2 バイトコード
_SERIAL_END_TERMINATION_CHAR	休止条件
_SERIAL_END_RCV_SIZE	受信サイズ

スタートコードとエンドコードの動作の詳細は、「スタートコードとエンドコードの動作 (P.2-1164)」を参照してください。

オプションを指定する場合は、入力変数「Option」で指定します。
入力変数「Option」のデータ型は、構造体_sSERIAL_RCV_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sSERIAL_RCV_OPTION	–	–	–
TimeOut *1	タイムアウト時間	タイムアウト時間	UINT	データ型に従う	0.1s	20
LastDatRcv (Reserved)	最終データ受信	最終データ受信	BOOL	FALSE *2	–	FALSE
ClearBuf (Reserved)	受信バッファクリア条件	受信バッファクリア条件	BOOL	データ型に従う *3	–	FALSE

*1. 指定した時間内に正常終了しない場合は、異常となります。
また、「TimeOut」を「0」に指定した場合は、処理完了をいつまでも待ち続けます。
*2. 常に FALSE としてください。
*3. TRUE/FALSE のどちらを指定しても受信バッファクリアは、実行されません。

使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

スタートコードとエンドコードの動作

入力変数「RcvCfg.StartTrig」で受信データにスタートコードの条件を指定し、入力変数「RcvCfg.EndTrig」で受信データにエンドコードの条件を指定します。
「StartTrig」と「EndTrig」の組み合わせによる動作を次に示します。

StartTrig	EndTrig	動作
_SERIAL_START_NONE	_SERIAL_END_NONE	受信バッファにあるデータを受信します。受信バッファ内に受信データがない場合、出力変数「RcvSize」を0バイトと出力して受信命令は正常終了します。 この条件を指定すると、受信バッファにたまっているデータを格納サイズ分読み出します。
	_SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2	受信バッファ内のデータを、先頭からエンドコードまで受信処理をします。 例: ETX
	_SERIAL_END_TERMINATION_CHAR	受信バッファ内のデータを、先頭から終端検出したデータまで受信処理をします。*1
	_SERIAL_END_RCV_SIZE	受信バッファ内のデータを、先頭から「RcvSizeCfg」で指定した受信サイズ分の受信処理をします。 バッファに受信サイズ分のデータがたまるまで待ちます。
_SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	_SERIAL_END_NONE	スタートコードから受信バッファにあるデータをすべて受信します。
	_SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2	スタートコードからエンドコードまで受信処理をします。 例: ETX
	_SERIAL_END_TERMINATION_CHAR	スタートコードから終端を検出したデータまで受信処理をします。*1
	_SERIAL_END_RCV_SIZE	スタートコードから「RcvSize」で指定した受信サイズ分のデータを受信処理します。 バッファに受信サイズ分のデータがたまるまで待ちます。

*1. コミュニケーションインタフェースユニットの終端検出するキャラクタ数を「0（終端検出をしない）」にすると、入力変数「Size」で指定した格納サイズ分を受信するまで受信を完了しません。

受信データ格納先が足りない場合の動作

入力変数「Size」で指定された受信データ格納先のサイズが受信データよりも小さい場合は、スタートコード、エンドコードの組み合わせによって動作が以下のように異なります。

StartTrig	EndTrig	動作
_SERIAL_START_NONE	_SERIAL_END_NONE	正常完了
	_SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2	受信するが異常で完了 例: ETX
	_SERIAL_END_TERMINATION_CHAR	受信するが異常で完了
	_SERIAL_END_RCV_SIZE	• 入力値チェックエラーによる異常で完了 • データの受信は不可

StartTrig	EndTrig	動作
_SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	_SERIAL_END_NONE	受信するが異常で完了
	_SERIAL_END_ENDCODE1	受信するが異常で完了 例: ETX
	_SERIAL_END_ENDCODE2	
	_SERIAL_END_TERMINATION_CHAR	受信するが異常で完了
	_SERIAL_END_RCV_SIZE	• 入力値チェックエラーによる異常で完了 • データの受信は不可

受信データ格納先「RcvDat」分を受信処理し、格納できない分は受信バッファ内に保持します。
保持した受信データは、次回の SerialRcv 命令にて受信処理が可能です。

たとえば、受信バッファに 10 バイトあり、受信データ格納先「RcvDat」が 5 バイトの場合、5 バイト分を受信処理し、残りの 5 バイトは受信バッファに保持します。出力変数「RcvSize」は、格納したデータのサイズである 5 バイト分を出力します。

受信バッファ		受信データ格納先 RcvDat[ ]
第 1 バイト	受信処理します。	1
第 2 バイト		2
第 3 バイト		3
第 4 バイト		4
第 5 バイト		5
第 6 バイト	「RcvDat」に格納できないので、受信バッファに保持します。 次回の NX_SerialRcv 命令で受信処理します。	−
第 7 バイト		
第 8 バイト		
第 9 バイト		
第 10 バイト		

命令実行の中断

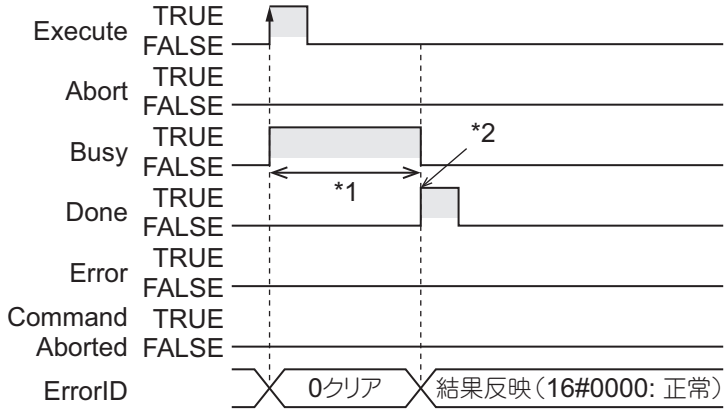
命令実行中に「Abort」を TRUE にすると、命令実行を中断します。
命令実行を中断すると「CommandAborted」が TRUE となります。送信の途中でも命令を中断します。
命令実行の中断が間に合わなかった場合は、「Done」が TRUE となり、命令が正常終了します。
「Abort」と「Execute」の両方を TRUE にした場合は、「CommandAborted」が TRUE となります。

中断の動作は、「Busy」処理を終了させるだけで、送信バッファをクリアしません。バッファのクリアが必要な場合、「NX_SerialBufClear (P.2-1213)」命令を使用してください。

タイミングチャート

タイミングチャートを、以下に示します。

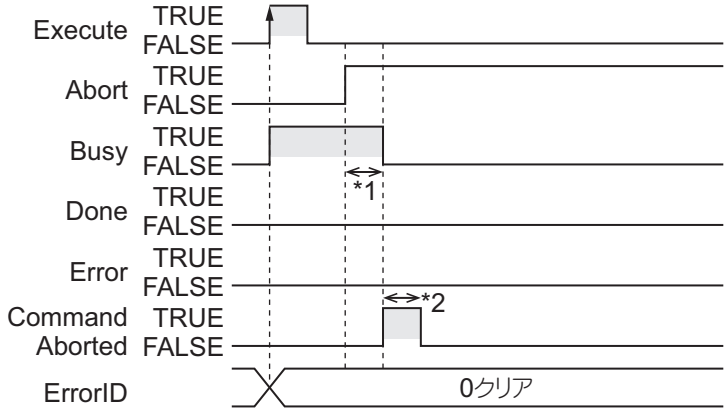
● 正常終了時



*1. 受信処理
*2. 無手順でデータを受信

● 中断実行時（「Busy」が TRUE の場合）

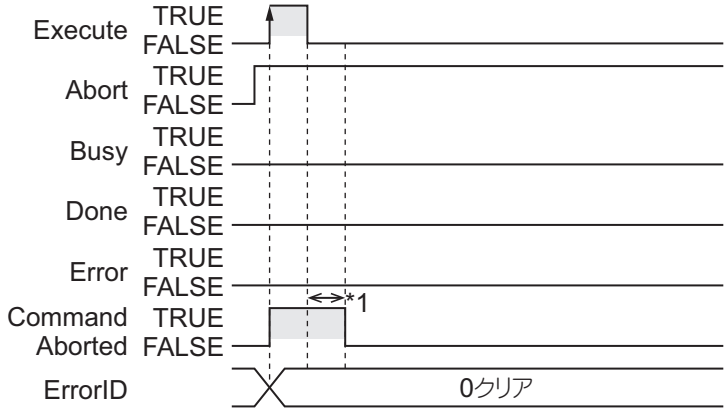
「Busy」が TRUE の状態で、「Abort」を TRUE にする場合は、次の動作になります。



*1. 中断処理
*2. 1 タスク周期後に FALSE

● 中断実行時（「Execute」が TRUE の場合）

「Abort」と「Execute」の両方を TRUE にする場合は、次の動作になります。



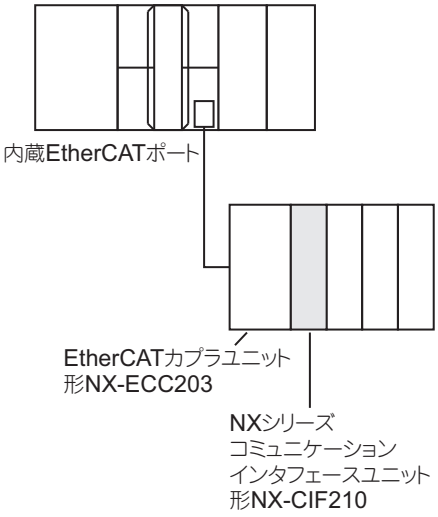
*1. 1 タスク周期後に FALSE

使用上の注意

- 「Abort」が FALSE のままの場合、本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。
  処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。命令実行中に「Abort」が TRUE になった場合は、「CommandAborted」または「Done」が TRUE になります。
- 「RcvCfg.EndTrig」が_SERIAL_END_RCV_SIZE で、入力変数「RcvCfg.RcvSizeCfg」の値が「0」の場合は、データを受信しません。その場合、命令実行時に「Done」の値が TRUE になります。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「RcvCfg.EndTrig」で_SERIAL_END_RCV_SIZE 指定した場合で、「RcvCfg.RcvSizeCfg」が範囲外の場合。
  b)「Size」、「DevicePort.DevicePortType」、「DevicePort.PortNo」の値が、有効範囲外の場合。
  c)「Option.LastDatRcv」が TRUE の場合。
  d) 入出力変数「RcvDat」に指定した配列変数が、入力変数「Size」で指定したサイズより小さい場合。
  e)「Size」で指定した「RcvDat」の受信データ格納サイズが、受信データよりも小さい場合。
  f)「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
  g)「DevicePort」でデータ型が不正な場合。
  h) NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  i) 受信バッファが一杯になった場合。
  j) 実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令です。
  k) 受信データでパリティエラーが発生した場合。
  l) 受信データでフレーミングエラーが発生した場合。
  m) 受信データでオーバーランエラーが発生した場合。
  n) タイムアウト時間を経過した場合。
  o) NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。



形 NX-CIF210 のシリアルポート 2 に接続したバーコードリーダの読み取り結果を取得します。受信データは、入出力変数「RecvDat」に格納します。スタートコードはなし、エンドコードは 16#OD(CR)とします。

形 NX-CIF210 の設定は、以下のとおりとします。

項目名	設定値
ポート 2: 伝送速度	38400bps
ポート 2: データ長	8 ビット
ポート 2: パリティ	なし
ポート 2: ストップビット	1 ビット
ポート 2: フロー制御	なし

グローバル変数の定義

● グローバル変数

名称	データ型	割付先	コメント
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	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

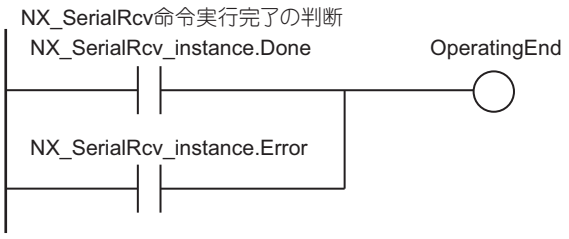
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

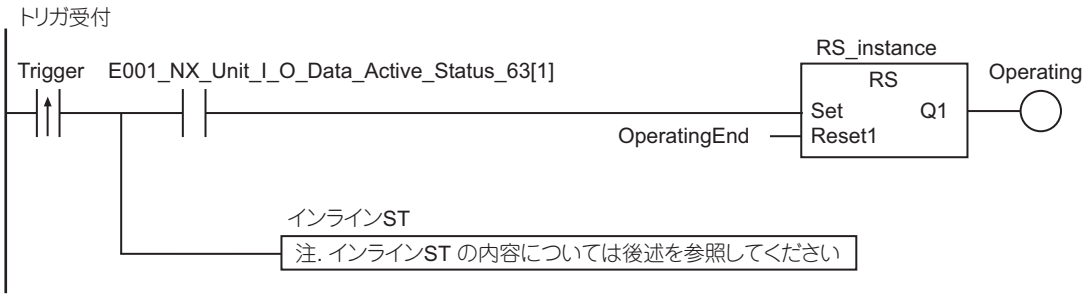
LD

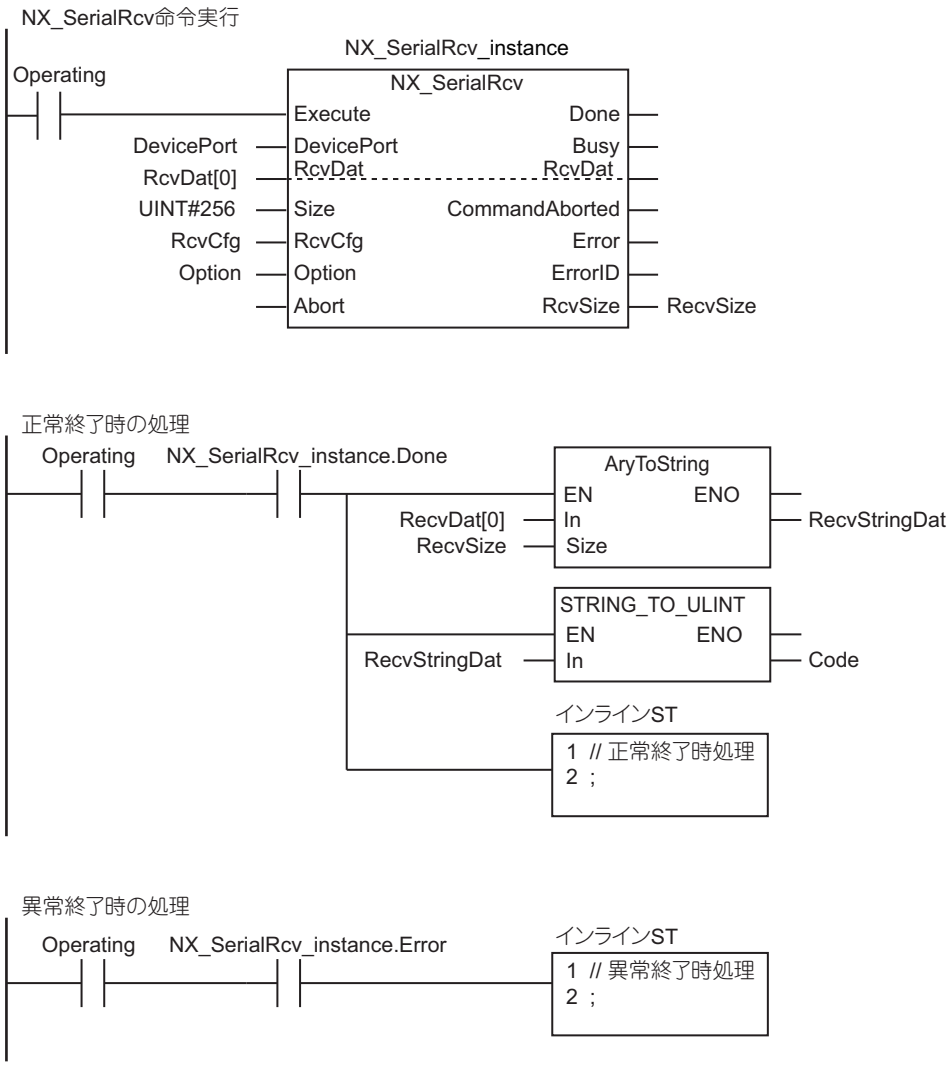
内部変数	名称	データ型	初期値	コメント
	OperationEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定

内部変数	名称	データ型	初期値	コメント
	RecvDat	ARRAY [0..255] of BYTE	[256(16#0)]	受信データ
	RecvSize	UINT	0	受信データサイズ
	RecvStringDat	STRING[257]	''	
	Code	ULINT	0	バーコード（整数）
	RS_instance	RS		
	NX_SerialRcv_instance	NX_SerialRcv		
	RcvCfg	_sSERIAL_CFG		受信完了設定
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	スタートコードなし
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	エンドコードあり
	EndCode	BYTE[2]	[16#0D, 16#00]	16#0D(CR)
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		オプション
	TimeOut	UINT	0	
	LastDatRcv	BOOL	FALSE	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数







● インライン ST の内容

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	RecvDat	ARRAY [0..255] of BYTE	[256(16#0)]	受信データ
	RecvSize	UINT	0	受信データサイズ
	RecvStringDat	STRING[257]	''	
	Code	ULINT	0	バーコード（整数）

内部変数	名称	データ型	初期値	コメント
	NX_SerialRcv_instance	NX_SerialRcv		
	RcvCfg	_sSERIAL_CFG		受信完了設定
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	スタートコードなし
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	エンドコードあり
	EndCode	BYTE[2]	[16#0D, 16#00]	16#0D(CR)
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		オプション
	TimeOut	UINT	0	
	LastDatRcv	BOOL	FALSE	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

```
// Trigger の立ち上がり検出
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;

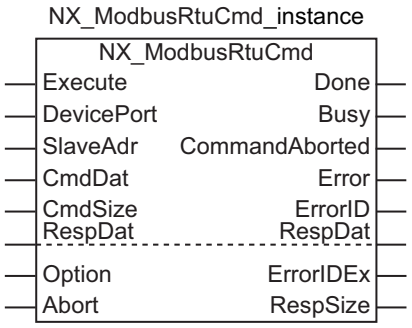
// 通信パラメータ設定と SerialRcv 命令初期化
IF (OperatingStart=TRUE) THEN
         NX_SerialRcv_instance(
                         Execute:=FALSE,          // インスタンス初期化
                         DevicePort:=DevicePort,  // ポート指定
                         Size:=UINT#256,,         // 受信データサイズ
                         RcvDat:=RecvDat,         // 受信データ
                         RcvSize=>RecvSize);      // 実際に受信したデータサイズ
         OperatingStart:=FALSE;
END_IF;
// NX_SerialRcv 命令実行
IF (Operating=TRUE) THEN
         NX_SerialRcv_instance(
                         Execute:=TRUE,
                         DevicePort:=DevicePort,
                         Size:=UINT#256,
```

```
                    RcvDat:=RecvDat,
                    RcvSize=>RecvSize);
        IF (NX_SerialRcv_instance.Done=TRUE) THEN
            // 正常終了時処理
            RecvStringDat:=AryToString(In:=RecvDat[0],Size:=RecvSize); // 文字コードを文字列に変換
            Code:=STRING_TO_ULINT(RecvDat); // 文字列を整数に変換

            Operating:=FALSE;
        END_IF;
        IF (NX_SerialRcv_instance.Error=TRUE) THEN
                // 異常終了時処理
                Operating:=FALSE;
        END_IF;
END_IF;
```

NX_ModbusRtuCmd

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに対して汎用コマンドを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	FB		NX_ModbusRtuCmd_instance(Execute, DevicePort, SlaveAdr, CmdDat, CmdSize, RespDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, RespSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	－	－	－
SlaveAdr	スレーブアドレス		Modbus-RTU スレーブのアドレス *1	0 ～ 247	－	1
CmdDat[] 配列	コマンドデータ		コマンドデータ	データ型に従う	－	*2
CmdSize	コマンドデータサイズ		コマンドデータサイズ	1 ～ 253	バイト	*2 *3
Option	オプション		オプション	－	－	－
Abort	中断		命令実行の中断	データ型に従う	－	FALSE
RespDat[] 配列	読み出しデータ	入出力	読み出しデータを格納する変数	データ型に従う	－	－
Command Aborted	中断完了	出力	中断完了	データ型に従う	－	－
RespSize	受信サイズ		受信データサイズ	1 ～ 253	バイト	*4

*1. 「0」を指定した場合は、ブロードキャストで Modbus-RTU スレーブにコマンドを送信することができます。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*3. ファンクションコードとコマンドデータの合計バイト数を指定します。ファンクションコードは 1 バイトです。
*4. ファンクションコートと読み出しデータの合計バイト数が格納されます。ファンクションコードは 1 バイトです。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1175)」参照																			
SlaveAdr							○													
CmdDat[] 配列		○																		
CmdSize							○													

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Option	構造体_sSERIAL_MODBUSRTU_OPTION　詳細は「機能 (P.2-1175)」参照																			
Abort	○																			
RespDat[] 配列		○																		
CommandAborted	○																			
RespSize							○													

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブへ汎用コマンドを送信します。
送信したコマンドに対する正常レスポンスを受信した場合、正常終了します。
ブロードキャストで送信した場合は、送信完了後 、各スレーブからのレスポンスは待たずに本命令は正常終了します。

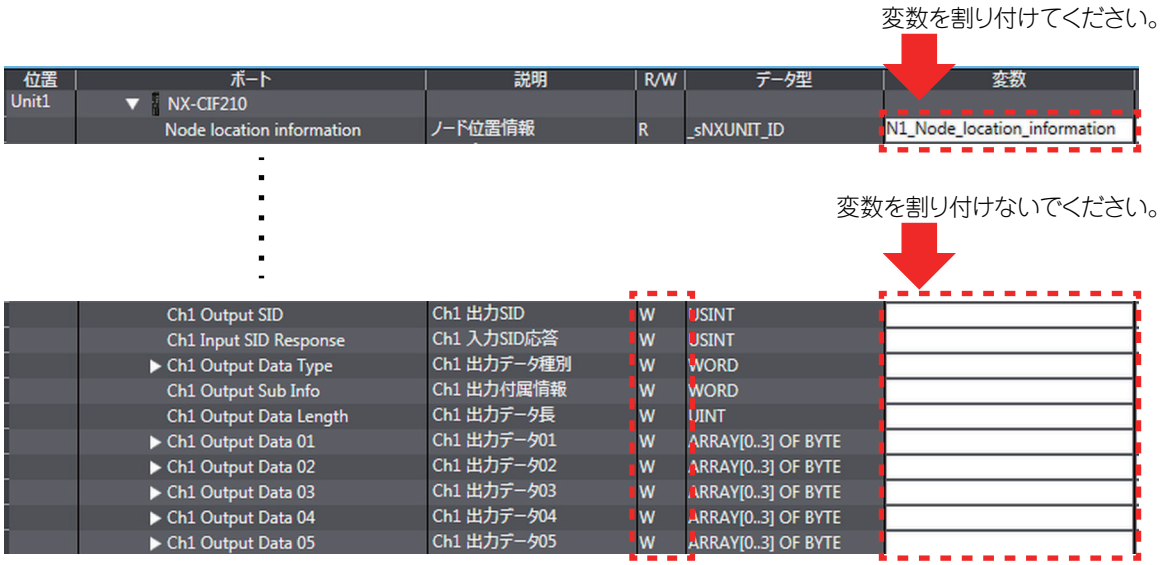
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	−	−	−
Reserved	予約	予約	予約	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2
NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

Modbus-RTU スレーブのアドレスは入力変数「SlaveAdr」で指定します。
入力変数「SlaveAdr」に「0」を設定すると、ブロードキャストで Modbus-RTU スレーブにコマンドを送信します。

コマンドデータは入力変数「CmdDat」で、コマンドデータのサイズは入力変数「CmdSize」で指定します。
CRC は命令が付加します。
読み出すデータを格納する変数は、入出力変数「RespDat」で指定します。
出力変数「RespSize」は、受信データのサイズを表します。

オプションを指定する場合は、入力変数「Option」で指定します。
入力変数「Option」のデータ型は、構造体_sSERIAL_MODBUSRTU_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sSERIAL_MODBUS RTU_OPTION	−	−	−
SendDelay	送信ディレー時間	送信ディレー時間を0.01s 単位で指定	UINT	データ型に従う	0.01s	0
TimeOut	タイムアウト時間	タイムアウト時間 「0」の場合は、2.0s	UINT	データ型に従う	0.1s	20
NoResponse	レスポンスなし	• 送信するコマンドのレスポンスがない場合に、TRUE を指定 • TRUE を指定した場合、コマンド送信後、タイムアウト時間を待たずに本命令が正常終了	BOOL	データ型に従う	−	FALSE
Retry	再送回数	再送回数を指定	USINT	0～15	−	0

使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

命令実行の中断

命令実行中に「Abort」を TRUE にすると、命令実行を中断します。
命令実行を中断すると「CommandAborted」が TRUE となります。
命令実行の中断が間に合わなかった場合は、「Done」が TRUE となり、命令が正常終了します。
「Abort」と「Execute」の両方を TRUE にした場合は、「CommandAborted」が TRUE となります。

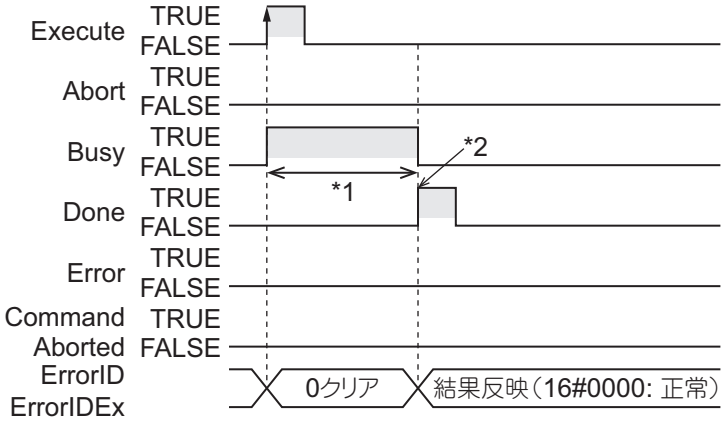
中断の動作は、「Busy」処理を終了させるだけで、送受信バッファをクリアしません。バッファのクリアが必要な場合、「NX_SerialBufClear (P.2-1213)」命令を使用してください。

タイミングチャート

タイミングチャートを、以下に示します。

● 正常終了時（「SendDelay」が「0」(0s)の場合）

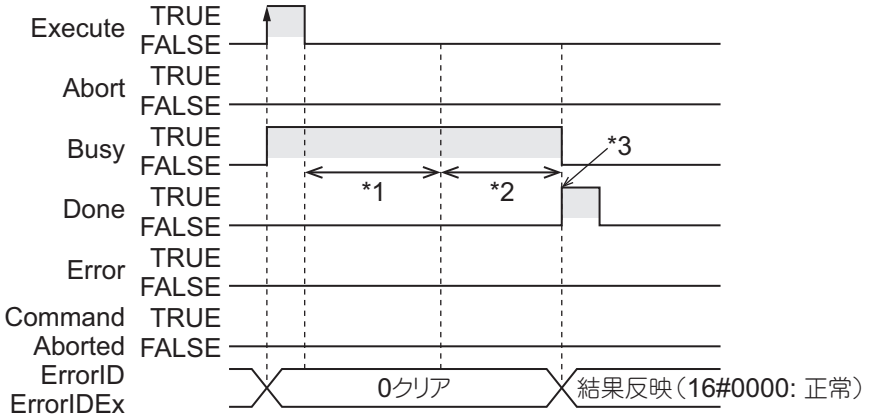
「SendDelay」が「0」(0s)の場合は、次の動作になります。



*1. Modbus-RTU スレーブとの処理
*2. コマンドに対するレスポンスを受信

● 正常終了時（「SendDelay」が「100」(1s)の場合）

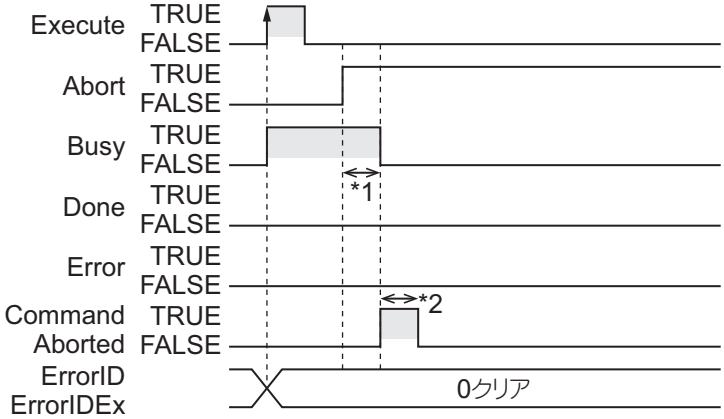
「SendDelay」が「100」(1s)の場合は、次の動作になります。



*1. 送信ディレー時間 1s
*2. Modbus-RTU スレーブへのコマンド送信、Modbus-RTU スレーブからのレスポンス受信
*3. コマンドに対するレスポンスを受信

● 中断実行時（「Busy」が TRUE の場合）

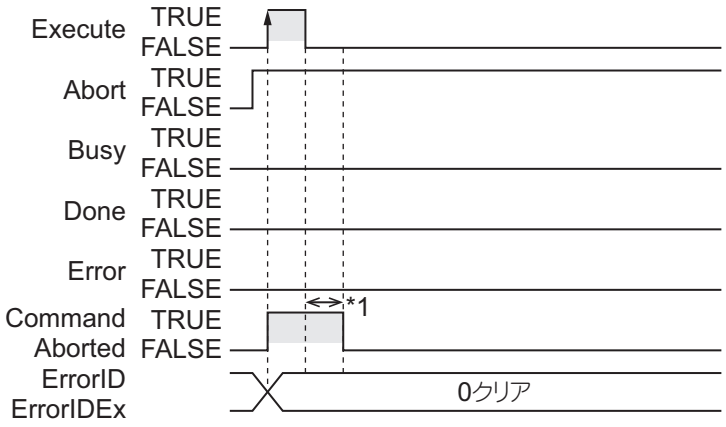
「Busy」が TRUE の状態で、「Abort」を TRUE にする場合は、次の動作になります。



*1. 中断処理
*2. 1 タスク周期後に FALSE

● 中断実行時（「Execute」が TRUE の場合）

「Abort」と「Execute」の両方を TRUE にする場合は、次の動作になります。



*1. 1 タスク周期後に FALSE

参考

Modbus-RTU モードのフレームフォーマットは以下のとおりです。

Slave Address	Function Code	Data	CRC
1バイト	1バイト	0～252バイト	2バイト*

*CRCコードは、Lowバイト、Highバイトの順です。

MODBUS 通信プロトコルの仕様については、「MODBUS Application Protocol Specification」を参照してください。
「MODBUS Application Protocol Specification」は MODBUS Organization, Inc.から入手できます。
http://www.modbus.org/

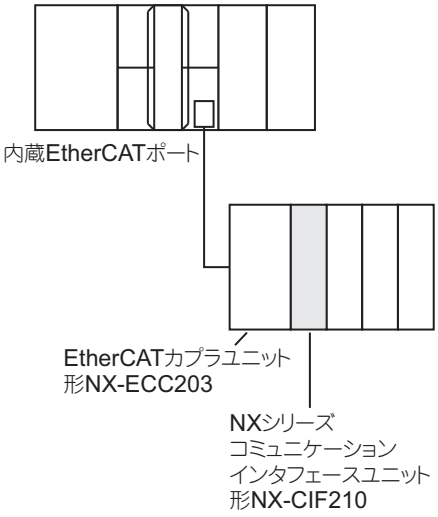
使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。
  処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。命令実行中に「Abort」が TRUE になった場合は、「CommandAborted」または「Done」が TRUE になります。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。

- つぎの場合、対象となるデバイスポートのバッファにデータが残っている可能性があります。バッファをクリアするため、命令実行前に NX_SerialBufClear 命令を実行してください。
  ここでの命令とは、NX_ModbusRtuCmd 命令, NX_ModbusRtuRead 命令,
  NX_ModbusRtuWrite 命令を指します。
  a）運転開始後または、運転モードに変化させた場合
  b）前回の命令実行時にリトライの設定がされていた(「Option.Retry」≠ 0)場合
  c）前回の命令実行が中断された（出力変数「CommandAborted」= TRUE）場合
  d）前回の命令実行でエラー発生(「Error」= TRUE)した場合
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「CmdSize」、「Option.Retry」、「DevicePort.DevicePortType」、「DevicePort.PortNo」、「SlaveAdr」に範囲外の値を指定した場合。
  b）「CmdDat」に指定した変数が、「CmdSize」で指定したサイズより小さい場合。
  c）「RespDat」に指定した変数のサイズより受信データのサイズが大きい場合。
  d）「DevicePort」で存在しないユニットあるいはポートを指定した場合。
  e）「DevicePort」でデータ型が不正な場合。
  f） NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  g）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令です。
  h）受信データでパリティエラーが発生した場合。
  i） 受信データでフレーミングエラーが発生した場合。
  j） 受信データでオーバーランエラーが発生した場合。
  k）受信データが CRC 不一致の場合。
  l） タイムアウト時間を経過した場合。
  m) NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。
  n）Modbus-RTU スレーブから Exception Response を受信した場合。Exception Code は、出力変数「ErrorIDEx」で確認します。
  o）Modbus-RTU スレーブからのレスポンスデータのファンクションコードや受信サイズが不正な場合。
- 拡張エラーコード「ErrorIDEx」は、本命令では、Modbus-RTU スレーブで異常を検出したときに表示します。「ErrorIDEx」は、エラーコード「ErrorID」の値が「WORD#16#0C10」の場合に出力され、「ErrorIDEx=000000XX」と表示します。「XX」については、MODBUS 通信プロトコルの Exception Code 仕様を確認してください。
  MODBUS 通信プロトコルの Exception Code 仕様については、「MODBUS Application Protocol Specification」を参照してください。
  「MODBUS Application Protocol Specification」は、MODBUS Organization, Inc.から入手できます。
  http://www.modbus.org/

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。
形 NX-CIF210 のユニット動作設定の［Ch2 終端検出キャラクタ数］を「35」に設定します。終端検出キャラクタ数は、単位が 0.1 キャラクタ数のため、3.5 キャラクタとして動作します。



「Trigger」が ON になると、形 NX-CIF210 のシリアルポート 2 のバッファをクリアし、その後 Modbus-RTU コマンドを送信します。
スレーブアドレス 1 の読み出し開始アドレス 32(BYTE#16#0020)から 1 つの保持レジスタを読み出します。
汎用コマンド送受信を使用し、変数の読み出しを実行します。

ST

内部変数	名称	データ型	初期値	コメント
	Stage	INT	0	
	Trigger	BOOL	FALSE	実行条件
	DevicePort	_sDEVICE_PORT		ポート指定
	NX_SerialBufClear_instance	NX_SerialBufClear		バッファクリア
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuCmd_instance	NX_ModbusRtuCmd		
	ModbusSlaveAdr	UINT	UINT#0	スレーブアドレス
	ModbusCmdDat	ARRAY[0..19] OF BYTE		MODBUS コマンドデータ
	ModbusDatSize	UINT	UINT#0	MODBUS コマンドデータ総サイズ（バイト）
	ModbusRespDat	ARRAY[0..275] OF BYTE		受信データ格納エリア
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		
	ModbusError	BOOL		
	ModbusRspSize	UINT		実際に受信したサイズ（バイト）
	DoModbusTrigger	BOOL		

外部変数	名称	データ型	コンスタント	コメント
	N1_Node_location_information	_sNXUNIT_ID	☑	

```
//Trigger の立ち上がりでシーケンス開始
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; // 初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1: // バッファクリア要求
        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; // バッファクリア 正常終了
        ELSIF ( ClearError = TRUE ) THEN
            Stage := 99; // バッファクリア 異常終了
        END_IF;

    2: //Modbus Cmd 送信要求
        ModbusSlaveAdr := 1; // スレーブアドレス
        ModbusCmdDat[1]:=BYTE#16#03; // ファンクションコード ( 変数読出し)
        ModbusCmdDat[2]:=BYTE#16#00; // 読出し開始アドレス (H)
        ModbusCmdDat[3]:=BYTE#16#20; // 読出し開始アドレス (L)
        ModbusCmdDat[4]:=BYTE#16#00; // データ数 (H)
        ModbusCmdDat[5]:=BYTE#16#01; // データ数 (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);

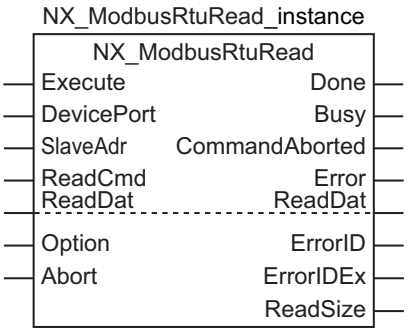
        IF (ModbusDone = TRUE) THEN
            Stage := 3; //NX_ModbusRtuCmd 命令 正常終了
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
            Stage :=99; //NX_ModbusRtuCmd 命令 異常終了 or Abort
        END_IF;

    3: //NX_ModbusRtuCmd 命令正常終了後の処理
        Trigger := FALSE;
        DoModbusTrigger := FALSE;

    99: // 異常処理
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
    END_CASE;
END_IF;
```

NX_ModbusRtuRead

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに読み出しコマンドを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuRead	Modbus RTU Read コマンド送信	FB		NX_ModbusRtuRead_instance(Execute, DevicePort, SlaveAdr, ReadCmd, ReadDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, ReadSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	−	−	−
SlaveAdr	スレーブアドレス		Modbus-RTU スレーブのアドレス *1	1 ～ 247	−	1
ReadCmd	読み出しコマンド		読み出しコマンド	−	−	*2
Option	オプション		オプション	−	−	−
Abort	中断		命令実行の中断	データ型に従う	−	FALSE
ReadDat[] 配列	読み出しデータ	入出力	読み出しデータを格納する変数	データ型に従う	−	−
Command Aborted	中断完了	出力	中断完了	データ型に従う	−	−
ReadSize	受信サイズ		受信データサイズ	1 ～ 2000 *3	−*4	−

*1. 「0」を設定した場合は、異常になります。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。
*3. 受信データが WORD データの場合、上限値は「125」となります。
*4. 「ReadCmd.Fun」で指定する読み出しデータの単位と同じです。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1185)」参照																			
SlaveAdr							○													
ReadCmd	構造体_sSERIAL_MODBUSRTU_READ　詳細は「機能 (P.2-1185)」参照																			
Option	構造体_sSERIAL_MODBUSRTU_OPTION　詳細は「機能 (P.2-1185)」参照																			
Abort	○																			
ReadDat[] 配列	○		○																	
	配列全体も指定可能																			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CommandAborted	○																			
ReadSize							○													

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに対して読み出しコマンドを送信し、要求したデータを Modbus-RTU スレーブから読み出します。
送信したコマンドに対する正常レスポンスを受信した場合、正常終了します。

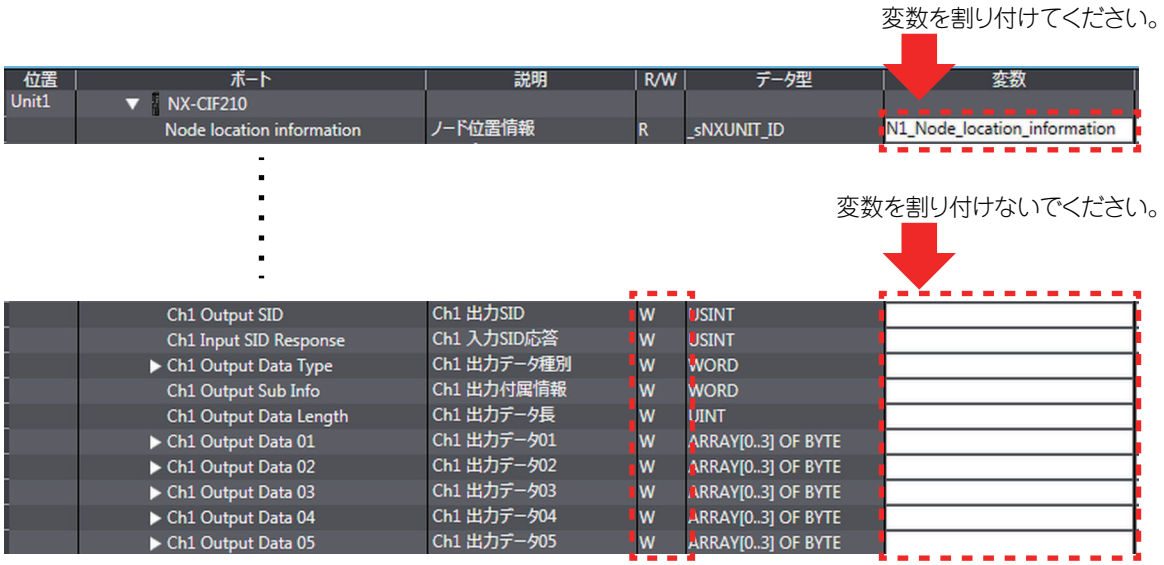
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	−	−	−
Reserved	予約	予約	予約	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2
NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

Modbus-RTU スレーブのアドレスは入力変数「SlaveAdr」で指定します。
入力変数「SlaveAdr」に「0」を指定した場合は異常となり、ブロードキャストで Modbus-RTU スレーブにコマンドを送信することはできません。

読み出しコマンドは入力変数「ReadCmd」で指定します。
CRC は命令が付加します。
入力変数「ReadCmd」のデータ型は、構造体_sSERIAL_MODBUSRTU_READ です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
ReadCmd	読み出しコマンド	読み出しコマンド	_sSERIAL_MODBUSRTU_READ	−	−	−
Fun	ファンクションコード	ファンクションコード	_eMDB_FUN	_MDB_READ_COILS _MDB_READ_DISCRETE_INPUTS _MDB_READ_HOLDING_REGISTERS _MDB__READ_INPUT_REGISTERS	−	_MDB_READ_COILS
ReadAdr	読み出しアドレス	読み出し開始アドレス	UINT	データ型に従う	−	0
ReadSize	読み出しサイズ	読み出しサイズ	UINT	ファンクションコードに従う	−*1	1

*1. 「ReadCmd.Fun」で指定する読み出しデータの単位と同じです。

「Fun」のデータ型は、列挙型_eMDB_FUN です。
列挙型_eMDB_FUN の、列挙子の意味は以下のとおりです。

列挙子	意味
_MDB_READ_COILS	コイル読み出し（ビット）
_MDB_READ_DISCRETE_INPUTS	入力読み出し（ビット）
_MDB_READ_HOLDING_REGISTERS	保持レジスタ読み出し（ワード）
_MDB_READ_INPUT_REGISTERS	入力レジスタ読み出し（ワード）

「ReadSize」で指定できる有効範囲は、ファンクションコードにより異なります。
各数値は、読み出し対象のデータサイズと最大コマンド長により決まります。
仕様は、以下のとおりです。

ファンクションコード	ReadSize
_MDB_READ_COILS	1～2000（ビット）
_MDB_READ_DISCRETE_INPUTS	1～2000（ビット）
_MDB_READ_HOLDING_REGISTERS	1～125（ワード）
_MDB_READ_INPUT_REGISTERS	1～125（ワード）

読み出すデータを格納する変数は、入出力変数「ReadDat」で指定します。
「ReadDat」で使用できるデータ型は、ファンクションコードにより異なります。
仕様は、以下のとおりです。

ファンクションコード	データ型
_MDB_READ_COILS	BOOL BOOL[]
_MDB_READ_DISCRETE_INPUTS	BOOL BOOL[]
_MDB_READ_HOLDING_REGISTERS	WORD WORD[]
_MDB_READ_INPUT_REGISTERS	WORD WORD[]

出力変数「ReadSize」は、読み出したデータのサイズを表します。

オプションを指定する場合は、入力変数「Option」で指定します。
入力変数「Option」のデータ型は、構造体_sSERIAL_MODBUSRTU_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sSERIAL_MODBUSRTU_OPTION	−	−	−
SendDelay	送信ディレー時間	送信ディレー時間	UINT	データ型に従う	0.01s	0
TimeOut	タイムアウト時間	タイムアウト時間 「0」の場合は、2.0s	UINT	データ型に従う	0.1s	20
NoResponse	レスポンスなし	本命令では不使用	BOOL	データ型に従う	−	FALSE
Retry	再送回数	再送回数	USINT	0～15	−	0

使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

命令実行の中断

命令実行中に「Abort」を TRUE にすると、命令実行を中断します。
命令実行を中断すると「CommandAborted」が TRUE となります。
命令実行の中断が間に合わなかった場合は、「Done」が TRUE となり、命令が正常終了します。
「Abort」と「Execute」の両方を TRUE にした場合は、「CommandAborted」が TRUE となります。

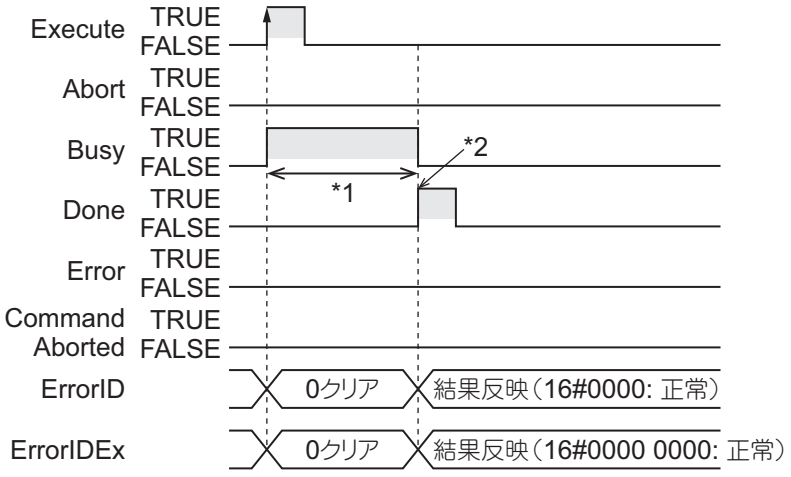
中断の動作は、「Busy」処理を終了させるだけで、送受信バッファをクリアしません。バッファのクリアが必要な場合、「NX_SerialBufClear (P.2-1213)」命令を使用してください。

タイミングチャート

タイミングチャートを、以下に示します。

● 正常終了時（「SendDelay」が「0」(0s)の場合）

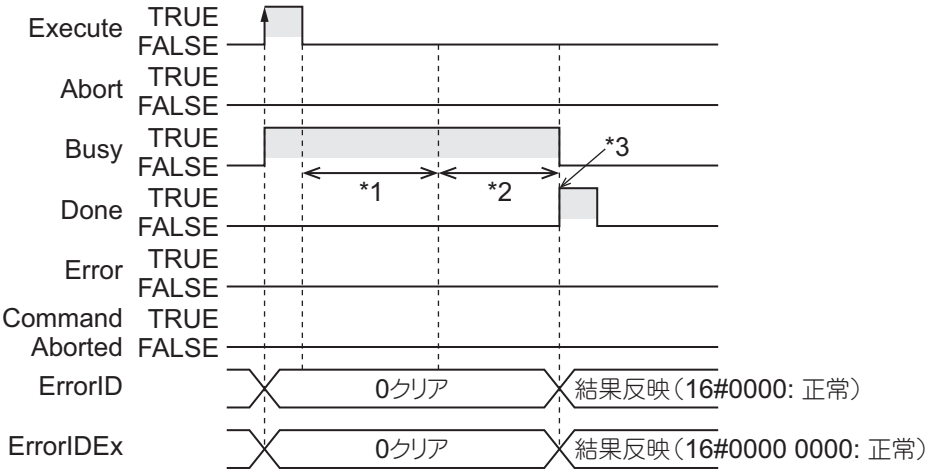
「SendDelay」が「0」(0s)の場合は、次の動作になります。



*1. Modbus-RTU スレーブとの処理
*2. コマンドに対するレスポンスを受信

● 正常終了時（「SendDelay」が「100」(1s)の場合）

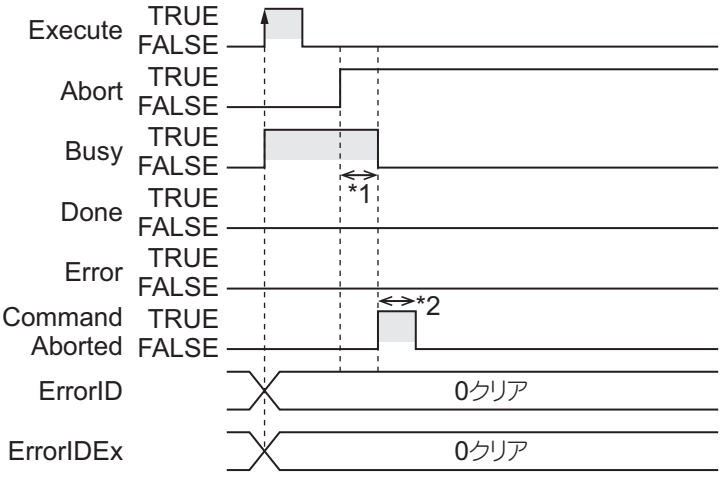
「SendDelay」が「100」(1s)の場合は、次の動作になります。



*1. 送信ディレー時間 1s
*2. Modbus-RTU スレーブへ読み出しコマンド送信、Modbus-RTU スレーブからレスポンス受信
*3. コマンドに対するレスポンスを受信

● 中断実行時（「Busy」が TRUE の場合）

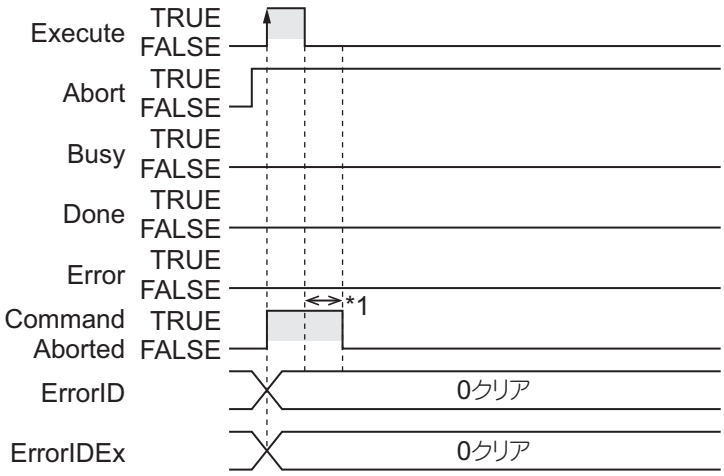
「Busy」が TRUE の状態で、「Abort」を TRUE にする場合は、次の動作になります。



*1. 中断処理
*2. 1 タスク周期後に FALSE

● 中断実行時（「Execute」が TRUE の場合）

「Abort」と「Execute」の両方を TRUE にする場合は、次の動作になります。



*1. 1 タスク周期後に FALSE

参考

MODBUS 通信プロトコルの仕様については、「MODBUS Application Protocol Specification」を参照してください。
「MODBUS Application Protocol Specification」は MODBUS Organization, Inc.から入手できます。
http://www.modbus.org/

使用上の注意

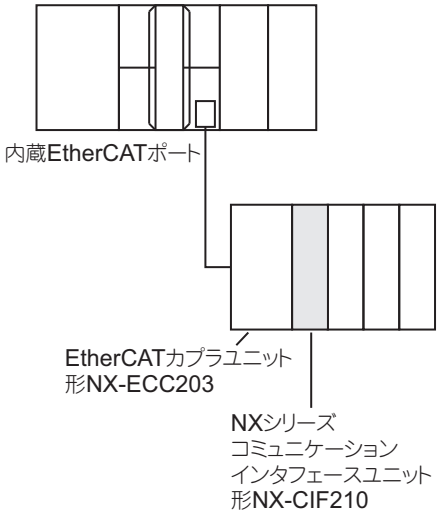
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。
  処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。命令実行中に「Abort」が TRUE になった場合は、「CommandAborted」または「Done」が TRUE になります。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、対象となるデバイスポートのバッファにデータが残っている可能性があります。バッファをクリアするため、命令実行前に NX_SerialBufClear 命令を実行してください。
  ここでの命令とは、NX_ModbusRtuCmd 命令, NX_ModbusRtuRead 命令, NX_ModbusRtuWrite 命令を指します。
  a）運転開始後または、運転モードに変化させた場合
  b）前回の命令実行時にリトライの設定がされていた(「Option.Retry」≠ 0)場合
  c）前回の命令実行が中断された（出力変数「CommandAborted」= TRUE）場合
  d）前回の命令実行でエラー発生(「Error」= TRUE)した場合
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「SlaveAdr」、「ReadCmd.ReadSize」、「ReadCmd.Fun」、「Option.Retry」、「DevicePort.DevicePortType」、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b）「ReadDat」に指定した変数が、「ReadCmd.ReadSize」で指定したサイズより小さい場合。

c）「DevicePort」で存在しないユニットあるいはポートを指定した場合。
d）「DevicePort」、「RespDat」のデータ型が不正な場合。
e）NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
f）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
対象となる命令は、NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令です。
g）受信データでパリティエラーが発生した場合。
h）受信データでフレーミングエラーが発生した場合。
i）受信データでオーバーランエラーが発生した場合。
j）受信データが CRC 不一致の場合。
k）タイムアウト時間を経過した場合。（リトライ設定時はリトライ回数×タイムアウト時間）
l）NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。
m）Modbus-RTU スレーブから Exception Response を受信した場合。Exception Code は、出力変数「ErrorIDEx」で確認します。
n）Modbus-RTU スレーブからのレスポンスデータのファンクションコードや受信サイズが不正な場合。

- 拡張エラーコード「ErrorIDEx」は、本命令では、Modbus-RTU スレーブで異常を検出したときに表示します。「ErrorIDEx」は、エラーコード「ErrorID」の値が「WORD#16#0C10」の場合に出力され、「ErrorIDEx=000000XX」と表示します。「XX」については、MODBUS 通信プロトコルの Exception Code 仕様を確認してください。
MODBUS 通信プロトコルの Exception Code 仕様については、「MODBUS Application Protocol Specification」を参照してください。
「MODBUS Application Protocol Specification」は、MODBUS Organization, Inc.から入手できます。
http://www.modbus.org/

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。
形 NX-CIF210 のユニット動作設定の〔Ch2 終端検出キャラクタ数〕を「35」に設定します。終端検出キャラクタ数は、単位が 0.1 キャラクタ数のため、3.5 キャラクタとして動作します。



「Trigger」が ON になると、形 NX-CIF210 のシリアルポート 2 のバッファをクリアし、その後 Modbus-RTU コマンドを送信します。
スレーブアドレス 1 の読み出し開始アドレス 19 から 1 個、コイルのステータスを読み出します。
読み出しコマンド送信を使用し、変数の読み出しを実行します。

ST

内部変数	名称	データ型	初期値	コメント
	Stage	INT	0	
	Trigger	BOOL	FALSE	実行条件
	DevicePort	_sDEVICE_PORT		ポート指定
	NX_SerialBufClear_instance	NX_SerialBufClear		バッファクリア
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuRead_instance	NX_ModbusRtuRead		
	ModbusSlaveAdr	UINT	UINT#0	スレーブアドレス
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		
	ModbusError	BOOL		
	ModbusReadSize	UINT		実際に受信したサイズ（バイト）
	DoModbusTrigger	BOOL		
	ModbusReadDat	BOOL		
	ModbusReadCmd	_sSERIAL_MODBUS RTU_READ		

外部変数	名称	データ型	コンスタント	コメント
	N1_Node_location_information	_sNXUNIT_ID	☑	

```
//Trigger の立ち上がりでシーケンス開始
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; //初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1: // バッファクリア要求
        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; //バッファクリア　正常終了
        ELSIF (ClearError = TRUE) THEN
            Stage := 99; //バッファクリア　異常終了
        END_IF;

    2: //Modbus　Read 要求
        ModbusSlaveAdr := 1; // スレーブアドレス
        ModbusReadCmd.Fun:=_MDB_READ_COILS; // ファンクションコード
        ModbusReadCmd.ReadAdr:=19; // 読み出しアドレス
        ModbusReadCmd.ReadSize:=1; // 読み出しサイズ

        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; //NX_ModbusRead 命令　正常終了
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
            Stage :=99; //NX_ModbusRead 命令　異常終了 or Abort
        END_IF;

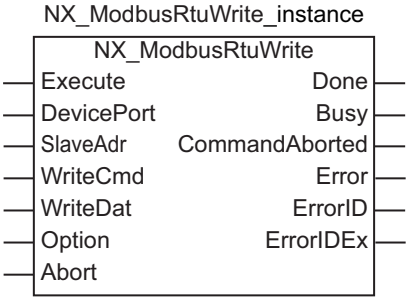
    3: //NX_ModbusRead 命令正常終了後の処理
        Trigger := FALSE;
```

```
            DoModbusTrigger := FALSE;

        99: //異常処理
            Trigger := FALSE;
            DoModbusTrigger := FALSE;
        END_CASE;
    END_IF;
```

NX_ModbusRtuWrite

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに書き込みコマンドを送信します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	FB		NX_ModbusRtuWrite_instance(Execute, DevicePort, SlaveAdr, WriteCmd, WriteDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	−	−	−
SlaveAdr	スレーブアドレス		Modbus-RTU スレーブのアドレス *1	0 ～ 247	−	1
WriteCmd	書き込みコマンド		書き込みコマンド	−	−	*2
WriteDat[] 配列	書き込みデータ		書き込みデータ	データ型に従う	−	*2
Option	オプション		オプション	−	−	−
Abort	中断		命令実行の中断	データ型に従う	−	FALSE
Command Aborted	中断完了	出力	中断完了	データ型に従う	−	−

*1. 「0」を設定した場合は、ブロードキャストで Modbus-RTU スレーブにコマンドを送信することができます。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1196)」参照																			
SlaveAdr							○													
WriteCmd	構造体_sSERIAL_MODBUSRTU_WRITE　詳細は「機能 (P.2-1196)」参照																			
WriteDat[] 配列	○		○																	
	配列全体も指定可能																			
Option	構造体_sSERIAL_MODBUSRTU_OPTION　詳細は「機能 (P.2-1196)」参照																			
Abort	○																			
CommandAborted	○																			

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートから、Modbus-RTU 方式の手順で、Modbus-RTU スレーブに対して書き込みコマンドを送信します。
送信したコマンドに対する正常レスポンスを受信した場合、正常終了します。
ブロードキャストで送信した場合は、送信完了後 、各スレーブからのレスポンスは待たずに本命令は正常終了します。

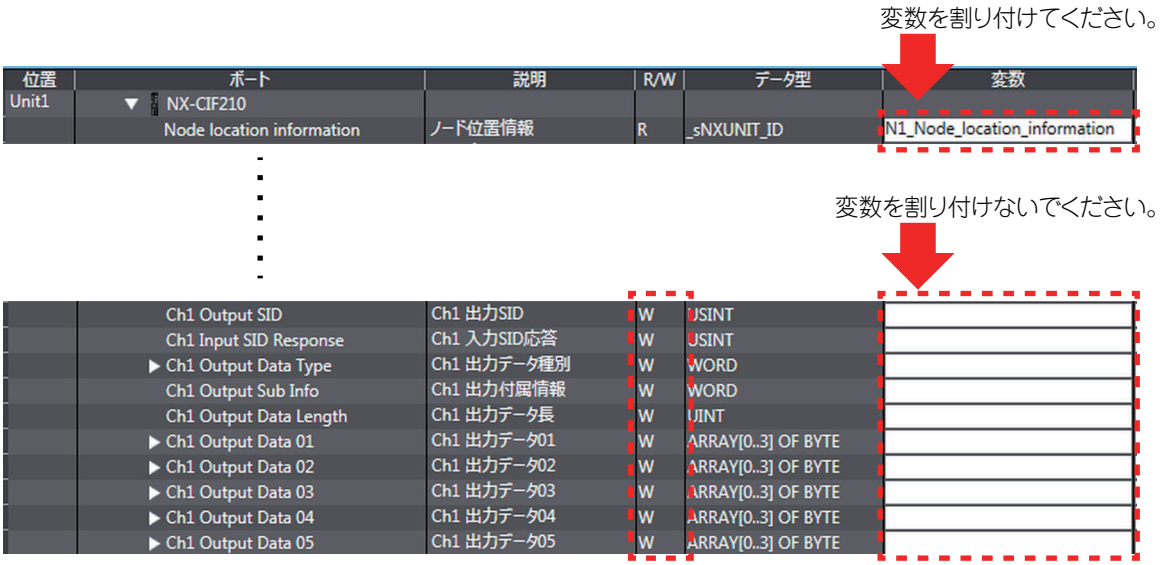
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	−	−	−
Reserved	予約	予約	予約	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2
NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

Modbus-RTU スレーブのアドレスは入力変数「SlaveAdr」で指定します。
入力変数「SlaveAdr」に「0」を設定すると、ブロードキャストで Modbus-RTU スレーブにコマンドを送信します。

書き込みコマンドは入力変数「WriteCmd」で指定します。
CRC は命令が付加します。
入力変数「WriteCmd」のデータ型は、構造体_sSERIAL_MODBUSRTU_WRITE です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
WriteCmd	書き込みコマンド	書き込みコマンド	_sSERIAL_MODBUSRTU_WRITE	−	−	−
Fun	ファンクションコード	ファンクションコード	_eMDB_FUN	_MDB_WRITE_SINGLE_COIL _MDB_WRITE_SINGLE_REGISTER _MDB_WRITE_MULTIPLE_COILS _MDB_WRITE_MULTIPLE_REGISTERS	−	_eMDB_WRITE_SINGLE_COIL
WriteAdr	書き込みアドレス	書き込み開始アドレス	UINT	データ型に従う	−	0
WriteSize	書き込みサイズ	書き込みサイズ	UINT	ファンクションコードに従う	−	_MDB_WRITE_SINGLE_COIL

「Fun」のデータ型は、列挙型_eMDB_FUN です。
列挙型_eMDB_FUN の、列挙子の意味は以下のとおりです。

列挙子	意味
_MDB_WRITE_SINGLE_COIL	コイル 1 点書き込み（ビット）
_MDB_WRITE_SINGLE_REGISTER	保持レジスタ 1 点書き込み（ワード）
_MDB_WRITE_MULTIPLE_COILS	複数コイル書き込み（ビット）
_MDB_WRITE_MULTIPLE_REGISTERS	複数保持レジスタ書き込み（ワード）

「WriteSize」で指定できる有効範囲は、ファンクションコードにより異なります。
各数値は、書き込み対象のデータサイズと最大コマンド長により決まります。
仕様は、以下のとおりです。

ファンクションコード	WriteSize
_MDB_WRITE_SINGLE_COIL	1（ビット）
_MDB_WRITE_SINGLE_REGISTER	1（ワード）
_MDB_WRITE_MULTIPLE_COILS	1～1968（ビット）
_MDB_WRITE_MULTIPLE_REGISTERS	1～123（ワード）

書き込むデータは、入力変数「WriteDat」で指定します。
「WriteDat」で使用できるデータ型は、ファンクションコードにより異なります。
仕様は、以下のとおりです。

ファンクションコード	データ型
_MDB_WRITE_SINGLE_COIL	BOOL BOOL[]
_MDB_WRITE_SINGLE_REGISTER	WORD WORD[]
_MDB_WRITE_MULTIPLE_COILS	BOOL BOOL[]
_MDB_WRITE_MULTIPLE_REGISTERS	WORD WORD[]

オプションを指定する場合は、入力変数「Option」で指定します。
入力変数「Option」のデータ型は、構造体_sSERIAL_MODBUS_OPTION です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
Option	オプション	オプション	_sSERIAL_MODBUSRTU_OPTION	−	−	−
SendDelay	送信ディレー時間	送信ディレー時間	UINT	データ型に従う	0.01s	0
TimeOut	タイムアウト時間	タイムアウト時間 「0」の場合は、2.0s	UINT	データ型に従う	0.1s	20
NoResponse	レスポンスなし	本命令では不使用	BOOL	データ型に従う	−	FALSE
Retry	再送回数	再送回数	USINT	0 ～ 15	−	0

使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

命令実行の中断

命令実行中に「Abort」を TRUE にすると、命令実行を中断します。
命令実行を中断すると「CommandAborted」が TRUE となります。
命令実行の中断が間に合わなかった場合は、「Done」が TRUE となり、命令が正常終了します。
「Abort」と「Execute」の両方を TRUE にした場合は、「CommandAborted」が TRUE となります。

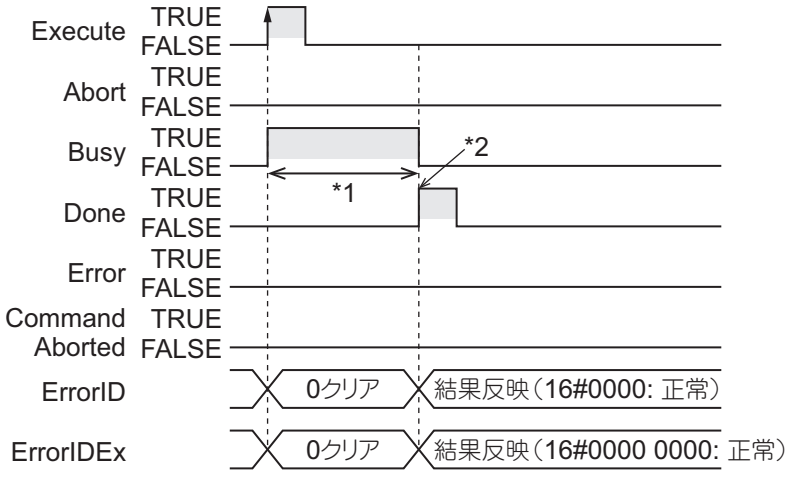
中断の動作は、「Busy」処理を終了させるだけで、送受信バッファをクリアしません。バッファのクリアが必要な場合、「NX_SerialBufClear (P.2-1213)」命令を使用してください。

タイミングチャート

タイミングチャートを、以下に示します。

● 正常終了時（「SendDelay」が「0」(0s)の場合）

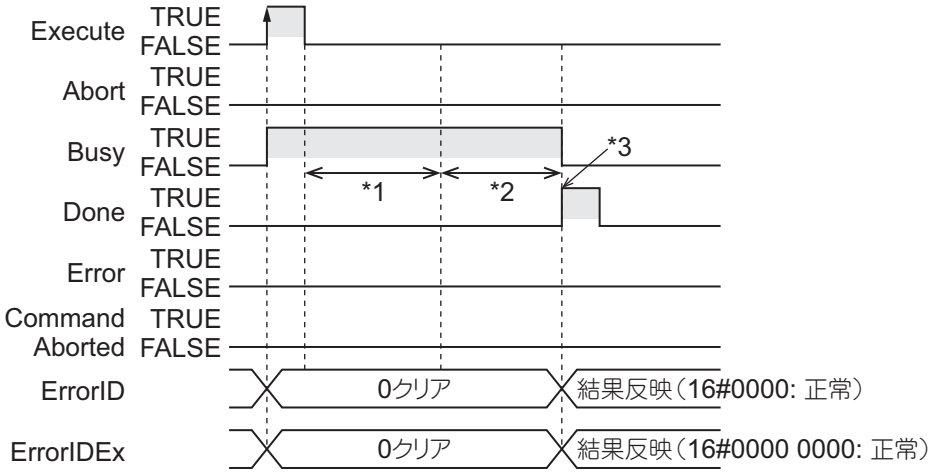
「SendDelay」が「0」(0s)の場合は、次の動作になります。



*1. Modbus-RTU スレーブとの処理
*2. コマンドに対するレスポンスを受信

● 正常終了時（「SendDelay」が「100」(1s)の場合）

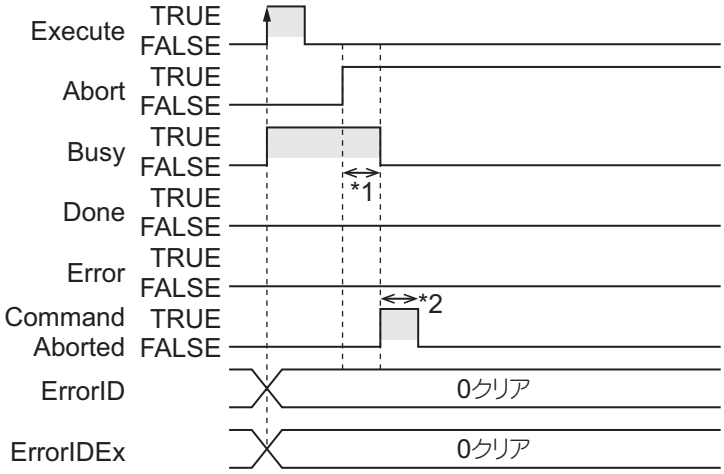
「SendDelay」が「100」(1s)の場合は、次の動作になります。



*1. 送信ディレー時間 1s
*2. Modbus-RTU スレーブへ書き込みコマンド送信、Modbus-RTU スレーブからレスポンス受信
*3. コマンドに対するレスポンスを受信

● 中断実行時（「Busy」が TRUE の場合）

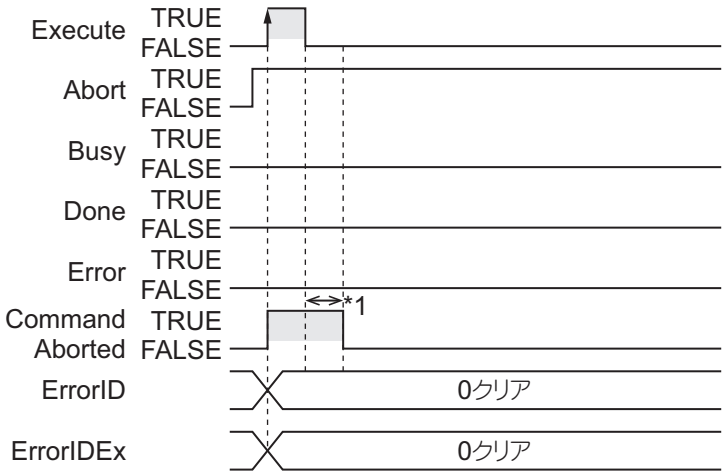
「Busy」が TRUE の状態で、「Abort」を TRUE にする場合は、次の動作になります。



*1. 中断処理
*2. 1 タスク周期後に FALSE

● 中断実行時（「Execute」が TRUE の場合）

「Abort」と「Execute」の両方を TRUE にする場合は、次の動作になります。



*1. 1 タスク周期後に FALSE

参考

MODBUS 通信プロトコルの仕様については、「MODBUS Application Protocol Specification」を参照してください。
「MODBUS Application Protocol Specification」は MODBUS Organization, Inc.から入手できます。
http://www.modbus.org/

使用上の注意

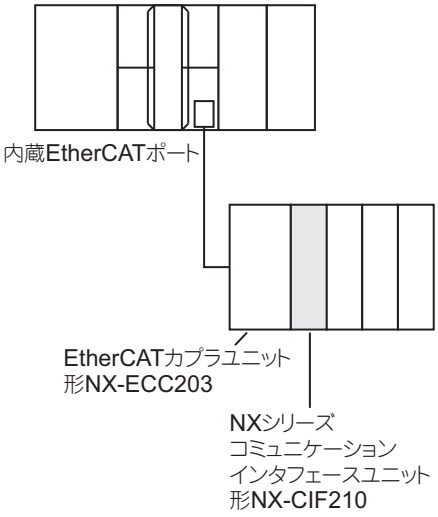
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。
  処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。命令実行中に「Abort」が TRUE になった場合は、「CommandAborted」または「Done」が TRUE になります。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、対象となるデバイスポートのバッファにデータが残っている可能性があります。バッファをクリアするため、命令実行前に NX_SerialBufClear 命令を実行してください。
  ここでの命令とは、NX_ModbusRtuCmd 命令, NX_ModbusRtuRead 命令, NX_ModbusRtuWrite 命令を指します。
  a）運転開始後または、運転モードに変化させた場合
  b）前回の命令実行時にリトライの設定がされていた(「Option.Retry」≠ 0)場合
  c）前回の命令実行が中断された（出力変数「CommandAborted」= TRUE）場合
  d）前回の命令実行でエラー発生(「Error」= TRUE)した場合
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「SlaveAdr」、「WriteCmd.Fun」、「WriteCmd.WriteSize」、「Option.Retry」、「DevicePort.DevicePortType」、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b）「WriteDat」に指定した変数が、「WriteCmd.WriteSize」で指定したサイズより小さい場合。

c）「DevicePort」で存在しないユニットあるいはポートを指定した場合。
d）「DevicePort」、「WriteDat」のデータ型が不正な場合。
e）NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
f）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
対象となる命令は、NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令です。
g）受信データでパリティエラーが発生した場合。
h）受信データでフレーミングエラーが発生した場合。
i）受信データでオーバーランエラーが発生した場合。
j）受信データが CRC 不一致の場合。
k）タイムアウト時間を経過した場合。
l）NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。
m）Modbus-RTU スレーブから Exception Response を受信した場合。Exception Code は、出力変数「ErrorIDEx」で確認します。
n）Modbus-RTU スレーブからのレスポンスデータのファンクションコードや受信サイズが不正な場合。

- 拡張エラーコード「ErrorIDEx」は、本命令では、Modbus-RTU スレーブで異常を検出したときに表示します。「ErrorIDEx」は、エラーコード「ErrorID」の値が「WORD#16#0C10」の場合に出力され、「ErrorIDEx=000000XX」と表示します。「XX」については、MODBUS 通信プロトコルの Exception Code 仕様を確認してください。
MODBUS 通信プロトコルの Exception Code 仕様については、「MODBUS Application Protocol Specification」を参照してください。
「MODBUS Application Protocol Specification」は、MODBUS Organization, Inc.から入手できます。
http://www.modbus.org/

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。
形 NX-CIF210 のユニット動作設定の［Ch2 終端検出キャラクタ数］を「35」に設定します。終端検出キャラクタ数は、単位が 0.1 キャラクタ数のため、3.5 キャラクタとして動作します。



「Trigger」が ON になると、形 NX-CIF210 のシリアルポート 2 のバッファをクリアし、その後 Modbus-RTU コマンドを送信します。
スレーブアドレス 1 の書き込み開始アドレス 149 から 1 個、コイルを ON にします。
書き込みコマンド送受信を使用し、変数書き込みを実行します。

ST

内部変数	名称	データ型	初期値	コメント
	Stage	INT	0	
	Trigger	BOOL	FALSE	実行条件
	DevicePort	_sDEVICE_PORT		ポート指定
	NX_SerialBufClear_instance	NX_SerialBufClear		バッファクリア
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuWrite_instance	NX_ModbusRtuWrite		
	ModbusSlaveAdr	UINT	UINT#0	スレーブアドレス
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		
	ModbusError	BOOL		
	DoModbusTrigger	BOOL		
	ModbusWriteDat	ARRAY[0..5] OF BOOL	[6(FALSE)]	
	ModbusWriteCmd	_sSERIAL_MODBUS RTU_WRITE		

外部変数	名称	データ型	コンスタント	コメント
	N1_Node_location_information	_sNXUNIT_ID	☑	

```
//Trigger の立ち上がりでシーケンス開始
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; //初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1: // バッファクリア要求
        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; //バッファクリア　正常終了
        ELSIF (ClearError = TRUE) THEN
            Stage := 99; //バッファクリア　異常終了
        END_IF;

    2: //Modbus  Write 要求
        ModbusSlaveAdr := 1; // スレーブアドレス
        ModbusWriteCmd.Fun:=_MDB_WRITE_SINGLE_COIL; // ファンクションコード
        ModbusWriteCmd.WriteAdr:=149; // 書き込みアドレス
        ModbusWriteCmd.WriteSize:=1; // 書き込みサイズ

        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; //NX_ModbusRtuWrite 命令　正常終了
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
            Stage :=99; //NX_ModbusRtuWrite 命令　異常終了 or Abort
        END_IF;

    3: //NX_ModbusRtuWrite 命令正常終了後の処理
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
```

```
    99: //異常処理
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
    END_CASE;
END_IF;
```

NX_SerialSigCtl

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートの ER 信号または RS 信号を ON または OFF にします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	FB	NX_SerialSigCtl_instance NX_SerialSigCtl Execute / DevicePort / Kind / Sig / TimeOut Done / Busy / Error / ErrorID	NX_SerialSigCtl_instance(Execute, DevicePort, Kind, Sig, TimeOut, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	–	–	–
Kind	信号指令		信号指令	_RS_SIG _ER_SIG *1	–	*2
Sig	ON/OFF 指令		ON/OFF 指令	データ型に従う	–	*2
TimeOut	タイムアウト時間		タイムアウト時間 「0」の場合は、2.0s	データ型に従う	0.1s	0

*1. _CS_SIG と_DR_SIG は、使用できません。指定した場合、命令実行時に異常となります。
*2. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常となります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1206)」参照																			
Kind	列挙型_eSERIAL_SIG　列挙子は「機能 (P.2-1206)」参照																			
Sig	○																			
TimeOut							○													

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルポートの ER 信号または RS 信号を ON または OFF にします。

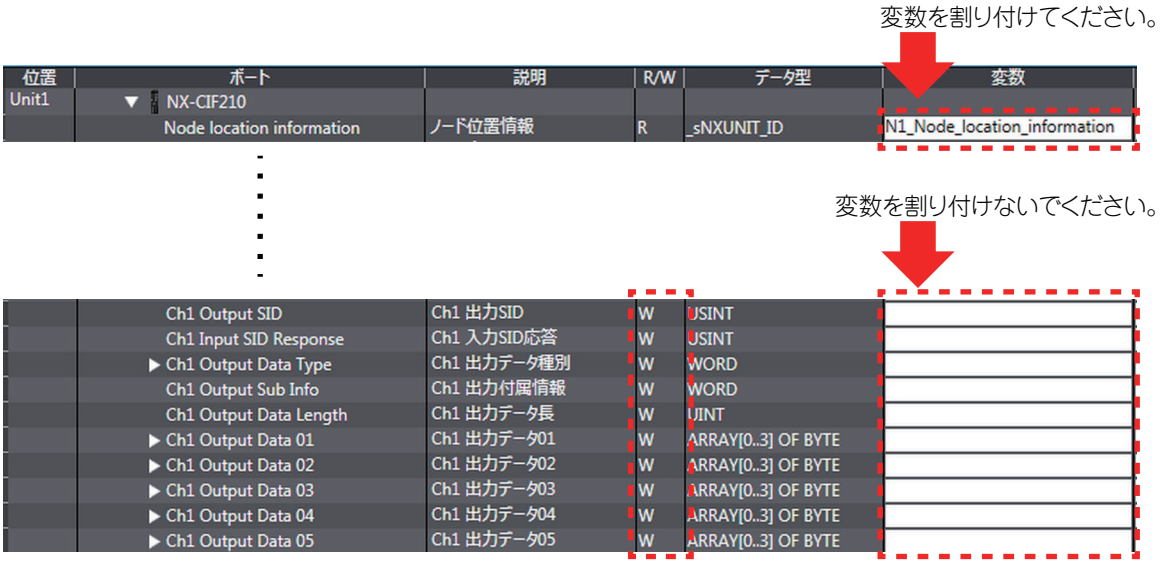
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	−	−	−
Reserved	予約	予約	予約	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2

NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

ER 信号または RS 信号は、入力変数「Kind」で選択します。
入力変数「Sig」が TRUE の場合、ER 信号または RS 信号が ON になります。
入力変数「Sig」が FALSE の場合、ER 信号または RS 信号が OFF になります。

入力変数「Kind」のデータ型は、列挙型_eSERIAL_SIG です。
列挙型_eSERIAL_SIG の、列挙子の意味は以下のとおりです。

列挙子	意味
_RS_SIG	RS 信号
_ER_SIG	ER 信号
_CS_SIG	CS 信号
_DR_SIG	DR 信号

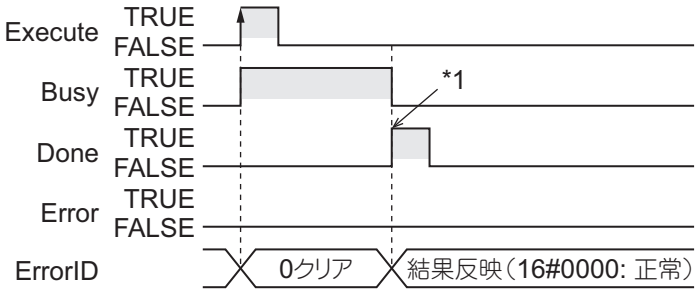
使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

タイミングチャート

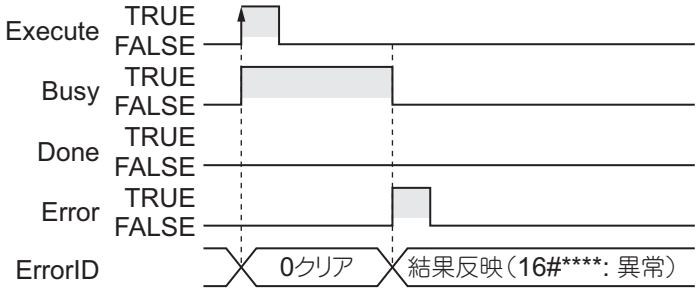
タイミングチャートを、以下に示します。

● 正常終了時



*1. 信号の ON/OFF 制御が完了

● **異常終了時**

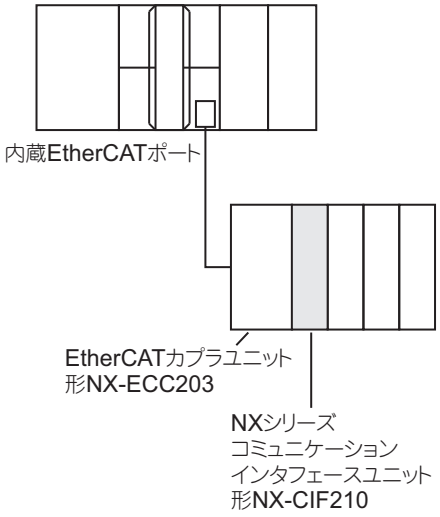


使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- 本命令は、通信プロトコルや配線状態によるチェックは行いません。配線状態や通信プロトコルを確認の上、使用してください。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「Kind」、「DevicePort.DevicePortType」 、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b）「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
  c）「DevicePort」で RS-422A/485 のシリアルポートを指定した場合。
  d）NX シリーズ コミュニケーションインタフェースユニットのフロー制御方法の設定が［RS/CS フロー制御］の場合に、本命令で「RS 信号 ON」または「RS 信号 OFF」を送信した。
  e）NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  f）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令です。
  g）タイムアウト時間を経過した場合。
  h）NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。



形 NX-CIF210 のシリアルポート 2 に接続した無手順の通信相手に、SetER 信号を ON にすると ER 信号を ON にし、ResetER 信号を ON にすると ER 信号を OFF にします。

グローバル変数の定義

● グローバル変数

名称	データ型	割付先	コメント
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	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

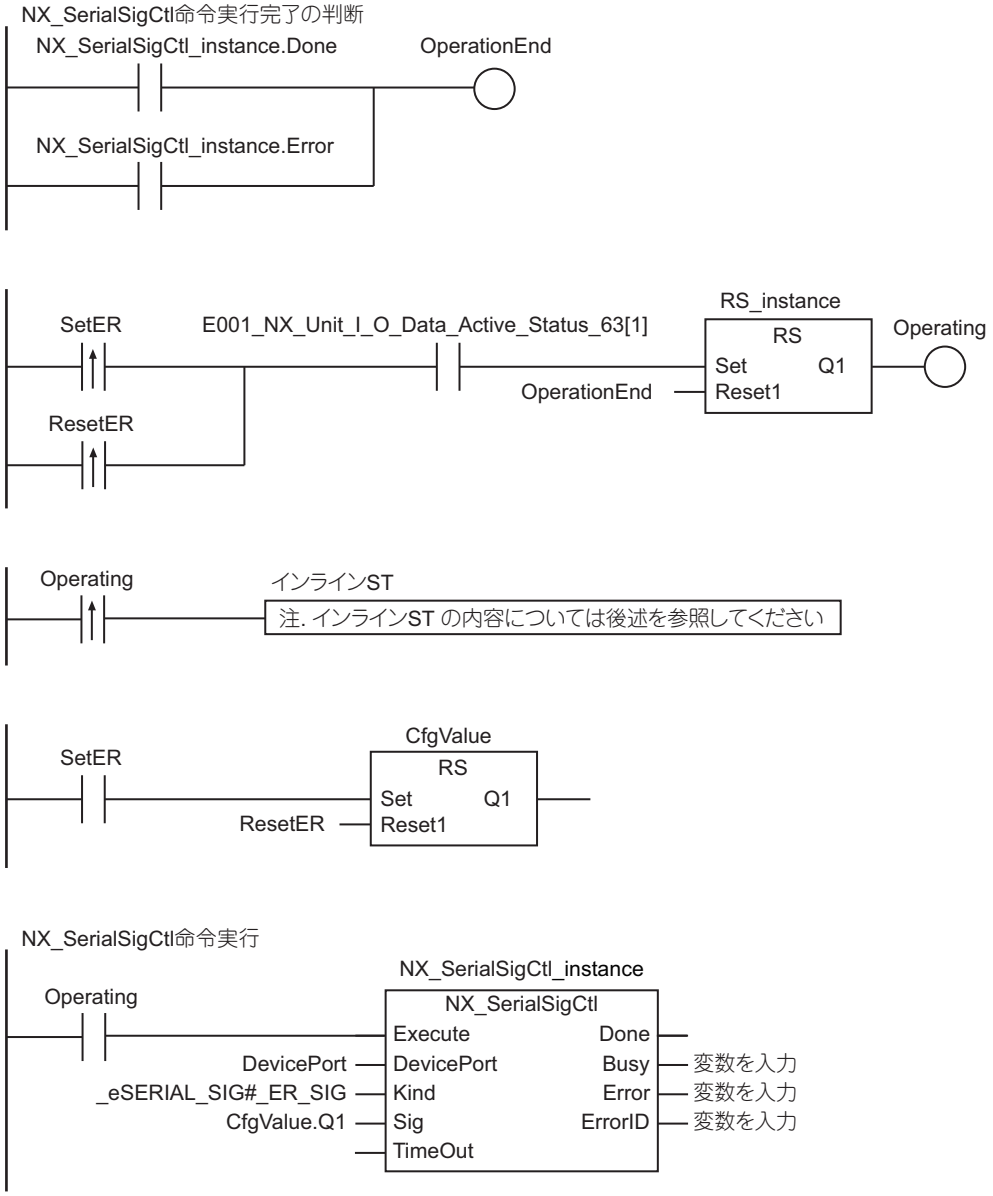
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

LD

内部変数	名称	データ型	初期値	コメント
	OperationEnd	BOOL	FALSE	処理終了
	SetER	BOOL	FALSE	ER 信号 ON 実行条件
	ResetER	BOOL	FALSE	ER 信号 OFF 実行条件
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	RS_instance	RS	−	「Operating」の保持
	CfgValue	RS	−	「SetER/ResetER」に応じて値を決める
	NX_SerialSigCtl_instance	NX_SerialSigCtl	−	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用

外部変数	名称	データ型	コメント
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数



● インライン ST の内容

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

内部変数	名称	データ型	初期値	コメント
	OperatingStart	BOOL	FALSE	処理開始
	SetER	BOOL	FALSE	ER 信号 ON 実行条件
	ResetER	BOOL	FALSE	ER 信号 OFF 実行条件
	DevicePort	_sDEVICE_PORT		ポート指定

内部変数	名称	データ型	初期値	コメント
	CfgValue	RS	−	「SetER/ResetER」に応じて値を決定
	NX_SerialSigCtl_instance	NX_SerialSigCtl	−	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

```
// SetER または ResetER の検出
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 信号の値を決める
CfgValue(Set:=SetER, Reset1:=ResetER);

// NX_SerialSigCtl 命令の実行
NX_SerialSigCtl_instance(Execute:=OperatingStart,
                         DevicePort:=DevicePort,
                         Kind:=_eSERIAL_SIG#_SIG_ER,
                         Sig:=CfgValue.Q1);
```

NX_SerialBufClear

送受信バッファをクリアします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialBufClear	バッファクリア	FB	NX_SerialBufClear_instance NX_SerialBufClear Execute / Done DevicePort / Busy BufKind / Error TimeOut / ErrorID	NX_SerialBufClear_instance(Execute, DevicePort, BufKind, TimeOut, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	–	–	–
BufKind	バッファの種類		送信、または受信バッファの種類の指定	_BUF_SENDRCV _BUF_SEND _BUF_RCV	–	_BUFSENDRCV
TimeOut	タイムアウト時間		タイムアウト時間 「0」の場合は、2.0s	データ型に従う	0.1s	0

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1213)」参照																			
BufKind	列挙型_eSERIAL_BUF_KIND　列挙子は「機能 (P.2-1213)」参照																			
TimeOut							○													

機能

ポートとバッファの種類を指定し、そのバッファのデータをクリアします。クリアが完了すると正常終了します。

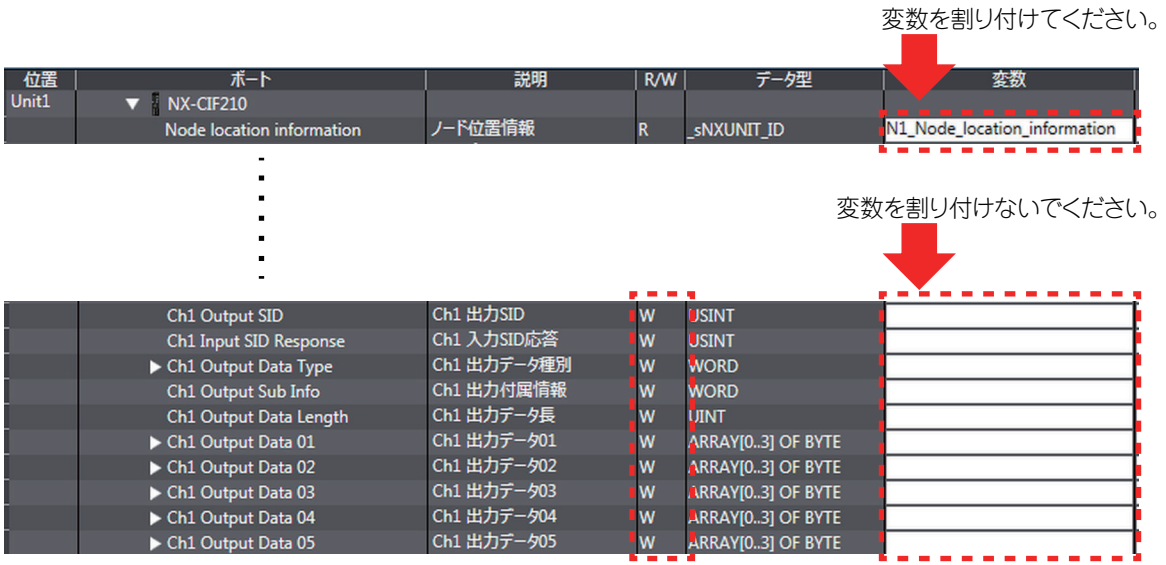
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	−	−	−
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	−	−
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	−	−	−
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	−	−	−
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	−	−	−
Reserved	予約	予約	予約	−	−	−
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	−	−

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit をを設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

NX ユニットを指定する場合は、「NxUnit」を使用してデバイスを指定します。
この場合、「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2

NX ユニットの場合は、ポート 1、またはポート 2 を指定します。

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では_DeviceNXUnit を指定できます。

ポートは「Port」で、クリアするバッファは「BufKind」で指定します。

受信バッファをクリアしたあとに NX シリーズ コミュニケーションインタフェースユニットが外部機器から受信したデータは、クリアしません。

入力変数「BufKind」のデータ型は、列挙型_eSERIAL_BUF_KIND です。
列挙型_eSERIAL_BUF_KIND の、列挙子の意味は以下のとおりです。

列挙子	意味
_BUF_SENDRCV	送信バッファと受信バッファ
_BUF_SEND	送信バッファ
_BUF_RCV	受信バッファ

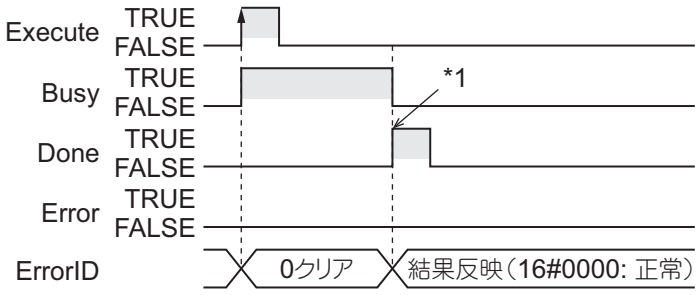
使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

タイミングチャート

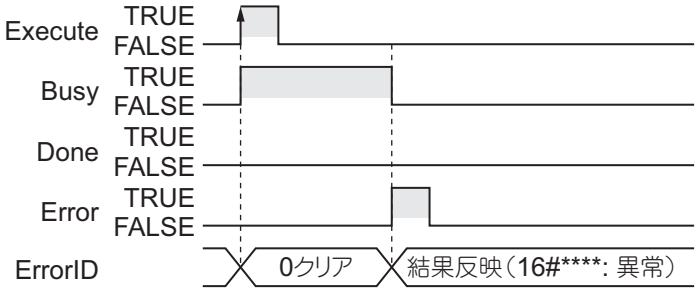
タイミングチャートを、以下に示します。

● 正常終了時



*1. バッファのクリアが完了

● **異常終了時**

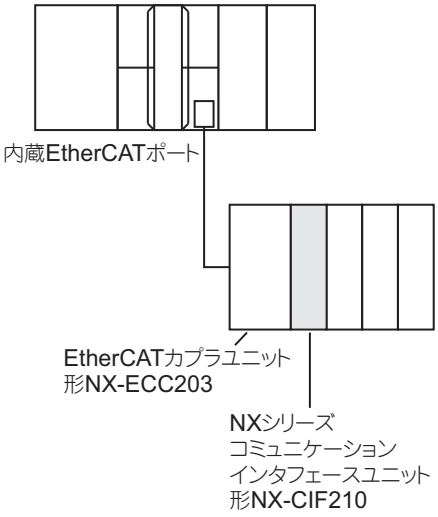


使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- 本命令は、通信プロトコルや配線状態によるチェックは行いません。配線状態や通信プロトコルを確認の上、使用してください。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）「BufKind」、「DevicePort.DevicePortType」 、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b）「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
  c）NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  d）実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令です。
  e）タイムアウト時間を経過した場合。
  f）NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

サンプルプログラム

EtherCAT カプラユニット形 NX-ECC203 に NX シリーズ コミュニケーションインタフェースユニット形 NX-CIF210 を接続する構成とします。
形 NX-CIF210 のユニット番号は「1」とします。



形 NX-CIF210 のシリアルポート 2 の受信バッファをクリアします。クリアが完了すると、スタートコードはなし、エンドコードは「CR」の受信待ちとします。

グローバル変数の定義

● グローバル変数

名称	データ型	割付先	コメント
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	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

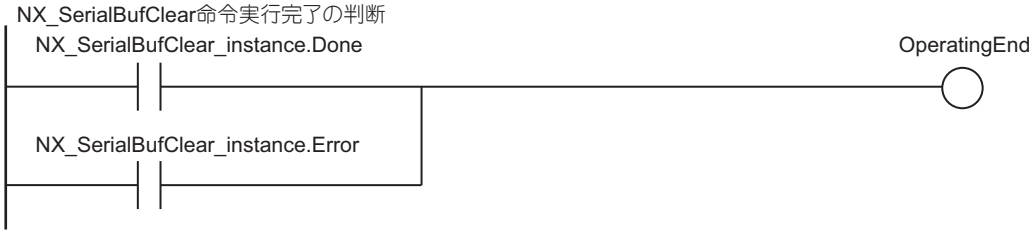
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

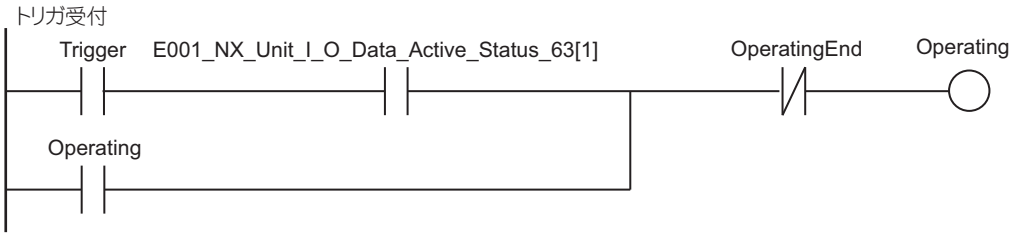
LD

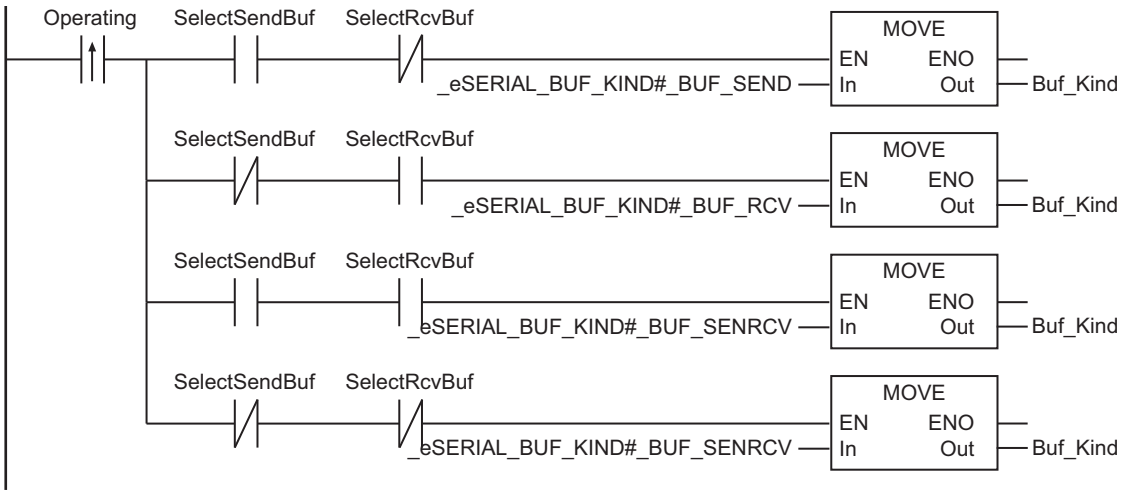
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	バッファクリア処理終了
	Trigger	BOOL	FALSE	バッファクリア実行条件
	Operating	BOOL	FALSE	バッファクリア処理中
	SelectSendBuf	BOOL	FALSE	送信バッファ選択
	SelectRcvBuf	BOOL	FALSE	受信バッファ選択
	BufKind	_eSERIAL_BUF_KIND	_BUF_SENDRCV	バッファ指定
	DevicePort	_sDEVICE_PORT		ポート指定
	NX_SerialBufClear_instance	NX_SerialBufClear	−	
	RcvingEnd	BOOL		受信処理完了
	Rcving	BOOL		受信処理中
	RcvCfg	_sSERIAL_CFG		受信完了設定

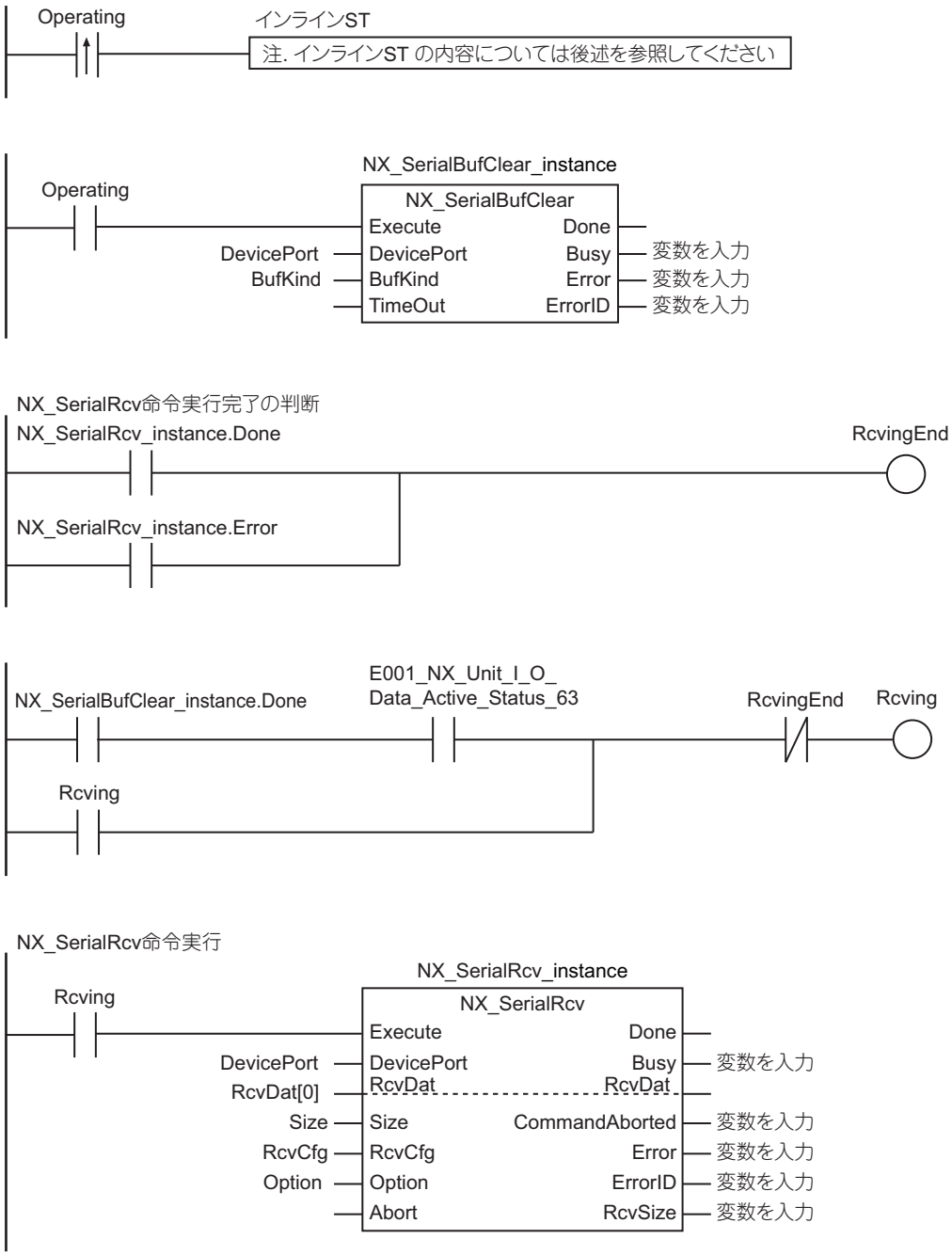
内部変数	名称	データ型	初期値	コメント
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	
	StartCode	ARRAY[0..1] OF BYTE	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_CODE1	
	EndCode	ARRAY[0..1] OF BYTE	[16#0D, 16#00]	エンドコード: CR
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	
	ClearBuf	BOOL	FALSE	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数









● インライン ST の内容

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	バッファクリア処理終了
	Trigger	BOOL	FALSE	バッファクリア実行条件
	Operating	BOOL	FALSE	バッファクリア処理中
	SelectSendBuf	BOOL	FALSE	送信バッファ選択

内部変数	名称	データ型	初期値	コメント
	SelectRcvBuf	BOOL	FALSE	受信バッファ選択
	BufKind	_eSERIAL_BUF_KIND	_BUF_SENDRCV	バッファ指定
	DevicePort	_sDEVICE_PORT		ポート指定
	NX_SerialBufClear_instance	NX_SerialBufClear	−	
	RcvingEnd	BOOL		受信処理完了
	Rcving	BOOL		受信処理中
	RcvCfg	_sSERIAL_CFG		受信完了設定
	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]	エンドコード: CR
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	
	ClearBuf	BOOL	FALSE	

外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

```
// 条件設定
RS_instance1(Set:=Trigger AND E001_NX_Unit_I_O_Data_Active_Status_63[1]
             Reset1:=OperatingEnd,
             Q1=>Operating);
R_Trigger_instance(Clk:=Operating);
IF ( (R_Trigger_instance.Q=TRUE) ) THEN
          DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
          DevicePort.NxUnit:=N1_Node_location_information;
          DevicePort.PortNo:=2;
          IF( (SelectSendBuf=TRUE) THEN
                    IF(SelectRcvBuf=TRUE) THEN
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_SENDRCV;
                    ELSE
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_SEND;
                    END_IF;
          ELSE
                    IF (SelectRcvBuf=TRUE) THEN
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_RCV;
                    ELSE
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_SENDRCV;
                    END_IF
          END_IF;
```

```
END_IF;

// バッファクリア実行
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

NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを開始します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialStartMon	シリアルラインモニタの開始	FB	NX_SerialStartMon_instance NX_SerialStartMon Execute / Done DevicePort / Busy Continuous / Error TimeOut / ErrorID	NX_SerialStartMon_instance(Execute, DevicePort, Continuous, TimeOut, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	−	−	−
Continuous	連続モニタ		シリアルラインモニタの動作方法の指定 TRUE: 連続 FALSE: ワンショット	データ型に従う	−	FALSE
TimeOut	タイムアウト時間		タイムアウト時間 「0」の場合は、2.0s	データ型に従う	0.1s	0

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1222)」参照																			
Continuous	○																			
TimeOut							○													

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを開始します。
シリアルラインモニタが起動すると正常終了します。

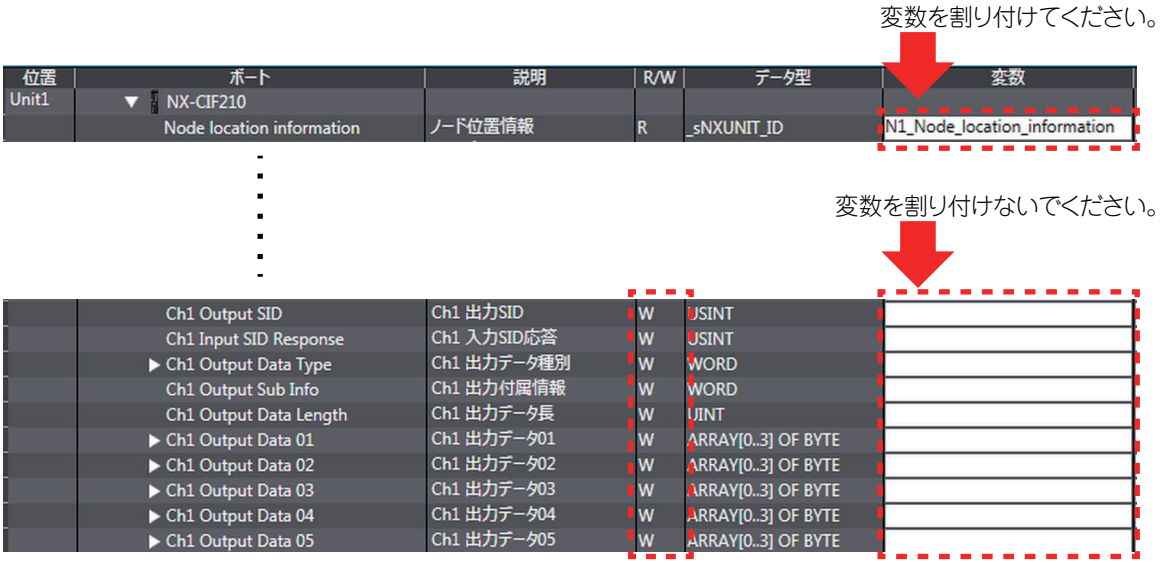
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	－	－	－
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	－	－
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	－	－	－
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	－	－	－
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	－	－	－
Reserved	予約	予約	予約	－	－	－
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	－	－

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

この命令では「NxUnit」を使用してデバイスを指定します。「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では、_DeviceNXUnit を指定できます。

入力変数「Continuous」が TRUE の場合は連続モニタとなり、NX_SerialStopMon 命令が実行されるまでモニタを継続します。
入力変数「Continuous」が FALSE の場合はワンショットモニタとなり、バッファがいっぱいになるか、NX_SerialStopMon 命令が実行されるまでシリアルラインモニタを継続します。

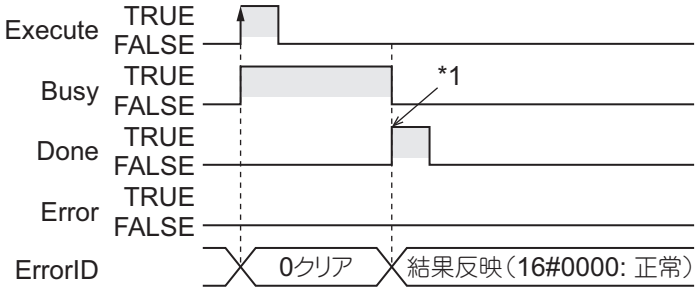
使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

タイミングチャート

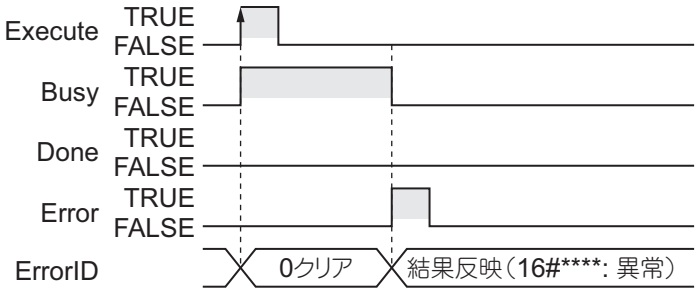
タイミングチャートを、以下に示します。

● 正常終了時



*1. シリアルラインモニタを開始

● 異常終了時



使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「DevicePort.DevicePortType」、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b)「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
  c) NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  d) 実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令です。
  e) タイムアウト時間を経過した場合。
  f) NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

NX_SerialStopMon

NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを停止します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialStopMon	シリアルラインモニタの停止	FB	NX_SerialStopMon_instance NX_SerialStopMon Execute / DevicePort / TimeOut Done / Busy / Error / ErrorID	NX_SerialStopMon_instance(Execute, DevicePort, TimeOut, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DevicePort	デバイスポート	入力	デバイスポートを表すオブジェクト	–	–	–
TimeOut	タイムアウト時間		タイムアウト時間 「0」の場合は、2.0s	データ型に従う	0.1s	0

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	構造体_sDEVICE_PORT　詳細は「機能 (P.2-1226)」参照																			
TimeOut							○													

機能

NX シリーズ コミュニケーションインタフェースユニットのシリアルラインモニタを停止します。
シリアルラインモニタが停止すると正常終了します。

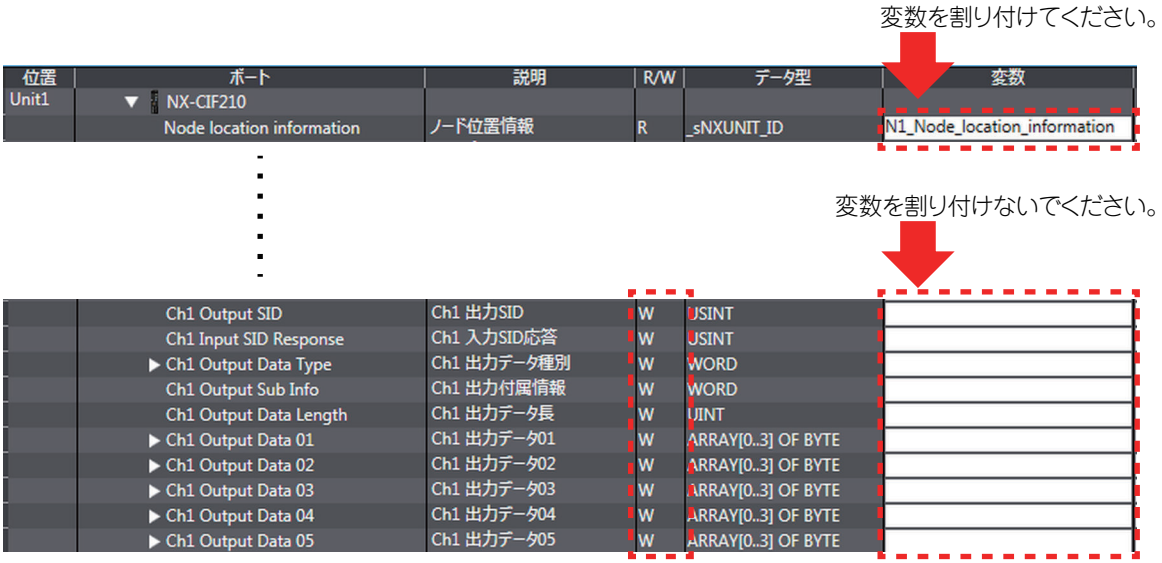
入力変数「DevicePort」のデータ型は、構造体_sDEVICE_PORT です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
DevicePort	デバイスポート	デバイスポートを表すオブジェクト	_sDEVICE_PORT	－	－	－
DeviceType	デバイスタイプ	指定するデバイスの種別を指定	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	－	－
NxUnit	指定ユニット	制御対象の NX ユニットを指定	_sNXUNIT_ID	－	－	－
EcatSlave	指定スレーブ	制御対象の EtherCAT スレーブを指定	_sECAT_ID	－	－	－
OptBoard	指定オプションボード	制御対象のオプションボードを指定	_sOPTBOARD_ID	－	－	－
Reserved	予約	予約	予約	－	－	－
PortNo	ポート番号	ポート番号の指定 1: ポート 1 2: ポート 2	USINT	データ型に従う	－	－

「DeviceType」でデバイスの種別を指定します。
NX ユニットの場合は_DeviceNXUnit を設定してください。
指定した種別により、デバイス指定に使用する変数が決まっています。

この命令では「NxUnit」を使用してデバイスを指定します。「EcatSlave」、「OptBoard」は使用しません。
「NxUnit」には、指定するデバイスの I/O マップ上のノード位置情報に割り付けられたデバイス変数を渡してください。

この命令を使用する場合は、必ずノード位置情報にデバイス変数を割り付け、ノード位置情報以降の I/O ポートで、R/W 欄が「W」のものにはデバイス変数を割り付けないでください。
例えば、形 NX-CIF210 のポート 1 で、この命令を使用する場合は、次のようになります。



ノード位置情報にデバイス変数を割り付ける方法は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

「PortNo」でポート番号を指定します。
1: ポート 1
2: ポート 2

「DeviceType」のデータ型は、列挙型_eDEVICE_TYPE です。
列挙型_eDEVICE_TYPE の、列挙子の意味は以下のとおりです。

列挙子	意味
_DeviceNXUnit	NX ユニットを指定
_DeviceEcatSlave	EtherCAT スレーブを指定
_DeviceOptionBoard	オプションボードを指定

この命令では、_DeviceNXUnit を指定できます。

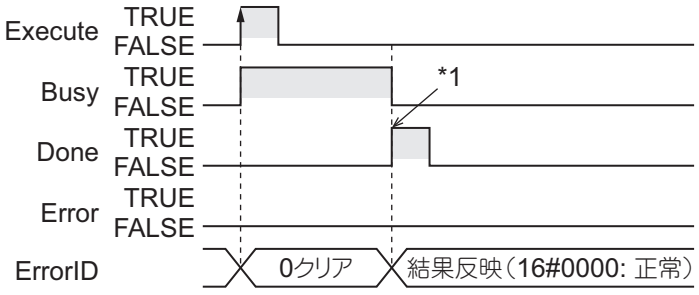
使用上の注意

NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行すると、異常となります。

タイミングチャート

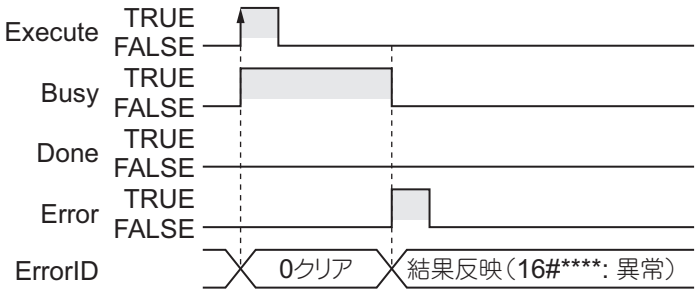
タイミングチャートを、以下に示します。

● 正常終了時



*1. シリアルラインモニタを停止

● 異常終了時



使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 本命令をイベントタスクに使用した場合は、コンパイルエラーとなります。そのため、イベントタスクでは使用することができません。

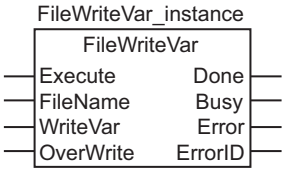
- NX シリーズ コミュニケーションインタフェースユニットがリスタートされると「CIF ユニット初期化」が発生する場合があります。必要に応じてデータの送信または受信をやりなおしてください。
- 本命令を使用する場合、対象となる NX シリーズ コミュニケーションインタフェースユニットに対して、Sysmac Studio の I/O マップ画面の R/W 欄が「W」である I/O ポートに、デバイス変数を割り付けないでください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a)「DevicePort.DevicePortType」 、「DevicePort.PortNo」に範囲外の値を指定した場合。
  b)「DevicePort」で存在しないユニット、オプションボード、あるいはポートを指定した場合。
  c) NX_SerialSend 命令、NX_SerialRcv 命令、NX_ModbusRtuCmd 命令、NX_ModbusRtuRead 命令、NX_ModbusRtuWrite 命令、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令を、同時に 32 個超えて実行した場合。
  d) 実行中の以下の命令で指定したデバイスポート変数と、同じデバイスポート変数を指定して本命令を実行した場合。
  対象となる命令は、NX_SerialSigCtl 命令、NX_SerialBufClear 命令、NX_SerialStartMon 命令、NX_SerialStopMon 命令です。
  e) タイムアウト時間を経過した場合。
  f) NX シリーズ コミュニケーションインタフェースユニット以外のユニットに本命令を実行した場合。

SD メモリカード命令

命令	名称	ページ
FileWriteVar	変数ファイル書込	P.2-1232
FileReadVar	変数ファイル読出	P.2-1237
FileOpen	ファイルオープン	P.2-1242
FileClose	ファイルクローズ	P.2-1246
FileSeek	ファイルシーク	P.2-1249
FileRead	ファイル読出	P.2-1252
FileWrite	ファイル書込	P.2-1260
FileGets	文字列読出	P.2-1268
FilePuts	文字列書込	P.2-1275
FileCopy	ファイルコピー	P.2-1284
FileRemove	ファイル削除	P.2-1291
FileRename	ファイル名変更	P.2-1295
DirCreate	ディレクトリ作成	P.2-1300
DirRemove	ディレクトリ削除	P.2-1303
BackupToMemoryCard	SD メモリカードバックアップ	P.2-1306

FileWriteVar

SD メモリカード内の指定したファイルに、1 個の変数の値を、バイナリ形式で書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileWriteVar	変数ファイル書込	FB		FileWriteVar_instance(Execute, FileName, WriteVar, OverWrite, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	指定ファイル名	入力	書き込むファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	–	''
WriteVar	指定変数		書き込む変数	データ型に従う		*1
OverWrite	上書許可		TRUE: 上書許可 FALSE: 上書禁止			FALSE

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○
WriteVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
OverWrite	○																			

機能

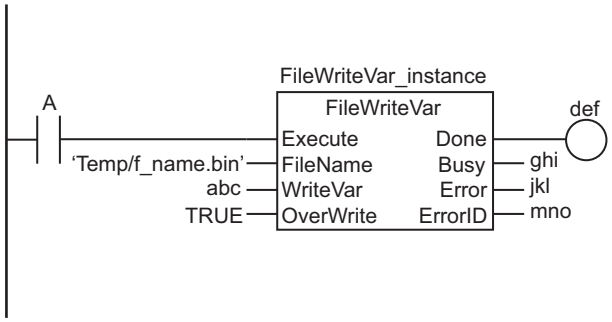
SD メモリカード内の、「FileName」で指定したファイルに、1 個の変数「WriteVar」の値を、バイナリ形式で書き込みます。
「WriteVar」には、列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバを指定することも可能です。

SD メモリカード内に、「FileName」と同じ名前のファイルが存在しない場合は、新規にファイルを作成します。
「FileName」がディレクトリを含めた記述になっており、SD メモリカード内にそのディレクトリも存在しない場合は、ディレクトリごと新規作成します。ただし、ディレクトリごと新規作成するのは、指定されたディレクトリの最下層だけが存在しない場合です。
SD メモリカード内に、「FileName」と同じ名前のファイルがすでに存在している場合、上書許可「OverWrite」の値によって、以下のような処理を行います。

「OverWrite」の値	処理
TRUE（上書許可）	そのファイルに、上書します。
FALSE（上書禁止）	そのファイルに上書せず、異常になります。

記述例を、以下に示します。
'Temp/f_name.bin'という名前のファイルに、配列変数「abc」の全体を書き込みます。「abc」は、要素数 3 の INT 型配列変数とします。

LD



ST

```
FileWriteVar_instance(A, 'Temp/f_name.bin', abc,
                      TRUE, def, ghi, jkl, mno);
```

SDメモリカード内の、「FileName」で指定したファイルに、
変数「WriteVar」の値をバイナリ形式で書き込みます。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 命令内に書き込むデータ全体を保持しません。
  命令の実行タイミングとは同期せず、その都度「WriteVar」に渡した変数を参照するため、値の同時性は保証されません。
  命令実行中は対象の変数へアクセスしないようにしてください。
  命令実行中にアクセスした場合、意図しない値がファイルに書き込まれる可能性があります。
- 「WriteVar」に渡す入力パラメータは、必ず変数にしてください。定数を渡すとビルド時に異常になります。
- 「WriteVar」が列挙型の場合、列挙子を直接渡すことはできません。列挙子を直接渡すとビルド時に異常になります。
- 指定ファイルのサイズが、「WriteVar」のサイズより大きい場合、異常は発生せず、「WriteVar」のサイズ分だけデータを書き込みます。そして、本命令実行後、指定ファイルは「WriteVar」のサイズに合わせて小さくなります。
- 書き込まれるデータは、バイト単位で、下位バイト→上位バイトの順（リトルエンディアン）に並びます。
- 「WriteVar」が構造体全体の場合、その構成に応じて、各メンバ間に調整用の領域が挿入されることがあります。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- FileWriteVar（変数ファイル書込）命令を使用して SD メモリカードにデータの書き込みを行い、正常終了しても、SD メモリカードにデータが正しく書き込まれていない場合があります。
  SD メモリカードに正しくデータが書き込まれているか確認する場合は、書き込んだデータを FileReadVar（変数ファイル読出）命令で読み出し、元のデータと照合するようにユーザプログラムを作成してください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）SD メモリカードが、空き容量不足の場合。
  d）「FileName」の値が、ファイル名として不正の場合。
  e）「FileName」と同じ名前のファイルがすでに存在していて、かつ、アクセス中の場合。
  f）「FileName」と同じ名前のファイルがすでに存在していて、かつ、「OverWrite」の値が FALSE の場合。
  g）「FileName」と同じ名前のファイルがすでに存在していて、かつ、そのファイルが書込禁止の場合。
  h）「FileName」の値が、ファイル名として使用できるバイト数を超えている場合。
  i） 作成可能なファイル数、ディレクトリ数を超える場合。
  j） 変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
  k）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

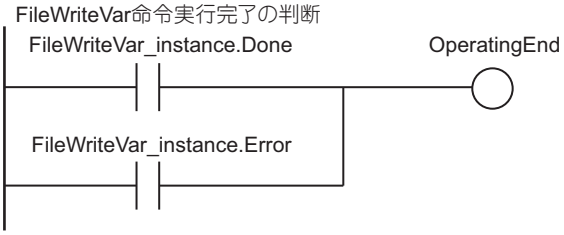
配列変数 Var1[]の全体を、ファイル'File1.dat'に書き込みます。

LD

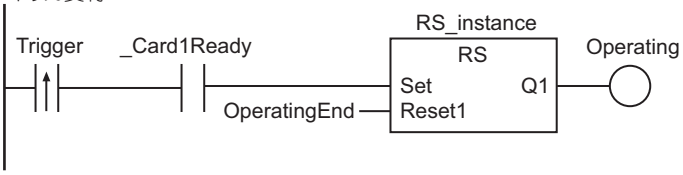
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Var1	ARRAY[0..999] OF INT	[1000(0)]	書込データ
	RS_instance	RS		
	FileWriteVar_instance	FileWriteVar		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

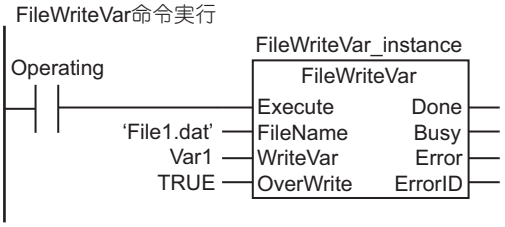
FileWriteVar命令実行完了の判断



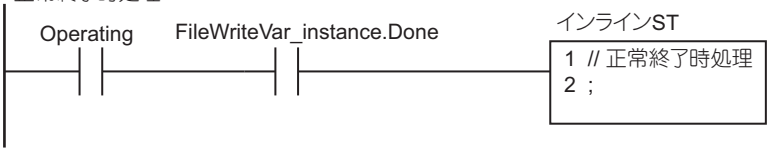
トリガ受付



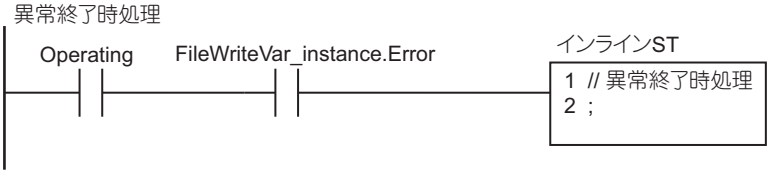
FileWriteVar命令実行



正常終了時処理



異常終了時処理



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	Var1	ARRAY[0..999] OF INT	[1000(0)]	指定変数
	FileWriteVar_instance	FileWriteVar		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// FileWriteVar 命令初期化
IF (OperatingStart=TRUE) THEN
    FileWriteVar_instance(
        Execute    :=FALSE,
        WriteVar   :=Var1);
    OperatingStart:=FALSE;
END_IF;

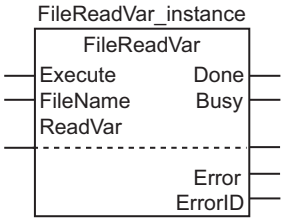
// FileWriteVar 命令実行
IF (Operating=TRUE) THEN
    FileWriteVar_instance(
        Execute  :=TRUE,
        FileName :='File1.dat',  // 指定ファイル名
        WriteVar :=Var1,           // 指定変数
        OverWrite:=TRUE);          // 上書許可

    IF (FileWriteVar_instance.Done=TRUE) THEN
        // 正常終了時処理
        Operating:=FALSE;
    END_IF;

    IF (FileWriteVar_instance.Error=TRUE) THEN
        // 異常終了時処理
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileReadVar

SD メモリカード内の、指定したファイルの値をバイナリ形式で読み出し、変数に書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileReadVar	変数ファイル読出	FB		FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	指定ファイル名	入力	読み出すファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	−	''
ReadVar	読出先変数	入出力	読み出した値の格納先変数	データ型に従う	−	−

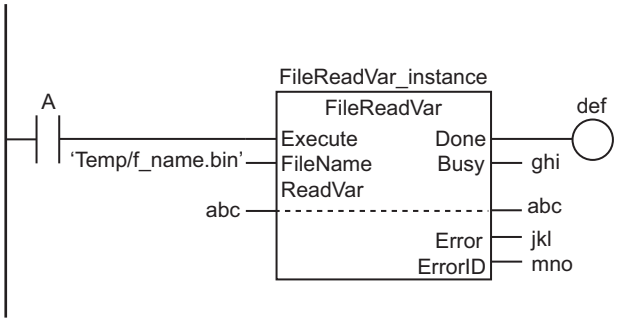
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○
ReadVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			

機能

SD メモリカード内の、「FileName」で指定したファイル内の値を、バイナリ形式で読み出します。読み出した値は、読出先変数「ReadVar」に代入します。
「ReadVar」には、列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバを指定することも可能です。

記述例を、以下に示します。
'Temp/f_name.bin'という名前のファイルの内容を読み出し、配列変数 abc[]に書き込みます。
「abc」は、要素数 3 の INT 型配列変数とします。

LD



ST

```
FileReadVar_instance(A, 'Temp/f_name.bin', abc,
                     def, ghi, jkl, mno);
```

SDメモリカード内の、「FileName」で指定したファイル内の値をバイナリ形式で読み出し、
変数「ReadVar」に代入します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 命令内に読み出したデータ全体を保持しません。
  命令の実行タイミングとは同期せず、その都度「ReadVar」に渡した変数を更新するため、値の同時性は保証されません。
  命令実行中は対象の変数へアクセスしないようにしてください。
  命令実行中にアクセスした場合、ファイルから読み出した値と異なる値で変数が更新される可能性があります。
- 指定ファイルのサイズが、「ReadVar」のサイズより大きい場合、異常は発生せず、「ReadVar」のサイズ分だけデータを読み出します。
- 指定ファイルのサイズが、「ReadVar」のサイズより小さい場合、異常は発生せず、指定ファイルのサイズ分だけデータを読み出します。そして、「ReadVar」の余った領域は、本命令実行前の値が保持されます。
- 読み出されるデータは、バイト単位で、下位バイト→上位バイトの順（リトルエンディアン）に並んでいます。
- 「ReadVar」が構造体全体の場合、その構成に応じて、各メンバ間に調整用の領域が挿入されることがあります。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- 「ReadVar」に、デバイス変数を指定することはできません。指定した場合は、読み出した値が「ReadVar」に代入されません。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) SD メモリカードが、使用可能状態ではない場合。
  b)「FileName」で指定したファイルが、存在しない場合。
  c)「FileName」で指定したファイルが、アクセス中の場合。
  d)「FileName」の値が、ファイル名として不正の場合。
  e)「FileName」の値が、ファイル名として使用できるバイト数を超えている場合。
  f) 変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
  g) SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

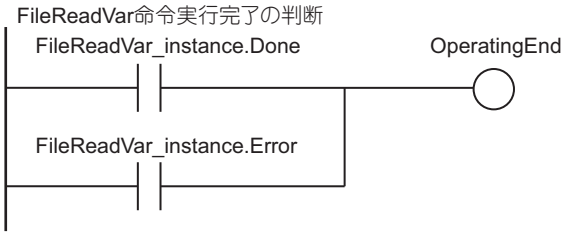
サンプルプログラム

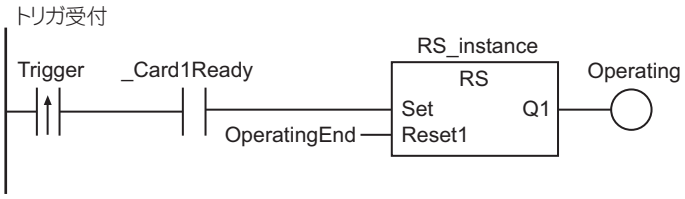
ファイル'File1.dat'の内容を読み込み、配列変数「Var1」に格納します。

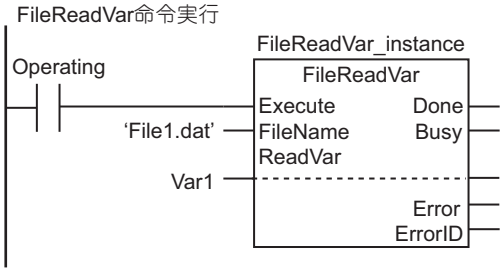
LD

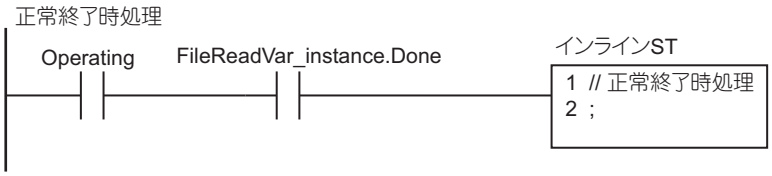
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Var1	ARRAY[0..999] OF INT	[1000(0)]	読出データ
	RS_instance	RS		
	FileReadVar_instance	FileReadVar		

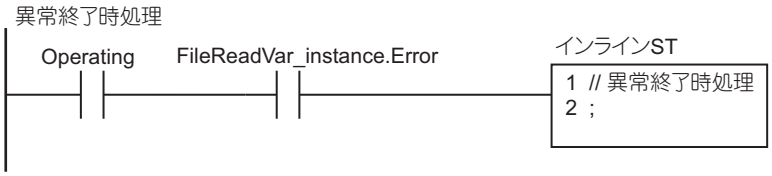
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ











ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	Var1	ARRAY[0..999] OF INT	[1000(0)]	読出先変数
	FileReadVar_instance	FileReadVar		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// FileReadVar 命令初期化
IF (OperatingStart=TRUE) THEN
    FileReadVar_instance(
        Execute    :=FALSE,
        ReadVar    :=Var1);
    OperatingStart:=FALSE;
END_IF;

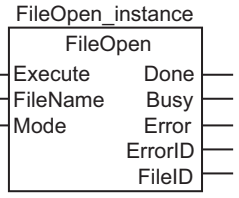
// FileReadVar 命令実行
IF (Operating=TRUE) THEN
    FileReadVar_instance(
        Execute :=TRUE,
        FileName:='File1.dat',     // 指定ファイル名
        ReadVar :=Var1);           // 読出先変数

    IF (FileReadVar_instance.Done=TRUE) THEN
        // 正常終了時処理
        Operating:=FALSE;
    END_IF;

    IF (FileReadVar_instance.Error=TRUE) THEN
        // 異常終了時処理
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileOpen

SD メモリカード内の指定したファイルを、オープンします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileOpen	ファイルオープン	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	指定ファイル名	入力	オープンするファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	—	"
Mode	オープンモード		ファイルオープンするモード	*1		_READ_EXIST
FileID	ファイル ID	出力	オープンしたファイルのファイル ID	データ型に従う	—	—

*1. _READ_EXIST, _RDWR_EXIST, _WRITE_CREATE, _RDWR_CREATE, _WRITE_APPEND, _RDWR_APPEND

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○
Mode	列挙型_eFOPEN_MODE　列挙子は「機能 (P.2-1242)」参照																			
FileID				○																

機能

SD メモリカード内の、「FileName」で指定したファイルを、「Mode」で指定したモードで、オープンします。
ファイルオープンした結果、ファイル ID「FileID」が出力されます。「FileID」は、FileRead 命令、FileWrite 命令などで、ファイルを指定する場合に使用します。

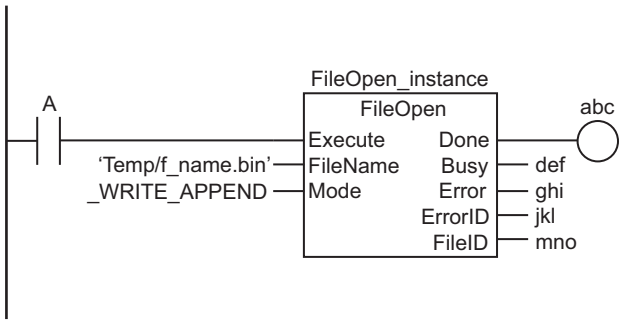
「Mode」のデータ型は、列挙型_eFOPEN_MODE です。列挙子の意味は、以下のとおりです。

列挙子	意味
_READ_EXIST	テキストファイルを読み出すためにオープンします。読出は、ファイルの先頭から始まります。
_RDWR_EXIST	読出および書込用にファイルオープンします。読出および書込は、ファイルの先頭から始まります。

列挙子	意味
_WRITE_CREATE	書込用にファイルオープンします。ファイルがすでに存在する場合は、その内容を破棄し、ファイルサイズを 0 にします。存在しない場合は、ファイルを新規作成します。書込は、ファイルの先頭から始まります。 ただし、ファイルがすでに存在し、かつ、そのファイルに書込禁止が設定されている場合は、異常になり、ファイルオープンできません。
_RDWR_CREATE	読出および書込用にファイルオープンします。ファイルがすでに存在する場合は、その内容を破棄し、ファイルサイズを 0 にします。存在しない場合は、ファイルを新規作成します。読出および書込は、ファイルの先頭から始まります。
_WRITE_APPEND	追加書込用にファイルオープンします。ファイルが存在しない場合は、ファイルを新規作成します。追加書込は、ファイルの末尾から始まります。 ただし、ファイルがすでに存在し、かつ、そのファイルに書込禁止が設定されている場合は、異常になり、ファイルオープンできません。
_RDWR_APPEND	読出および追加書込用にファイルオープンします。ファイルが存在しない場合は、ファイルを新規作成します。読出は、ファイルの先頭から始まります。追加書込は、ファイルの末尾から始まります。

記述例を、以下に示します。
'Temp/f_name.bin'という名前のファイルを、追加書込用にオープンします。変数「mno」にファイル ID が代入されます。

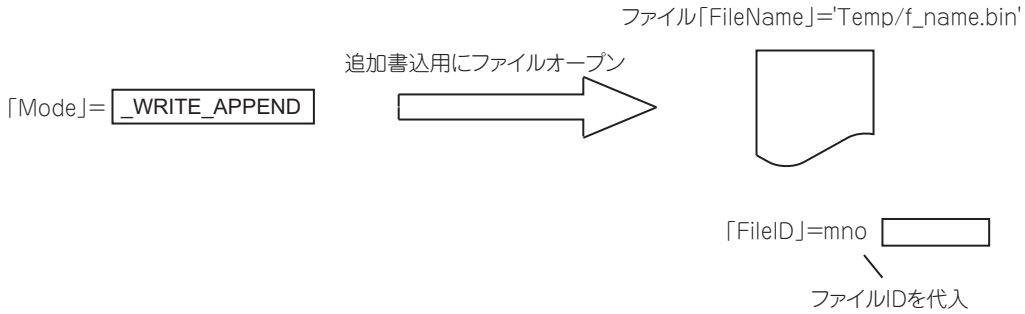
LD



ST

```
FileOpen_instance(A, 'Temp/f_name.bin',
                  _WRITE_APPEND, abc,
                  def, ghi, jkl, mno);
```

SDメモリカード内の、「FileName」で指定したファイルを、追加書込用にオープンします。
変数「FileID」にファイルIDが代入されます。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可

変数名	名称	データ型	内容
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令は、FileSeek 命令、FileRead 命令、FileWrite 命令、FileGets 命令、FilePuts 命令の前に実行しておく必要があります。
- 本命令でファイルオープンしたファイルは、使用後に必ず FileClose 命令を実行して、ファイルクローズしてください。
- 「FileID」に値が格納されるのは、本命令が完了した時点です。つまり、「Done」の値が、FALSE から TRUE に変化した時点です。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）「Mode」の値が_READ_EXIST または_RDWR_EXIST で、かつ、「FileName」で指定したファイルが存在しない場合。
  d）「Mode」の値が、有効範囲外の場合。
  e）「FileName」で指定したファイルが、アクセス中の場合。
  f）「FileName」の値が、ファイル名として不正の場合。
  g）「FileName」で指定したファイルが、書込禁止の場合。

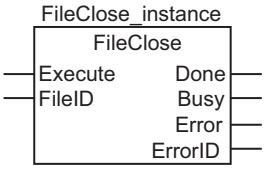
h)「FileName」の値が、ファイル名として使用できるバイト数を超えている場合。
i) 5 個以上のファイルを、同時にオープンしようとした場合。
j) 作成可能なファイル数、ディレクトリ数を超える場合。
k) オープン中のファイルに対してオープンした場合、「ファイルアクセス中」エラーとなり、出力変数「FileID」にオープン中のファイル ID を格納します。ほかのエラーが発生した場合、出力変数「FileID」は変更しません。
l) SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.2-1255)」、FileWrite 命令の「サンプルプログラム (P.2-1262)」、FileGets 命令の「サンプルプログラム (P.2-1270)」、FilePuts 命令の「サンプルプログラム (P.2-1277)」を、参照してください。

FileClose

SD メモリカード内の指定したファイルを、クローズします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileClose	ファイルクローズ	FB		FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	クローズするファイルのファイル ID	データ型に従う	ー	0

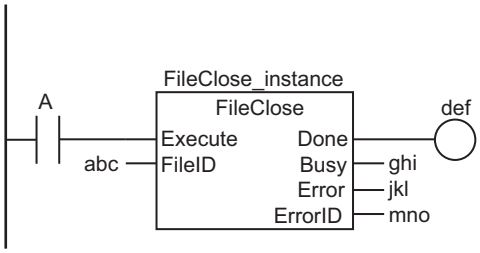
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																

機能

SD メモリカード内の、「FileID」で指定したファイルをクローズします。

記述例を、以下に示します。変数「abc」の値をファイル ID とするファイルを、クローズします。

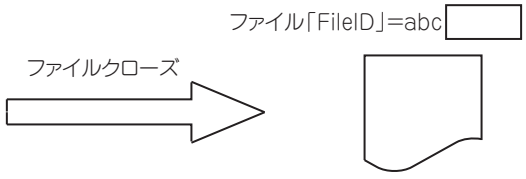
LD



ST

```
FileClose_instance(A, abc, def, ghi, jkl, mno);
```

SDメモリカード内の、「FileID」で指定したファイルをクローズします。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

FileOpen 命令でファイルオープンしておく必要がある命令は、FileSeek 命令、FileRead 命令、FileWrite 命令、FileGets 命令、FilePuts 命令です。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「FileID」は、あらかじめ FileOpen 命令を実行して、取得しておく必要があります。
- FileOpen 命令でファイルオープンしたファイルは、使用後に必ず本命令を実行して、ファイルクローズしてください。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）「FileID」で指定したファイルが、存在しない場合。
  c）「FileID」で指定したファイルが、すでにクローズしている場合。
  d）「FileID」で指定したファイルが、アクセス中の場合。

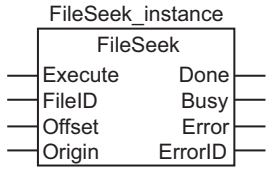
e）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.2-1255)」、FileWrite 命令の「サンプルプログラム (P.2-1262)」、FileGets 命令の「サンプルプログラム (P.2-1270)」、FilePuts 命令の「サンプルプログラム (P.2-1277)」を、参照してください。

FileSeek

SD メモリカード内の指定したファイルに、ファイル位置指示子を設定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileSeek	ファイルシーク	FB		FileSeek_instance(Execute, FileID, Offset, Origin, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	ファイル位置指示子を設定するファイルのファイル ID	データ型に従う	―	0
Offset	オフセット		「Origin」からのオフセット位置		バイト	
Origin	基準位置		フィル位置指示子の基準位置	_SEEK_SET, _SEEK_CUR, _SEEK_END	―	_SEEK_SET

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																
Offset												○								
Origin	列挙型_eFSEEK_ORIGIN　列挙子は「機能 (P.2-1249)」参照																			

機能

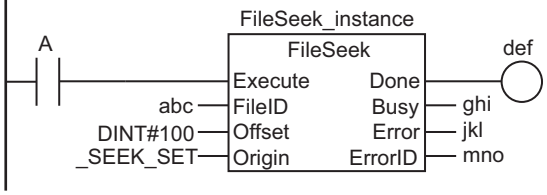
SD メモリカード内のファイル ID「FileID」で指定したファイルに、ファイル位置指示子を設定します。
ファイル位置指示子とは、FileRead 命令や FileWrite 命令などを実行した際に、読出や書込を開始するファイル内の位置のことです。
たとえばファイルの先頭から読出を行いたい場合は、FileSeek 命令でファイル位置指示子をファイルの先頭に設定した後、FileRead 命令を実行します。
ファイル位置指示子が設定されるのは、基準位置「Origin」にオフセット「Offset」を加えた位置です。

「Origin」のデータ型は、列挙型_eFSEEK_ORIGIN です。列挙子の意味は、以下のとおりです。

列挙子	意味
_SEEK_SET	ファイルの先頭
_SEEK_CUR	現在のファイル位置指示子の位置
_SEEK_END	ファイルの末尾

記述例を、以下に示します。ファイルの先頭から 100 バイトの位置に、ファイル位置指示子を設定します。

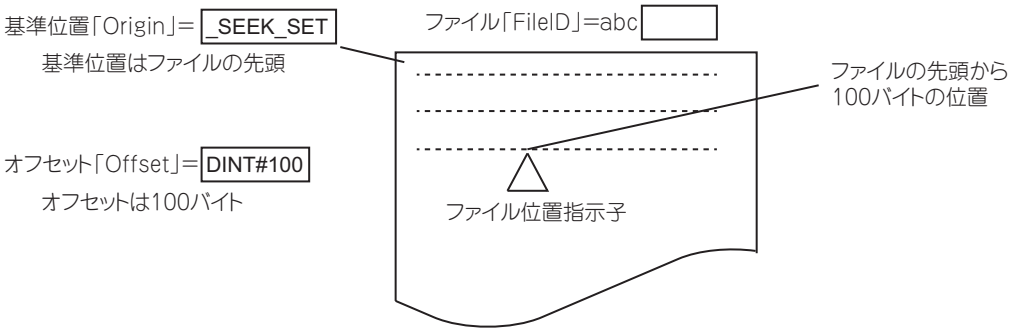
LD



ST

```
FileSeek_instance(A, abc, DINT#100,
                  _SEEK_SET, def,
                  ghi, jkl, mno);
```

SDメモリカード内の「FileID」で指定したファイルに、ファイル位置指示子を設定します。
ファイル位置指示子の位置は、ファイルの先頭から「Offset」を加えた位置です。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

使用上の注意

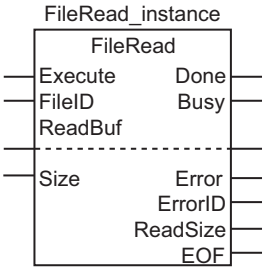
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令を実行する前に、FileOpen 命令を実行して、「FileID」を取得しておく必要があります。
- FileOpen（ファイルオープン）命令を実行する場合、「Mode」に［_WRITE_APPEND］、または［_RDWR_APPEND］を指定して、追加書込用としてファイルをオープンすると、書き込みは常にファイルの末尾に行われます。
  FileOpen（ファイルオープン）命令を実行する場合、「Mode」に［_RDWR_APPEND］を指定すると、FileSeek（ファイルシーク）命令で設定したファイル位置指示子は、読出の場合にだけ使用されます。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）「Origin」の値が、有効範囲外の場合。
  c）「Origin」と「Offset」で指定した位置が、ファイルサイズを超える場合。
  d）「FileID」で指定したファイルが、存在しない場合。
  e）「FileID」で指定したファイルが、アクセス中の場合。
  f）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.2-1255)」、FileWrite 命令の「サンプルプログラム (P.2-1262)」を、参照してください。

FileRead

SD メモリカード内の、指定したファイルのデータを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileRead	ファイル読出	FB		FileRead_instance(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	読み出すファイルのファイル ID	データ型に従う	−	0
Size	読出要素数		読み出す要素数			1
ReadBuf[] 配列	読出バッファ	入出力	読み出したデータの書込先	データ型に従う	−	−
ReadSize	実際に読み出した要素数	出力	実際に読み出した要素数	データ型に従う	−	−
EOF	ファイル終端		ファイル終端に達したかどうかの判定 TRUE: 達した FALSE: 達していない			

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																
Size							○													
ReadBuf[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
ReadSize							○													
EOF	○																			

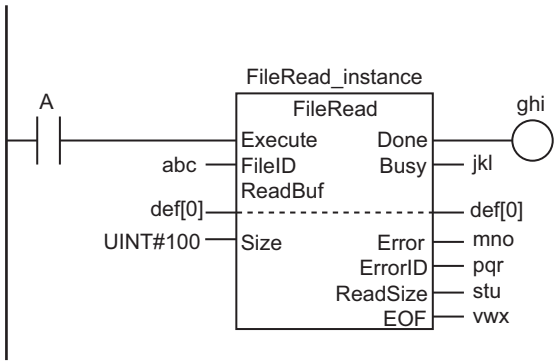
機能

SD メモリカード内のファイル ID「FileID」で指定したファイルの、ファイル位置指示子のある位置からデータを読み出し、読出バッファ ReadBuf[]に格納します。
ファイル位置指示子は、あらかじめ FileSeek 命令によって、任意の位置に設定しておきます。
読み出すデータの量は、（ReadBuf[]のデータ型のサイズ）×「Size」です。つまり、ReadBuf[]の「Size」個の要素分です。
ReadBuf[]は、列挙型を要素とする配列、構造体を要素とする配列でもかまいません。

実際に読み出した要素数が、「ReadSize」に格納されます。通常は、「Size」の値と「ReadSize」の値は一致します。もし、ファイル位置指示子のある位置からファイル終端までのデータ量が、「Size」より小さかった場合、異常は発生せず、ファイル終端までのデータが ReadBuf[]に格納されます。このとき、「ReadSize」の値は、「Size」の値より小さくなります。
また、ファイル終端まで読み出した場合、ファイル終端「EOF」の値が TRUE になります。そうでない場合は、「EOF」の値は FALSE になります。

記述例を、以下に示します。読出バッファ def[]が BYTE 型配列だとすると、ファイルから 100 バイトのデータを読み出します。

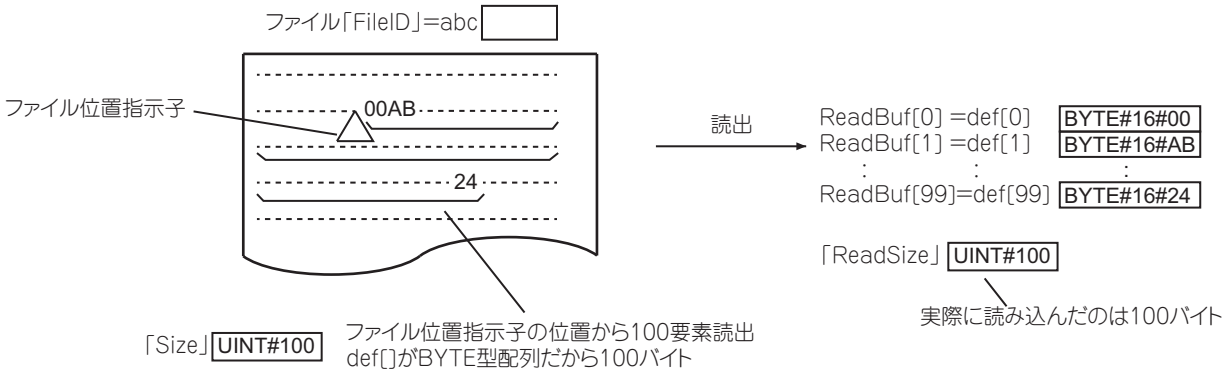
LD



ST

```
FileRead_instance(A, abc, def[0], UINT#100,
                  ghi, jkl, mno, pqr, stu, vwx);
```

SDメモリカード内の「FileID」で指定したファイルの、ファイル位置指示子のある位置から「Size」要素分だけ読み出し、読出バッファReadBuf[]に格納します。
その後、実際に読みだしたデータサイズが、「ReadSize」に出力されます。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし

変数名	名称	データ型	内容
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 命令内に読み出したデータ全体を保持しません。
  命令の実行タイミングとは同期せず、その都度 ReadBuf[]に渡した変数を更新するため、値の同時性は保証されません。
  命令実行中は対象の変数へアクセスしないようにしてください。
  命令実行中にアクセスした場合、ファイルから読み出したデータと異なる値が変数に格納される可能性があります。
- ファイルの終端までのデータを読み出し、かつ、そのサイズが ReadBuf[]のデータ型のサイズで割りきれない場合、ReadBuf[]のデータサイズに満たないデータは、破棄されます。ただし、ファイル位置指示子はファイル終端まで進み、「EOF」の値は TRUE になります。
- ReadBuf[]の「Size」番め以降の要素（読出により上書されない要素）は、本命令実行前の値が保持されます。
- 本命令を実行する前に、FileOpen 命令を実行して、「FileID」を取得しておく必要があります。
- 読み出されるデータは、バイト単位で、下位バイト→上位バイトの順（リトルエンディアン）に並んでいます。
- 「EOF」に値が格納されるのは、本命令が完了した時点です。つまり、「Done」の値が、FALSE から TRUE に変化した時点です。
- ReadBuf[]が構造体を要素とする配列の場合、その構成に応じて各メンバ間に調整用の領域が挿入されることがあります。
- 本命令実行中に、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- ReadBuf[]に、デバイス変数を指定することはできません。指定した場合は、読み出したデータが ReadBuf[]に代入されません。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）ReadBuf[]の配列要素数が、「Size」の値よりも小さい場合。
  c）「FileID」で指定したファイルが、存在しない場合。
  d）「FileID」で指定したファイルが、アクセス中の場合。
  e）「FileID」で指定したファイルが、読出可能なモードでオープンされていない場合。
  f）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

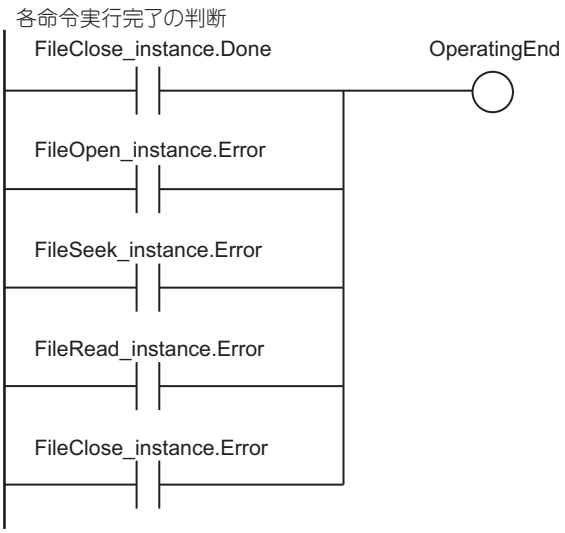
ファイル'ABC.bin'の、先頭から 2 バイトめの位置から、4 バイト分のデータを読み出して、BYTE 型配列変数 InDat[]に格納します。
処理手順は、以下のとおりです。

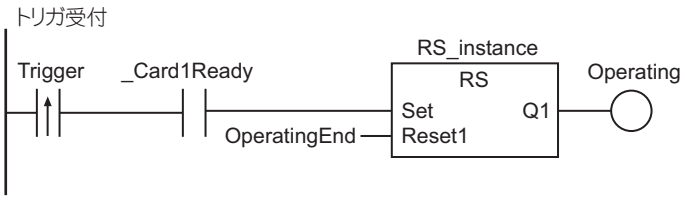
1. **FileOpen 命令を使って、ファイル'ABC.bin'をオープンします。**
2. **FileSeek 命令を使って、ファイル位置指示子を、ファイルの先頭から 2 バイトめに設定します。**
3. **FileRead 命令を使って、ファイル位置指示子の位置から 4 バイト分のデータを読み出し、配列変数 InDat[]に格納します。**
4. **FileClose 命令を使って、'ABC.bin'をクローズします。**

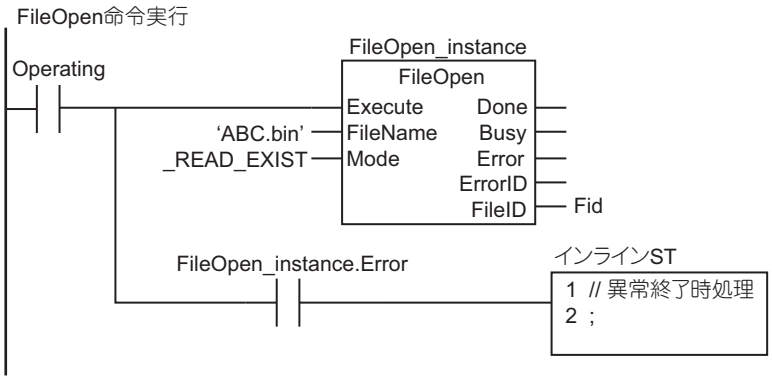
LD

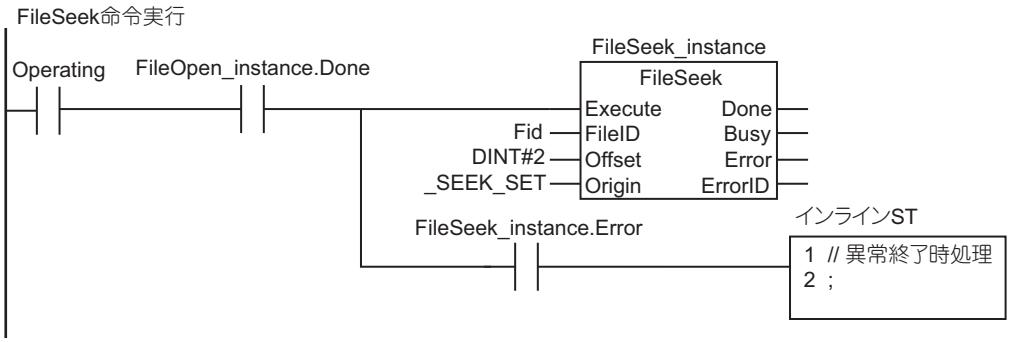
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Fid	DWORD	16#0	ファイル ID
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	読出データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

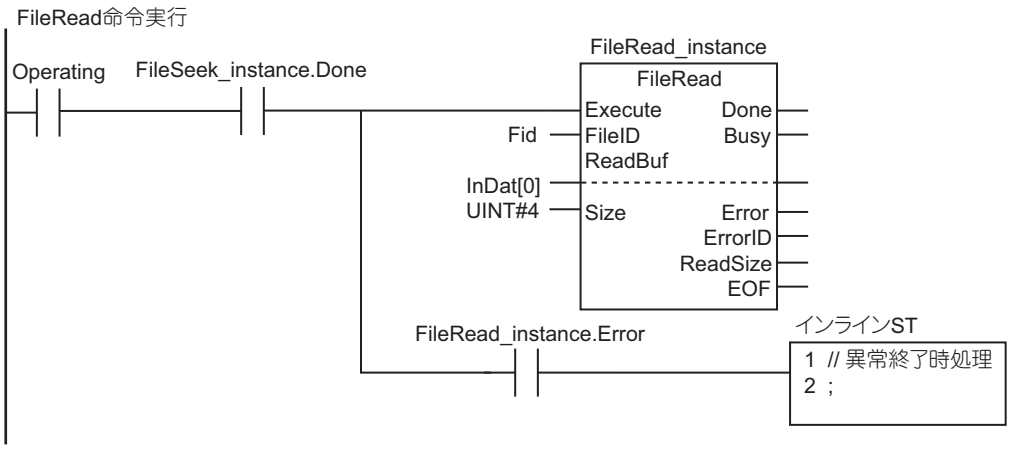
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

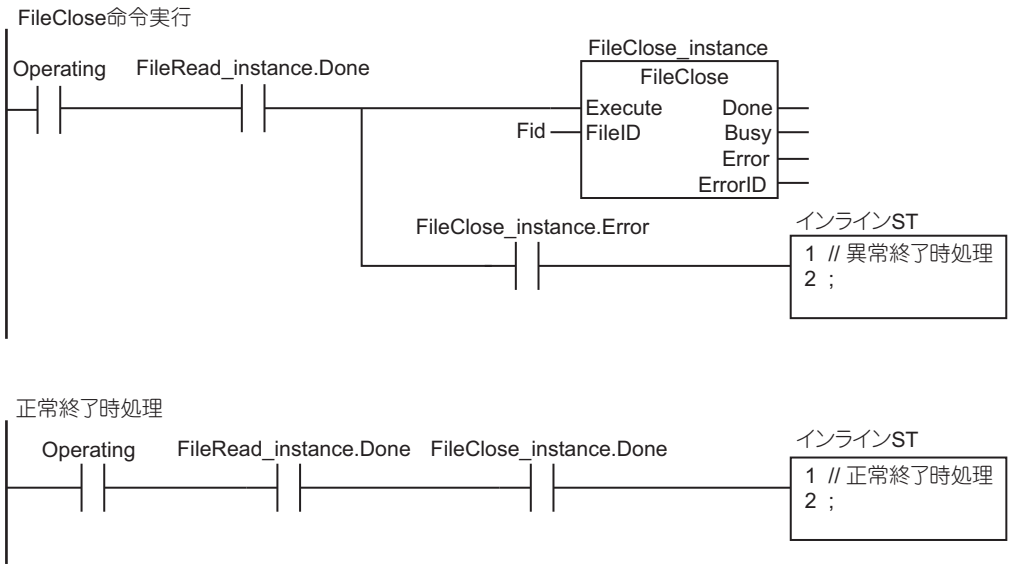



トリガ受付
Trigger
_Card1Ready
RS_instance
RS
Set
Reset1
Q1
OperatingEnd
Operating


FileOpen命令実行
Operating
FileOpen_instance
FileOpen
Execute
FileName
Mode
Done
Busy
Error
ErrorID
FileID
'ABC.bin'
_READ_EXIST
Fid
FileOpen_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileSeek命令実行
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
FileID
Offset
Origin
Done
Busy
Error
ErrorID
Fid
DINT#2
_SEEK_SET
FileSeek_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileRead命令実行
Operating
FileSeek_instance.Done
FileRead_instance
FileRead
Execute
FileID
ReadBuf
Size
Done
Busy
Error
ErrorID
ReadSize
EOF
Fid
InDat[0]
UINT#4
FileRead_instance.Error
インラインST
1 // 異常終了時処理
2 ;



ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	読出データ
	Stage	INT	0	状態遷移
	Fid	DWORD	16#0	ファイル ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);     // インスタンス初期化
    FileSeek_instance(Execute:=FALSE);     // インスタンス初期化
    FileRead_instance(
        Execute:=FALSE,                    // インスタンス初期化
```

```
        ReadBuf:=InDat[0]);                // ダミー
    FileClose_instance(Execute:=FALSE);    // インスタンス初期化
    Stage         :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// 各命令実行
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                          // ファイルオープン
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.bin',         // ファイル名
            Mode    :=_READ_EXIST,       // ファイル読出
            FileID  =>Fid);              // ファイル ID

        IF (FileOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;                // 正常終了
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;               // 異常終了
        END_IF;
    2 :                          // ファイルシーク
        FileSeek_instance(
            Execute:=TRUE,
            FileID :=Fid,                // ファイル ID
            Offset :=DINT#2,             // ファイル位置指示子 先頭から 2 バイトめ
            Origin :=_SEEK_SET);         //

        IF (FileSeek_instance.Done=TRUE) THEN
            Stage:=INT#3;                // 正常終了
        END_IF;

        IF (FileSeek_instance.Error=TRUE) THEN
            Stage:=INT#99;               // 異常終了
        END_IF;

    3 :                          // ファイル読出
        FileRead_instance(
            Execute:=TRUE,
            FileID :=Fid,                // ファイル ID
            ReadBuf:=InDat[0],           // 読出バッファ
            Size   :=UINT#4);            // 読出要素数 4 バイト

        IF (FileRead_instance.Done=TRUE) THEN
            Stage:=INT#4;            // 正常終了
        END_IF;

        IF (FileRead_instance.Error=TRUE) THEN
            Stage:=INT#99;           // 異常終了
```

```
        END_IF;

    4 :                         // ファイルクローズ
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);                  // ファイル ID

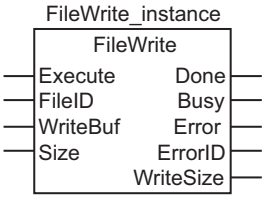
        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;           // 正常終了
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;              // 異常終了
        END_IF;

    99 :
        Operating:=FALSE;               // 異常終了処理
    END_CASE;
END_IF;
```

FileWrite

SD メモリカード内の、指定したファイルにデータを書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileWrite	ファイル書込	FB		FileWrite_instance(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	書き込むファイルのファイル ID	データ型に従う	―	0
WriteBuf[] 配列	書込バッファ		書き込むデータ			*1
Size	書込要素数		書き込む要素数			1
WriteSize	実際に書き込んだ要素数	出力	実際に書き込んだ要素数	データ型に従う	―	―

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

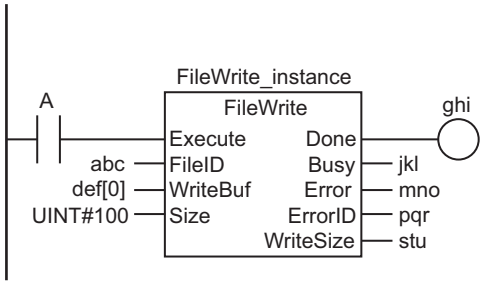
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																
WriteBuf[] 配列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型を要素とする配列、構造体を要素とする配列も指定可能																			
Size							○													
WriteSize							○													

機能

SD メモリカード内のファイル ID「FileID」で指定したファイルの、ファイル位置指示子のある位置に、データを書き込みます。
ファイル位置指示子は、あらかじめ FileSeek 命令によって、任意の位置に設定しておきます。
書き込むデータは、書込バッファ WriteBuf[]の中身です。
書き込むデータの量は、（WriteBuf[]のデータ型のサイズ）×「Size」です。つまり、WriteBuf[]の「Size」個の要素分です。
WriteBuf[]は、列挙型を要素とする配列、構造体を要素とする配列でもかまいません。
実際に書き込んだデータサイズが、「WriteSize」に出力されます。

記述例を、以下に示します。書込バッファ def[]が BYTE 型だとすると、ファイルに 100 バイトのデータを書き込みます。

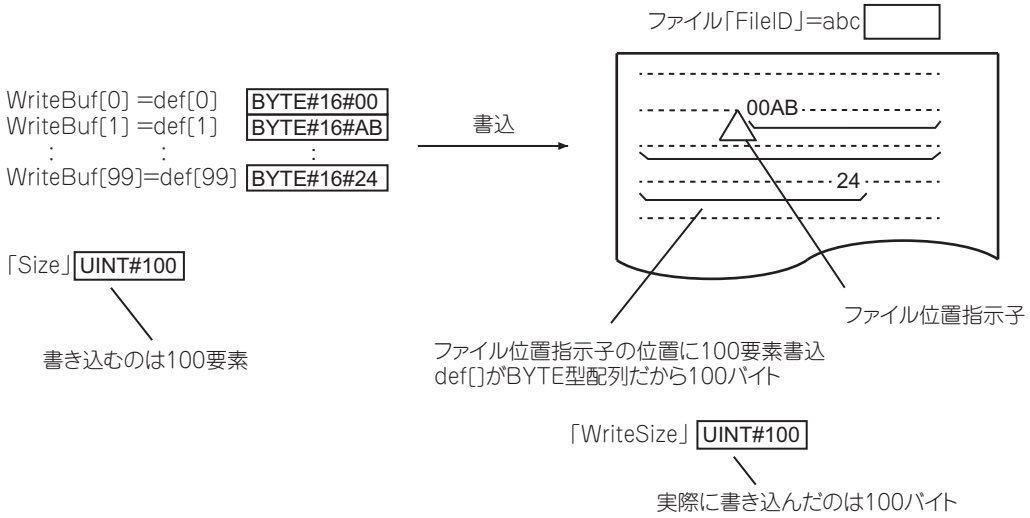
LD



ST

```
FileWrite_instance(A, abc, def[0], UINT#100, ghi,
                   jkl, mno, pqr, stu);
```

SDメモリカード内の「FileID」で指定したファイルの、ファイル位置指示子のある位置に、
書込バッファWriteBuf[]の中身を、「Size」要素分だけ書き込みます。
その後、実際に書き込んだデータサイズが、「WriteSize」に出力されます。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。

*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 命令内に書き込むデータ全体を保持しません。
  命令の実行タイミングとは同期せず、その都度 WriteBuf[]に渡した変数を参照するため、値の同時性は保証されません。
  命令実行中は対象の変数へアクセスしないようにしてください。
  命令実行中にアクセスした場合、意図しない値がファイルに書き込まれる可能性があります。
- 本命令を実行する前に、FileOpen 命令を実行して、「FileID」を取得しておく必要があります。
- 書き込まれるデータは、バイト単位で、下位バイト→上位バイトの順（リトルエンディアン）に並びます。
- WriteBuf[]が構造体を要素とする配列の場合、その構成に応じて各メンバ間に調整用の領域が挿入されることがあります。
- 本命令実行中に、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- FileWrite（ファイル書込）命令を使用して SD メモリカードにデータの書き込みを行い、正常終了しても、SD メモリカードにデータが正しく書き込まれていない場合があります。
  SD メモリカードに正しくデータが書き込まれているか確認する場合は、書き込んだデータを FileRead（ファイル読出）命令で読み出し、元のデータと照合するようにユーザプログラムを作成してください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）SD メモリカードが、空き容量不足の場合。
  d）WriteBuf[]の配列要素数が、「Size」の値よりも小さい場合。
  e）「FileID」で指定したファイルが、存在しない場合。
  f）「FileID」で指定したファイルが、アクセス中の場合。
  g）「FileID」で指定したファイルが、書込可能なモードでオープンされていない場合。
  h）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

ファイル名'ABC.bin'の、先頭から 2 バイトめの位置に、4 バイト分のデータを書き込みます。書き込むデータは、BYTE 型配列変数 OutDat[]の中身です。
処理手順は、以下のとおりです。

1 **FileOpen 命令を使って、ファイル'ABC.bin'をオープンします。**

2 **FileSeek 命令を使って、ファイル位置指示子を、ファイルの先頭から 2 バイトめに設定します。**

3 **FileWrite 命令を使って、ファイル位置指示子の位置に、配列変数 OutDat[]の値を 4 バイト分書き込みます。**

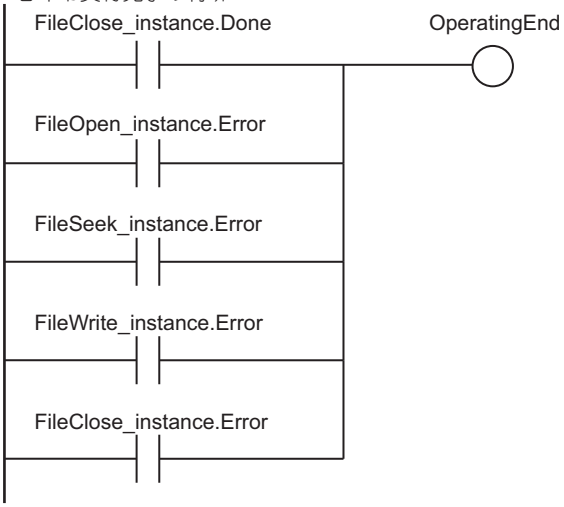
4 **FileClose 命令を使って、'ABC.bin'をクローズします。**

LD

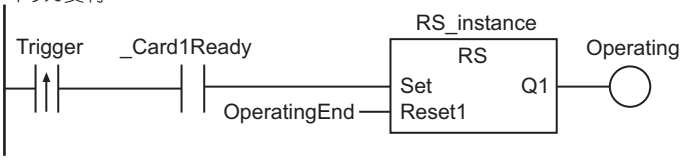
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Fid	DWORD	16#0	ファイル ID
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	書込データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

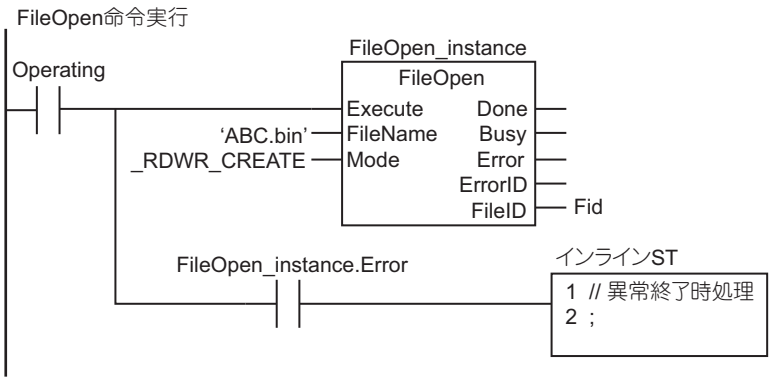
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

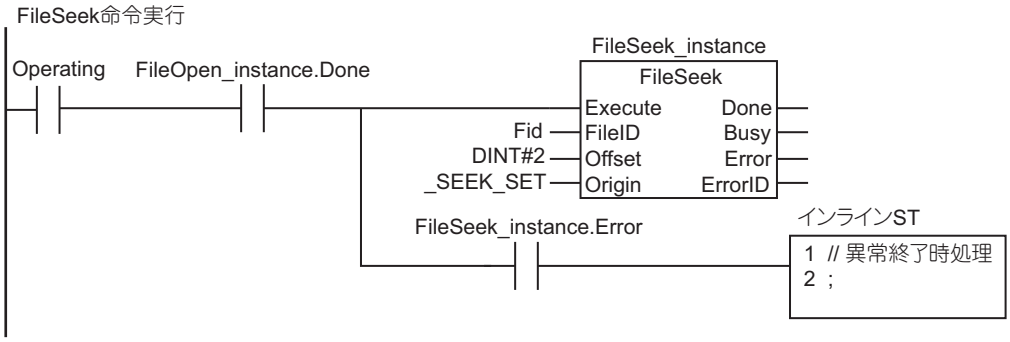
各命令実行完了の判断

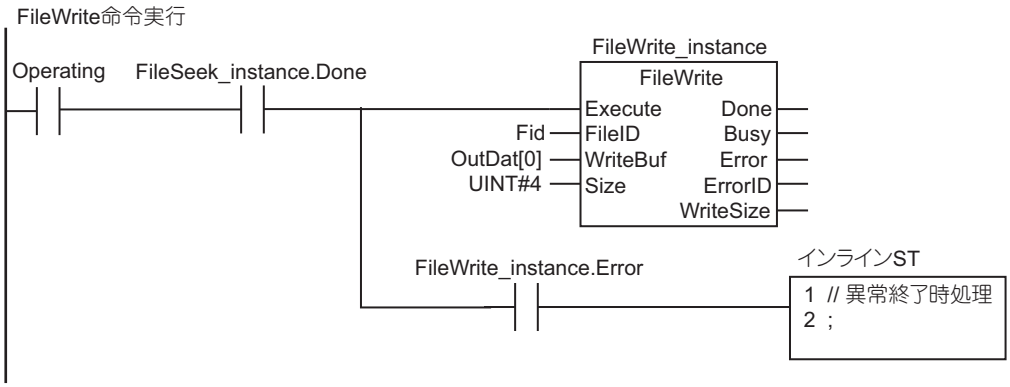


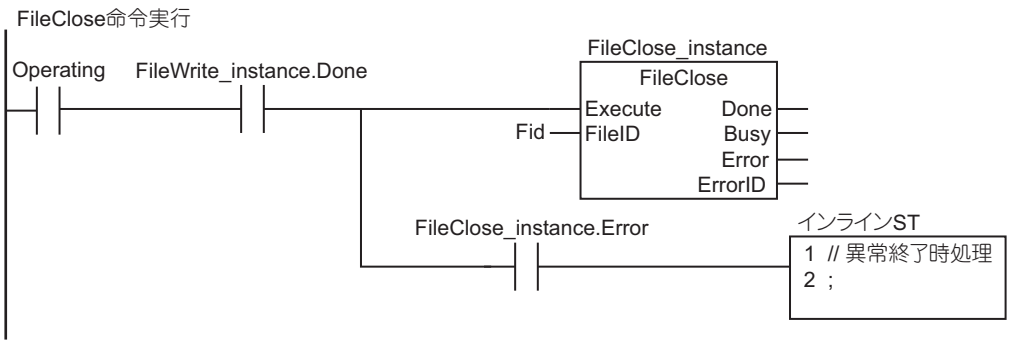
トリガ受付

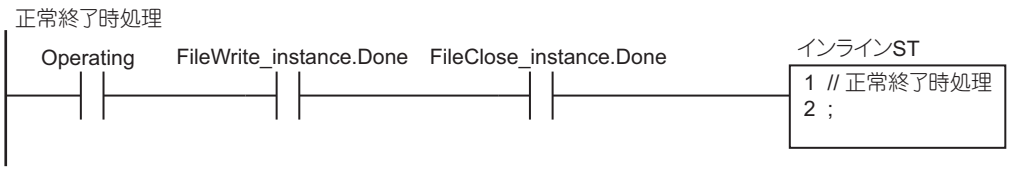



FileOpen命令実行
Operating
FileOpen_instance
FileOpen
Execute
FileName
Mode
Done
Busy
Error
ErrorID
FileID
'ABC.bin'
_RDWR_CREATE
Fid
FileOpen_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileSeek命令実行
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
FileID
Offset
Origin
Done
Busy
Error
ErrorID
Fid
DINT#2
_SEEK_SET
FileSeek_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileWrite命令実行
Operating
FileSeek_instance.Done
FileWrite_instance
FileWrite
Execute
FileID
WriteBuf
Size
Done
Busy
Error
ErrorID
WriteSize
Fid
OutDat[0]
UINT#4
FileWrite_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileClose命令実行
Operating
FileWrite_instance.Done
FileClose_instance
FileClose
Execute
FileID
Done
Busy
Error
ErrorID
Fid
FileClose_instance.Error
インラインST
1 // 異常終了時処理
2 ;


正常終了時処理
Operating
FileWrite_instance.Done
FileClose_instance.Done
インラインST
1 // 正常終了時処理
2 ;

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	書込データ
	Stage	INT	0	状態遷移
	Fid	DWORD	16#0	ファイル ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
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;

// 各命令実行
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                    // ファイルオープン
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.bin',        // ファイル名
            Mode    :=_RDWR_CREATE,     // ファイル読出および書込
            FileID  =>Fid);             // ファイル ID

        IF (FileOpen_instance.Done=TRUE) THEN
```

```
        Stage:=INT#2;          // 正常終了
    END_IF;

    IF (FileOpen_instance.Error=TRUE) THEN
        Stage:=INT#99;         // 異常終了
    END_IF;
2 :                    // ファイルシーク
    FileSeek_instance(
        Execute:=TRUE,
        FileID :=Fid,              // ファイル ID
        Offset :=DINT#2,           // ファイル位置指示子 先頭から 2 バイトめ
        Origin :=_SEEK_SET);       //

    IF (FileSeek_instance.Done=TRUE) THEN
        Stage:=INT#3;          // 正常終了
    END_IF;

    IF (FileSeek_instance.Error=TRUE) THEN
        Stage:=INT#99;         // 異常終了
    END_IF;

3 :                    // ファイル書込
    FileWrite_instance(
        Execute :=TRUE,
        FileID  :=Fid,             // ファイル ID
        WriteBuf:=OutDat[0],       // 書込バッファ
        Size    :=UINT#4);         // 書込要素数 4 バイト

    IF (FileWrite_instance.Done=TRUE) THEN
        Stage:=INT#4;          // 正常終了
    END_IF;

    IF (FileWrite_instance.Error=TRUE) THEN
        Stage:=INT#99;         // 異常終了
    END_IF;

4 :                    // ファイルクローズ
    FileClose_instance(
        Execute:=TRUE,
        FileID :=Fid);             // ファイル ID

    IF (FileClose_instance.Done=TRUE) THEN
        Operating:=FALSE;      // 正常終了
    END_IF;

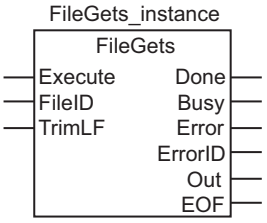
    IF (FileClose_instance.Error=TRUE) THEN
        Stage:=INT#99;         // 異常終了
    END_IF;

99 :
    Operating:=FALSE;          // 異常終了処理
```

```
    END_CASE;
END_IF;
```

FileGets

SD メモリカード内の指定したファイルから、1 行分の文字列を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileGets	文字列読出	FB		FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	読み出すファイルのファイル ID	データ型に従う	—	0
TrimLF	改行コード削除指定		読み出す文字列の改行コードの削除指定 TRUE: 削除する FALSE: 削除しない			FALSE
Out	読出文字列	出力	読み出した文字列	データ型に従う	—	—
EOF	ファイル終端		ファイル終端に達したかどうかの判定 TRUE: 達した FALSE: 達していない			

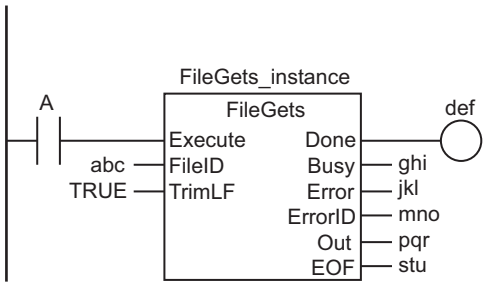
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																
TrimLF	○																			
Out																				○
EOF	○																			

機能

SD メモリカード内のファイル ID「FileID」で指定したファイルの、ファイル位置指示子のある位置から 1 行分の文字列を読み出します。
ファイル位置指示子は、あらかじめ FileSeek 命令によって、任意の位置に設定しておきます。
行と行の区切りは、改行コードで識別します。
読み出した文字列は、読出文字列「Out」に書き込みます。
改行コードは CR、LF、CR+LF の 3 種類の中から自動的に判別します。
改行コード削除指定「TrimLF」の値が TRUE の場合、文字列中の改行コードを削除したうえで「Out」に書き込みます。
また、ファイル終端まで読み出した場合、ファイル終端「EOF」の値が TRUE になります。そうでない場合、「EOF」の値は FALSE になります。

記述例を、以下に示します。ファイルから、1 行分の文字列を読み出し、改行コードを削除して、「pqr」に書き込みます。

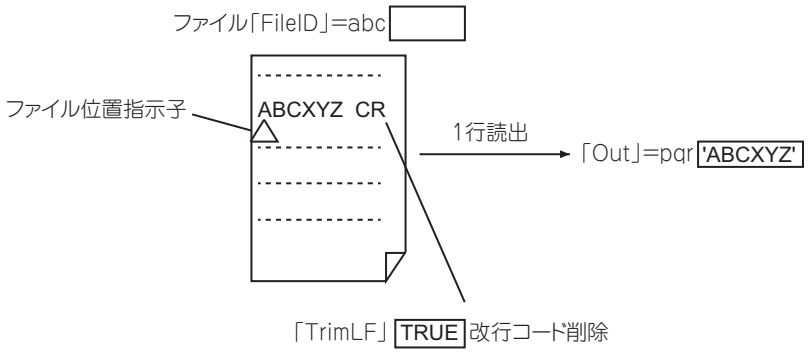
LD



ST

```
FileGets_instance(A, abc, TRUE, def, ghi,
                  jkl, mno, pqr, stu);
```

SDメモリカード内の「FileID」で指定したファイルの、ファイル位置指示子のある位置から
1行文の文字列を読み出し、読出文字列「Out」に格納します。その際、改行コードは削除します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

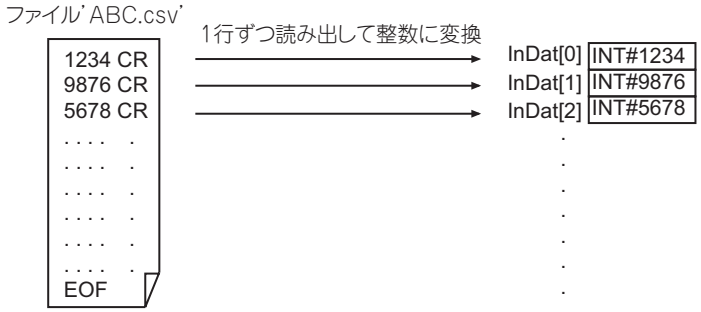
*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 1 行の文字列の長さが 1986 バイト（文字コードは UTF-8 で、終端の NULL 文字を含みます）を超える場合、1985 バイトめまでの文字列に終端の NULL 文字を付けて、「Out」に格納します。
- 本命令を実行する前に、FileOpen 命令を実行して、「FileID」を取得しておく必要があります。
- 本命令実行中に、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）「FileID」で指定したファイルが、存在しない場合。
  c）「FileID」で指定したファイルが、アクセス中の場合。
  d）「FileID」で指定したファイルが、読出可能なモードでオープンされていない場合。
  e）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

ファイル名'ABC.csv'のファイルには、改行コード CR で区切られた文字列が、複数行格納されているとします。それらはすべて、数字の文字列です。
このファイルを 1 行ずつ読み出し、文字列を整数に変換して、INT 型配列変数 InDat[]に格納します。
ファイルを最後まで読んだら（ファイル終端 EOF まで読み出したら）、処理を終了します。



処理手順は、以下のとおりです。

1 **FileOpen 命令を使って、ファイル'ABC.csv'をオープンします。**

2 **FileGets 命令を使って、ファイルの内容を 1 行読み出します。**

3 **STRING_TO_INT 命令を使って、読み出した文字列を整数に変換し、InDat[]に格納します。**

4 **上記 2～3 を、ファイル終端 EOF を読み出すまで続けます。**

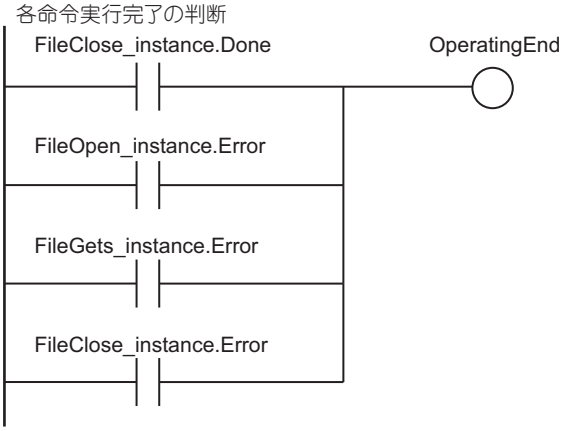
5 **FileClose 命令を使って、ファイルをクローズします。**

LD

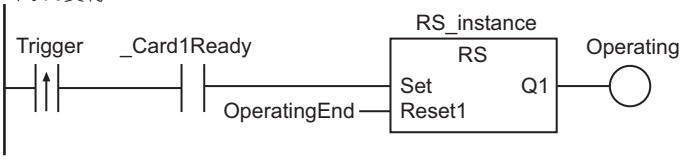
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Index	INT	0	InDat[]の要素インデックス
	Fid	DWORD	16#0	ファイル ID
	InDat	ARRAY[0..999] OF INT	[1000(0)]	整数データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

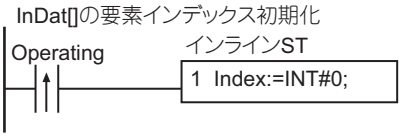
各命令実行完了の判断

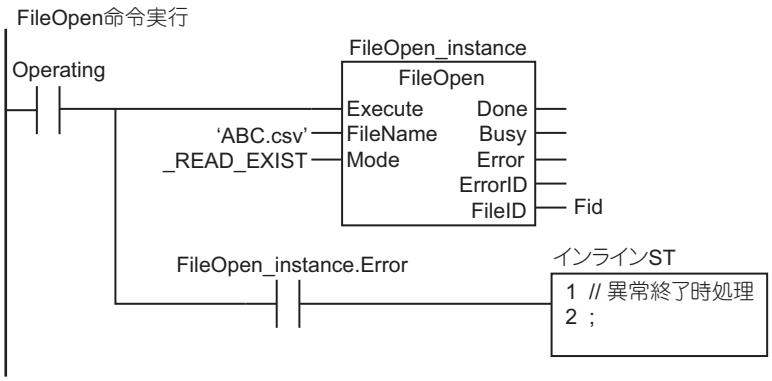


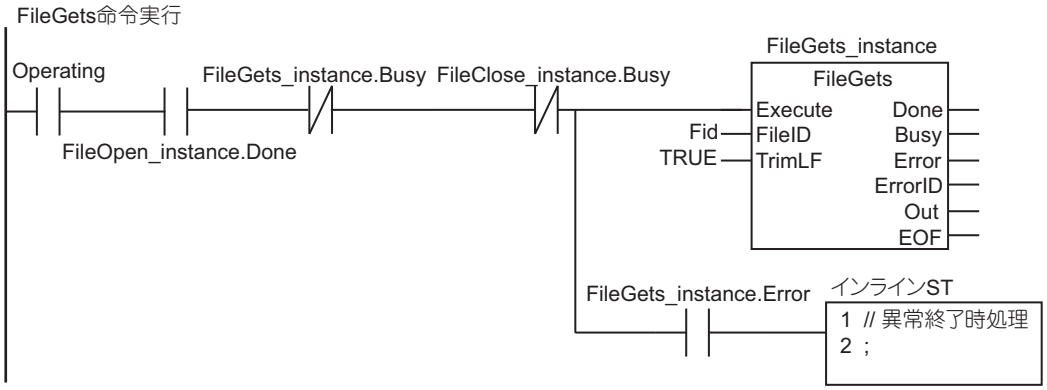
トリガ受付

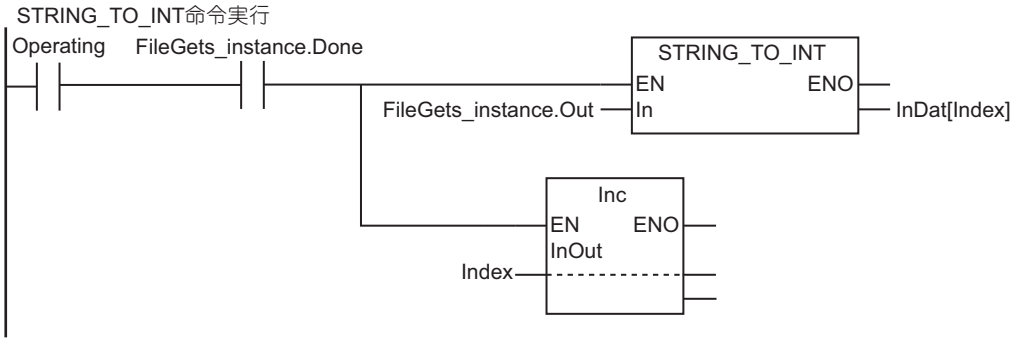


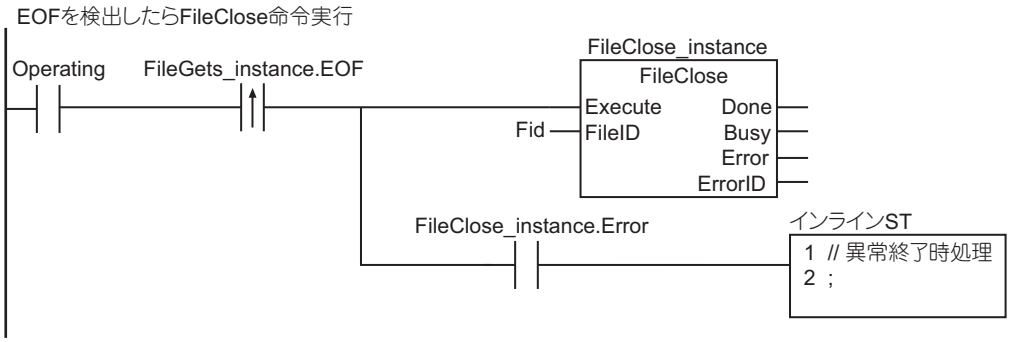
InDat[]の要素インデックス初期化




FileOpen命令実行
Operating
FileOpen_instance
FileOpen
Execute
FileName
Mode
Done
Busy
Error
ErrorID
FileID
'ABC.csv'
_READ_EXIST
Fid
FileOpen_instance.Error
インラインST
1 // 異常終了時処理
2 ;


FileGets命令実行
Operating
FileGets_instance.Busy
FileClose_instance.Busy
FileOpen_instance.Done
FileGets_instance
FileGets
Execute
FileID
TrimLF
Done
Busy
Error
ErrorID
Out
EOF
Fid
TRUE
FileGets_instance.Error
インラインST
1 // 異常終了時処理
2 ;


STRING_TO_INT命令実行
Operating
FileGets_instance.Done
STRING_TO_INT
EN
In
ENO
FileGets_instance.Out
InDat[Index]
Inc
EN
InOut
ENO
Index


EOFを検出したらFileClose命令実行
Operating
FileGets_instance.EOF
FileClose_instance
FileClose
Execute
FileID
Done
Busy
Error
ErrorID
Fid
FileClose_instance.Error
インラインST
1 // 異常終了時処理
2 ;


正常終了時処理
Operating
FileClose_instance.Done
インラインST
1 // 正常終了時処理
2 ;

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	InDat	ARRAY[0..999] OF INT	[1000(0)]	整数データ
	Stage	INT	0	状態遷移
	Index	INT	0	InDat[]の要素インデックス
	Fid	DWORD	16#0	ファイル ID
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
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;

// 各命令実行
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                  // ファイルオープン
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.csv',      // ファイル名
            Mode    :=_READ_EXIST,    // ファイル読出
            FileID  =>Fid);           // ファイル ID

        IF (FileOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;        // 正常終了
        END_IF;
```

```
        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;      // 異常終了
        END_IF;
    2 :                 // 文字列読出
        FileGets_instance(
            Execute:=TRUE,
            FileID :=Fid,
            TrimLF :=TRUE);

        IF (FileGets_instance.Done=TRUE) THEN
            // 読み出した文字列を整数に変換
            InDat[Index]:=STRING_TO_INT(FileGets_instance.Out);
            Index:=Index+INT#1;

            // ファイル終端に達した
            IF (FileGets_instance.EOF=TRUE) THEN
                Stage:=INT#3;   // 正常終了
            ELSE
                FileGets_instance(Execute:=FALSE);
            END_IF;
        END_IF;

        IF (FileGets_instance.Error=TRUE) THEN
            Stage:=INT#99;      // 異常終了
        END_IF;

    3 :                 // ファイルクローズ
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);          // ファイル ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;   // 正常終了
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;      // 異常終了
        END_IF;

     99 :                // 異常終了処理
         Operating:=FALSE;
    END_CASE;
END_IF;
```

FilePuts

SD メモリカード内の指定したファイルに、文字列を書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FilePuts	文字列書込	FB	FilePuts_instance FilePuts Execute / FileID / In Done / Busy / Error / ErrorID	FilePuts_instance(Execute, FileID, In, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileID	ファイル ID	入力	書き込むファイルのファイル ID	データ型に従う	—	0
In	書込文字列		書き込む文字列			"

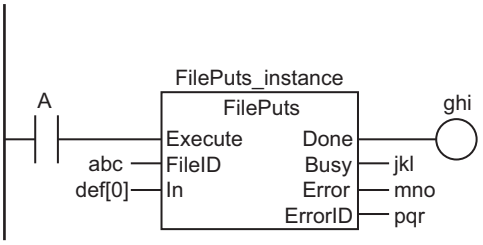
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				○																
In																				○

機能

SD メモリカード内のファイル ID「FileID」で指定したファイルの、ファイル位置指示子のある位置に文字列を書き込みます。
ファイル位置指示子は、あらかじめ FileSeek 命令によって、任意の位置に設定しておきます。
書き込む文字列は、書込文字列「In」の中身です。

記述例を、以下に示します。配列の 1 要素 def[0]の内容を、ファイルに書き込みます。

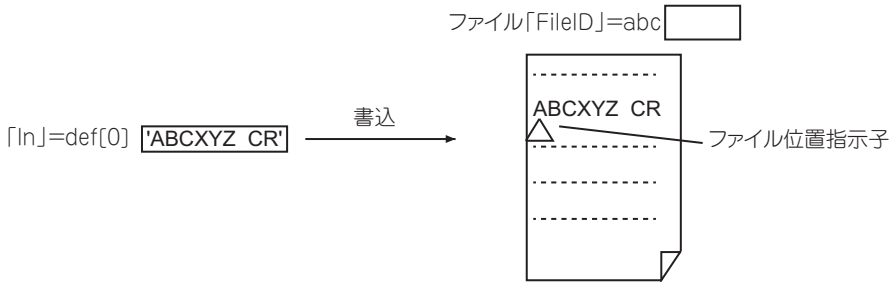
LD



ST

```
FilePuts_instance(A, abc, def[0], ghi, jkl, mno, pqr);
```

SDメモリカード内の「FileID」で指定したファイルの、ファイル位置指示子のある位置に、
書込文字列「In」の中身を書き込みます。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

文字列を書き込んだ後に改行したい場合は、「In」の最後に改行コードを付加してください。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 本命令を実行する前に、FileOpen 命令を実行して、「FileID」を取得しておく必要があります。

- 本命令実行中に、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- FilePuts（文字列書込）命令を使用して SD メモリカードにデータの書き込みを行い、正常終了しても、SD メモリカードにデータが正しく書き込まれていない場合があります。
  SD メモリカードに正しくデータが書き込まれているか確認する場合は、書き込んだデータを FileGets（文字列読出）命令で読み出し、元のデータと照合するようにユーザプログラムを作成してください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）SD メモリカードが、空き容量不足の場合。
  d）「FileID」で指定したファイルが、存在しない場合。
  e）「FileID」で指定したファイルが、アクセス中の場合。
  f）「FileID」で指定したファイルが、書込可能なモードでオープンされていない場合。
  g）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

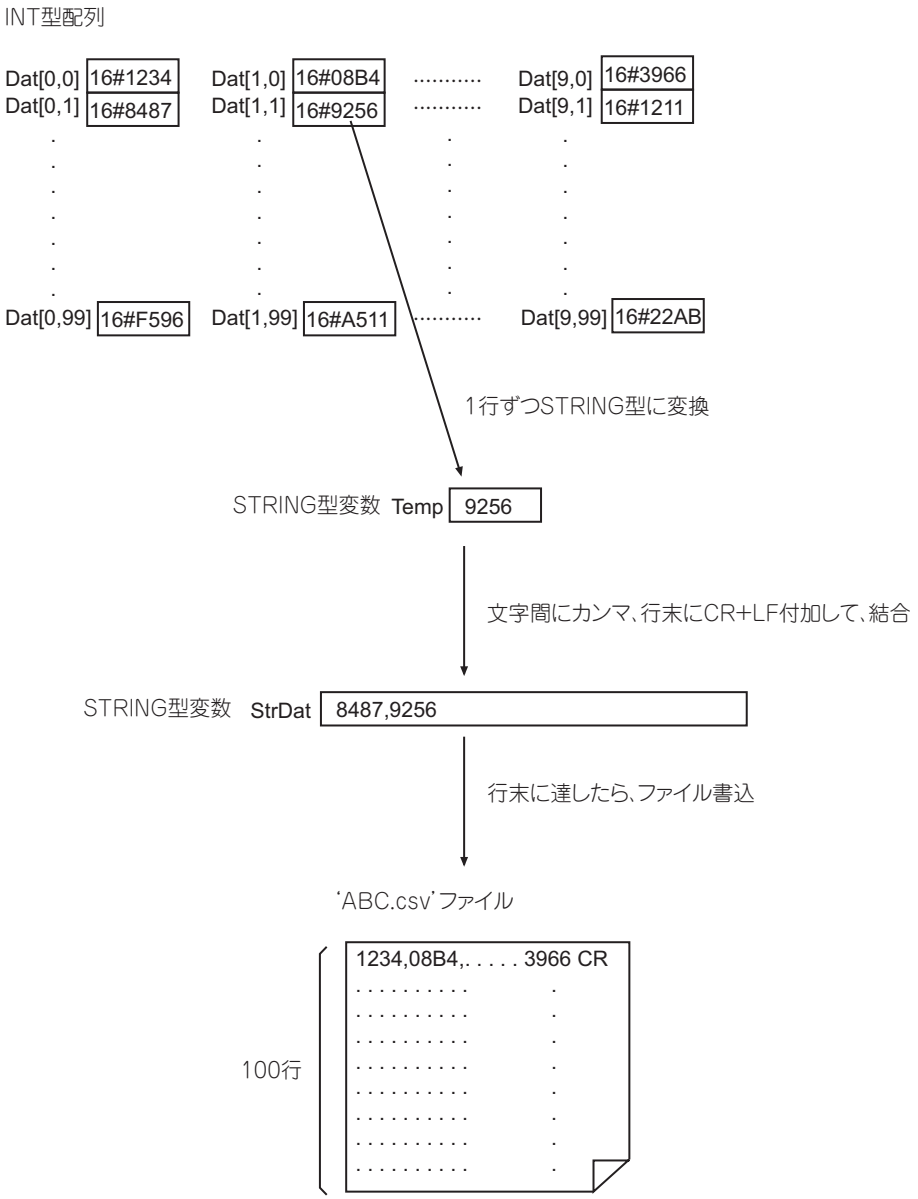
INT 型の配列変数 Dat[0..9,0..99]の内容を、100 行の CSV ファイル形式で、ファイル'ABC.csv'に格納します。
1 行には、数値の文字列が 10 個並び、それらの間にはカンマを挿入します。行末には、改行コード CR + LF を付加します。
手順は以下のとおりです。

1 **Dat[]の要素を 1 個文字列変換し、STRING 型の変数「Temp」に格納します。**

2 **「Temp」が行末でなければ後ろにカンマを付加し、STRING 型変数「StrDat」に結合します。**
「Temp」が行末ならば後ろに CR + LF を付加し、STRING 型変数「StrDat」に結合します。

3 **行末に達したら、「StrDat」を、ファイルに書き込みます。**

4 **1 から 3 を、100 行分繰り返します。**

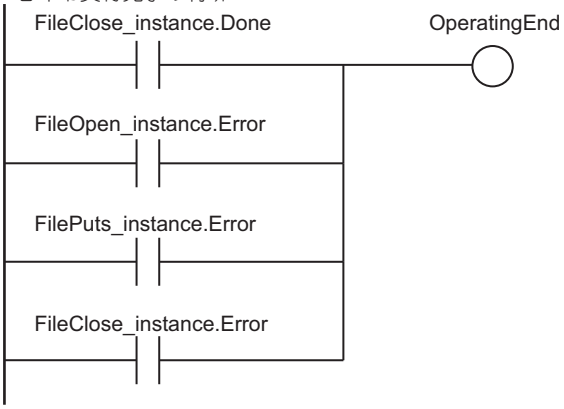


LD

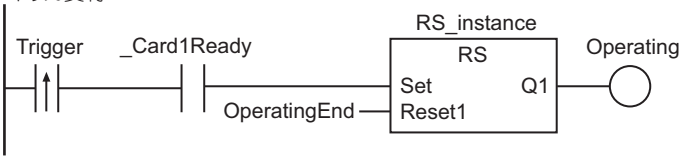
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Index0	INT	0	列インデックス
	Index1	INT	0	行インデックス
	Fid	DWORD	16#0	ファイル ID
	StrDat	STRING[255]	''	文字列データ
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	数値データ
	Temp	STRING[255]	''	テンポラリ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

各命令実行完了の判断



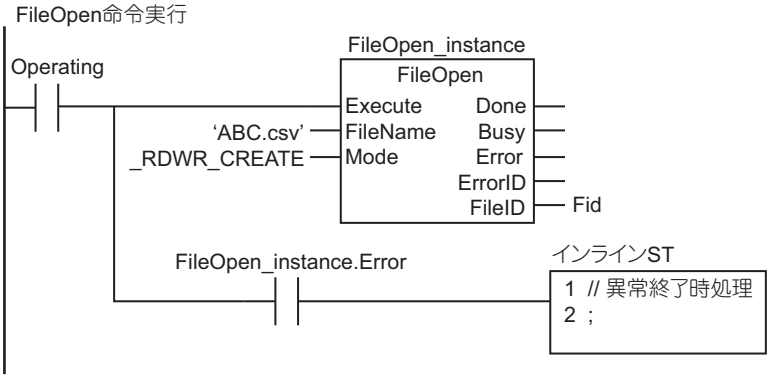
トリガ受付



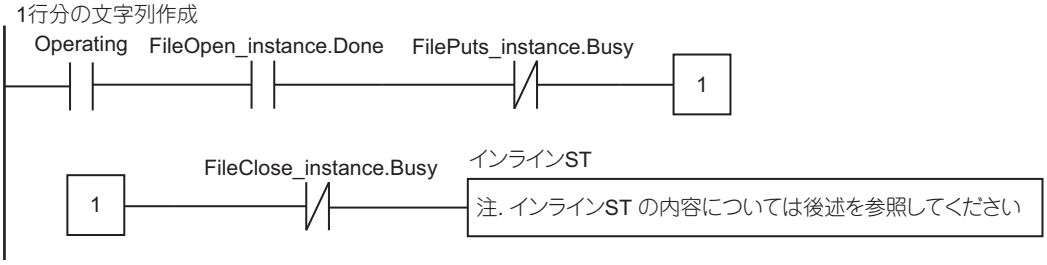
行インデックス初期化

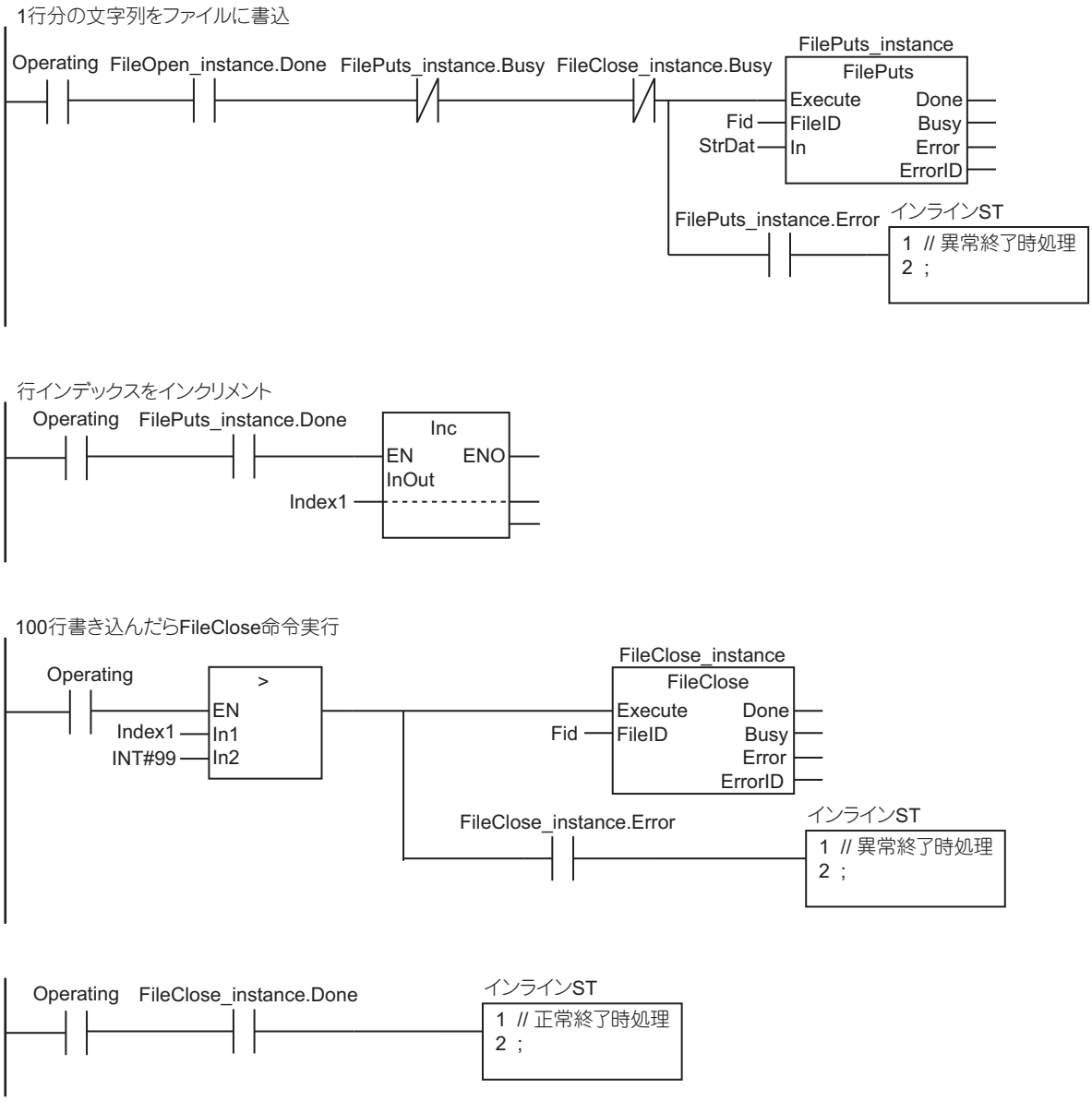


FileOpen命令実行



1行分の文字列作成





● インライン ST の内容

```
StrDat:='';

// 0～8 番めの文字列を連結

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;

// 9 番め文字列を連結し、CR+LF を付加
Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
Temp  :=CONCAT(In1:=Temp, In2:='$r$l');
StrDat:=CONCAT(In1:=StrDat, In2:=Temp);
```

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の Trigger の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	Index0	INT	0	列インデックス
	Index1	INT	0	行インデックス
	Fid	DWORD	16#0	ファイル ID
	StrDat	STRING[255]	''	文字列データ
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	数値データ
	Temp	STRING[255]	''	テンポラリ
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FilePuts_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage         :=INT#1;
    Index1        :=INT#0;       // 行インデックス初期化
    OperatingStart:=FALSE;
END_IF;

// 各命令実行
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                   // ファイルオープン
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.csv',      // ファイル名
            Mode    :=_RDWR_CREATE,   // ファイル読出
            FileID  =>Fid);           // ファイル ID

        IF (FileOpen_instance.Done=TRUE) THEN
```

```
        Stage:=INT#2;         // 正常終了
    END_IF;

    IF (FileOpen_instance.Error=TRUE) THEN
        Stage:=INT#99;        // 異常終了
    END_IF;
2 :                   // 1 行分の文字列作成
    StrDat:='';

    // 0～8 番めの文字列を連結
    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;

    // 9 番めの文字列を連結し、CR+LF を付加
    Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
    Temp  :=CONCAT(In1:=Temp, In2:='$r$l');
    StrDat:=CONCAT(In1:=StrDat, In2:=Temp);

    Stage:=INT#3;

3 :                   // 文字列書込
    FilePuts_instance(
        Execute:=TRUE,
        FileID :=Fid,
        In     :=StrDat);

    IF (FilePuts_instance.Done=TRUE) THEN
        Index1:=Index1+INT#1;

        IF (Index1>INT#99) THEN   // 100 行書き込んだ
            Stage:=INT#4;
        ELSE
            FilePuts_instance(Execute:=FALSE);
            Stage:=INT#2;
        END_IF;
    END_IF;

    IF (FilePuts_instance.Error=TRUE) THEN
        Stage:=INT#99;        // 異常終了
    END_IF;

4 :                   // ファイルクローズ
    FileClose_instance(
        Execute:=TRUE,
        FileID :=Fid);            // ファイル ID

    IF (FileClose_instance.Done=TRUE) THEN
        Operating:=FALSE;     // 正常終了
```

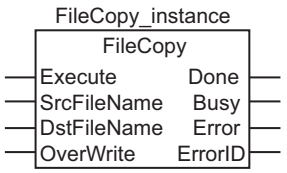
```
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;       // 異常終了
        END_IF;

    99 :                    // 異常終了処理
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileCopy

SD メモリカード内の指定したファイルを、コピーします。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileCopy	ファイルコピー	FB		FileCopy_instance(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
SrcFileName	コピー元ファイル名	入力	コピー元ファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	−	"
DstFileName	コピー先ファイル名		コピー先ファイル名			
OverWrite	上書許可		TRUE: 上書許可 FALSE: 上書禁止	データ型に従う		FALSE

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcFileName																				○
DstFileName																				○
OverWrite	○																			

機能

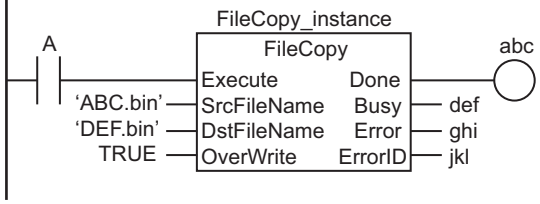
SD メモリカード内の、コピー元ファイル名「SrcFileName」で指定したファイルを、コピー先ファイル名「DstFileName」にコピーします。

SD メモリカード内に、「DstFileName」と同じ名前のファイルがすでに存在している場合、上書許可「OverWrite」の値によって、以下のような処理を行います。

「OverWrite」の値	処理
TRUE（上書許可）	そのファイルに、上書します。
FALSE（上書禁止）	そのファイルに上書せず、異常になります。

記述例を、以下に示します。ファイル'ABC.bin'を、ファイル'DEF.bin'に上書します。

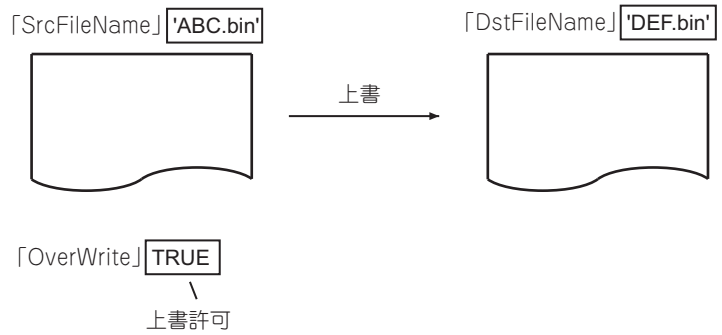
LD



ST

```
FileCopy_instance(A, 'ABC.bin', 'DEF.bin',
                  TRUE, abc, def, ghi, jkl);
```

SDメモリカード内の、コピー元ファイル名「SrcFileName」で指定したファイルを、
コピー先ファイル名「DstFileName」に上書します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- コピーに失敗した場合、SD メモリカード内に「DstFileName」で指定したファイルが、不完全な状態で残ることがあります。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) SD メモリカードが、使用可能状態ではない場合。
  b) SD メモリカードが、ライトプロテクト中の場合。
  c) SD メモリカードが、空き容量不足の場合。
  d)「SrcFileName」で指定したファイルが、存在しない場合。
  e)「SrcFileName」の値が、ファイル名として不正の場合。
  f)「SrcFileName」または「DstFileName」で指定したファイルが、アクセス中の場合。
  g)「DstFileName」の値が、ファイル名として不正の場合。
  h)「DstFileName」と同じ名前のファイルがすでに存在していて、かつ、「OverWrite」の値が FALSE の場合。
  i)「DstFileName」と同じ名前のファイルがすでに存在していて、かつ、そのファイルが書込禁止の場合。
  j)「DstFileName」の値が、ファイル名として使用できるバイト数を超えている場合。
  k) 作成可能なファイル数、ディレクトリ数を超える場合。
  l) 変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
  m) SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

以下の手順で、ファイルの移動を行います。

1 **DirCreate 命令を使って、SD メモリカード内に、新規のディレクトリ'Dir1'を作成します。**

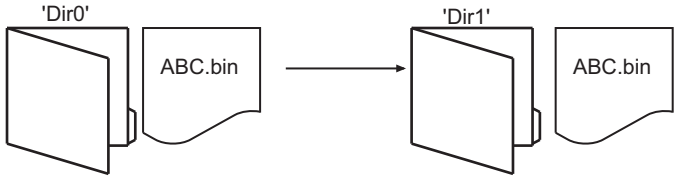
2 **FileCopy 命令を使って、既存のディレクトリ'Dir0'内のファイル'ABC.bin'を、ディレクトリ'Dir1'にコピーします。**

3 **DirRemove 命令を使って、コピー元ディレクトリ'Dir0'を削除します。**

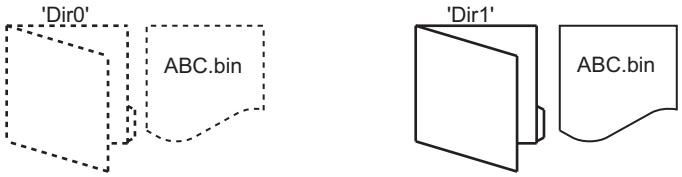
1.ディレクトリ作成



2.ファイルコピー



3.ディレクトリ削除

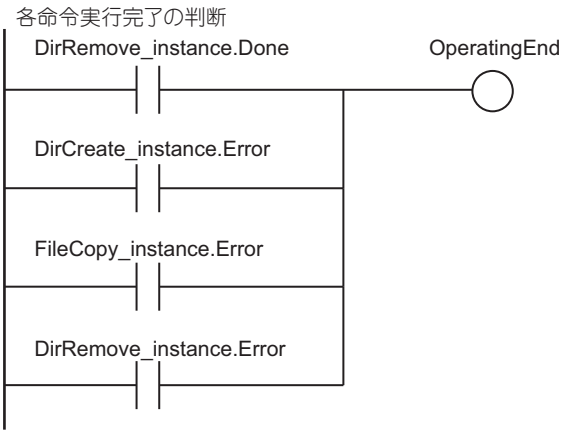


LD

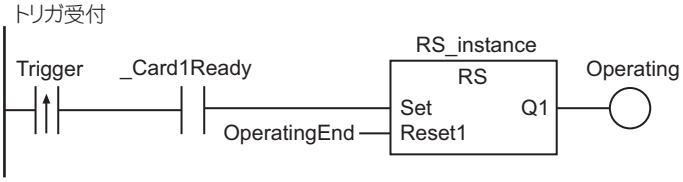
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	DirCreate_instance	DirCreate		
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

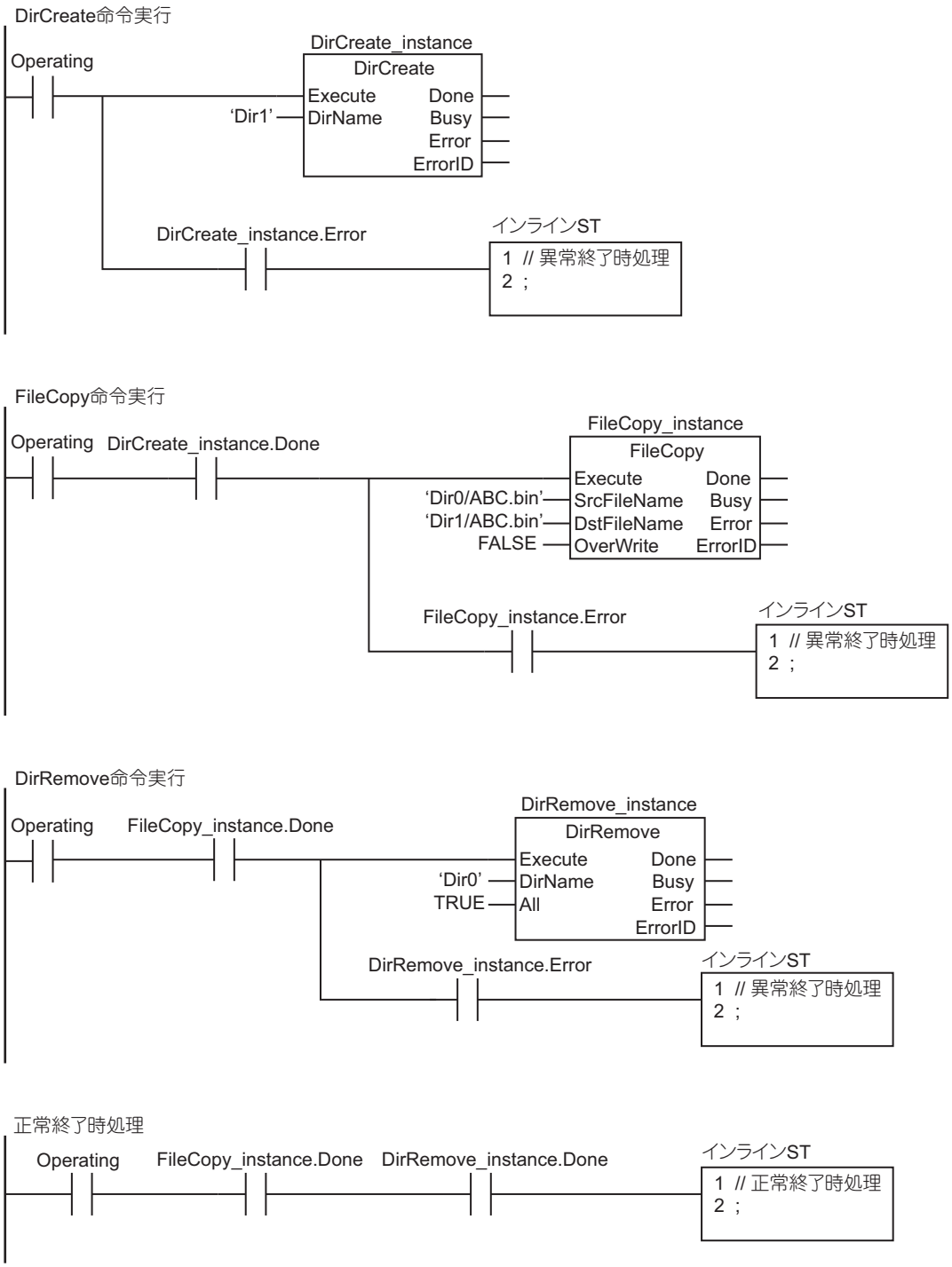
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

各命令実行完了の判断



トリガ受付





ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	Stage	INT	0	状態遷移
	DirCreate_instance	DirCreate		

内部変数	名称	データ型	初期値	コメント
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
    DirCreate_instance(Execute:=FALSE);
    FileCopy_instance(Execute:=FALSE);
    DirRemove_instance(Execute:=FALSE);
    Stage         :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// 各命令実行
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                             // ディレクトリ作成
        DirCreate_instance(
            Execute:=TRUE,
            DirName:='Dir1');               // ディレクトリ名

        IF (DirCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;                   // 正常終了
        END_IF;

        IF (DirCreate_instance.Error=TRUE) THEN
            Stage:=INT#99;                  // 異常終了
        END_IF;
    2 :                             // ファイルコピー
        FileCopy_instance(
            Execute    :=TRUE,
            SrcFileName:='Dir0/ABC.bin',    // コピー元ファイル名
            DstFileName:='Dir1/ABC.bin',    // コピー先ファイル名
            OverWrite  :=FALSE);            // 上書禁止

        IF (FileCopy_instance.Done=TRUE) THEN
            Stage:=INT#3;
        END_IF;

        IF (FileCopy_instance.Error=TRUE) THEN
```

```
            Stage:=INT#99;
        END_IF;

    3 :                             // ディレクトリ削除
        DirRemove_instance(
            Execute:=TRUE,
            DirName:='Dir0',                // ディレクトリ名
            All    :=TRUE);                 // ファイルやサブディレクトリごと削除

        IF (DirRemove_instance.Done=TRUE) THEN
            Operating:=FALSE;               // 正常終了
        END_IF;

        IF (DirRemove_instance.Error=TRUE) THEN
            Stage:=INT#99;                  // 異常終了
        END_IF;

    99 :                            // 異常終了処理
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileRemove

SD メモリカード内の指定したファイルを、削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileRemove	ファイル削除	FB	FileRemove_instance FileRemove Execute / FileName Done / Busy / Error / ErrorID	FileRemove_instance(Execute, FileName, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	指定ファイル名	入力	削除するファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	–	''

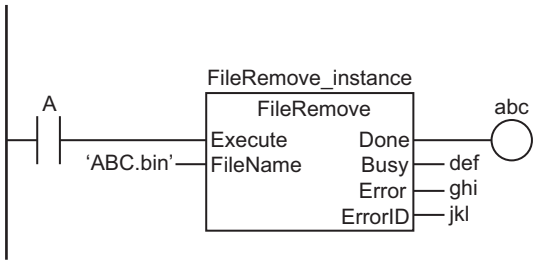
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○

機能

SD メモリカード内の、指定ファイル名「FileName」で指定したファイルを削除します。

記述例を、以下に示します。ファイル'ABC.bin'を削除します。

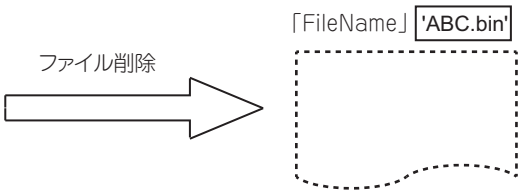
LD



ST

```
FileRemove_instance(A, 'ABC.bin', abc,
                    def, ghi, jkl);
```

SDメモリカード内の、「FileName」で指定したファイルを削除します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）「FileName」で指定したファイルが、存在しない場合。
  d）「FileName」で指定したファイルが、アクセス中の場合。
  e）「FileName」と同じ名前のファイルがすでに存在していて、かつ、そのファイルが書込禁止の場合。
  f）「FileName」の値が、ファイル名として使用できるバイト数を超えている場合。

g）変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
h）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

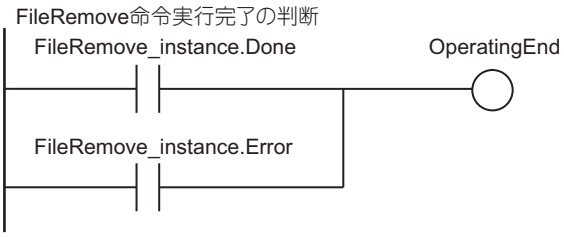
サンプルプログラム

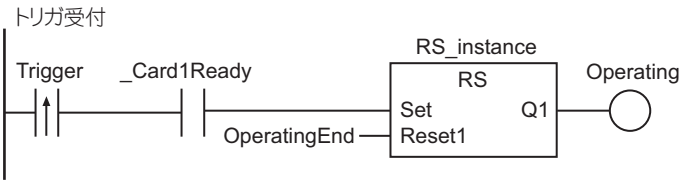
SD メモリカード内の、ファイル'ABC.bin'を削除します。

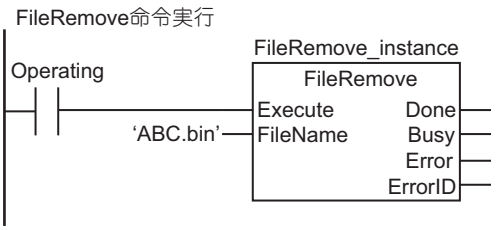
LD

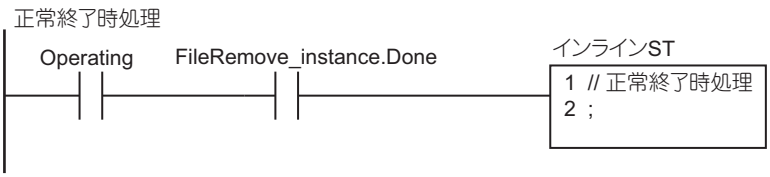
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	FileRemove_instance	FileRemove		

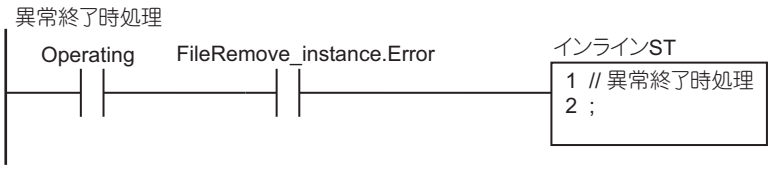
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ











ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	FileRemove_instance	FileRemove		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
    FileRemove_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

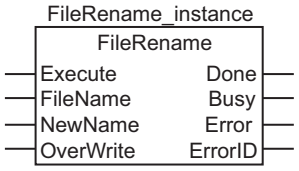
// FileRemove 命令実行
IF (Operating=TRUE) THEN
    FileRemove_instance(
        Execute :=TRUE,
        FileName:='ABC.bin');   // ファイル名

    IF (FileRemove_instance.Done=TRUE) THEN
        Operating:=FALSE;       // 正常終了
    END_IF;

    IF (FileRemove_instance.Error=TRUE) THEN
        Operating:=FALSE;       // 異常終了
    END_IF;
END_IF;
```

FileRename

SD メモリカード内の指定したファイルやディレクトリの、名称を変更します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
FileRename	ファイル名変更	FB		FileRename_instance(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
FileName	元ファイル名	入力	元のファイル名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	–	"
NewName	変更後ファイル名		変更後のファイル名			
OverWrite	上書許可		TRUE: 上書許可 FALSE: 上書禁止	データ型に従う		FALSE

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				○
NewName																				○
OverWrite	○																			

機能

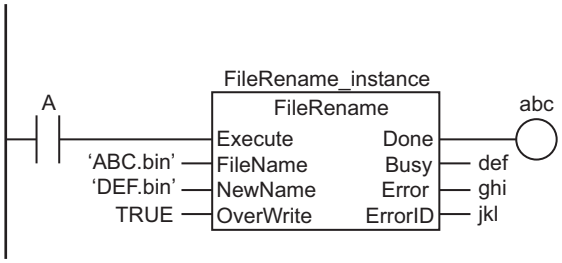
SD メモリカード内の、元ファイル名「FileName」で指定したファイルやディレクトリの名称を、変更後ファイル名「NewName」に変更します。

SD メモリカード内に、「NewName」と同じ名前のファイルやディレクトリがすでに存在している場合、上書許可「OverWrite」の値によって、以下のような処理を行います。

「OverWrite」の値	処理
TRUE（上書許可）	そのファイルやディレクトリに、上書します。
FALSE（上書禁止）	そのファイルやディレクトリに上書せず、異常になります。

記述例を、以下に示します。ファイル'ABC.bin'の名称を、'DEF.bin'に変更します。

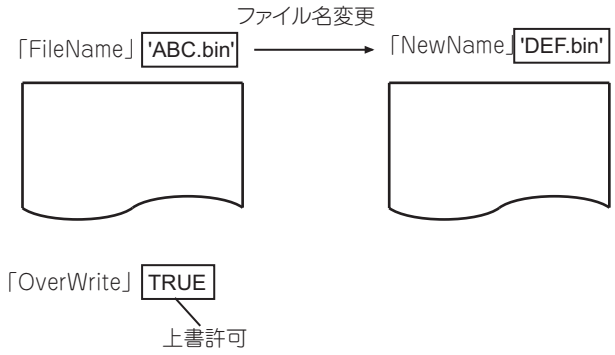
LD



ST

```
FileRename_instance(A, 'ABC.bin', 'DEF.bin',
                    TRUE, abc, def, ghi, jkl);
```

SDメモリカード内の、元ファイル名「FileName」で指定したファイルの名称を、
変更後ファイル名「NewName」に上書で変更します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- 「FileName」と「NewName」のディレクトリが異なる場合、ファイルは「NewName」で指定したディクトリに移動します。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) SD メモリカードが、使用可能状態ではない場合。
  b) SD メモリカードが、ライトプロテクト中の場合。
  c) 「FileName」で指定したファイルディレクトリが、存在しない場合。
  d) 「FileName」または「NewName」の値が、ファイル名やディレクトリ名として不正の場合。
  e) 「FileName」で指定したファイルが、アクセス中の場合。
  f) 「FileName」に指定したディレクトリ内にサブディレクトリが存在していて、かつ、「OverWrite」の値が TRUE の場合。
  g) 「NewName」と同じ名前のファイルがすでに存在していて、かつ、「OverWrite」の値が FALSE の場合。
  h) 「NewName」と同じ名前のファイルがすでに存在していて、そのファイルが書込禁止、かつ、「OverWrite」の値が TRUE の場合。
  i) 「NewName」の値が、ファイル名やディレクトリ名として使用できるバイト数を超えている場合。
  j) 作成可能な、ディレクトリ数を超える場合。
  k) 変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
  l) SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

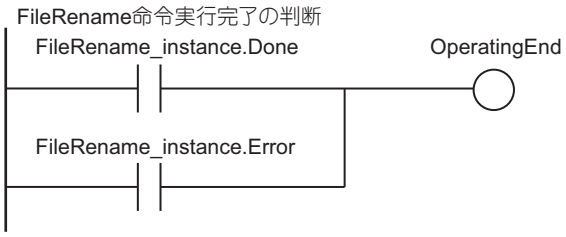
サンプルプログラム

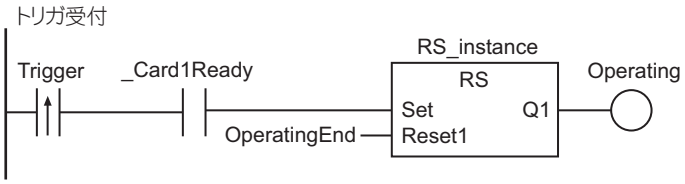
SD メモリカード内の、ファイル'ABC.bin'の名称を、'DEF.bin'に変更します。

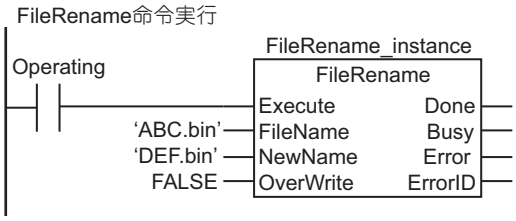
LD

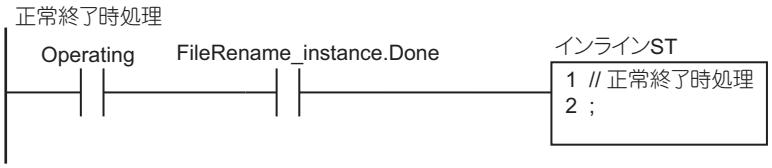
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	FileRename_instance	FileRename		

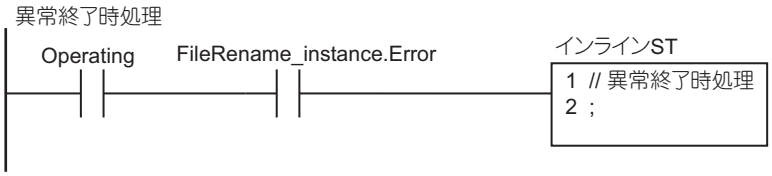
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ











ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	LastTrigger	BOOL	FALSE	前タスク周期の「Trigger」の値
	OperatingStart	BOOL	FALSE	処理開始
	Operating	BOOL	FALSE	処理中
	FileRename_instance	FileRename		

外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// インスタンス初期化
IF (OperatingStart=TRUE) THEN
    FileRename_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// FileRename 命令実行
IF (Operating=TRUE) THEN
    FileRename_instance(
        Execute  :=TRUE,
        FileName :='ABC.bin',     // 元ファイル名
        NewName  :='DEF.bin',     // 変更後ファイル名
        OverWrite:=FALSE);        // 上書禁止

    IF (FileRename_instance.Done=TRUE) THEN
        Operating:=FALSE;         // 正常終了
    END_IF;

    IF (FileRename_instance.Error=TRUE) THEN
        Operating:=FALSE;         // 異常終了
    END_IF;
END_IF;
```

DirCreate

SD メモリカード内に、指定した名称のディレクトリを作成します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DirCreate	ディレクトリ作成	FB	DirCreate_instance DirCreate Execute / DirName Done / Busy / Error / ErrorID	DirCreate_instance(Execute, DirName, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DirName	作成ディレクトリ名	入力	作成するディレクトリ名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	—	"

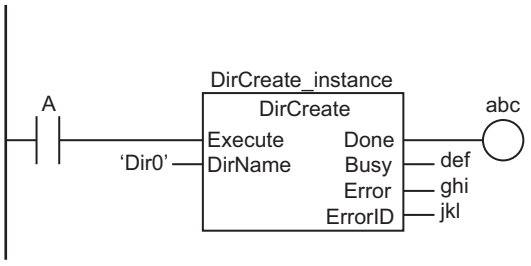
	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				○

機能

SD メモリカード内に、作成ディレクトリ名「DirName」で指定した名称のディレクトリを作成します。

記述例を、以下に示します。ディレクトリ'Dir0'を、作成します。

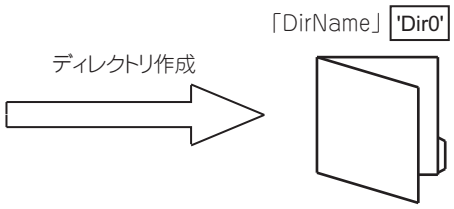
LD



ST

```
DirCreate_instance(A, 'Dir0', abc,
                   def, ghi, jkl);
```

SDメモリカード内に、「DirName」で指定した名称のディレクトリを作成します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。

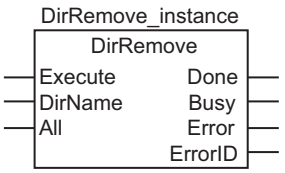
・つぎの場合、異常となります。「Error」は TRUE となります。
a）SD メモリカードが、使用可能状態ではない場合。
b）SD メモリカードが、ライトプロテクト中の場合。
c）SD メモリカードが、空き容量不足の場合。
d）「FileName」で指定したファイルが、アクセス中の場合。
e）作成可能な、ディレクトリ数を超える場合。
f）変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
g）「DirName」で指定したディレクトリが、すでに存在している場合。
h）「DirName」の値が、ディレクトリ名として不正の場合。
i）「DirName」の値が、ディレクトリ名として使用できるバイト数を超えている場合。
j）SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

FileCopy 命令の「サンプルプログラム (P.2-1286)」を、参照してください。

DirRemove

SD メモリカード内の、指定したディレクトリを削除します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
DirRemove	ディレクトリ削除	FB		DirRemove_instance(Execute, DirName, All, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DirName	削除ディレクトリ名	入力	削除するディレクトリ名	最大 66 バイト（半角英数字 65 文字+終端 NULL 文字）	–	''
All	全指定		ディレクトリ内にファイル／サブディレクトリが存在する場合の指定 TRUE: ファイル／サブディレクトリごと削除する FALSE: 削除しない	データ型に従う		FALSE

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				○
All	○																			

機能

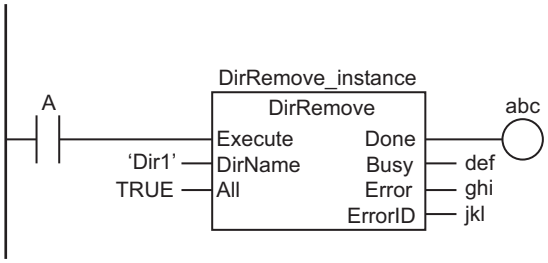
SD メモリカード内の、削除ディレクトリ名「DirName」で指定したディレクトリを、削除します。

指定したディレクトリ内にファイルやサブディレクトリが存在している場合、全指定「All」の値によって、以下のような処理を行います。

「All」の値	処理
TRUE	ファイルやサブディレクトリごと、指定したディレクトリを削除します。
FALSE	指定したディレクトリを削除せず、異常になります。

記述例を、以下に示します。ディレクトリ'Dir1'を削除します。

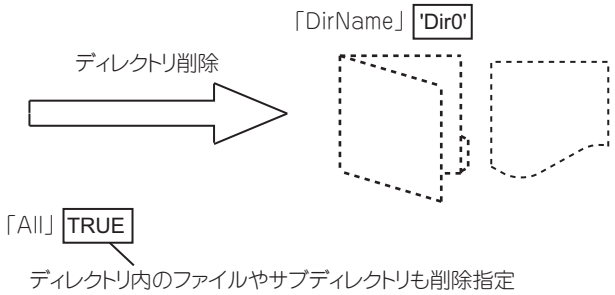
LD



ST

```
DirRemove_instance(A, 'Dir1', TRUE, abc,
                   def, ghi, jkl);
```

SDメモリカード内の、「DirName」で指定した名称のディレクトリを削除します。
ディレクトリ内のファイルやサブディレクトリも削除します。



関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

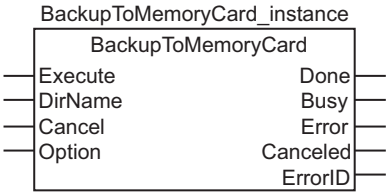
- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- ファイルがオープンされている状態で、CPU ユニット動作モードをプログラムに変更した場合、または全停止フォールトレベルのコントローラ異常が発生した場合、そのファイルを強制的にクローズします。ただし、そのときに実行中のデータの読み書きは、最後まで行います。
- 「DirName」で指定したディレクトリが書込禁止の場合、異常となり、そのディレクトリは削除されません。しかし、そのディレクトリ内に存在する、書込禁止でないファイルや、書込禁止でないディレクトリは削除されます。
- 同一ファイルへのアクセスを同時に行わないでください。ユーザプログラム内で、SD メモリカード命令どうしの排他制御を行ってください。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a) SD メモリカードが、使用可能状態ではない場合。
  b) SD メモリカードが、ライトプロテクト中の場合。
  c) 「All」の値が TRUE で、かつ、「DirName」で指定したディレクトリが、別の命令からアクセス中の場合。
  d) 「All」の値が FALSE で、かつ「DirName」で指定したディレクトリ内に、ファイルやディレクトリが存在する場合。
  e) 「DirName」で指定したディレクトリが、書込禁止の場合。
  f) 「DirName」で指定したディレクトリ内に、書込禁止のファイルや書込禁止のディレクトリが存在する場合。
  g) 「DirName」で指定したディレクトリが、存在しない場合。
  h) 「DirName」の値が、ディレクトリ名として使用できるバイト数を超えている場合。
  i) 変数に「FileID」を持たない SD メモリカード関連命令(FileWriteVar、FileReadVar、FileCopy、DirCreate、FileRemove、DirRemove、FileRename)を 5 個以上、同時に実行した場合。
  j) SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

FileCopy 命令の「サンプルプログラム (P.2-1286)」を、参照してください。

BackupToMemoryCard

SD メモリカードバックアップを実行します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
BackupToMemoryCard	SD メモリカードバックアップ	FB		BackupToMemoryCard_instance(Execute, DirName, Cancel, Option, Done, Busy, Error, Canceled, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
DirName	保存ディレクトリ名	入力	バックアップデータが保存されるディレクトリ名	最大 64 バイト（半角英数字 63 文字+終端 NULL 文字）	－	"
Cancel	取り消し		バックアップの取り消し TRUE: 取り消し実行 FALSE: 取り消し非実行	データ型に従う		FALSE
Option	将来拡張用		将来拡張用の変数 パラメータ接続は不要	－		－
Canceled	取り消し完了	出力	取り消し完了したかどうかを示すフラグ TRUE: 取り消し完了 FALSE: 取り消し失敗	データ型に従う	－	－

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				○
Cancel	○																			
Option	将来拡張用の変数　パラメータ接続は不要																			
Canceled	○																			

機能

SD メモリカードバックアップを、実行します。
SD メモリカードバックアップ命令の処理内容は、CPU ユニット前面スイッチ、システム定義変数 _Card1BkupCmd、または SysmacStudio の SD メモリカード画面を使って行う SD メモリカードバックアップの場合と同じです。

バックアップデータが保存されるディレクトリ名は、保存ディレクトリ名「DirName」で指定します。

「DirName」の値が ''（長さ 0 文字の文字列）の場合は、SD メモリカードのルートディレクトリに、バックアップデータが保存されます。
「DirName」は、省略可能です。「DirName」を省略した場合の保存ディレクトリは、以下のようになります。

本命令の実行回数	保存ディレクトリ
1 回め	ルートディレクトリ
2 回め以降	前回指定したディレクトリ

「DirName」で指定した保存ディレクトリが SD メモリカード内に存在しない場合は、ディレクトリを新規作成したうえで、バックアップデータが保存されます。
「DirName」で指定した保存ディレクトリに、バックアップファイルと同じ名前のファイルが存在している場合、上書きされます。

バックアップ処理中に、取り消し「Cancel」の値を FALSE から TRUE にすると、バックアップ処理が取り消されます。
バックアップ処理が取り消されると、バックアップファイルは作成されません。

「DirName」で指定した保存ディレクトリに、すでにバックアップファイルが存在している場合、そのファイルは上書きされずに残ります。
取り消しできるのは、同一の FB インスタンスで実行中のバックアップ処理だけです。
取り消しが完了すると、取り消し完了「Canceled」の値が TRUE になります。
「Cancel」の値を TRUE にするタイミングによっては、取り消しの受付が間に合わず、バックアップ処理が最後まで実行される場合があります。取り消しの受付が間に合わなかった場合、「Canceled」の値が FALSE、「Done」の値が TRUE になります。

「Cancel」の値が TRUE の場合、「Execute」の値を TRUE にしても、バックアップ処理は実行されません。

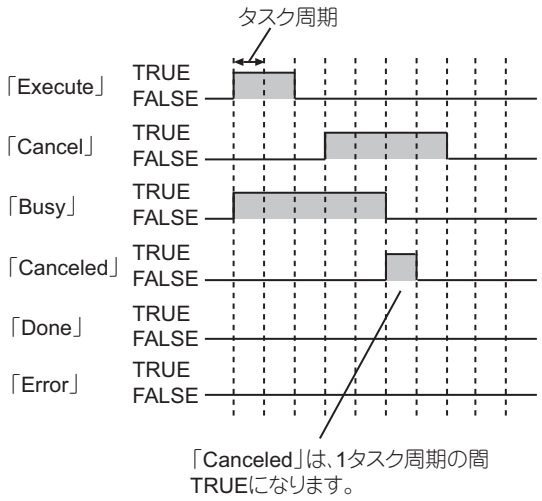
「Option」は、将来拡張用の変数です。パラメータを接続しないでください。

取り消し実行時のタイミングチャート

取り消しを実行した場合の、各変数のタイミングチャートを示します。

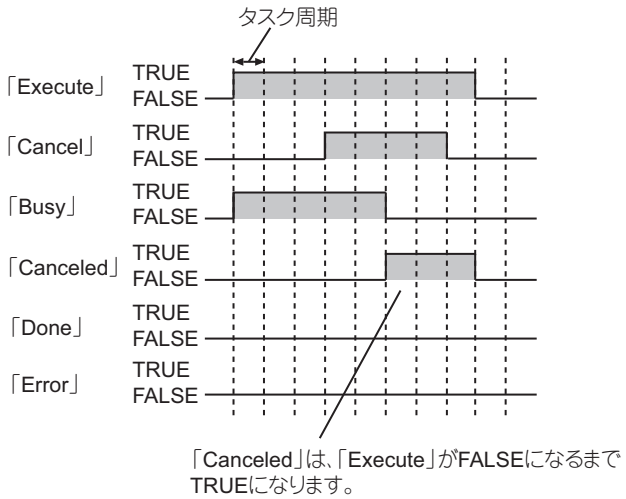
● 取り消しが成功し、「Canceled」が TRUE になる前に「Execute」を FALSE にする場合

- 「Execute」の値を TRUE にすると、バックアップ処理が実行されます。「Busy」の値が TRUE になります。
- 「Cancel」の値を TRUE にすると、バックアップ処理が取り消されます。
- 取り消しが完了すると、「Busy」の値が FALSE になり、「Canceled」の値が TRUE になります。
- 「Canceled」の値が TRUE になる前に、「Execute」の値を FALSE にします。
- 「Canceled」の値は、1 タスク周期後に FALSE になります。
- 取り消しが完了したため、「Done」の値は FALSE のままになります。



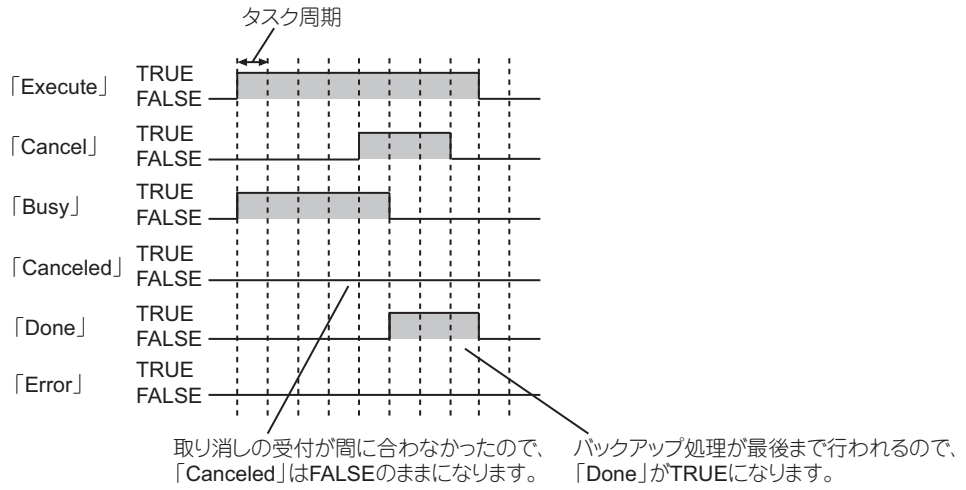
● 取り消しが成功し、「Canceled」が TRUE になった後に「Execute」を FALSE にする場合

- 「Execute」の値を TRUE にすると、バックアップ処理が実行されます。「Busy」の値が TRUE になります。
- 「Cancel」の値を TRUE にすると、バックアップ処理が取り消されます。
- 取り消しが完了すると、「Busy」の値が FALSE になり、「Canceled」の値が TRUE になります。
- 「Canceled」の値が TRUE になった後に、「Execute」の値を FALSE にします。
- 「Canceled」の値は、「Execute」の値が FALSE になるまで TRUE になります。
- 取り消しが完了したため、「Done」の値は FALSE のままになります。



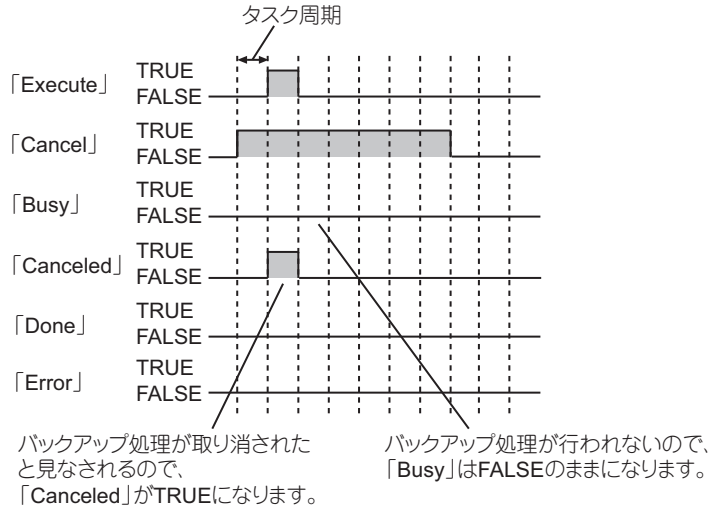
● 取り消しの受付が間に合わなかった場合

- 「Execute」の値を TRUE にすると、バックアップ処理が実行されます。「Busy」の値が TRUE になります。
- 「Cancel」の値を TRUE にします。しかし、取り消しの受付が間に合わず、バックアップ処理が継続されます。
- バックアップ処理が完了すると、「Busy」の値が FALSE になります。
- バックアップ処理が最後まで行われるので、「Done」の値が TRUE になります。
- 取り消しの受付が間に合わなかったので、「Canceled」の値は FALSE のままになります。



●「Cancel」の値が TRUE で、「Execute」の値を TRUE にする場合

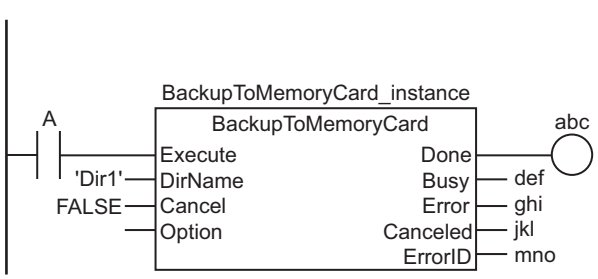
- 「Cancel」の値を TRUE にします。
- 「Execute」の値を TRUE にしても、バックアップ処理は行われません。したがって、「Busy」の値は FALSE のままになります。
- バックアップ処理が取り消されたと見なされるので、「Canceled」の値が TRUE になります。
- 「Execute」の値を FLASE にすると、「Canceled」の値が FALSE になります。



記述例

記述例を、以下に示します。ディレクトリ'Dir1'にバックアップファイルが保存されます。

LD



ST

```
BackupToMemoryCard_instance(A, 'Dir1', FALSE,
                            , abc, def, ghi, jkl,
                            mno);
```

関連するシステム定義変数

変数名	名称	データ型	内容
_Card1Ready	SD メモリカード使用可フラグ	BOOL	命令や通信コマンドでアクセス可能になっているかどうかを示すフラグです。 *1 TRUE: 使用可 FALSE: 使用不可
_Card1Protect *2	SD メモリカードライトプロテクトフラグ	BOOL	SD メモリカードが装着され、使用可能となっている状態で、かつ、SD メモリカードにライトプロテクトがかかっているかどうかを示すフラグです。 TRUE: ライトプロテクトあり FALSE: ライトプロテクトなし
_Card1Err *2	SD メモリカードエラーフラグ	BOOL	使用不可の SD メモリカードが装着されたかどうか、またはフォーマットに異常があるかどうかを示すフラグです。 TRUE: エラーあり FALSE: エラーなし
_Card1Access *2	SD メモリカードアクセス中フラグ	BOOL	SD メモリカードにアクセス中かどうかを示すフラグです。 TRUE: アクセス中 FALSE: 非アクセス中
_Card1Deteriorated *2	SD メモリカード寿命警告フラグ	BOOL	SD メモリカードの寿命を検知するフラグです。 TRUE: 寿命検知 FALSE: 検知せず
_Card1PowerFail	SD メモリカードアクセス中電源断フラグ	BOOL	アクセス中 *3 に電源断があり、実行中の処理が異常終了したことを示すフラグです。本フラグは、自動的にはクリアされません。 TRUE: 異常あり FALSE: 異常なし
_BackupBusy	バックアップ関連機能実行中フラグ	BOOL	バックアップ、リストア、照合の実行中であることを示すフラグです。 TRUE: バックアップ、リストア、照合実行中 FALSE: バックアップ、リストア、照合非実行中

*1. 共有フォルダ（仮想 SD メモリカード）が認識されていることが前提条件になります。
*2. NY シリーズコントローラでは、使用しません。FALSE 固定となります。
*3. 共有フォルダ（仮想 SD メモリカード）のアクセス中になります。

参考

- バックアップ機能の詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編(SBCA-436)』を参照してください。
- ファイル名のルートディレクトリは、SD メモリカードの直下を指します。

使用上の注意

- 本命令は、「Execute」の値が FALSE になっても、また実行時間がタスク周期を超えても、処理を最後まで続けます。処理が正常終了したかどうかは、「Done」の値が TRUE になることで確認してください。
- 「Execute」、「Done」、「Busy」、「Error」のタイミングチャートは、「本章の見方 (P.2-3)」を参照してください。
- SD メモリカードへのバックアップ禁止が設定されている場合でも、本命令を実行すると異常にはならず、バックアップが実行されます。
- 本命令を実行しても、以下のバックアップ関連のシステム定義変数の値は、変化しません。
  a）SD メモリカードバックアップ指示 _CardBkupCmd
  b）SD メモリカードバックアップステータス _Card1BkupSta

- 本命令の実行中に、バックアップ関連ファイルを読み書きしないでください。書込途中のファイルを読み出すなど、意図しない処理が行われる可能性があります。
- 本命令の実行中に、コントローラの動作モードを変更しても、バックアップ処理は継続されます。ただし、いったん運転モード→プログラムモード→運転モードと変化させた場合、バックアップ処理中でも「Busy」の値は FALSE になります。また、その場合でも取り消しは実行でき、「Canceled」の値も、TRUE になります。
- 本命令の実行中にコントローラの電源を OFF すると、次回電源投入時に全停止フォールトが発生する場合があります。
- つぎの場合、異常となります。「Error」は TRUE となります。
  a）SD メモリカードが、使用可能状態ではない場合。
  b）SD メモリカードが、ライトプロテクト中の場合。
  c）SD メモリカードが、空き容量不足の場合。
  d）作成可能なファイル数、ディレクトリ数を超える場合。
  e）「DirName」で指定したディレクトリと同じ名前のファイルが存在する場合。
  f）「DirName」の値が、ディレクトリ名として不正の場合。
  g）他のバックアップ処理が動作中の場合。
  h）バックアップ処理が失敗した場合。
  i） SD メモリカードアクセス中に、何らかの異常が発生し、アクセスができなくなった場合。

サンプルプログラム

毎日、日付が変わるたびに SD メモリカードバックアップを実行します。
バックアップ関連ファイルは、SD メモリカード内のディレクトリ/Backup/yyyy-mm-dd の下に格納されます。
ディレクトリ名は、バックアップ実行の年月日を表します。
“yyyy”は年、“mm”は月、“dd”は日です。

タッチパネルの仕様

本プログラムは、コントローラにタッチパネルを接続することを前提としています。
タッチパネルには、以下のランプがあります。

ランプ名	意味
バックアップ正常終了ランプ	バックアップ処理が正常終了した場合に点灯します。
バックアップ取り消しランプ	バックアップ処理の取り消しが成功した場合に点灯します。
バックアップ異常終了ランプ	バックアップ処理が異常終了した場合に点灯します。
SD メモリカード寿命警告ランプ	SD メモリカードの寿命を検知した場合に点灯します。
SD メモリカード電源断ランプ	バックアップ処理中に、SD メモリカードに電源断が発生した場合に点灯します。

また、タッチパネルには以下のボタンがあります。

ボタン名	押下した場合の動作
ランプ消灯ボタン	バックアップ正常終了ランプ、バックアップ取り消しランプ、バックアップ異常終了ランプ、SD メモリカード電源断ランプを消灯します。
取り消しボタン	実行中のバックアップを取り消します。

グローバル変数の定義

● グローバル変数

名称	データ型	初期値	コメント
PTOut_Warning_SDLife	BOOL	FALSE	SD メモリカード寿命警告ランプ出力
PTOut_Warning_PwrFail_onBackup	BOOL	FALSE	SD メモリカード電源断ランプ出力
PTOut_Done	BOOL	FALSE	バックアップ正常終了ランプ出力
PTOut_Cancel	BOOL	FALSE	バックアップ取り消しランプ出力
PTOut_Error	BOOL	FALSE	バックアップ異常終了ランプ出力
PTIn_Check_Backup	BOOL	FALSE	ランプ消灯ボタン入力
PTIn_Cancel	BOOL	FALSE	取り消しボタン入力

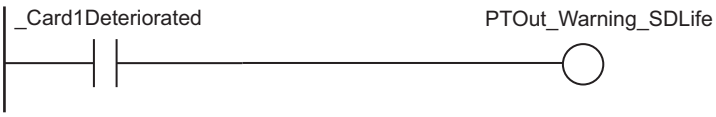
LD

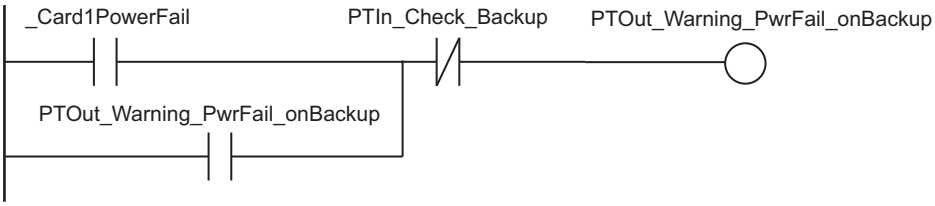
内部変数	名称	データ型	初期値	コメント
	CardOK	BOOL	FALSE	SD メモリカード正常フラグ
	Backup_inst	BackupToMemoryCard		BackupToMemoryCard 命令のインスタンス
	PreviousDay	USINT	0	前回タスク周期の日付
	CurrentDT	DATE_AND_TIME	ST#1970-01-01-00:00:00.000000000	現在の日付時刻
	Current_sDt	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	現在の日付時刻を、年、月、日、時、分、秒、ナノ秒に分解したもの
	BackupCondition	BOOL	FALSE	バックアップ条件成立フラグ
	tmpString	STRING[256]	''	ディレクトリ名作成途中の一時的な文字列
	tmpString2	STRING[256]	''	ディレクトリ名作成途中の一時的な文字列
	BackupPath	STRING[64]	''	ディレクトリ名
	Cancel	BOOL	FALSE	取り消し条件成立フラグ

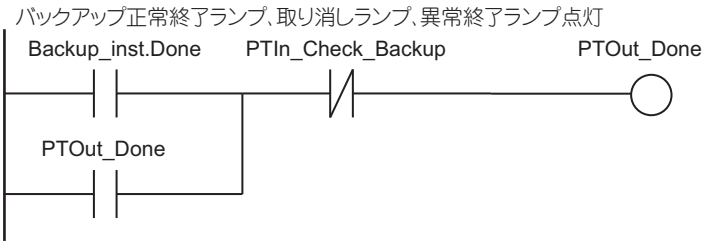
外部変数	名称	データ型	コンスタント	コメント
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ
	_Card1Protect	BOOL	☑	SD メモリカードライトプロテクトフラグ
	_Card1Err	BOOL	☑	SD メモリカードエラーフラグ
	_Card1Deteriorated	BOOL	☑	SD メモリカード寿命警告フラグ
	_Card1PowerFail	BOOL	☐	SD メモリカードアクセス中電源断フラグ
	_BackupBusy	BOOL	☑	バックアップ関連機能実行中フラグ

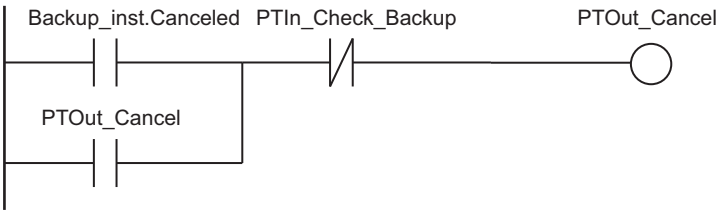
外部変数	名称	データ型	コンスタント	コメント
	PTOut_Warning_SDLife	BOOL	☐	SD メモリカード寿命警告ランプ出力
	PTOut_Warning_PwrFail_onBackup	BOOL	☐	SD メモリカード電源断ランプ出力
	PTOut_Done	BOOL	☐	バックアップ正常終了ランプ出力
	PTOut_Cancel	BOOL	☐	バックアップ取り消しランプ出力
	PTOut_Error	BOOL	☐	バックアップ異常終了ランプ出力
	PTIn_Check_Backup	BOOL	☐	ランプ消灯ボタン入力
	PTIn_Cancel	BOOL	☐	取り消しボタン入力

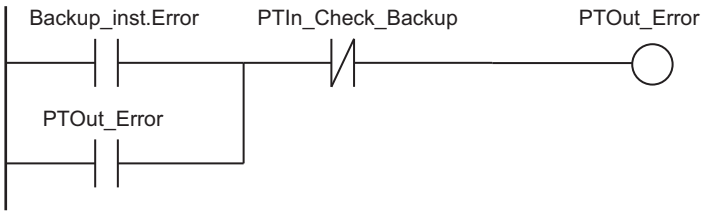




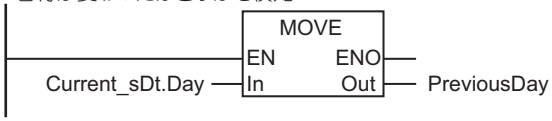


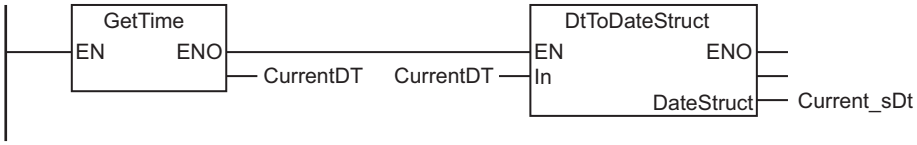


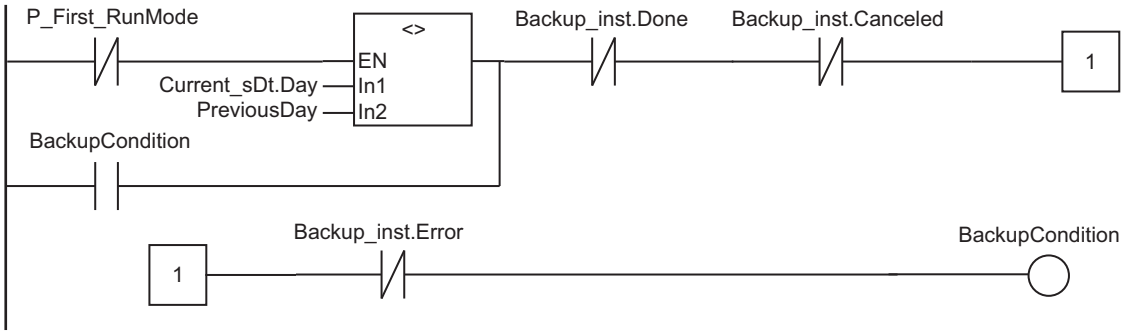




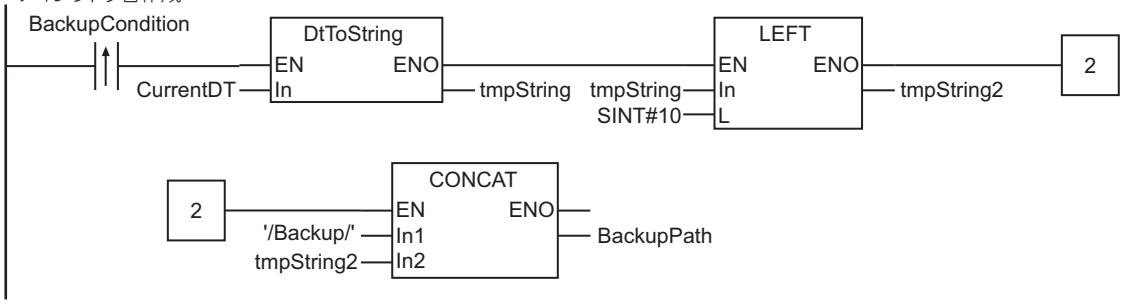
日付が変わったかどうかを検知



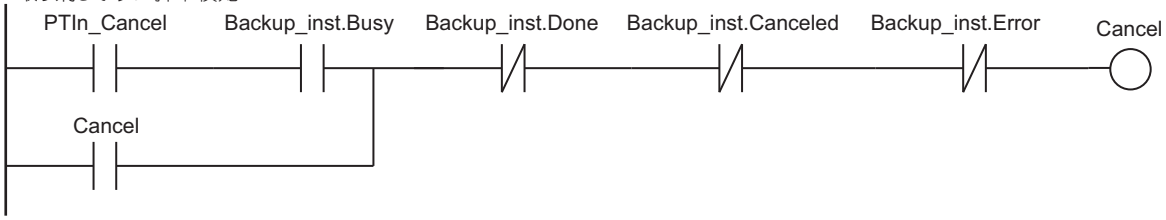


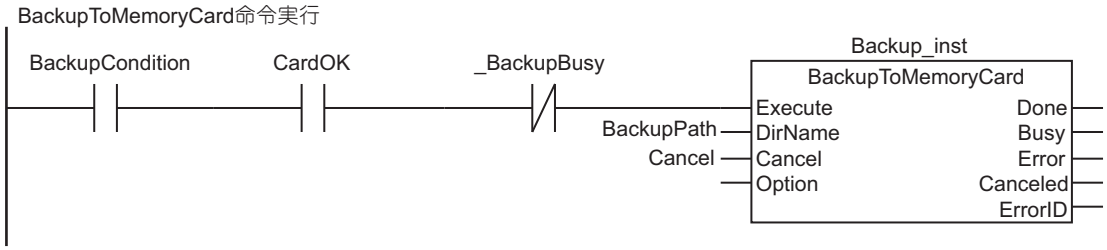


ディレクトリ名作成



取り消しボタン押下検知





ST

内部変数	名称	データ型	初期値	コメント
	CardOK	BOOL	FALSE	SD メモリカード正常フラグ
	Backup_inst	BackupToMemoryCard		BackupToMemoryCard 命令のインスタンス
	PreviousDay	USINT	0	前回タスク周期の日付
	CurrentDT	DATE_AND_TIME	ST#1970-01-01-00:00:00.000000000	現在の日付時刻
	Current_sDt	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	現在の日付時刻を、年、月、日、時、分、秒、ナノ秒に分解したもの
	BackupCondition	BOOL	FALSE	バックアップ条件成立フラグ
	tmpString	STRING[256]	''	ディレクトリ名作成途中の一時的な文字列
	tmpString2	STRING[256]	''	ディレクトリ名作成途中の一時的な文字列
	BackupPath	STRING[64]	''	ディレクトリ名
	Cancel	BOOL	FALSE	取り消し条件成立フラグ
	RS1	RS		リセット優先キープ命令のインスタンス 1
	RS2	RS		リセット優先キープ命令のインスタンス 2
	RS3	RS		リセット優先キープ命令のインスタンス 3
	RS4	RS		リセット優先キープ命令のインスタンス 4
	RS5	RS		リセット優先キープ命令のインスタンス 5
	RS6	RS		リセット優先キープ命令のインスタンス 6

外部変数	名称	データ型	コンスタント	コメント
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ
	_Card1Protect	BOOL	☑	SD メモリカードライトプロテクトフラグ
	_Card1Err	BOOL	☑	SD メモリカードエラーフラグ

外部変数	名称	データ型	コンスタント	コメント
	_Card1Deteriorated	BOOL	☑	SD メモリカード寿命警告フラグ
	_Card1PowerFail	BOOL	☐	SD メモリカードアクセス中電源断フラグ
	_BackupBusy	BOOL	☑	バックアップ関連機能実行中フラグ
	PTOut_Warning_SDLife	BOOL	☐	SD メモリカード寿命警告ランプ出力
	PTOut_Warning_PwrFail_onBackup	BOOL	☐	SD メモリカード電源断ランプ出力
	PTOut_Done	BOOL	☐	バックアップ正常終了ランプ出力
	PTOut_Cancel	BOOL	☐	バックアップ取り消しランプ出力
	PTOut_Error	BOOL	☐	バックアップ異常終了ランプ出力
	PTIn_Check_Backup	BOOL	☐	ランプ消灯ボタン入力
	PTIn_Cancel	BOOL	☐	取り消しボタン入力

```
// SD メモリカードの状態確認
CardOK := _Card1Ready OR NOT(_Card1Protect) OR NOT(_Card1Err);
PTOut_Warning_SDCardLife := _Card1Deteriorated;
RS1(Set := _Card1PowerFail, Reset1 := PTIn_Check_Backup, Q1=>PTOut_Warning_PwrFail_onBackup);

 // バックアップ正常終了ランプ、取り消しランプ、異常終了ランプ点灯
 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);

 // 日付が変わったかどうかを検知
 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);

 // ディレクトリ名作成
 IF(BackupCondition) THEN
    BackupPath := CONCAT('/Backup/', Left(In:= DtToString(CurrentDT), L:=SINT#10));
 END_IF;

 // 取り消しボタン押下検知
```

```
 RS6(Set    := (PTIn_Cancel &Backup_inst.Busy),
     Reset1 := (Backup_inst.Done OR Backup_inst.Canceled OR Backup_inst.Error),
     Q1     => Cancel);

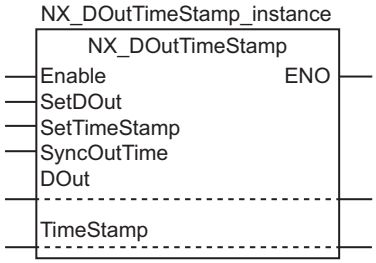
// BackupToMemoryCard 命令実行
Backup_inst(Execute := (BackupCondition & CardOK & NOT (_BackupBusy)),
            DirName := BackupPath,
            Cancel  := Cancel);
```

タイムスタンプ命令

命令	名称	ページ
NX_DOutTimeStamp	タイムスタンプデジタル出力書込	P.2-1320
NX_AryDOutTimeStamp	タイムスタンプデジタル出力配列書込	P.2-1325

NX_DOutTimeStamp

タイムスタンプ方式対応のデジタル出力ユニットの出力接点に、値を書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_DOutTimeStamp	タイムスタンプデジタル出力書込	FB		NX_DOutTimeStamp_instance(Enable, SetDOut, SetTimeStamp, SyncOutTime, DOut, TimeStamp);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	有効	入力	TRUE:「SetDOut」の値を出力する FALSE: 立ち下がり時に出力を FALSE にする	データ型に従う	−	FALSE
SetDOut	出力値		出力値			
SetTimeStamp	指定タイムスタンプ		出力する時刻		ns	0
SyncOutTime	出力同期時刻情報		EtherCAT カプラユニットのデバイス変数「Time Stamp of Synchronous Output」			*1
DOut	DOut ユニット出力接点	入出力	タイムスタンプ方式対応のデジタル出力ユニットのデバイス変数「Output Bit **」	データ型に従う	−	−
TimeStamp	タイムスタンプ		タイムスタンプ方式対応のデジタル出力ユニットのデバイス変数「Output Bit ** Time Stamp」		ns	

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
SetDOut	○																			
SetTimeStamp									○											
SyncOutTime									○											

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DOut	○																			
TimeStamp									○											

機能

有効「Enable」の値が TRUE の場合、指定した時刻に、タイムスタンプ方式対応のデジタル出力ユニットの出力接点に、出力値「SetDOut」を書き込みます。
「Enable」の値が TRUE から FALSE に立ち下がると、つぎのタスク周期から出力接点の値が FALSE になります。
指定時刻と出力時刻の誤差は、最大±1μs です。

出力同期時刻情報「SyncOutTime」は、タイムスタンプ方式対応のデジタル出力ユニットが接続している EtherCAT カプラユニットの基準時刻情報です。

接続している EtherCAT カプラユニットのデバイス変数「Time Stamp of Syncronous Output」を指定してください。
ただし、EtherCAT カプラユニットの I/O エントリに、0x200A:02(Time Stamp of Synchronous Output)を追加しておく必要があります。

DOut ユニット出力接点「DOut」には、タイムスタンプ方式対応のデジタル出力ユニットの、出力接点に割り付けたデバイス変数「Output Bit **」を指定してください。

タイムスタンプ「TimeStamp」には、タイムスタンプ方式対応のデジタル出力ユニットの、出力接点の時刻情報に割り付けたデバイス変数「Output Bit ** Time Stamp」を指定してください。

出力時刻の指定

出力時刻の指定は、以下の手順で行います。

1 **基準時刻とするユニットの接点の、時刻情報に割り付けたデバイス変数を取得します。**

2 **取得した時刻情報と出力接点にデータを書き込む時刻の差を、ns 単位の数値にして、1.で取得した変数に加えます。**

3 **加えた結果を、本命令の指定タイムスタンプ「SetTimeStamp」に渡します。**

詳細は、本命令の「サンプルプログラム (P.2-1322)」を参考にしてください。

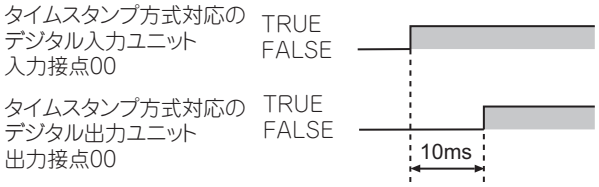
使用上の注意

- 本命令は、タイムスタンプ方式対応のデジタル出力ユニットに対してだけ実行可能です。ただし、タイムスタンプ方式対応のデジタル出力ユニットを接続していない状態で本命令を実行しても、異常にはなりません。
- EtherCAT 通信異常またはタスク周期超過が発生した場合は、指定時刻に書込できないことがあります。その場合、つぎのタスク周期以降に出力されます。

- 本命令で使用するデバイス変数を、他の命令やプログラムで変更や参照する場合は、排他処理を行ってください。
- 「SyncOutTime」には、タイムスタンプ方式対応のデジタル出力ユニットが直接接続されているEtherCAT カプラユニットのデバイス変数「Time Stamp of Synchronous Output」を指定してください。
  ただし、それ以外の変数を指定しても、異常にはなりません。
- 「DOut」と「TimeStamp」には、出力するタイムスタンプ方式対応のデジタル出力ユニットのデバイス変数を指定してください。ただし、それ以外の変数を指定しても、異常にはなりません。
- 「DOut」と「TimeStamp」には、同一ユニットの同一チャネル番号のデバイス変数を指定してください。ただし、それ以外の変数を指定しても、異常にはなりません。
- 「TimeStamp」の値が過去の時刻となる場合、「TimeStamp」の値が「0」になります。
  この場合、タイムスタンプ方式対応のデジタル出力ユニットの出力は、即時出力更新となります。
  詳細は、『NX シリーズ デジタル I/O ユニット ユーザーズマニュアル (SBCA-407)』を参照してください。

サンプルプログラム

タイムスタンプ方式対応のデジタル入力ユニットの入力接点 00 の値が FALSE から TRUE に変化したら、その 10ms 後にタイムスタンプ方式対応のデジタル出力ユニットの出力接点 00 の値を、FALSE から TRUE に変化させます。
入力接点 00 の値は、NX バスの I/O リフレッシュの時間よりも長い期間、TRUE であるとします。
本サンプルプログラムでは、入力接点 00 の立ち上がりを入力トリガとしています。
入力接点 00 の値が TRUE である期間が NX バスの I/O リフレッシュの時間よりも短い場合は、入力接点 00 の立ち上がりを検知できない可能性があります。その場合は、入力接点 00 の変化時刻の変化を入力トリガにするなどの変更が必要です。
センサ入力変化から一定の時間で出力を ON にするサンプルプログラムは、『NX シリーズ デジタル I/O ユニット ユーザーズマニュアル(SBCA-407)』を参照してください。



ネットワーク構成

ネットワーク構成は、以下のとおりです。EtherCAT のノードアドレス 1 に、以下の構成のスレーブターミナルが接続しています。それぞれに、以下のデバイス名が割り付けられています。

ユニット番号	形式	ユニット	デバイス名
0	形 NX-ECC201	EtherCAT カプラユニット	E001
1	形 NX-ID3344	タイムスタンプ方式対応のデジタル入力ユニット	N1
2	形 NX-OD2154	タイムスタンプ方式対応のデジタル出力ユニット	N2

ユニット動作設定

タイムスタンプ方式対応のデジタル入力ユニットのユニット動作設定は、以下のとおりです。

項目名	設定値	意味
タイムスタンプ（トリガ設定）／Input Bit 00 Trigger Setting	FALSE	変化時刻を取得するエッジ: 立ち上がり
タイムスタンプ（モード設定）／Input Bit 00 Mode Setting	TRUE	変化時刻を取得する動作モード: 単発取得（最初に変化した時刻）

I/O マップ

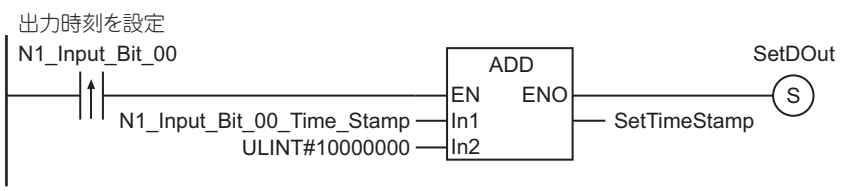
I/O マップの設定は、以下のとおりです。

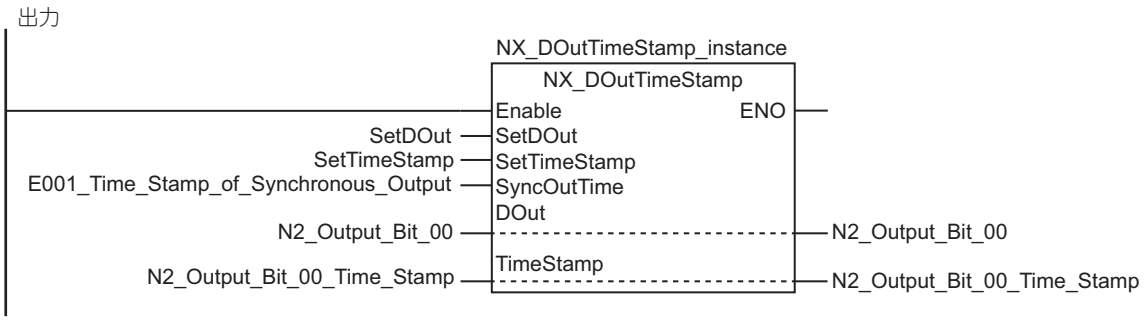
位置	ポート	説明	R/W	データ型	変数	変数種別
Node1	Time Stamp of Synchronous Output	接続された NX ユニットが同期出力したタイミングの時刻情報が格納されます。（単位:ns）	R	ULINT	E001_Time_Stamp_of_Synchronous_Output	グローバル変数
Unit1	Input Bit 00	入力接点 00	R	BOOL	N1_Input_Bit_00	グローバル変数
Unit1	Input Bit 00 Time Stamp	入力接点 00 の変化時刻	R	ULINT	N1_Input_Bit_00_Time_Stamp	グローバル変数
Unit2	Output Bit 00 Time Stamp	出力接点 00 の指定時刻	W	ULINT	N2_Output_Bit_00_Time_Stamp	グローバル変数
Unit2	Output Bit 00	出力接点 00	W	BOOL	N2_Output_Bit_00	グローバル変数

LD

内部変数	名称	データ型	初期値	コメント
	SetTimeStamp	ULINT	0	指定タイムスタンプ
	SetDOut	BOOL	FALSE	出力値
	NX_DOutTimeStamp_instance	NX_DOutTimeStamp		

外部変数	名称	データ型	コンスタント	コメント
	N1_Input_Bit_00	BOOL	□	入力接点 00
	N1_Input_Bit_00_Time_Stamp	ULINT	□	入力接点 00 の変化時刻
	E001_Time_Stamp_of_Synchronous_Output	ULINT	□	接続された NX ユニットが同期出力したタイミングの時刻情報
	N2_Output_Bit_00	BOOL	□	出力接点 00
	N2_Output_Bit_00_Time_Stamp	ULINT	□	出力接点 00 の指定時刻





ST

内部変数	名称	データ型	初期値	コメント
	SetEN	BOOL	FALSE	実行条件
	SetTimeStamp	ULINT	0	指定タイムスタンプ
	SetDOut	BOOL	FALSE	出力値
	R_TRIG_instance	R_TRIG		
	NX_DOutTimeStamp_instance	NX_DOutTimeStamp		

外部変数	名称	データ型	コンスタント	コメント
	N1_Input_Bit_00	BOOL	┌┐	入力接点 00
	N1_Input_Bit_00_Time_Stamp	ULINT	┌┐	入力接点 00 の変化時刻
	E001_Time_Stamp_of_Synchronous_Output	ULINT	┌┐	接続された NX ユニットが同期出力したタイミングの時刻情報
	N2_Output_Bit_00	BOOL	┌┐	出力接点 00
	N2_Output_Bit_00_Time_Stamp	ULINT	┌┐	出力接点 00 の指定時刻

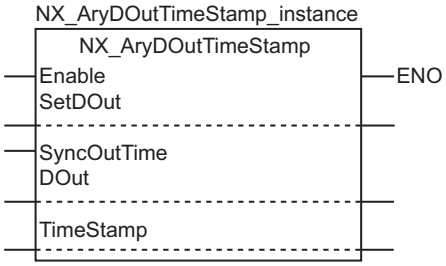
```
// 実行トリガ入力
R_TRIG_instance( N1_Input_Bit_00, SetEN);

// 出力時刻設定
IF ( SetEN = TRUE ) THEN
   SetDOut      := TRUE;
   SetTimeStamp := N1_Input_Bit_00_Time_Stamp + ULINT#10000000;
END_IF;

// 出力
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

タイムスタンプ方式対応のデジタル出力ユニットから、パルスを出力させます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_AryDOutTimeStamp	タイムスタンプデジタル出力配列書込	FB		NX_AryDOutTimeStamp_instance(Enable, SetDOut, SyncOutTime, DOut, TimeStamp);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Enable	有効	入力	TRUE:「SetDOut」の設定に従った出力を行う FALSE: 立ち下がり時に出力を FALSE にする	データ型に従う	−	FALSE
SyncOutTime	出力同期時刻情報		EtherCAT カプラユニットのデバイス変数「Time Stamp of Synchronous Output」		ns	*1
SetDOut	出力パルス	入出力	出力パルス	−	−	−
DOut	DOut ユニット出力接点		タイムスタンプ方式対応のデジタル出力ユニットのデバイス変数「Output Bit **」	データ型に従う		
TimeStamp	タイムスタンプ		タイムスタンプ方式対応のデジタル出力ユニットのデバイス変数「Output Bit ** Time Stamp」		ns	

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	○																			
SyncOutTime									○											
SetDOut	構造体_sOUTPUT_REF　詳細は「出力パルスの指定 (P.2-1326)」参照																			
DOut	○																			
TimeStamp									○											

機能

有効「Enable」の値が TRUE の場合、指定した時刻に、タイムスタンプ方式対応のデジタル出力ユニットから、出力パルス「SetDOut」で設定したパルスを出力します。

「Enable」の値が TRUE から FALSE に立ち下がったときに、タイムスタンプ方式対応のデジタル出力ユニットへの出力を FALSE にします。
指定時刻と出力時刻の誤差は、最大±1μs です。

出力同期時刻情報「SyncOutTime」は、タイムスタンプ方式対応のデジタル出力ユニットが接続している EtherCAT カプラユニットの基準時刻情報です。
接続している EtherCAT カプラユニットのデバイス変数「Time Stamp of Syncronous Output」を指定してください。
ただし、EtherCAT カプラユニットの I/O エントリに、0x200A:02(Time Stamp of Synchronous Output)を追加しておく必要があります。

DOut ユニット出力接点「DOut」には、タイムスタンプ方式対応のデジタル出力ユニットの、出力接点に割り付けたデバイス変数「Output Bit **」を指定してください。

タイムスタンプ「TimeStamp」には、タイムスタンプ方式対応のデジタル出力ユニットの、出力接点の時刻情報に割り付けたデバイス変数「Output Bit ** Time Stamp」を指定してください。

出力時刻の指定

出力時刻の指定は、以下の手順で行います。

1 **基準時刻とするユニットの接点の、時刻情報に割り付けたデバイス変数を取得します。**

2 **取得した時刻情報と出力接点を ON する時刻の差を、ns 単位の数値にして、1.で取得した変数に加えます。**

3 **加えた結果を、本命令の SetDOut.OnTime[]に渡します。**

4 **2.と同様に、取得した時刻情報と出力接点を OFF する時刻の差を、ns 単位の数値にして、1.で取得した変数に加えます。**

5 **加えた結果を、本命令の SetDOut.OffTime[]に渡します。**

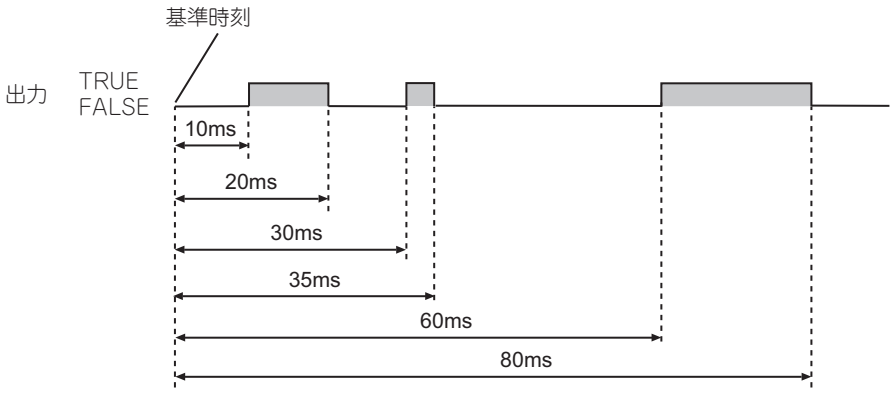
出力パルスの指定

出力パルス「SetDOut」のデータ型は、構造体_sOUTPUT_REF です。仕様は、以下のとおりです。

変数	名称	内容	データ型	有効範囲	単位	初期値
SetDOut	出力パルス	出力パルス	_sOUTPUT_REF	−	−	−
EnableOut	出力有効	出力有効フラグ TRUE:「OnTime」、「OffTime」の設定が有効 FALSE:「OnTime」、「OffTime」の設定が無効	BOOL	データ型に従う	−	FALSE
OnTime[]配列	ON 時刻	出力接点を ON にする時刻	ARRAY[0..15] OF ULINT		ns	すべての要素が 0
OffTime[]配列	OFF 時刻	出力接点を OFF にする時刻	ARRAY[0..15] OF ULINT			

ON 時刻 OnTime[]と OFF 時刻 OffTime[]は、それぞれ要素数 16 の配列です。
2 つの配列の同一要素番号の値が、ひとつのパルスの ON 時刻と OFF 時刻を表します。
したがって、最大 16 個のパルスを指定できます。
2 つの配列の同一要素番号の値が両方とも 0 であれば、その要素番号以降の値は無効になります。
たとえば、OnTime[]と OffTime[]の要素の値が以下の場合、出力は下図のようになります。下表の値は、指定時刻を基準時刻の何 ms 後かで表現しています。

変数	要素番号				
	0	1	2	3	4
OnTime[]	10ms 後	30ms 後	60ms 後	0	90ms 後
OffTime[]	20ms 後	35ms 後	80ms 後	0	100ms 後



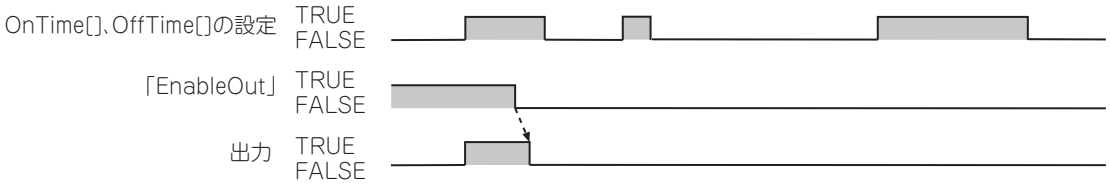
OnTime[]と OffTime[]の要素の値は、時間の昇順に並んでいる必要はありません。したがって、2 つの配列の要素の値が以下の場合も、出力は上図と同じになります。

変数	要素番号				
	0	1	2	3	4
OnTime[]	30ms 後	60ms 後	10ms 後	0	90ms 後
OffTime[]	35ms 後	80ms 後	20ms 後	0	100ms 後

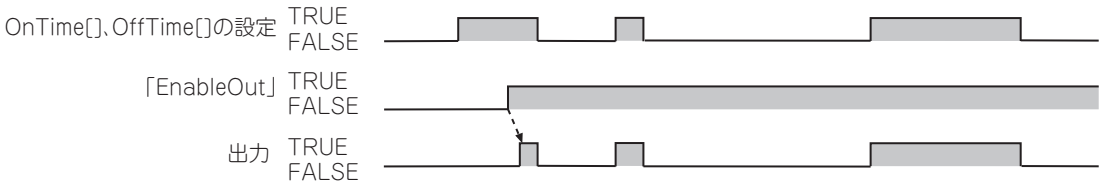
● 出力有効「EnableOut」

出力有効「EnableOut」は、OnTime[]、OffTime[]の設定を有効にするフラグです。
「EnableOut」の値が FALSE の場合、OnTime[]、OffTime[]の値に関係なく、出力の値は FALSE になります。

「EnableOut」の値を、本命令実行中に変更することが可能です。
「EnableOut」の値を TRUE から FALSE に変更すると、出力の値が FALSE になります。

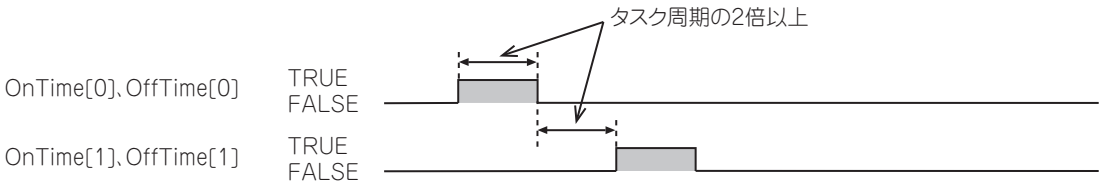


「EnableOut」の値を FALSE から TRUE に変更すると、OnTime[]、OffTime[]の値が有効になります。



● 出力パルスの最小幅

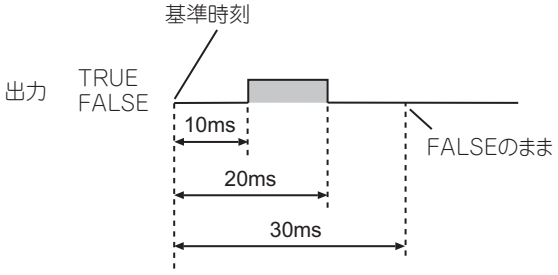
出力パルスを 1µs の時間精度で出力するためには、OnTime[]と OffTime[]の間隔を、タスク周期の 2 倍以上としてください。
2 倍未満の値を指定すると、指定したパルスが出力されない場合や、指定した ON/OFF 時刻から 1 タスク周期遅れて出力される場合があります。



● OnTime[]と OffTime[]の同一要素番号の値が同じ場合

OnTime[]と OffTime[]の同一要素番号の値が同じ場合、出力は FALSE になります。したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

変数	要素番号		
	0	1	2
OnTime[]	10ms 後	30ms 後	0
OffTime[]	20ms 後	30ms 後	0

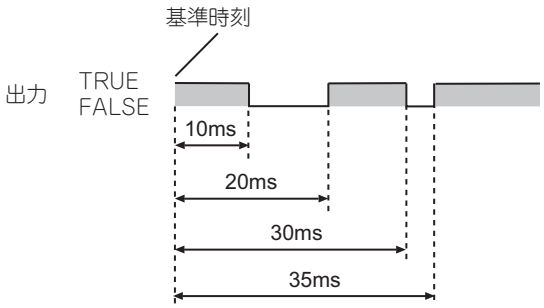


● 同一要素番号の OffTime[]の要素の値より OnTime[]の要素の値の方が大きい場合

同一要素番号の OffTime[]の要素の値より OnTime[]の要素の値の方が大きい場合、出力の値は先に FALSE になった後、TRUE になります。

また、OffTime[]の要素の最小値より OnTime[]の要素の最小値の方が大きい場合、本命令実行後直ちに、出力の値が TRUE になります。
さらに、OffTime[]の要素の最大値より OnTime[]の要素の最大値の方が大きい場合、本命令実行終了後、出力の値は TRUE が保持されます。
したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

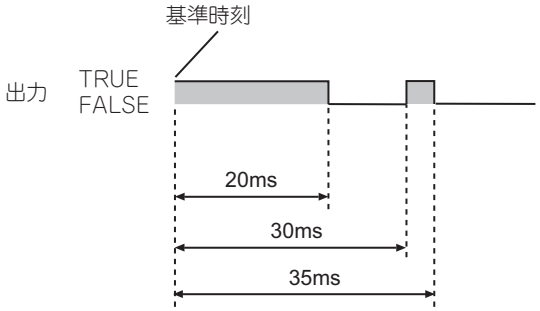
変数	要素番号		
	0	1	2
OnTime[]	20ms 後	35ms 後	0
OffTime[]	10ms 後	30ms 後	0



● OnTime[]と OffTime[]のいずれかの要素の値が 0 の場合

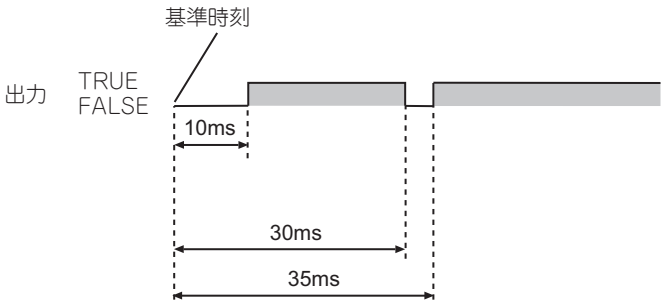
OnTime[]と OffTime[]のいずれかの要素が 0 の場合、本命令実行後、直ちに出力が TRUE または FALSE にすることを意味します。
OnTime[]の要素の値だけが 0 の場合、本命令実行後、出力の値が直ちに TRUE になります。したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

変数	要素番号		
	0	1	2
OnTime[]	0	30ms 後	0
OffTime[]	20ms 後	35ms 後	0



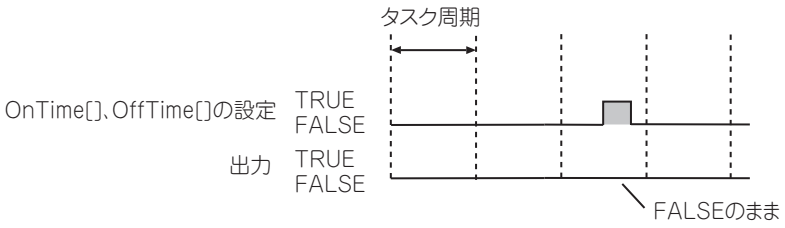
OffTime[]の要素の値だけが 0 の場合、本命令実行後、出力の値が直ちに FALSE になります。したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

変数	要素番号		
	0	1	2
OnTime[]	10ms 後	35ms 後	0
OffTime[]	0	30ms 後	0



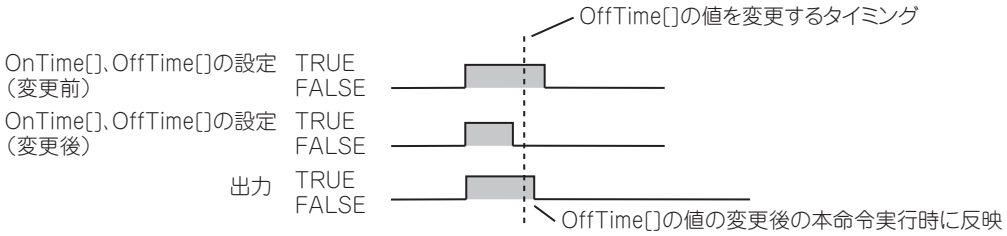
● 同一タスク周期内で、出力の値を TRUE にする設定と FALSE にする設定が連続した場合

同一タスク周期内で、出力を TRUE にする設定と FALSE にする設定が連続した場合、出力の値は変化しません。



● 本命令が有効中に OnTime[]や OffTime[]の値を変更した場合

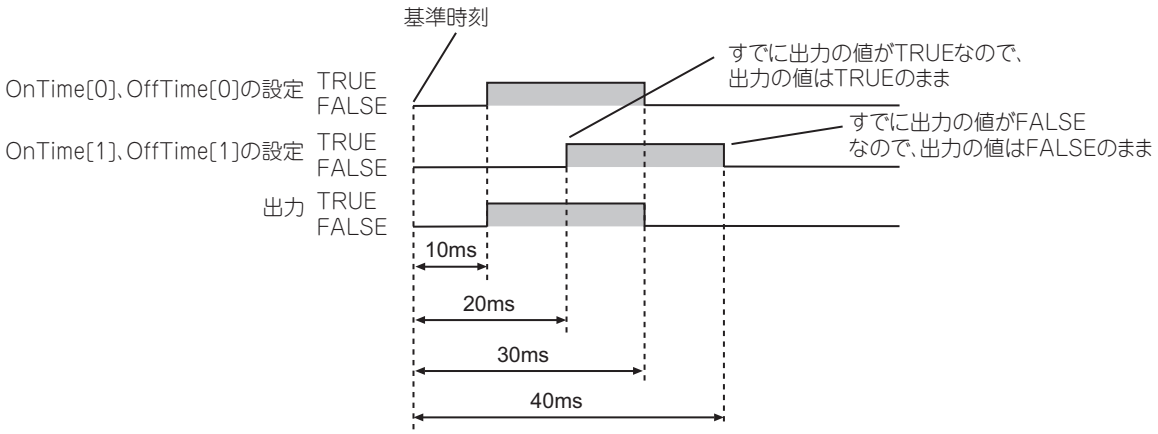
本命令が有効中に、OnTime[]や OffTime[]の値を変更することは可能です。
変更内容は、変更後の本命令実行時から反映されます。



● 出力の値を TRUE にする設定が重なった場合

出力の値を TRUE にする設定が重なった場合、異常にはならず、出力の値は TRUE のままになります。
出力の値を FALSE にする設定が重なった場合も同様です。
したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

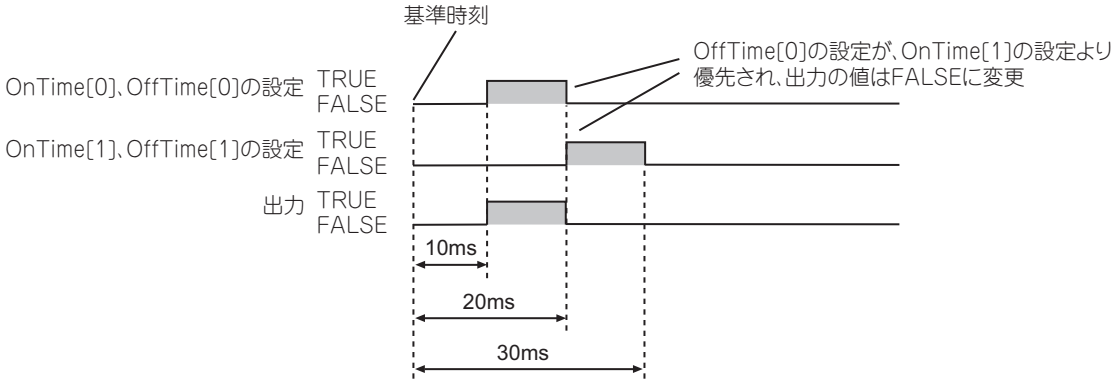
変数	要素番号		
	0	1	2
OnTime[]	10ms 後	20ms 後	0
OffTime[]	30ms 後	40ms 後	0



● 出力の値を TRUE にする設定と FALSE にする設定が、同一時刻で重なった場合

出力の値を TRUE にする設定と FALSE にする設定が、同一時刻が重なった場合、異常にはならず、OnTime[]と OffTime[]の要素番号が小さい方の設定が優先されます。したがって、2 つの配列の要素の値が以下の場合は、出力は下図のようになります。

変数	要素番号		
	0	1	2
OnTime[]	10ms 後	20ms 後	0
OffTime[]	20ms 後	30ms 後	0



参考

本命令は、MC_DigitalCamSwitch 命令を使用する場合に使用します。
MC_DigitalCamSwitch 命令の詳細は、『NY シリーズ コマンドリファレンスマニュアル モーション編(SBCE-380)』を、参照してください。

使用上の注意

- 本命令は、タイムスタンプ方式対応のデジタル出力ユニットに対してだけ実行可能です。ただし、タイムスタンプ方式対応のデジタル出力ユニットを接続していない状態で本命令を実行しても、異常にはなりません。
- EtherCAT 通信異常またはタスク周期超過が発生した場合は、指定時刻に出力できないことがあります。その場合、つぎのタスク周期以降に出力されます。
- 本命令で使用するデバイス変数を、他の命令やプログラムで変更や参照する場合は、排他処理を行ってください。

- 「SyncOutTime」には、タイムスタンプ方式対応のデジタル出力ユニットが直接接続されている EtherCAT カプラユニットのデバイス変数「Time Stamp of Synchronous Output」を指定してください。
  ただし、それ以外の変数を指定しても、異常にはなりません。
- 「DOut」と「TimeStamp」には、出力するタイムスタンプ方式対応のデジタル出力ユニットのデバイス変数を指定してください。ただし、それ以外の変数を指定しても、異常にはなりません。
- 「DOut」と「TimeStamp」には、同一ユニットの同一チャネル番号のデバイス変数を指定してください。ただし、それ以外の変数を指定しても、異常にはなりません。
- 「TimeStamp」の値が過去の時刻となる場合、「TimeStamp」の値が「0」になります。
  この場合、タイムスタンプ方式対応のデジタル出力ユニットの出力は、即時出力更新となります。
  詳細は、『NX シリーズ デジタル I/O ユニット ユーザーズマニュアル (SBCA-407)』を参照してください。

サンプルプログラム

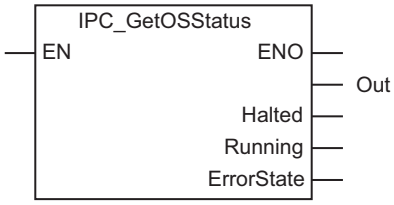
サンプルプログラムは、『NY シリーズ コマンドリファレンスマニュアル モーション編(SBCE-380)』の、MC_DigitalCamSwitch 命令を参照してください。

OS 制御命令

命令	名称	ページ
IPC_GetOSStatus	OS ステータス読み出し	P.2-1334
IPC_RebootOS	OS 再起動	P.2-1337
IPC_Shutdown	シャットダウン	P.2-1340

IPC_GetOSStatus

産業用 PC のオペレーティングシステム(Windows)のステータスを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_GetOSStatus	OS ステータス読み出し	FUN		IPC_GetOSStatus(Halted, Running, ErrorState);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−
Halted	OS 停止中フラグ		システム変数「_OSHalted」の値を返す	データ型に従う	−	−
Running	OS 稼働中フラグ		システム変数「_OSRunning」の値を返す	データ型に従う	−	−
ErrorState	OS の異常状態		システム変数「_OSErrorState」の値を返す	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			
Halted	○																			
Running	○																			
ErrorState	○																			

機能

産業用 PC のオペレーティングシステム(Windows)のステータスを読み出します。
読み出す情報は、OS 停止中フラグ「Halted」、OS 稼働中フラグ「Running」、OS の異常状態「ErrorState」です。

参考

本命令をシミュレータで動作させた場合、常に「Running」が TRUE となります。

関連するシステム定義変数

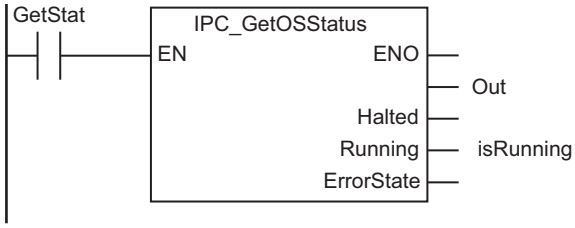
変数名	名称	データ型	有効範囲	内容
_OSRunning	OS 稼働中フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを稼働中と判断した場合、TRUE になります。
_OSHalted	OS 停止中フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを停止中と判断した場合、TRUE になります。
_OSErrorState	OS 異常状態フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを異常発生中と判断した場合、TRUE になります。

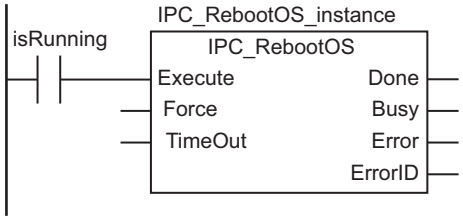
サンプルプログラム

IPC_GetOSStatus（OS ステータス読み出し）命令でオペレーティングシステム(Windows)のステータスを読み出します。
出力変数「Running」の値が TRUE の場合に IPC_RebootOS（OS 再起動）命令を実行し、オペレーティングシステム(Windows)を再起動します。

LD

内部変数	名称	データ型	初期値	コメント
	GetStat	BOOL	FALSE	OS ステータス読み出し
	Out	BOOL	FALSE	
	isRunning	BOOL	FALSE	OS 稼働中フラグ
	IPC_RebootOS_instance	IPC_RebootOS		





ST

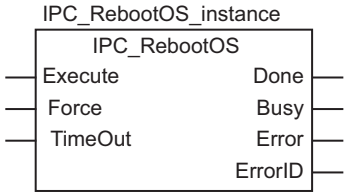
内部変数	名称	データ型	初期値	コメント
	GetStat	BOOL	FALSE	OS ステータス読み出し
	Out	BOOL	FALSE	
	isRunning	BOOL	FALSE	OS 稼働中フラグ
	IPC_RebootOS_instance	IPC_RebootOS		

```
IF GetStat THEN
  Out:=IPC_GetOSStatus(Running=>isRunning);
```

```
End_IF;
IPC_RebootOS_instance(Execute:=isRunning);
```

IPC_RebootOS

産業用 PC のオペレーティングシステム(Windows)を再起動します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_RebootOS	OS 再起動	FB		IPC_RebootOS_instance(Execute, Force, TimeOut, Done, Busy, Error, ErrorID);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Force	強制実行	入力	強制再起動	データ型に従う	−	FALSE
TimeOut	監視時間		監視時間を指定 「0」の場合は、初期値の「600s」で監視	0ns ～ 24h	ns	600s

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Force	○																			
TimeOut																○				

機能

「Force」（強制実行）が FALSE の場合に、「Execute」（起動）を FALSE から TRUE にすると、オペレーティングシステム(Windows)を再起動させます。
「Force」（強制実行）が TRUE の場合に、「Execute」（起動）を FALSE から TRUE にすると、オペレーティングシステム(Windows)を強制再起動させます。

強制再起動は、オペレーティングシステム(Windows)がどのような状態でも強制的に再起動します。
強制再起動は、オペレーティングシステム(Windows)の状態によっては、システムに何らかのダメージを与える可能性があります。
強制再起動は、オペレーティングシステム(Windows)がブルースクリーンのような致命的なエラーが発生している場合に使用してください。

本命令を実行すると、「Busy」の値は TRUE となり、オペレーティングシステム(Windows)が再起動します。
オペレーティングシステム(Windows)が稼働中になると、「Done」の値は TRUE、「Busy」の値は FALSE となります。
本命令が実行されてから、「TimeOut」（監視時間）以内にオペレーティングシステム(Windows)の状態が稼働中にならない場合は、タイムアウトでエラーとなります。

参考

本命令をシミュレータで動作させた場合、「Execute」の値が TRUE になると「Done」の値は TRUE となりますが、オペレーティングシステム(Windows)は再起動しません。

関連するシステム定義変数

変数名	名称	データ型	有効範囲	内容
_OSRunning	OS 稼働中フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを稼働中と判断した場合、TRUE になります。
_OSHalted	OS 停止中フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを停止中と判断した場合、TRUE になります。
_OSErrorState	OS 異常状態フラグ	BOOL	FALSE, TRUE	コントローラがオペレーティングシステム(Windows)のステータスを異常発生中と判断した場合、TRUE になります。

使用上の注意

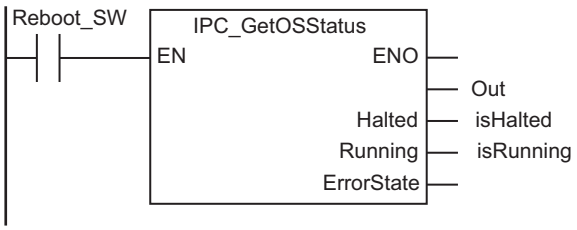
つぎの場合、異常となります。
- 監視時間が有効範囲外の場合。
- 「TimeOut」で指定した時間内にオペレーティングシステムが再起動しない場合。
- オペレーティングシステムが稼働中でも停止中でもない場合に、再起動を実行した場合。

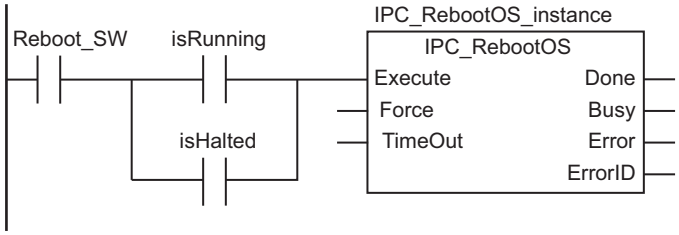
サンプルプログラム

オペレーティングシステム(Windows)が稼働中または停止中の場合に、「Reboot_SW」の値が TRUE になると、オペレーティングシステム(Windows)が再起動します。
オペレーティングシステム(Windows)が再起動したことは、「IPC_RebootOS_instance.Done」の値が TRUE になることで判断します。

LD

内部変数	名称	データ型	初期値	コメント
	Reboot_SW	BOOL	FALSE	OS 再起動
	Out	BOOL	FALSE	
	isHalted	BOOL	FALSE	OS 停止中フラグ
	isRunning	BOOL	FALSE	OS 稼働中フラグ
	IPC_RebootOS_instance	IPC_RebootOS		





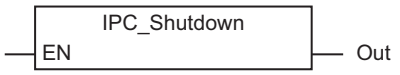
ST

内部変数	名称	データ型	初期値	コメント
	Reboot_SW	BOOL	FALSE	OS 再起動
	Out	BOOL	FALSE	
	isHalted	BOOL	FALSE	OS 停止中フラグ
	isRunning	BOOL	FALSE	OS 稼働中フラグ
	IPC_RebootOS_instance	IPC_RebootOS		

```
IF Reboot_SW THEN
  Out:=IPC_GetOSStatus(Halted=>isHalted,Running=>isRunning);
End_IF;
IPC_RebootOS_instance(Execute:=(Reboot_SW AND (isHalted OR isRunning)));
```

IPC_Shutdown

産業用 PC のシステムでのシャットダウン処理を開始し、完了後に Windows 側にシャットダウン要求を通知します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_Shutdown	シャットダウン	FUN		IPC_Shutdown(EN);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			

機能

産業用 PC のシステムでのシャットダウン処理を開始し、完了後に Windows 側にシャットダウン要求を通知します。

本命令は、UPS からの一時電断信号を検知した場合や、産業用 PC の電源ボタンを押した場合に使用します。
本命令を実行すると、実行中の POU を最後まで実行し、ユーザプログラムは停止します。命令を実行する前に、コントローラ側のシャットダウン処理を行ってください。

コントローラ側のシャットダウン処理とは、たとえば、以下のようなものがあります。
- データ（変数）の保存
- モーション制御軸の停止、定位置への移動
- 上位管理システムへの停止の通知

UPS からの一時電断信号を検知していない場合や、産業用 PC の電源ボタンを押していない場合でも、本命令を使用して、シャットダウンすることができます。

プロジェクト内に本命令が複数あった場合は、最初に命令を実行したときに、シャットダウン処理を開始します。

使用上の注意

本命令によるプログラムミスで、繰り返しシャットダウンする状態となる場合は、セーフモードで起動し、電源 ON 時に本命令が実行されないようにユーザプログラムを変更してください。

参考

本命令をシミュレータで動作させた場合、命令は無視されます。「EN」を TRUE にしても、運転は継続されます。

関連するシステム定義変数

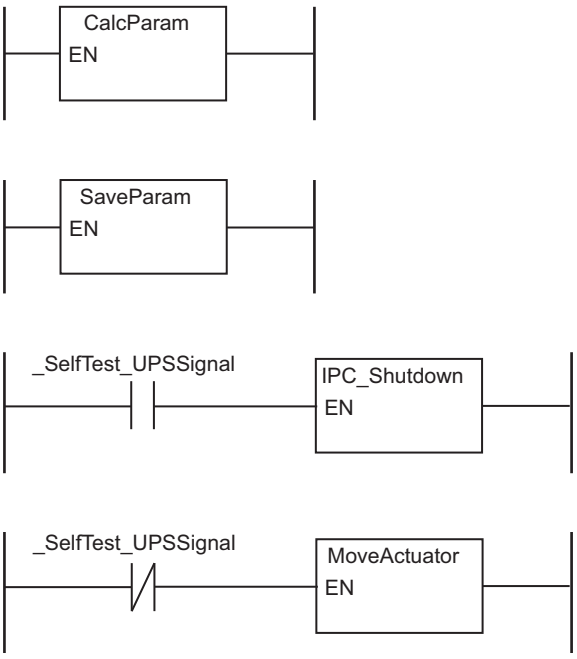
変数名	名称	データ型	有効範囲	内容
_SelfTest_UPSSignal	UPS 信号検知フラグ	BOOL	FALSE, TRUE	UPS からの一時電断信号を検知した場合に TRUE となります。
_RequestShutdown	シャットダウン要求フラグ	BOOL	FALSE, TRUE	稼働中に電源ボタンを押した場合に TRUE となります。

サンプルプログラム

毎周期、パラメータを計算し、それに応じてアクチュエータを動作させます。
UPS からの一時電断信号を検知すると、計算したパラメータが保存され、オペレーティングシステム(Windows)がシャットダウンします。
UPS からの一時電断信号を検知したとき、MoveActuator 命令が動作しないように、「_SelfTest_UPSSignal」の B 接点が挿入されています。

LD

外部変数	名称	データ型	コンスタント	コメント
	_SelfTest_UPSSignal	BOOL	☑	UPS 信号検知フラグ



CalcParam 命令　：ユーザ POU。パラメータを演算します。
SaveParam 命令　：ユーザ POU。パラメータを保存します。
MoveActuator 命令：計算したパラメータに基づいて、アクチュエータを動作させます。

ST

外部変数	名称	データ型	コンスタント	コメント
	_SelfTest_UPSSignal	BOOL	☑	UPS 信号検知フラグ

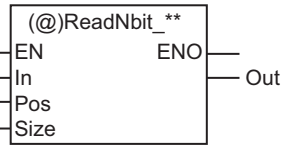
```
CalcParam();
SaveParam();
IF_SelfTest_UPSSignal THEN
  IPC_Shutdown();
ELSE
  MoveActuator();
End_IF;
```

その他の命令

命令	名称	ページ
ReadNbit_**	N ビット読出グループ	P.2-1344
WriteNbit_**	N ビット書込グループ	P.2-1346
ChkRange	範囲型変数チェック	P.2-1348
GetMyTaskStatus	自タスク状態読出	P.2-1350
GetMyTaskInterval	自タスク設定周期読出	P.2-1353
Task_IsActive	タスク実行中判定	P.2-1355
Lock / Unlock	タスク間排他ロック／タスク間排他ロック解除	P.2-1357
ActEventTask	イベントタスク起動	P.2-1363
Get**Clk	クロックパルス取得グループ	P.2-1370
Get**Cnt	加算フリーランニングカウンタ取得グループ	P.2-1371

ReadNbit_**

ビット列内の、複数ビットを読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ReadNbit_**	N ビット読出グループ	FUN	 ただし、“**”はビット列のデータ型名	Out:=ReadNbit_**(In, Pos, Size); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	読出元	入力	読出元のビット列	データ型に従う	−	0
Pos	読出位置		読み出すビット位置	0～「In」のビット数-1		
Size	読出サイズ		読み出すビット数	0～「In」のビット数		1
Out	読出結果	出力	読出結果	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
Out	「In」と同じデータ型																			

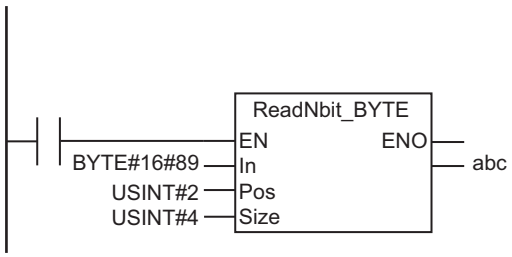
機能

読出元ビット列「In」の、読出位置「Pos」を含む上位「Size」ビットの値を、読出結果「Out」に代入します。

命令名称は、「In」、「Out」のデータ型によって、異なります。たとえば、「In」、「Out」が WORD 型の場合、命令名称は ReadNbit_WORD となります。

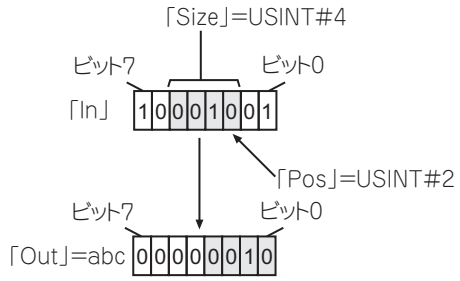
ReadNbit_BYTE 命令で、「In」=BYTE#16#89、「Pos」=USINT#2、「Size」=USINT#4 の場合の例を、以下に示します。

LD



ST

```
abc:=ReadNbit_BYTE(BYTE#16#89, USINT#2,
                   USINT#4);
```



参考

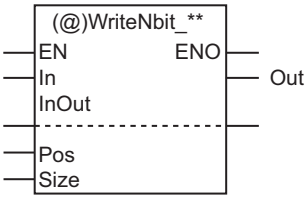
ビット列に複数ビットを書き込む場合は、「WriteNbit_** (P.2-1346)」命令を使用してください。

使用上の注意

- 「In」と「Out」のデータ型は、同じにしてください。
- 「Size」の値が 0 の場合、「Out」の値は 16#0 になります。
- つぎの場合、異常となります。「ENO」は FALSE となり、「Out」は変化しません。
  a)「Size」の値が有効範囲外の場合。
  b)「Pos」の値が有効範囲外の場合。
  c)「In」が、「Pos」の値で示した位置から「Size」の値で示した数だけのビット列を持たない場合。

WriteNbit_**

ビット列内に、複数ビットを書き込みます。

命令	名称	FB/FUN	グラフィック表現	ST 表現
WriteNbit_**	N ビット書込グループ	FUN	 ただし、“**”はビット列のデータ型名	WriteNbit_**(In, InOut, Pos, Size); ただし、“**”はビット列のデータ型名

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	読出元	入力	「InOut」に書き込む値を読み出すビット列	データ型に従う	−	0
Pos	書込位置		書き込むビット位置	0〜「InOut」のビット数-1		
Size	書込サイズ		書き込むビット数	0〜「In」のビット数		1
InOut	書込対象	入出力	読出結果	データ型に従う	−	−
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
InOut	「In」と同じデータ型																			
Out	○																			

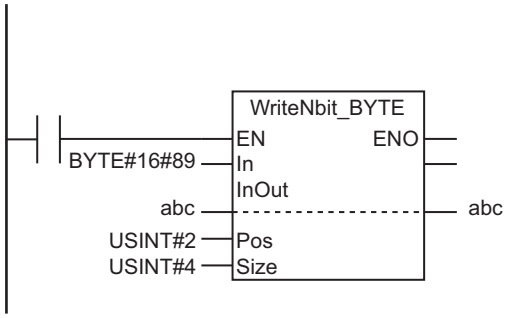
機能

まず、読出元「In」の下位「Size」ビットを読み出します。その後、読み出した値を、書込対象「InOut」の書込位置「Pos」に上書します。

命令名称は、「In」、「Out」のデータ型によって、異なります。たとえば、「In」、「Out」が WORD 型の場合、命令名称は WriteNbit_WORD となります。

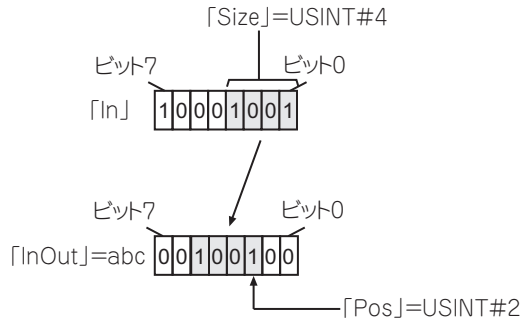
WriteNbit_BYTE 命令で、「In」=BYTE#16#89、「Pos」=USINT#2、「Size」=USINT#4 の場合の例を、以下に示します。

LD



ST

```
WriteNbit_BYTE(BYTE#16#89, abc,
                USINT#2, USINT#4);
```



参考

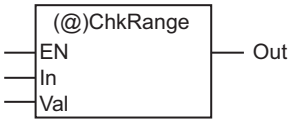
ビット列の複数ビットを読み出す場合は、「ReadNbit_** (P.2-1344)」命令を使用してください。

使用上の注意

- 「In」と「InOut」のデータ型は、同じにしてください。
- 「Size」の値が 0 の場合、「InOut」の値は変化しません。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。
- つぎの場合、異常となります。「ENO」は FALSE となり、「InOut」は変化しません。
  a)「Size」の値が有効範囲外の場合。
  b)「Pos」の値が有効範囲外の場合。
  c)「InOut」が、「Pos」の値で示した位置から「Size」の値で示した数だけのビット列を持たない場合。

ChkRange

変数の値が、範囲型指定の有効範囲内にあるかどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ChkRange	範囲型変数チェック	FUN		Out:=ChkRange(In, Val);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
In	チェック対象	入力	チェック対象	データ型に従う	−	*1
Val	範囲指定変数		範囲指定変数	範囲指定に従う		
Out	チェック結果	出力	チェック結果	データ型に従う	−	−

*1. 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
Val	範囲指定　元となる基本データ型は「In」と同じ																			
Out	○																			

機能

チェック対象「In」の値が、範囲指定変数「Val」の有効範囲内にあるかどうかを判定します。
有効範囲内にあればチェック結果「Out」は TRUE、なければ FALSE になります。

参考

範囲指定は、整数型(USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT)の変数に対して、定義することができます。

使用上の注意

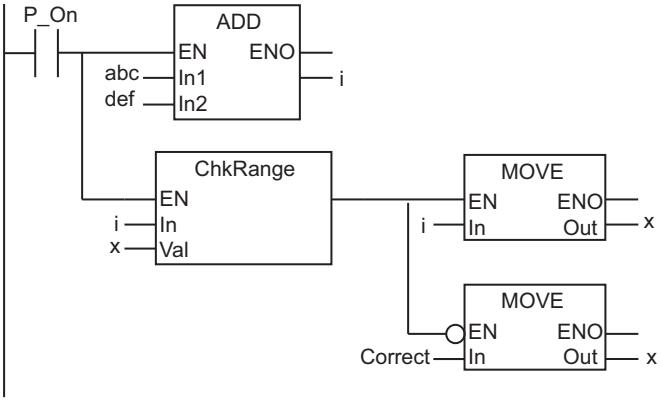
- 「In」が範囲指定変数ではない場合、「Out」の値は TRUE になります。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Out」の値は FALSE になります。

サンプルプログラム

加算結果「i」が、範囲指定変数「x」の有効範囲内（10～99）にあるかどうかを判定します。
もし、有効範囲内にあれば、「i」の値を「x」に代入します。有効範囲内になければ、変数「Correct」の値を「x」に代入します。

LD

名称	データ型	初期値
i	INT	0
abc	INT	0
def	INT	0
x	INT(10..99)	10
Correct	INT	0



ST

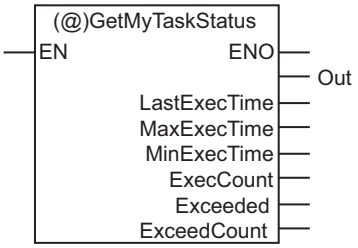
名称	データ型	初期値
i	INT	0
abc	INT	0
def	INT	0
Chk	BOOL	FALSE
x	INT(10..99)	10
Correct	INT	0

```
i := abc+def;
Chk:=ChkRange(i, x);    // 範囲型変数チェック

IF (Chk=TRUE) THEN
    x := i;             // i の値が範囲内なら i を x に代入
ELSE
    x := Correct;       // i の値が範囲内でなければ Correct を x に代入
END_IF;
```

GetMyTaskStatus

自タスクのタスク状態を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetMyTaskStatus	自タスク状態読出	FUN		GetMyTaskStatus(LastExecTime, MaxExecTime, MinExecTime, ExecCount, Exceeded, ExceedCount);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	戻り値	出力	常に TRUE	TRUE だけ	－	－
LastExecTime	前回タスク実行時間		前回の自タスク実行時間	データ型に従う *1	ns	
MaxExecTime	タスク実行時間最大値		自タスク実行時間の最大値			
MinExecTime	タスク実行時間最小値		自タスク実行時間の最小値			
ExecCount	タスク実行回数		自タスク実行回数	データ型に従う	－	
Exceeded	タスク周期超過フラグ		TRUE: 前回の自タスクの実行が、タスク周期内に完了しなかった。 FALSE: 前回の自タスクの実行が、タスク周期内に完了した。			
ExceedCount	タスク周期超過回数		自タスクが、タスク周期を超過した回数			

*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out						○														
LastExecTime																○				
MaxExecTime																○				
MinExecTime																○				
ExecCount								○												
Exceeded	○																			
ExceedCount								○												

機能

自タスクの、タスク状態を取得します。
タスク状態とは、前回タスク実行時間「LastExecTime」、タスク実行時間最大値「MaxExecTime」、タスク実行時間最小値「MinExecTime」、タスク実行回数「ExecCount」、タスク周期超過フラグ「Exceeded」、タスク周期超過回数「ExceedCount」です。

参考

「MaxExecTime」、「MinExecTime」、「ExecCount」、「ExceedCount」の値がリセットされるのは、つぎのタイミングです。

- 運転開始時
- Sysmac Studio のタスク実行時間モニタで、リセット実行時

使用上の注意

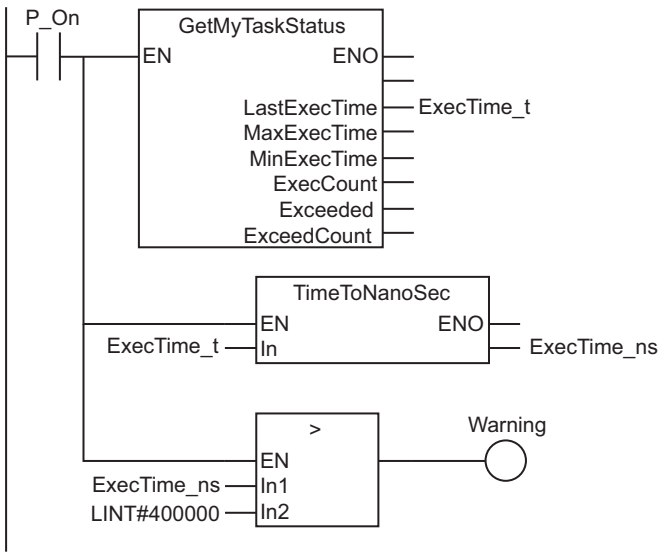
- 「ExecCount」および「ExceedCount」の値は、UDINT 型の最大値(4,294,967,295)を超えると、0 に戻ります。
- 本命令を、ST プログラムで使用する場合、戻り値「Out」は使用しません。

サンプルプログラム

自タスクのタスク状態を取得します。
もし、前回タスク実行時間が 400μs(400000ns)を超えていたら、変数「Warning」の値を TRUE にします。

LD

名称	データ型	初期値	コメント
ExecTime_t	TIME	T#0s	前タスク実行時間（TIME 型）
ExecTime_ns	LINT	0	前タスク実行時間（LINT 型 ナノ秒数）
Warning	BOOL	FALSE	ワーニング



ST

名称	データ型	初期値	コメント
ExecTime_t	TIME	T#0s	前タスク実行時間（TIME 型）
ExecTime_ns	LINT	0	前タスク実行時間（LINT 型 ナノ秒数）
Warning	BOOL	FALSE	ワーニング

```
GetMyTaskStatus(LastExecTime=>ExecTime_t);  // 前回のタスク周期を取得
ExecTime_ns:=TimeToNanoSec(ExecTime_t);  // 前回のタスク周期を TIME 型からナノ秒数に変換
IF (ExecTime_ns>DINT#400000) THEN  // 前回のタスク周期が 400,000ns を超えているか
    Warning:=TRUE;  // 変数 Warning に TRUE を代入
ELSE
    Warning:=FALSE;
END_IF;
```

GetMyTaskInterval

自タスクのタスク周期を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
GetMyTaskInterval	自タスク設定周期読出	FUN	(@)GetMyTaskInterval EN ENO Out	Out:=GetMyTaskInterval();

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	タスク周期	出力	自タスクのタスク周期	データ型に従う *1	ms	−

*1. 負数は含みません。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																○				

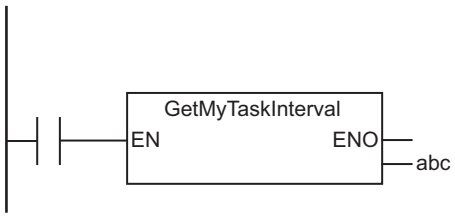
機能

本命令を実行するタスクが、プライマリ定周期タスクまたは定周期タスクの場合、自タスクのタスク周期が、タスク周期「Out」に格納されます。

本命令を実行するタスクがイベントタスクの場合、「Out」の値は T#0s になります。

記述例を、以下に示します。自タスクのタスク周期が 1ms の場合、「abc」の値が T#1ms になります。

LD



ST

```
abc:=GetMyTaskInterval();
```

サンプルプログラム

運転開始後、最初に本プログラムを実行するときに、自タスクのタスク周期を読み出します。その後、読み出したタスク周期を、TIME 型から LREAL 型の ms 単位に変換します。
タスク周期ごとの軸の目標位置を算出するためなどに使用するサンプルプログラムです。

TIME 型から LREAL 型の ms 単位に変換する手順は、以下のとおりです。

1 GerMyTaskInterval 命令を使用して、TIME 型のタスク周期を読み出します。

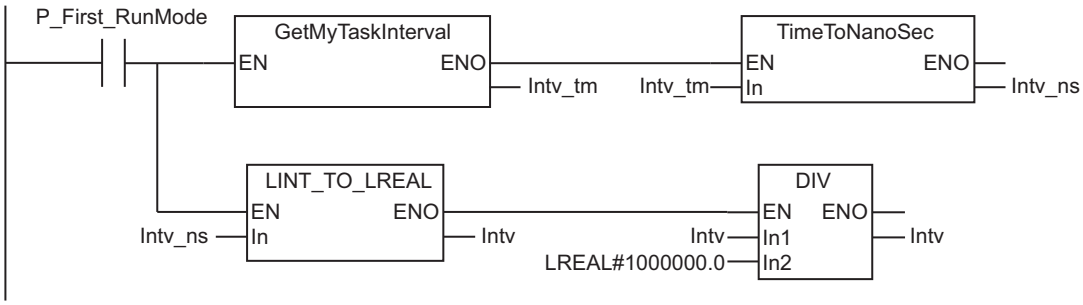
2 TimeToNanoSec 命令を使用して、TIME 型から LINT 型の ns 単位に変換します。

3 LINT_TO_LREAL 命令を使用して、LINT 型の ns 単位から、LREAL 型の ns 単位に変換します。

4 DIV 命令を使用して、3 の結果を 1,000,000 で除算し、ms 単位に変換します。

LD

名称	データ型	初期値	コメント
Intv_tm	TIME	T#0s	TIME 型で表現したタスク周期
Intv_ns	LINT	0	LINT 型の ns 単位で表現したタスク周期
Intv	LREAL	0	LREAL 型の ms 単位で表現したタスク周期



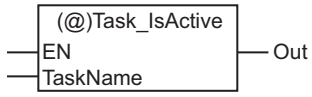
ST

名称	データ型	初期値	コメント
Intv	LREAL	0	LREAL 型の ms 単位で表現したタスク周期

```
IF P_First_RunMode = TRUE THEN
   Intv := LINT_TO_LREAL(TimeToNanoSec(GetMyTaskInterval()))/1000000;
END_IF;
```

Task_IsActive

指定したタスクが、実行中かどうかを判定します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Task_IsActive	タスク実行中判定	FUN		Out:=Task_IsActive(TaskName);

変数

	名称	入／出力	内容	有効範囲	単位	初期値
TaskName	タスク名	入力	タスク名	最大 64 バイト（半角英数字 63 文字+終端 NULL 文字）	–	"
Out	実行中判定	出力	TRUE: 実行中または待機状態 FALSE: 実行中ではない	データ型に従う	–	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TaskName																				○
Out	○																			

機能

タスク名「TaskName」で指定したタスクが、実行中または待機状態かどうかを判定します。
“待機状態”とは、タスクが起動後、それよりも優先順位が高いタスクが起動したために、処理が中断している状態のことです。
実行中または待機状態ならば、実行中判定「Out」の値が TRUE になります。非実行中ならば、「Out」の値が FALSE になります。

使用上の注意

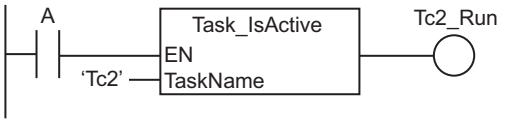
- 「TaskName」には、文字列を代入した変数は指定できません。文字列を直接指定してください。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生したら、「Out」の値は FALSE になります。
- つぎの場合、異常となります。「Out」は変化しません。
  a)「TaskName」で指定したタスクが、存在しない場合。

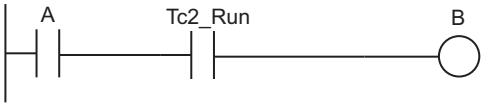
サンプルプログラム

変数「A」の値が TRUE になったら、周期タスク Tc2 が起動中かどうかを調べます。起動中であれば、変数「B」の値を TRUE にします。

LD

名称	データ型	初期値	コメント
A	BOOL	FALSE	
B	BOOL	FALSE	
Tc2_Run	BOOL	FALSE	タスク Tc2 の実行状態





ST

名称	データ型	初期値	コメント
A	BOOL	FALSE	
B	BOOL	FALSE	
Tc2_Run	BOOL	FALSE	タスク Tc2 の実行状態

```
IF (A=TRUE) THEN
    // タスク実行中判定
    Tc2_Run:=Task_IsActive('Tc2');
    // Tc2 が実行中なら変数 B に TRUE を代入
    IF (Tc2_Run=TRUE) THEN
        B := TRUE;
    END_IF;

END_IF;
```

Lock / Unlock

Lock　：タスク間排他ロックを開始します。他のタスクの同じロック番号の区間は、実行できなくなります。
Unlock ：タスク間排他ロックを終了します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Lock	タスク間排他ロック	FUN	(@)Lock EN ENO Index Out	Lock(Index);
Unlock	タスク間排他ロック解除	FUN	(@)Unlock EN ENO Index Out	Unlock(Index);

変数

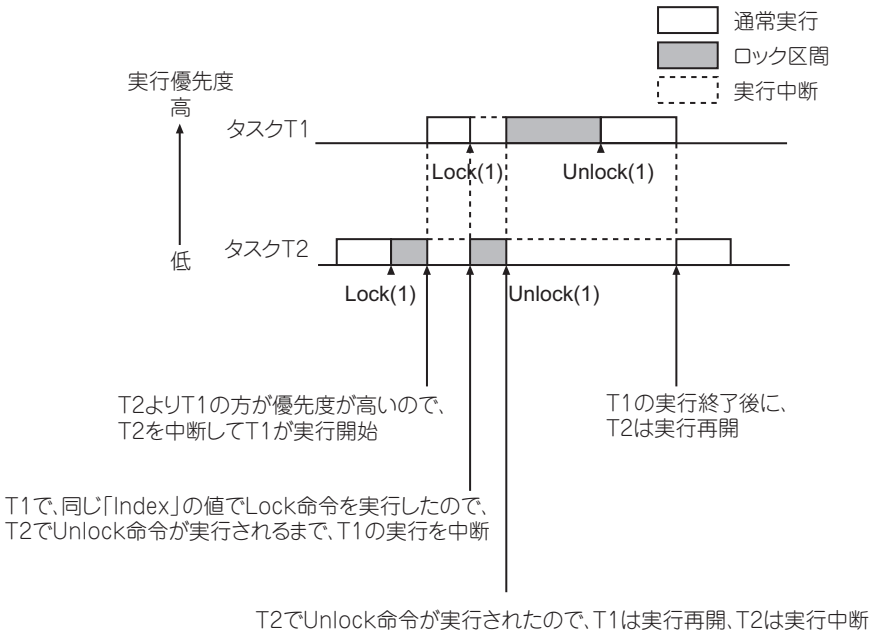
	名称	入／出力	内容	有効範囲	単位	初期値
Index	ロック番号	入力	ロック番号	データ型に従う	−	0
Out	戻り値	出力	常に TRUE	TRUE だけ	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Index						○														
Out	○																			

機能

Lock 命令から Unlock 命令までの区間（ロック区間）を実行中に、他のタスクの同じロック番号のロック区間を実行できなくします。
ロック番号は「Index」で指定します。

動作例を、以下に示します。
タスク T1、T2 は、どちらも「Index」=1 のロック区間を持つとします。先に T2 の Lock 命令が実行されると、T2 で Unlock 命令が実行されるまで、T1 のロック区間は実行できません。



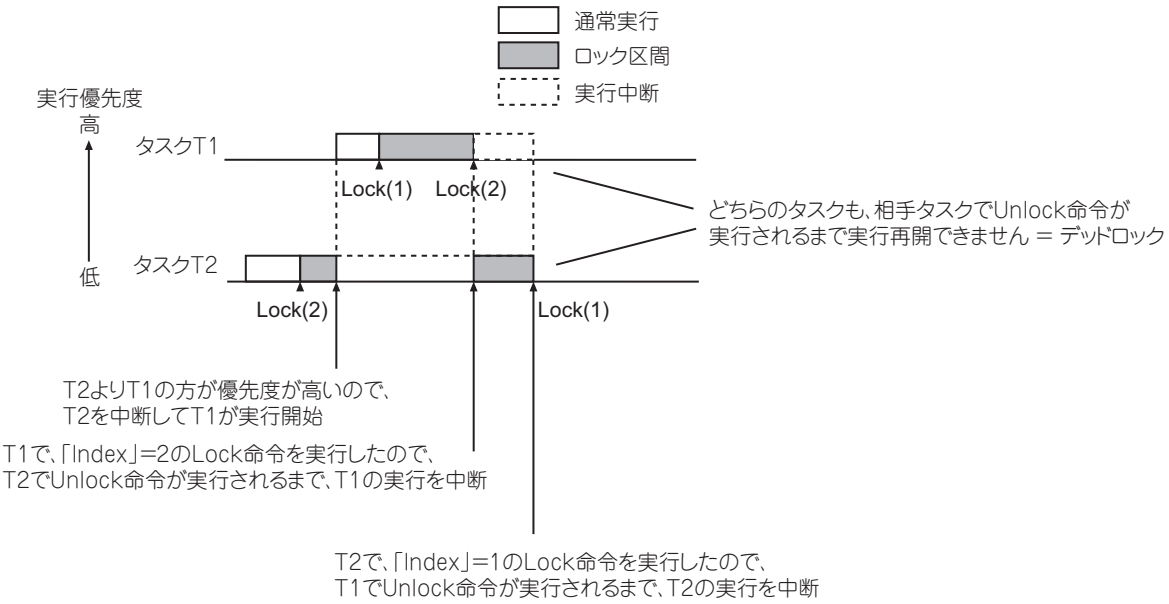
「Index」の値が異なるロック区間は、互いに影響を与えません。

参考

- 本命令は、複数のタスクから同じデータを読み書きする可能性がある場合などに使用します。
  あるタスクがデータを読み書きしている間は、他のタスクからそのデータを読み書きできないようにします。
- 「Index」の値が異なっていれば、複数の Lock 命令と Unlock 命令の組が、同一の POU に存在していてもかまいません。また、それらの配置が入れ子構造になっていても、かまいません。

使用上の注意

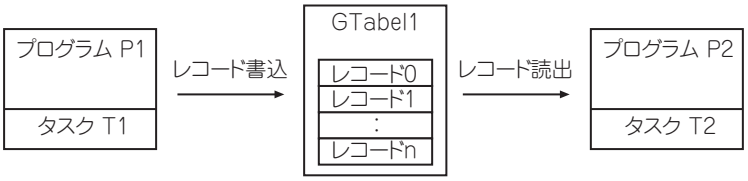
- ロック区間は、必要最小限の長さにしてください。ロック区間が長いと、タスクの実行周期を超過する可能性があります。
- Lock 命令と Unlock 命令は、同一 POU の同一セクション内で、必ず対にして使用してください。
- 同時に設定できるロック区間は、最大 16777215 個です。
- 複数のタスクで、複数の Lock 命令を使用する場合、それらの配置によっては、デッドロックになることがあります。デッドロックになった場合は、タスク実行タイムアウト異常が発生し、全停止します。デッドロックになる場合の例を、以下に示します。



・ つぎの場合、異常となります。「Out」は変化しません。
a）ロック区間を、同時に 16777215 個を超えて設定しようとした場合。

サンプルプログラム

タスク T1 のプログラム P1 と、タスク T2 のプログラム P2 とで、同じグローバル変数「GTable1」にアクセスします。
書込要求「WriteReq」の値が TRUE になると、P1 はレコード配列 GTable1.Record[]に 1 レコードの書込を行い、インデックス GTable1.Index をインクリメントします。
読出要求「ReadReq」の値が TRUE になると、P2 は GTable1.Index をデクリメントし、GTable1.Record[]から 1 レコードの読出を行います。
Lock 命令を使用して、読出と書込が同時に起こらないようにします。



グローバル変数「GTable」の定義

● データ型

名称	データ型	コメント
USERTABLE	STRUCT	レコード格納構造体
Index	INT	インデックス
Record	ARRAY[0..99] OF LREAL	レコード配列

● グローバル変数

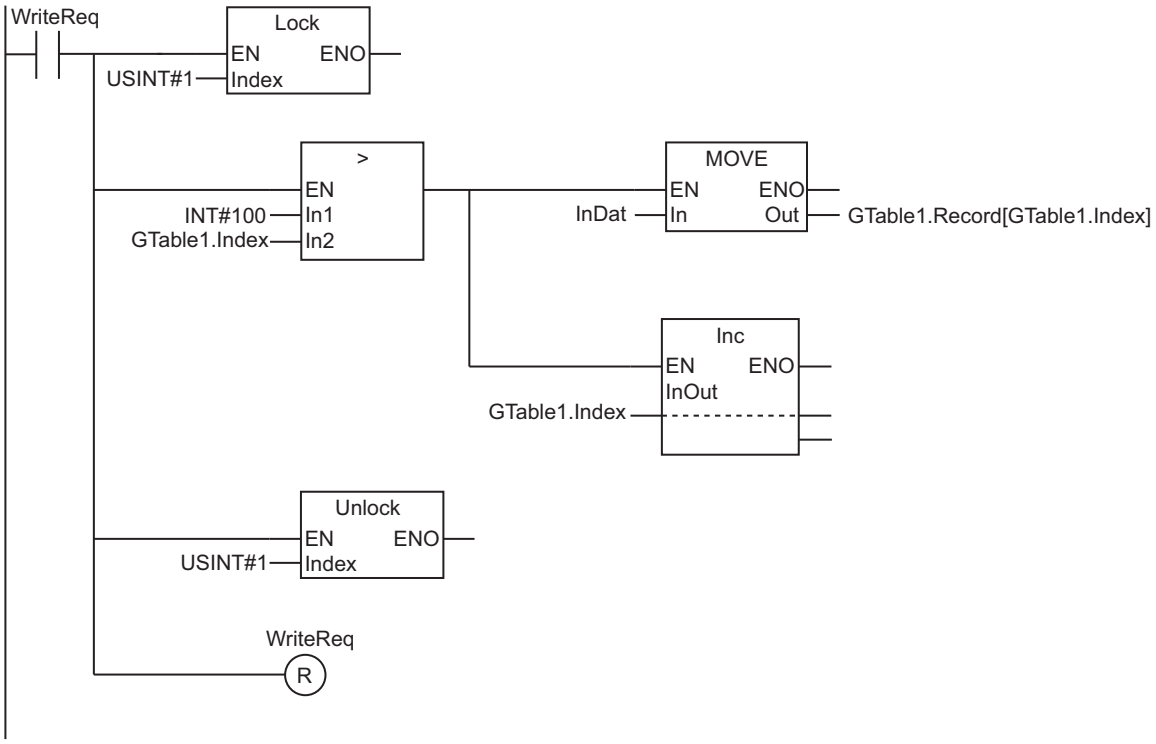
名称	データ型	初期値	コメント
GTable1	USERTABLE	(Index:=0, Record:=[100(0.0)])	レコード格納構造体

プログラム P1

● LD

内部変数	名称	データ型	初期値	コメント
	WriteReq	BOOL	FALSE	書込要求
	InDat	LREAL	0.0	書込データ

外部変数	名称	データ型	コメント
	GTable1	USERTABLE	レコード格納構造体



● ST

内部変数	名称	データ型	初期値	コメント
	WriteReq	BOOL	FALSE	書込要求
	InDat	LREAL	0.0	書込データ

外部変数	名称	データ型	コメント
	GTable1	USERTABLE	レコード格納構造体

```
// 書込要求検出
IF (WriteReq=TRUE) THEN

    // Lock 命令実行
    Lock(USINT#1);

    IF (INT#100>GTable1.Index) THEN
        GTable1.Record[GTable1.Index]:=InDat;
        GTable1.Index                :=GTable1.Index+INT#1;
```

```
    END_IF;

    // Unlock 命令実行
    Unlock(USINT#1);
    WriteReq:=FALSE;

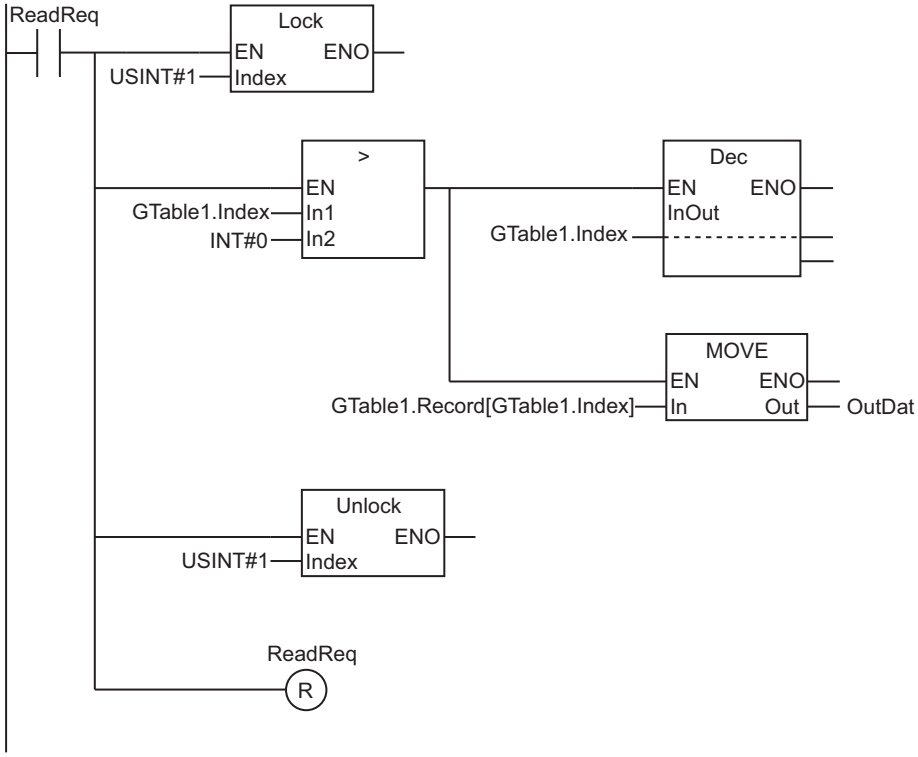
END_IF;
```


プログラム P2

● LD

内部変数	名称	データ型	初期値	コメント
	ReadReq	BOOL	FALSE	読出要求
	OutDat	LREAL	0.0	読出データ

外部変数	名称	データ型	コメント
	GTable1	USERTABLE	レコード格納構造体



● ST

内部変数	名称	データ型	初期値	コメント
	ReadReq	BOOL	FALSE	読出要求
	OutDat	LREAL	0.0	読出データ

外部変数	名称	データ型	コメント
	GTable1	USERTABLE	レコード格納構造体

```
// 読出要求検出
IF (ReadReq=TRUE) THEN

    // Lock 命令実行
    Lock(USINT#1);

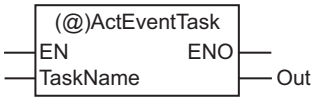
    IF (GTable1.Index>INT#0) THEN
        GTable1.Index:=GTable1.Index-INT#1;
        OutDat       :=GTable1.Record[GTable1.Index];
    END_IF;

    // Unlock 命令実行
    Unlock(USINT#1);
    ReadReq:=FALSE;

END_IF;
```

ActEventTask

イベントタスクを起動します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ActEventTask	イベントタスク起動	FUN		ActEventTask(TaskName);

変数

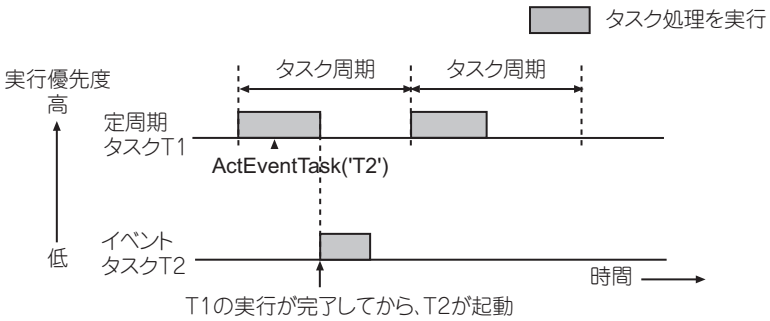
	名称	入／出力	内容	有効範囲	単位	初期値
TaskName	タスク名	入力	起動するイベントタスクのタスク名	最大 64 バイト（半角英数字 63 文字+ 終端 NULL 文字）	–	''
Out	戻り値	出力	TRUE: 命令を実行、かつ異常でない FALSE: 命令を実行しない、または異常である	データ型に従う	–	–

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TaskName																				○
Out	○																			

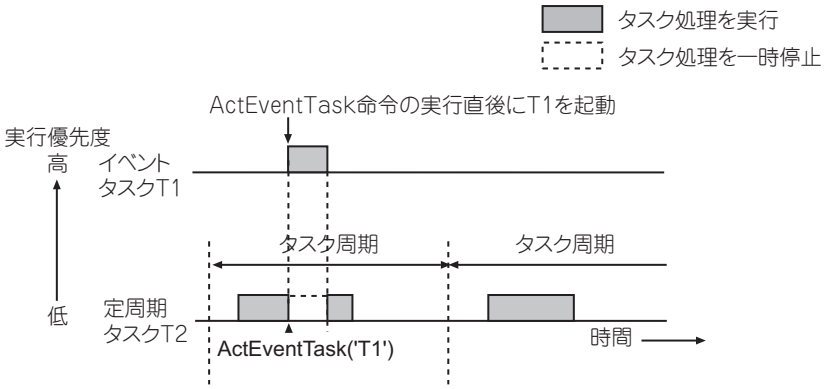
機能

タスク名「TaskName」のイベントタスクを 1 回起動します。
イベントタスクは、タスクの実行優先度にしたがって動作します。

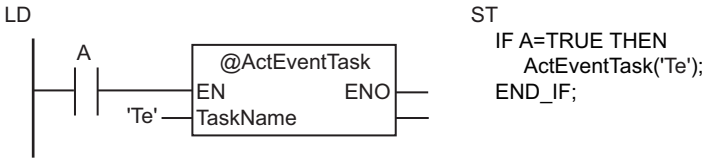
本命令を実行するタスクよりも、実行優先度が低いイベントタスクを起動する場合、本命令を実行するタスクの実行が完了してから、イベントタスクを起動します。
たとえば、定周期タスク T1 よりイベントタスク T2 の実行優先度が低いとします。
T1 で、T2 を指定して ActEventTask 命令を実行すると、T1 の実行が完了してから T2 を起動します。



本命令を実行するタスクよりも、実行優先度が高いイベントタスクを起動する場合、本命令を実行するタスクの実行を一時停止して、イベントタスクを起動します。
たとえば、イベントタスク T1 より定周期タスク T2 の実行優先度が低いとします。
T2 で、T1 を指定して ActEventTask 命令を実行すると、T2 実行を一時停止して、T1 を起動します。

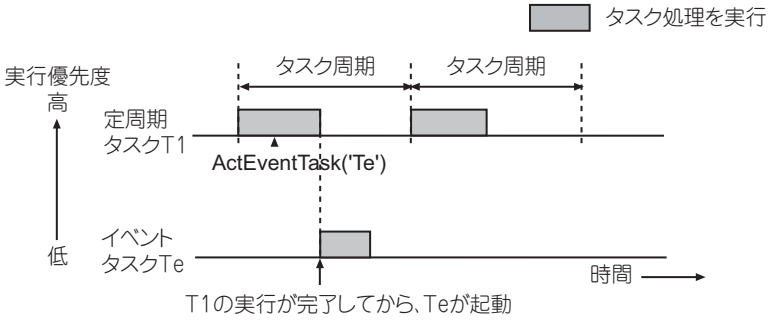


記述例を以下に示します。変数「A」の値が TRUE の場合に、イベントタスク'Te'を起動します。




```
ST
  IF A=TRUE THEN
    ActEventTask('Te');
  END_IF;
```

この記述があるプログラムが定周期タスク T1 に割り付けられており、T1 より Te の実行優先度が低いとします。この場合、T1 で本命令が実行されると、T1 の実行が完了してから Te を起動します。



関連するシステム定義変数

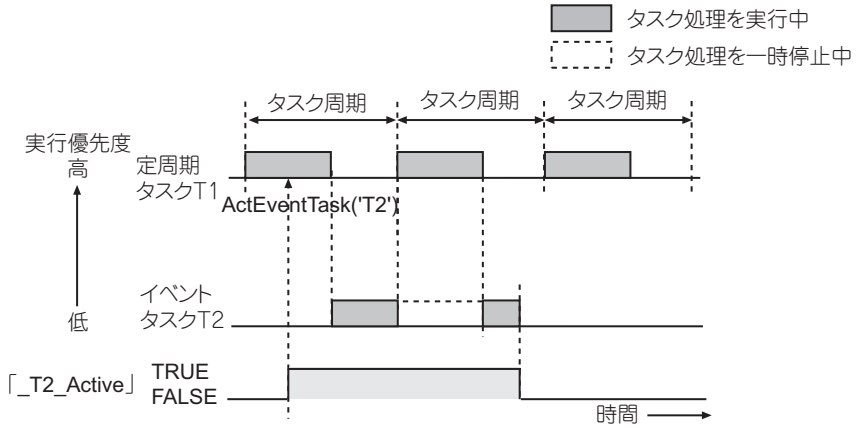
変数名	名称	データ型	内容
_**_Active *1	タスク実行中フラグ	BOOL	タスクの実行状態 *2 TRUE: 実行中 FALSE: 停止中

*1. **は、タスク名です。
*2. 詳細は、『NY シリーズ 産業用パネル型 PC／産業用ボックス型 PC ユーザーズマニュアル ソフトウェア編 (SBCA-436)』を参照してください。

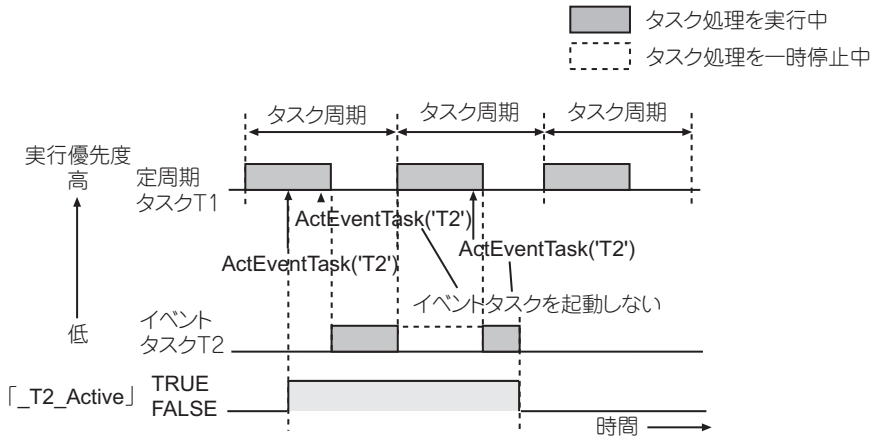
参考

システム定義変数「_**_Active」の動作

・本命令を実行すると、指定されたイベントタスクのシステム定義変数「_**_Active」が TRUE になります。
イベントタスクの実行が完了すると FALSE になります。
たとえば、定周期タスク T1 よりイベントタスク T2 の実行優先度が低いとします。
T1 で、T2 を指定して ActEventTask 命令を実行すると、システム定義変数「_T2_Active」は、下図のように変化します。



・イベントタスクのシステム定義変数「_**_Active」が TRUE の間に、本命令を実行してもイベントタスクは起動しません。

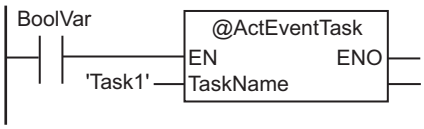


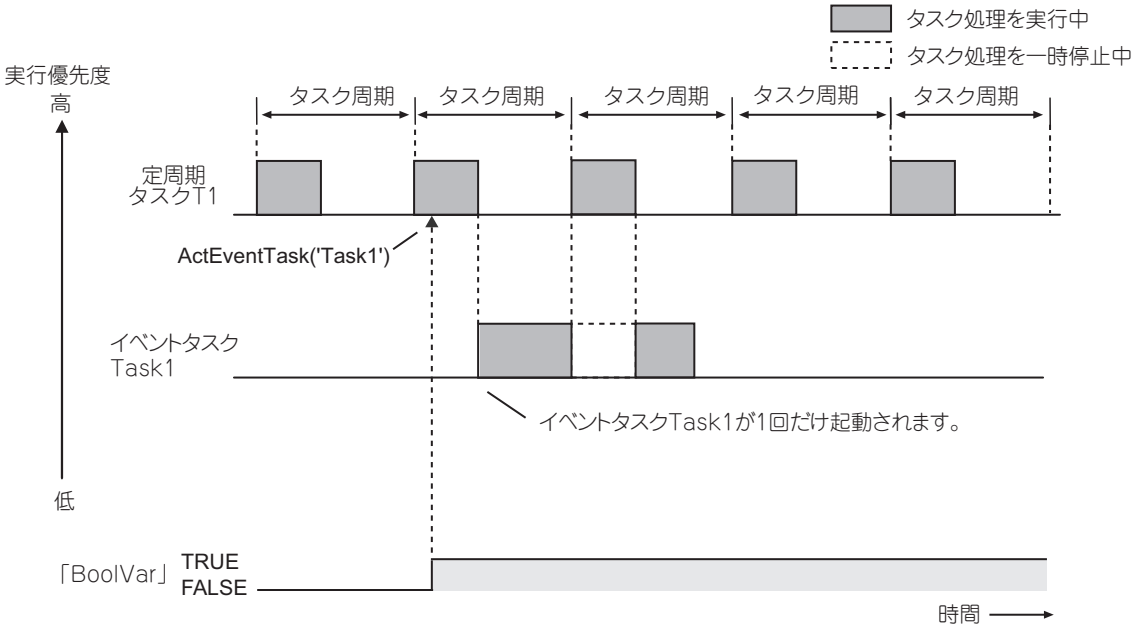
イベントタスクを１回だけ起動する場合と、繰り返し起動する場合

指定した変数の値が変化したときに 1 回だけイベントタスクを起動する場合と、変数が指定した値で、ある間繰り返しイベントタスクを起動する場合とでは、ユーザプログラムを以下のように書き換えてください。

● 変数の値が変化したときにイベントタスクを１回だけ起動する場合

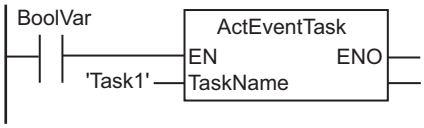
下図のように、本命令に入力立ち上がり微分の動作オプションを付けると、BOOL 型変数「BoolVar」の値が FALSE から TRUE に変化した際に、イベントタスク'Task1'が 1 回だけ起動されます。

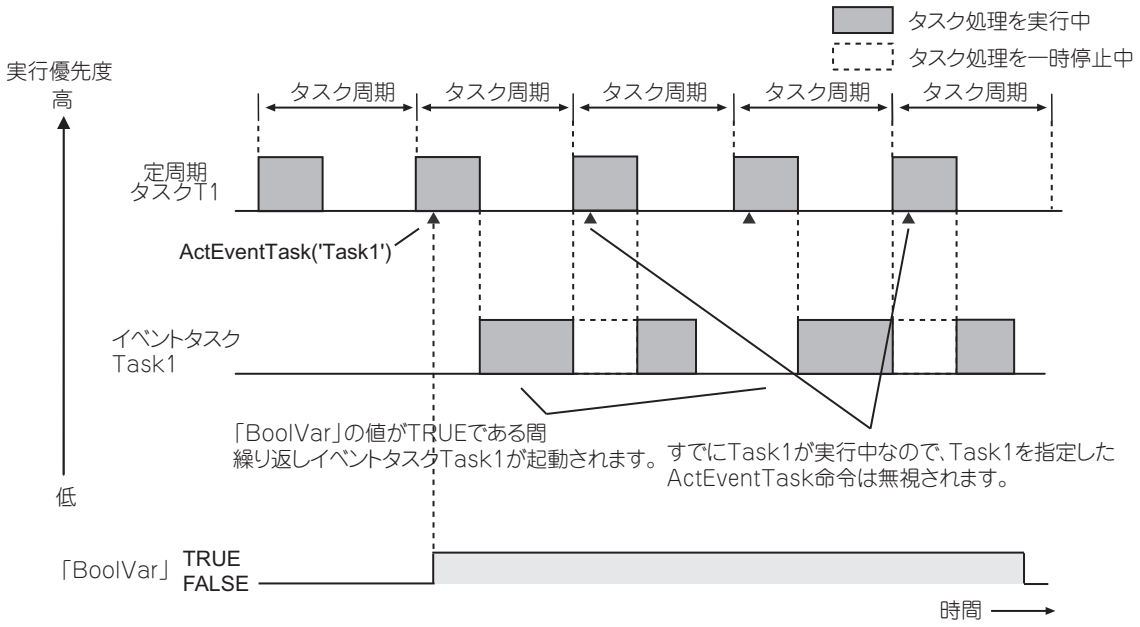




● 変数が指定した値で、ある間繰り返しイベントタスクを起動する場合

下図のように、本命令に入力立ち上がり微分の動作オプションを付けないと、BOOL 型変数「BoolVar」の値が TRUE である間、繰り返しイベントタスク'Task1'が起動されます。ただし、Task1 が実行中に、Task1 を指定した本命令を実行しても、無視されます。





使用上の注意

- 命令実行時間を短縮したい場合は、イベントタスクを起動させたいときだけ本命令を実行してください。
  システム定義変数「_**_Active」が TRUE の間に本命令を実行する場合、イベントタスクを起動しなくても、命令実行時間が必要になります。
- 「TaskName」のイベントタスクが存在しない場合、異常となります。「ENO」は FALSE となります。

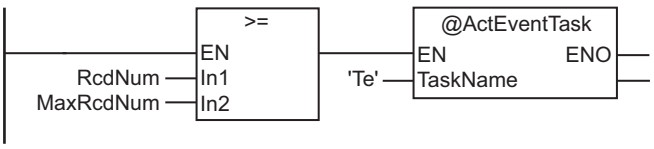
サンプルプログラム

変数の値が指定した条件に一致した場合に、イベントタスクを起動する例

変数「RcdNum」が、変数「MaxRcdNum」未満から「MaxRcdNum」以上に変化した場合に、イベントタスク'Te'を 1 回だけ起動します。

● **LD**

名称	データ型	初期値
RcdNum	INT	0
MaxRcdNum	INT	100



● **ST**

名称	データ型	初期値
RcdNum	INT	0

名称	データ型	初期値
MaxRcdNum	INT	100
met	BOOL	FALSE

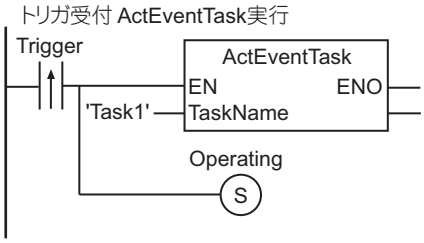
```
IF (RcdNum>=MaxRcdNum) THEN
   IF (met=FALSE) THEN
      ActEventTask('Te');
      met:=TRUE;
   END_IF;
ELSE
   met:=FALSE;
END_IF;
```

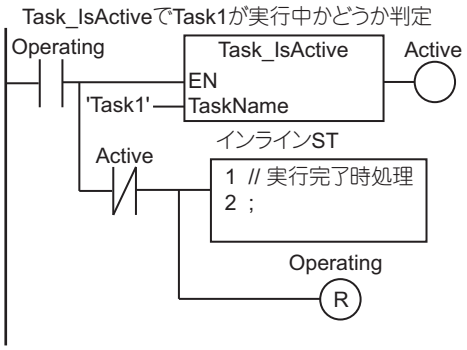
イベントタスクの実行完了を確認してから、つぎの処理を行う例

変数「Trigger」の立ち上がり時に、イベントタスク'Task1'を起動します。
Task_IsActive 命令を使用して、Task1 の実行完了を確認します。

● LD

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
Operating	BOOL	FALSE	イベントタスク実行確認中
Active	BOOL	FALSE	イベントタスク実行中





● ST

名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
LastTrigger	BOOL	FALSE	前タスク周期の「Trigger」の値
Operating	BOOL	FALSE	イベントタスク実行確認中
Active	BOOL	FALSE	イベントタスク実行中

```
// Trigger の立ち上がりでシーケンス開始
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
   ActEventTask('Task1');        // イベントタスク Task1 起動
   Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Task1 が実行中かどうか判定
IF (Operating=TRUE) THEN
   Active:=Task_IsActive('Task1');

   IF (Active=FALSE) THEN        // Task1 実行完了
      Operating:=FALSE;
   END_IF;

END_IF;
```

Get**Clk

指定した周期の、クロックパルスを出力します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Get**Clk	クロックパルス取得グループ	FUN	Get**Clk EN — Out ただし、“**”は 100µs, 1ms, 10ms, 20ms, 100ms, 1s, 1min のいずれか	Out:=Get**Clk(); ただし、“**”は 100µs, 1ms, 10ms, 20ms, 100ms, 1s, 1min のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	クロックパルス	出力	クロックパルス	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	○																			

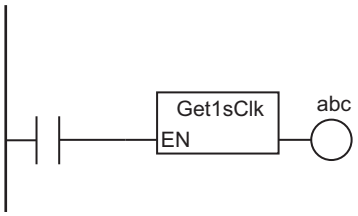
機能

指定した周期の、クロックパルスを出力します。
クロックパルスの周期は、100µs, 1ms, 10ms, 20ms, 100ms, 1s, 1min のいずれかです。

命令名称は、クロックパルスの周期によって、異なります。たとえば、クロックパルスの周期が 10ms の場合、命令名称は Get10msClk となります。

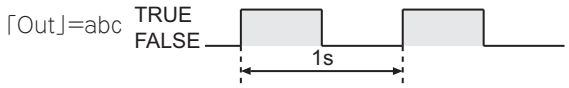
Get1sClk 命令の場合の例を、以下に示します。

LD



ST

```
abc:=Get1sClk();
```



使用上の注意

- 命令実行時、最初の「Out」の値が TRUE か FALSE かは不定です。
- 本命令をラダー図プログラムで使用した場合、本命令の回路の前段で異常が発生すると、「Out」の値は FALSE になります。

Get**Cnt

指定した周期の、フリーランニングカウンタの値を取得します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
Get**Cnt	加算フリーランニングカウンタ取得グループ	FUN	Get**Cnt EN ENO Out ただし、“**” は 100ns, 1µs, 1ms, 10ms, 100ms, 1s のいずれか	Out:=Get**Cnt(); ただし、“**” は 100ns, 1µs, 1ms, 10ms, 100ms, 1s のいずれか

変数

	名称	入／出力	内容	有効範囲	単位	初期値
Out	カウント値	出力	フリーランニングカウンタの値	データ型に従う	−	−

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out									O											

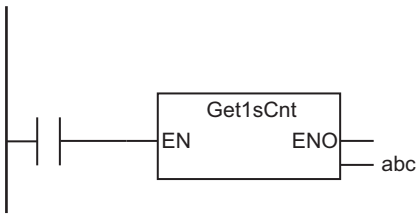
機能

指定した周期のフリーランニングカウンタの値を取得します。
フリーランニングカウンタとは、一定周期でカウントアップするカウンタのことで、「Out」が、カウント値です。カウンタの周期は、100ns, 1µs, 1ms, 10ms, 100ms, 1s のいずれかです。

命令名称は、カウンタの周期によって、異なります。たとえば、カウンタの周期が 10ms の場合、命令名称は Get10msCnt となります。

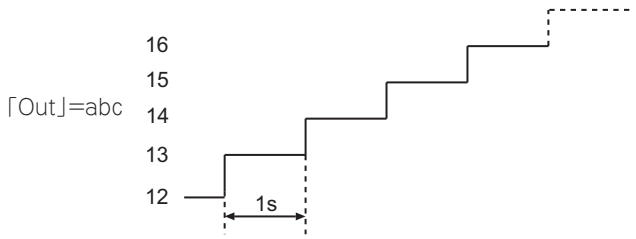
Get1sCnt 命令の場合の例を、以下に示します。

LD



ST

```
abc:=Get1sCnt();
```



使用上の注意

- フリーランニングカウンタは、電源 ON とともにカウントを開始します。そして、ULINT 型の有効範囲 18,446,744,073,709,551,615 を超えると、0 に戻ってカウントアップを続けます。
- 本命令は、フリーランニングカウンタ値を取得するだけであり、0 リセットするものではありません。
- 「Out」の値が、いくつから始まるかは不定です。必ず 0 から始まるわけではありません。

A

付録

A-1　ErrorID で確認できるエラーコード一覧 A-2
A-2　イベントタスクで使用できない命令 A-20
A-3　NX メッセージ通信異常に関連する命令 A-22
A-4　SDO アボートコード一覧 A-23
A-5　バージョン関連情報 A-24

A-1 ErrorID で確認できるエラーコード一覧

命令の実行時に発生する異常には、その内容に応じてエラーコードが割り当てられています。出力変数のエラーコード「ErrorID」を持つ命令を使用する場合、エラーコードを使って異常処理のプログラミングができます。

以下に、「ErrorID」を持つ命令と、その命令で発生する可能性があるエラーコードの一覧を示します。エラーコードの内容は、『NY シリーズ トラブルシューティングマニュアル(SBCA-438)』を参照してください。



参考

「ErrorID」を持たない命令の異常は、イベントログ機能でイベントとして確認できます。

種別	命令	名称	エラーコード	エラー名
アナログ制御命令	PIDAT	オートチューニング付き PID 演算	16#0400	入力値範囲外
			16#0401	数値関係不正
	PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	16#0400	入力値範囲外
			16#0401	数値関係不正
	AC_StepProgram	ステッププログラム	16#0400	入力値範囲外
システム制御命令	ResetPLCError	PLC 異常解除	−	−
	ResetMCError	モーション制御異常解除	−	−
	ResetECError	EtherCAT 異常解除	16#041A	命令多重起動
	RestartNXUnit	NX ユニットリスタート	16#0400	入力値範囲外
			16#0419	データ型不正
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
			16#2C05	NX メッセージネットワーク異常(EtherCAT)
			16#2C06	指定ユニット外部リスタート既実行
			16#2C07	命令対象外ユニット指定
	NX_ChangeWriteMode	NX ユニット書込モード変更	16#0400	入力値範囲外
			16#0419	データ型不正
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
			16#2C05	NX メッセージネットワーク異常(EtherCAT)
			16#2C07	命令対象外ユニット指定

種別	命令	名称	エラーコード	エラー名
	NX_SaveParam	NX ユニットパラメータ保存	16#0400	入力値範囲外
			16#0419	データ型不正
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
	NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	16#0400	入力値範囲外
			16#0419	データ型不正
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
			16#2C08	累積通電時間記録不正
EtherCAT 通信命令	EC_CoESDOWrite	CoE SDO 書込	16#0400	入力値範囲外
			16#1800	EtherCAT 通信エラー
			16#1801	EtherCAT スレーブ不在
			16#1802	EtherCAT タイムアウト
			16#1804	SDO アボートエラー
			16#1808	通信リソースオーバー
	EC_CoESDORead	CoE SDO 読出	16#0400	入力値範囲外
			16#1800	EtherCAT 通信エラー
			16#1801	EtherCAT スレーブ不在
			16#1802	EtherCAT タイムアウト
			16#1803	受信バッファオーバー
			16#1804	SDO アボートエラー
			16#1808	通信リソースオーバー
	EC_StartMon	EtherCAT パケットモニタ機能開始	16#1805	パケットモニタ保存中
			16#1807	パケットモニタ機能動作中
			16#1808	通信リソースオーバー
			16#1809	パケットモニタ機能非サポート
	EC_StopMon	EtherCAT パケットモニタ機能停止	16#1806	パケットモニタ機能未開始
			16#1808	通信リソースオーバー
			16#1809	パケットモニタ機能非サポート

種別	命令	名称	エラーコード	エラー名
	EC_SaveMon	EtherCAT パケット保存	16#1805	パケットモニタ保存中
			16#1807	パケットモニタ機能動作中
			16#1808	通信リソースオーバー
			16#1809	パケットモニタ機能非サポート
	EC_CopyMon	EtherCAT パケット転送	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1403	指定ファイル不正
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
			16#1808	通信リソースオーバー
			16#1809	パケットモニタ機能非サポート
	EC_DisconnectSlave	EtherCAT スレーブ離脱	16#1800	EtherCAT 通信エラー
			16#1801	EtherCAT スレーブ不在
			16#1808	通信リソースオーバー
			16#180A	対象スレーブへの命令実行不可
	EC_ConnectSlave	EtherCAT スレーブ再加入	16#1800	EtherCAT 通信エラー
			16#1801	EtherCAT スレーブ不在
			16#1808	通信リソースオーバー
			16#180A	対象スレーブへの命令実行不可
	EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	16#1800	EtherCAT 通信エラー
			16#1801	EtherCAT スレーブ不在
			16#1808	通信リソースオーバー
			16#180A	対象スレーブへの命令実行不可

種別	命令	名称	エラーコード	エラー名
	EC_GetMasterStatistics	EtherCAT マスタ診断・統計情報読出	16#0419	データ型不正
			16#1800	EtherCAT 通信エラー
			16#1808	通信リソースオーバー
			16#180D	診断・統計情報ログ機能実行中
			16#180E	マスタ診断・統計情報命令多重起動不可
	EC_ClearMasterStatistics	EtherCAT マスタ診断・統計情報クリア	16#1800	EtherCAT 通信エラー
			16#1808	通信リソースオーバー
			16#180D	診断・統計情報ログ機能実行中
			16#180E	マスタ診断・統計情報命令多重起動不可
	EC_GetSlaveStatistics	EtherCAT スレーブ診断・統計情報読出	16#0400	入力値範囲外
			16#0419	データ型不正
			16#1800	EtherCAT 通信エラー
			16#1808	通信リソースオーバー
			16#180D	診断・統計情報ログ機能実行中
			16#180F	スレーブ診断・統計情報命令多重起動不可
	EC_ClearSlaveStatistics	EtherCAT スレーブ診断・統計情報クリア	16#1800	EtherCAT 通信エラー
			16#1808	通信リソースオーバー
			16#180D	診断・統計情報ログ機能実行中
			16#180F	スレーブ診断・統計情報命令多重起動不可
	NX_WriteObj	NX オブジェクト書込	16#0400	入力値範囲外
			16#0419	データ型不正
			16#041B	データ容量超過
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
			16#2C03	NX メッセージ長不正

種別	命令	名称	エラーコード	エラー名
	NX_ReadObj	NX オブジェクト読出	16#0400	入力値範囲外
			16#0410	文字列フォーマット異常
			16#0419	データ型不正
			16#041C	データサイズ不一致
			16#2C00	NX メッセージ異常
			16#2C01	NX メッセージリソースオーバー
			16#2C02	NX メッセージタイムアウト
IO-Link 通信命令	IOL_ReadObj	IO-Link デバイスオブジェクト読出	16#0400	入力値範囲外
			16#0410	文字列フォーマット異常
			16#0419	データ型不正
			16#041C	データサイズ不一致
			16#4800	デバイスエラー受信
			16#4801	指定ユニット不在
			16#4802	メッセージ処理数オーバー
			16#4803	指定ユニット状態異常
			16#4804	同時実行数オーバー
			16#4805	通信タイムアウト
			16#4806	モード不正
			16#4807	I/O 電源 OFF 状態
			16#4808	照合異常発生
	IOL_WriteObj	IO-Link デバイスオブジェクト書込	16#0400	入力値範囲外
			16#0419	データ型不正
			16#041B	データ容量超過
			16#4800	デバイスエラー受信
			16#4801	指定ユニット不在
			16#4802	メッセージ処理数オーバー
			16#4803	指定ユニット状態異常
			16#4804	同時実行数オーバー
			16#4805	通信タイムアウト
			16#4806	モード不正
			16#4807	I/O 電源 OFF 状態
			16#4808	照合異常発生
EtherNet/IP 通信命令	CIPOpen	CIPClass3 (Large_Forward_Open)コネクション確立	16#0400	入力値範囲外
			16#1C00	Explicit 異常
			16#1C01	ルートパス不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#1C05	Class3 コネクション確立失敗
			16#2000	自 IP アドレス設定エラー
			16#2004	自 IP アドレス未決定

種別	命令	名称	エラーコード	エラー名
	CIPOpenWithDataSize	CIPClass3 コネクション確立（サイズ指定あり）	16#0400	入力値範囲外
			16#1C00	Explicit 異常
			16#1C01	ルートパス不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#1C05	Class3 コネクション確立失敗
			16#2000	自 IP アドレス設定エラー
			16#2004	自 IP アドレス未決定
	CIPRead	変数読出（Class3 Explicit メッセージ）	16#0400	入力値範囲外
			16#0407	領域オーバー
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C02	CIP ハンドル不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#1C06	CIP 通信データサイズオーバー
	CIPWrite	変数書込（Class3 Explicit メッセージ）	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0407	領域オーバー
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C02	CIP ハンドル不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#1C06	CIP 通信データサイズオーバー
	CIPSend	任意 Explicit メッセージ送信（Class3）	16#0400	入力値範囲外
			16#0401	数値関係不正
			16#0406	領域範囲外指定
			16#0407	領域オーバー
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C02	CIP ハンドル不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#1C06	CIP 通信データサイズオーバー
	CIPClose	CIP Class3 コネクションの切断	16#1C02	CIP ハンドル不正
			16#1C03	CIP 通信リソースオーバー

種別	命令	名称	エラーコード	エラー名
	CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	16#0400	入力値範囲外
			16#0407	領域オーバー
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C01	ルートパス不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#2000	自 IP アドレス設定エラー
			16#2004	自 IP アドレス未決定
	CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C01	ルートパス不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#2000	自 IP アドレス設定エラー
			16#2004	自 IP アドレス未決定
	CIPUCMMSend	任意 Explicit メッセージ送信（UCMM）	16#0400	入力値範囲外
			16#0401	数値関係不正
			16#0406	領域範囲外指定
			16#0407	領域オーバー
			16#0419	データ型不正
			16#1C00	Explicit 異常
			16#1C01	ルートパス不正
			16#1C03	CIP 通信リソースオーバー
			16#1C04	CIP タイムアウト
			16#2000	自 IP アドレス設定エラー
			16#2004	自 IP アドレス未決定
	SktUDPCreate	UDP ソケットクリエイト	16#0400	入力値範囲外
			16#2000	自 IP アドレス設定エラー
			16#2001	TCP/UDP ポート使用不可
			16#2003	ソケット状態異常
			16#2004	自 IP アドレス未決定
			16#2008	ソケット通信リソースオーバー

種別	命令	名称	エラーコード	エラー名
	SktUDPRcv	UDP ソケット受信	16#0400	入力値範囲外
			16#0407	領域オーバー
			16#0419	データ型不正
			16#2003	ソケット状態異常
			16#2006	ソケットタイムアウト
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktUDPSend	UDP ソケット送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0419	データ型不正
			16#2002	アドレス解決失敗
			16#2003	ソケット状態異常
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktTCPAccept	TCP ソケットアクセプト	16#0400	入力値範囲外
			16#2000	自 IP アドレス設定エラー
			16#2001	TCP/UDP ポート使用不可
			16#2002	アドレス解決失敗
			16#2003	ソケット状態異常
			16#2004	自 IP アドレス未決定
			16#2006	ソケットタイムアウト
			16#2008	ソケット通信リソースオーバー
	SktTCPConnect	TCP ソケットコネクト	16#0400	入力値範囲外
			16#2000	自 IP アドレス設定エラー
			16#2001	TCP/UDP ポート使用不可
			16#2002	アドレス解決失敗
			16#2003	ソケット状態異常
			16#2004	自 IP アドレス未決定
			16#2005	内蔵 EtherNet/IP ポート利用不可
			16#2006	ソケットタイムアウト
			16#2008	ソケット通信リソースオーバー

種別	命令	名称	エラーコード	エラー名
	SktTCPRcv	TCP ソケット受信	16#0400	入力値範囲外
			16#0407	領域オーバー
			16#0419	データ型不正
			16#2003	ソケット状態異常
			16#2006	ソケットタイムアウト
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktTCPSend	TCP ソケット送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0419	データ型不正
			16#2003	ソケット状態異常
			16#2006	ソケットタイムアウト
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktGetTCPStatus	TCP ソケットのステータス読出	16#2003	ソケット状態異常
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktClose	TCP/UDP ソケットクローズ	16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktClearBuf	TCP/UDP ソケット受信バッファクリア	16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	SktSetOption	TCP ソケットオプション設定	16#0400	入力値範囲外
			16#0419	データ型不正
			16#2003	ソケット状態異常
			16#2007	ソケットハンドル不正
			16#2008	ソケット通信リソースオーバー
	ChangeIPAdr	IP アドレス変更	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#2400	実行権限なし
	ChangeFTPAccount	FTP アカウント変更	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#2400	実行権限なし

種別	命令	名称	エラーコード	エラー名
	FTPGetFileList	FTP サーバのファイルリスト取得	16#0400	入力値範囲外
			16#2403	FTP クライアント実行数オーバー
			16#2405	指定ディレクトリ不正(FTP)
			16#2406	FTP サーバ接続失敗
			16#2407	接続先 FTP サーバ実行失敗
	FTPGetFile	FTP サーバからのファイルダウンロード	16#0400	入力値範囲外
			16#2403	FTP クライアント実行数オーバー
			16#2404	ファイル数制限オーバー
			16#2405	指定ディレクトリ不正(FTP)
			16#2406	FTP サーバ接続失敗
			16#2407	接続先 FTP サーバ実行失敗
			16#2408	SD メモリカードアクセス失敗(FTP)
			16#2409	指定ファイル不在
			16#240A	指定ファイル上書き禁止
			16#240C	指定ファイルアクセス失敗
	FTPPutFile	FTP サーバへのファイルアップロード	16#0400	入力値範囲外
			16#2403	FTP クライアント実行数オーバー
			16#2404	ファイル数制限オーバー
			16#2405	指定ディレクトリ不正(FTP)
			16#2406	FTP サーバ接続失敗
			16#2407	接続先 FTP サーバ実行失敗
			16#2408	SD メモリカードアクセス失敗(FTP)
			16#2409	指定ファイル不在
			16#240A	指定ファイル上書き禁止
			16#240B	指定ファイル削除失敗
			16#240C	指定ファイルアクセス失敗

種別	命令	名称	エラーコード	エラー名
	FTPRemoveFile	FTP サーバのファイル削除	16#0400	入力値範囲外
			16#2403	FTP クライアント実行数オーバー
			16#2404	ファイル数制限オーバー
			16#2405	指定ディレクトリ不正(FTP)
			16#2406	FTP サーバ接続失敗
			16#2407	接続先 FTP サーバ実行失敗
			16#2409	指定ファイル不在
	FTPRemoveDir	FTP サーバのディレクトリ削除	16#0400	入力値範囲外
			16#2405	指定ディレクトリ不正(FTP)
			16#2406	FTP サーバ接続失敗
			16#2407	接続先 FTP サーバ実行失敗
シリアル通信命令	NX_SerialSend	無手順データの送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C04	ポート多重起動
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
	NX_SerialRcv	無手順データの受信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0407	領域オーバー
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C03	受信バッファフル
			16#0C04	ポート多重起動
			16#0C05	パリティエラー
			16#0C06	フレーミングエラー
			16#0C07	オーバーランエラー
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化

種別	命令	名称	エラーコード	エラー名
	NX_ModbusRtuCmd	Modbus-RTU 汎用コマンド送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#0407	領域オーバー
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C03	受信バッファフル
			16#0C04	ポート多重起動
			16#0C05	パリティエラー
			16#0C06	フレーミングエラー
			16#0C07	オーバーランエラー
			16#0C08	CRC 不一致
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
			16#0C10	Modbus 例外レスポンス
			16#0C11	Modbus レスポンス不正
	NX_ModbusRtuRead	Modbus-RTU Read コマンド送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C03	受信バッファフル
			16#0C04	ポート多重起動
			16#0C05	パリティエラー
			16#0C06	フレーミングエラー
			16#0C07	オーバーランエラー
			16#0C08	CRC 不一致
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
			16#0C10	Modbus 例外レスポンス
			16#0C11	Modbus レスポンス不正

種別	命令	名称	エラーコード	エラー名
	NX_ModbusRtuWrite	Modbus-RTU Write コマンド送信	16#0400	入力値範囲外
			16#0406	領域範囲外指定
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C03	受信バッファフル
			16#0C04	ポート多重起動
			16#0C05	パリティエラー
			16#0C06	フレーミングエラー
			16#0C07	オーバーランエラー
			16#0C08	CRC 不一致
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
			16#0C10	Modbus 例外レスポンス
			16#0C11	Modbus レスポンス不正
	NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C04	ポート多重起動
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
	NX_SerialBufClear	バッファクリア	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C04	ポート多重起動
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化

種別	命令	名称	エラーコード	エラー名
	NX_SerialStartMon	シリアルラインモニタの開始	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C04	ポート多重起動
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
	NX_SerialStopMon	シリアルラインモニタの停止	16#0400	入力値範囲外
			16#040D	指定ユニット不正
			16#0419	データ型不正
			16#041D	命令同時実行リソースオーバ
			16#0C04	ポート多重起動
			16#0C0B	シリアル通信タイムアウト
			16#0C0C	対象外ポートへの命令実行
			16#0C0D	CIF ユニット初期化
SD メモリカード命令	FileWriteVar	変数ファイル書込	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1403	指定ファイル不正
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#1409	同一ファイル名あり
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
	FileReadVar	変数ファイル読出	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗

種別	命令	名称	エラーコード	エラー名
	FileOpen	ファイルオープン	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1403	指定ファイル不正
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
	FileClose	ファイルクローズ	16#1400	SD メモリカード使用不可
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#140E	SD メモリカードアクセス失敗
	FileSeek	ファイルシーク	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#1407	オフセット範囲外
			16#140E	SD メモリカードアクセス失敗
	FileRead	ファイル読出	16#0406	領域範囲外指定
			16#0419	データ型不正
			16#1400	SD メモリカード使用不可
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#1406	オープンモード不一致
			16#140E	SD メモリカードアクセス失敗

種別	命令	名称	エラーコード	エラー名
	FileWrite	ファイル書込	16#0406	領域範囲外指定
			16#0419	データ型不正
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#1406	オープンモード不一致
			16#140E	SD メモリカードアクセス失敗
	FileGets	文字列読出	16#1400	SD メモリカード使用不可
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#1406	オープンモード不一致
			16#140E	SD メモリカードアクセス失敗
	FilePuts	文字列書込	16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#1406	オープンモード不一致
			16#140E	SD メモリカードアクセス失敗
	FileCopy	ファイルコピー	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1403	指定ファイル不正
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#1409	同一ファイル名あり
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗

種別	命令	名称	エラーコード	エラー名
	FileRemove	ファイル削除	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1403	指定ファイル不正
			16#1405	ファイルアクセス中
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
	FileRename	ファイル名変更	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1403	指定ファイル不正
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#1408	空ディレクトリ以外
			16#1409	同一ファイル名あり
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
	DirCreate	ディレクトリ作成	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1404	ファイル・ディレクトリ最大数超過
			16#1405	ファイルアクセス中
			16#1409	同一ファイル名あり
			16#140B	ファイルオープン最大数超過
			16#140C	指定ディレクトリ不正
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗

種別	命令	名称	エラーコード	エラー名
	DirRemove	ディレクトリ削除	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1405	ファイルアクセス中
			16#1408	空ディレクトリ以外
			16#140A	指定ファイル書込禁止
			16#140B	ファイルオープン最大数超過
			16#140C	指定ディレクトリ不正
			16#140D	ファイル・ディレクトリ名最大長超過
			16#140E	SD メモリカードアクセス失敗
	BackupToMemoryCard	SD メモリカードバックアップ	16#0400	入力値範囲外
			16#1400	SD メモリカード使用不可
			16#1401	SD メモリカードライトプロテクト
			16#1402	SD メモリカード容量不足
			16#1404	ファイル・ディレクトリ最大数超過
			16#1409	同一ファイル名あり
			16#140C	指定ディレクトリ不正
			16#140E	SD メモリカードアクセス失敗
			16#140F	バックアップ機能既実行
			16#1410	バックアップ実行不可
			16#1411	ユニット／スレーブバックアップ失敗
OS 制御命令	IPC_RebootOS	OS 再起動	16#0400	入力値範囲外
			16#4000	OS タイムアウト
			16#4002	OS 再起動実行エラー

A-2　イベントタスクで使用できない命令

イベントタスクは、指定した実行条件が成立したときに 1 回だけ実行されるタスクです。タスク周期ごとに繰り返し実行されるタスクではありません。したがって、複数のタスク周期にまたがって実行される命令を含むユーザプログラムは、イベントタスクに割り付けることができません。
下表の命令が、複数のタスク周期にまたがって実行される命令です。これらの命令を、イベントタスクに割りつけるユーザプログラム内では使用しないでください。使用すると、ビルド時に異常になります。

種別	命令	名称	ページ
スタック／テーブル命令	RecSort	レコードソート	P.2-525
アナログ制御命令	PIDAT	オートチューニング付き PID 演算	P.2-672
	PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	P.2-699
	AC_StepProgram	ステッププログラム	P.2-777
システム制御命令	ResetPLCError	PLC 異常解除	P.2-820
	ResetMCError	モーション制御異常解除	P.2-827
	ResetECError	EtherCAT 異常解除	P.2-834
	RestartNXUnit	NX ユニットリスタート	P.2-840
	NX_ChangeWriteMode	NX ユニット書込モード変更	P.2-845
	NX_SaveParam	NX ユニットパラメータ保存	P.2-850
	NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	P.2-855
EtherCAT 通信命令	EC_CoESDOWrite	CoE SDO 書込	P.2-896
	EC_CoESDORead	CoE SDO 読出	P.2-899
	EC_StartMon	EtherCAT パケットモニタ機能開始	P.2-904
	EC_StopMon	EtherCAT パケットモニタ機能停止	P.2-910
	EC_SaveMon	EtherCAT パケット保存	P.2-912
	EC_CopyMon	EtherCAT パケット転送	P.2-914
	EC_DisconnectSlave	EtherCAT スレーブ離脱	P.2-916
	EC_ConnectSlave	EtherCAT スレーブ再加入	P.2-923
	EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	P.2-925
	EC_GetMasterStatistics	EtherCAT マスタ診断・統計情報読出	P.2-943
	EC_ClearMasterStatistics	EtherCAT マスタ診断・統計情報クリア	P.2-946
	EC_GetSlaveStatistics	EtherCAT スレーブ診断・統計情報読出	P.2-948
	EC_ClearSlaveStatistics	EtherCAT スレーブ診断・統計情報クリア	P.2-951
	NX_WriteObj	NX オブジェクト書込	P.2-953
	NX_ReadObj	NX オブジェクト読出	P.2-968
IO-Link 通信命令	IOL_ReadObj	IO-Link デバイスオブジェクト読出	P.2-976
	IOL_WriteObj	IO-Link デバイスオブジェクト書込	P.2-984
EtherNet/IP 通信命令	CIPOpen	CIPClass3 (Large_Forward_Open)コネクション確立	P.2-995
	CIPOpenWithDataSize	CIPClass3 コネクション確立（サイズ指定あり）	P.2-1004
	CIPRead	変数読出（Class3 Explicit メッセージ）	P.2-1007
	CIPWrite	変数書込（Class3 Explicit メッセージ）	P.2-1012
	CIPSend	任意 Explicit メッセージ送信（Class3）	P.2-1018
	CIPClose	CIP Class3 コネクションの切断	P.2-1023
	CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	P.2-1025
	CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	P.2-1030
	CIPUCMMSend	任意 Explicit メッセージ送信(UCMM)	P.2-1036
	SktUDPCreate	UDP ソケットクリエイト	P.2-1047

種別	命令	名称	ページ
	SktUDPRcv	UDP ソケット受信	P.2-1055
	SktUDPSend	UDP ソケット送信	P.2-1058
	SktTCPAccept	TCP ソケットアクセプト	P.2-1061
	SktTCPConnect	TCP ソケットコネクト	P.2-1064
	SktTCPRcv	TCP ソケット受信	P.2-1072
	SktTCPSend	TCP ソケット送信	P.2-1075
	SktGetTCPStatus	TCP ソケットのステータス読出	P.2-1078
	SktClose	TCP/UDP ソケットクローズ	P.2-1081
	SktClearBuf	TCP/UDP ソケット受信バッファクリア	P.2-1084
	SktSetOption	TCP ソケットオプション設定	P.2-1087
	ChangeIPAdr	IP アドレス変更	P.2-1092
	ChangeFTPAccount	FTP アカウント変更	P.2-1100
	FTPGetFileList	FTP サーバのファイルリスト取得	P.2-1104
	FTPGetFile	FTP サーバからのファイルダウンロード	P.2-1117
	FTPPutFile	FTP サーバへのファイルアップロード	P.2-1125
	FTPRemoveFile	FTP サーバのファイル削除	P.2-1135
	FTPRemoveDir	FTP サーバのディレクトリ削除	P.2-1144
シリアル通信命令	NX_SerialSend	無手順データの送信	P.2-1150
	NX_SerialRcv	無手順データの受信	P.2-1161
	NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	P.2-1174
	NX_ModbusRtuRead	Modbus RTU Read コマンド送信	P.2-1184
	NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	P.2-1195
	NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	P.2-1206
	NX_SerialBufClear	バッファクリア	P.2-1213
	NX_SerialStartMon	シリアルラインモニタの開始	P.2-1222
	NX_SerialStopMon	シリアルラインモニタの停止	P.2-1226
SD メモリカード命令	FileWriteVar	変数ファイル書込	P.2-1232
	FileReadVar	変数ファイル読出	P.2-1237
	FileOpen	ファイルオープン	P.2-1242
	FileClose	ファイルクローズ	P.2-1246
	FileSeek	ファイルシーク	P.2-1249
	FileRead	ファイル読出	P.2-1252
	FileWrite	ファイル書込	P.2-1260
	FileGets	文字列読出	P.2-1268
	FilePuts	文字列書込	P.2-1275
	FileCopy	ファイルコピー	P.2-1284
	FileRemove	ファイル削除	P.2-1291
	FileRename	ファイル名変更	P.2-1295
	DirCreate	ディレクトリ作成	P.2-1300
	DirRemove	ディレクトリ削除	P.2-1303
	BackupToMemoryCard	SD メモリカードバックアップ	P.2-1306
タイムスタンプ命令	NX_DOutTimeStamp	タイムスタンプデジタル出力書込	P.2-1320
	NX_AryDOutTimeStamp	タイムスタンプデジタル出力配列書込	P.2-1325
OS 制御命令	IPC_RebootOS	OS 再起動	P.2-1337

A-3　NX メッセージ通信異常に関連する命令

以下の命令を一度に多数実行すると、「NX メッセージ通信異常」という異常が発生する可能性があります。「NX メッセージ通信異常」が発生した場合は、以下の命令の実行数を減らしてください。「NX メッセージ通信異常」が発生する条件は、通信負荷などによって変わります。

種別	命令	名称	ページ
システム制御命令	RestartNXUnit	NX ユニットリスタート	P.2-840
	NX_ChangeWriteMode	NX ユニット書込モード変更	P.2-845
	NX_SaveParam	NX ユニットパラメータ保存	P.2-850
	NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	P.2-855
EtherCAT 通信命令	EC_CoESDOWrite	CoE SDO 書込	P.2-896
	EC_CoESDORead	CoE SDO 読出	P.2-899
	EC_StartMon	EtherCAT パケットモニタ機能開始	P.2-904
	EC_StopMon	EtherCAT パケットモニタ機能停止	P.2-910
	EC_SaveMon	EtherCAT パケット保存	P.2-912
	EC_CopyMon	EtherCAT パケット転送	P.2-914
	EC_DisconnectSlave	EtherCAT スレーブ離脱	P.2-916
	EC_ConnectSlave	EtherCAT スレーブ再加入	P.2-923
	EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	P.2-925
	NX_WriteObj	NX オブジェクト書込	P.2-953
	NX_ReadObj	NX オブジェクト読出	P.2-968
IO-Link 通信命令	IOL_ReadObj	IO-Link デバイスオブジェクト読出	P.2-976
	IOL_WriteObj	IO-Link デバイスオブジェクト書込	P.2-984

A-4 SDO アボートコード一覧

参考情報として、EtherCAT 通信の、SDO アボートコードを示します。実際の通信で使用されるアボートコードは、スレーブ側によって規定されます。プログラミングの際には、スレーブ側のマニュアルを、参照してください。

値	意味
16#05030000	トグルビット変化なし
16#05040000	SDO プロトコルタイムアウト
16#05040001	無効／不明なクライアント／サーバコマンド指定子
16#05040005	メモリ範囲外
16#06010000	オブジェクトへの未サポートアクセス
16#06010001	書込専用オブジェクトへのリードアクセス
16#06010002	読取専用オブジェクトへのライトアクセス
16#06020000	オブジェクトディクショナリに存在しないオブジェクト
16#06040041	オブジェクトを PDO にマッピング不能
16#06040042	マッピングされたオブジェクトの数／長さが PDO 長を超える
16#06040043	一般的なパラメータ不一致
16#06040047	デバイスの一般的な内部不一致
16#06060000	ハードウェアエラーによるアクセス失敗
16#06070010	データ型不一致、サービスパラメータ長不一致
16#06070012	データ型不一致、サービスパラメータが長すぎる
16#06070013	データ型不一致、サービスパラメータが短すぎる
16#06090011	サブインデックスが存在しない
16#06090030	パラメータ値が範囲外（ライトアクセスのみ）
16#06090031	書き込まれたパラメータの値が大きすぎる
16#06090032	書き込まれたパラメータの値が小さすぎる
16#06090036	最大値が最小値より小さい
16#08000000	一般的なエラー
16#08000020	データがアプリケーションに転送／格納できない
16#08000021	ローカル制御 *1 のため、データをアプリケーションに転送／格納できない
16#08000022	現在のデバイス状態では、データをアプリケーションに転送／格納できない
16#08000023	オブジェクトディクショナリ動的生成失敗、あるいはオブジェクトディクショナリが存在しない

*1. スレーブ固有の内部状態の問題

引用元: EtherCAT 仕様書 Part6 アプリケーション層プロトコル仕様
文書番号: ETG.1000.6 S(R) V1.0.2(J02)

A-5　バージョン関連情報

ここでは、コントローラと Sysmac Studio の各バージョンの更新により、仕様変更された命令と新規命令の一覧を示します。
命令には、コントローラと Sysmac Studio の各バージョンによって、対応状況や仕様が異なるものがあります。以下に、その内容を示します。
コントローラと Sysmac Studio の両方のバージョンが書かれている命令は、両方ともそのバージョン以降である必要があることを意味します。

種別	命令	名称	新規／仕様変更	バージョン		ページ
				CPU ユニット	Sysmac Studio	
算術命令	EXPT,**	べき乗演算	仕様変更	Ver.1.16	Ver.1.20	P.2-213
データ変換命令	LOWER_BOUND	配列開始番号取得	新規	Ver.1.18	Ver.1.22	P.2-495
	UPPER_BOUND	配列終了番号取得	新規	Ver.1.18	Ver.1.22	P.2-495
アナログ制御命令	AC_StepProgram	ステッププログラム	仕様変更	Ver.1.21	Ver.1.29	P.2-777
EtherNet/IP 通信命令	FTPGetFileList	FTP サーバのファイルリスト取得	仕様変更	Ver.1.16	－	P.2-1104
	FTPGetFile	FTP サーバからのファイルダウンロード	仕様変更	Ver.1.16	－	P.2-1117
	FTPPutFile	FTP サーバへのファイルアップロード	仕様変更	Ver.1.16	－	P.2-1125
	FTPRemoveFile	FTP サーバのファイル削除	仕様変更	Ver.1.16	－	P.2-1135
	FTPRemoveDir	FTP サーバのディレクトリ削除	仕様変更	Ver.1.16	－	P.2-1144

I

索引

索引

Numerics

- (減算)....2-176
-OU (減算オーバーフローチェック付き)....2-179
/ (除算)....2-190
* (乗算)....2-183
** (べき乗演算)....2-213
_BCD_TO_* (BCD→符号なし整数変換グループ)....2-242
_TO_* (実数→実数変換グループ)....2-282
_TO_* (実数→整数変換グループ)....2-277
_TO_* (実数→ビット列変換グループ)....2-280
_TO_* (整数→実数変換グループ)....2-268
_TO_* (整数→整数変換グループ)....2-262
_TO_* (整数→ビット列変換グループ)....2-265
_TO_* (ビット列→実数変換グループ)....2-275
_TO_* (ビット列→整数変換グループ)....2-270
_TO_* (ビット列→ビット列変換グループ)....2-273
_TO_BCD_* (符号なし整数→BCD 変換グループ)....2-245
**_TO_STRING (実数→文字列変換グループ)....2-288
**_TO_STRING (整数→文字列変換グループ)....2-284
**_TO_STRING (ビット列→文字列変換グループ)....2-286
*OU (乗算 オーバーフローチェック付き)....2-186
& (論理積)....2-324
+ (加算)....2-169
+OU (加算 オーバーフローチェック付き)....2-173
< (比較 LT)....2-100
<= (比較 LE)....2-100
<> (比較 NE)....2-98
= (比較 EQ)....2-96
> (比較 GT)....2-100
>= (比較 GE)....2-100
100ms タイマ....2-141
16 進文字列→数値変換グループ....2-451
1 ビットアウト....2-67
1 ビットセット....2-65
1 ビットリセット....2-65
2 バイト結合....2-491
4 バイト結合....2-493
4 ビット結合グループ....2-472
4 ビット分離....2-470

A

ABS (絶対値)....2-195
AC_StepProgram (ステッププログラム)....2-777
AccumulationTimer (積算タイマ)....2-137
ACOS (COS^{-1} 演算)....2-202
ActEventTask (イベントタスク起動)....2-1363
ADD (加算)....2-169
ADD_DT_TIME (日付時刻と時間の加算)....2-603
ADD_TIME (時間加算)....2-599
ADD_TOD_TIME (時刻と時間の加算)....2-601
AddDelimiter (デリミタ付加文字列書込)....2-575
AddOU (加算オーバーフローチェック付き))....2-173
AND (アンド)....2-17
AND (論理積)....2-324
ANDN (アンドノット)....2-17
AryAdd (配列一括加算)....2-221
AryAddV (配列要素加算)....2-223
AryAnd (配列論理積)....2-331
AryByteTo (バイト配列からの変換)....2-484
AryCmpEQ (配列一括比較 EQ)....2-117
AryCmpEQV (配列要素比較 EQ)....2-121
AryCmpGE (配列一括比較 GE)....2-119
AryCmpGEV (配列要素比較 GE)....2-123
AryCmpGT (配列一括比較 GT)....2-119
AryCmpGTV (配列要素比較 GT)....2-123
AryCmpLE (配列一括比較 LE)....2-119
AryCmpLEV (配列要素比較 LE)....2-123
AryCmpLT (配列一括比較 LT)....2-119
AryCmpLTV (配列要素比較 LT)....2-123
AryCmpNE (配列一括比較 NE)....2-117
AryCmpNEV (配列要素比較 NE)....2-121
AryCRC16 (CRC-16 値算出 配列)....2-550
AryCRCCCITT (CRC-CCITT 値算出 配列)....2-548
AryExchange (配列データ交換)....2-371
AryLRC_** (LRC 値算出 配列 グループ)....2-546
AryMax (配列変数の最大値検索)....2-349
AryMean (配列要素の平均値算出)....2-229
AryMin (配列変数の最小値検索)....2-349
AryMove (配列の転送)....2-373
AryOr (配列論理和)....2-331
arySD (配列要素の標準偏差)....2-231
arySearch (配列検索)....2-352
aryShiftReg (シフトレジスタ)....2-390
aryShiftRegLR (左右シフトレジスタ)....2-392
arySHL (配列の N 要素左シフト)....2-395
arySHR (配列の N 要素右シフト)....2-395
arySub (配列一括減算)....2-225
arySubV (配列要素減算)....2-227
AryToBCD (配列一括 BCD 変換)....2-257
AryToBin (配列一括符号なし整数変換)....2-259
AryToString (配列→文字列変換)....2-468
AryXor (配列排他的論理和)....2-331
AryXorN (配列排他的論理和否定)....2-331
ASIN (SIN^{-1} 演算)....2-202
ATAN (TAN^{-1} 演算)....2-202

B

BackupToMemoryCard (SD メモリカードバックアップ)....2-1306
Band (デッドバンド制御)....2-343
BCD_TO_** (BCD 群→符号なし整数変換グループ)....2-248
BCDsToBin (符号付き BCD→符号付き整数変換)....2-251
BCD 群→符号なし整数変換グループ....2-248
BCD→符号なし整数変換グループ....2-242

BinToBCDs_** (符号付き整数→BCD 変換グループ).....2-254
BinToGray_** (BIN コード→グレイコード変換グループ)..2-463
BIN コード→ グレイコード変換グループ.....2-463
BitCnt (ビットカウンタ).....2-415
BREAK (ループブレイク).....2-93

C

CASE (多方向分岐).....2-32
ChangeFTPAccount (FTP アカウント変更) 2-1100
ChangeIPAdr (IP アドレス変更) 2-1092
CheckReal (実数チェック).....2-237
ChkLeapYear (うるう年判別).....2-645
ChkRange (範囲型変数チェック).....2-1348
CIP Class3 コネクション確立.....2-995
CIP Class3 コネクション確立 (サイズ指定あり) .2-1004
CIP Class3 コネクションの切断.....2-1023
CIPClose (CIP Class3 コネクションの切断).....2-1023
CIPOpen (CIP Class3 コネクション確立).....2-995
CIPOpenWithDataSize (CIP Class3 コネクション確立 (サイズ指定あり) 2-1004
CIPRead (変数読出 Class3 Explicit メッセージ)...2-1007
CIPSend (任意 Explicit メッセージ送信 Class3)...2-1018
CIPUCMMRead (変数読出 UCMM Explicit メッセージ)...2-1025
CIPUCMMSend (任意 Explicit メッセージ送信 UCMM)....2-1036
CIPUCMMWrite (変数書込 UCMM Explicit メッセージ)....2-1030
CIPWrite (変数書込 Class3 Explicit メッセージ)...2-1012
Clear (初期化).....2-375
ClearString (文字列・クリア).....2-570
Cmp (比較).....2-110
CoE SDO 書込.....2-896
CoE SDO 読出.....2-899
ColmToLine_** (ビット列→ビット行変換グループ) 2-416
CONCAT (文字列・連結).....2-554
CONCAT_ DATE_TOD (日付と時刻の結合).....2-619
Copy**To*** (ビットパターンコピー 実数→ビット列 グループ).....2-385
Copy**To*** (ビットパターンコピー ビット列→実数 グループ).....2-379
Copy**ToNum (ビットパターンコピー 実数→符号付き整数 グループ).....2-386
Copy**ToNum (ビットパターンコピー ビット列→符号付き整数 グループ).....2-377
CopyNumTo** (ビットパターンコピー 符号付き整数→実数 グループ).....2-383
CopyNumTo** (ビットパターンコピー 符号付き整数→ビット列 グループ).....2-381
COS (COS 演算).....2-199
COS^{-1} 演算.....2-202
COS 演算.....2-199
CRC-CCITT 値算出 (文字列) 2-542
CRC-16 値算出 (配列) 2-550
CRC-16 値算出 (文字列) 2-544
CRC-CCITT 値算出 (配列) 2-548
CTD (減算カウンタ).....2-146
CTD_** (減算カウンタグループ) 2-148
CTU (加算カウンタ).....2-151
CTU_** (加算カウンタグループ).....2-153
CTUD (可逆カウンタ).....2-156
CTUD_** (可逆カウンタグループ).....2-161

D

DateStructToDt (時刻結合).....2-656
DateToSec (日付→秒変換).....2-629
DateToString (日付→文字列変換).....2-460
DaysToMonth (日数→月変換).....2-648
Dec (デクリメント).....2-217
Decoder (ビットデコーダ).....2-410
DegToRad (角度→ラジアン変換).....2-197
DELETE (文字列・削除).....2-565
DirCreate (ディレクトリ作成).....2-1300
DirRemove (ディレクトリ削除).....2-1303
Dispart8Bit (バイト単位データ分離).....2-474
DispartDigit (4 ビット分離).....2-470
DispartReal (実数の仮数・指数分離).....2-443
DIV (除算).....2-190
DIVTIME (時間除算).....2-617
Down (立ち下がり微分).....2-46
DT_TO_DATE (日付時刻からの日付切出).....2-623
DT_TO_TOD (日付時刻からの時刻切出).....2-621
DtToDateStruct (時刻分解).....2-654
DtToSec (日付時刻→秒変換).....2-627
DtToString (日時→文字列変換).....2-458

E

EC_ChangeEnableSetting (EtherCAT スレーブ有効／無効切替).....2-925
EC_ClearMasterStatistics (EtherCAT マスタ診断・統計情報クリア).....2-946
EC_ClearSlaveStatistics (EtherCAT スレーブ診断・統計情報クリア).....2-951
EC_CoESDOWrite (CoE SDO 書込).....2-896
EC_CoESDORead (CoE SDO 読出).....2-899
EC_ConnectSlave (EtherCAT スレーブ再加入).....2-923
EC_CopyMon (EtherCAT パケット転送).....2-914
EC_DisconnectSlave (EtherCAT スレーブ離脱)).....2-916
EC_GetMasterStatistics (EtherCAT マスタ診断・統計情報読出).....2-943
EC_GetSlaveStatistics (EtherCAT スレーブ診断・統計情報読出).....2-948
EC_SaveMon (EtherCAT パケット保存).....2-912
EC_StartMon (EtherCAT パケットモニタ機能開始)2-904
EC_StopMon (EtherCAT パケットモニタ機能停止)2-910
Encoder (ビットエンコーダ).....2-413
End (エンド).....2-70
EnumToNum (列挙型→整数変換) 2-315
EQ (比較 EQ).....2-96
EQascii (文字列比較 EQ).....2-103
EtherCAT 異常解除.....2-834
EtherCAT 異常ステータス取得.....2-836
EtherCAT スレーブ再加入.....2-923
EtherCAT スレーブ診断・統計情報クリア.....2-951

EtherCAT スレーブ診断・統計情報読出.....2-948
EtherCAT スレーブ有効／無効切替.....2-925
EtherCAT スレーブ離脱.....2-916
EtherCAT パケット転送.....2-914
EtherCAT パケット保存.....2-912
EtherCAT パケットモニタ機能開始.....2-904
EtherCAT パケットモニタ機能停止.....2-910
EtherCAT マスタ診断・統計情報クリア.....2-946
EtherCAT マスタ診断・統計情報読出.....2-943
EtherNet/IP 異常ステータス取得.....2-825
Exchange (データ交換).....2-369
EXIT (ループブレイク)..... 2-40
EXP (自然指数演算).....2-211
EXPT (べき乗演算).....2-213

F

F_TRIG (立ち下がり微分).....2-46
FileClose (ファイルクローズ).....2-1246
FileCopy (ファイルコピー).....2-1284
FileGets (文字列読出).....2-1268
FileOpen (ファイルオープン).....2-1242
FilePuts (文字列書込).....2-1275
FileRead (ファイル読出).....2-1252
FileReadVar (変数ファイル読出).....2-1237
FileRemove (ファイル削除).....2-1291
FileRename (ファイル名変更)..... 2-1295
FileSeek (ファイルシーク)..... 2-1249
FileWrite (ファイル書込).....2-1260
FileWriteVar (変数ファイル書込)..... 2-1232
FIND (文字列・検索)..... 2-560
FixNumToString (固定小数点数→文字列変換).....2-453
FOR (繰り返し開始)..... 2-86
Fraction (実数小数部抽出).....2-235
FTPGetFileList (FTP サーバのファイル一覧取得) 2-1104
FTPGetFile (FTP サーバからのファイルダウンロード) ....
.....2-1117
FTPPu t File (FTP サーバへのファイルアップロード) ...
.....2-1125
FTPRemoveDir (FTP サーバのディレクトリ削除) 2-1144
FTPRemoveFile (FTP サーバのファイル削除) 2-1135
FTP アカウント変更..... 2-1100
FTP サーバからのファイルダウンロード.....2-1117
FTP サーバのディレクトリ削除.....2-1144
FTP サーバのファイル削除.....2-1135
FTP サーバのファイルリスト取得.....2-1104
FTP サーバへのファイルアップロード..... 2-1125

G

GE (比較 GE).....2-100
GEascii (文字列比較 GE).....2-107
Get**Clk (クロックパルス取得グループ)..... 2-1370
Get**Cnt (加算フリーランニングカウンタ取得グループ)....
.....2-1371
GetAlarm (ユーザ異常ステータス取得).....2-818
GetByteLen (文字列・バイト数取得).....2-569
GetDayOfWeek (曜日取得).....2-650
GetDaysOfMonth (月の日数取得)..... 2-646
GetECError (EtherCAT 異常ステータス取得)..... 2-836
GetEIPError (EtherNet/IP 異常ステータス取得)..... 2-825
GetMCError (モーション制御異常ステータス取得).. 2-832
GetMyTaskInterval (自タスク設定周期読出)..... 2-1353
GetMyTaskStatus (自タスク状態読出).....2-1350
GetPLCError (PLC 異常ステータス取得)..... 2-823
GetTime (時刻取得).....2-625
GetTraceStatus (データトレースステータス読出)...2-808
GetWeekOfYear (週取得).....2-652
Gray (グレイコード変換).....2-420
GrayToBin_** (グレイコード→BIN コード変換グループ)..
.....2-463
GT (比較 GT)..... 2-100
GTascii (文字列比較 GT)..... 2-107

H

HexStringToNum_** (16 進文字列→数値変換グループ)....
.....2-451

I

IF (選択).....2-28
Inc (インクリメント)..... 2-217
INSERT (文字列・挿入).....2-567
IO-Link デバイスオブジェクト書込.....2-984
IO-Link デバイスオブジェクト読出.....2-976
IOL_ReadObj (IO-Link デバイスオブジェクト読出) 2-976
IOL_WriteObj (IO-Link デバイスオブジェクト書込) 2-984
IPC_GetOSStatus(OS ステータス読み出し).....2-1334
IPC_RebootOS(OS 再起動)..... 2-1337
IPC_Shutdown(シャットダウン).....2-1340
IP アドレス変更.....2-1092

J

JMP (ジャンプ).....2-84

L

LD (ロード)..... 2-14
LDN (ロードノット).....2-14
LE (比較 LE)..... 2-100
LEascii (文字列比較 LE).....2-107
LEFT (文字列・左からの取出)..... 2-556
LEN (文字列・長さ検出).....2-562
LIMIT (上下限リミット制限)..... 2-341
LimitAlarm_** (上下限警報グループ) 2-750
LimitAlarmDv_** (上下限偏差警報グループ) 2-754
LimitAlarmDvStbySeq_** (待機シーケンス付き上下限偏差
警報グループ) 2-758
LineToColm (ビット行→ビット列変換).....2-418
LN (自然対数演算)..... 2-207
Lock (タスク間排他ロック).....2-1357
LOG (常用対数演算).....2-207
LOWER_BOUND (配列開始番号取得) 2-495
LRC 値算出 (配列)グループ.....2-546
LRC 値算出 (文字列) 2-540
LrealToFormatString (実数 LREAL→文字列変換書式付
き).....2-296
LT (比較 LT).....2-100

LTascii (文字列比較 LT).....2-107

M

MAX (最大値検索).....2-347
MC (マスタコントロール開始).....2-72
MCR (マスタコントロール終了).....2-72
MemCopy (メモリコピー).....2-365
MID (文字列・任意位置からの取出).....2-558
MIN (最小値検索).....2-347
MOD (剰余).....2-193
Modbus RTU Read コマンド送信.....2-1184
Modbus RTU Write コマンド送信.....2-1195
Modbus RTU 汎用コマンド送信.....2-1174
ModReal (実数剰余).....2-233
MOVE (転送).....2-356
MoveBit (ビット転送).....2-359
MoveDigit (ディジット転送).....2-361
MovingAverage (移動平均).....2-437
MUL (乗算).....2-183
MulOU (乗算 オーバーフローチェック付き).....2-186
MULTIME (時間乗算).....2-615
MUX (マルチプレクサ).....2-338

N

NanoSecToTime (ナノ秒→時間変換).....2-642
NE (比較 NE).....2-98
NEascii (文字列比較 NE).....2-105
Neg (符号反転).....2-408
NEXT (繰り返し終了).....2-86
NOT (ビット反転).....2-329
NSHLC (N ビットデータ N ビット左シフト CY 付き).....2-400
NSHRC (N ビットデータ N ビット右シフト CY 付き).....2-400
NumToDecString (固定長 10 進文字列変換).....2-448
NumToEnum (整数→列挙型変換).....2-317
NumToHexString (固定長 16 進文字列変換).....2-448
NX_AryDOutTimeStamp(タイムスタンプデジタル出力配列書込).....2-1325
NX_ChangeWriteMode(NX ユニット書込モード変更).....2-845
NX_DOutTimeStamp(タイムスタンプデジタル出力書込).....2-1320
NX_ModbusRtuCmd (Modbus RTU 汎用コマンド送信).....2-1174
NX_ModbusRtuRead (Modbus RTU Read コマンド送信).....2-1184
NX_ModbusRtuWrite (Modbus RTU Write コマンド送信).....2-1195
NX_ReadObj(NX オブジェクト読出).....2-968
NX_ReadTotalPowerOnTime (NX ユニット累積通電時間読出).....2-855
NX_SaveParam(NX ユニットパラメータ保存).....2-850
NX_SerialBufClear (バッファクリア).....2-1213
NX_SerialRcv (無手順データの受信).....2-1161
NX_SerialSend (無手順データの送信).....2-1150
NX_SerialSigCtl (シリアル制御信号の ON/OFF 切り替え).....2-1206
NX_SerialStartMon (シリアルラインモニタの開始).....2-1222
NX_SerialStopMon (シリアルラインモニタの停止).....2-1226
NX_WriteObj(NX オブジェクト書込).....2-953
NX オブジェクト書込.....2-953
NX オブジェクト読出.....2-968
NX ユニット書込モード変更.....2-845
NX ユニットパラメータ保存.....2-850
NX ユニットリスタート.....2-840
NX ユニット累積通電時間読出.....2-855
N ビット書込グループ.....2-1346
N ビットデータ N ビット左シフト CY 付き.....2-400
N ビットデータ N ビット右シフト CY 付き.....2-400
N ビット左回転.....2-402
N ビット左シフト.....2-398
N ビット右回転.....2-402
N ビット右シフト.....2-398
N ビット読出グループ.....2-1344

O

OR (オア).....2-20
OR (論理和).....2-324
ORN (オアノット).....2-20
OS 再起動.....2-1337
OS ステータス読み出し.....2-1334
Out (出力).....2-23
OutABit (1 ビットアウト).....2-67
OutNot (否定出力).....2-23

P

PackDword (4 バイト結合).....2-493
PackWord (2 バイト結合).....2-491
PIDAT (オートチューニング付き PID 演算).....2-672
PIDAT_HeatCool.....2-699
PLC 異常解除.....2-820
PLC 異常ステータス取得.....2-823
PrgStart (プログラム実行指令).....2-864
PrgStatus (プログラム状態読出).....2-889
PrgStop (プログラム停止指令).....2-872
PWLApprox (折れ線近似変換 (折れ線チェックあり)).....2-429
PWLApproxNoLineChk (折れ線近似変換 (折れ線チェックなし)).....2-429
PWLDatChk (折れ線データチェック).....2-434

R

R_TRIG (立ち上がり微分).....2-46
RadToDeg (ラジアン→角度変換).....2-197
Rand (乱数発生).....2-219
ReadNbit_** (N ビット読出グループ).....2-1344
RealToFormatString (実数 REAL→文字列変換書式付き).....2-290
RecMax (レコード最大値検索).....2-533
RecMin (レコード最小値検索).....2-533
RecNum (レコード数取得).....2-531
RecRangeSearch (範囲指定レコード検索).....2-520

RecSearch (レコード検索)....2-516
RecSort (レコードソート)....2-525
REPEAT (繰り返し)....2-38
REPLACE (文字列・置換)....2-563
Reset (リセット)....2-60
ResetABit (1 ビットリセット)....2-65
ResetAlarm (ユーザ異常解除)....2-816
ResetBits (多ビットリセット)....2-63
ResetECError (EtherCAT 異常解除)....2-834
ResetMCError (モーション制御異常解除)....2-827
ResetPLCError (PLC 異常解除)....2-820
RestartNXUnit (NX ユニットリスタート)....2-840
RETURN (復帰)....2-43, 2-71
RIGHT (文字列・右からの取出)....2-556
ROL (N ビット左回転)....2-402
ROR (N ビット右回転)....2-402
Round (実数丸め)....2-319
RoundUp (実数切り上げ)....2-319
RS (リセット優先キープ)....2-54

S

ScaleTrans(スケール変換)....2-774
SD メモリカードバックアップ....2-1306
SecToDate (秒→日付変換)....2-635
SecToDt (秒→日付時刻変換)....2-633
SecToTime (秒→時間変換)....2-643
SecToTod (秒→時刻変換)....2-637
SEL (ビット選択)....2-336
Set (セット)....2-60
SetABit (1 ビットセット)....2-65
SetAlarm (ユーザ異常発生)....2-811
SetBits (多ビットセット)....2-63
SetBlock (ブロック設定)....2-367
SetInfo (ユーザ情報発生)....2-838
SHL (N ビット左シフト)....2-398
SHR (N ビット右シフト)....2-398
SIN (SIN 演算)....2-199
SIN^{-1} 演算....2-202
SIN 演算....2-199
SizeOfAry (配列要素数取得)....2-489
SJISToUTF8 (文字コード変換(SJIS→UTF-8))....2-427
SktClearBuf (TCP/UDP ソケット受信バッファクリア)....2-1084
SktClose (TCP/UDP ソケットクローズ)....2-1081
SktGetTCPStatus (TCP ソケットのステータス読出)....2-1078
SktSetOption(TCP ソケットオプション設定)....2-1087
SktTCPAccept (TCP ソケットアクセプト)....2-1061
SktTCPConnect (TCP ソケットコネクト)....2-1064
SktTCPRcv (TCP ソケット受信)....2-1072
SktTCPSend (TCP ソケット送信)....2-1075
SktUDPCreate (UDP ソケットクリエイト)....2-1047
SktUDPRcv (UDP ソケット受信)....2-1055
SktUDPSend (UDP ソケット送信)....2-1058
SQRT (平方根演算)....2-205
SR (セット優先キープ)....2-57
StackDel (スタックデータ削除)....2-514
StackFIFO (先入れ先出し)....2-509
StackIns (スタックデータ挿入)....2-512
StackLIFO (後入れ先出し)....2-509
StackPush (スタックデータ格納)....2-500
STRING_TO_** (文字列→実数変換グループ)....2-306
STRING_TO_** (文字列→整数変換グループ)....2-302
STRING_TO_** (文字列→ビット列変換グループ)....2-304
StringCRC16 (CRC-16 値算出 文字列)....2-544
StringCRCCCITT (CRC-CCITT 値算出 文字列)....2-542
StringLRC (LRC 値算出 文字列)....2-540
StringSum (サム値算出)....2-538
StringToAry (文字列→配列変換)....2-466
StringToFixNum (文字列→固定小数点数変換)....2-455
SUB (減算)....2-176
SUB_DATE_DATE (日付減算)....2-610
SUB_DT_DT (日付時刻減算)....2-611
SUB_DT_TIME (日付時刻と時間の減算)....2-613
SUB_TIME (時間減算)....2-605
SUB_TOD_TIME (時刻と時間の減算)....2-607
SUB_TOD_TOD (時刻減算)....2-609
SubDelimiter (文字列読出デリミタ削除)....2-586
SubOU (減算オーバーフローチェック付き)....2-179
Swap (バイトスワップ)....2-407

T

TableCmp (テーブル比較)....2-114
TAN (TAN 演算)....2-199
TAN^{-1} 演算....2-202
TAN 演算....2-199
Task_IsActive (タスク実行中判定)....2-1355
TCP/UDP ソケットクローズ....2-1081
TCP/UDP ソケット受信バッファクリア....2-1084
TCP ソケットアクセプト....2-1061
TCP ソケットオプション設定....2-1087
TCP ソケットコネクト....2-1064
TCP ソケット受信....2-1072
TCP ソケット送信....2-1075
TCP ソケットのステータス読出....2-1078
TestABit (ビットテスト)....2-50
TestABitN (ビットテスト否定)....2-50
TimeProportionalOut (時分割比例出力)....2-733
Timer (100ms タイマ)....2-141
TimeToNanoSec (時間→ナノ秒変換)....2-639
TimeToSec (時間→秒変換)....2-640
TO_** (実数変換グループ)....2-313
TO_** (整数変換グループ)....2-309
TO_** (ビット列変換グループ)....2-311
ToAryByte (バイト配列への変換)....2-478
TodToSec (時刻→秒変換)....2-631
TodToString (時刻→文字列変換)....2-461
TOF (オフディレータイマ)....2-131
ToLCase (文字列・小文字変換)....2-571
TON (オンディレータイマ)....2-126
ToUCase (文字列・大文字変換)....2-571
TP (パルス出力)....2-134
TraceSamp (データトレースサンプリング)....2-802
TraceTrig (データトレーストリガ条件成立)....2-805
TransBits (多ビット転送)....2-363
TrimL (文字列・左トリム)....2-573

TrimR (文字列・右トリム)....2-573
TRUNC (実数切り捨て)....2-319
TruncDt (日付時刻切り捨て)....2-662
TruncTime (時間切り捨て)....2-658
TruncTod (時刻切り捨て)....2-666

U

UDP ソケットクリエイト....2-1047
UDP ソケット受信....2-1055
UDP ソケット送信....2-1058
Unite8Bit_** (バイト単位データ結合グループ)....2-476
UniteDigit_** (4 ビット結合グループ)....2-472
UniteReal (仮数・指数の実数への結合)....2-446
Unlock (タスク間排他ロック解除)....2-1357
Up (立ち上がり微分)....2-46
UPPER_BOUND (配列終了番号取得)....2-495
UTF8ToSJIS (文字コード変換(UTF-8→SJIS))....2-425

W

WHILE (繰り返し)....2-36
WriteNbit_** (N ビット書込グループ)....2-1346

X

XOR (排他的論理和)....2-324
XORN (排他的論理和否定)....2-327

Z

Zone (デッドゾーン制御)....2-345
ZoneCmp (領域比較)....2-112

あ

後入れ先出し....2-509
アンド....2-17
アンドノット....2-17

い

移動平均....2-437
イベントタスク起動....2-1363
インクリメント....2-217

う

うるう年判別....2-645

え

エンド....2-70

お

オア....2-20
オアノット....2-20
オフディレータイマ....2-131
折れ線近似変換（折れ線チェックあり）....2-429
折れ線近似変換（折れ線チェックなし）....2-429
折れ線データチェック....2-434
オンディレータイマ....2-126
オートチューニング付き PID 演算....2-672
オートチューニング付き加熱冷却 PID 演算....2-699

か

可逆カウンタ....2-156
可逆カウンタグループ....2-161
角度→ラジアン変換....2-197
加算....2-169
加算（オーバーフローチェック付き）....2-173
加算カウンタ....2-151
加算カウンタグループ....2-153
加算フリーランニングカウンタ取得グループ....2-1371
仮数・指数の実数への結合....2-446

く

繰り返し (REPEAT)....2-38
繰り返し (WHILE)....2-36
繰り返し開始....2-86
繰り返し終了....2-86
グレイコード→BIN コード変換グループ....2-463
グレイコード変換....2-420
クロックパルス取得グループ....2-1370

け

減算....2-176
減算（オーバーフローチェック付き）....2-179
減算カウンタ....2-146
減算カウンタグループ....2-148

こ

固定小数点数→文字列変換....2-453
固定長 10 進文字列変換....2-448
固定長 16 進文字列変換....2-448

さ

最小値検索....2-347
最大値検索....2-347
先入れ先出し....2-509
サム値算出....2-538
左右シフトレジスタ....2-392

し

時間加算....2-599
時間切り捨て....2-658
時間減算....2-605
時間乗算....2-615
時間除算....2-617
時間→ナノ秒変換....2-639
時間→秒変換....2-640
時刻切り捨て....2-666
時刻結合....2-656
時刻減算....2-609

時刻取得.....2-625
時刻と時間の加算.....2-601
時刻と時間の減算.....2-607
時刻→秒変換.....2-631
時刻分解.....2-654
時刻→文字列変換.....2-461
自然指数演算.....2-211
自然対数演算.....2-207
自タスク状態読出.....2-1350
自タスク設定周期読出.....2-1353
実数(LREAL) →文字列変換書式付き.....2-296
実数(REAL) →文字列変換書式付き.....2-290
実数切り上げ.....2-319
実数切り捨て.....2-319
実数→実数変換グループ.....2-282
実数小数部抽出.....2-235
実数剰余.....2-233
実数→整数変換グループ.....2-277
実数チェック.....2-237
実数の仮数・指数分離.....2-443
実数→ビット列変換グループ.....2-280
実数変換グループ.....2-313
実数丸め.....2-319
実数→文字列変換グループ.....2-288
シフトレジスタ.....2-390
時分割比例出力.....2-733
シャットダウン.....2-1340
ジャンプ.....2-84
週取得.....2-652
出力.....2-23
上下限警報グループ.....2-750
上下限偏差警報グループ.....2-754
上下限リミット制限.....2-341
乗算.....2-183
乗算（オーバーフローチェック付き）.....2-186
剰余.....2-193
常用対数演算.....2-207
初期化.....2-375
除算.....2-190
シリアル制御信号の ON/OFF 切り替え.....2-1206
シリアルラインモニタの開始.....2-1222
シリアルラインモニタの停止.....2-1226

す

スケール変換.....2-774
スタックデータ格納.....2-500
スタックデータ削除.....2-514
スタックデータ挿入.....2-512
ステッププログラム.....2-777

せ

生産情報.....22
整数→実数変換グループ.....2-268
整数→整数変換グループ.....2-262
整数→ビット列変換グループ.....2-265
整数変換グループ.....2-309
整数→文字列変換グループ.....2-284
整数→列挙型変換.....2-317
積算タイマ.....2-137
絶対値.....2-195
セット.....2-60
セット優先キープ.....2-57
選択.....2-28

た

待機シーケンス付き上下限偏差警報グループ.....2-758
タイムスタンプデジタル出力書込.....2-1320
タイムスタンプデジタル出力配列書込.....2-1325
タスク間排他ロック.....2-1357
タスク間排他ロック解除.....2-1357
タスク実行中判定.....2-1355
立ち上がり微分.....2-46
立ち下がり微分.....2-46
多ビットセット.....2-63
多ビット転送.....2-363
多ビットリセット.....2-63
多方向分岐.....2-32

つ

月の日数取得.....2-646

て

ディジット転送.....2-361
ディレクトリ削除.....2-1303
ディレクトリ作成.....2-1300
デクリメント.....2-217
デッドゾーン制御.....2-345
デッドバンド制御.....2-343
デリミタ付加文字列書込.....2-575
転送.....2-356
データ交換.....2-369
データトレースサンプリング.....2-802
データトレースステータス読出.....2-808
データトレーストリガ条件成立.....2-805
テーブル比較.....2-114

な

ナノ秒→時間変換.....2-642

に

日時→文字列変換.....2-458
日数→月変換.....2-648
任意 Explicit メッセージ送信(Class3).....2-1018
任意 Explicit メッセージ送信(UCMM).....2-1036

は

排他的論理和.....2-324
排他的論理和否定.....2-327
バイトスワップ.....2-407
バイト単位データ結合グループ.....2-476
バイト単位データ分離.....2-474
バイト配列からの変換.....2-484

バイト配列への変換.....2-478
配列一括 BCD 変換.....2-257
配列一括加算.....2-221
配列一括減算.....2-225
配列一括比較 EQ.....2-117
配列一括比較 GE.....2-119
配列一括比較 GT.....2-119
配列一括比較 LE.....2-119
配列一括比較 LT.....2-119
配列一括比較 NE.....2-117
配列一括符号なし整数変換.....2-259
配列開始番号取得.....2-495
配列検索.....2-352
配列終了番号取得.....2-495
配列データ交換.....2-371
配列の N 要素左シフト.....2-395
配列の N 要素右シフト.....2-395
配列の転送.....2-373
配列排他的論理和.....2-331
配列排他的論理和否定.....2-331
配列変数の最小値検索.....2-349
配列変数の最大値検索.....2-349
配列→文字列変換.....2-468
配列要素加算.....2-223
配列要素減算.....2-227
配列要素数取得.....2-489
配列要素の標準偏差.....2-231
配列要素の平均値算出.....2-229
配列要素比較 EQ.....2-121
配列要素比較 GE.....2-123
配列要素比較 GT.....2-123
配列要素比較 LE.....2-123
配列要素比較 LT.....2-123
配列要素比較 NE.....2-121
配列論理積.....2-331
配列論理和.....2-331
バッファクリア.....2-1213
パルス出力.....2-134
範囲型変数チェック.....2-1348
範囲指定レコード検索.....2-520
バージョン.....22

ひ

比較.....2-110
比較 EQ.....2-96
比較 GE.....2-100
比較 GT.....2-100
比較 LE.....2-100
比較 LT.....2-100
比較 NE.....2-98
日付減算.....2-610
日付時刻からの時刻切出.....2-621
日付時刻からの日付切出.....2-623
日付時刻切り捨て.....2-662
日付時刻減算.....2-611
日付時刻と時間の加算.....2-603
日付時刻と時間の減算.....2-613
日付時刻→秒変換.....2-627
日付と時刻の結合.....2-619
日付→秒変換.....2-629
日付→文字列変換.....2-460
ビットエンコーダ.....2-413
ビットカウンタ.....2-415
ビット行→ビット列変換.....2-418
ビット選択.....2-336
ビットデコーダ.....2-410
ビットテスト.....2-50
ビットテスト否定.....2-50
ビット転送.....2-359
ビットパターンコピー （実数→ビット列）グループ.2-385
ビットパターンコピー （実数→符号付き整数）グループ.....2-386
ビットパターンコピー（ビット列→実数）グループ...2-379
ビットパターンコピー（ビット列→符号付き整数）グループ.....2-377
ビットパターンコピー （符号付き整数→実数）グループ.....2-383
ビットパターンコピー （符号付き整数→ビット列）グループ.....2-381
ビット反転.....2-329
ビット列→実数変換グループ.....2-275
ビット列→整数変換グループ.....2-270
ビット列→ビット行変換グループ.....2-416
ビット列→ビット列変換グループ.....2-273
ビット列変換グループ.....2-311
ビット列→文字列変換グループ.....2-286
否定出力.....2-23
秒→時間変換.....2-643
秒→時刻変換.....2-637
秒→日付時刻変換.....2-633
秒→日付変換.....2-635

ふ

ファイルオープン.....2-1242
ファイル書込.....2-1260
ファイルクローズ.....2-1246
ファイルコピー.....2-1284
ファイル削除.....2-1291
ファイルシーク.....2-1249
ファイル名変更.....2-1295
ファイル読出.....2-1252
符号付き BCD→符号付き整数変換.....2-251
符号付き整数→BCD 変換グループ.....2-254
符号なし整数→BCD 変換グループ.....2-245
符号反転.....2-408
復帰.....2-43, 2-71
プログラム実行指令.....2-864
プログラム状態読出.....2-889
プログラム停止指令.....2-872
ブロック設定.....2-367

へ

平方根演算.....2-205
べき乗演算.....2-213
変数書込 (Class3 Explicit メッセージ).....2-1012
変数書込 (UCMM Explicit メッセージ).....2-1030

変数ファイル書込.....2-1232
変数ファイル読出.....2-1237
変数読出 (Class3 Explicit メッセージ).....2-1007
変数読出 (UCMM Explicit メッセージ).....2-1025

ま

マスタコントロール開始..... 2-72
マスタコントロール終了..... 2-72
マルチプレクサ..... 2-338

む

無手順データの受信..... 2-1161
無手順データの送信..... 2-1150

め

メモリコピー..... 2-365

も

文字コード変換(SJIS→UTF-8)..... 2-427
文字コード変換(UTF-8→SJIS)..... 2-425
文字列・ 大文字変換..... 2-571
文字列書込.....2-1275
文字列・ クリア..... 2-570
文字列・検索..... 2-560
文字列→固定小数点数変換..... 2-455
文字列・ 小文字変換..... 2-571
文字列・削除..... 2-565
文字列→実数変換グループ..... 2-306
文字列→整数変換グループ..... 2-302
文字列・挿入..... 2-567
文字列・置換..... 2-563
文字列・ 長さ検出..... 2-562
文字列・ 任意位置からの取出.....2-558
文字列・ バイト数取得.....2-569
文字列→配列変換.....2-466
文字列比較 EQ.....2-103
文字列比較 GE.....2-107
文字列比較 GT..... 2-107
文字列比較 LE..... 2-107
文字列比較 LT.....2-107
文字列比較 NE.....2-105
文字列・ 左からの取出.....2-556
文字列・ 左トリム..... 2-573
文字列→ビット列変換グループ..... 2-304
文字列・ 右からの取出.....2-556
文字列・ 右トリム..... 2-573
文字列読出.....2-1268
文字列読出デリミタ削除.....2-586
文字列・連結..... 2-554
モーション制御異常解除.....2-827
モーション制御異常ステータス取得.....2-832

ゆ

ユーザ異常解除..... 2-816
ユーザ異常ステータス取得.....2-818
ユーザ異常発生..... 2-811
ユーザ情報発生..... 2-838

よ

曜日取得.....2-650

ら

ラジアン→角度変換.....2-197
乱数発生.....2-219

り

リセット..... 2-60
リセット優先キープ.....2-54
領域比較.....2-112

る

ループブレイク.....2-40, 2-93

れ

レコード検索..... 2-516
レコード最小値検索.....2-533
レコード最大値検索.....2-533
レコード数取得..... 2-531
レコードソート..... 2-525
列挙型→整数変換.....2-315

ろ

論理積.....2-324
論理和.....2-324
ロード..... 2-14
ロードノット.....2-14

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

製品に関するお問い合わせ先

お客様相談室

フリー通話 **0120-919-066**（クイック　オムロン）

携帯電話の場合、

055-982-5015（有料）をご利用ください。

受付時間：9:00～17:00（土・日・12/31～1/3を除く）

オムロンFAクイックチャット

www.fa.omron.co.jp/contact/tech/chat/

技術相談員にチャットでお問い合わせいただけます。（I-Webメンバーズ限定）

受付時間：平日9:00～12:00 / 13:00～17:00（土日祝日・年末年始・当社休業日を除く）

※受付時間、営業日は変更の可能性がございます。最新情報はリンク先をご確認ください。

その他のお問い合わせ：納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧いただけます。緊急時のご購入にもご利用ください。 **www.fa.omron.co.jp**

●本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。

●本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン商品のご用命は

Man. No. SBCA-437J　2024年7月現在

©OMRON Corporation 2016-2024 All Rights Reserved.
お断りなく仕様などを変更することがありますのでご了承ください