OMRON

マシンオートメーションコントローラ

NJ/NX/NYシリーズ

サンプルプログラム集

コマンドリファレンスマニュアル
基本編

形NX701-□□□□
形NX502-□□□□
形NX102-□□□□
形NX1P2-□□□□□□
形NJ501-□□□□
形NJ301-□□□□
形NJ101-□□□□
形NY532-□□□□
形NY512-□□□□

SYSMAC
always in control

SBCA-488B

おことわり

- 本資料には安全に関する注意事項を含め、使用する際の詳細内容については記載していません。必ず、本資料で使用する各機器のマニュアルや取扱説明書を入手し、「安全上の注意」、「安全上の要点」、「使用上の注意」などの安全に関する注意事項を含め、内容を確認した上で使用してください。
- 本資料の内容の一部または全部を、オムロン株式会社の許可なしに複写、複製、転載することを禁じます。
- 本資料の内容に関しては、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。
- 本資料の内容に関しては、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、お手数ですが担当者までご連絡ください。

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。

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

- 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

このサンプルプログラム集に記載している各命令は、NJ/NX/NY シリーズ共通で書かれています。
そのため、以下の点に注意してください。

- NJ/NX シリーズで使用できない命令、ならびに NY シリーズで使用できない命令も記載されています。
  命令については、『NJ/NX シリーズ コマンドリファレンスマニュアル 基本編(SBCA-468)』、および『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)』を参照してください。
- 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。
- NY シリーズ コントローラのユニットバージョンは、Ver.1.12 以降です。

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。

- NX シリーズ CPU ユニット
  - 形 NX701-□□□□
  - 形 NX502-□□□□
  - 形 NX102-□□□□
  - 形 NX1P2-□□□□□□
  - 形 NX1P2-□□□□□□1

- NJ シリーズ CPU ユニット
  - 形 NJ501-□□□□
  - 形 NJ301-□□□□
  - 形 NJ101-□□□□

- NY シリーズ コントローラ

- 形 NY532-□□□□
- 形 NY512-□□□□

目次構成

1 各命令のサンプルプログラム

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

目次構成 **3**

改訂履歴 **8**

第 1 章　各命令のサンプルプログラム

タイマ命令 **1-3**
TON 1-4

算術命令 **1-7**
ADD, + ... 1-8
AddOU, +OU 1-10
SUB, - 1-11
SubOU, -OU 1-12
MUL, * 1-13
MulOU, *OU 1-15
DIV, / 1-16
SIN / COS / TAN 1-17
ASIN / ACOS / ATAN 1-18
LN / LOG 1-19
Inc / Dec 1-20

BCD 変換命令 **1-21**
BCDsToBin 1-22
BinToBCDs_** 1-23

データ型変換命令 **1-25**
RealToFormatString 1-26
LrealToFormatString 1-30
STRING_TO_**（文字列→実数変換グループ） 1-34
EnumToNum 1-36
NumToEnum 1-38

シフト命令 **1-41**
ArySHL / ArySHR 1-42
SHL / SHR 1-43
NSHLC / NSHRC 1-45
ROL / ROR 1-46

データ変換命令 **1-49**
Gray 1-50
PWLApprox / PWLApproxNoLineChk 1-51
PWLLineChk 1-53
MovingAverage 1-55
NumToDecString / NumToHexString 1-58
GrayToBin_** / BinToGray_** 1-60
PackWord 1-62
PackDword 1-63
LOWER_BOUND / UPPER_BOUND 1-65

スタック／テーブル命令 **1-69**
StackPush 1-70
StackFIFO / StackLIFO 1-77
StackIns 1-78
RecSort 1-79
RecMax / RecMin 1-83

文字列命令 1-85
LEFT / RIGHT 1-86
ToUCase / ToLCase 1-88
TrimL / TrimR 1-89
AddDelimiter 1-90
SubDelimiter 1-97

時間／時刻命令 1-103
ADD_TIME 1-105
ADD_TOD_TIME 1-106
ADD_DT_TIME 1-107
SUB_TIME 1-108
SUB_TOD_TIME 1-109
SUB_TOD_TOD 1-110
SUB_DATE_DATE 1-111
SUB_DT_DT 1-112
SUB_DT_TIME 1-113
MULTIME 1-114
DIVTIME 1-115
CONCAT_DATE_TOD 1-116
DT_TO_TOD 1-117
DT_TO_DATE 1-118
SetTime 1-119
GetTime 1-120
DtToSec 1-121
DateToSec 1-122
TodToSec 1-123
SecToDt 1-124
SecToDate 1-125
SecToTod 1-126
TimeToNanoSec 1-127
TimeToSec 1-128
NanoSecToTime 1-129
SecToTime 1-130
ChkLeapYear 1-131
GetDaysOfMonth 1-132
DaysToMonth 1-134
GetDayOfWeek 1-135
TruncTime 1-136
TruncDt 1-139
TruncTod 1-142

アナログ制御命令 1-145
PIDAT 1-146
PIDAT_HeatCool 1-152
TimeProportionalOut 1-165
LimitAlarm_** 1-178
LimitAlarmDv_** 1-179
LimitAlarmDvStbySeq_** 1-180
ScaleTrans 1-192
AC_StepProgram 1-194

システム制御命令 1-205
TraceSamp 1-206
TraceTrig 1-208
GetTraceStatus 1-209
SetAlarm 1-211
ResetAlarm 1-214
ResetPLCError 1-215
ResetMCError 1-218
GetMCError 1-222
ResetECError 1-223
GetECError 1-224
GetNXUnitError 1-225
ResetUnit 1-227
RestartNXUnit 1-230

NX_ChangeWriteMode 1-231
NX_SaveParam 1-232
PLC_ReadTotalPowerOnTime 1-233
NX_ReadTotalPowerOnTime 1-235

プログラム制御命令 1-239
PrgStart 1-240
PrgStop 1-241
PrgStatus 1-252

EtherCAT 通信命令 1-255
EC_CoESDORead 1-256
EC_StartMon 1-259
EC_StopMon 1-264
EC_SaveMon 1-265
EC_CopyMon 1-266
EC_DisconnectSlave 1-267
EC_ConnectSlave 1-273
EC_ChangeEnableSetting 1-274
NX_WriteObj 1-289
NX_ReadObj 1-299

IO-Link 通信命令 1-303
IOL_ReadObj 1-304
IOL_WriteObj 1-308

EtherNet/IP 通信命令 1-313
CIPOpen 1-315
CIPRead 1-322
CIPWrite 1-323
CIPSend 1-324
CIPClose 1-325
CIPUCMMRead 1-326
CIPUCMMWrite 1-327
CIPUCMMSend 1-328
SktUDPCreate 1-334
SktUDPRcv 1-340
SktUDPSend 1-341
SktTCPAccept 1-342
SktTCPConnect 1-343
SktTCPRcv 1-349
SktTCPSend 1-350
SktGetTCPStatus 1-351
SktClose 1-352
SktClearBuf 1-353
SktSetOption 1-354
SktTLSConnect 1-357
SktTLSRead 1-364
SktTLSWrite 1-365
SktTLSDisconnect 1-366
SktTLSClearBuf 1-367
SktTLSStopLog 1-368
ModbusTCPCmd 1-370
ModbusTCPRead 1-374
ModbusTCPWrite 1-378
ChangeIPAdr 1-381
ChangeFTPAccount 1-385
ChangeNTPServerAdr 1-387
FTPGetFileList 1-389
FTPGetFile 1-395
FTPPutFile 1-396
FTPRemoveFile 1-401
FTPRemoveDir 1-406

シリアル通信命令 1-407
ExecPMCR 1-408
SerialSend 1-415

SerialRcv / SerialRcvNoClear 1-419
SendCmd 1-424
NX_SerialSend 1-430
NX_SerialRcv 1-435
NX_ModbusRtuCmd 1-440
NX_ModbusRtuRead 1-443
NX_ModbusRtuWrite 1-446
NX_SerialSigCtl 1-449
NX_SerialSigRead 1-453
NX_SerialStatusRead 1-454
NX_SerialBufClear 1-455

SD メモリカード命令 1-461
FileWriteVar 1-462
FileReadVar 1-465
FileOpen 1-468
FileClose 1-469
FileSeek 1-470
FileRead 1-471
FileWrite 1-476
FileGets 1-481
FilePuts 1-486
FileCopy 1-493
FileRemove 1-497
FileRename 1-500
DirCreate 1-503
DirRemove 1-504
BackupToMemoryCard 1-505

タイムスタンプ命令 1-513
NX_DOutTimeStamp 1-514
NX_AryDOutTimeStamp 1-517

OS 制御命令 1-519
IPC_GetOSStatus 1-520
IPC_RebootOS 1-522
IPC_Shutdown 1-524

その他の命令 1-527
ChkRange 1-528
GetMyTaskStatus 1-530
GetMyTaskInterval 1-532
Task_IsActive 1-534
Lock / Unlock 1-535
ActEventTask 1-539
GetPrgHashCode 1-542

索引

改訂履歴

改訂記号は、表表紙・裏表紙に記載されている資料番号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	2021 年 1 月	初版
B	2024 年 10 月	誤記修正

1

各命令のサンプルプログラム

この章では、NJ/NX/NY シリーズで使用できる命令の記述例やサンプルプログラムを説明します。

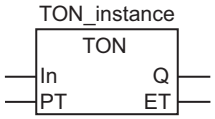
タイマ命令 1-3
算術命令 1-7
BCD 変換命令 1-21
データ型変換命令 1-25
シフト命令 1-41
データ変換命令 1-49
スタック／テーブル命令 1-69
文字列命令 1-85
時間／時刻命令 1-103
アナログ制御命令 1-145
システム制御命令 1-205
プログラム制御命令 1-239
EtherCAT 通信命令 1-255
IO-Link 通信命令 1-303
EtherNet/IP 通信命令 1-313
シリアル通信命令 1-407
SD メモリカード命令 1-461
タイムスタンプ命令 1-513
OS 制御命令 1-519
その他の命令 1-527

タイマ命令

命令	名称	ページ
TON	オンディレータイマ	P.1-4

TON

起動から設定時間経過後に TRUE を出力するタイマです。

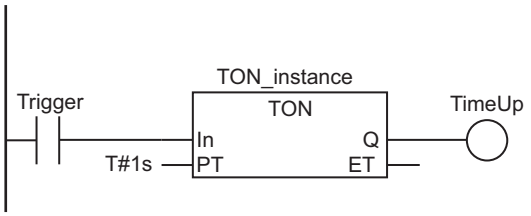
命令	名称	FB/FUN	グラフィック表現	ST 表現
TON	オンディレータイマ	FB		TON_instance(In, PT, Q, ET);

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）

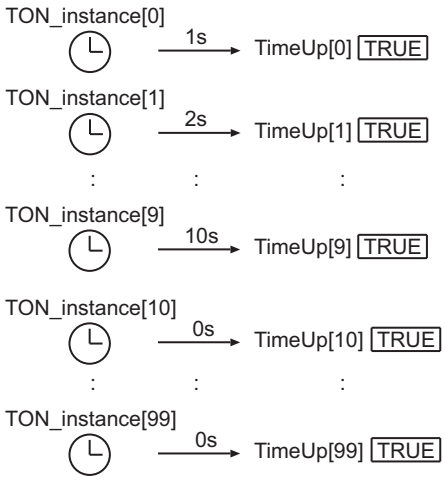
名称	データ型	初期値	コメント
Trigger	BOOL	FALSE	実行条件
TimeUp	BOOL	FALSE	タイマ出力

名称	データ型	初期値	コメント
TON_instance	TON		

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

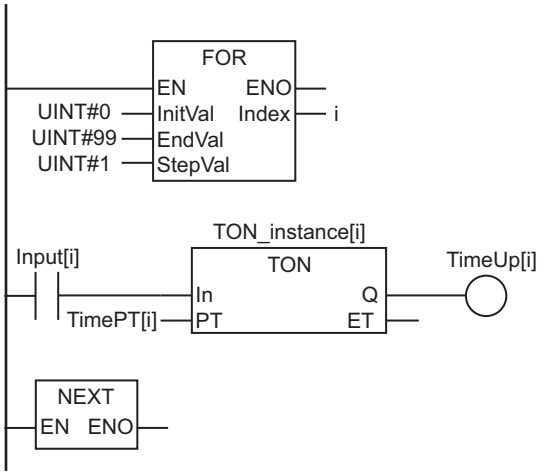
複数のオンディレータイマで時間を測る例

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
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;
```

算術命令

命令	名称	ページ
ADD, +	加算	P.1-8
AddOU, +OU	加算（オーバーフローチェック付き）	P.1-10
SUB, -	減算	P.1-11
SubOU, -OU	減算（オーバーフローチェック付き）	P.1-12
MUL, *	乗算	P.1-13
MulOU, *OU	乗算（オーバーフローチェック付き）	P.1-15
DIV, /	除算	P.1-16
SIN / COS / TAN	SIN 演算／COS 演算／TAN 演算	P.1-17
ASIN / ACOS / ATAN	SIN-1 演算／COS-1 演算／TAN-1 演算	P.1-18
LN / LOG	自然対数演算／常用対数演算	P.1-19
Inc / Dec	インクリメント／デクリメント	P.1-20

ADD, +

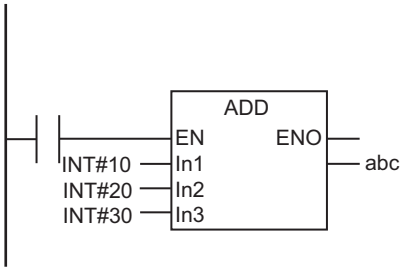
整数、実数を加算します。または、文字列を結合します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD, +	加算	FUN	(@)ADD EN ENO In1 Out : : InN (@)+ EN ENO In1 Out : : InN	Out:=In1 + In2;

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

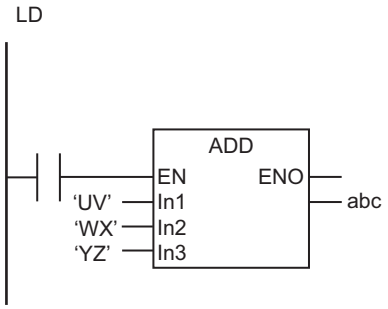
```
abc:=INT#10+INT#20+INT#30;
```

「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。

「In1」 INT#10 +「In2」 INT#20 +「In3」 INT#30 —加算→ 「Out」=abc INT#60

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。

「In1」 'UV' +「In2」 'WX' +「In3」 'YZ' → 文字列結合 →「Out」=abc 'UVWXYZ'

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。

AddOU, +OU

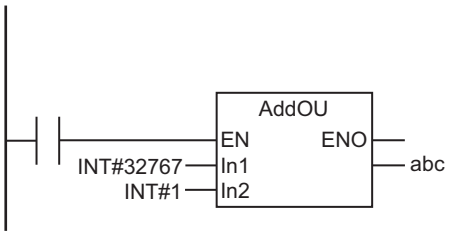
整数、実数を加算します。さらに、整数の加算結果に対して、オーバーフローのチェックを行います。

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	(@)AddOU EN ENO In1 — Out : : InN (@)+OU EN ENO In1 — Out : : InN	Out:=AddOU(In1, …, InN);

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）。

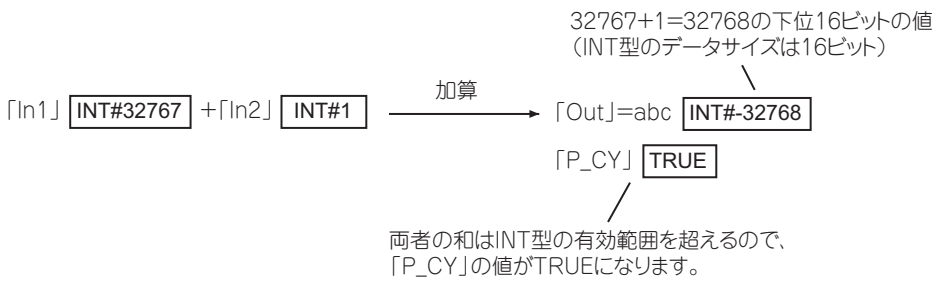
LD



ST

```
abc:=AddOU(INT#32767, INT#1);
```

「In1」～「InN」を加算します。
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。



SUB, -

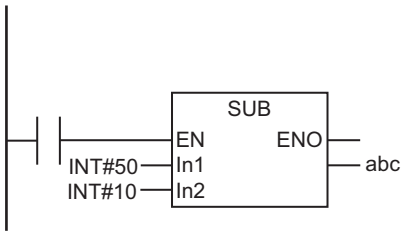
整数、実数を減算します。

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;

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。

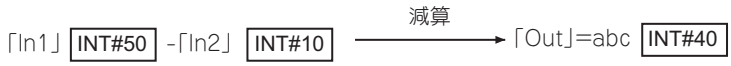
LD



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－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。

SubOU, -OU

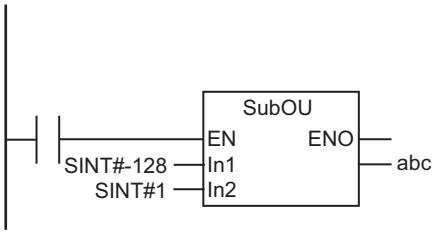
整数、実数を減算します。さらに、整数の減算結果に対して、オーバーフローのチェックを行います。

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	(@)SubOU EN ENO In1 Out In2 (@)-OU EN ENO In1 Out In2	Out:=SubOU(In1, In2);

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）。

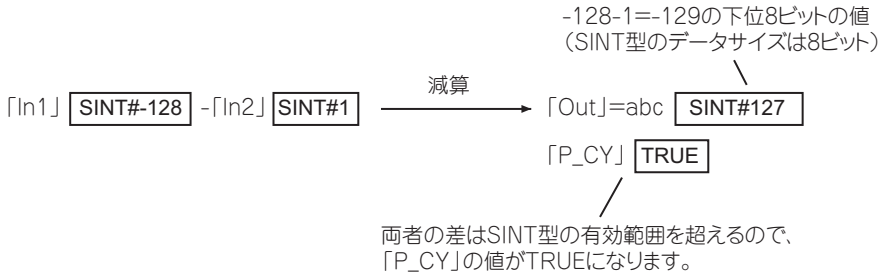
LD



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。



MUL, *

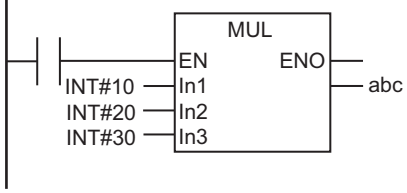
整数、実数を乗算します。

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;

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

仕様項目	ラダー図	ST
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。

MulOU, *OU

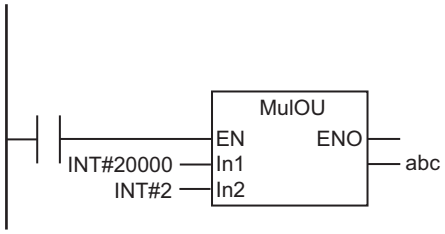
整数、実数を乗算した結果を出力します。さらに、整数の乗算結果に対して、オーバーフローのチェックを行います。

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	(@)MulOU EN ENO In1 Out : : InN (@)*OU EN ENO In1 Out : : InN	Out:=MulOU(In1, …, InN);

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）。

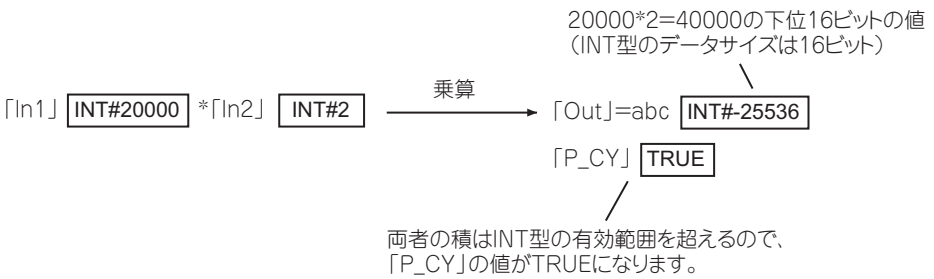
LD



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。



DIV, /

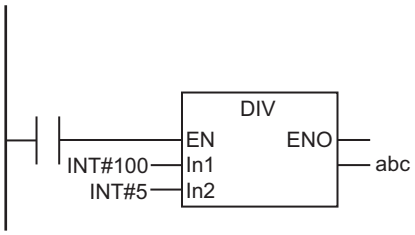
整数、実数を除算します。

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;

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。

「In1」 INT#100 /「In2」 INT#5 ——除算——→「Out」=abc INT#20

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。

SIN / COS / TAN

実数の三角関数演算を行います。

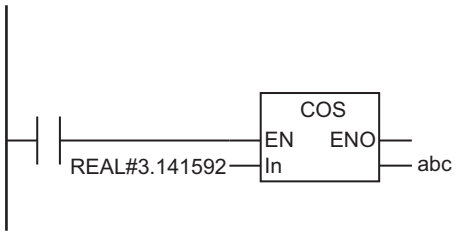
SIN　：sin（正弦値）
COS　：cos（余弦値）
TAN　：tan（正接値）

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);

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。

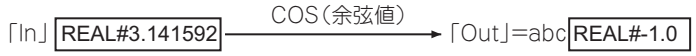
LD



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。



ASIN / ACOS / ATAN

実数の逆三角関数演算を行います。

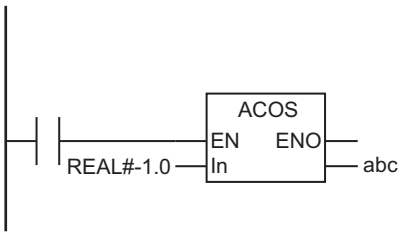
ASIN ： $\sin^{-1}$（逆正弦値）
ACOS ： $\cos^{-1}$（逆余弦値）
ATAN ： $\tan^{-1}$（逆正接値）

命令	名称	FB/FUN	グラフィック表現	ST 表現
ASIN	SIN^{-1} 演算	FUN	(@)ASIN EN ENO In Out	Out:=ASIN(In);
ACOS	COS^{-1} 演算	FUN	(@)ACOS EN ENO In Out	Out:=ACOS(In);
ATAN	TAN^{-1} 演算	FUN	(@)ATAN EN ENO In Out	Out:=ATAN(In);

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。

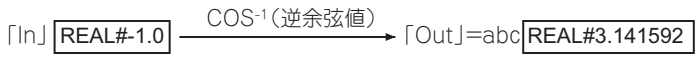
LD



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。



LN / LOG

実数の対数を求めます。

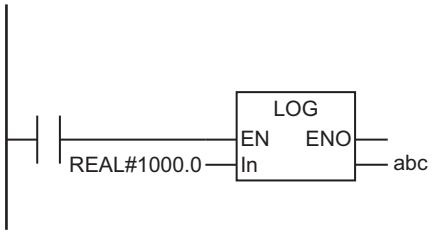
LN　：自然対数
LOG　：常用対数

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	(@)LN EN　ENO In　Out	Out:=LN(In);
LOG	常用対数演算	FUN	(@)LOG EN　ENO In　Out	Out:=LOG(In);

LOG 命令の記述例

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。

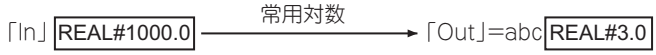
LD



ST

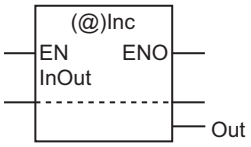
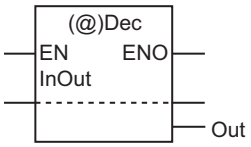
```
abc:=LOG(REAL#1000.0);
```

「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。



Inc / Dec

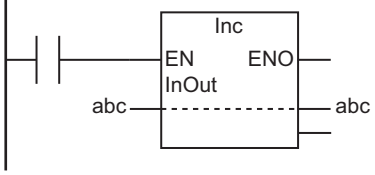
Inc ： 整数値をインクリメントします。
Dec ： 整数値をデクリメントします。

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);

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。

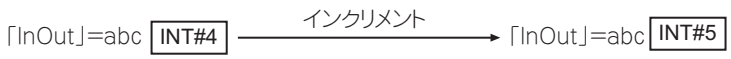
LD



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。



BCD 変換命令

命令	名称	ページ
BCDsToBin	符号付き BCD→符号付き整数変換	P.1-22
BinToBCDs_**	符号付き整数→BCD 変換グループ	P.1-23

BCDsToBin

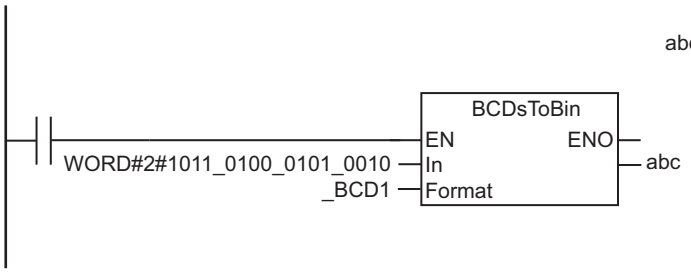
符号付き BCD のビット列を、符号付き整数に変換します。

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	(@)BCDsToBin EN ENO In Out Format	Out:=BCDsToBin(In, Format);

記述例

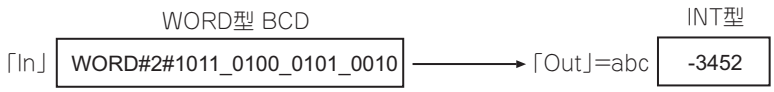
「In」=WORD#2#1011_0100_0101_0010、「Format」=_BCD1 の場合の例を、以下に示します。

LD



ST

```
abc:=BCDsToBin(WORD#2#1011_0100_0101_0010,
               _BCD1);
```



BinToBCDs_**

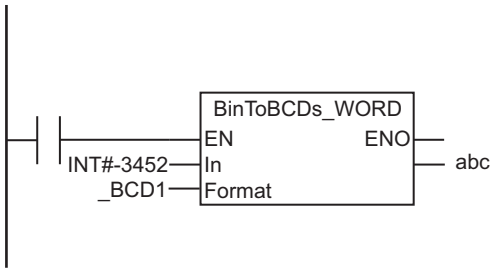
符号付き整数を、符号付き BCD のビット列に変換します。

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	(@)BinToBCDs_** EN ENO In Out Format 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

記述例

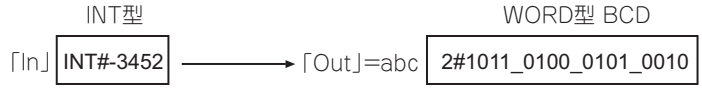
BinToBCDs_WORD 命令で、「In」=INT#-3452、「Format」=_BCD1 の場合の例を、以下に示します。

LD



ST

```
abc:=BinToBCDs_WORD(INT#-3452, _BCD1);
```



データ型変換命令

命令	名称	ページ
RealToFormatString	実数（REAL）→文字列変換書式付き	P.1-26
LrealToFormatString	実数（LREAL）→文字列変換書式付き	P.1-30
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.1-34
EnumToNum	列挙型→整数変換	P.1-36
NumToEnum	整数→列挙型変換	P.1-38

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	(@)RealToFormatString EN In Exponent Sign MinLen DecPlace ENO Out	Out:=RealToFormatString(In, Exponent, Sign, MinLen, DecPlace);

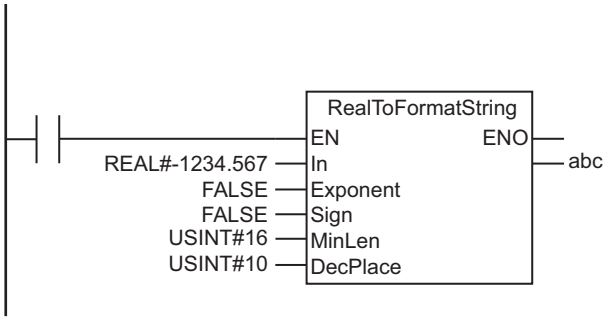
記述例

例 1

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	FALSE
MinLen	USINT#16
DecPlace	USINT#10

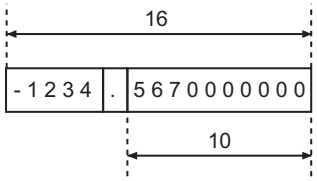
符号カラムなしの負数なので、整数部の先頭が、'-'（マイナス）になります。

LD



ST

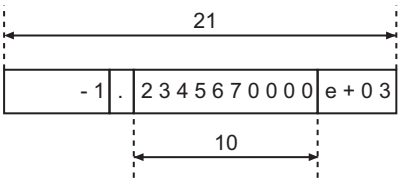
```
abc:=RealToFormatString(REAL#-1234.567, FALSE,
                        FALSE, USINT#16,
                        USINT#10);
```



例 2

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	FALSE
MinLen	USINT#21
DecPlace	USINT#10

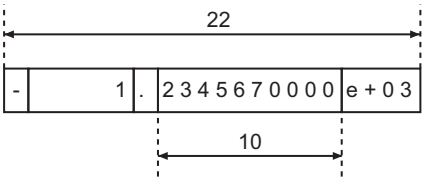
文字列のけた数より「MinLen」の方が大きいので、文字列を右詰めにし、先頭に' '（空白文字）を埋めます。



例 3

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE
MinLen	USINT#22
DecPlace	USINT#10

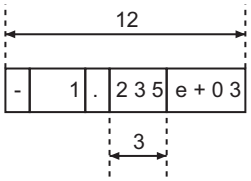
符号カラムは必ず左端です。整数部の先頭に、' '（空白文字）を埋めます。



例 4

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#3

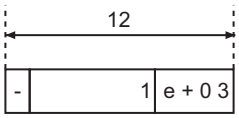
「DecPlace」=USINT#3 なので、小数第 4 位が五捨五入されます。



例 5

変数	設定内容
In	REAL#-1234.567
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#0

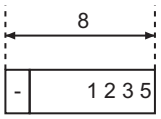
「DecPlace」=USINT#0 なので、小数第 1 位が五捨五入されます。小数点の表記もなくなります。



例 6

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	TRUE
MinLen	USINT#8
DecPlace	USINT#0

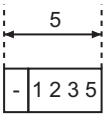
指数表記がなくなり、整数部が 4 けたになります。そのうえで、小数第 1 位が五捨五入されます。



例 7

変数	設定内容
In	REAL#-1234.567
Exponent	FALSE
Sign	TRUE
MinLen	USINT#2
DecPlace	USINT#0

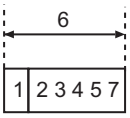
「MinLen」=USINT#2 より、「In」の整数部けた数 4 の方が大きいので、整数部は 4 けたで表記されます。



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。



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 / ENO In / Out Exponent Sign MinLen DecPlace	Out:=LrealToFormatString(In, Exponent, Sign, MinLen, DecPlace);

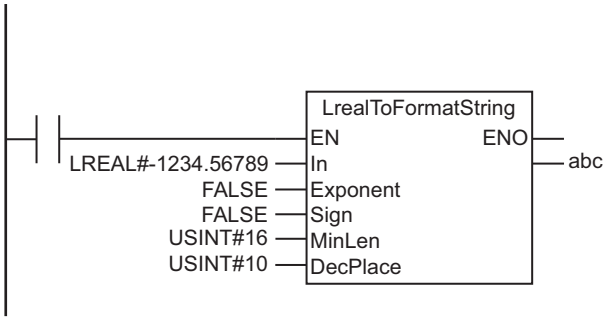
記述例

例 1

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	FALSE
MinLen	USINT#16
DecPlace	USINT#10

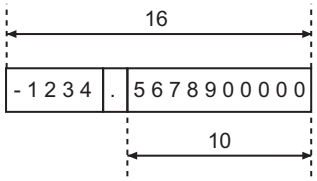
符号カラムなしの負数なので、整数部の先頭が、'-'（マイナス）になります。

LD



ST

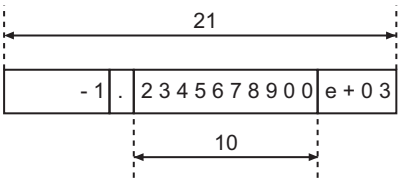
```
abc:=LrealToFormatString(LREAL#-1234.56789, FALSE,
                         FALSE, USINT#16,
                         USINT#10);
```



例 2

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	FALSE
MinLen	USINT#21
DecPlace	USINT#10

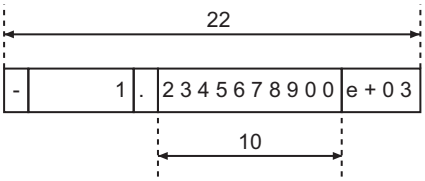
文字列のけた数より「MinLen」の方が大きいので、文字列を右詰めにし、先頭に' '（空白文字）を埋めます。



例 3

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE
MinLen	USINT#22
DecPlace	USINT#10

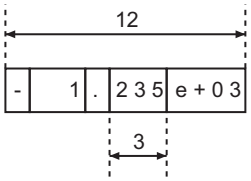
符号カラムは必ず左端です。整数部の先頭に、' '（空白文字）を埋めます。



例 4

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#3

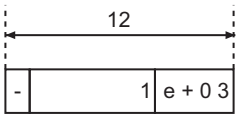
「DecPlace」=USINT#3 なので、小数第 4 位が五捨五入されます。



例 5

変数	設定内容
In	LREAL#-1234.56789
Exponent	TRUE
Sign	TRUE
MinLen	USINT#12
DecPlace	USINT#0

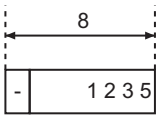
「DecPlace」=USINT#0 なので、小数第 1 位が五捨五入されます。小数点の表記もなくなります。



例 6

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	TRUE
MinLen	USINT#8
DecPlace	USINT#0

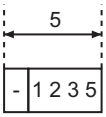
指数表記がなくなり、整数部が 4 けたになります。そのうえで、小数第 1 位が五捨五入されます。



例 7

変数	設定内容
In	LREAL#-1234.56789
Exponent	FALSE
Sign	TRUE
MinLen	USINT#2
DecPlace	USINT#0

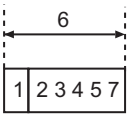
「MinLen」=USINT#2 より、「In」の整数部けた数 4 の方が大きいので、整数部は 4 けたで表記されます。



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。



STRING_TO_**（文字列→実数変換グループ）

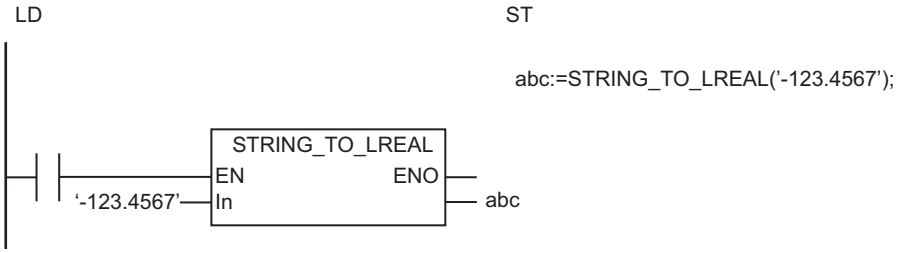
文字列を、実数に変換します。

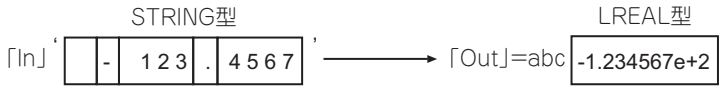
命令	名称	FB/FUN	グラフィック表現	ST 表現
STRING_TO_**	文字列→実数変換グループ	FUN	(@)STRING_TO_** EN　ENO In　Out ただし、“**”は実数のデータ型名	Out:=STRING_TO_**(In); ただし、“**”は実数のデータ型名

記述例

例 1

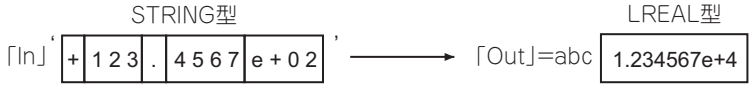
符号あり、小数点あり、小数部あり、指数部なしの場合の例を以下に示します。





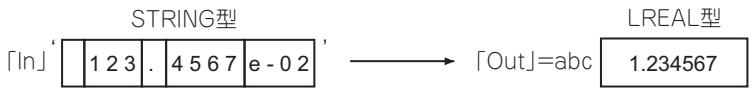
例 2

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。



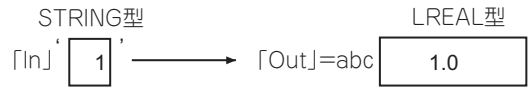
例 3

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。



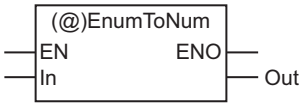
例 4

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。



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		Out:=EnumToNum(In);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

サンプルプログラム

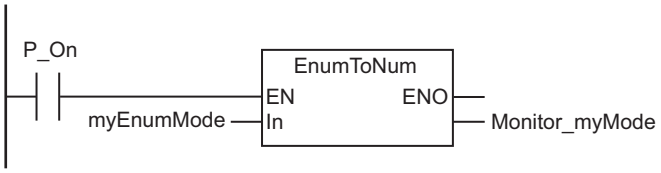
ユーザプログラムの動作モードを列挙型 EnumMode で定義しているとします。
動作モードを表示器でモニタするため、列挙型 EnumMode の変数「myEnumMode」の値を DINT 型の変数「Monitor_myMode」に変換します。たとえば、「myEnumMode」の値が mode2 の場合、「Monitor_myMode」の値は 2 となります。

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

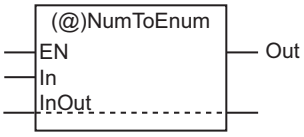
名称	データ型	初期値	コメント
myEnumMode	EnumMode	mode0	列挙型のモードの値

名称	データ型	初期値	コメント
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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

サンプルプログラム

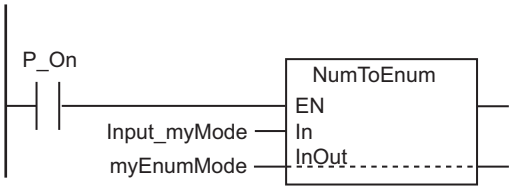
ユーザプログラムの動作モードを列挙型 EnumMode で定義しているとします。
表示器から動作モードを変更するため、DINT 型の値「Input_myMode」を書き込みます。
ユーザプログラムで、「Input_myMode」を列挙型 EnumMode の変数「myEnumMode」に変換します。たとえば、「Input_myMode」の値が「1」の場合、「myEnumMode」の値は mode1 となります。

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);
```

シフト命令

命令	名称	ページ
ArySHL / ArySHR	配列の N 要素左シフト／配列の N 要素右シフト	P.1-42
SHL / SHR	N ビット左シフト／N ビット右シフト	P.1-43
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.1-45
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.1-46

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。

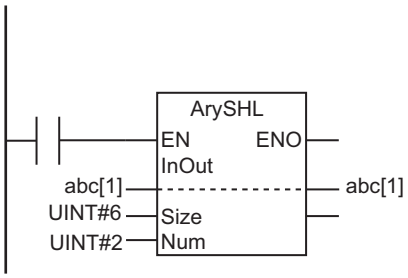
命令	名称	FB/FUN	グラフィック表現	ST 表現
ArySHL	配列の N 要素左シフト	FUN	(@)ArySHL EN ENO InOut Size Num Out	ArySHL(InOut, Size, Num);
ArySHR	配列の N 要素右シフト	FUN	(@)ArySHR EN ENO InOut Size Num Out	ArySHR(InOut, Size, Num);

ArySHL の記述例

左方向（配列の上位方向）にシフトします。

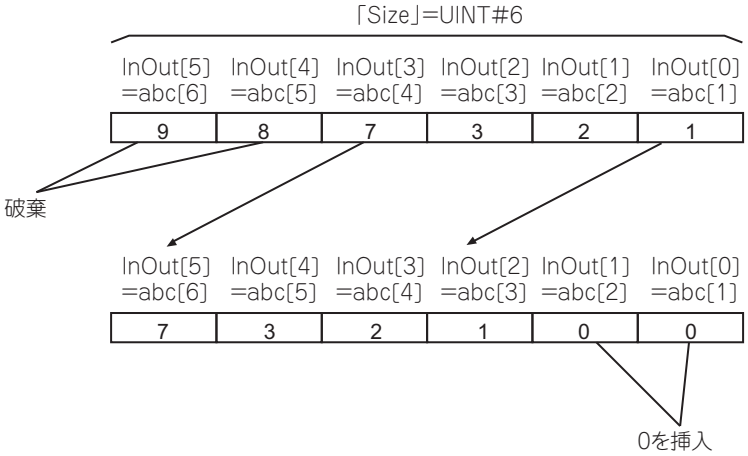
ArySHL 命令で、「Size」=UINT#6、「Num」=UINT#2 の場合の例を、以下に示します。

LD



ST

```
ArySHL(abc[1], UINT#6, UINT#2);
```



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。

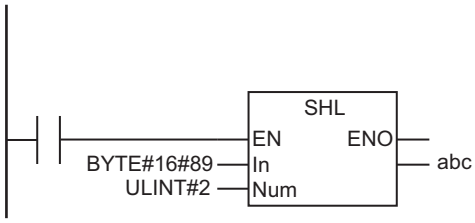
命令	名称	FB/FUN	グラフィック表現	ST 表現
SHL	N ビット左シフト	FUN	(@)SHL EN ENO In Out Num	Out:=SHL(In, Num);
SHR	N ビット右シフト	FUN	(@)SHR EN ENO In Out Num	Out:=SHR(In, Num);

SHL の記述例

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。

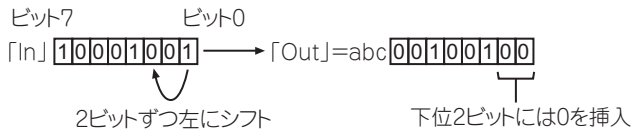
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=SHL(BYTE#16#89, ULINT#2);
```

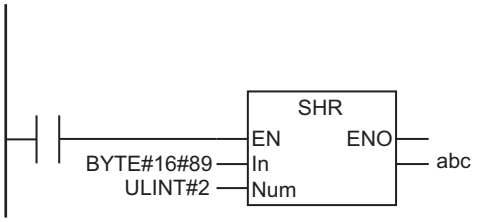


SHR の記述例

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。

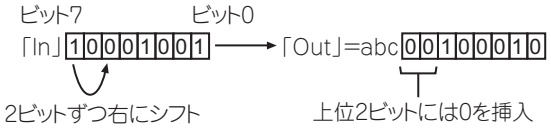
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=SHR(BYTE#16#89, ULINT#2);
```



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。

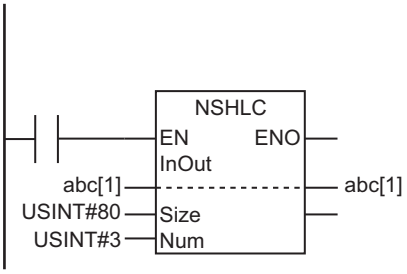
命令	名称	FB/FUN	グラフィック表現	ST 表現
NSHLC	N ビットデータ N ビット左シフト CY 付き	FUN	(@)NSHLC EN ENO InOut Size Num Out	NSHLC(InOut, Size, Num);
NSHRC	N ビットデータ N ビット右シフト CY 付き	FUN	(@)NSHRC EN ENO InOut Size Num Out	NSHRC(InOut, Size, Num);

NSHLC の記述例

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。

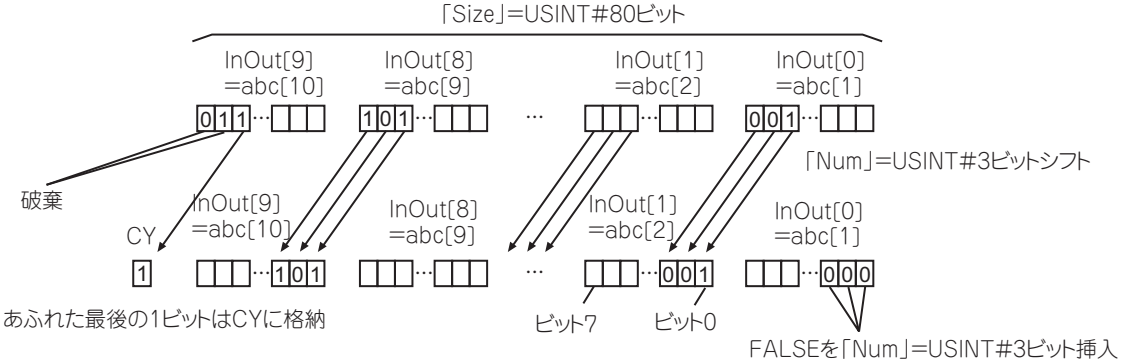
NSHLC 命令で、InOut[]が BYTE 型配列、「Size」= USINT#80、「Num」=USINT#3 の場合の例を、以下に示します。

LD



ST

```
NSHLC(abc[1], USINT#80, USINT#3);
```



ROL / ROR

ビット列を複数ビット回転します。

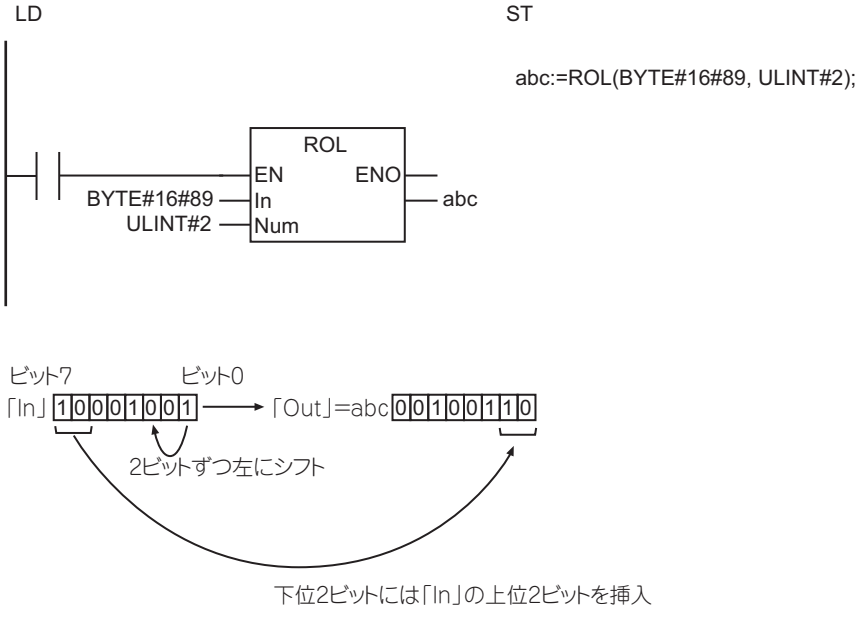
ROL ： 左方向（上位方向）に回転します。
ROR ： 右方向（下位方向）に回転します。

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	(@)ROL EN ENO In Out Num	Out:=ROL(In, Num);
ROR	N ビット右回転	FUN	(@)ROR EN ENO In Out Num	Out:=ROR(In, Num);

ROL の記述例

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。

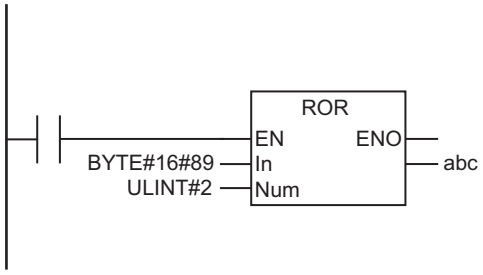


ROR の記述例

左から右（上位ビットから下位ビット）方向に回転します。

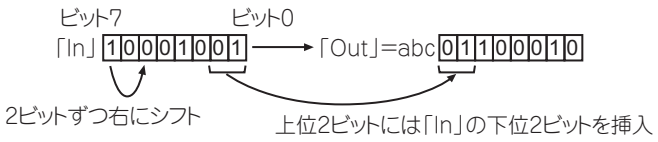
「In」=BYTE#16#89、「Num」=ULINT#2 の場合の例を、以下に示します。

LD



ST

```
abc:=ROR(BYTE#16#89, ULINT#2);
```



データ変換命令

命令	名称	ページ
Gray	グレイコード変換	P.1-50
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.1-51
PWLLineChk	折れ線データチェック	P.1-53
MovingAverage	移動平均	P.1-55
NumToDecString / NumToHexString	固定長 10 進文字列変換／固定長 16 進文字列変換	P.1-58
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.1-60
PackWord	2 バイト結合	P.1-62
PackDword	4 バイト結合	P.1-63
LOWER_BOUND / UPPER_BOUND	配列開始番号取得／配列終了番号取得	P.1-65

Gray

グレイコードを角度に変換します。

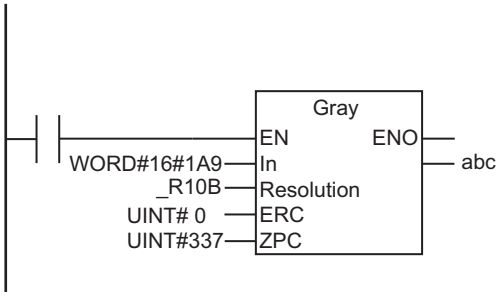
命令	名称	FB/FUN	グラフィック表現	ST 表現
Gray	グレイコード変換	FUN	(@)Gray EN ENO In Out Resolution ERC ZPC	Out:=Gray(In, Resolution, ERC, ZPC);

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。

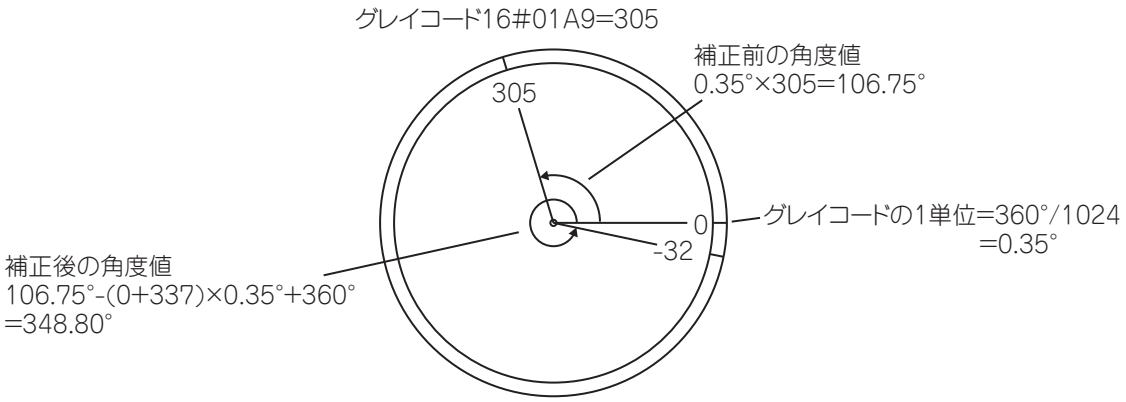
まず、分解能が 10 ビットなので、グレイコードの 1 単位は、360°/1024=0.35°となります。
グレイコード 16#01A9 に対応する 10 進数の値は、305 です。したがって、補正前の角度値は 0.35°×305=106.75°となります。
「ERC」の値が 0、「ZPC」の値が 337 です。したがって、これらを補正した角度値は 106.75°-（0+337）×0.35°=-11.20°となります。
「Out」の値の範囲は 0 以上なので、-11.20°+360°=348.80°となります。「Out」の値は LREAL#348.8 となります。

LD



ST

```
abc:=Gray(WORD#16#1A9, _R10B, UINT#0, UINT#337);
```



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	(@)PWLApprox EN　ENO In　Out Line Num	Out:=PWLApprox(In, Line, Num);
PWLApprox NoLineChk	折れ線近似変換（折れ線チェックなし）	FUN	(@)PWLApproxNoLineChk EN　ENO In　Out Line Num	Out:=PWLApproxNoLineChk(In, Line, Num);

バージョン関連情報

PWLApproxNoLineChk 命令は、CPU ユニット Ver.1.03 以降、かつ Sysmac Studio Ver.1.04 以降で、使用可能です。

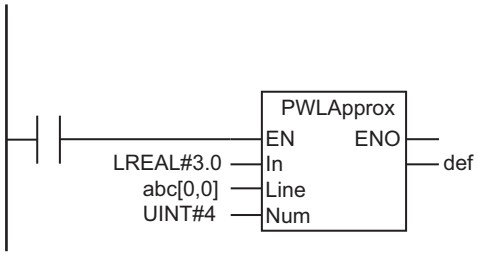
記述例

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

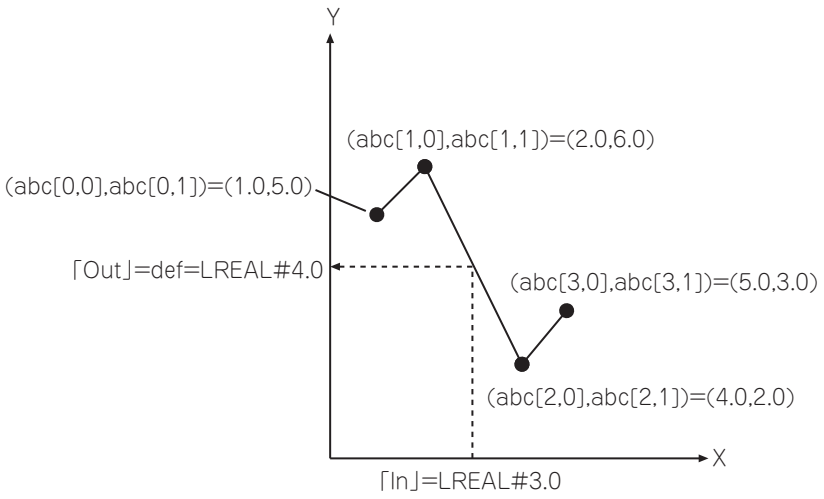
変換の結果、「Out」の値は LREAL#4.0 になります。

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.1-53)」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。

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	(@)PWLLineChk EN Line Num Out	Out:=PWLLineChk(Line, Num);

バージョン関連情報

本命令は、CPU ユニット Ver.1.03 以降、かつ Sysmac Studio Ver.1.04 以降で、使用可能です。

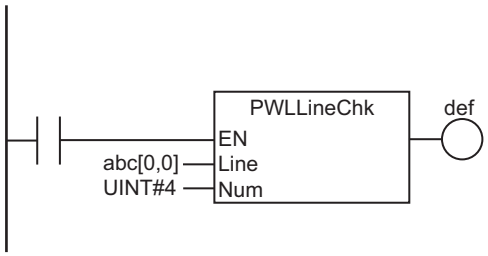
記述例

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

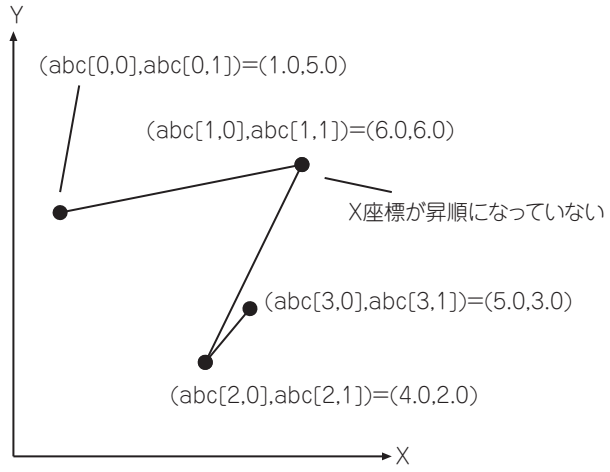
X 座標が昇順になっていないので、「Out」の値は FALSE になります。

LD



ST

```
def:=PWLLineChk(abc[0,0], UINT#4);
```


Y
(abc[0,0],abc[0,1])=(1.0,5.0)
(abc[1,0],abc[1,1])=(6.0,6.0)
X座標が昇順になっていない
(abc[3,0],abc[3,1])=(5.0,3.0)
(abc[2,0],abc[2,1])=(4.0,2.0)
X

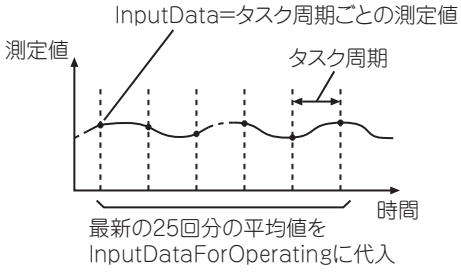
MovingAverage

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	(@)MovingAverage EN ENO In Out CurIndex Buf BufSize Q	Out:=MovingAverage(In, CurIndex, Buf, BufSize, Q);

サンプルプログラム

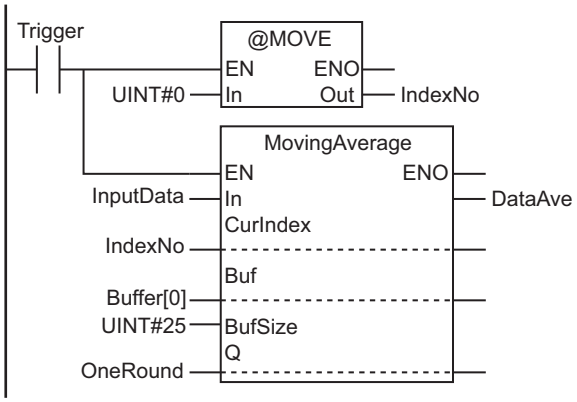
センサなどのアナログ入力データに含まれる、ノイズなどの外乱の影響を除去する例です。
入力データ「InputData」の、最新の 25 回分の平均値「DataAve」を、次工程の入力「InputDataForOperating」に代入します。
「InputData」は、実行条件「Trigger」の値が TRUE である間、タスク周期ごとに入力します。
「InputData」が 25 回以上入力されるまでは、データ量が不十分なので、平均値ではなく、最新の 1 回の「InputData」の値を、「InputDataForOperating」に代入します。
「Trigger」の値が FALSE→TRUE 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。



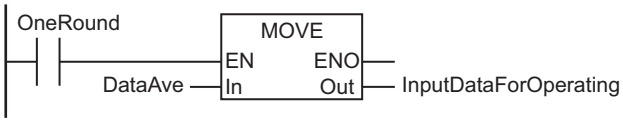
LD

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

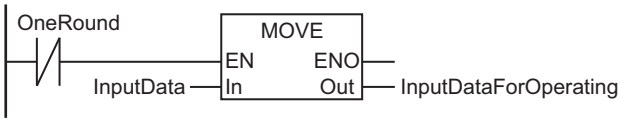
Triggerが立ち上がったら、IndexNoに、0を代入
TriggerがTRUEの間、タスク周期ごとにInputDataを入力して平均値計算



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;
```

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。

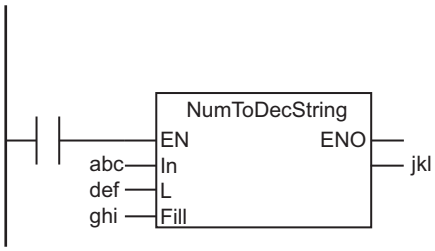
命令	名称	FB/FUN	グラフィック表現	ST 表現
NumToDecString	固定長 10 進文字列変換	FUN	(@)NumToDecString EN ENO In Out L Fill	Out:=NumToDecString(In, L, Fill);
NumToHexString	固定長 16 進文字列変換	FUN	(@)NumToHexString EN ENO In Out L Fill	Out:=NumToHexString(In, L, Fill);

NumToDecString の記述例

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 □□□□□128

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl □□□-128

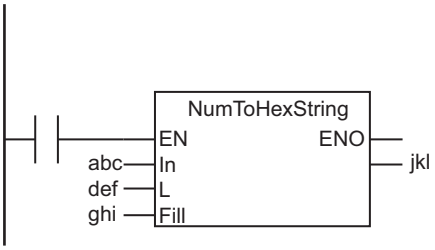
「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl -0000128

NumToHexString の記述例

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 8 0

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl F F F F F 8 0

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	(@)GrayToBin_** EN ENO In Out 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	(@)BinToGray_** EN ENO In Out 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

GrayToBin_**の記述例

グレイコードの変換対象「In」をビット列に変換します。

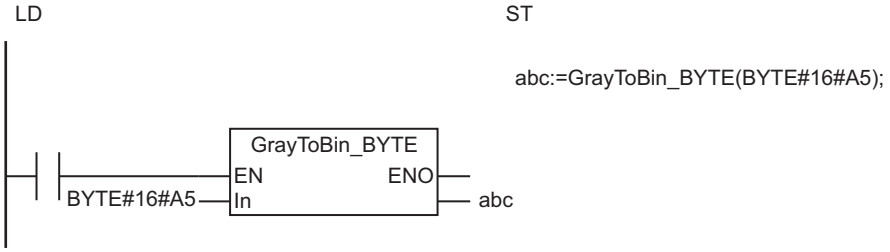
変換の手順を、「In」、「Out」が BYTE 型の場合で説明します。

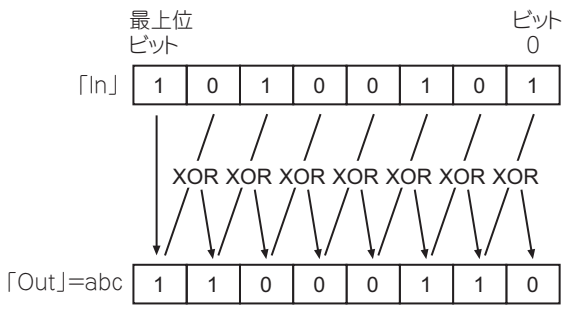
1 「In」の最上位ビット（ビット 7）を、そのまま「Out」の最上位ビット（ビット 7）とします。

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_**の記述例

ビット列の変換対象「In」をグレイコードに変換します。

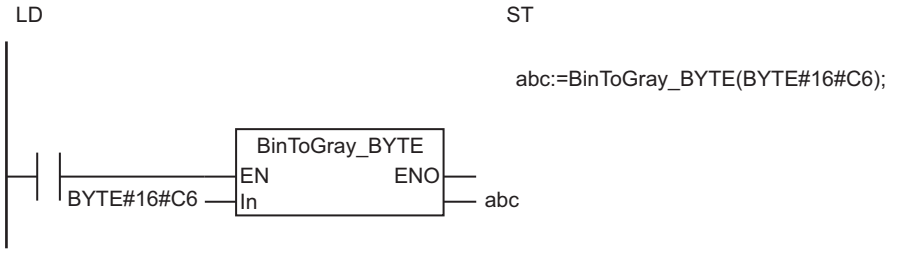
変換の手順を、「In」、「Out」が BYTE 型の場合で説明します。

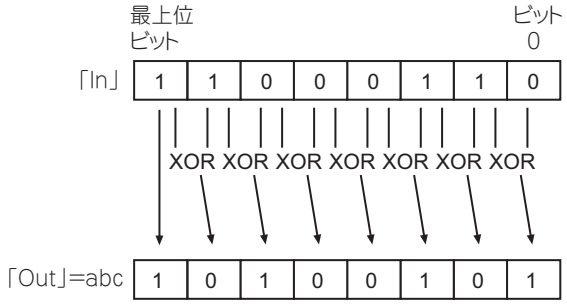
1 **「In」の最上位ビット（ビット 7）を、そのまま「Out」の最上位ビット（ビット 7）とします。**

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。





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);

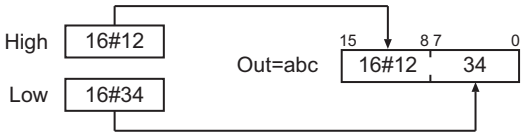
バージョン関連情報

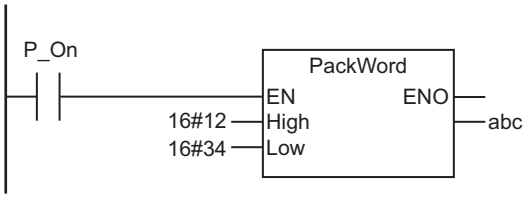
本命令は、CPU ユニット Ver.1.12 以降、かつ Sysmac Studio Ver.1.16 以降で、使用可能です。

サンプルプログラム

LD

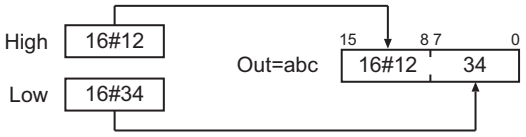
「High」=16#12、「Low」=16#34 の場合の例を、以下に示します。
変数「abc」は 16#1234 になります。





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	(@)PackDword EN ENO HighHigh Out HighLow LowHigh LowLow	Out:=PackDword(HighHigh, HighLow, LowHigh, LowLow);

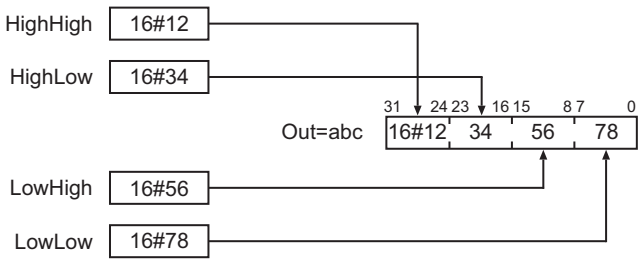
バージョン関連情報

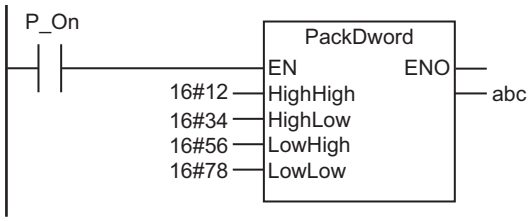
本命令は、CPU ユニット Ver.1.12 以降、かつ Sysmac Studio Ver.1.16 以降で、使用可能です。

サンプルプログラム

LD

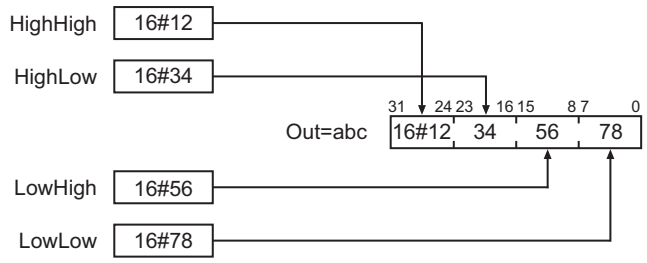
「HighHigh」=16#12、「HighLow」=16#34、「LowHigh」=16#56、「LowLow」=16#78 の場合の例を、以下に示します。
変数「abc」は 16#12345678 になります。





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。

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		

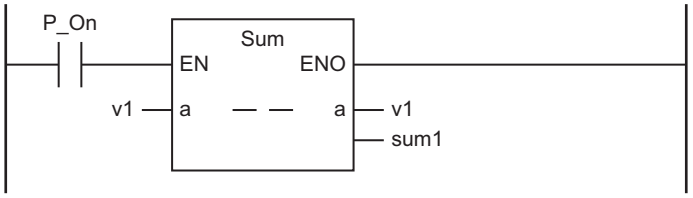
```
Sum := 0;
FOR i := LOWER_BOUND(a,1) TO UPPER_BOUND(a,1) DO
```

```
    Sum := Sum + a[i];
END_FOR;
```

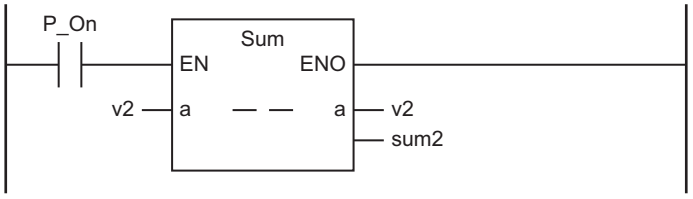
● 呼び出し側プログラム

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]	
	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.1-70
StackFIFO / StackLIFO	先入れ先出し／後入れ先出し	P.1-77
StackIns	スタックデータ挿入	P.1-78
RecSort	レコードソート	P.1-79
RecMax / RecMin	レコード最大値検索／レコード最小値検索	P.1-83

StackPush

スタックに、値を格納します。

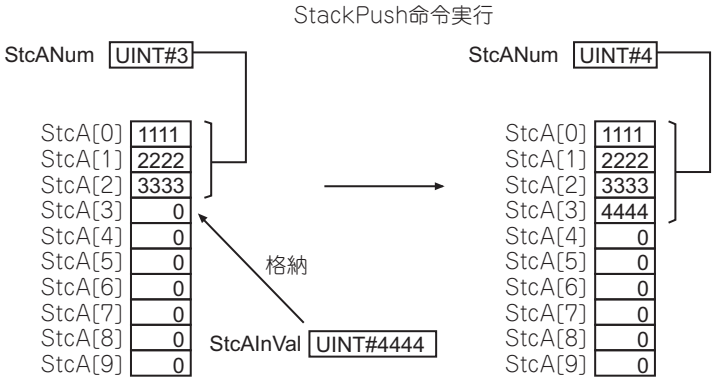
命令	名称	FB/FUN	グラフィック表現	ST 表現
StackPush	スタックデータ格納	FUN	(@)StackPush EN ENO In Out InOut Size Num	StackPush(In, InOut, Size, Num);

サンプルプログラム

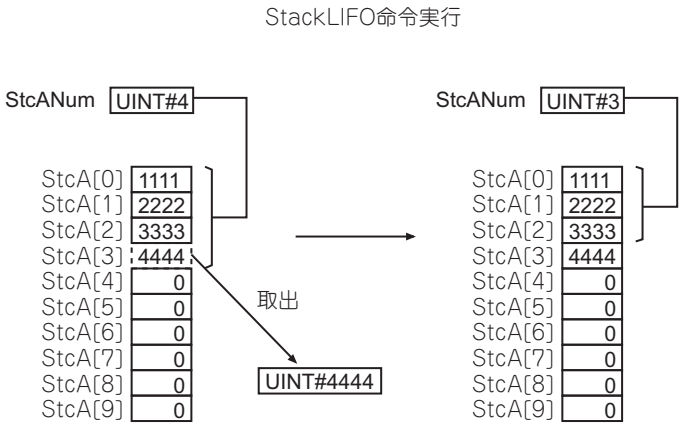
配列変数 StcA[0..9]をスタックとします。まず準備として、スタックに 3 個のデータ(UINT#1111、UINT#2222、UINT#3333)を格納しておきます。

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

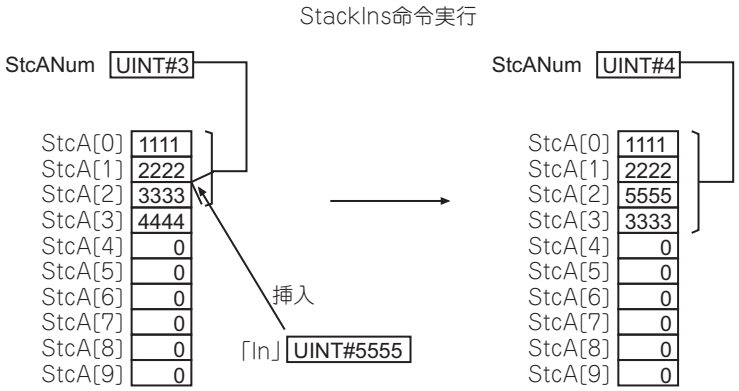
StackPush 命令を使って、スタックの最上位 StcA[3]に、新しいデータ(UINT#4444)を 1 個格納します。スタック内のデータ数は、4 個になります。



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。



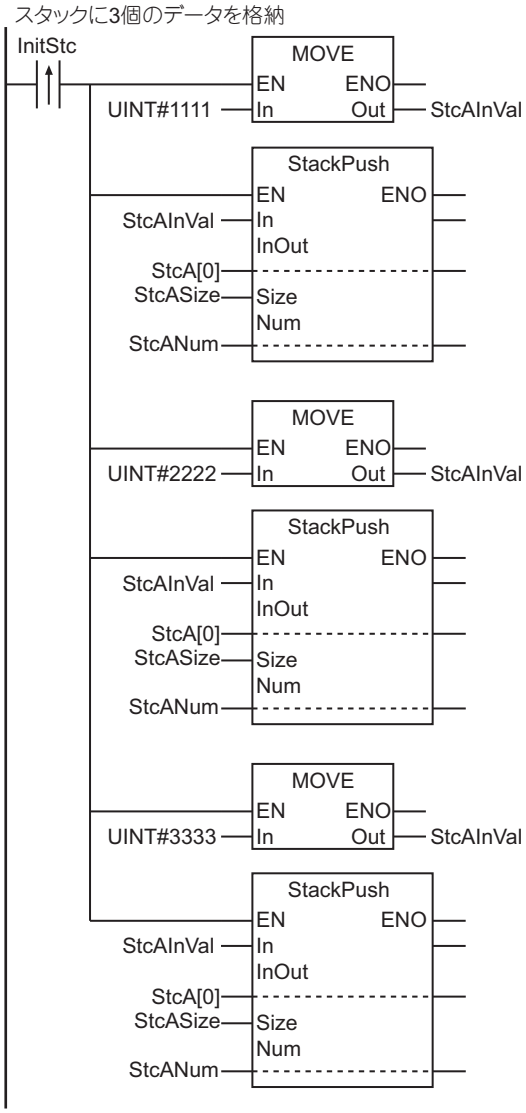
LD

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

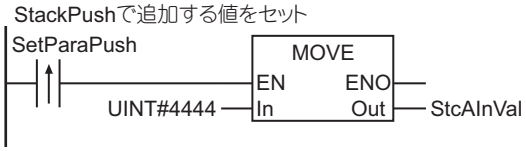
スタックの初期化



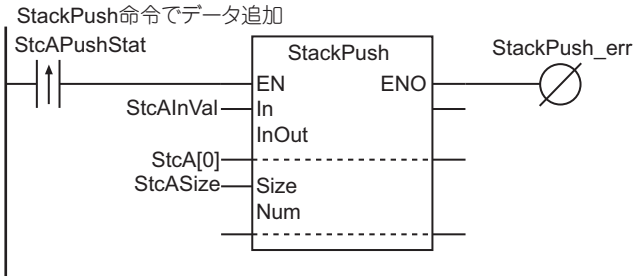
スタックに3個のデータを格納



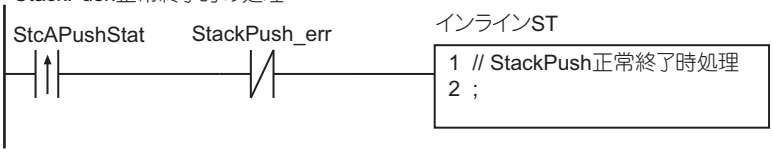
StackPushで追加する値をセット



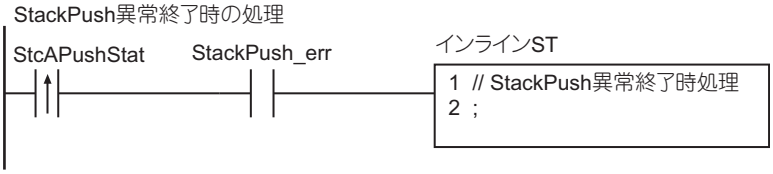
StackPush命令でデータ追加



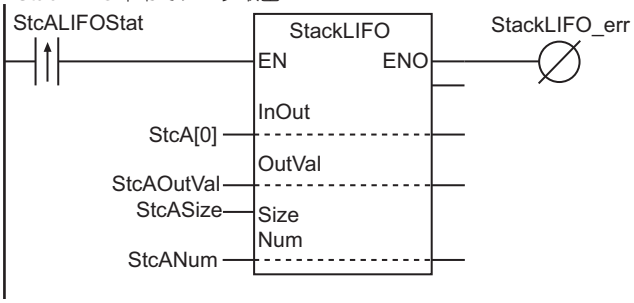
StackPush正常終了時の処理



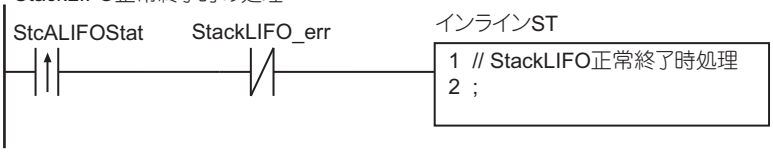
StackPush異常終了時の処理



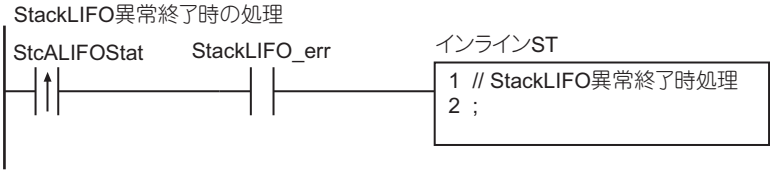
StackLIFO命令でデータ取出



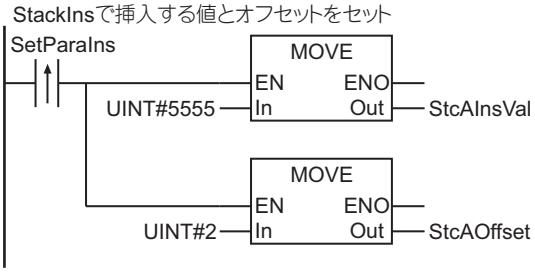
StackLIFO正常終了時の処理

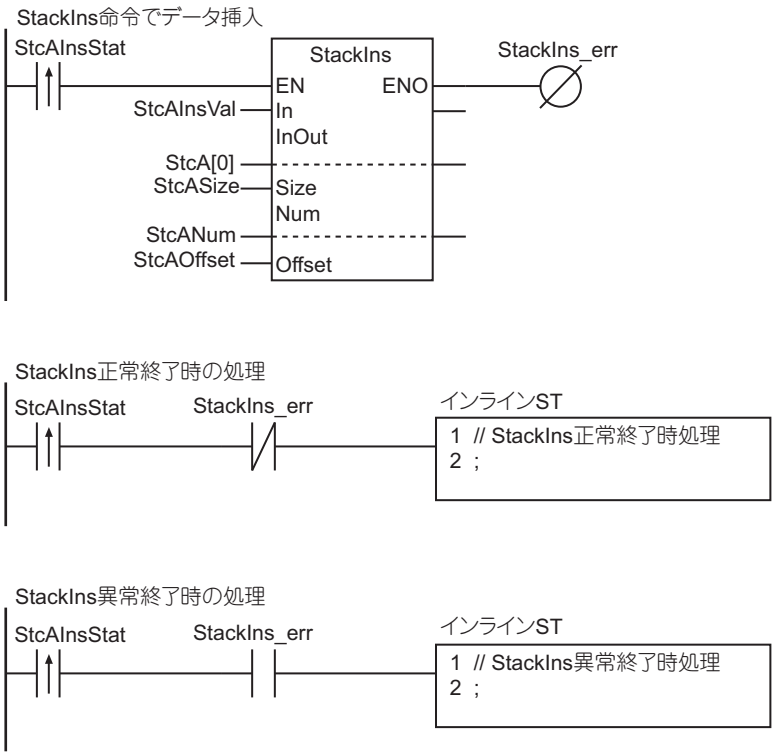


StackLIFO異常終了時の処理



StackInsで挿入する値とオフセットをセット





● 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

名称	データ型	初期値	コメント
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;

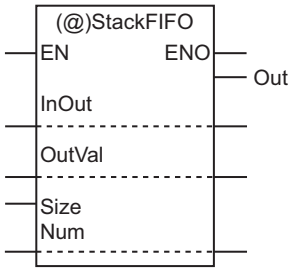
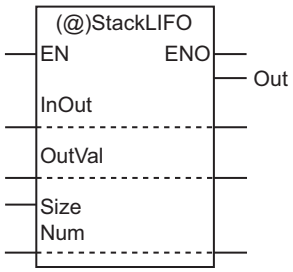
preStcAPushStat:=StcAPushStat;
// StackLIFO 命令でデータ取出
IF ( (StcALIFOStat=TRUE) AND (preStcALIFOStat=FALSE) ) THEN
  StackLIFO(
    InOut :=StcA[0], // スタック配列の先頭要素
    OutVal :=StcAOutVal, // スタックから取り出す値
    Size :=StcASize, // スタックの要素数
    Num :=StcANum, // スタックの格納要素数
    ENO =>StcALIFO_OK); // 正常終了フラグ
  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。

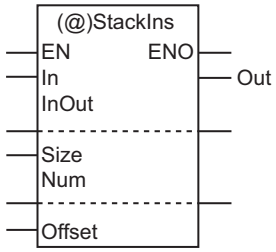
命令	名称	FB/FUN	グラフィック表現	ST 表現
StackFIFO	先入れ先出し	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	後入れ先出し	FUN		StackLIFO(InOut, OutVal, Size, Num);

サンプルプログラム

StackPush 命令の「サンプルプログラム (P.1-70)」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。

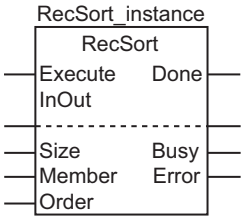
命令	名称	FB/FUN	グラフィック表現	ST 表現
StackIns	スタックデータ挿入	FUN		StackIns(In, InOut, Size, Num, Offset);

サンプルプログラム

StackPush 命令の「サンプルプログラム (P.1-70)」を、参照してください。

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);

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**

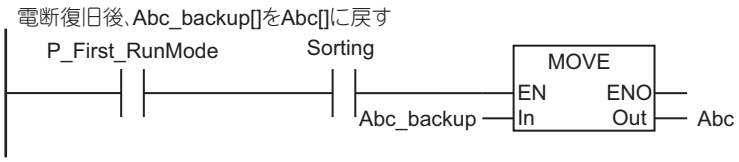
名称	データ型	初期値	保持	コメント
Abc	ARRAY[0..4] OF MyStr	[5((l:=FALSE,m:=0,n:=0.0))]	☑	ソート対象配列
Abc_backup	ARRAY[0..4] OF MyStr	[5((l:=FALSE,m:=0,n:=0.0))]	☑	Abc[]のバックアップ

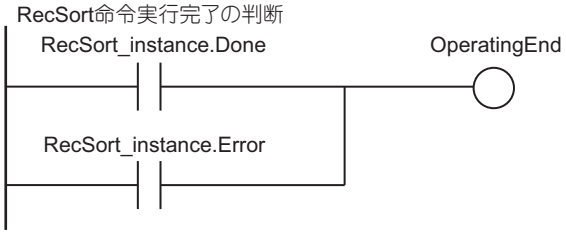
LD

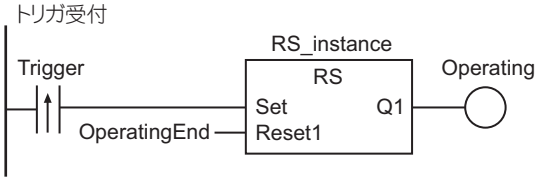
内部変数	名称	データ型	初期値	保持	コメント
	Sorting	BOOL	FALSE	☑	処理中（保持あり）
	OperatingEnd	BOOL	FALSE	☐	処理終了
	Trigger	BOOL	FALSE	☐	実行条件

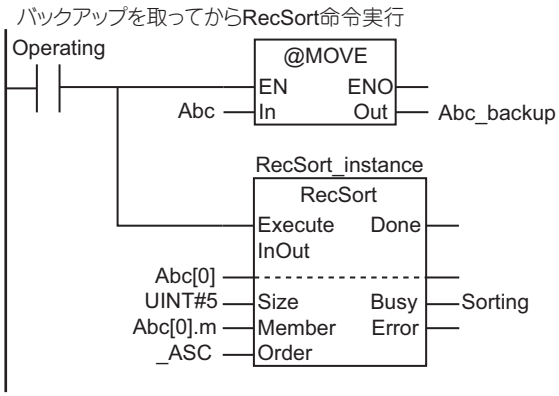
内部変数	名称	データ型	初期値	保持	コメント
	Operating	BOOL	FALSE	☐	処理中
	RS_instance	RS		☐	
	RecSort_instance	RecSort		☐	

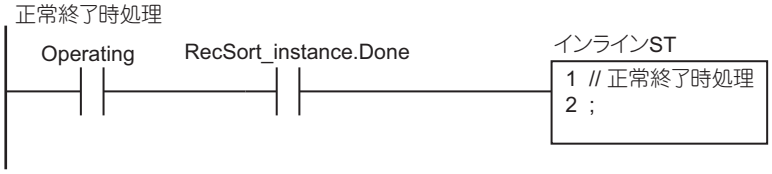
外部変数	名称	データ型	コメント
	Abc	ARRAY[0..4] OF MyStr	ソート対象配列
	Abc_backup	ARRAY[0..4] OF MyStr	Abc[]のバックアップ

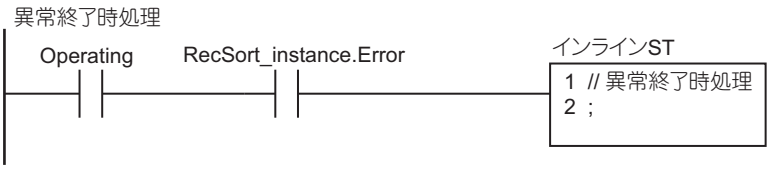












ST

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
```

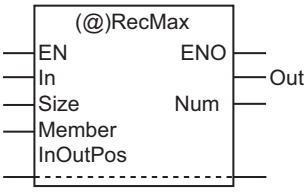
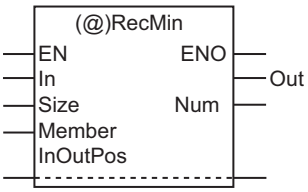
```
IF (Operating=TRUE) THEN
  RecSort_instance (
    Execute:=TRUE,
    InOut :=Abc[0],
    Size :=UINT#5,
    Member :=Abc[0].m,
    Order :=_ASC,
    Busy =>Sorting) ;

  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;
```

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。

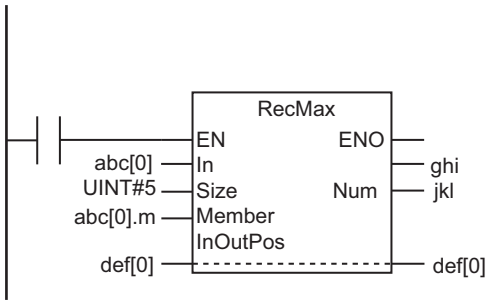
命令	名称	FB/FUN	グラフィック表現	ST 表現
RecMax	レコード最大値検索	FUN		Out:=RecMax(In, Size,Member, InOutPos, Num);
RecMin	レコード最小値検索	FUN		Out:=RecMin(In, Size, Member, InOutPos, Num);

RecMax の記述例

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。

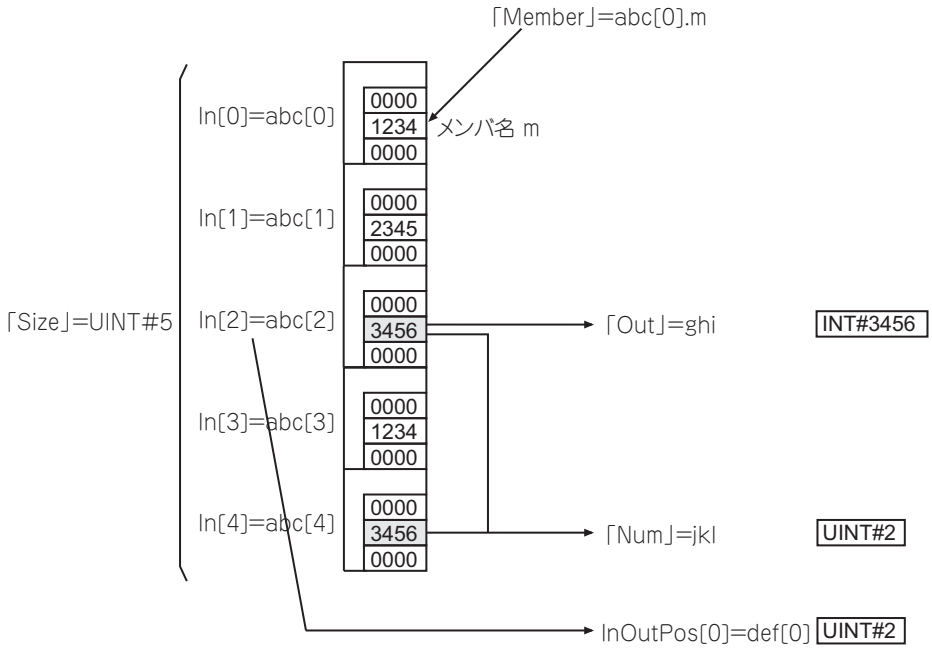
RecMax 命令で、「Size」=UINT#5 の場合の例を、以下に示します。

LD



ST

```
ghi:=RecMax(abc[0], UINT#5, abc[0].m, def[0], jkl);
```


「Member」=abc[0].m
In[0]=abc[0]
0000
1234
0000
メンバ名 m
In[1]=abc[1]
0000
2345
0000
「Size」=UINT#5
In[2]=abc[2]
0000
3456
0000
「Out」=ghi
INT#3456
In[3]=abc[3]
0000
1234
0000
In[4]=abc[4]
0000
3456
0000
「Num」=jkl
UINT#2
InOutPos[0]=def[0]
UINT#2

文字列命令

命令	名称	ページ
LEFT / RIGHT	文字列・左からの取出／文字列・右からの取出	P.1-86
ToUCase / ToLCase	文字列・大文字変換／文字列・小文字変換	P.1-88
TrimL / TrimR	文字列・左トリム／文字列・右トリム	P.1-89
AddDelimiter	デリミタ付加文字列書込	P.1-90
SubDelimiter	文字列読出デリミタ削除	P.1-97

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。

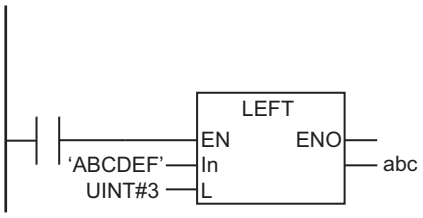
命令	名称	FB/FUN	グラフィック表現	ST 表現
LEFT	文字列・左からの取出	FUN	(@)LEFT EN　ENO In　Out L	Out:=LEFT(In, L);
RIGHT	文字列・右からの取出	FUN	(@)RIGHT EN　ENO In　Out L	Out:=RIGHT(In, L);

LEFT の記述例

「In」の左（先頭）から、取り出します。

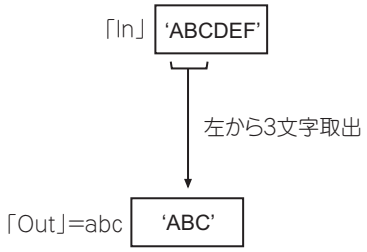
「In」='ABCDEF'、「L」=UINT#3 の場合の例を、以下に示します。変数「abc」の値は、'ABC'になります。

LD



ST

```
abc:=LEFT('ABCDEF', UINT#3);
```

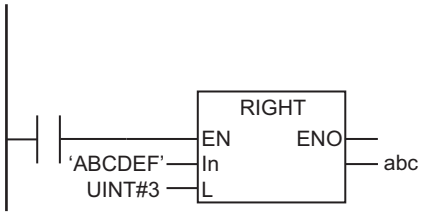


RIGHT の記述例

「In」の右（末尾）から、取り出します

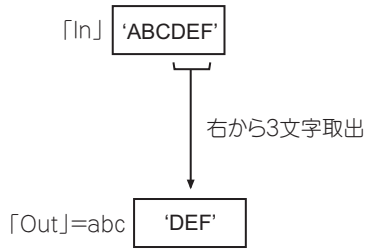
「In」='ABCDEF'、「L」=UINT#3 の場合の例を、以下に示します。変数「abc」の値は、'DEF'になります。

LD



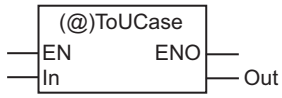
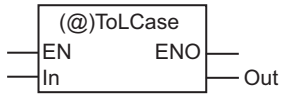
ST

```
abc:=RIGHT('ABCDEF', UINT#3);
```



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。

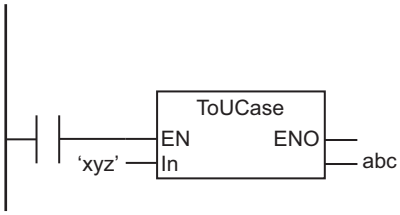
命令	名称	FB/FUN	グラフィック表現	ST 表現
ToUCase	文字列・大文字変換	FUN		Out:=ToUCase(In);
ToLCase	文字列・小文字変換	FUN		Out:=ToLCase(In);

ToUCase の記述例

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。

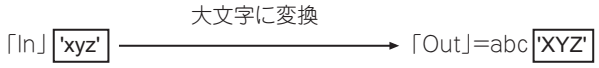
LD



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。



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。

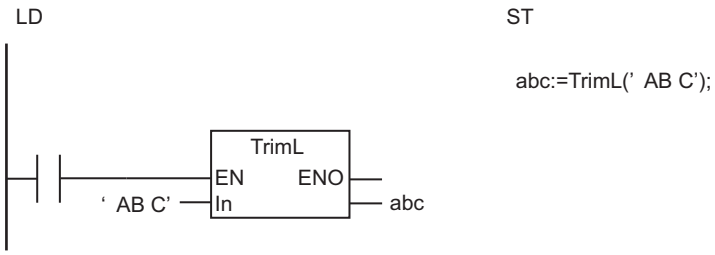
命令	名称	FB/FUN	グラフィック表現	ST 表現
TrimL	文字列・左トリム	FUN	(@)TrimL EN ENO In Out	Out:=TrimL(In);
TrimR	文字列・右トリム	FUN	(@)TrimR EN ENO In Out	Out:=TrimR(In);

TrimL の記述例

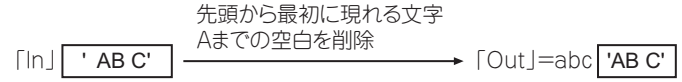
削除対象文字列「In」の、先頭から最初に現れる文字までの空白を、削除します。
先頭に空白がない場合は、何も処理しません。

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。

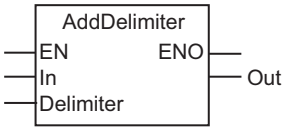


「In」の先頭から最初に現れる文字までの空白を削除します。



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。

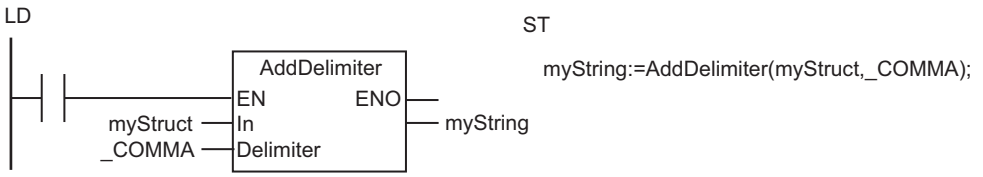
命令	名称	FB/FUN	グラフィック表現	ST 表現
AddDelimiter	デリミタ付加文字列書込	FUN		Out:=AddDelimiter(In, Delimiter);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

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。



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



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。**

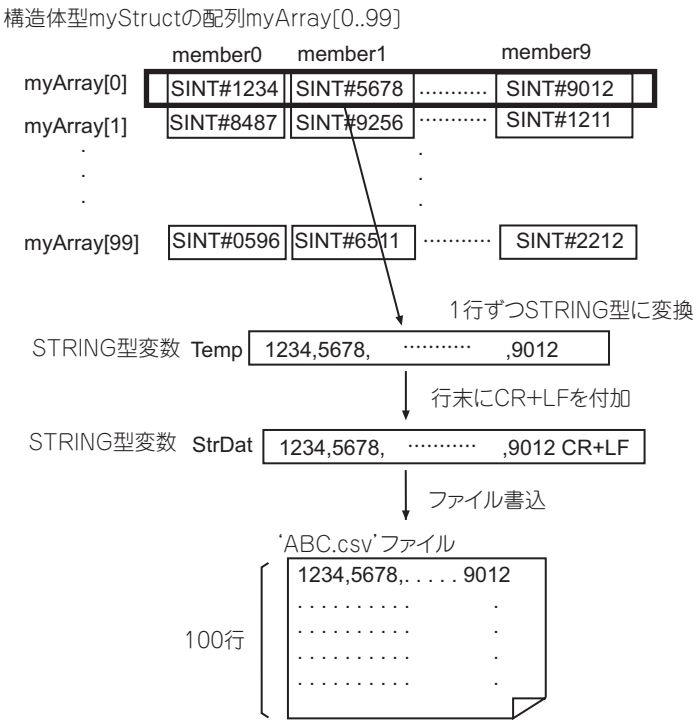
3 **CONCAT 命令で、「Temp」と CR+LF を結合し、STRING 型の変数「StrDat」に格納します。**

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



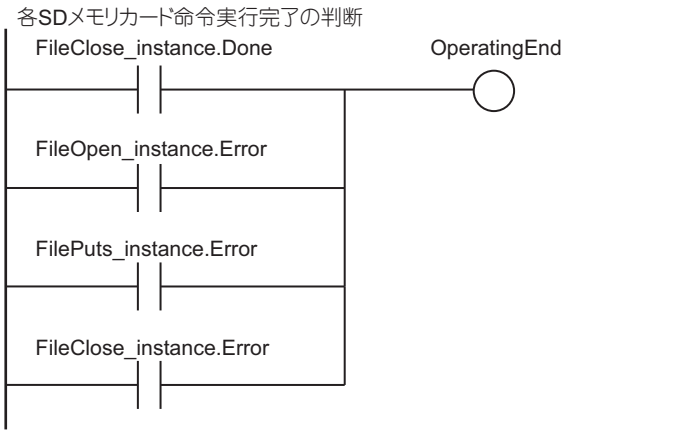
データ型の定義

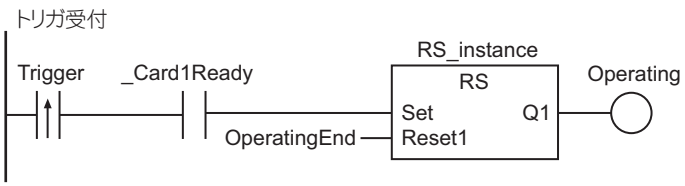
名称	データ型	コメント
myStruct	STRUCT	構造体
member0	SINT	メンバ
member1	SINT	メンバ
member2	SINT	メンバ
member3	SINT	メンバ
member4	SINT	メンバ
member5	SINT	メンバ
member6	SINT	メンバ
member7	SINT	メンバ
member8	SINT	メンバ
member9	SINT	メンバ

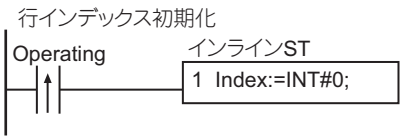
LD

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		

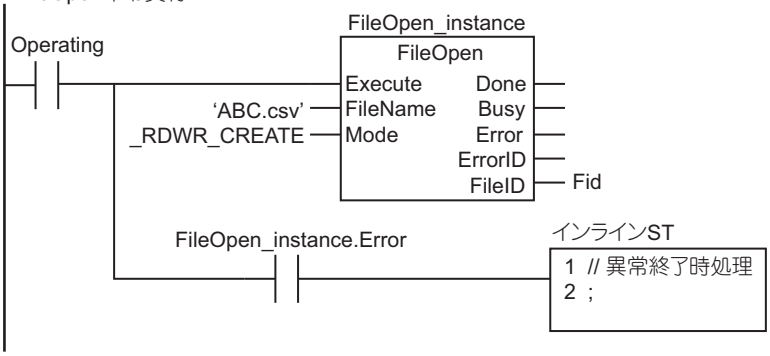
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ



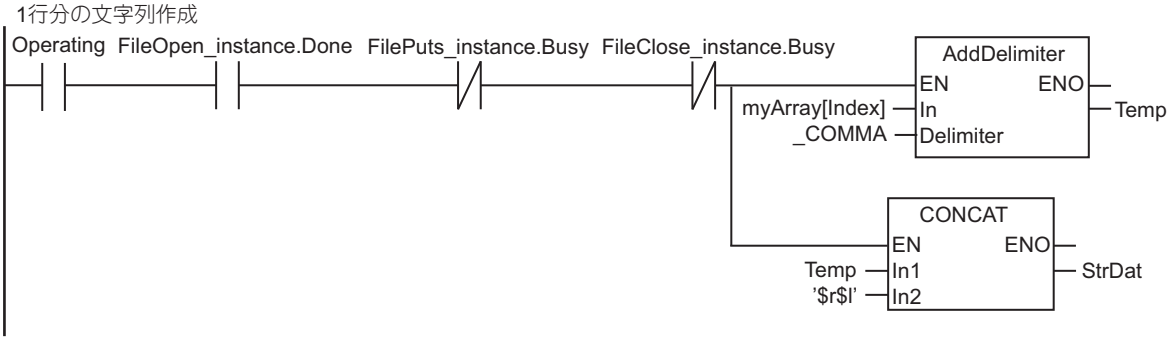




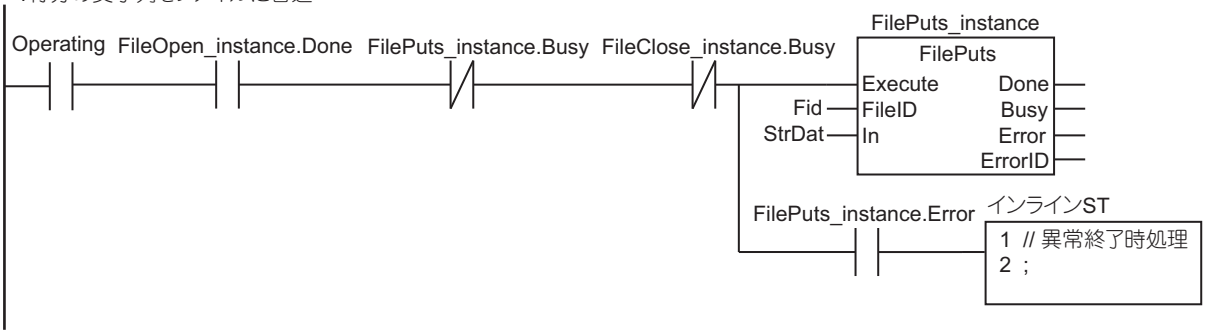
FileOpen命令実行



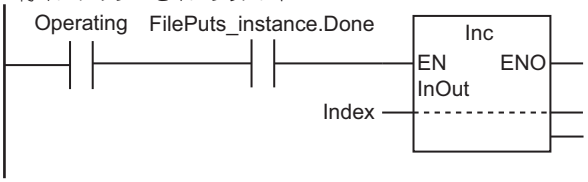
1行分の文字列作成



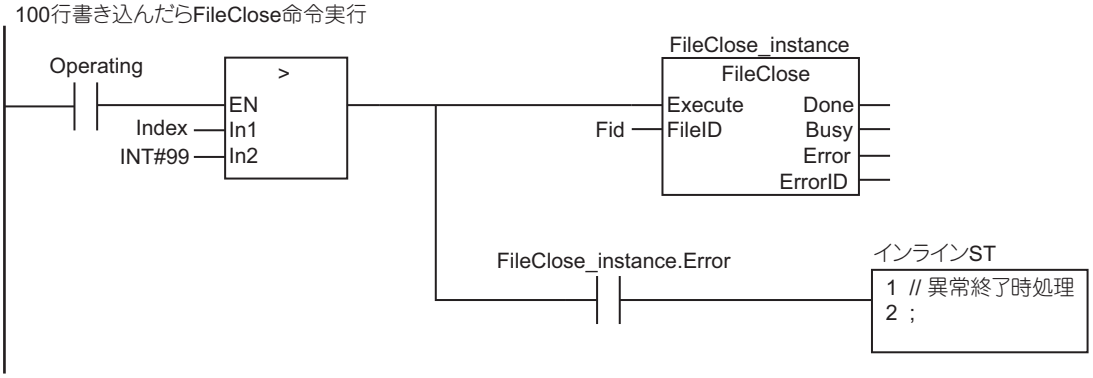
1行分の文字列をファイルに書込

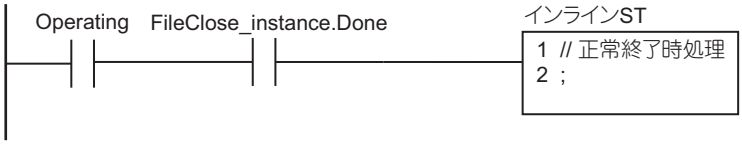


行インデックスをインクリメント



100行書き込んだらFileClose命令実行





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
	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
```

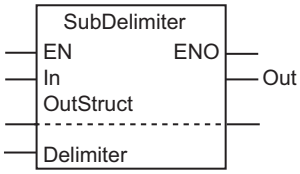
```
    IF (FileClose_instance.Done=TRUE) THEN
      Operating:=FALSE; // 正常終了
    END_IF;

    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。

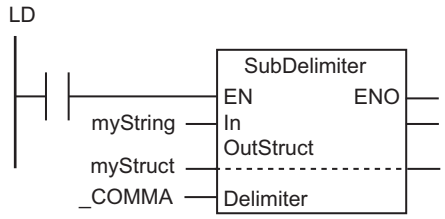
命令	名称	FB/FUN	グラフィック表現	ST 表現
SubDelimiter	文字列読出デリミタ削除	FUN		Out:=SubDelimiter(In, OutStruct, Delimiter);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

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。




```
ST

SubDelimiter(myString,myStruct,_COMMA);
```

myString 'XYZ,1234,aB,3.14'

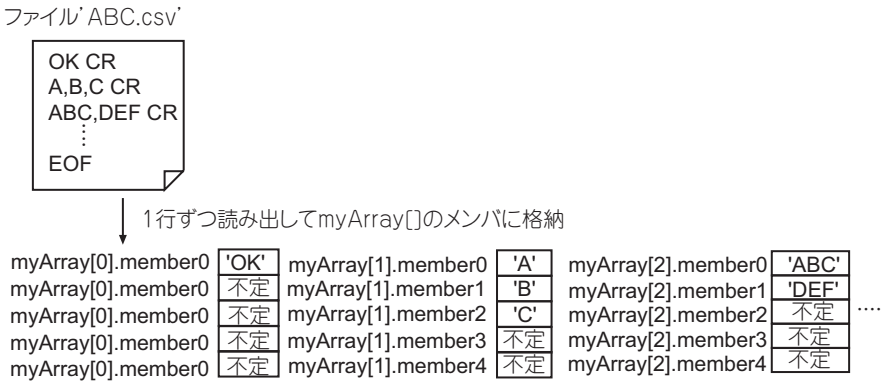
構造体	メンバ	データ型	値
myStruct	member1	STRING	''
	member2	INT	INT#0
	member3	WORD	WORD#16#0000
	member4	REAL	REAL#0

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

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

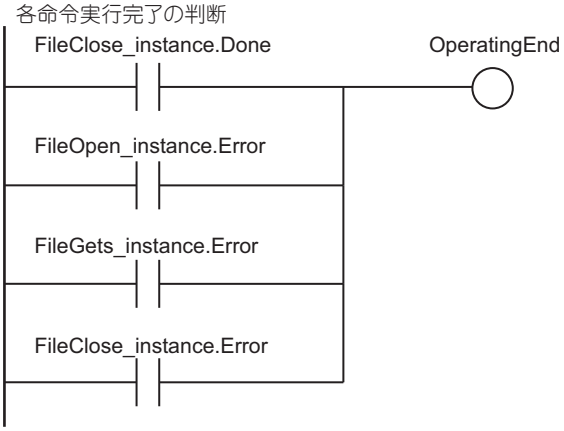
LD

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		

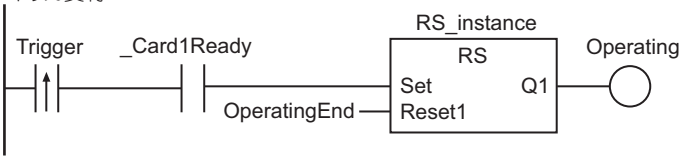
内部変数	名称	データ型	初期値	コメント
	FileClose_instance	FileClose		

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

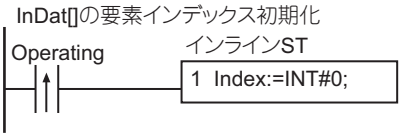
各命令実行完了の判断



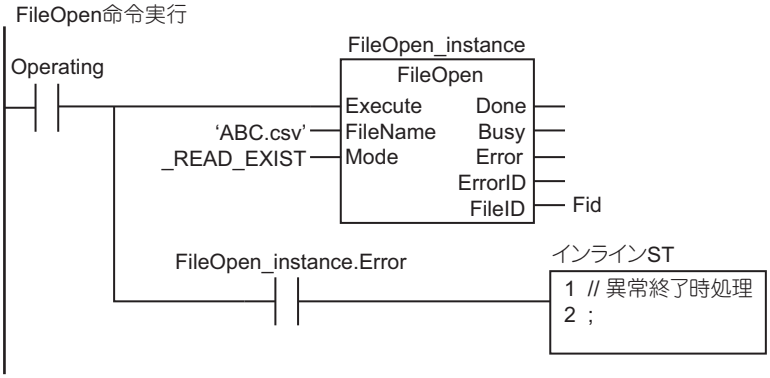
トリガ受付

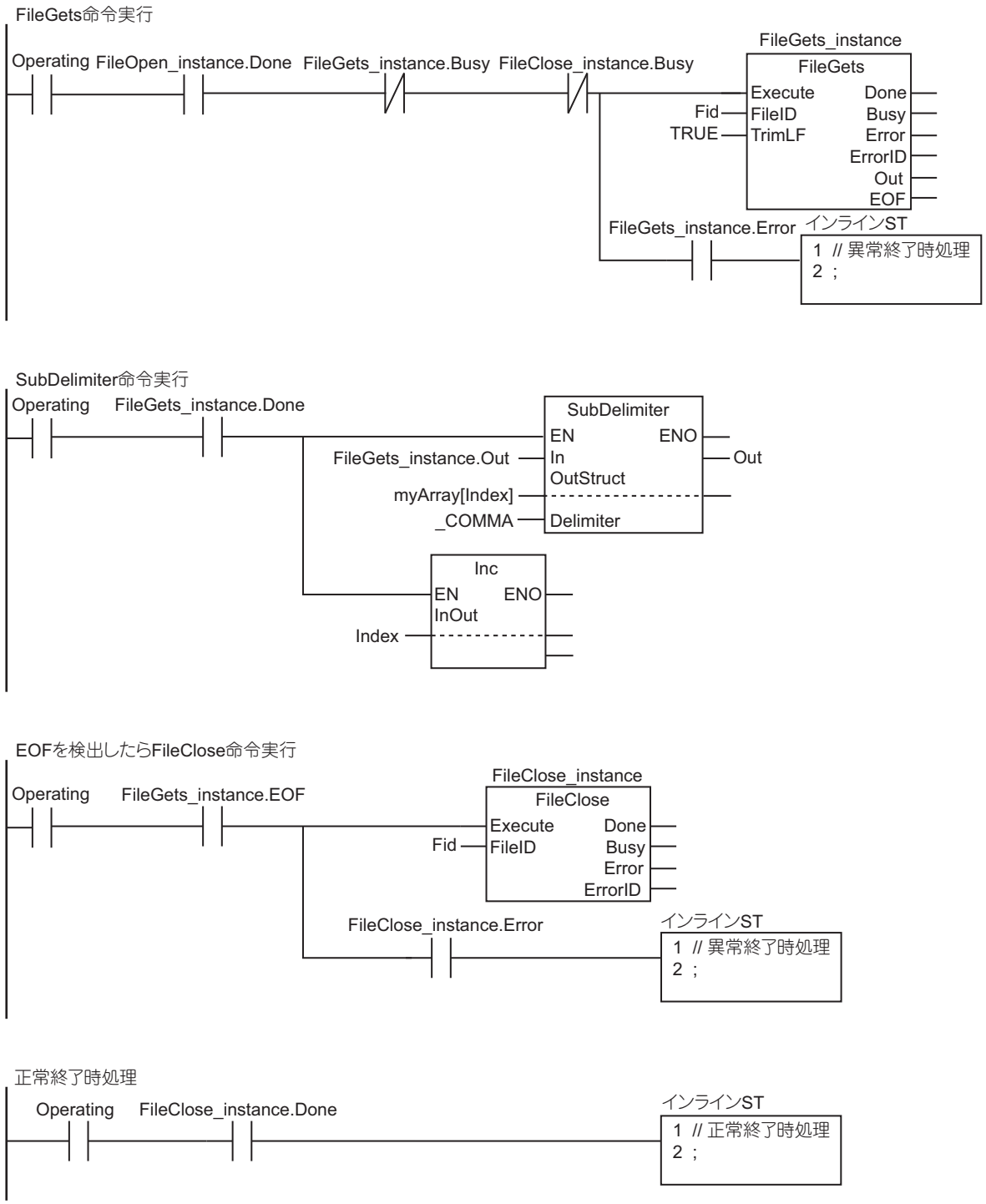


InDat[]の要素インデックス初期化



FileOpen命令実行





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

内部変数	名称	データ型	初期値	コメント
	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.1-105
ADD_TOD_TIME	時刻と時間の加算	P.1-106
ADD_DT_TIME	日付時刻と時間の加算	P.1-107
SUB_TIME	時間減算	P.1-108
SUB_TOD_TIME	時刻と時間の減算	P.1-109
SUB_TOD_TOD	時刻減算	P.1-110
SUB_DATE_DATE	日付減算	P.1-111
SUB_DT_DT	日付時刻減算	P.1-112
SUB_DT_TIME	日付時刻と時間の減算	P.1-113
MULTIME	時間乗算	P.1-114
DIVTIME	時間除算	P.1-115
CONCAT_DATE_TOD	日付と時刻の結合	P.1-116
DT_TO_TOD	日付時刻からの時刻切出	P.1-117
DT_TO_DATE	日付時刻からの日付切出	P.1-118
SetTime	時計補正	P.1-119
GetTime	時刻取得	P.1-120
DtToSec	日付時刻→秒変換	P.1-121
DateToSec	日付→秒変換	P.1-122
TodToSec	時刻→秒変換	P.1-123
SecToDt	秒→日付時刻変換	P.1-124
SecToDate	秒→日付変換	P.1-125
SecToTod	秒→時刻変換	P.1-126
TimeToNanoSec	時間→ナノ秒変換	P.1-127

命令	名称	ページ
TimeToSec	時間→秒変換	P.1-128
NanoSecToTime	ナノ秒→時間変換	P.1-129
SecToTime	秒→時間変換	P.1-130
ChkLeapYear	うるう年判別	P.1-131
GetDaysOfMonth	月の日数取得	P.1-132
DaysToMonth	日数→月変換	P.1-134
GetDayOfWeek	曜日取得	P.1-135
TruncTime	時間切り捨て	P.1-136
TruncDt	日付時刻切り捨て	P.1-139
TruncTod	時刻切り捨て	P.1-142

ADD_TIME

時間と時間を加算します。

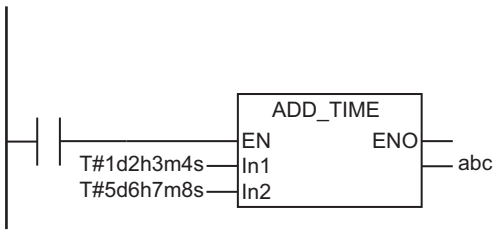
命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_TIME	時間加算	FUN	(@)ADD_TIME EN ENO In1 Out In2	Out:=ADD_TIME(In1, In2);

記述例

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

ADD_TOD_TIME

時刻に時間を加算します。

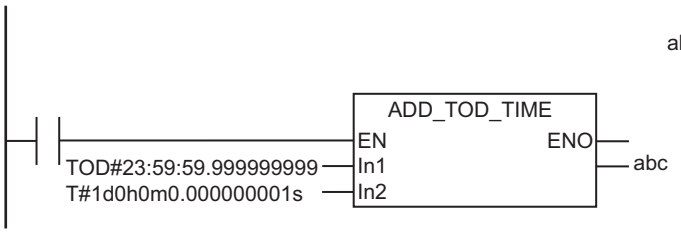
命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_TOD_TIME	時刻と時間の加算	FUN	(@)ADD_TOD_TIME EN ENO In1 Out In2	Out:=ADD_TOD_TIME(In1, In2);

記述例

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

ADD_DT_TIME

日付時刻に時間を加算します。

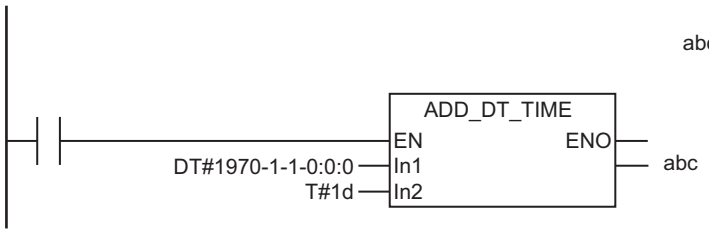
命令	名称	FB/FUN	グラフィック表現	ST 表現
ADD_DT_TIME	日付時刻と時間の加算	FUN	(@)ADD_DT_TIME EN ENO In1 Out In2	Out:=ADD_DT_TIME(In1, In2);

記述例

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。

LD



ST

```
abc:=ADD_DT_TIME(DT#1970-1-1-0:0:0, T#1d);
```

	In1	DT#1970-1-1-0:0:0
+	In2	T#1d
	Out=abc	DT#1970-1-2-0:0:0

SUB_TIME

時間から時間を減算します。

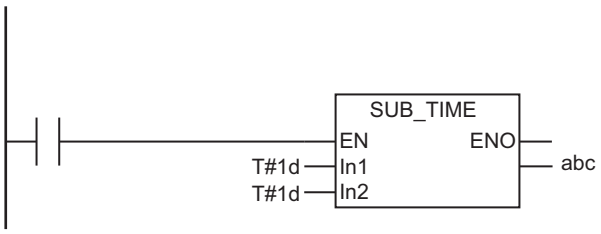
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TIME	時間減算	FUN	(@)SUB_TIME EN ENO In1 Out In2	Out:=SUB_TIME(In1, In2);

記述例

時間「In1」から、時間「In2」を減算します。減算結果「Out」も、時間です。

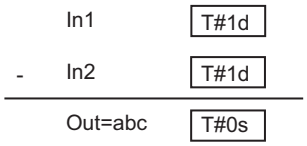
「In1」＝「In2」=T#1d の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_TIME(T#1d, T#1d);
```



SUB_TOD_TIME

時刻から時間を減算します。

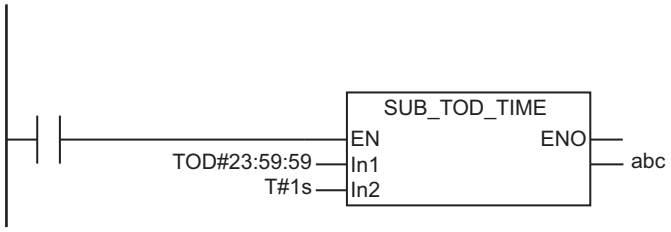
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TOD_TIME	時刻と時間の減算	FUN	(@)SUB_TOD_TIME EN ENO In1 Out In2	Out:=SUB_TOD_TIME(In1, In2);

記述例

時刻「In1」から、時間「In2」を減算します。減算結果「Out」は、時刻です。

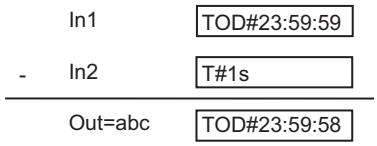
「In1」=TOD#23:59:59、「In2」=T#1s の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_TOD_TIME(TOD#23:59:59, T#1s);
```



SUB_TOD_TOD

時刻から時刻を減算します。

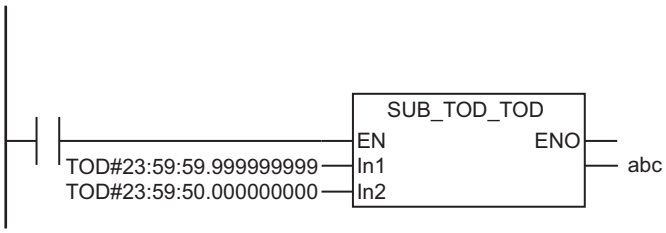
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_TOD_TOD	時刻減算	FUN	(@)SUB_TOD_TOD EN ENO In1 Out In2	Out:=SUB_TOD_TOD(In1, In2);

記述例

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。

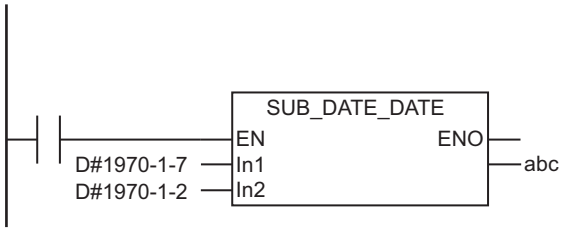
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DATE_DATE	日付減算	FUN	(@)SUB_DATE_DATE EN　ENO In1　Out In2	Out:=SUB_DATE_DATE(In1, In2);

記述例

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。

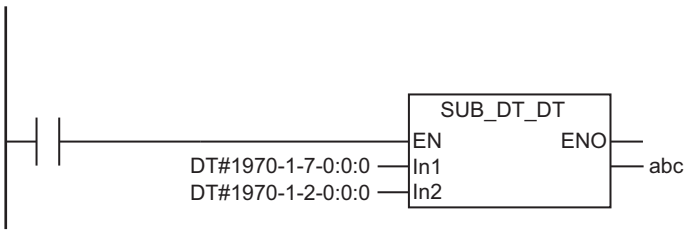
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DT_DT	日付時刻減算	FUN	(@)SUB_DT_DT EN ENO In1 Out In2	Out:=SUB_DT_DT(In1, In2);

記述例

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。

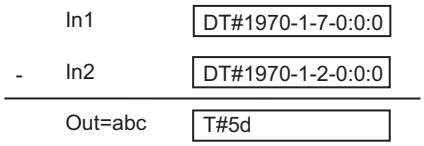
「In1」=DT#1970-1-7-0:0:0、「In2」=DT#1970-1-2-0:0:0 の場合の例を、以下に示します。

LD



ST

```
abc:=SUB_DT_DT(DT#1970-1-7-0:0:0,
               DT#1970-1-2-0:0:0);
```



SUB_DT_TIME

日付時刻から時間を減算します。

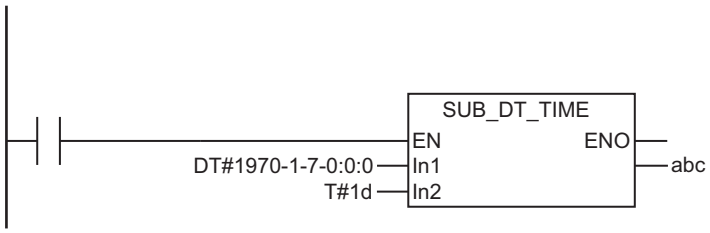
命令	名称	FB/FUN	グラフィック表現	ST 表現
SUB_DT_TIME	日付時刻と時間の減算	FUN	(@)SUB_DT_TIME EN ENO In1 Out In2	Out:=SUB_DT_TIME(In1, In2);

記述例

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

MULTIME

時間を指定乗数で乗算します。

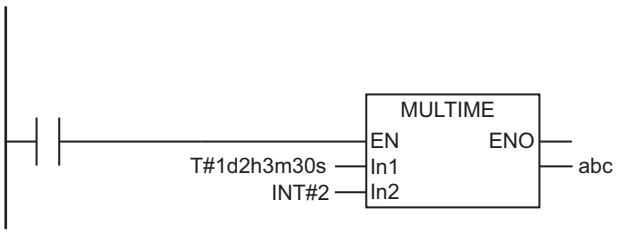
命令	名称	FB/FUN	グラフィック表現	ST 表現
MULTIME	時間乗算	FUN	(@)MULTIME EN ENO In1 Out In2	Out:=MULTIME(In1, In2);

記述例

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

DIVTIME

時間を指定除数で除算します。

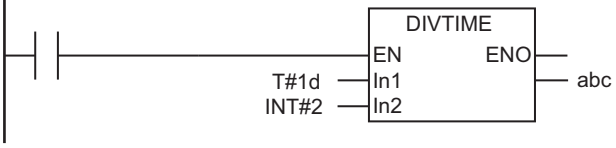
命令	名称	FB/FUN	グラフィック表現	ST 表現
DIVTIME	時間除算	FUN	(@)DIVTIME EN ENO In1 Out In2	Out:=DIVTIME(In1, In2);

記述例

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

CONCAT_DATE_TOD

日付と時刻を結合します。

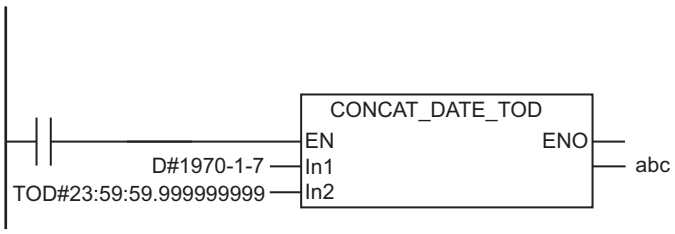
命令	名称	FB/FUN	グラフィック表現	ST 表現
CONCAT_DATE_TOD	日付と時刻の結合	FUN	(@)CONCAT_DATE_TOD EN In1 In2 ENO Out	Out:=CONCAT_DATE_TOD(In1, In2);

記述例

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

DT_TO_TOD

日付時刻から時刻を切り出します。

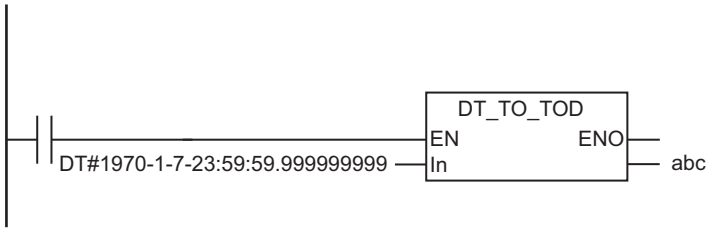
命令	名称	FB/FUN	グラフィック表現	ST 表現
DT_TO_TOD	日付時刻からの時刻切出	FUN	(@)DT_TO_TOD EN ENO In Out	Out:=DT_TO_TOD(In);

記述例

日付時刻「In」から、時刻の部分だけを切り出します。

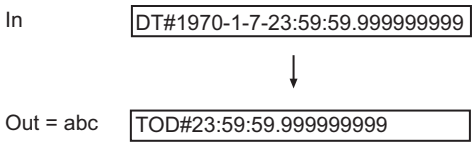
「In」=DT#1970-1-7-23:59:59.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DT_TO_TOD(
      DT#1970-1-7-23:59:59.999999999);
```



DT_TO_DATE

日付時刻から日付を切り出します。

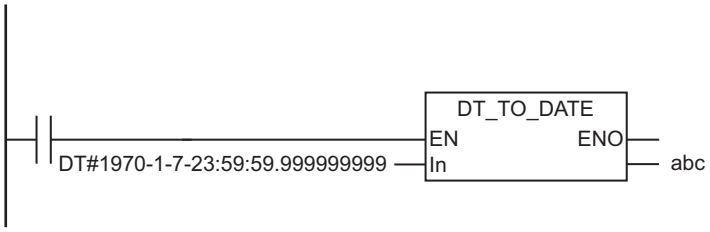
命令	名称	FB/FUN	グラフィック表現	ST 表現
DT_TO_DATE	日付時刻からの日付切出	FUN	(@)DT_TO_DATE EN ENO In Out	Out:=DT_TO_DATE(In);

記述例

日付時刻「In」から、日付の部分だけを切り出します。

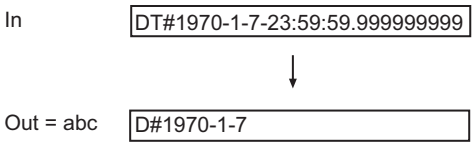
「In」=DT#1970-1-7-23:59:59.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DT_TO_DATE(
      DT#1970-1-7-23:59:59.999999999);
```



SetTime

システム時刻を補正します。

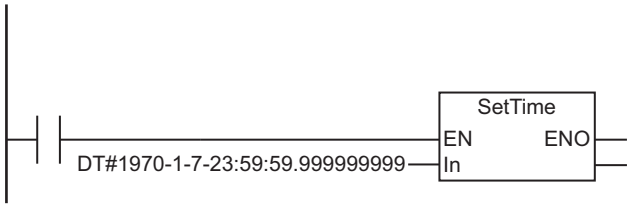
命令	名称	FB/FUN	グラフィック表現	ST 表現
SetTime	時計補正	FUN	(@)SetTime EN ENO In Out	SetTime(In);

記述例

日付時刻「In」の値を、システム時刻にセットします。

「In」=DT#1970-1-7-23:59:59.999999999 の場合の記述例を、以下に示します。

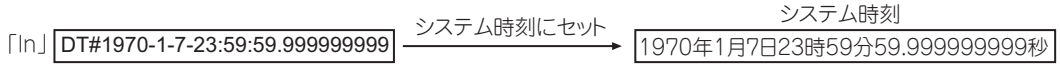
LD



ST

```
SetTime(DT#1970-1-7-23:59:59.999999999);
```

「In」の値を、システム時刻にセットします。



GetTime

現在時刻を読み出します。

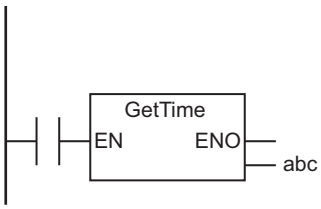
命令	名称	FB/FUN	グラフィック表現	ST 表現
GetTime	時刻取得	FUN	(@)GetTime EN ENO Out	Out:=GetTime();

記述例

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



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。

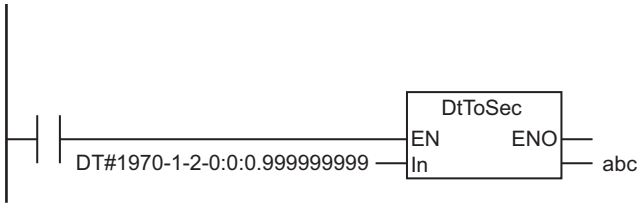
命令	名称	FB/FUN	グラフィック表現	ST 表現
DtToSec	日付時刻→秒変換	FUN	(@)DtToSec EN ENO In Out	Out:=DtToSec(In);

記述例

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。

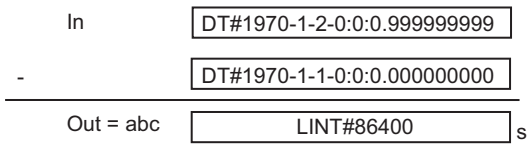
「In」=DT#1970-1-2-0:0:0.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=DtToSec(DT#1970-1-2-0:0:0.999999999);
```



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。

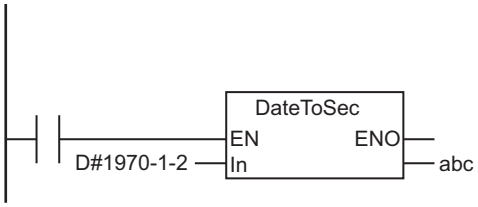
命令	名称	FB/FUN	グラフィック表現	ST 表現
DateToSec	日付→秒変換	FUN	(@)DateToSec EN ENO In Out	Out:=DateToSec(In);

記述例

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。

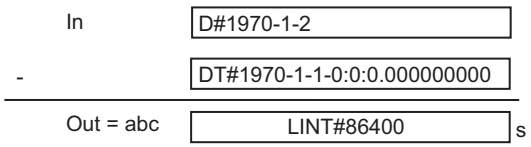
「In」=D#1970-1-2 の場合の例を、以下に示します。

LD



ST

```
abc:=DateToSec(D#1970-1-2);
```



TodToSec

時刻を、0 時 0 分 0 秒からの秒数に変換します。

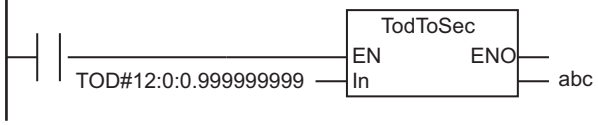
命令	名称	FB/FUN	グラフィック表現	ST 表現
TodToSec	時刻→秒変換	FUN	(@)TodToSec EN ENO In Out	Out:=TodToSec(In);

記述例

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。

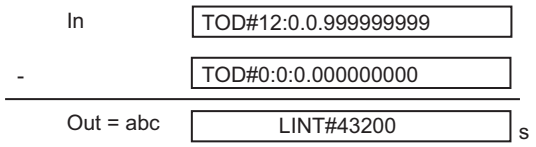
「In」=TOD#12:0:0.999999999 の場合の例を、以下に示します。

LD



ST

```
abc:=TodToSec(TOD#12:0:0.999999999);
```



SecToDt

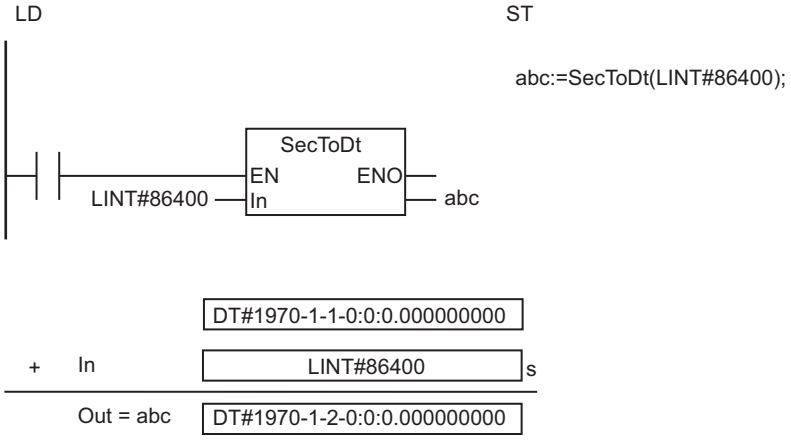
1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付時刻に変換します。

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);

記述例

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。



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。

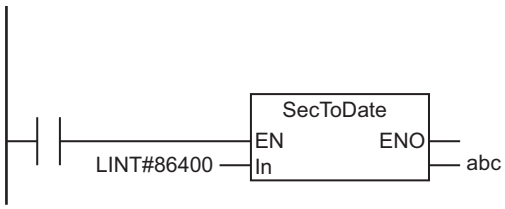
命令	名称	FB/FUN	グラフィック表現	ST 表現
SecToDate	秒→日付変換	FUN	(@)SecToDate EN ENO In Out	Out:=SecToDate(In);

記述例

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。

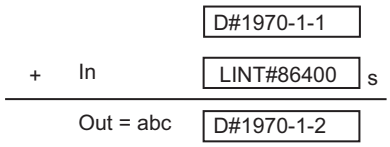
「In」=LINT#86400 の場合の例を、以下に示します。

LD



ST

```
abc:=SecToDate(LINT#86400);
```



SecToTod

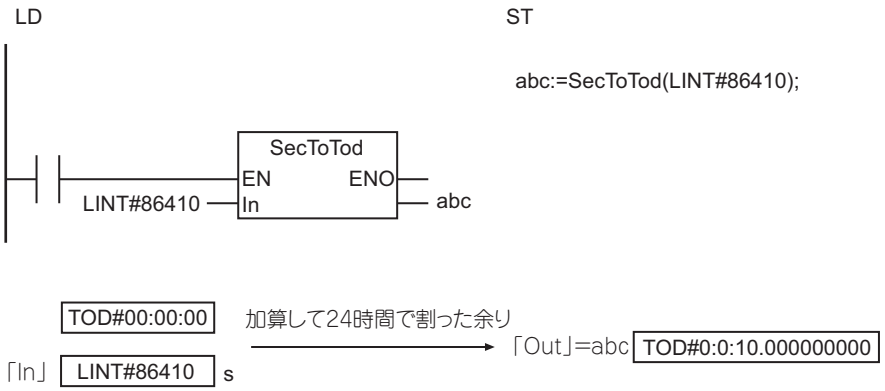
0 時 0 分 0 秒からの秒数を、時刻に変換します。

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);

記述例

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。



TimeToNanoSec

時間を、ナノ秒数に変換します。

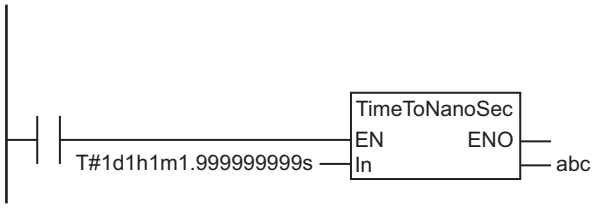
命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeToNanoSec	時間→ナノ秒変換	FUN	(@)TimeToNanoSec EN　ENO In　Out	Out:=TimeToNanoSec(In);

記述例

時間「In」を、ナノ秒数に変換します。

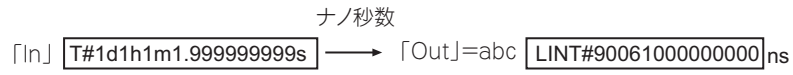
「In」=T#1d1h1m1.999999999s の場合の例を、以下に示します。

LD



ST

```
abc:=TimeToNanoSec(T#1d1h1m1.999999999s);
```



TimeToSec

時間を、秒数に変換します。

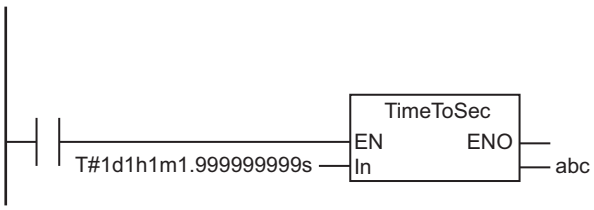
命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeToSec	時間→秒変換	FUN	(@)TimeToSec EN ENO In Out	Out:=TimeToSec(In);

記述例

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);
```



NanoSecToTime

ナノ秒数を、時間に変換します。

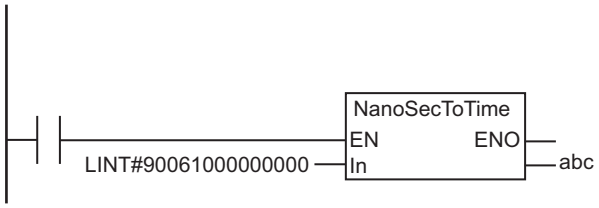
命令	名称	FB/FUN	グラフィック表現	ST 表現
NanoSecToTime	ナノ秒→時間変換	FUN	(@)NanoSecToTime EN　ENO In　Out	Out:=NanoSecToTime(In);

記述例

ナノ秒数「In」を、時間に変換します。

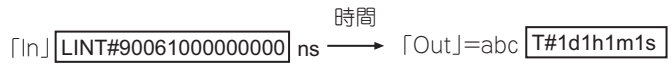
「In」=LINT#90061000000000 の場合の例を、以下に示します。

LD



ST

```
abc:=NanoSecToTime(LINT#90061000000000);
```



SecToTime

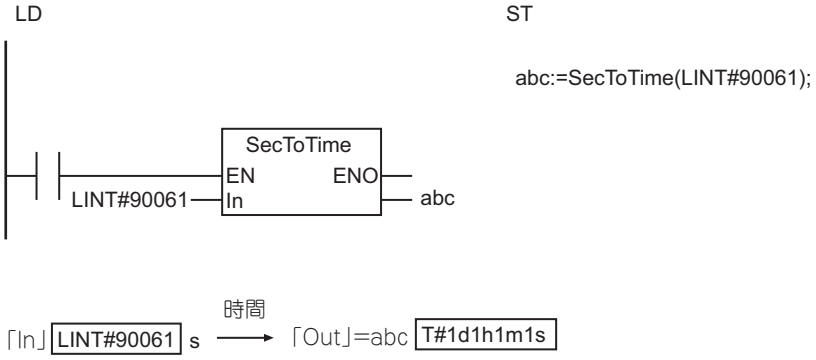
秒数を、時間に変換します。

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);

記述例

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。



ChkLeapYear

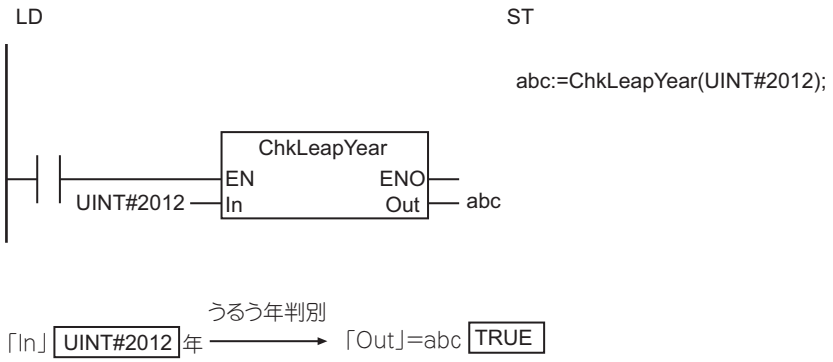
指定された年が、うるう年かどうかを判定します。

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	(@)ChkLeapYear EN ENO In Out	Out:=ChkLeapYear(In);

記述例

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。



GetDaysOfMonth

指定された月の、日数を取得します。

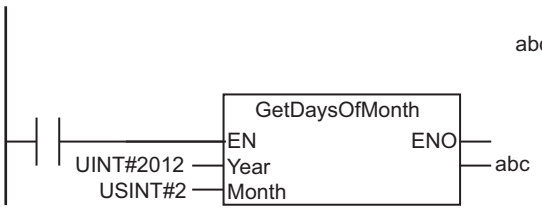
命令	名称	FB/FUN	グラフィック表現	ST 表現
GetDaysOfMonth	月の日数取得	FUN	(@)GetDaysOfMonth EN ENO Year Out Month	Out:=GetDaysOfMonth(Year, Month);

記述例

「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);
```



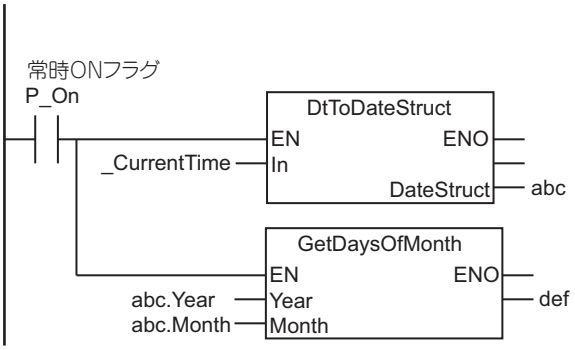
サンプルプログラム

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。

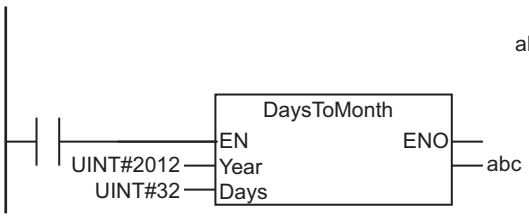
命令	名称	FB/FUN	グラフィック表現	ST 表現
DaysToMonth	日数→月変換	FUN	(@)DaysToMonth EN ENO Year Out Days	Out:=DaysToMonth(Year, Days);

記述例

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);
```



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	(@)GetDayOfWeek EN ENO In Out	Out:=GetDayOfWeek(In);

記述例

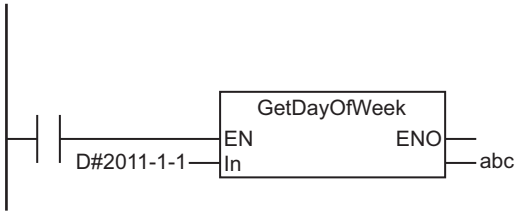
年月日「In」で示された年月日の、曜日を取得します。

「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



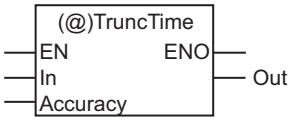
ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```



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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.01 以降、かつ Sysmac Studio Ver.1.02 以降で、使用可能です。

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**

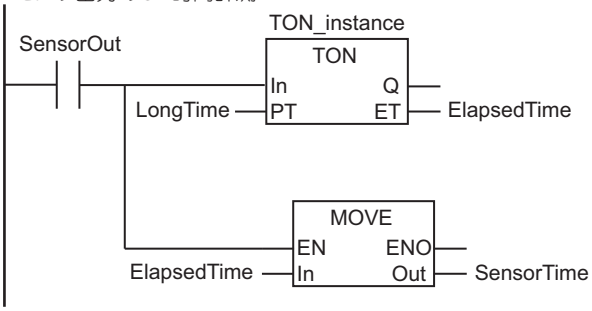
名称	データ型	初期値	コメント
RecentMode	Mode	SET	現在の動作モード

LD

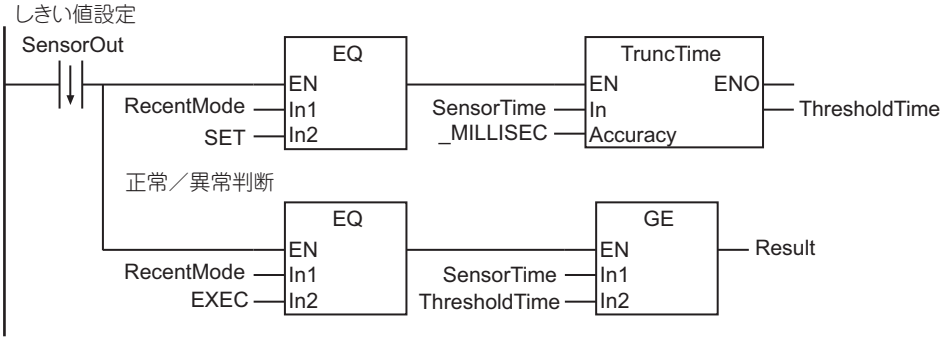
内部変数	名称	データ型	初期値	コメント
	SensorOut	BOOL	FALSE	センサ出力
	ElapsedTime	TIME	T#0s	経過時間
	SensorTime	TIME	T#0s	センサの ON 時間
	LongTime	TIME	T#1h	センサの ON 時間より十分に長い時間
	ThresholdTime	TIME	T#0s	しきい値
	Result	BOOL	FALSE	判断結果　TRUE: 正常　FALSE: 異常
	TON_instance	TON		

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

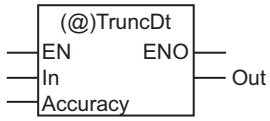
// TON の経過時間をセンサの ON 時間とする
IF (SensorOut=TRUE) THEN
  SensorTime:=ElapsedTime;
END_IF;

// 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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.01 以降、かつ Sysmac Studio Ver.1.02 以降で、使用可能です。

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**

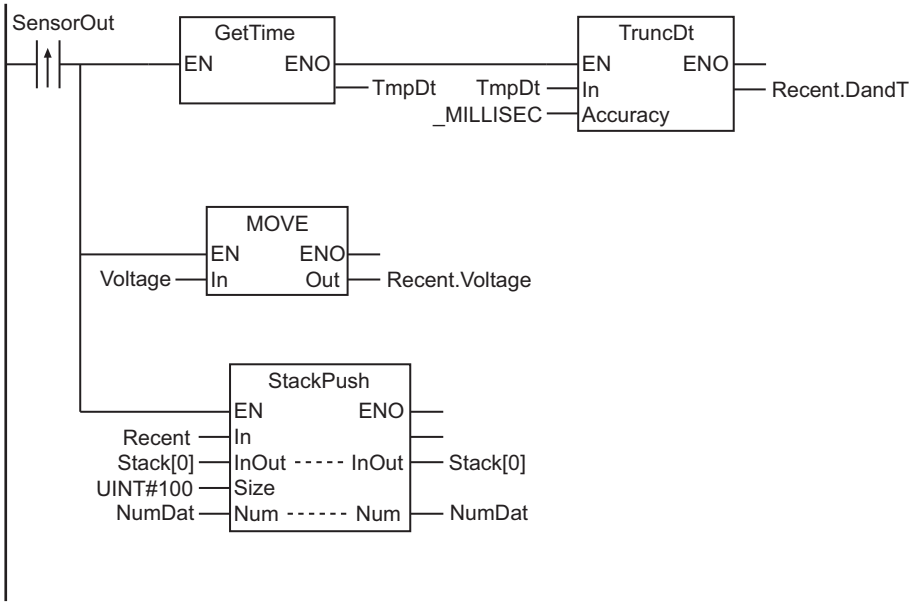
名称	データ型	初期値	コメント
Recent	Record	(DandT:=DT#1970-1-1-0:0:0, Voltage:=0.0)	現在値
Stack	ARRAY[0..99] OF Record	[100((DandT:=DT#1970-1-1-0:0:0, Voltage:=0.0))]	スタック

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

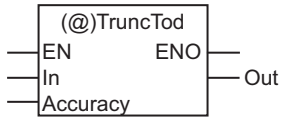
  // 電圧値の現在値を取得
  Recent.Voltage:=Voltage;

  // 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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.01 以降、かつ Sysmac Studio Ver.1.02 以降で、使用可能です。

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

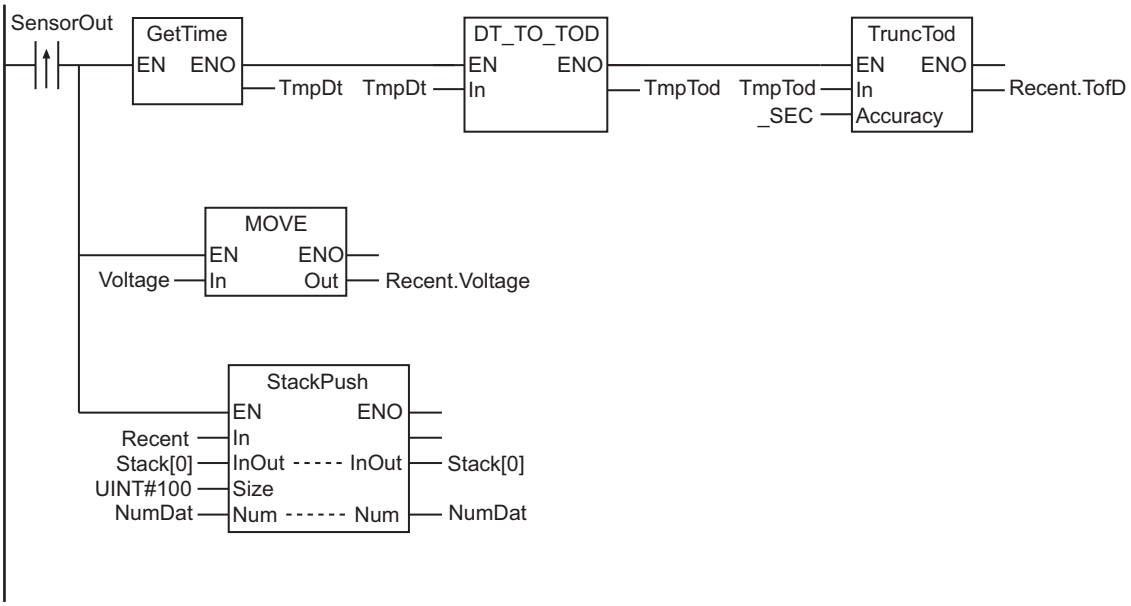
LD

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;

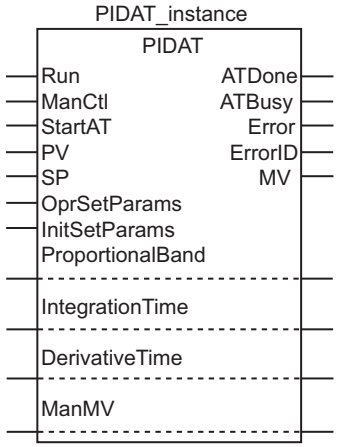
    // 時刻と電圧値をスタックに格納
    StackPush(
      In :=Recent, // 時刻と電圧値
      InOut:=Stack[0], // スタック配列
      Size :=UINT#100, // スタック配列の要素数 100
      Num :=NumDat); // 現在の記録データ数
  END_IF;
```

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.1-146
PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	P.1-152
TimeProportionalOut	時分割比例出力	P.1-165
LimitAlarm_**	上下限警報グループ	P.1-178
LimitAlarmDv_**	上下限偏差警報グループ	P.1-179
LimitAlarmDvStbySeq_**	待機シーケンス付き上下限偏差警報グループ	P.1-180
ScaleTrans	スケール変換	P.1-192
AC_StepProgram	ステッププログラム	P.1-194

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);

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.1-165)」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。

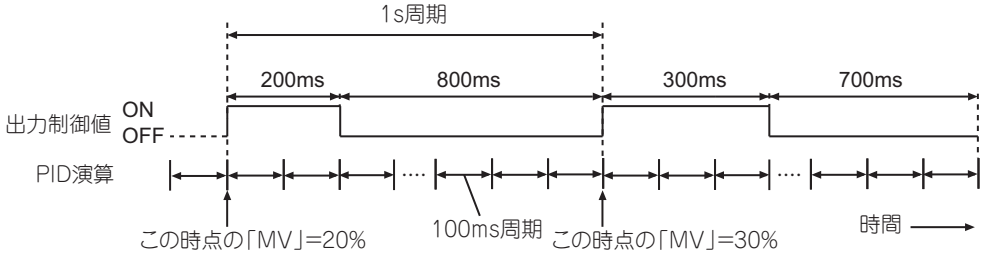
項目名	設定値
Input1:Input signal type	K(1)

I/O マップ設定は、以下のとおりとします。

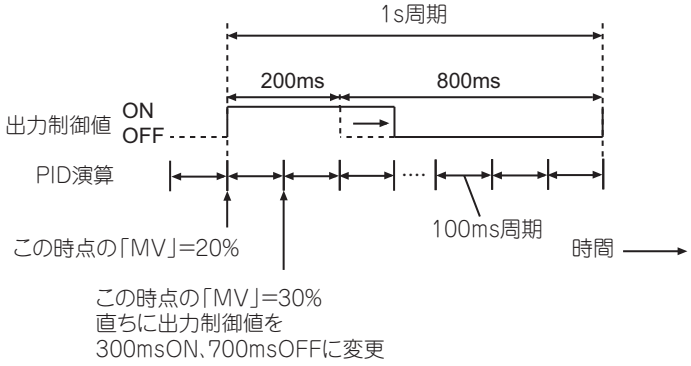
ユニット	ポート	説明	変数
形 CJ1W-PH41U	Ch1_AIInPV	入力 No.1 測定値（INT 型）	AI1
形 CJ1W-OD212	Ch1_Out00	出力 CH1 接点 00	DO1

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

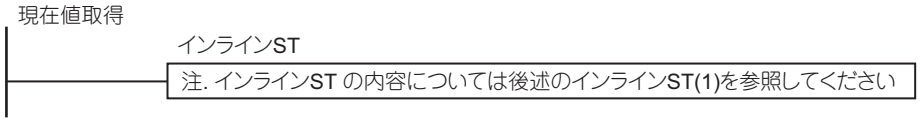
*1. 形 CJ1W-PH41U をラック番号 0 のスロット番号 0、形 CJ1W-OD212 をラック番号 0 のスロット番号 1 に装着した場合の割付先です。

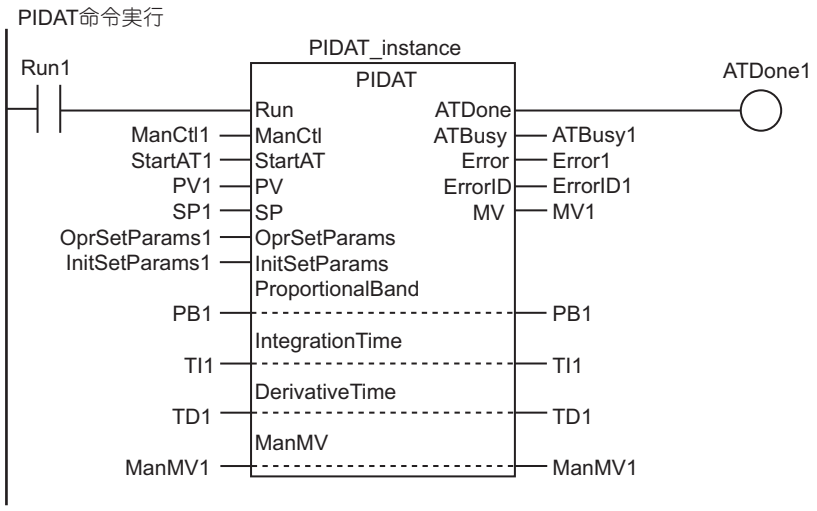
（注）I/O マップ設定により、各ユニットのポートに割り付けたグローバル変数が自動的に生成されます。

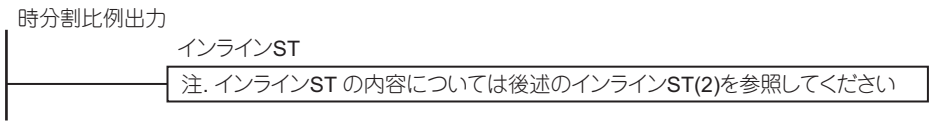
LD

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
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

名称	データ型	初期値	保持	コメント
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		☐	







● 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
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

名称	データ型	初期値	保持	コメント
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。

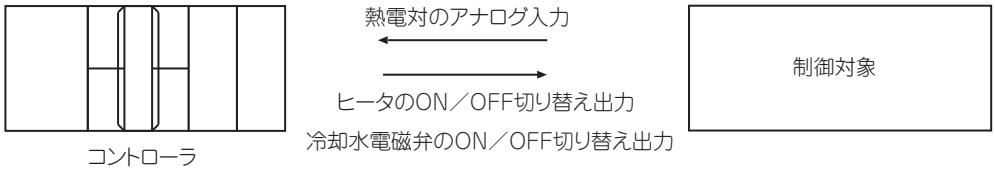
命令	名称	FB/FUN	グラフィック表現	ST 表現
PIDAT_HeatCool	オートチューニング付き加熱冷却 PID 演算	FB	PIDAT_HeatCool_instance PIDAT_HeatCool Run / ManCtl / StartAT / PV / SP / DeadBand / OprSetParams / InitSetParams / ProportionalBand_Heat / IntegrationTime_Heat / DerivativeTime_Heat / ProportionalBand_Cool / IntegrationTime_Cool / DerivativeTime_Cool / ManMV / CtlPrd_Cool ATDone / ATBusy / Error / ErrorID / MV / MV_Heat / MV_Cool	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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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_AllnPV	入力 No.1 測定値	R	INT	J01_Ch1_AllnPV	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
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.1-165)」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
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）

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_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
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

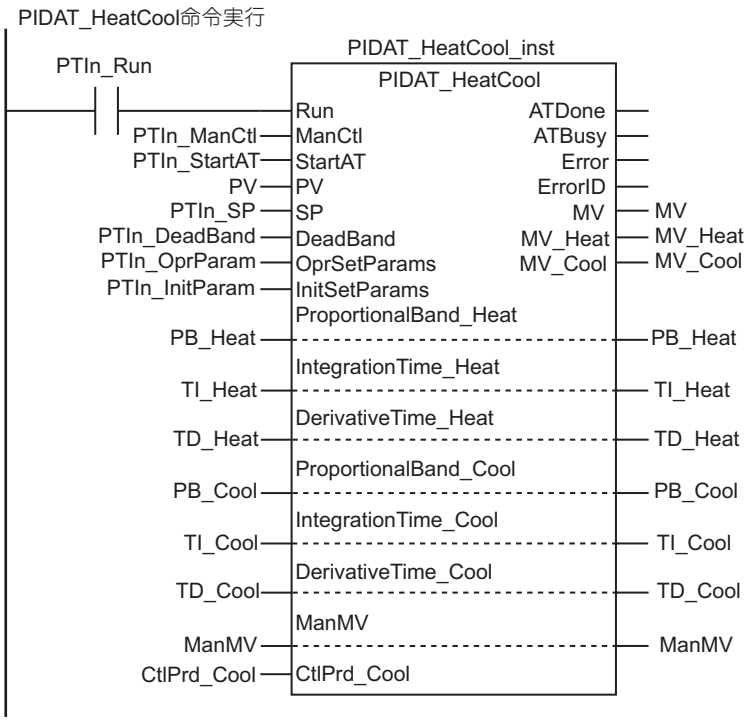
外部変数	名称	データ型	コメント
	J01_Ch1_AllnPV	INT	形 CJ1W-AD04U からの熱電対入力
	J02_Ch1_Out00	BOOL	形 CJ1W-OC201 への加熱用出力
	J02_Ch1_Out04	BOOL	形 CJ1W-OC201 への冷却用出力
	PTIn_Run	BOOL	タッチパネルからの本サンプルプログラム実行フラグ入力
	PTIn_ManCtl	BOOL	タッチパネルからのマニュアル／オート切替フラグ入力
	PTIn_SP	REAL	タッチパネルからの目標値入力
	PTIn_StartAT	BOOL	タッチパネルからのオートチューニング実行フラグ入力
	PTIn_DeadBand	REAL	タッチパネルからのデッドバンド入力

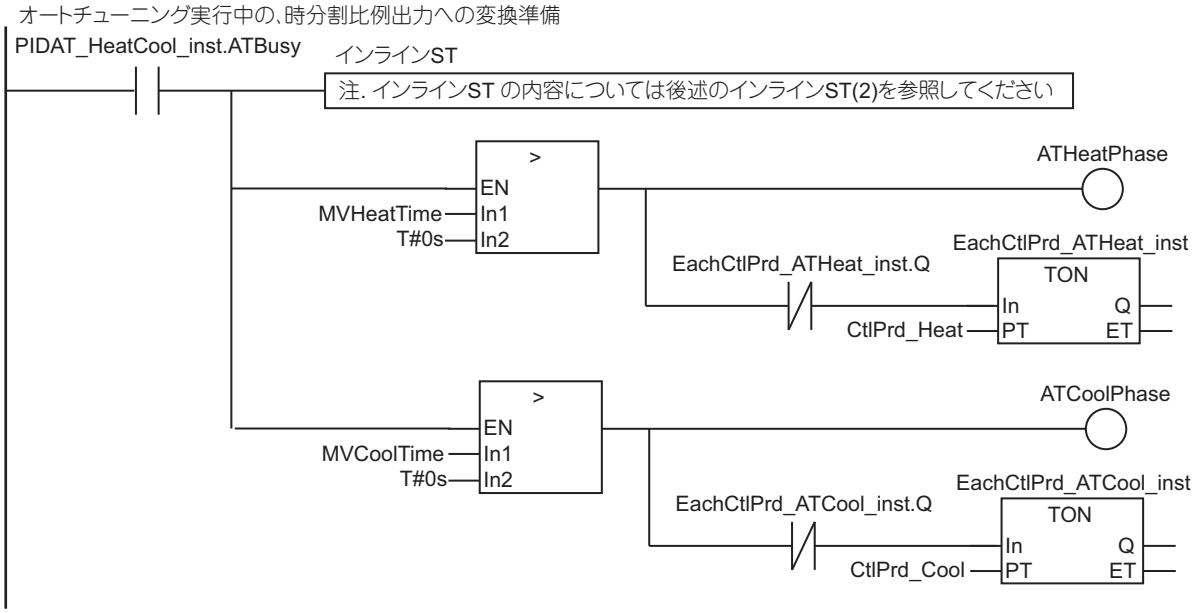
外部変数	名称	データ型	コメント
	PTIn_InitParam	_sINIT_SET_PARAMS	タッチパネルからの初期設定パラメータ入力
	PTIn_InitSetOpr_SampTime	LINT	タッチパネルからのサンプリング周期入力（ms 単位）
	PTIn_OprParam	_sOPR_SET_PARAMS	タッチパネルからの運転中設定パラメータ入力
	PTOut_PV	REAL	タッチパネルへの測定値出力
	PT_PB_Heat	REAL	タッチパネルからの加熱操作の比例帯入出力
	PT_TI_Heat	LINT	タッチパネルからの、加熱操作の積分時間入出力（ms 単位）
	PT_TD_Heat	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

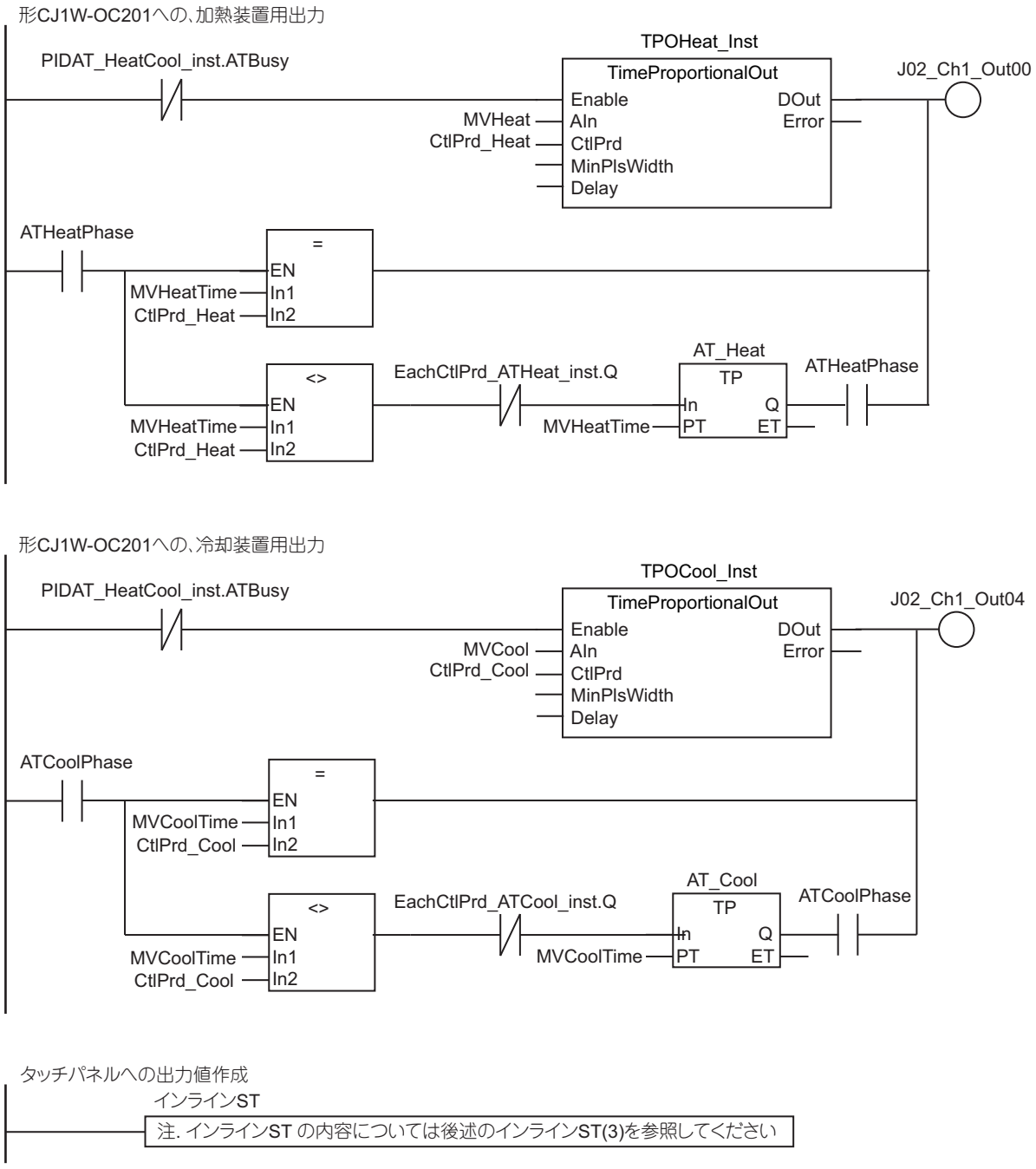
CJ1W-AD04Uおよびタッチパネルからの入力値の、単位変換

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。

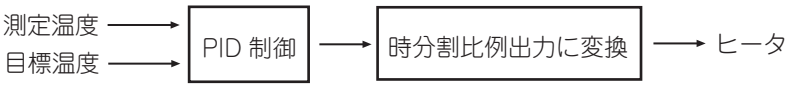
命令	名称	FB/FUN	グラフィック表現	ST 表現
TimeProportionalOut	時分割比例出力	FB	TimeProportionalOut_Instance TimeProportionalOut Enable AIn CtlPrd MinPlsWidth Delay DOut Error	TimeProportionalOut_instance(Enable, AIn, CtlPrd, MinPlsWidth, Delay, DOut, Error);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

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)
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。

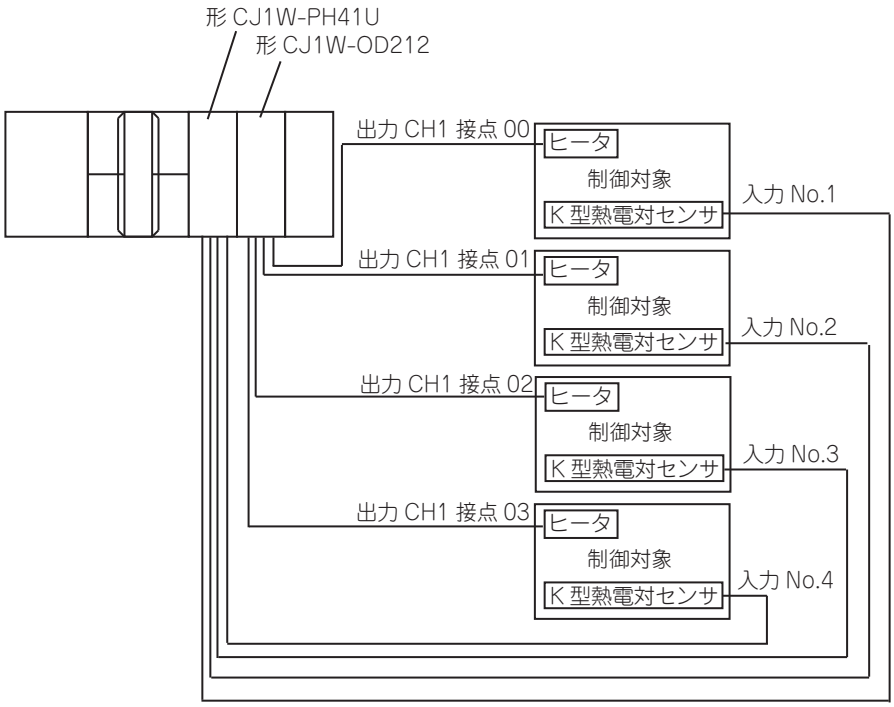
ユニット	ポート	説明	変数
形 CJ1W-PH41U	Ch1_AllnPV	入力 No.1 測定値（INT 型）	AI1
	Ch2_AllnPV	入力 No.2 測定値（INT 型）	AI2
	Ch3_AllnPV	入力 No.3 測定値（INT 型）	AI3
	Ch4_AllnPV	入力 No.4 測定値（INT 型）	AI4
形 CJ1W-OD212	Ch1_Out00	出力 CH1 接点 00	DO1
	Ch1_Out01	出力 CH1 接点 01	DO2
	Ch1_Out02	出力 CH1 接点 02	DO3
	Ch1_Out03	出力 CH1 接点 03	DO4

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**

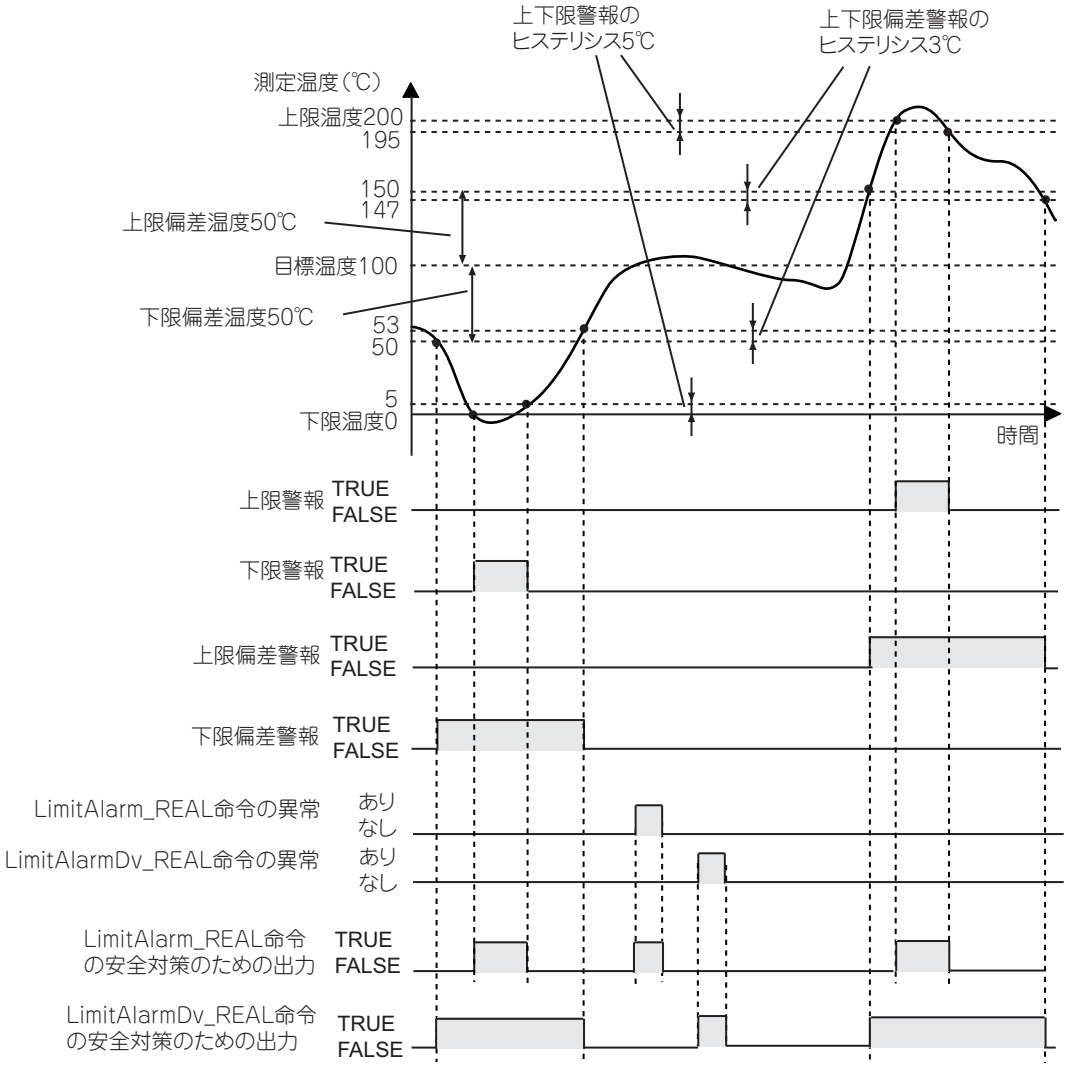


処理内容

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_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

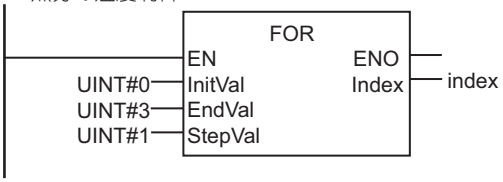
内部変数	名称	データ型	初期値	保持	コメント
	index	UINT	0	□	ループインデックス
	LimitAlarm_ON	BOOL	True	□	上下限警報実行
	LimitAlarmDv_ON	BOOL	True	□	上下限偏差警報実行
	TimeProportionalOut_ON	BOOL	True	□	時分割比例出力実行
	AI	INT	0	□	現在値
	PV	ARRAY[0..3] OF REAL	[4(0.0)]	□	測定値
	SP	ARRAY[0..3] OF REAL	[4(100)]	□	目標温度
	DOut_TPO	BOOL	False	□	時分割比例出力
	HighVal	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
	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

内部変数	名称	データ型	初期値	保持	コメント
	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	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		☐	

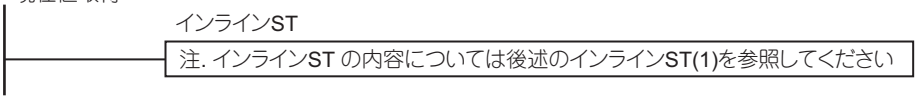
外部変数	名称	データ型	コメント
	AI1	INT	入力 No.1（測定値）
	AI2	INT	入力 No.2（測定値）
	AI3	INT	入力 No.3（測定値）
	AI4	INT	入力 No.4（測定値）

外部変数	名称	データ型	コメント
	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）

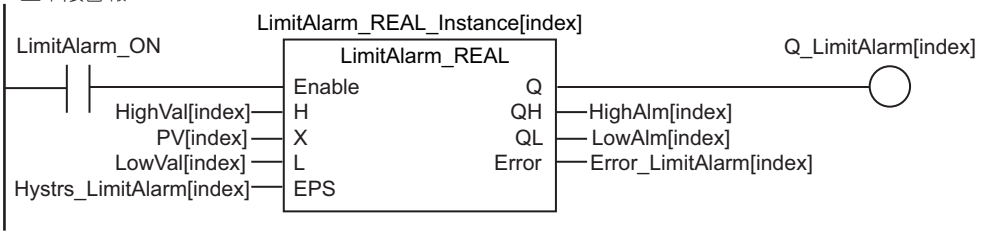
4点分の温度制御



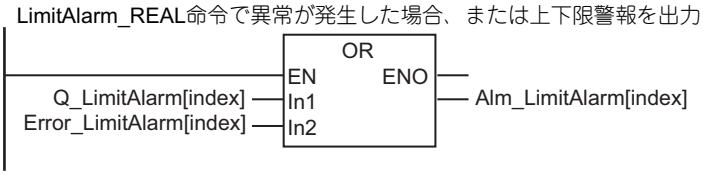
現在値取得



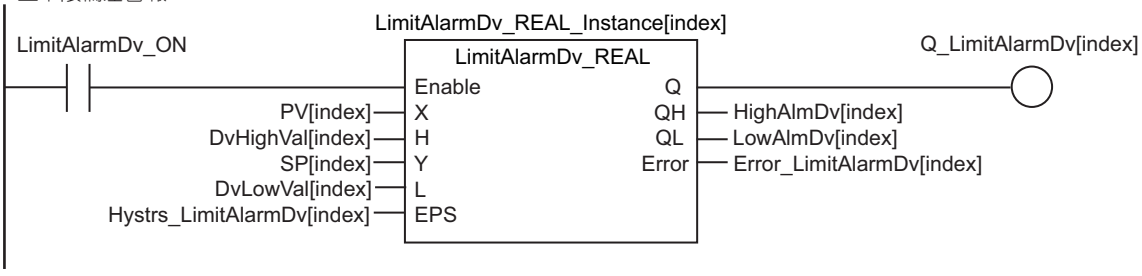
上下限警報



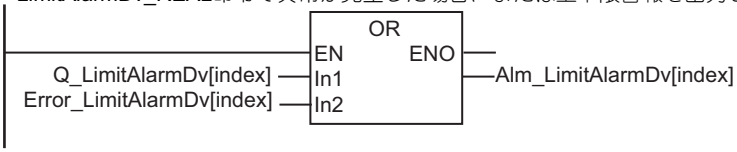
LimitAlarm_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力

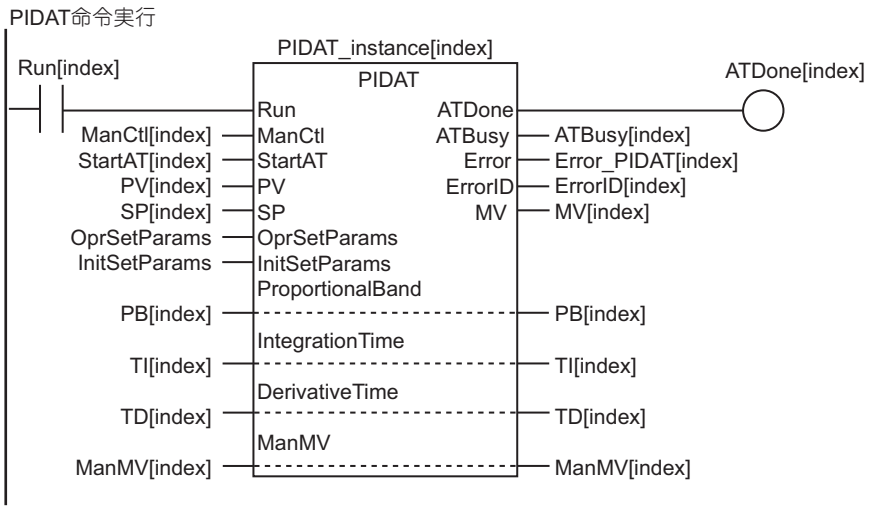


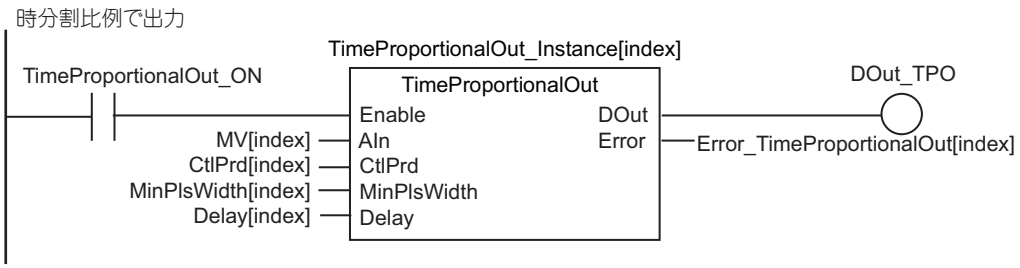
上下限偏差警報

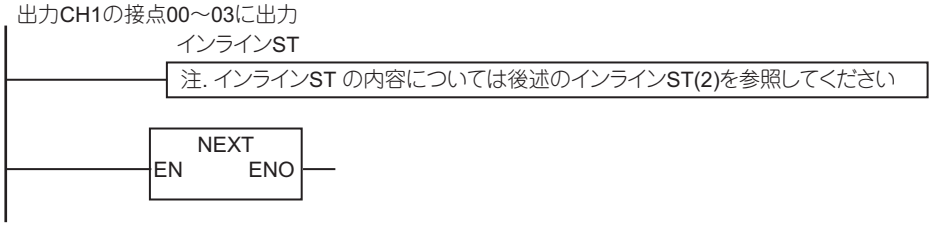


LimitAlarmDv_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力









● 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
	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

内部変数	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
	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
	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

内部変数	名称	データ型	初期値	保持	コメント
	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]);

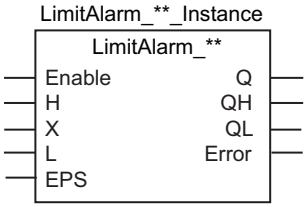
//出力 Ch1 の接点 00～03 に出力
CASE index OF
INT#0:
```

```
      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

バージョン関連情報

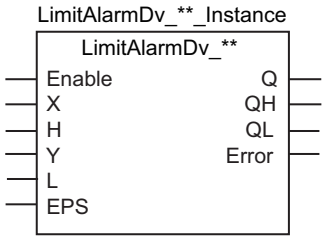
本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

サンプルプログラム

TimeProportionalOut 命令の「サンプルプログラム (P.1-165)」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

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

サンプルプログラム

TimeProportionalOut 命令の「サンプルプログラム (P.1-165)」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。

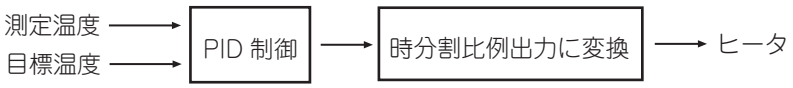
命令	名称	FB/FUN	グラフィック表現	ST 表現
LimitAlarmDvStbySeq_**	待機シーケンス付き上下限偏差警報グループ	FB	LimitAlarmDvStbySeq_**_Instance LimitAlarmDvStbySeq_** Enable, X, H, Y, L, EPS Q, QH, QL, StbySeqFlag, Error ただし、“**”は REAL、LREAL のいずれか	LimitAlarmDvStbySeq_**_instance(Enable, X, H, Y, L, EPS, Q, QH, QL, StbySeqFlag, Error); ただし、“**”は REAL、LREAL のいずれか

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

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.1-165)」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。**

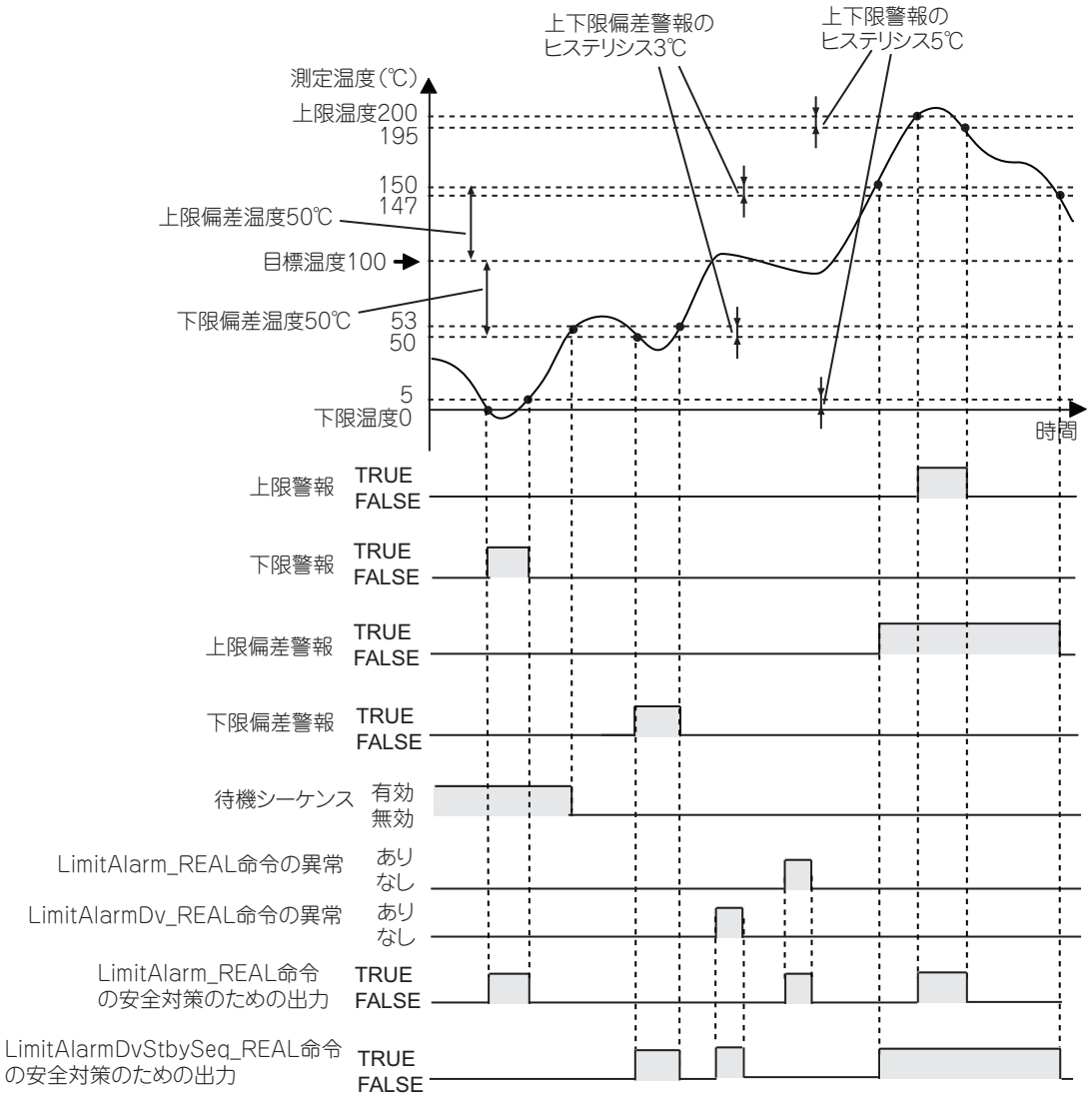
4 **LimitAlarmDvStbySeq_REAL 命令を使って、目標温度からの測定温度の偏差に対して待機シーケンス付き上下限偏差警報を出力します。**

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_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。

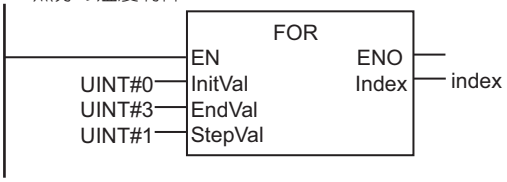
（注）I/O マップ設定により、各ユニットのポートに割り付けたグローバル変数が自動的に生成されます。

LD

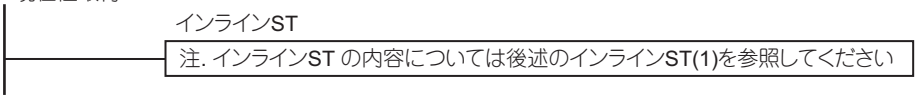
名称	データ型	初期値	保持	コメント
index	UINT	0	□	ループインデックス
LimitAlarm_ON	BOOL	True	□	上下限警報実行
LimitAlarmDvStbySeq_ON	BOOL	True	□	待機シーケンス付き上下限偏差警報実行
TimeProportionalOut_ON	BOOL	True	□	時分割比例出力実行
AI	INT	0	□	現在値
PV	ARRAY[0..3] OF REAL	[4(0.0)]	□	測定値
SP	ARRAY[0..3] OF REAL	[4(100)]	□	目標温度
DOut_TPO	BOOL	False	□	時分割比例出力
HighVal	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

名称	データ型	初期値	保持	コメント
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
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		□	

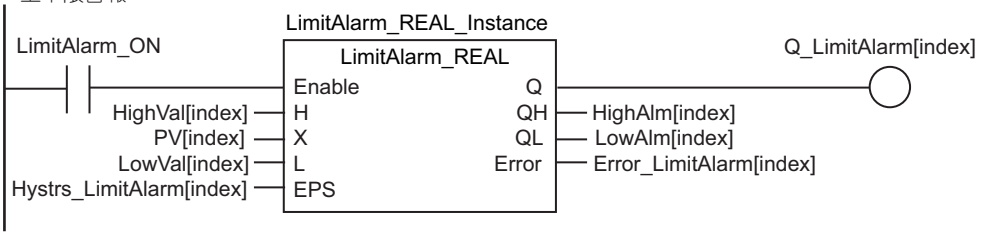
4点分の温度制御



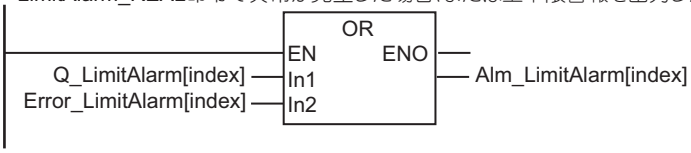
現在値取得



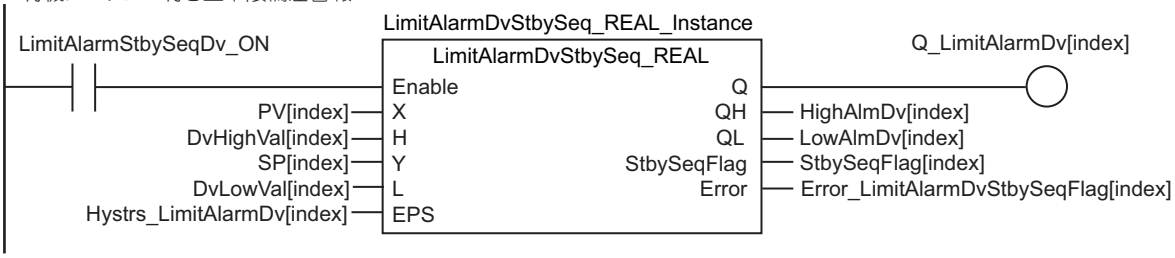
上下限警報



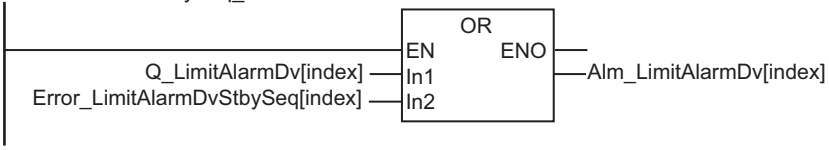
LimitAlarm_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力

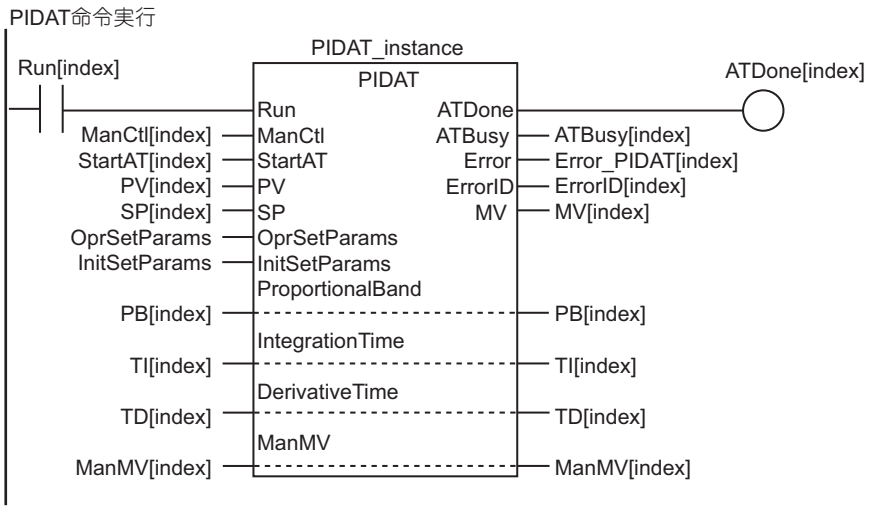


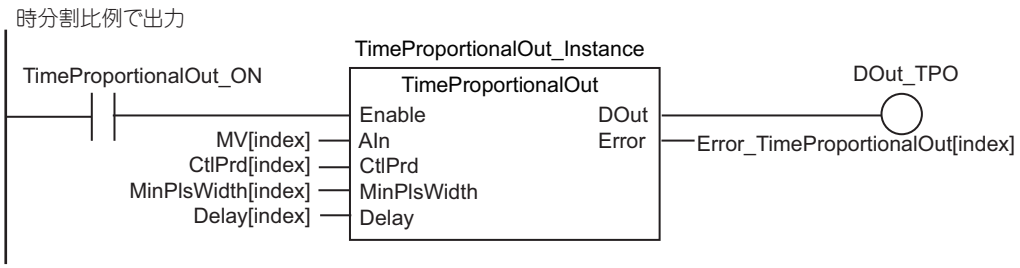
待機シーケンス付き上下限偏差警報

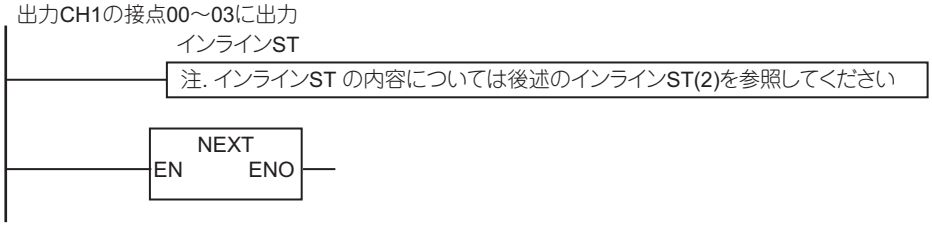


LimitAlarmDvStbySeq_REAL命令で異常が発生した場合、または上下限警報を出力した場合の安全対策のための出力









● 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]);

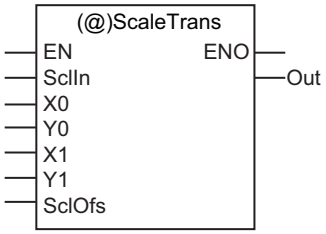
//出力 Ch1 の接点 00～03 に出力
CASE index OF
INT#0:
  DO1:=DOut_TPO;
INT#1:
  DO2:=DOut_TPO;
INT#2:
```

```
      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(SclIn, X0, Y0, X1, Y1, SclOfs);

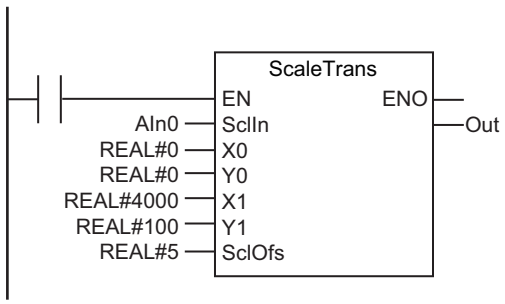
バージョン関連情報

本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。

記述例

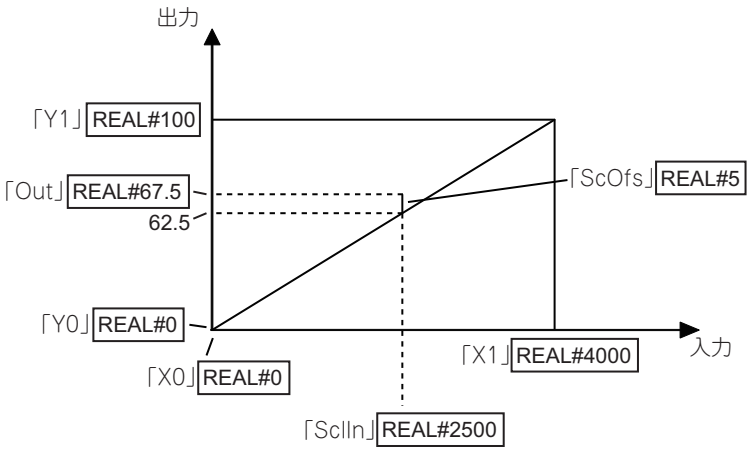
入力レンジが 0～4000、出力レンジが 0～100%で、入力値 2500 を変換する場合の記述例を以下に示します。出力に加えるオフセット値は 5%とします。
「SclIn」=REAL#2500、「X0」=REAL#0、「X1」= REAL#4000、「Y0」=REAL#0、「Y1」=REAL#100、「SclOfs」=REAL#5 とします。
「Out」の値は、REAL#67.5 となります。

LD



ST

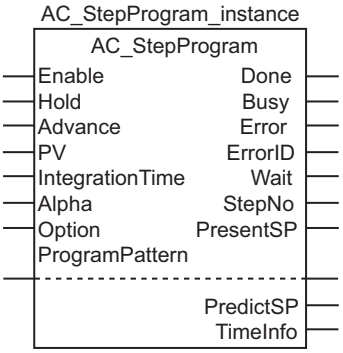
```
Out :=ScaleTrans(AIn0,REAL#0,REAL#0,REAL#4000,
                 REAL#100,REAL#5);
```



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。

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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.06 以降、かつ Sysmac Studio Ver.1.07 以降で、使用可能です。

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
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	オプション

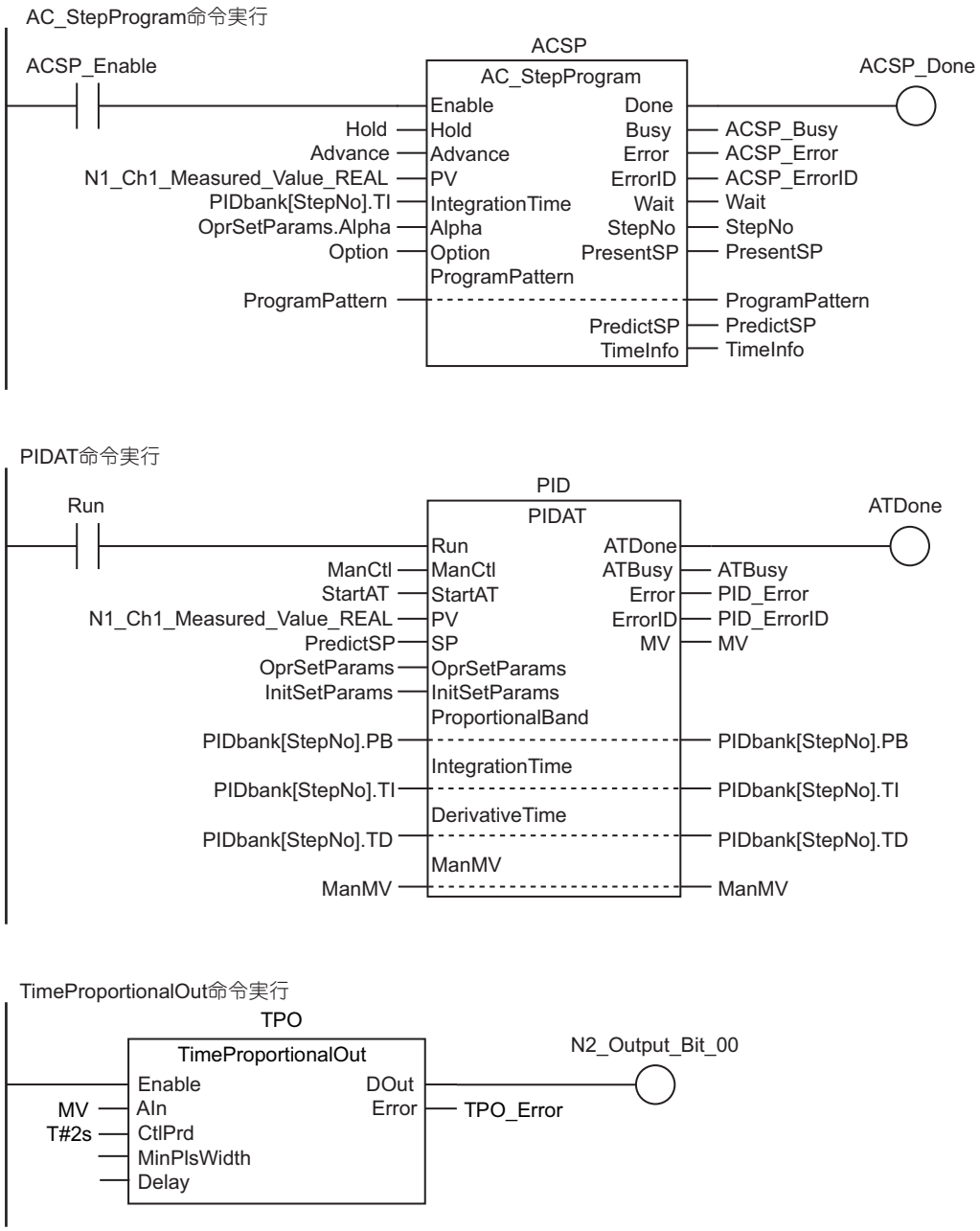
内部変数	名称	データ型	初期値	コメント
	ProgramPattern	ARRAY[0..7] OF _sAC_STEP_DATA	[(ReachSP:=30.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=100.0, TimeWidth:=T#10m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=120.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, 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
	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

内部変数	名称	データ型	初期値	コメント
	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

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





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
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	オプション

内部変数	名称	データ型	初期値	コメント
	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
	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

内部変数	名称	データ型	初期値	コメント
	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	TimeProportionalOut 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	TimeProportionalOut 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	TimeProportionalOut		

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 =>TP0_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.1-206
TraceTrig	データトレーストリガ条件成立	P.1-208
GetTraceStatus	データトレースステータス読出	P.1-209
SetAlarm	ユーザ異常発生	P.1-211
ResetAlarm	ユーザ異常解除	P.1-214
ResetPLCError	PLC 異常解除	P.1-215
ResetMCError	モーション制御異常解除	P.1-218
GetMCError	モーション制御異常ステータス取得	P.1-222
ResetECError	EtherCAT 異常解除	P.1-223
GetECError	EtherCAT 異常ステータス取得	P.1-224
GetNXUnitError	NX ユニット異常ステータス取得	P.1-225
ResetUnit	ユニットリスタート	P.1-227
RestartNXUnit	NX ユニットリスタート	P.1-230
NX_ChangeWriteMode	NX ユニット書込モード変更	P.1-231
NX_SaveParam	NX ユニットパラメータ保存	P.1-232
PLC_ReadTotalPowerOnTime	PLC 累積通電時間読出	P.1-233
NX_ReadTotalPowerOnTime	NX ユニット累積通電時間読出	P.1-235

TraceSamp

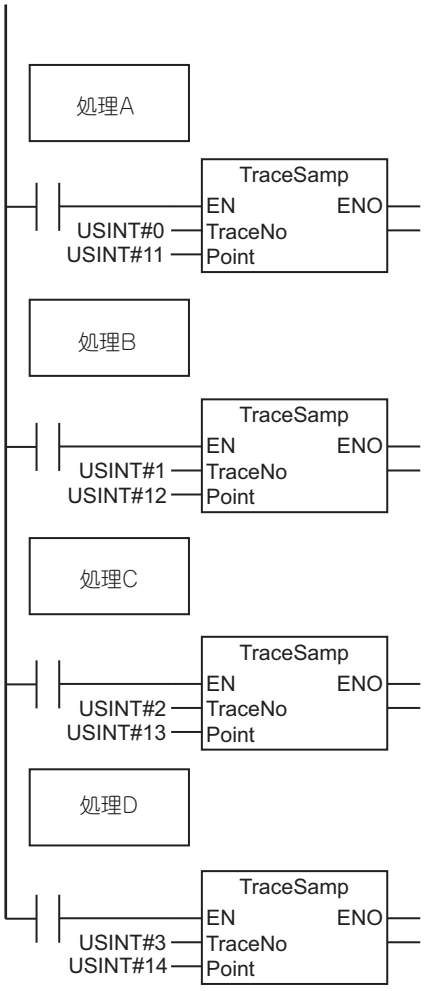
データトレースのサンプリングを実行します。

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);

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。

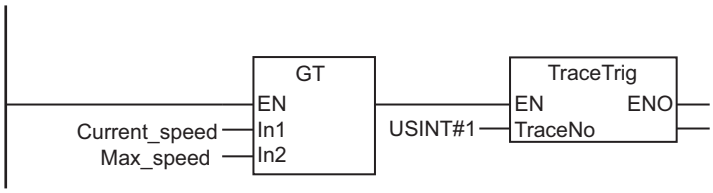
命令	名称	FB/FUN	グラフィック表現	ST 表現
TraceTrig	データトレーストリガ条件成立	FUN	(@)TraceTrig EN ENO TraceNo Out	TraceTrig(TraceNo);

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。

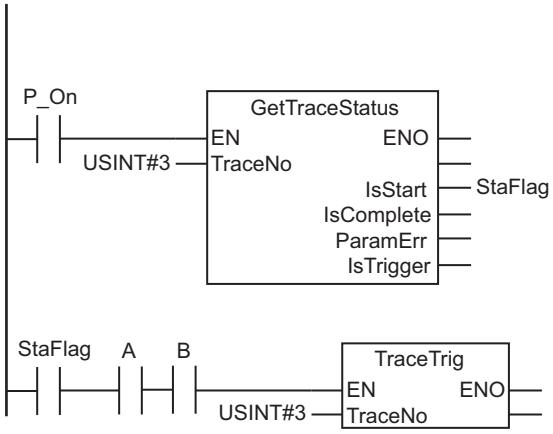
命令	名称	FB/FUN	グラフィック表現	ST 表現
GetTraceStatus	データトレースステータス読出	FUN	(@)GetTraceStatus EN TraceNo ENO Out IsStart IsComplete ParamErr IsTrigger	GetTraceStatus(TraceNo, IsStart, IsComplete, ParamErr, IsTrigger);

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

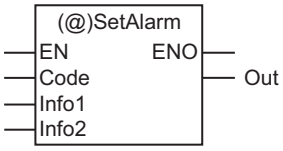
名称	データ型	初期値	コメント
StaFlag	BOOL	FALSE	トレースの実行ステータス
A	BOOL	FALSE	
B	BOOL	FALSE	

```
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);

サンプルプログラム

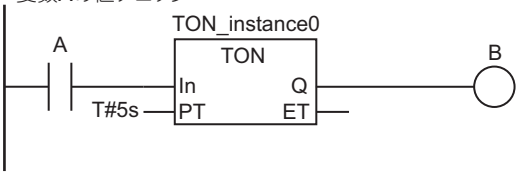
変数「A」の値は、正常時には、5s ごとに TRUE と FALSE を繰り返すものとします。
「A」の値を監視し、5s 以上値が変化しなかった場合に、イベントコード 102 のユーザ異常を発生させます。付属情報は、UINT#123 と UINT#456 とします。
変数「F」の値が TRUE になったら、ユーザ異常を解除します。

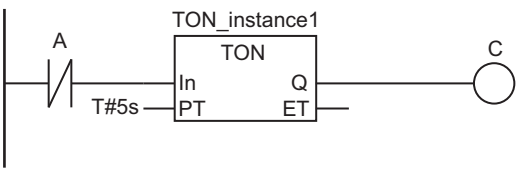
LD

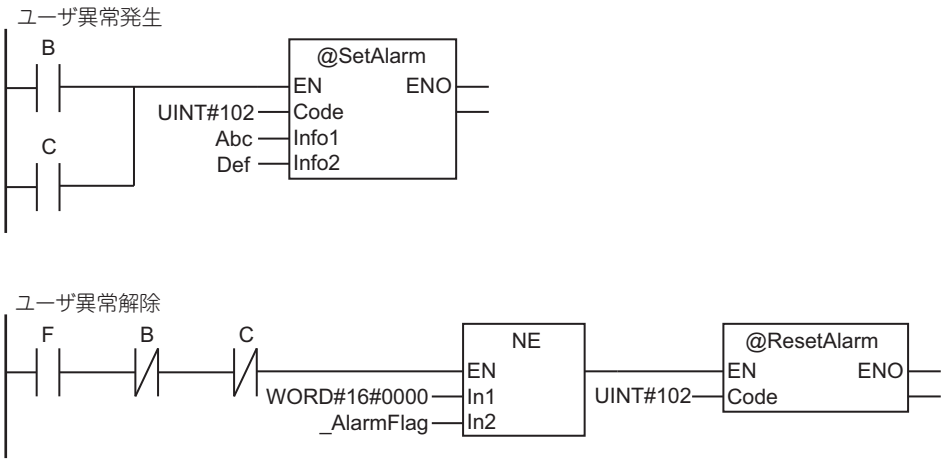
内部変数	名称	データ型	初期値
	A	BOOL	FALSE
	B	BOOL	FALSE
	C	BOOL	FALSE
	F	BOOL	FALSE
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

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

変数Aの値チェック







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;

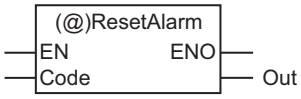
IF (A=FALSE) THEN
  TON_instance1(In:=TRUE, PT:=T#5s, Q=>C);
ELSE
  TON_instance1(In:=FALSE, Q=>C);
END_IF;

// 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。

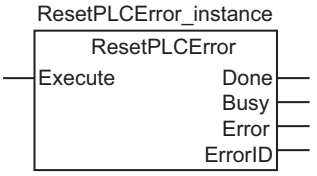
命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetAlarm	ユーザ異常解除	FUN		ResetAlarm(Code);

サンプルプログラム

SetAlarm 命令の「サンプルプログラム (P.1-211)」を参照してください。

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。

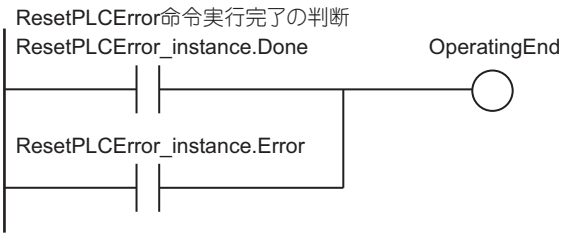
命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetPLCError	PLC 異常解除	FB		ResetPLCError(Execute, Done, Busy, Error, ErrorID);

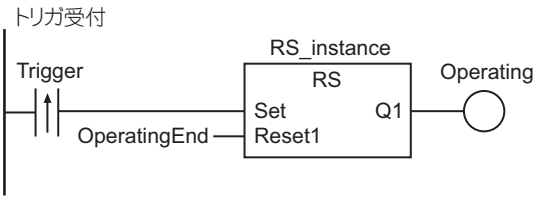
サンプルプログラム

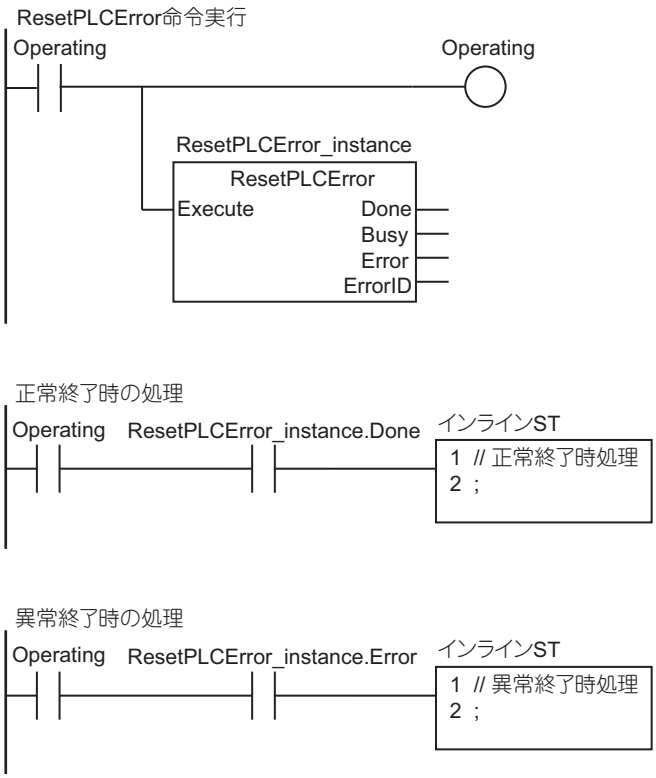
「Trigger」の値が FALSE→TRUE になったら、ResetPLCError 命令を実行します。ResetPLCError 命令が正常終了（「Done」の値が TRUE）したら、正常終了時処理を実行します。異常終了（「Error」の値が TRUE）したら、異常終了時処理を実行します。

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);

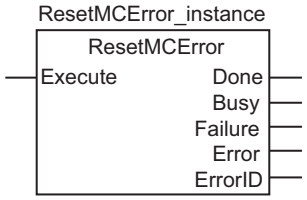
  IF (ResetPLCError_instance.Done=TRUE) THEN
```

```
    // 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;
```

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);

サンプルプログラム

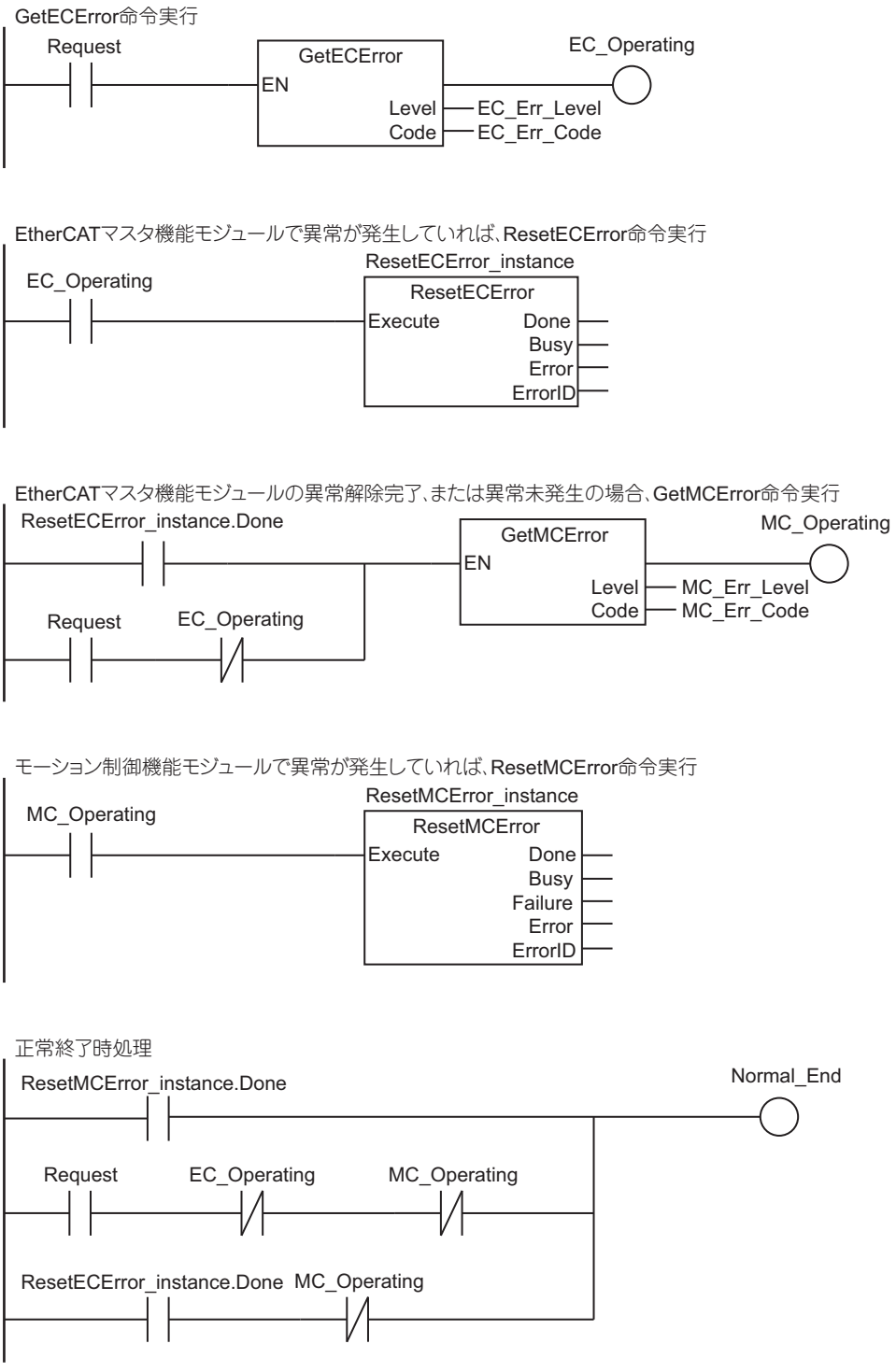
EtherCAT マスタ機能モジュールおよびモーション制御機能モジュールで発生中の、コントローラ異常を検知します。もし、異常が発生していたら、その異常を解除します。

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
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		



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

名称	データ型	初期値	コメント
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);

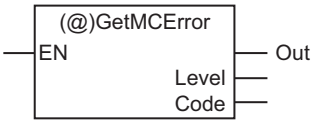
  IF (EC_Error=TRUE) THEN // EtherCAT マスタ機能モジュールのコントローラ異常あり
    CASE EC_Stage OF
    0 : // 初期化
      ResetECError_instance(Execute:=FALSE);
      EC_Stage:=INT#1;
    1 : // EtherCAT マスタ機能モジュールのコントローラ異常解除中
      ResetECError_instance(Execute:=TRUE);
      IF (ResetECError_instance.Done=TRUE) THEN
        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。

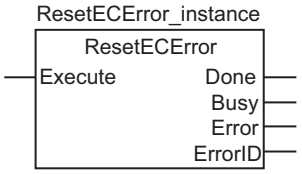
命令	名称	FB/FUN	グラフィック表現	ST 表現
GetMCError	モーション制御異常ステータス取得	FUN		Out:=GetMCError(Level, Code);

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.1-218)」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。

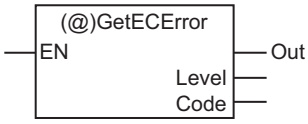
命令	名称	FB/FUN	グラフィック表現	ST 表現
ResetECError	EtherCAT 異常解除	FB		ResetECError_instance(Execute, Done, Busy, Error, ErrorID);

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.1-218)」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);

バージョン関連情報

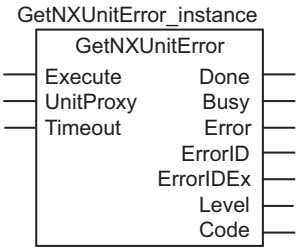
本命令でスレーブ異常が検出できるのは、CPU ユニット Ver.1.02 以降です。

サンプルプログラム

ResetMCError 命令の「サンプルプログラム (P.1-218)」を参照してください。

GetNXUnitError

NX シリーズ CPU ユニットの NX バス機能モジュールまたは NX ユニットで発生中のコントローラ異常の、最重要のステータスと、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
GetNXUnitError	NX ユニット異常ステータス取得	FB		GetNXUnitError_instance(Execute, UnitProxy, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, Level, Code);

使用上の注意

本命令は、形 NX502、形 NX102 および形 NX1P2 CPU ユニットで使用することができます。

サンプルプログラム

NX バス上で異常が発生した場合、発生しているユニット、レベル、およびコードを、タッチパネル表示用の変数に値を渡します。

次のようなシステム構成とします。
形 NX1P2 CPU ユニットに 3 つの NX ユニットを接続したものとします。

デバイス変数

NX ユニット	デバイス変数
1 つめの NX ユニット	N1_Node_location_information
2 つめの NX ユニット	N2_Node_location_information
3 つめの NX ユニット	N3_Node_location_information

グローバル変数の定義

● **グローバル変数**

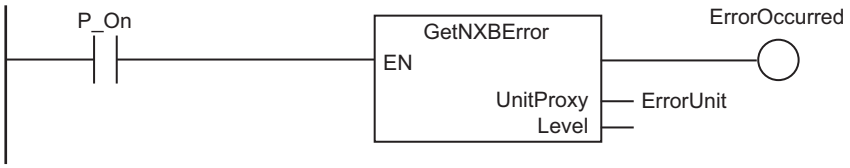
名称	データ型	コンスタント	コメント
N1_Node_location_information	_sNXUNIT_ID	☑	
N2_Node_location_information	_sNXUNIT_ID	☑	
N3_Node_location_information	_sNXUNIT_ID	☑	
HMI_Level *1	UINT	☐	
HMI_Code *1	DWORD	☐	

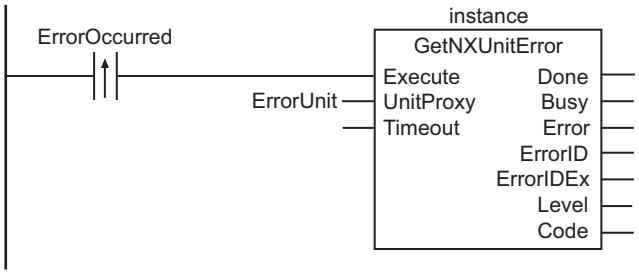
名称	データ型	コンスタント	コメント
HMI_Unit *1	UDINT	☐	

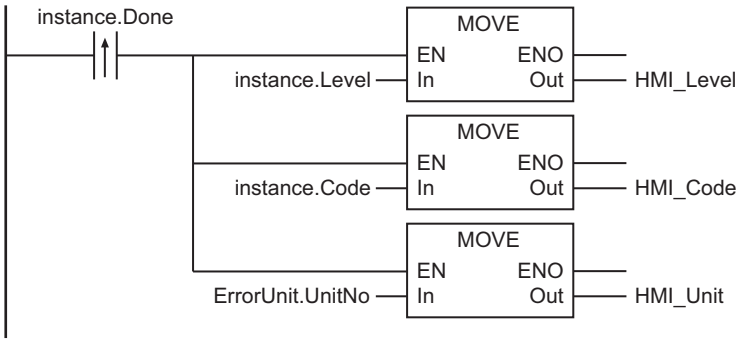
*1. 「HMI_」で始まる名称はタッチパネル表示用の変数です。

LD

内部変数	名称	データ型	コンスタント	コメント
	instance	GetNXUnitError	☐	
	ErrorOccurred	BOOL	☐	
	ErrorUnit	_sNXUNIT_ID	☐	

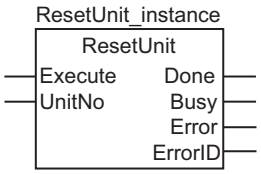






ResetUnit

CPU 高機能ユニット、または高機能 I/O 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
ResetUnit	ユニットリスタート	FB		ResetUnit_instance(Execute, UnitNo, Done, Busy, Error, ErrorID);

使用上の注意

本命令は、NX シリーズ CPU ユニットでは使用できません。

サンプルプログラム

「Trigger」の値が FALSE→TRUE になったら、ユニット番号 0 のシリアルコミュニケーションユニットのシリアルポート 1 の伝送速度を 38400bps に設定したうえで、ユニットのリスタートを行います。

グローバル変数の定義

● **グローバル変数**

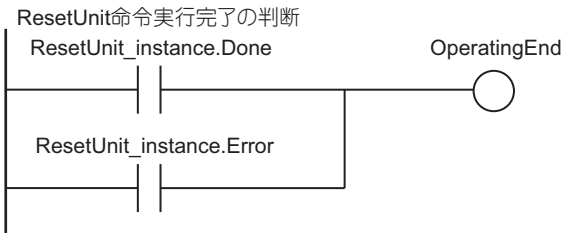
名称	データ型	初期値	割付先 *1	保持	コメント
SCU_P1_BaudrateCfg	USINT	0	IOBus://rack#0/slot#0/P1_BaudrateCfg	☑	伝送速度

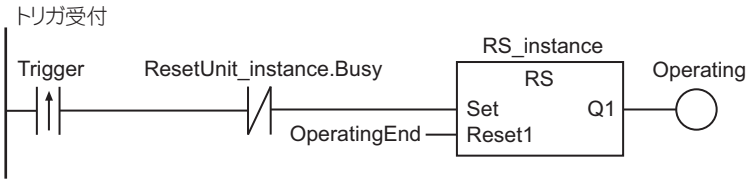
*1. シリアルコミュニケーションユニットをラック番号 0 のスロット番号 0 に装着した場合の割付先です。

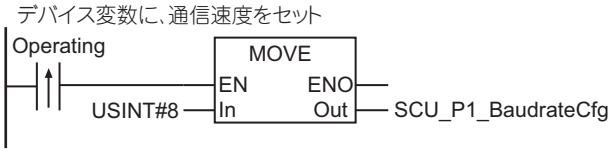
LD

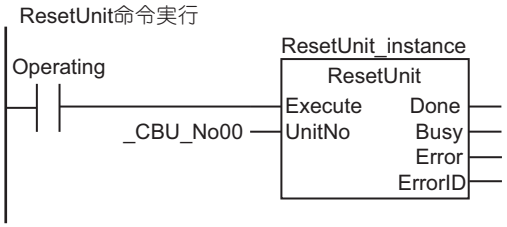
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	ResetUnit_instance	ResetUnit		

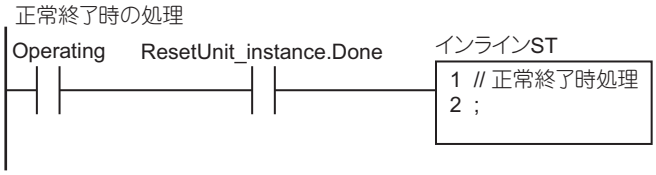
外部変数	名称	データ型	コメント
	SCU_P1_BaudrateCfg	USINT	伝送速度

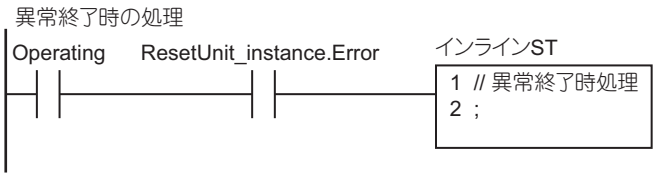












ST

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
	ResetUnit_instance	ResetUnit		

外部変数	名称	データ型	コメント
	SCU_P1_BaudrateCfg	USINT	伝送速度

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (ResetUnit_instance.Busy=FALSE) ) THEN
  OperatingStart:=TRUE;
  Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// ResetUnit_instance 初期化と、デバイス変数に伝送速度セット
IF (OperatingStart=TRUE) THEN
  ResetUnit_instance(Execute:=FALSE);
  SCU_P1_BaudrateCfg:=USINT#8;
  OperatingStart:=FALSE;
END_IF;

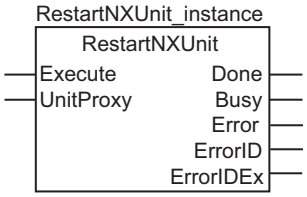
// ResetUnit 命令実行
IF (Operating=TRUE) THEN
  ResetUnit_instance(
    Execute:=TRUE, // 起動条件
    UnitNo :=_CBU_No00); // ユニット番号

  IF (ResetUnit_instance.Done=TRUE) THEN
    // 正常終了時処理
    Operating:=FALSE;
  END_IF;

  IF (ResetUnit_instance.Error=TRUE) THEN
    // 異常終了時処理
    Operating:=FALSE;
  END_IF;
END_IF;
```

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

本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。
ただし、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。

- CPU ユニット
- Sysmac Studio
- EtherCAT カプラユニット
- NX ユニット

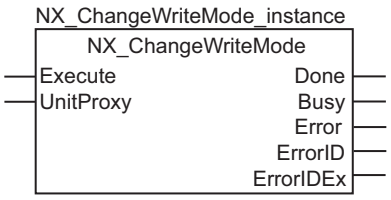
「指定した NX ユニットをリスタートする機能」に対応していないバージョンを使用する場合は、リスタートするユニットに EtherCAT カプラユニットだけ指定することが可能です。
「指定した NX ユニットをリスタートする機能」に対応しているバージョンは、『NX シリーズ EtherCAT カプラユニットユーザーズマニュアル(SBCD-361)』を参照してください。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.1-289)」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);

バージョン関連情報

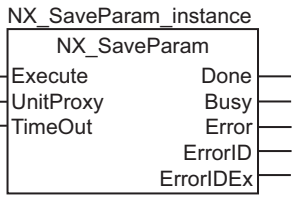
本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.1-289)」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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。

サンプルプログラム

NX_WriteObj 命令の「サンプルプログラム (P.1-289)」を、参照してください。

PLC_ReadTotalPowerOnTime

指定した CPU 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
PLC_ReadTotalPowerOnTime	PLC 累積通電時間読出	FUN	PLC_ReadTotalPowerOnTime EN　ENO UnitType　Out	Out:=PLC_ReadTotalPowerOnTime(UnitType);

使用上の注意

本命令は、形 NX502、形 NX102 および形 NX1P2 CPU ユニットで使用することができます。

サンプルプログラム

プログラム上で Maintenance_Mode と Run_Mode の 2 つを作ります。
このうち、Maintenance_Mode 時に累積時間読み出しボタンを押すと、形 NX1P2 CPU ユニット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 シリーズ CPU ユニット上の NX バスに接続されているユニット 1 NX シリーズ I/O ユニット(ID)
ユニット 2	NX シリーズ CPU ユニット上の NX バスに接続されているユニット 2 NX シリーズ I/O ユニット(IO)
ユニット 3	形 NX1P2（追跡通電時間を読み出したい対象）

ST

外部変数	名称	データ型	コメント
	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
	J02_Ch1_Out00	BOOL	ユニット交換お知らせランプ
	Maintenance_Mode	BOOL	
	Run_Mode	BOOL	
	PushButton_Read	BOOL	
	Lamp_Warning_UnitLifeTime	BOOL	
	PowerOnTime	TIME	
	R_TRIG_instance1	R_TRIG	
	PushButton_Read_R_TRIG	BOOL	
	RS_instance	RS	

```
// 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;

R_TRIG_instance1(clk:=PushButton_Read, Q=>PushButton_Read_R_TRIG);

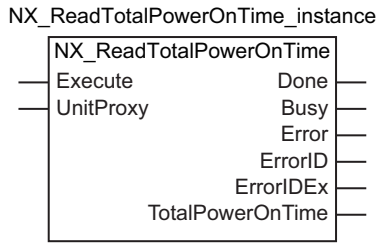
// 累積稼働時間読み出し
PowerOnTime := PLC_ReadTotalPowerOnTime(EN:=(Maintenance_Mode & PushButton_Read_R_TRIG),
  UnitType:=_CPU_UNIT);

RS_instance( Set:=(PowerOnTime > T#1825d),
  Reset1:=Maintenance_Mode,
  Q1=>Lamp_Warning_UnitLifeTime);

// ランプへ出力
J02_Ch1_Out00 := Lamp_Warning_UnitLifeTime;
```

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(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx, TotalPowerOnTime);

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。

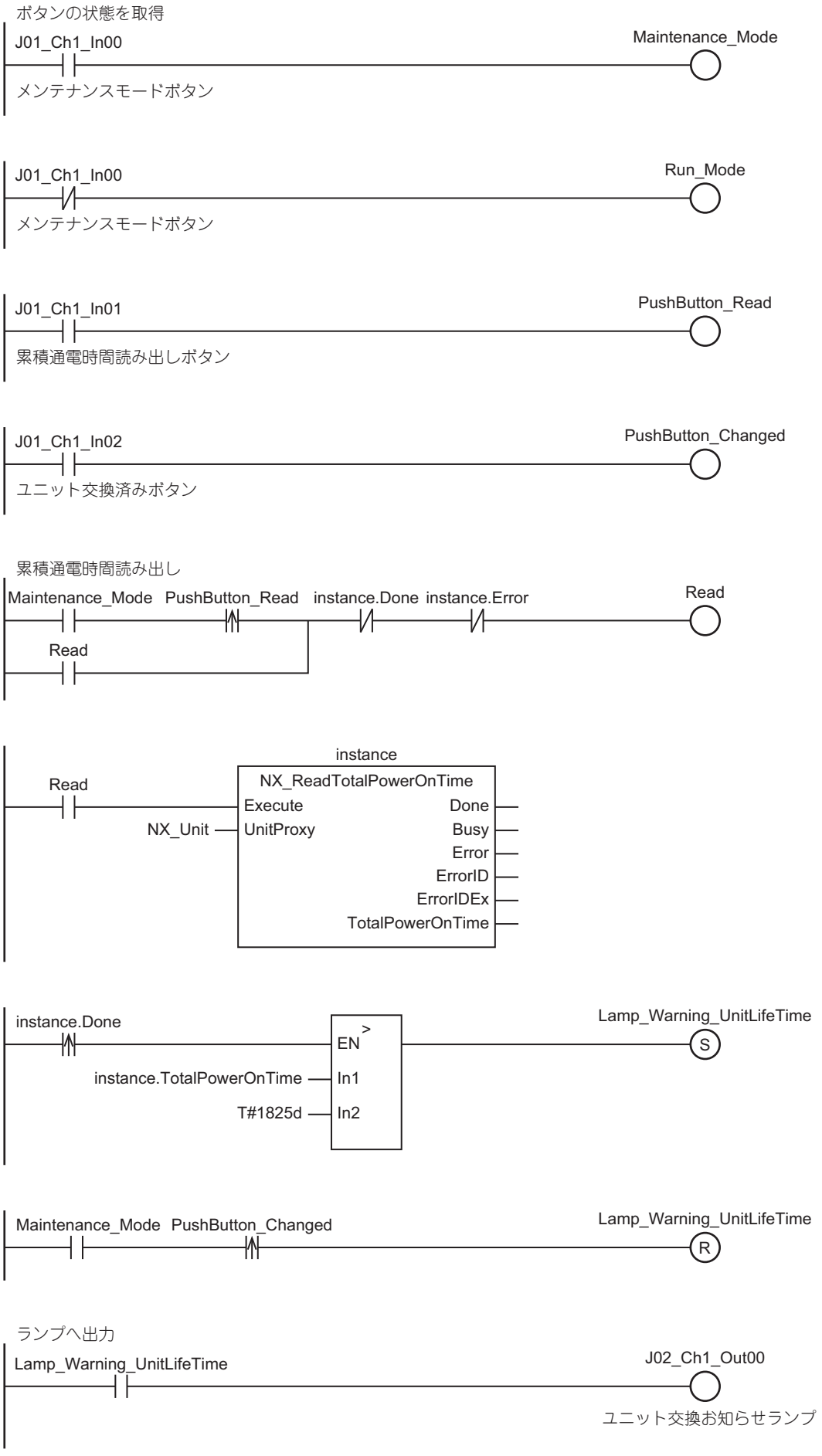
次のようなシステム構成とします。

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	
	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


ボタンの状態を取得
J01_Ch1_In00
Maintenance_Mode
メンテナンスモードボタン
J01_Ch1_In00
Run_Mode
メンテナンスモードボタン
J01_Ch1_In01
PushButton_Read
累積通電時間読み出しボタン
J01_Ch1_In02
PushButton_Changed
ユニット交換済みボタン
累積通電時間読み出し
Maintenance_Mode
PushButton_Read
instance.Done
instance.Error
Read
Read
instance
Read
NX_ReadTotalPowerOnTime
Execute
Done
NX_Unit
UnitProxy
Busy
Error
ErrorID
ErrorIDEx
TotalPowerOnTime
instance.Done
Lamp_Warning_UnitLifeTime
EN
>
instance.TotalPowerOnTime
In1
T#1825d
In2
S
Maintenance_Mode
PushButton_Changed
Lamp_Warning_UnitLifeTime
R
ランプへ出力
Lamp_Warning_UnitLifeTime
J02_Ch1_Out00
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);
```

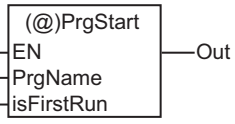
```
// ランプへ出力
J02_Ch1_Out00 := Lamp_Warning_UnitLifeTime;
```

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.1-240
PrgStop	プログラム停止指令	P.1-241
PrgStatus	プログラム状態読出	P.1-252

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		Out:=PrgStart(PrgName, isFirstRun);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

サンプルプログラム

PrgStop 命令の「サンプルプログラム (P.1-241)」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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

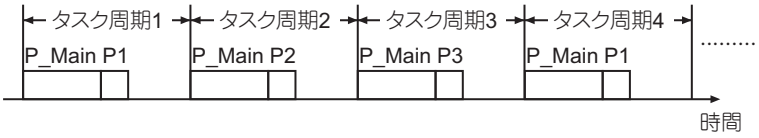
サンプルプログラム

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

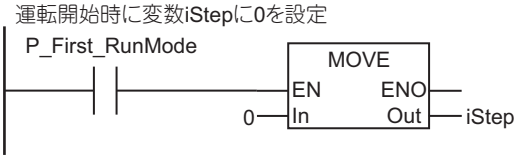
3 つのプログラムを、タスク周期ごとに順番に実行させる例を示します。

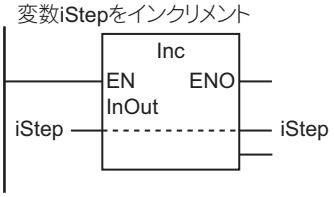
P1、P2 および P3 の 3 つのプログラムがあります。
これらを、タスク周期ごとに順番に繰り返し実行させます。
3 つのプログラムへの実行と停止の指令を出すプログラムは、P_Main です。

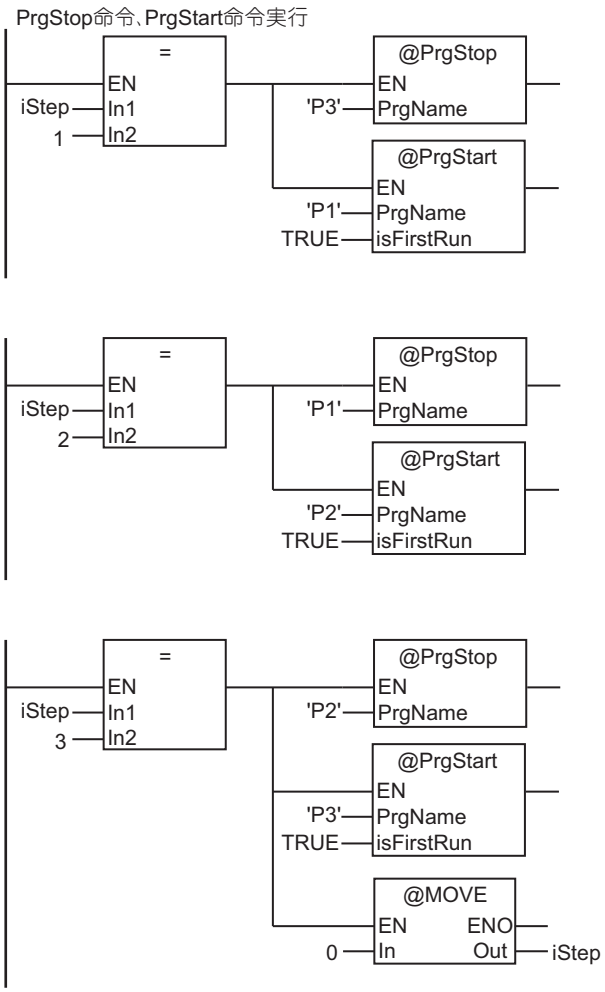


- **LD**

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

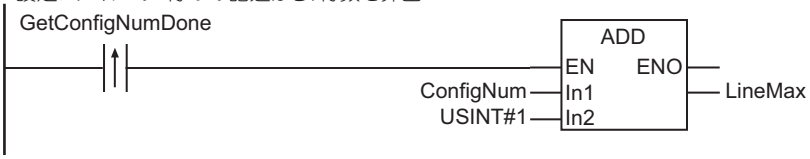
名称	データ型	初期値	保持	コメント
Open	FileOpen		☐	FileOpen 命令のインスタンス
TopLineGetter	FileGets		☐	FileGets 命令のインスタンス
LineGetter	FileGets		☐	FileGets 命令のインスタンス
Close	FileClose		☐	FileClose 命令のインスタンス
PTInput_TargetConfigNum_Retain	USINT	0	☑	次回運転時に実行させるモジュール構成番号
CurrentLineNum	USINT	1	☐	設定ファイルの、現在読出中の行
TargetLineNum	USINT	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

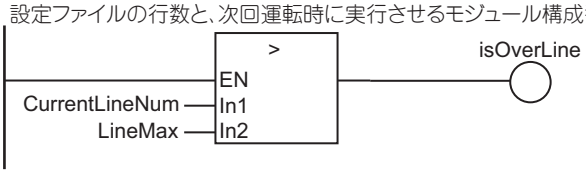
次回運転時に実行させるモジュール構成番号取得



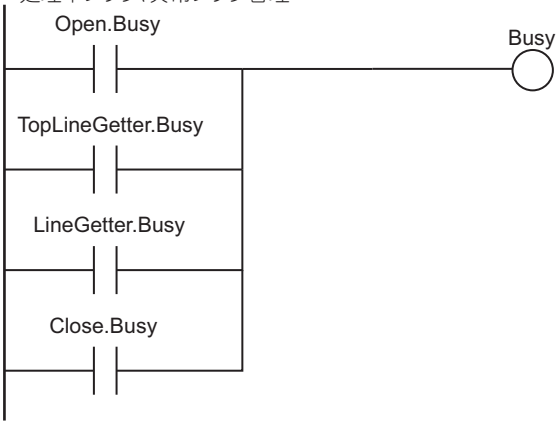
設定ファイルの1行めの記述から、行数を算出

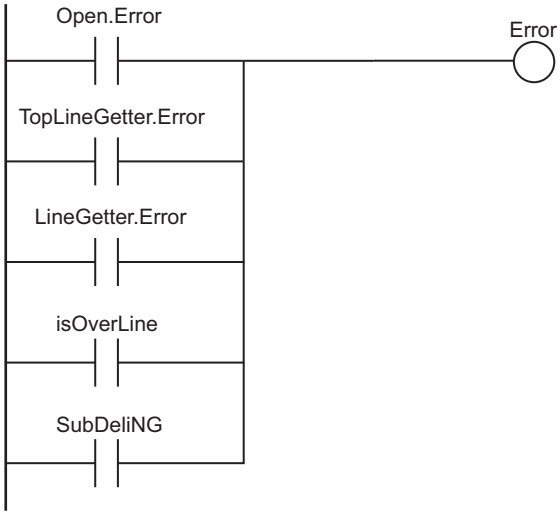


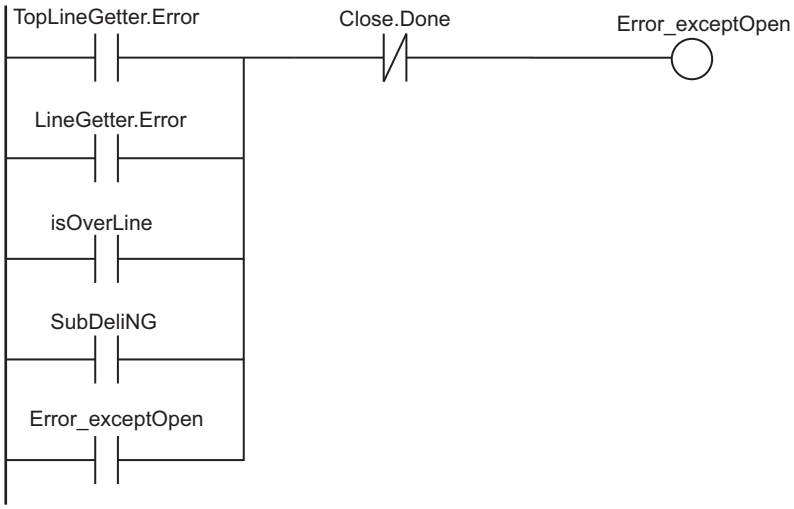
設定ファイルの行数と、次回運転時に実行させるモジュール構成番号が合わない異常検知

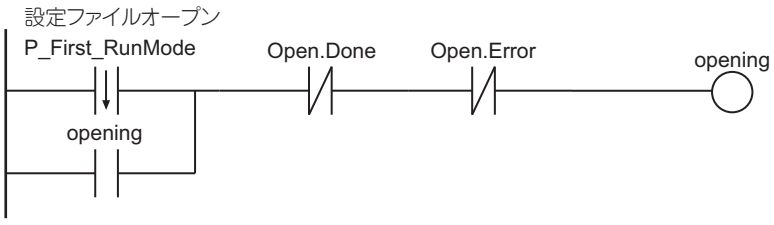


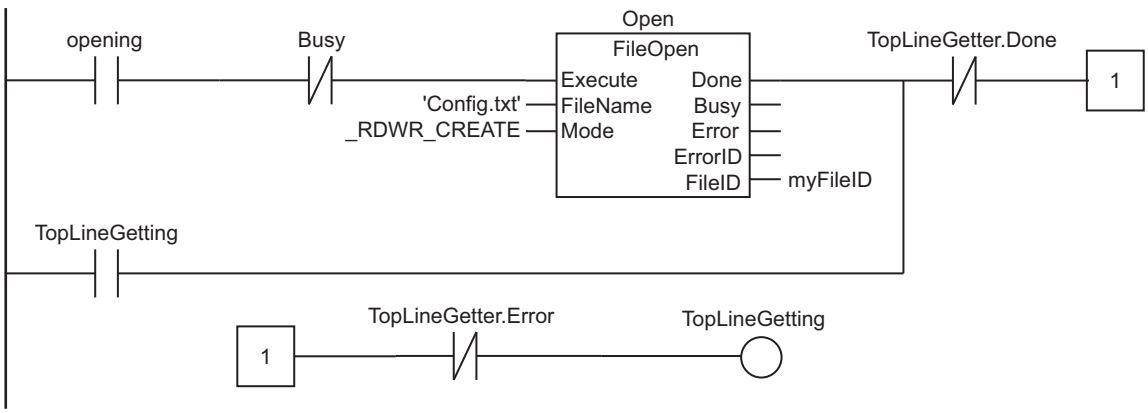
処理中フラグ、異常フラグ管理



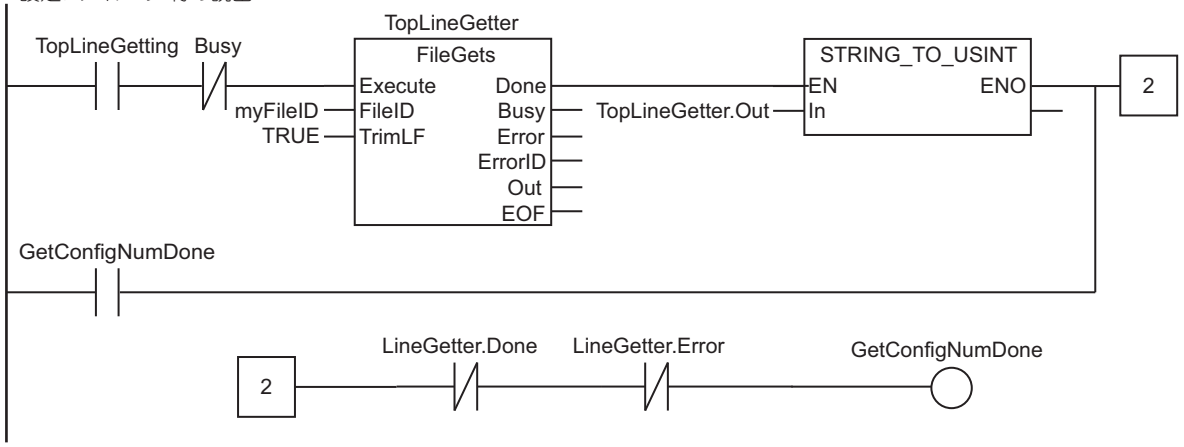

Open.Error
Error
TopLineGetter.Error
LineGetter.Error
isOverLine
SubDeliNG

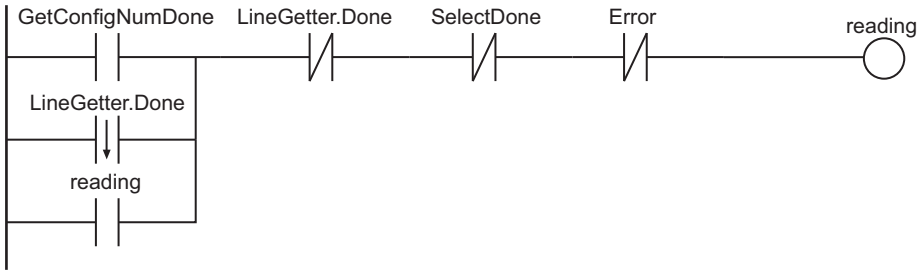

TopLineGetter.Error
Close.Done
Error_exceptOpen
LineGetter.Error
isOverLine
SubDeliNG
Error_exceptOpen


設定ファイルオープン
P_First_RunMode
Open.Done
Open.Error
opening
opening

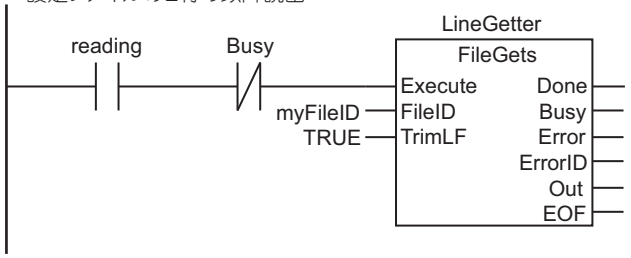

opening
Busy
Open
FileOpen
Execute
Done
'Config.txt'
FileName
Busy
_RDWR_CREATE
Mode
Error
ErrorID
FileID
myFileID
TopLineGetter.Done
1
TopLineGetting
TopLineGetter.Error
TopLineGetting
1

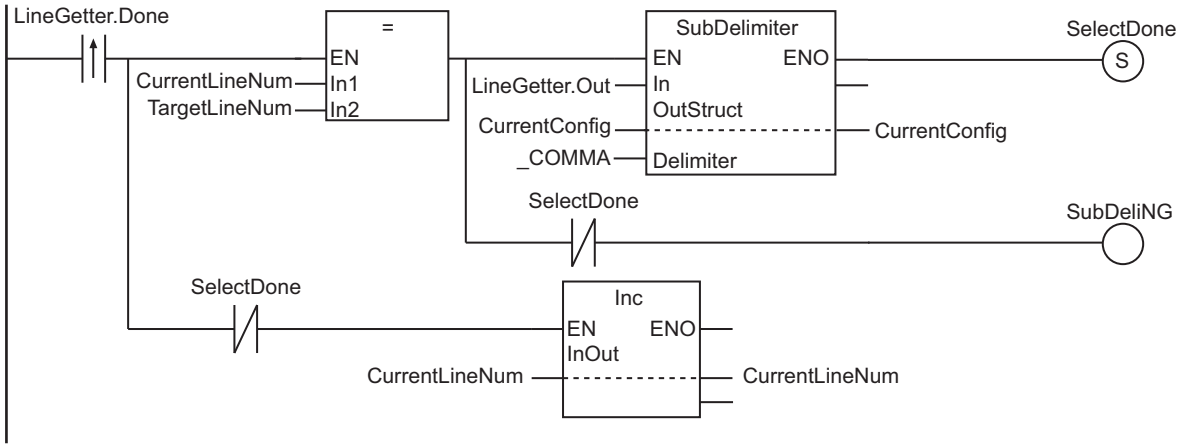
設定ファイルの1行め読出

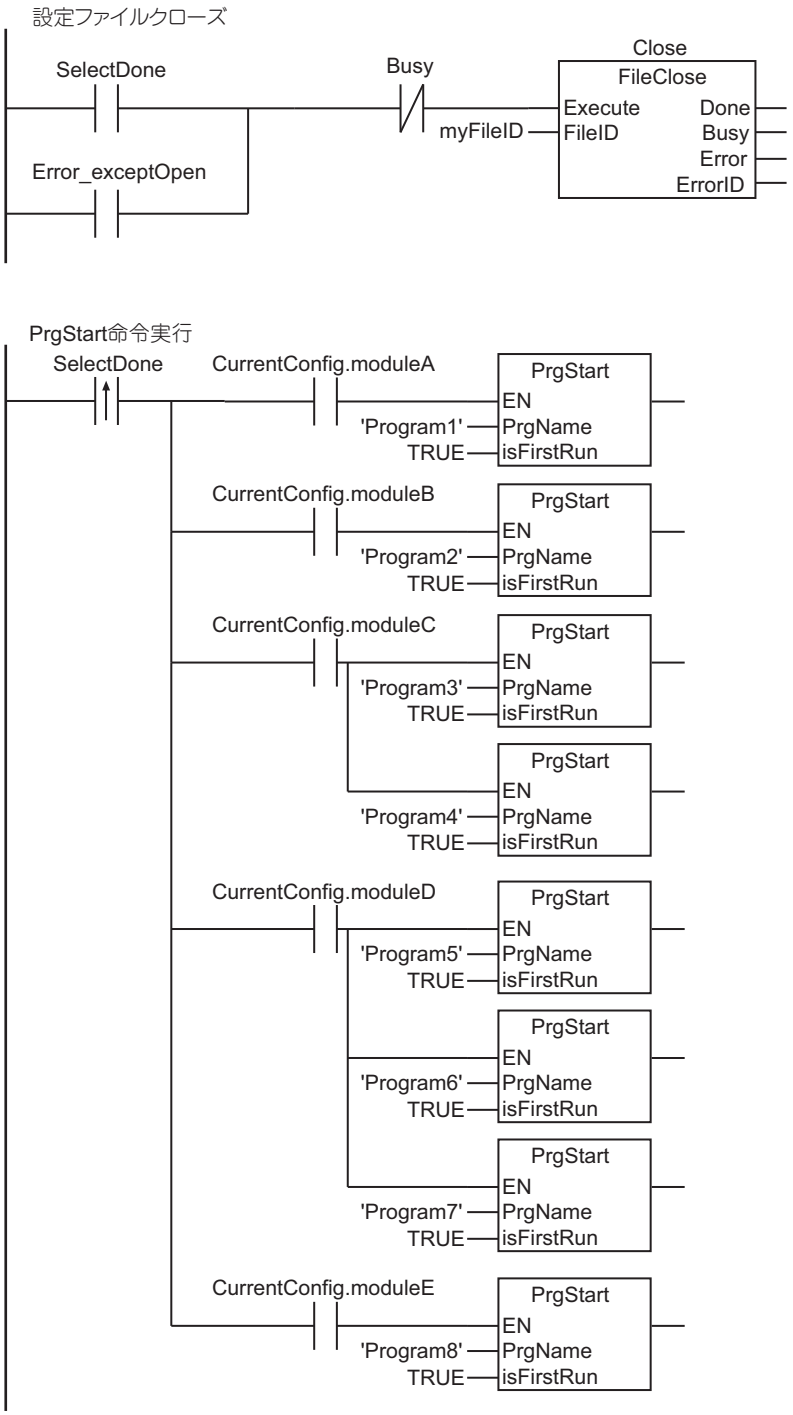




設定ファイルの2行め以降読出







● 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.Don
e, 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 Erro
r), Q1=>reading);

// 設定ファイルの 2 行め以降読出
```

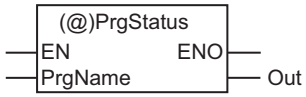
```
LineGetter(Execute:=(reading & NOT(Busy)), FileID:=myFileID, TrimLF := TRUE);
R_LineGetterDone(Clk:=LineGetter.Done);
isTargetLine := (CurrentLineNum = TargetLineNum);
SubDeliCondition := (R_LineGetterDone.Q & isTargetLine);
SubDelimiter(EN := SubDeliCondition, In := LineGetter.Out, OutStruct := 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		Out:=PrgStatus(PrgName);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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

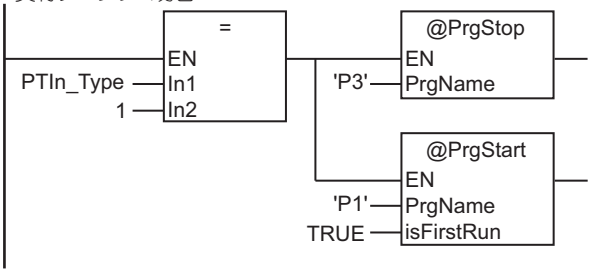
名称	データ型	初期値	コメント
PTIn_Type	INT	0	実行プログラム切替ボタン入力
PTOut_P1Status	BOOL	FALSE	P1 実行中ランプ出力
PTOut_P2Status	BOOL	FALSE	P2 実行中ランプ出力
PTOut_P3Status	BOOL	FALSE	P3 実行中ランプ出力

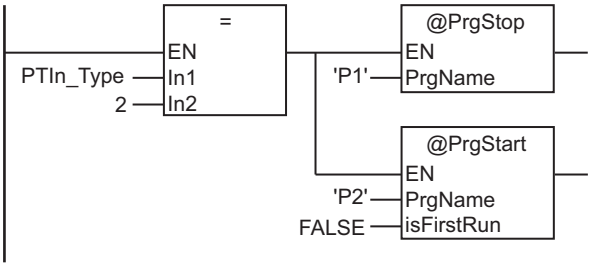
LD

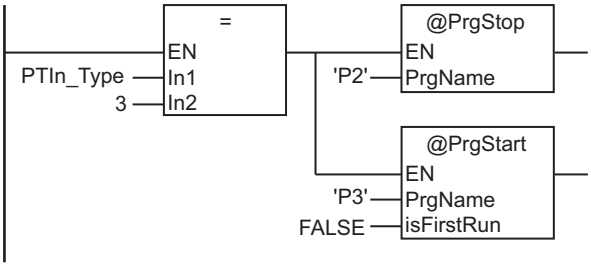
外部変数	名称	データ型	コメント
	PTIn_Type	INT	実行プログラム切替ボタン入力

外部変数	名称	データ型	コメント
	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

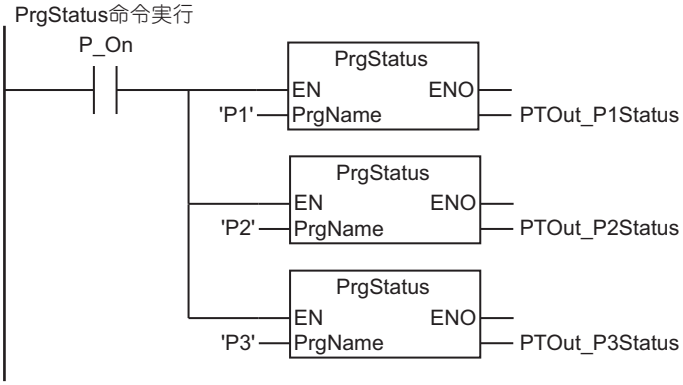
実行プログラム切替







PrgStatus命令実行



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_CoESDORead	CoE SDO 読出	P.1-256
EC_StartMon	EtherCAT パケットモニタ機能開始	P.1-259
EC_StopMon	EtherCAT パケットモニタ機能停止	P.1-264
EC_SaveMon	EtherCAT パケット保存	P.1-265
EC_CopyMon	EtherCAT パケット転送	P.1-266
EC_DisconnectSlave	EtherCAT スレーブ離脱	P.1-267
EC_ConnectSlave	EtherCAT スレーブ再加入	P.1-273
EC_ChangeEnableSetting	EtherCAT スレーブ有効／無効切替	P.1-274
NX_WriteObj	NX オブジェクト書込	P.1-289
NX_ReadObj	NX オブジェクト読出	P.1-299

EC_CoESDORead

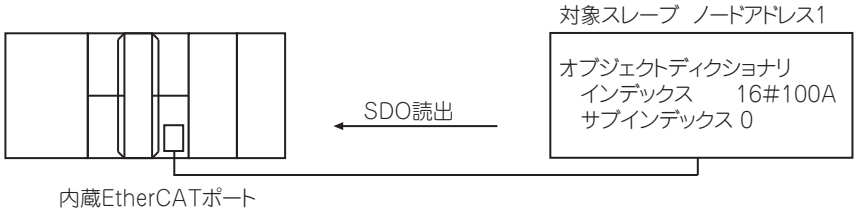
EtherCAT ネットワーク上の、指定したスレーブが持つ CoE（CAN application protocol over EtherCAT の略）のオブジェクトから、値を読み出します。

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 EC_CoESDORead Execute / NodeAdr / SdoObj / TimeOut / ReadDat Done / Busy / Error / ErrorID AbortCode / ReadSize	EC_CoESDORead_instance(Execute, NodeAdr, SdoObj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, AbortCode, ReadSize);

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。

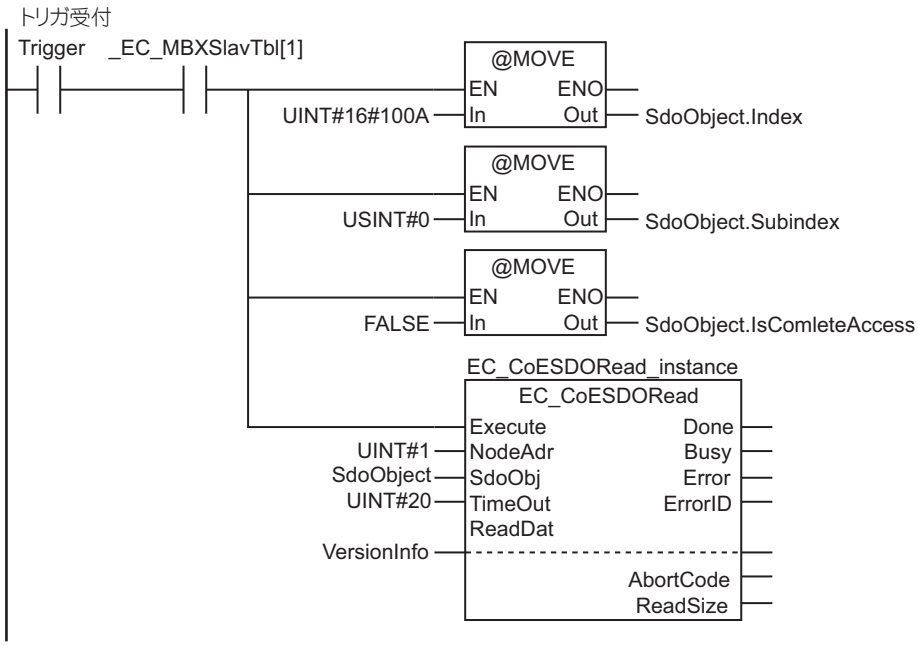


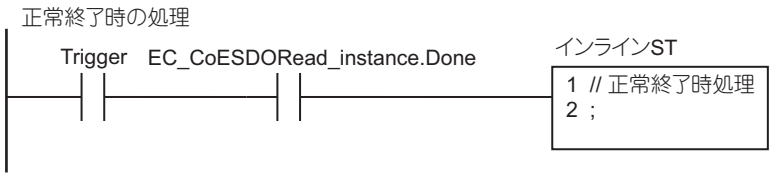
LD

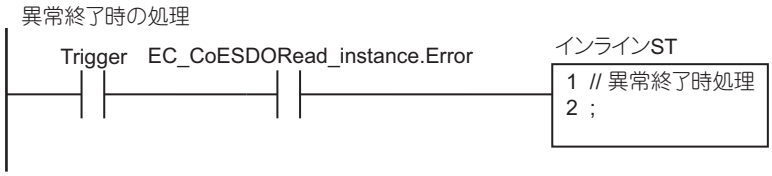
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO パラメータ
	VersionInfo	STRING[256]	''	読出データ
	EC_CoESDORead_instance	EC_CoESDORead		

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 *1	☑	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。







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
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO パラメータ
	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 *1	☑	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。

形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

```
// 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;

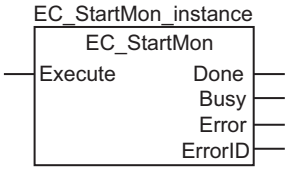
// EC_CoESDORead 命令実行
IF (DoSdoRead=TRUE) THEN
  EC_CoESDORead_instance(
    Execute:=TRUE,
    NodeAdr:=UINT#1, // ノードアドレス 1
    SdoObj :=SdoObject, // SDO パラメータ
    TimeOut:=UINT#20, // タイムアウト時間 2.0s
    ReadDat:=VersionInfo); // 読出データ

  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、CPU ユニット、および Sysmac Studio のバージョンにより、以下のような制約があります。

- 設定ユニットバージョン Ver.1.40 以降は、本命令を使用できません。
- 形 NX701 CPU ユニット、および形 NJ101 CPU ユニットは、Sysmac Studio Ver.1.13 以降で使用可能です。
- 形 NX1P2 CPU ユニットは、Sysmac Studio Ver.1.17 以降で使用可能です。
- 形 NJ301 CPU ユニットは、CPU ユニット Ver.1.10 以降、かつ Sysmac Studio Ver.1.12 以降で使用可能です。

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。**

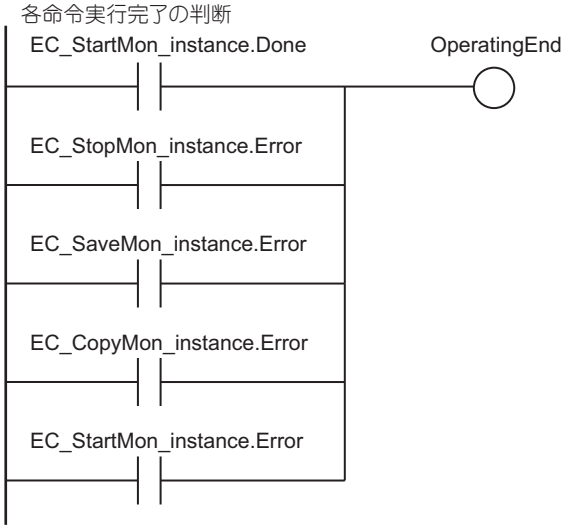
LD

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		

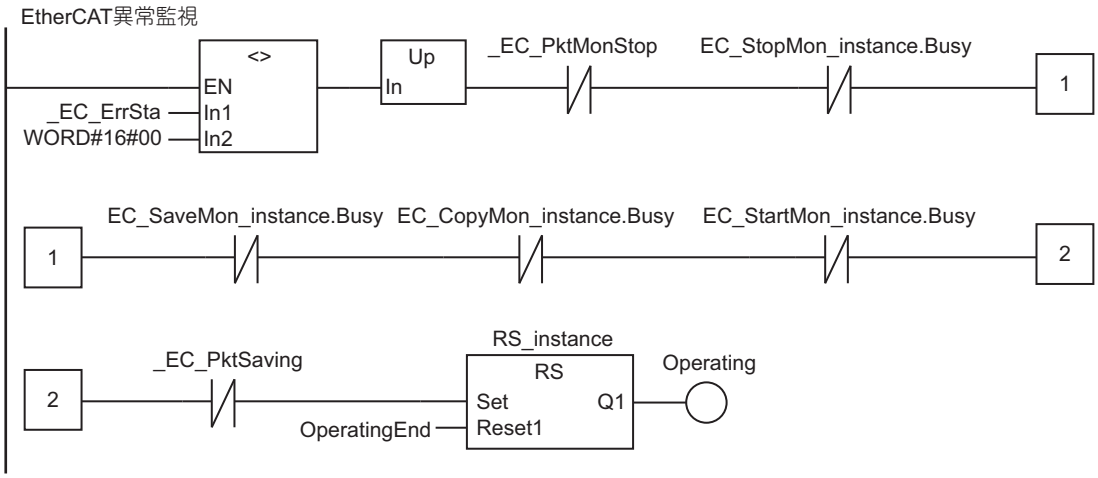
内部変数	名称	データ型	初期値	コメント
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

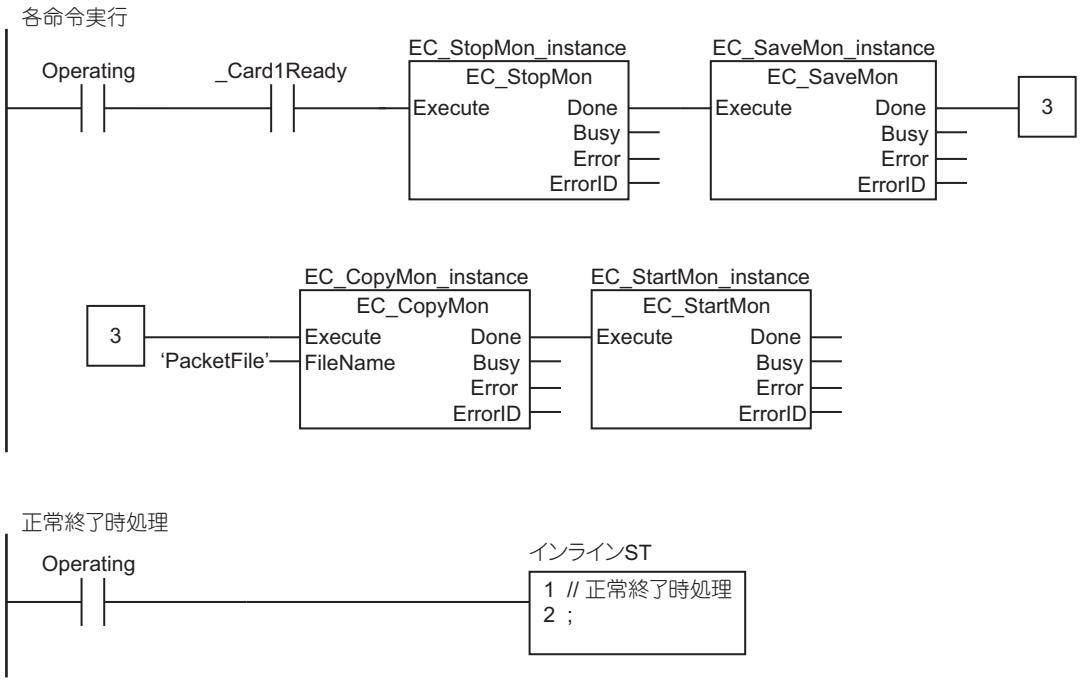
外部変数	名称	データ型	コンスタント	コメント
	_EC_ErrSta	WORD	☑	EtherCAT 異常
	_EC_PktMonStop	BOOL	☑	パケットモニタ停止中
	_EC_PktSaving	BOOL	☑	パケットデータファイル保存中
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ

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;

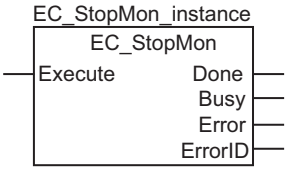
    4 : // EtherCAT パケットモニタ機能再開
      EC_StartMon_instance(
        Execute:=TRUE);
      IF (EC_StartMon_instance.Done=TRUE) THEN
        Stage:=INT#0; // 正常終了
      ELSIF (EC_StartMon_instance.Error=TRUE) THEN
        Stage:=INT#40; // 異常終了
      END_IF;

    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、CPU ユニット、および Sysmac Studio のバージョンにより、以下のような制約があります。

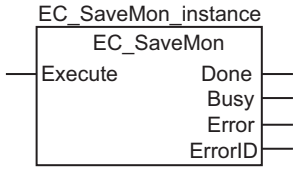
- 設定ユニットバージョン Ver.1.40 以降は、本命令を使用できません。
- 形 NX701 CPU ユニット、および形 NJ101 CPU ユニットは、Sysmac Studio Ver.1.13 以降で使用可能です。
- 形 NX1P2 CPU ユニットは、Sysmac Studio Ver.1.17 以降で使用可能です。
- 形 NJ301 CPU ユニットは、CPU ユニット Ver.1.10 以降、かつ Sysmac Studio Ver.1.12 以降で使用可能です。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.1-259)」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、CPU ユニット、および Sysmac Studio のバージョンにより、以下のような制約があります。

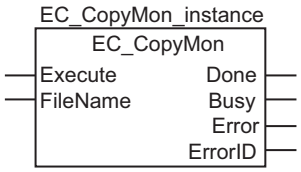
- 設定ユニットバージョン Ver.1.40 以降は、本命令を使用できません。
- 形 NX701 CPU ユニット、および形 NJ101 CPU ユニットは、Sysmac Studio Ver.1.13 以降で使用可能です。
- 形 NX1P2 CPU ユニットは、Sysmac Studio Ver.1.17 以降で使用可能です。
- 形 NJ301 CPU ユニットは、CPU ユニット Ver.1.10 以降、かつ Sysmac Studio Ver.1.12 以降で使用可能です。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.1-259)」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、CPU ユニット、および Sysmac Studio のバージョンにより、以下のような制約があります。

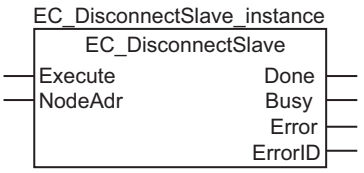
- 設定ユニットバージョン Ver.1.40 以降は、本命令を使用できません。
- 形 NX701 CPU ユニット、および形 NJ101 CPU ユニットは、Sysmac Studio Ver.1.13 以降で使用可能です。
- 形 NX1P2 CPU ユニットは、Sysmac Studio Ver.1.17 以降で使用可能です。
- 形 NJ301 CPU ユニットは、CPU ユニット Ver.1.10 以降、かつ Sysmac Studio Ver.1.12 以降で使用可能です。

サンプルプログラム

EC_StartMon 命令の、「サンプルプログラム (P.1-259)」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);

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.1-274)」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

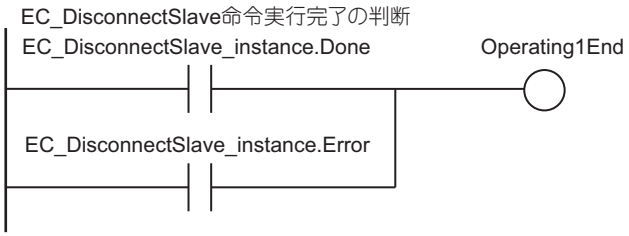
LD

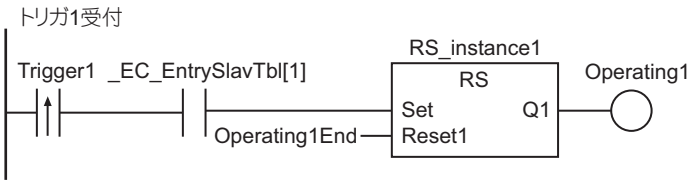
内部変数	名称	データ型	初期値	コメント
	Operating1End	BOOL	FALSE	処理終了 1
	Trigger1	BOOL	FALSE	実行条件 1
	Operating1	BOOL	FALSE	処理中 1
	RS_instance1	RS		
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Operating2End	BOOL	FALSE	処理終了 2
	Trigger2	BOOL	FALSE	実行条件 2
	Operating2	BOOL	FALSE	処理中 2
	RS_instance2	RS		
	EC_ConnectSlave_instance	EC_ConnectSlave		
	HMI_ConnectErrorID *1	WORD	16#0000	

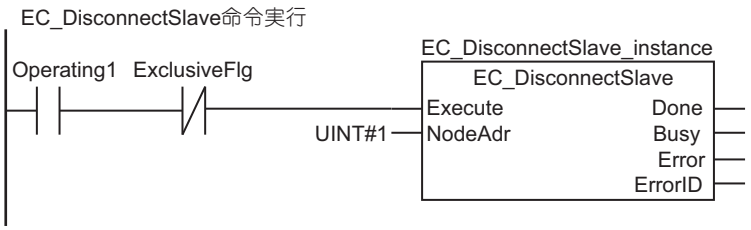
*1. 「HMI_」で始まる名称はタッチパネル表示用の変数です。

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 *1	☑	ネットワーク加入スレーブテーブル
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL *1	☑	離脱指令中スレーブテーブル
	ExclusiveFlg	BOOL	☐	命令排他フラグ

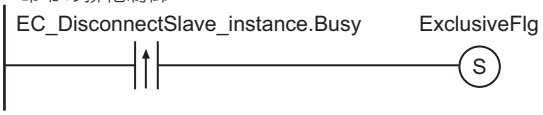
*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

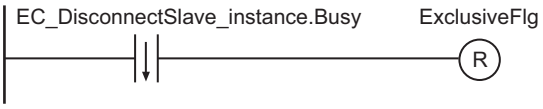




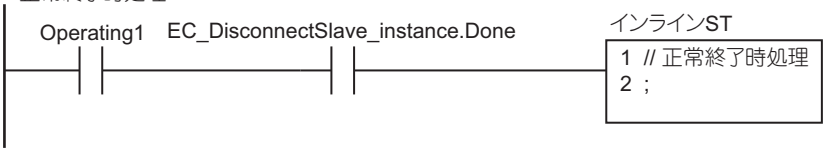


命令の排他制御

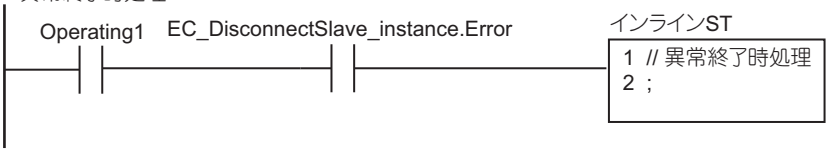




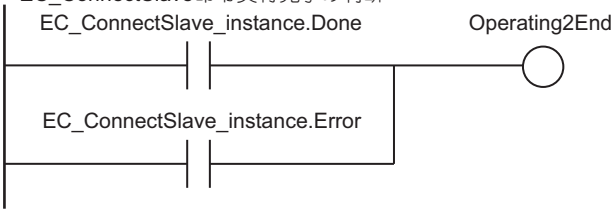
正常終了時処理



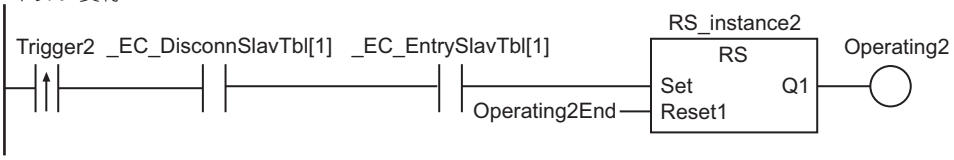
異常終了時処理



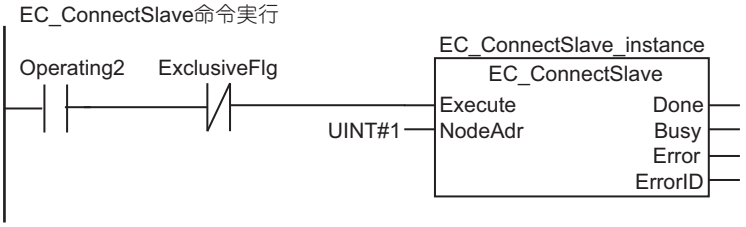
EC_ConnectSlave命令実行完了の判断



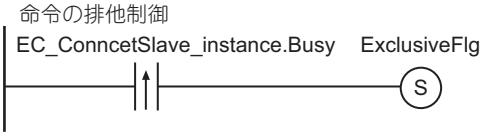
トリガ2受付

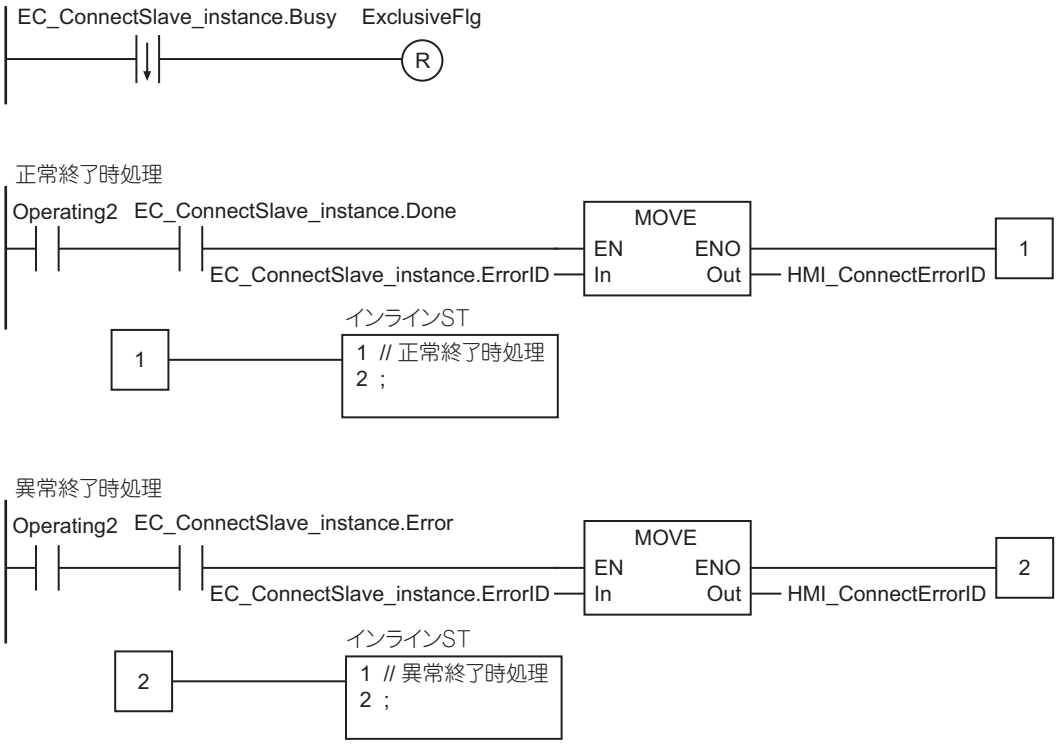


EC_ConnectSlave命令実行



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
	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
	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		

内部変数	名称	データ型	初期値	コメント
	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 *1	☑	ネットワーク加入スレーブテーブル
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL *1	☑	離脱指令中スレーブテーブル
	ExclusiveFlg	BOOL		命令排他フラグ

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

```
// 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;

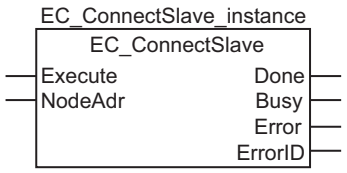
// EC_ConnectSlave 命令実行
IF (Operating2=TRUE) THEN
  EC_ConnectSlave_instance(
    Execute:=NOT(ExclusiveFlg),
  NodeAdr:=UINT#1);

  // 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);

サンプルプログラム

EC_DisconnectSlave 命令の「サンプルプログラム (P.1-267)」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 EC_ChangeEnableSetting Execute / NodeAdr / IsEnable Done / Busy / Error / ErrorID	EC_ChangeEnableSetting_instance(Execute, NodeAdr, IsEnable, Done, Busy, Error, ErrorID);

バージョン関連情報

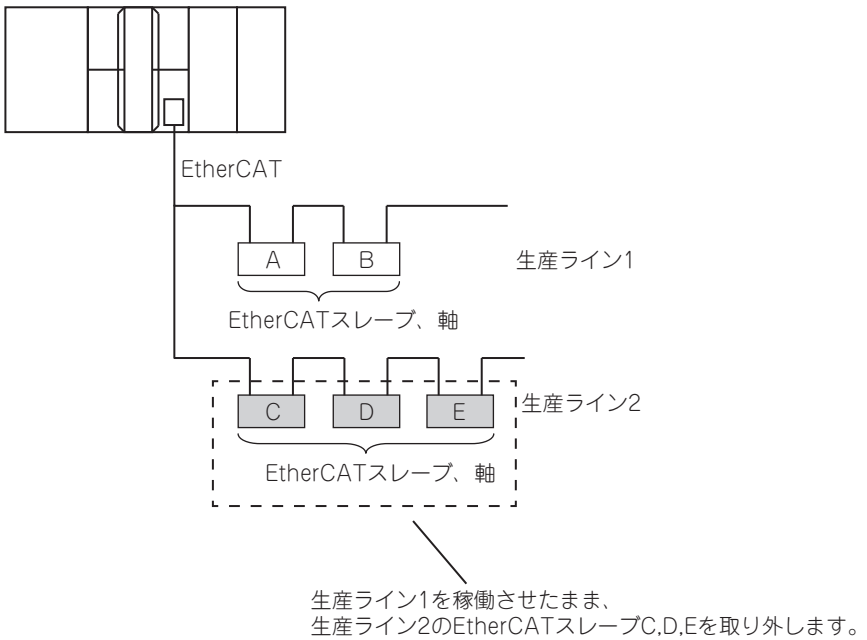
本命令は、CPU ユニット Ver.1.04 以降、かつ Sysmac Studio Ver.1.05 以降で、使用可能です。

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。
- 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、『』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.1-267)」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

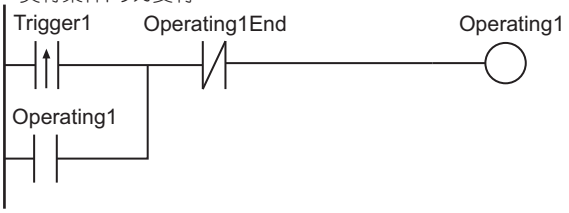
● LD

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
	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		

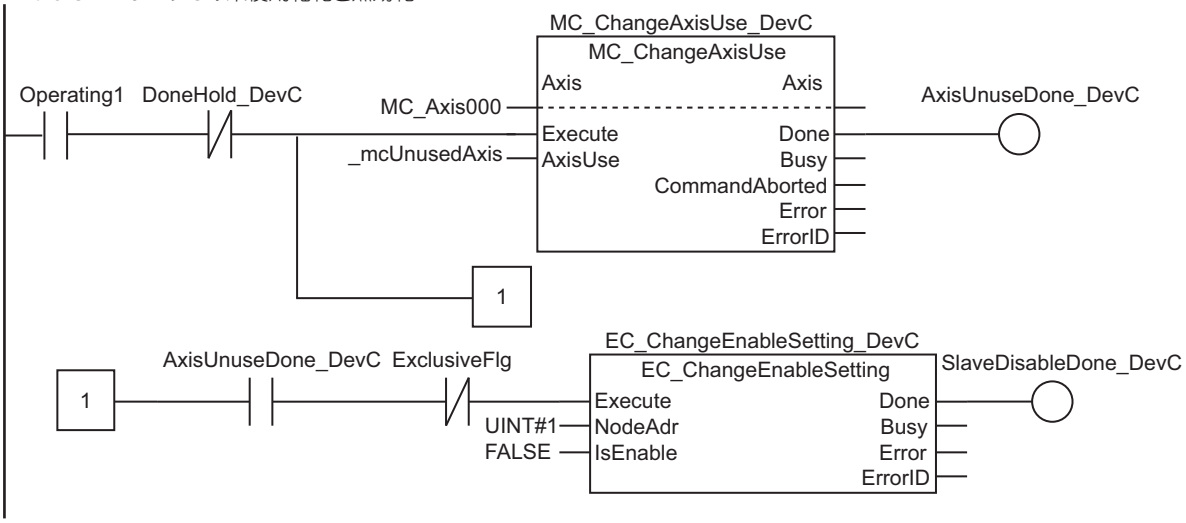
外部変数	名称	データ型	コンスタント	コメント
	MC_Axis000	_sAXIS_REF	☑	EtherCAT スレーブ C の軸変数
	MC_Axis001	_sAXIS_REF	☑	EtherCAT スレーブ D の軸変数

外部変数	名称	データ型	コンスタント	コメント
	MC_Axis002	_sAXIS_REF	☑	EtherCAT スレーブ E の軸変数
	ExclusiveFlg	BOOL	☐	命令排他フラグ

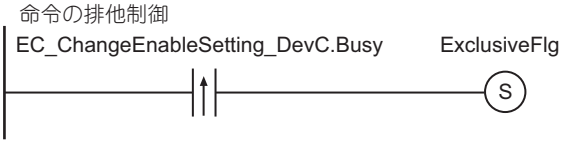
実行条件トリガ受付

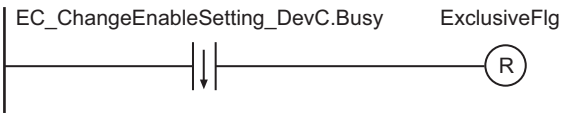


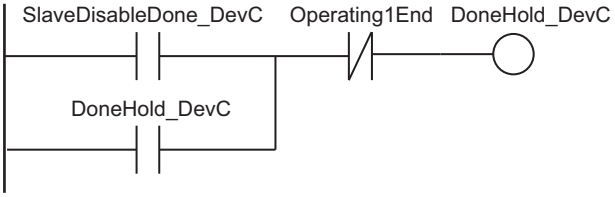
EtherCATスレーブCの未使用軸化と無効化



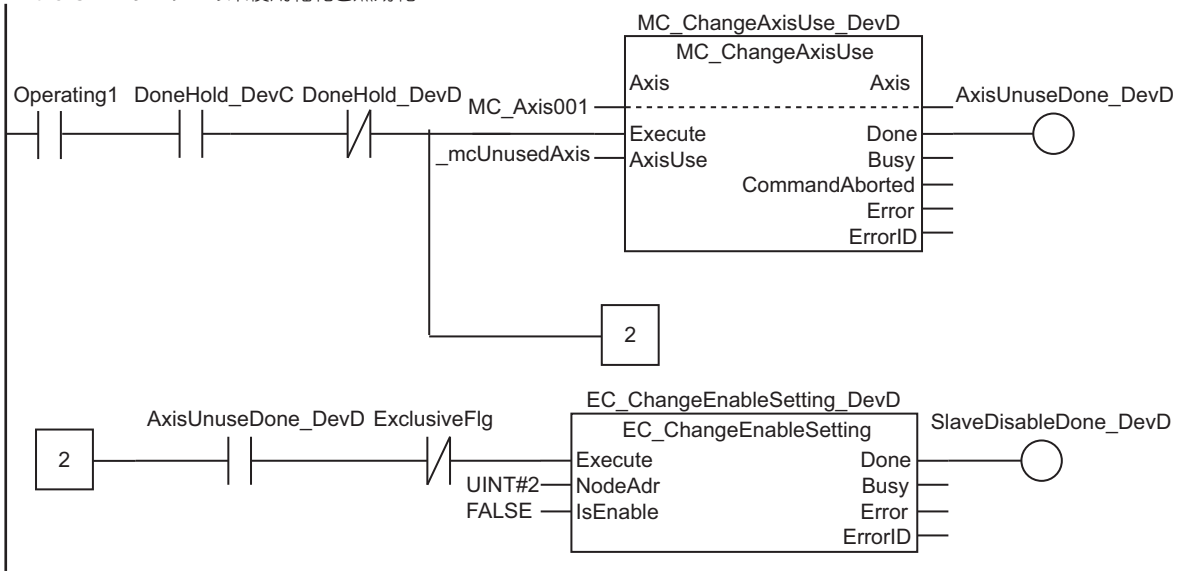
命令の排他制御



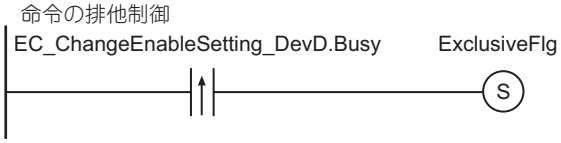


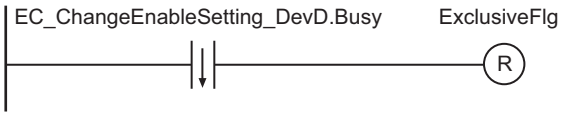


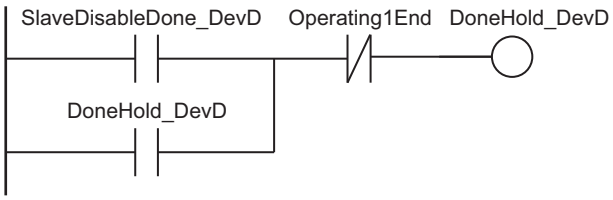
EtherCATスレーブDの未使用軸化と無効化



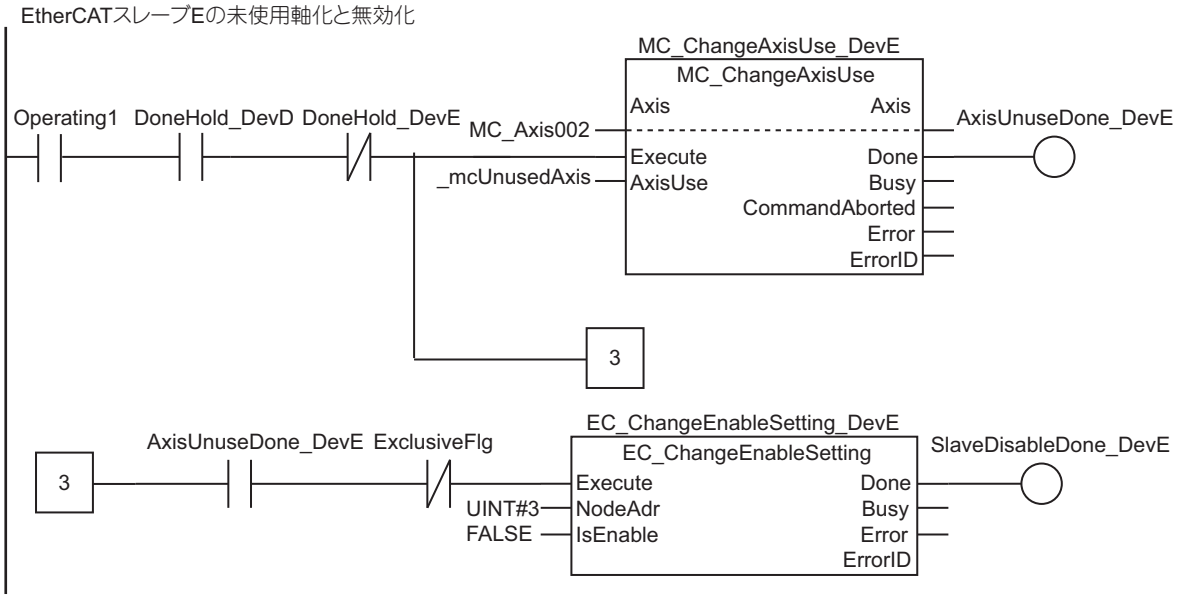
命令の排他制御

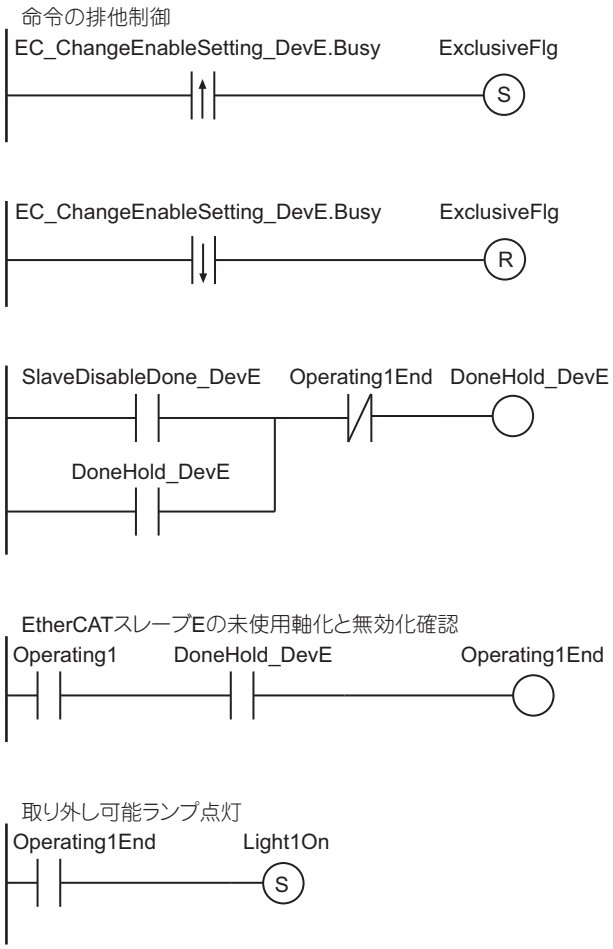






EtherCATスレーブEの未使用軸化と無効化





● 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
	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		

内部変数	名称	データ型	初期値	コメント
	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,
```

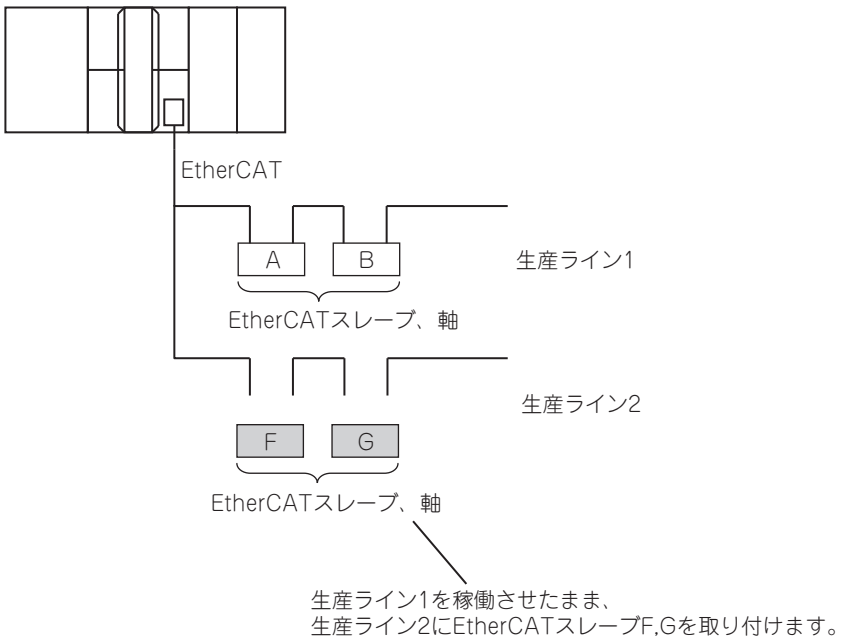
```
  Q1 =>ExclusiveFlg);
RS_DevE(
  Set :=EC_ChangeEnableSetting_DevE.Done,
  Reset1:=Operating1End,
  Q1 =>DoneHold_DevE);

// 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、『』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.1-267)」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

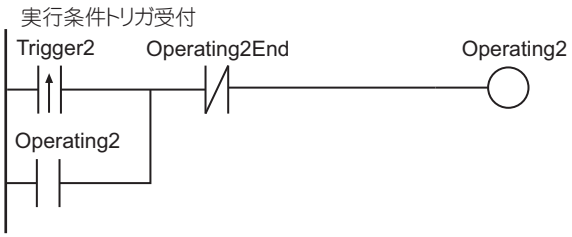
名称	データ型	初期値	割付先	コンスタント	コメント
MC_Axis003	_sAXIS_REF		MC://_MC_AX[3]	☑	EtherCAT スレーブ F の軸変数
MC_Axis004	_sAXIS_REF		MC://_MC_AX[4]	☑	EtherCAT スレーブ G の軸変数
ExclusiveFlg	BOOL	FALSE		☐	命令排他フラグ

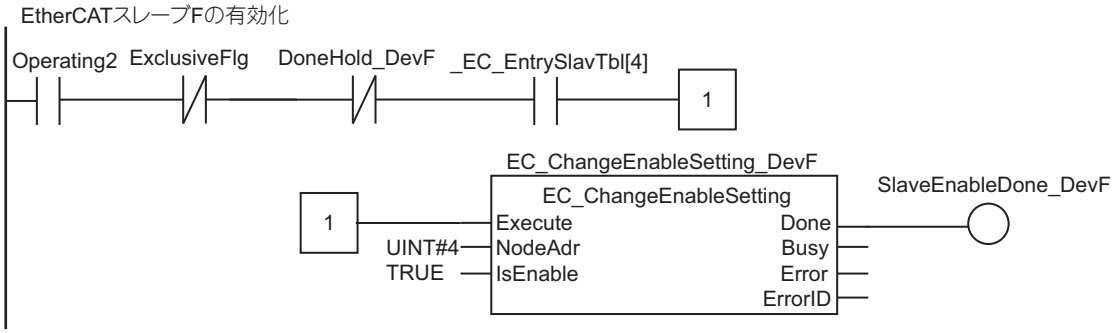
● LD

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

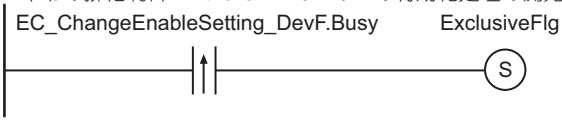
内部変数	名称	データ型	初期値	コメント
	Operating2	BOOL	FALSE	処理中
	AxisUseDone_DevF	BOOL	FALSE	EtherCAT スレーブ F の軸使用の使用軸化完了
	SlaveEnableDone_DevF	BOOL	FALSE	EtherCAT スレーブ F の有効化完了
	DoneHold_DevF	BOOL	FALSE	EtherCAT スレーブ F の処理完了の保持
	AxisUseDone_DevG	BOOL	FALSE	EtherCAT スレーブ G の軸使用の使用軸化完了
	SlaveEnableDone_DevG	BOOL	FALSE	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



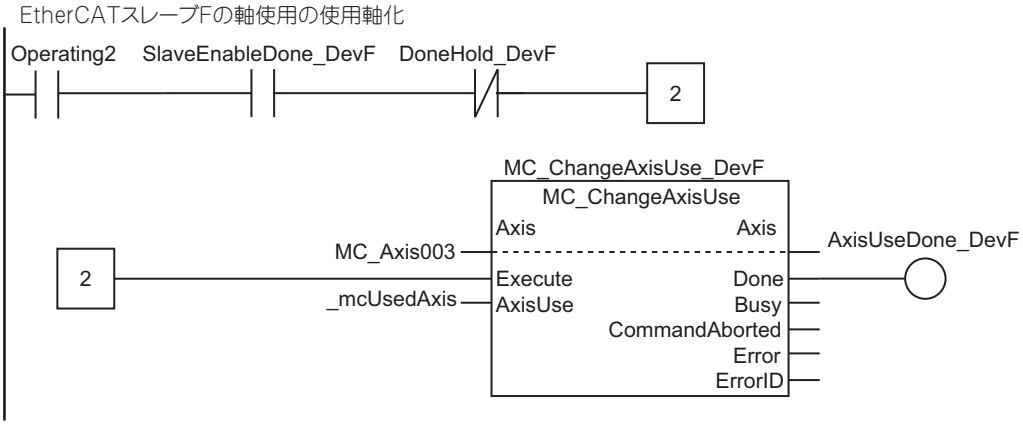


命令の排他制御　EtherCATスレーブFの有効化処理の開始と完了の確認

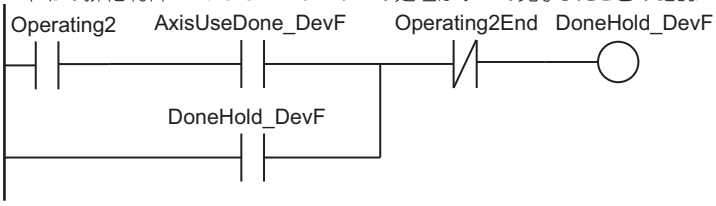




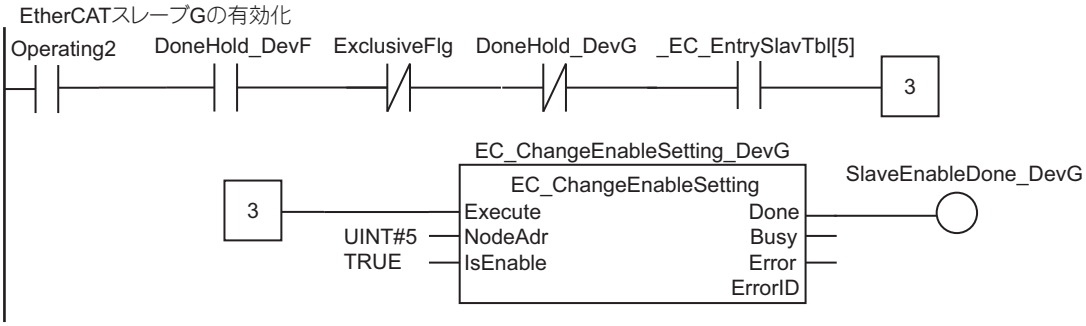
EtherCATスレーブFの軸使用の使用軸化



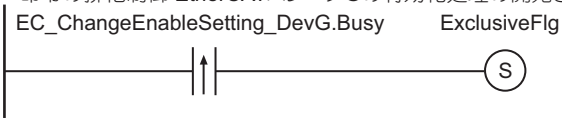
命令の排他制御　EtherCATスレーブFの処理がすべて完了したことの確認

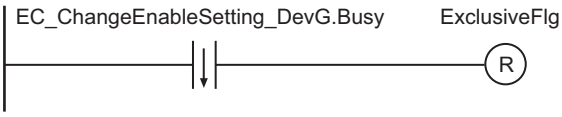


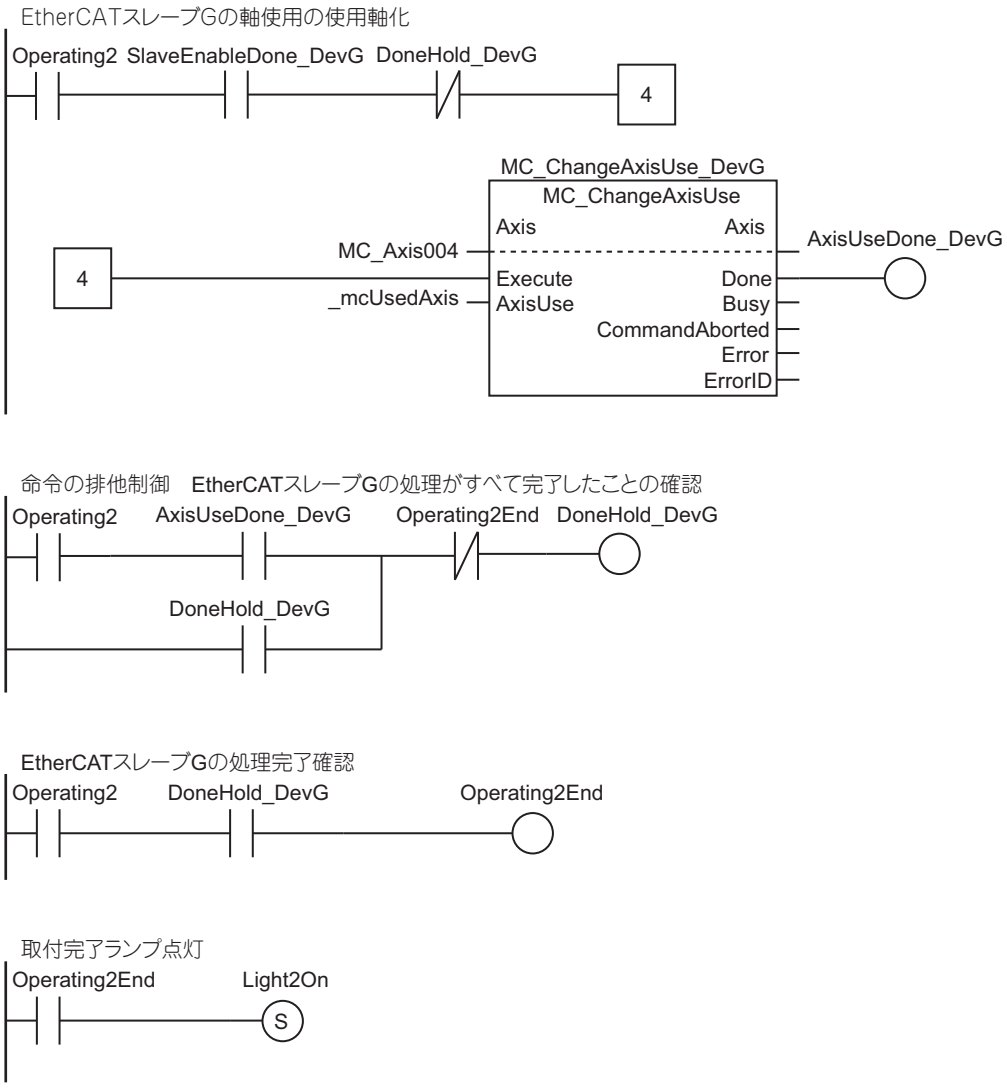
EtherCATスレーブGの有効化



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







● 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		

内部変数	名称	データ型	初期値	コメント
	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);

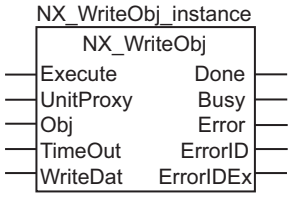
// 命令の排他制御 EtherCAT スレーブ G の処理がすべて完了したことの確認
RS_DevG(
  Set :=(Operating2 & MC_ChangeAxisUse_DevG.Done),
  Reset1:=Operating2End,
  Q1 =>DoneHold_DevG);

// 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);
```

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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。

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。**

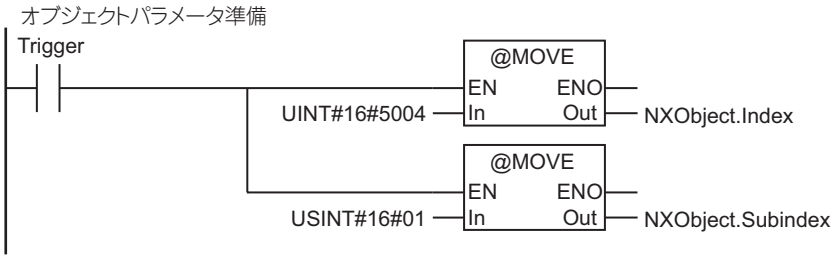
4 **RestartNXUnit 命令を使って、ユニットをリスタートさせます。**

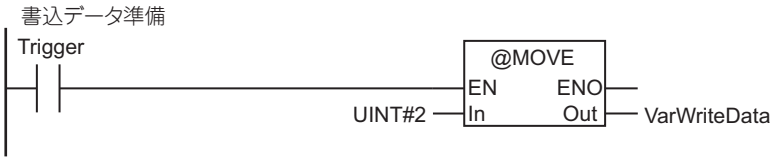
● LD

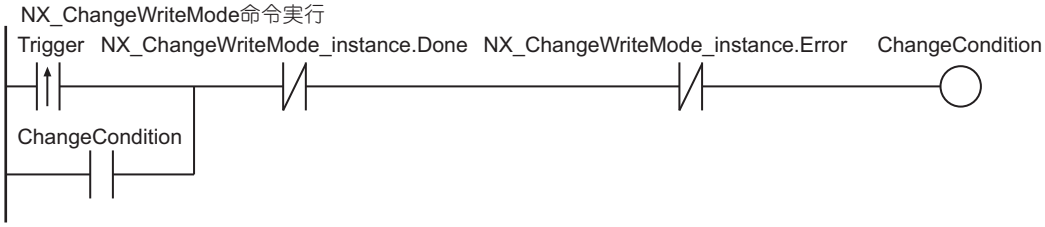
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ChangeCondition	BOOL	FALSE	書込モード変更実行条件
	WriteCondition	BOOL	FALSE	データ書込実行条件
	SaveCondition	BOOL	FALSE	データ保存実行条件
	RestartCondition	BOOL	FALSE	ユニットリスタート実行条件
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（DC 入力ユニット）
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）
	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		

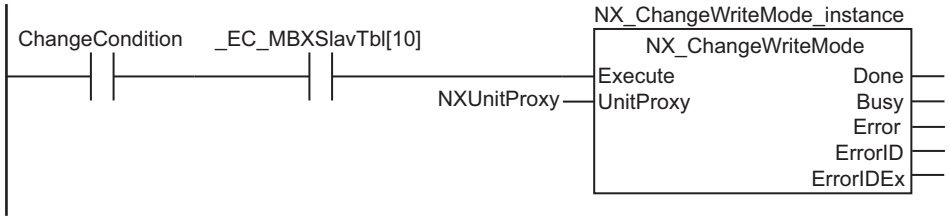
外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL *1	メッセージ通信可能スレーブテーブル

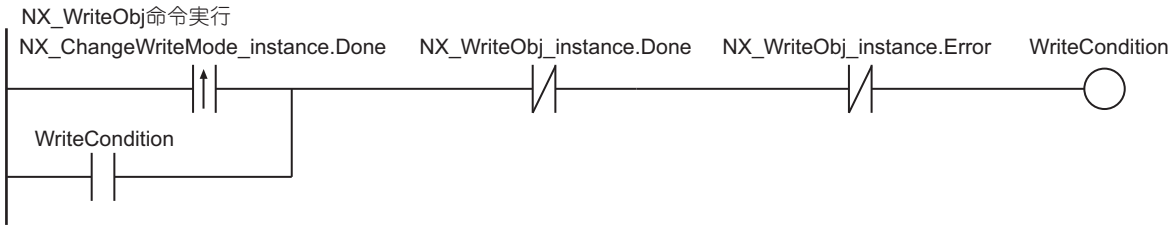
*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

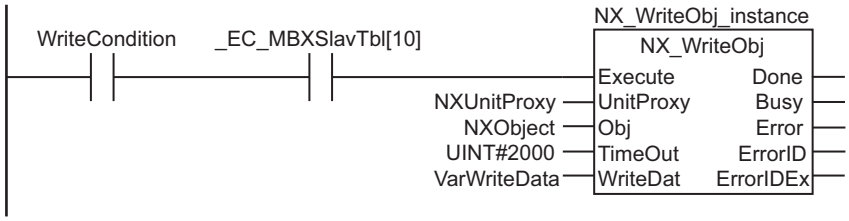


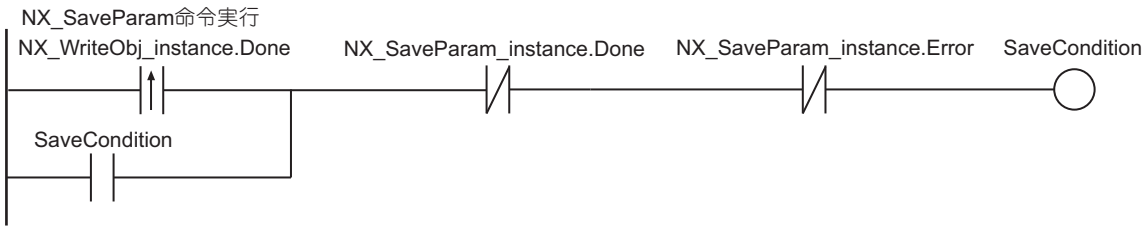


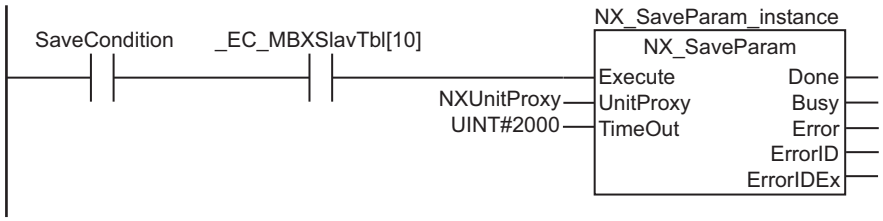


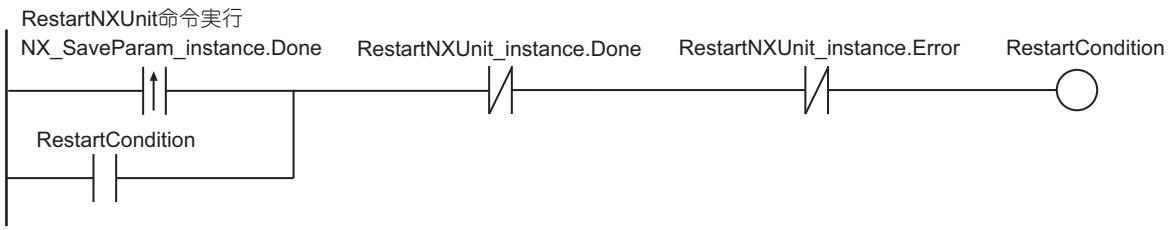

ChangeCondition
_EC_MBXSlavTbl[10]
NXUnitProxy
NX_ChangeWriteMode_instance
NX_ChangeWriteMode
Execute
UnitProxy
Done
Busy
Error
ErrorID
ErrorIDEx

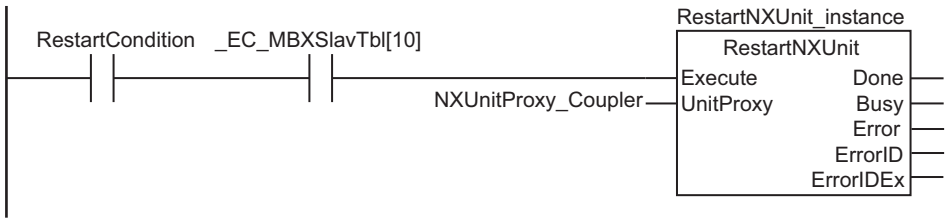

NX_WriteObj命令実行
NX_ChangeWriteMode_instance.Done
NX_WriteObj_instance.Done
NX_WriteObj_instance.Error
WriteCondition
WriteCondition

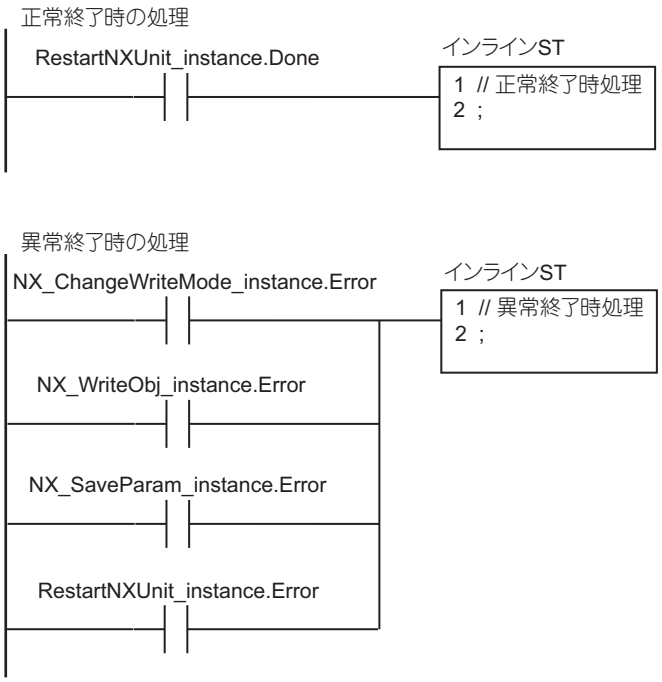

WriteCondition
_EC_MBXSlavTbl[10]
NXUnitProxy
NXObject
UINT#2000
VarWriteData
NX_WriteObj_instance
NX_WriteObj
Execute
UnitProxy
Obj
TimeOut
WriteDat
Done
Busy
Error
ErrorID
ErrorIDEx


NX_SaveParam命令実行
NX_WriteObj_instance.Done
NX_SaveParam_instance.Done
NX_SaveParam_instance.Error
SaveCondition
SaveCondition


SaveCondition
_EC_MBXSlavTbl[10]
NXUnitProxy
UINT#2000
NX_SaveParam_instance
NX_SaveParam
Execute
UnitProxy
TimeOut
Done
Busy
Error
ErrorID
ErrorIDEx


RestartNXUnit命令実行
NX_SaveParam_instance.Done
RestartNXUnit_instance.Done
RestartNXUnit_instance.Error
RestartCondition
RestartCondition


RestartCondition
_EC_MBXSlavTbl[10]
NXUnitProxy_Coupler
RestartNXUnit_instance
RestartNXUnit
Execute
UnitProxy
Done
Busy
Error
ErrorID
ErrorIDEx



● 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 *1	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

```
// 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。

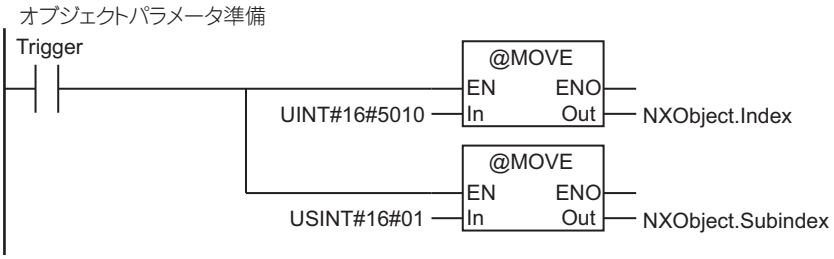
2 NX_SaveParam 命令を使って、書き込んだデータを保存します。

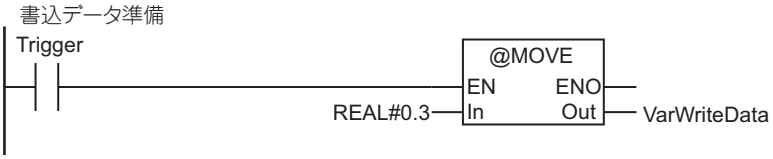
● LD

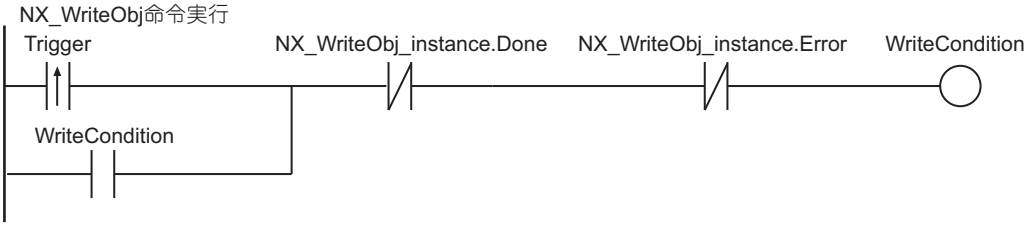
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteCondition	BOOL	FALSE	データ書込実行条件
	SaveCondition	BOOL	FALSE	データ保存実行条件
	NXUnitProxy	_sNXUNIT_ID		ユニット指定（AC 入力ユニット）
	NXUnitProxy_Coupler	_sNXUNIT_ID		ユニット指定（EtherCAT カプラユニット）
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	オブジェクトパラメータ
	VarWriteData	Real	0.0	書込データ
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		

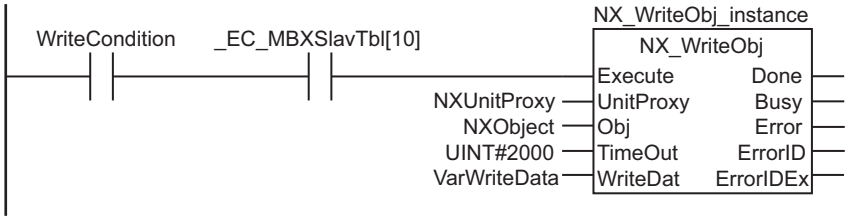
外部変数	名称	コンスタント	データ型	コメント
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL *1	メッセージ通信可能スレーブテーブル

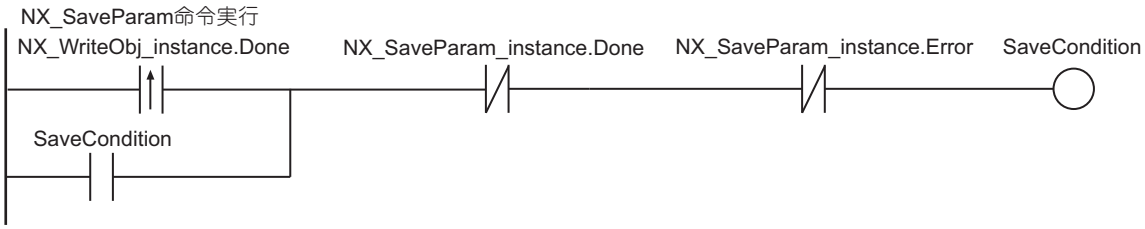
*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

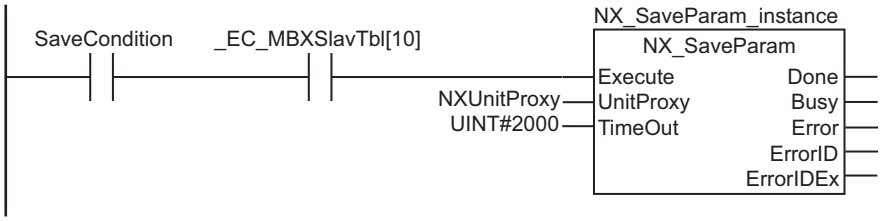


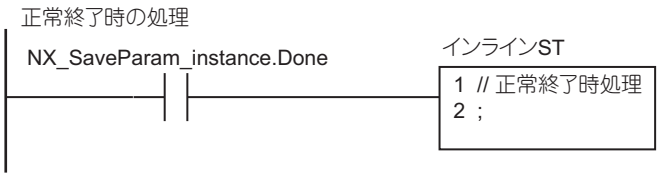


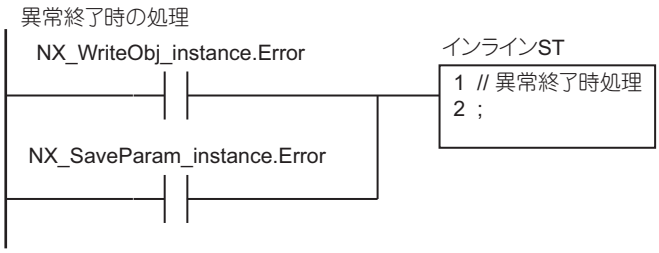












● ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	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）

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	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		
	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 *1	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

```
// 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;
```

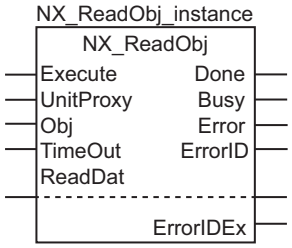
```
IF ((NX_SaveParam_instance.Done=TRUE) OR (NX_SaveParam_instance.Error=TRUE))THEN
  SaveCondition := FALSE;
END_IF;

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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.05 以降、かつ Sysmac Studio Ver.1.06 以降で、使用可能です。

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

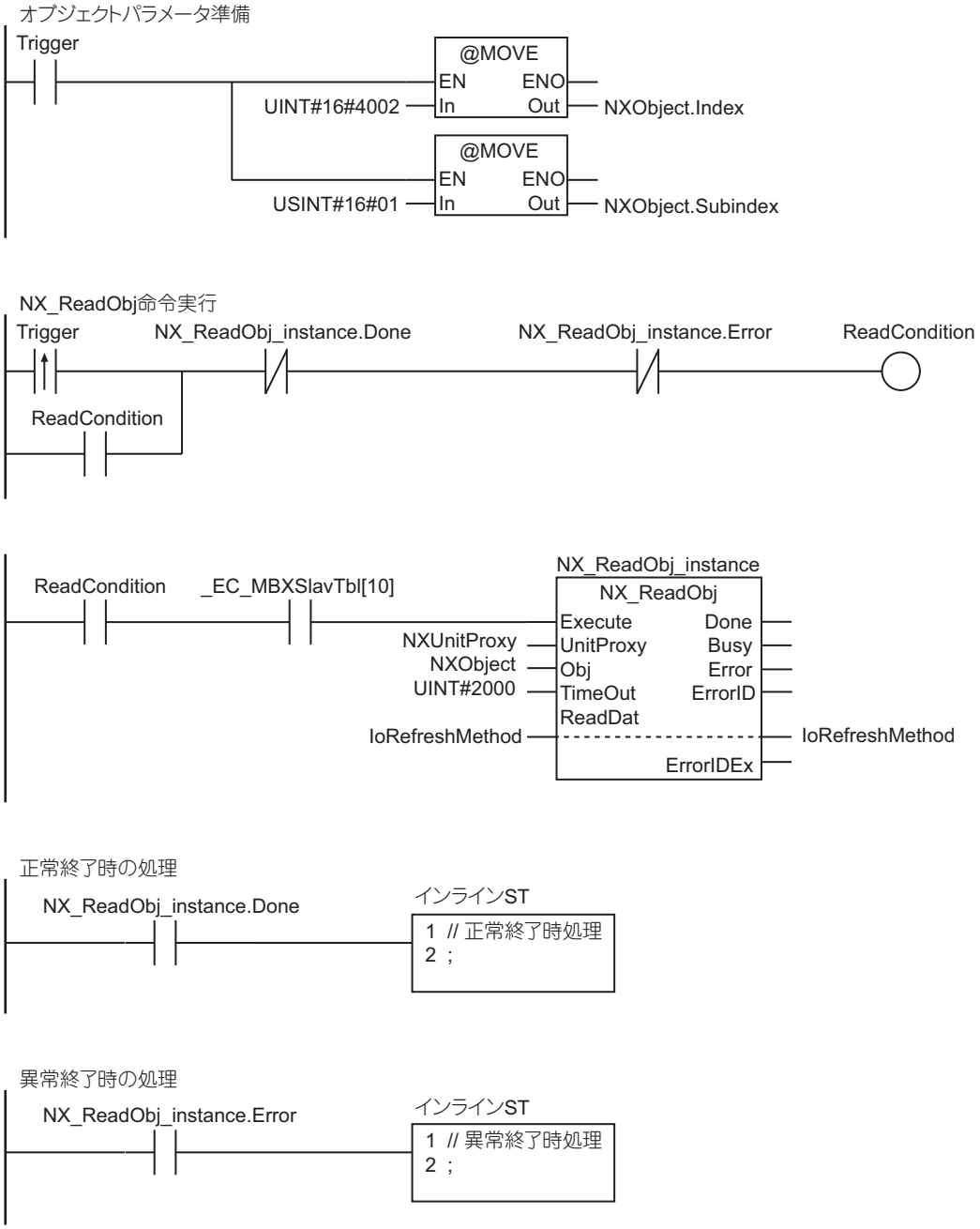
LD

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 *1	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。

形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。



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
	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
	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

内部変数	名称	データ型	初期値	コメント
	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 *1	メッセージ通信可能スレーブテーブル

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合は、「ARRAY [1..192] OF BOOL」です。

```
// 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.1-304
IOL_WriteObj	IO-Link デバイスオブジェクト書込	P.1-308

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

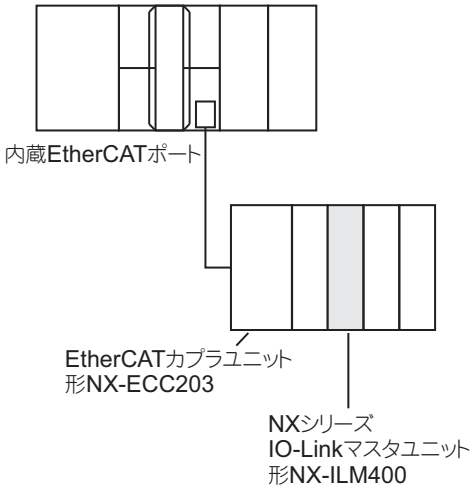
本命令は、EtherNet/IP タイプの IO-Link マスタユニットでは使用できません。

バージョン関連情報

本命令は、CPU ユニット Ver.1.12 以降、かつ Sysmac Studio Ver.1.16 以降で、使用可能です。

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。



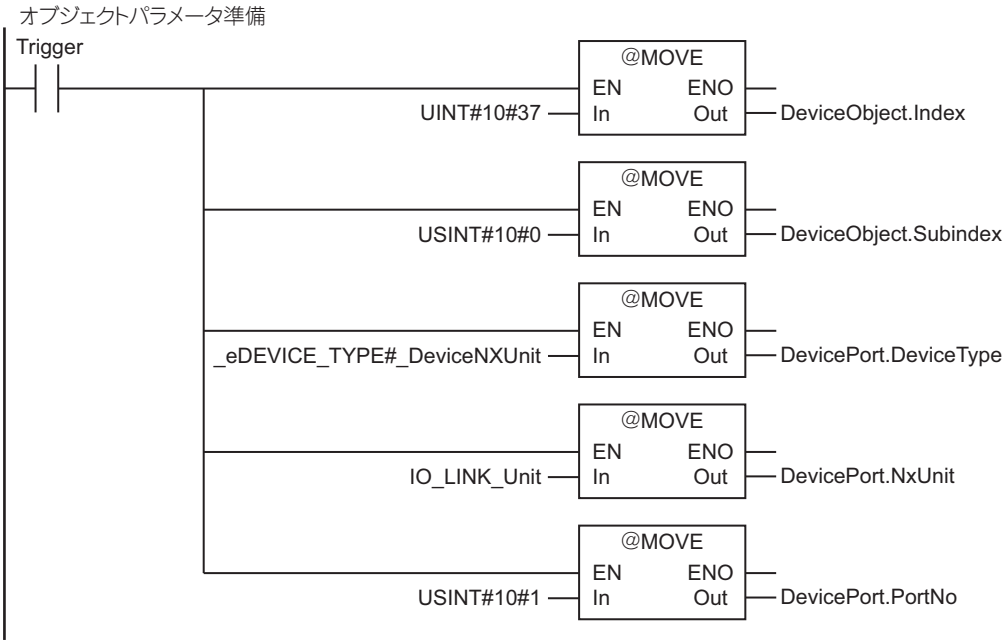
形 NX-ILM400 のポート 1 に接続の光電センサ（形 E3Z）のエラーログ(Index:37/Subindex:0)30byte を読み出します。読み出しデータは、「DeviceErrorLog」に格納します。
形 NX-ECC203 のノードアドレスは 10 とします。

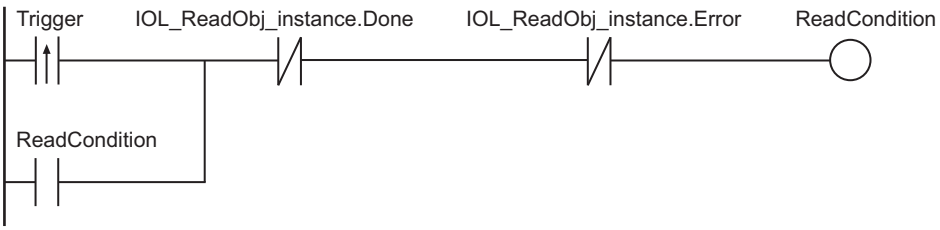
LD

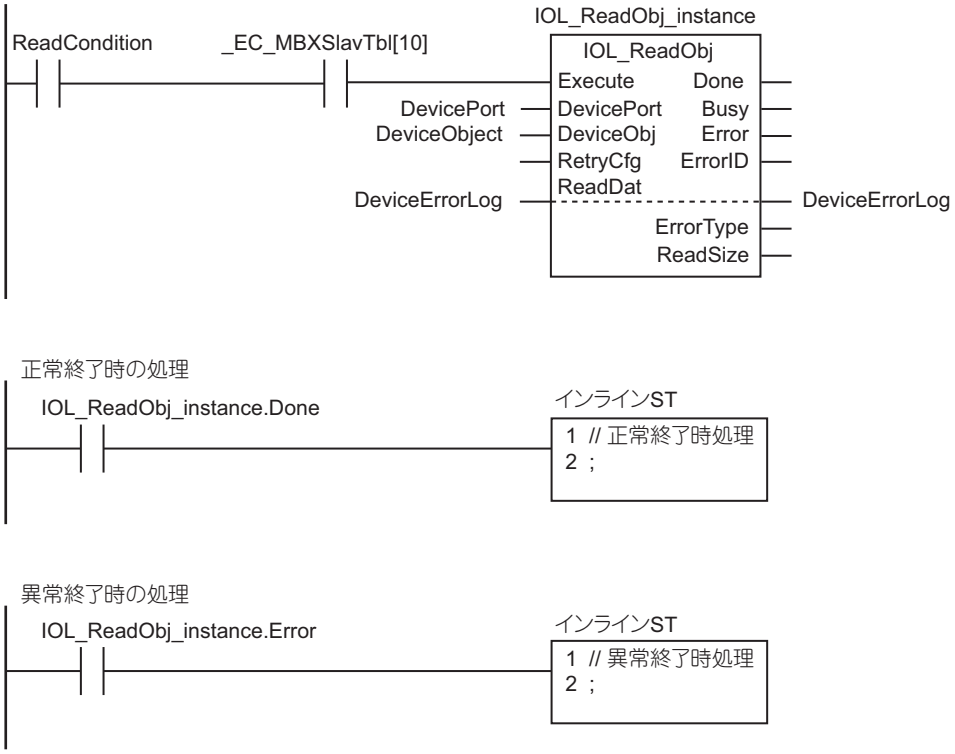
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	ReadCondition	BOOL	FALSE	データ読み出し実行条件
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	DeviceErrorLog	ARRAY[1..30] OF BYTE		読み出しデータ
	IOL_ReadObj_instance	IOL_ReadObj		

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 *1	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	

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合、「ARRAY [1..192] OF BOOL」となります。







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
	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 *1	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	

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合、「ARRAY [1..192] OF BOOL」となります。

```
// 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

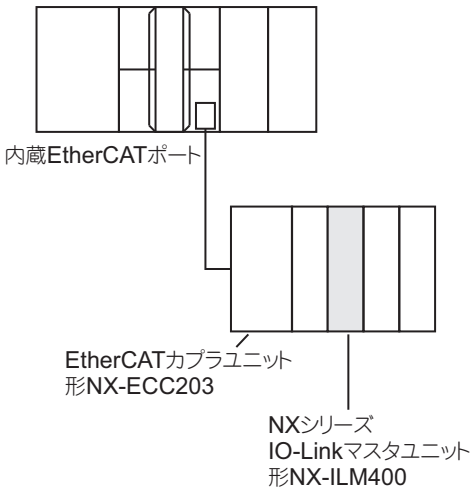
本命令は、EtherNet/IP タイプの IO-Link マスタユニットでは使用できません。

バージョン関連情報

本命令は、CPU ユニット Ver.1.12 以降、かつ Sysmac Studio Ver.1.16 以降で、使用可能です。

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。



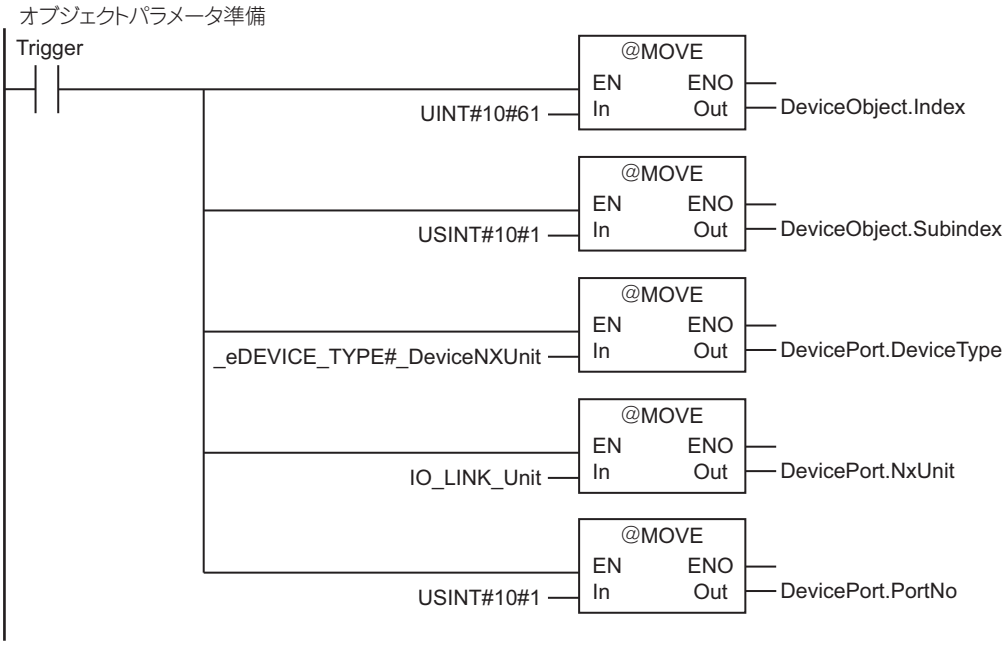
形 NX-ILM400 のポート 1 に接続している光電センサ（形 E3Z）の制御出力 1 動作モード切替 (SwitchPoint Logic Output1)(Index:61/Subindex:1)1byte に「01」を書き込みます。書き込みデータは、「SwitchPoint」に格納します。
形 NX-ECC203 のノードアドレスは 10 とします。

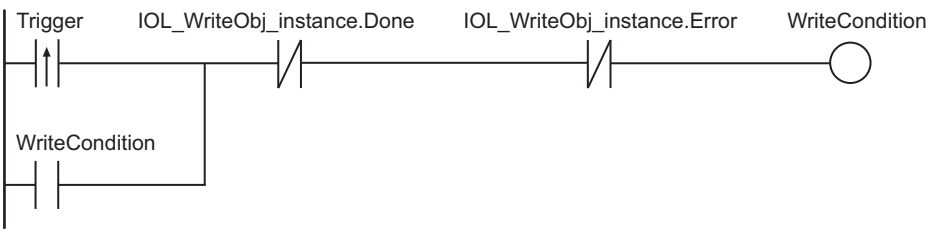
LD

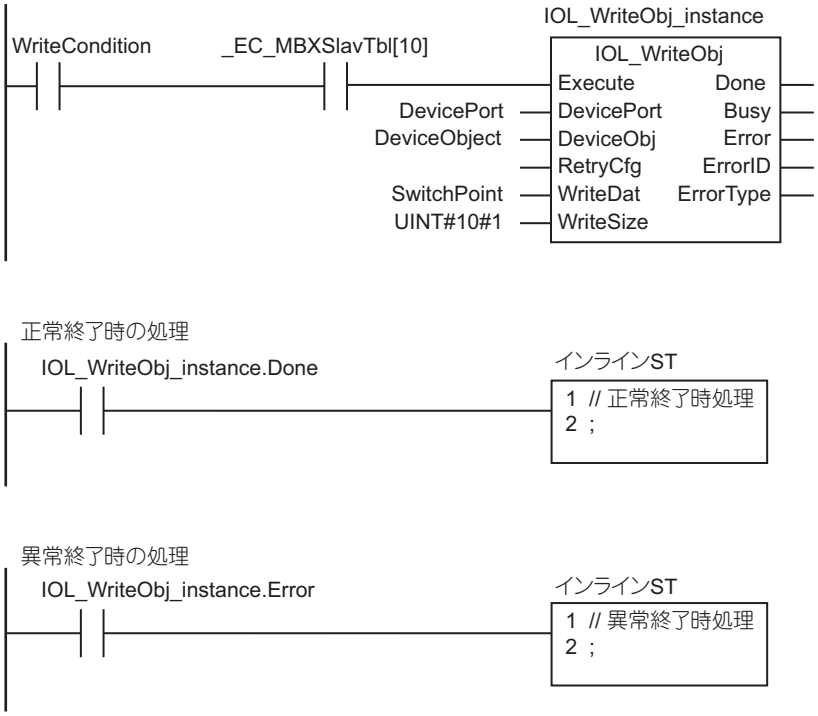
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	実行条件
	WriteCondition	BOOL	FALSE	データ書き込み実行条件
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	IO-Link デバイスのオブジェクト指定
	SwitchPoint	USINT	USINT#01	書き込みデータ
	IOL_WriteObj_instance	IOL_WriteObj		

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 *1	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	

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合、「ARRAY [1..192] OF BOOL」となります。







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
	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 *1	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	

*1. 形 NX502 CPU ユニットの場合は、「ARRAY [1..256] OF BOOL」です。
形 NX102 CPU ユニット、形 NX1P2 CPU ユニットおよび NJ シリーズ CPU ユニットの場合、「ARRAY [1..192] OF BOOL」となります。

```
// 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;
```

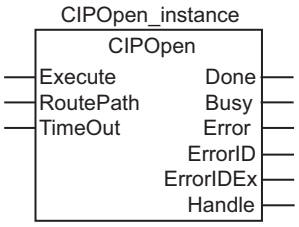
EtherNet/IP 通信命令

命令	名称	ページ
CIPOpen	CIPClass3(Large_Forward_Open)コネクション確立	P.1-315
CIPRead	変数読出（Class3 Explicit メッセージ）	P.1-322
CIPWrite	変数書込（Class3 Explicit メッセージ）	P.1-323
CIPSend	任意 Explicit メッセージ送信(Class3)	P.1-324
CIPClose	CIP Class3 コネクションの切断	P.1-325
CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	P.1-326
CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	P.1-327
CIPUCMMSend	任意 Explicit メッセージ送信(UCMM)	P.1-328
SktUDPCreate	UDP ソケットクリエイト	P.1-334
SktUDPRcv	UDP ソケット受信	P.1-340
SktUDPSend	UDP ソケット送信	P.1-341
SktTCPAccept	TCP ソケットアクセプト	P.1-342
SktTCPConnect	TCP ソケットコネクト	P.1-343
SktTCPRcv	TCP ソケット受信	P.1-349
SktTCPSend	TCP ソケット送信	P.1-350
SktGetTCPStatus	TCP ソケットのステータス読出	P.1-351
SktClose	TCP/UDP ソケットクローズ	P.1-352
SktClearBuf	TCP/UDP ソケット受信バッファクリア	P.1-353
SktSetOption	TCP ソケットオプション設定	P.1-354
SktTLSConnect	TLS セッション確立	P.1-357
SktTLSRead	TLS 受信	P.1-364
SktTLSWrite	TLS 送信	P.1-365
SktTLSDisconnect	TLS セッション切断	P.1-366

命令	名称	ページ
SktTLSClearBuf	TLS セッション受信バッファクリア	P.1-367
SktTLSStopLog	セキュアソケット通信ログ停止	P.1-368
ModbusTCPCmd	Modbus TCP 汎用コマンド送信	P.1-370
ModbusTCPRead	Modbus TCP Read コマンド送信	P.1-374
ModbusTCPWrite	Modbus TCP Write コマンド送信	P.1-378
ChangeIPAdr	IP アドレス変更	P.1-381
ChangeFTPAccount	FTP アカウント変更	P.1-385
ChangeNTPServerAdr	NTP サーバ宛先変更	P.1-387
FTPGetFileList	FTP サーバのファイルリスト取得	P.1-389
FTPGetFile	FTP サーバからのファイルダウンロード	P.1-395
FTPPutFile	FTP サーバへのファイルアップロード	P.1-396
FTPRemoveFile	FTP サーバのファイル削除	P.1-401
FTPRemoveDir	FTP サーバのディレクトリ削除	P.1-406

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);

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。**

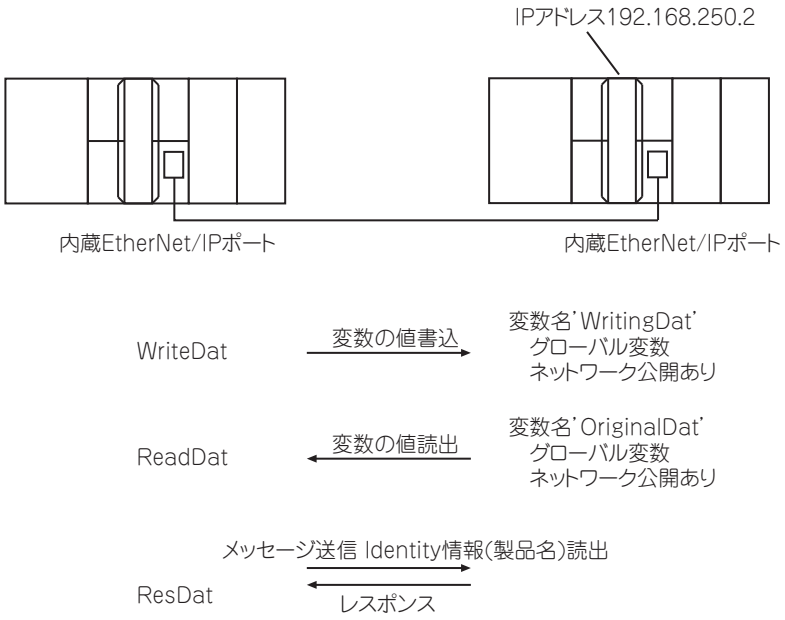
2 **CIPWrite 命令を使用して、相手ノードの変数に値を書き込みます。他局変数名は「WritingDat」、書き込む値は、変数「WriteDat」の中身とします。**
「WritingDat」は、相手ノードでグローバル変数として定義されており、かつ、ネットワーク公開されていることが必要です。

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。

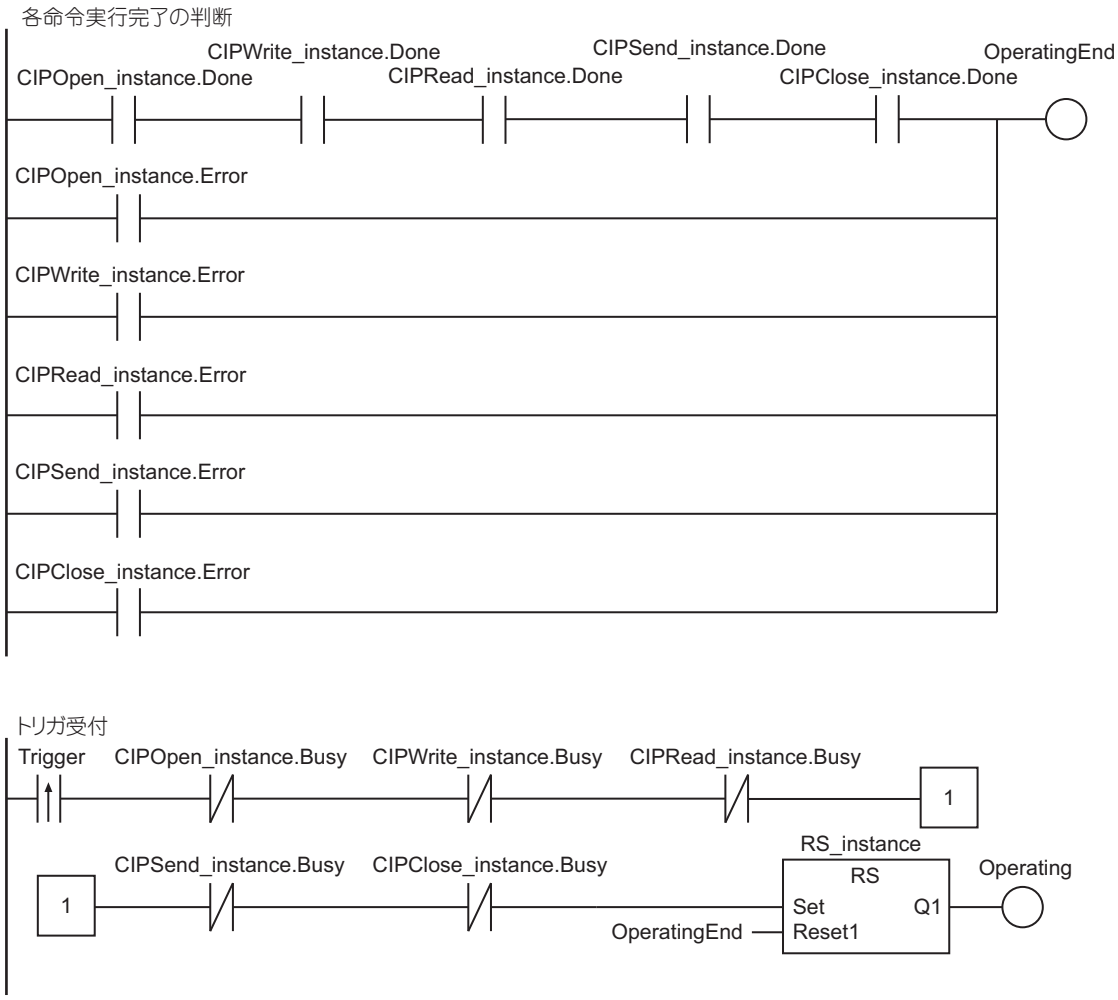
項目	値
クラス ID	1
インスタンス ID	1
アトリビュート ID	7
サービスコード	16#0E

5 **CIPClose 命令を使用して、Class3 コネクションを切断します。**

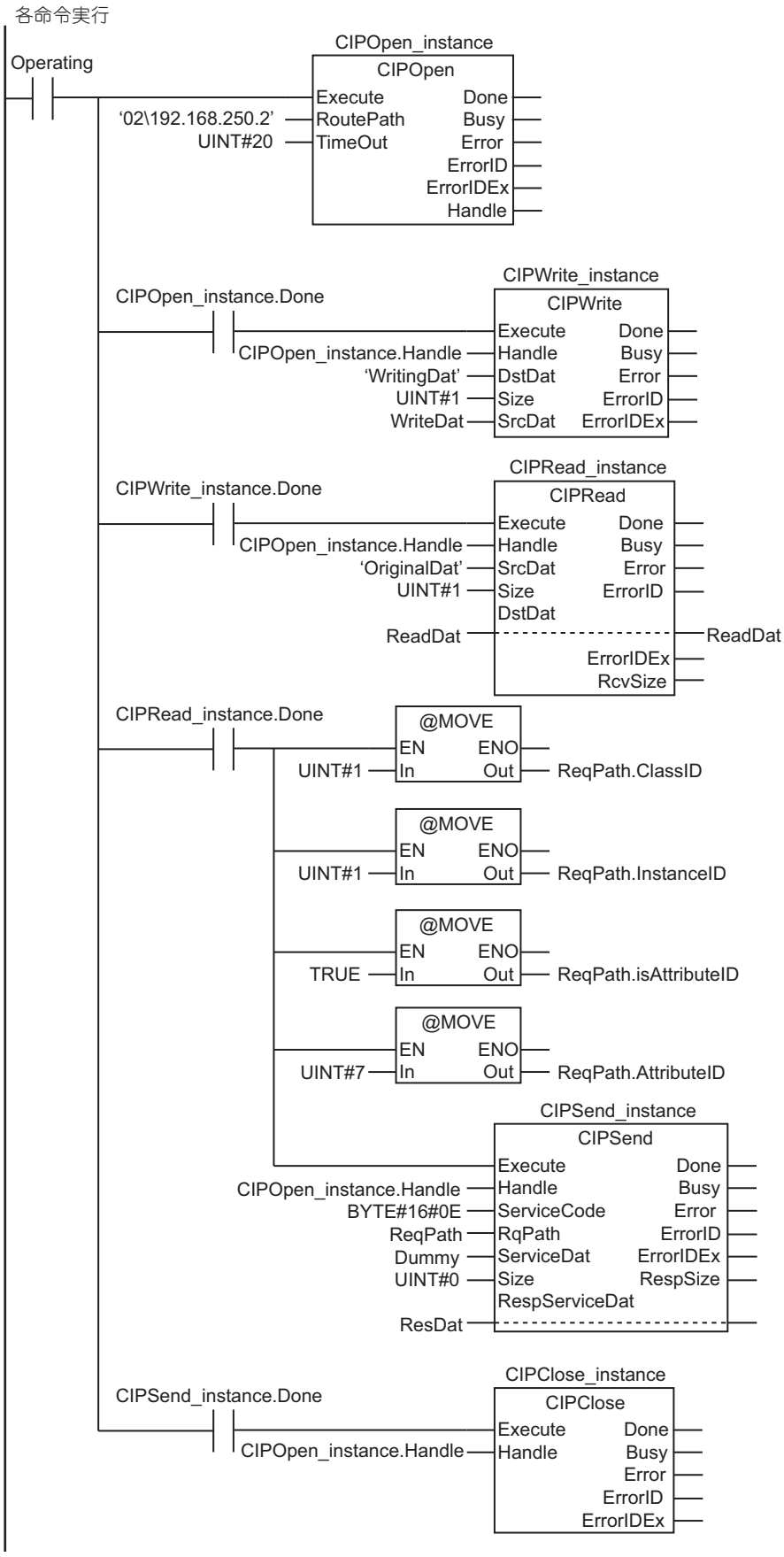


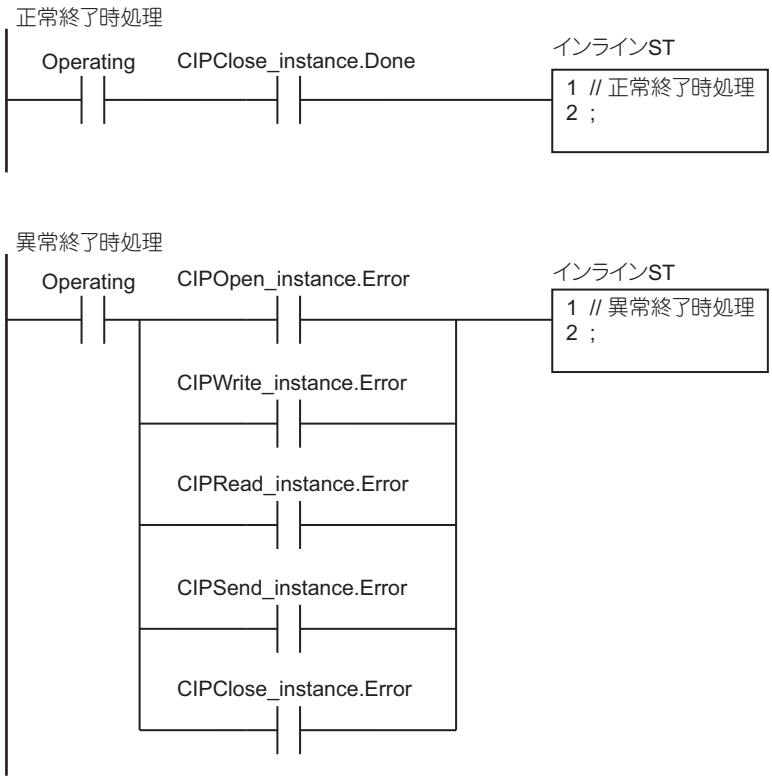
LD

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		
CIPSend_instance	CIPSend		
CIPClose_instance	CIPClose		


各命令実行完了の判断
CIPOpen_instance.Done
CIPWrite_instance.Done
CIPRead_instance.Done
CIPSend_instance.Done
CIPClose_instance.Done
OperatingEnd
CIPOpen_instance.Error
CIPWrite_instance.Error
CIPRead_instance.Error
CIPSend_instance.Error
CIPClose_instance.Error

トリガ受付
Trigger
CIPOpen_instance.Busy
CIPWrite_instance.Busy
CIPRead_instance.Busy
1
CIPSend_instance.Busy
CIPClose_instance.Busy
1
RS_instance
RS
Set
Reset1
Q1
OperatingEnd
Operating


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

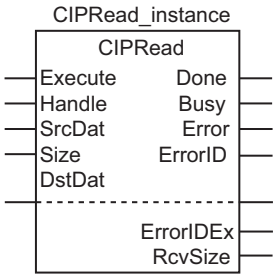
        IF (CIPClose_instance.Done=TRUE) THEN
            Stage:=INT#0;
        ELSIF (CIPClose_instance.Error=TRUE) THEN
            Stage:=INT#50;
        END_IF;

    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;
```

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）。

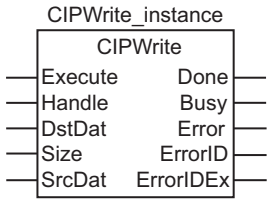
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPRead	変数読出（Class3 Explicit メッセージ）	FB		CIPRead_instance(Execute, Handle, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.1-315)」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）。

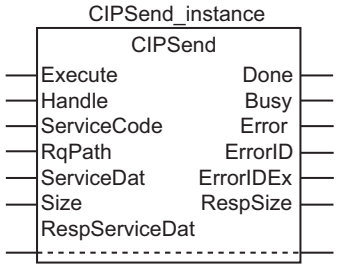
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPWrite	変数書込（Class3 Explicit メッセージ）	FB		CIPWrite_instance(Execute, Handle, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.1-315)」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。

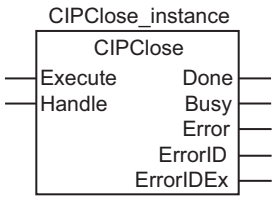
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPSend	任意 Explicit メッセージ送信（Class3）	FB		CIPSend_instance(Execute, Handle, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.1-315)」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。

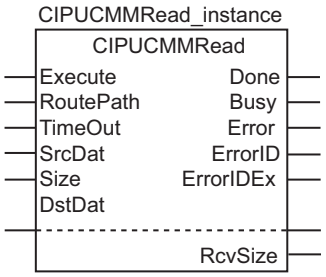
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPClose	CIP Class3 コネクションの切断	FB		CIPClose_instance(Execute, Handle, Done, Busy, Error, ErrorID, ErrorIDEx);

サンプルプログラム

CIPOpen 命令の「サンプルプログラム (P.1-315)」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）。

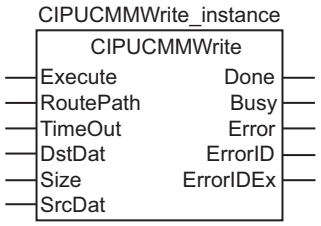
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPUCMMRead	変数読出（UCMM Explicit メッセージ）	FB		CIPUCMMRead_instance(Execute, RoutePath, TimeOut, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

サンプルプログラム

CIPUCMMSend 命令の「サンプルプログラム (P.1-328)」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）。

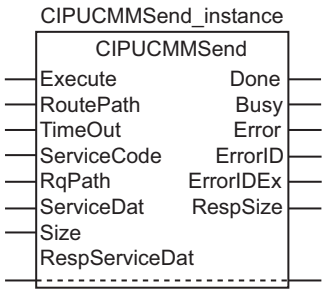
命令	名称	FB/FUN	グラフィック表現	ST 表現
CIPUCMMWrite	変数書込（UCMM Explicit メッセージ）	FB		CIPUCMMWrite_instance(Execute, RoutePath, TimeOut, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

サンプルプログラム

CIPUCMMSend 命令の「サンプルプログラム (P.1-328)」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);

サンプルプログラム

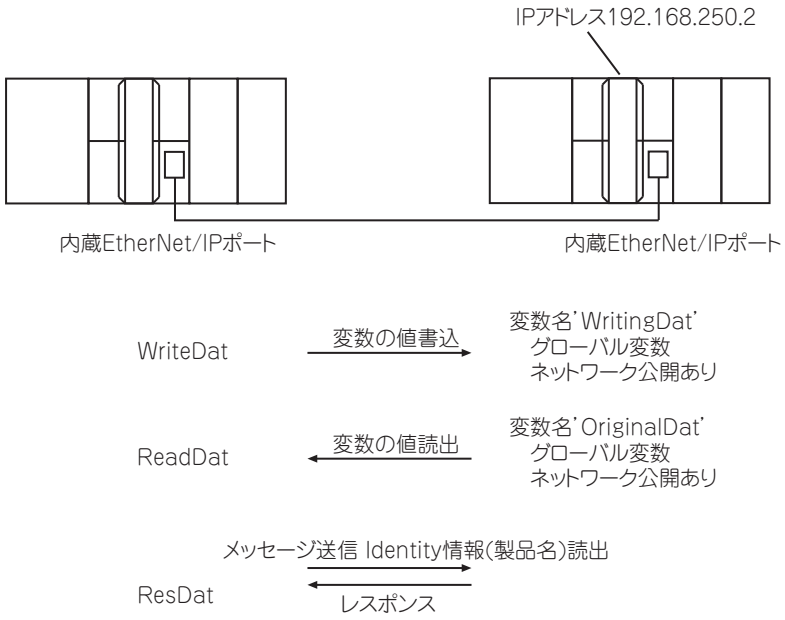
CIP の UCMM メッセージを使用して、変数書込、変数読出、メッセージ送信を行います。
コントローラどうしは、EtherNet/IP ネットワークで接続しており、相手ノードの IP アドレスは 192.168.250.2 とします。
処理の手順は、以下のようになります。

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。

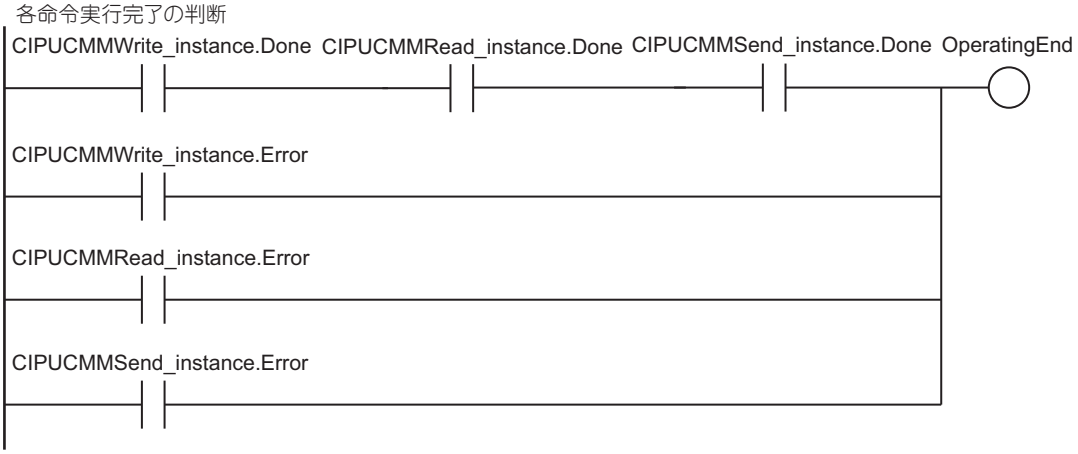
項目	値
クラス ID	1
インスタンス ID	1
アトリビュート ID	7
サービスコード	16#0E

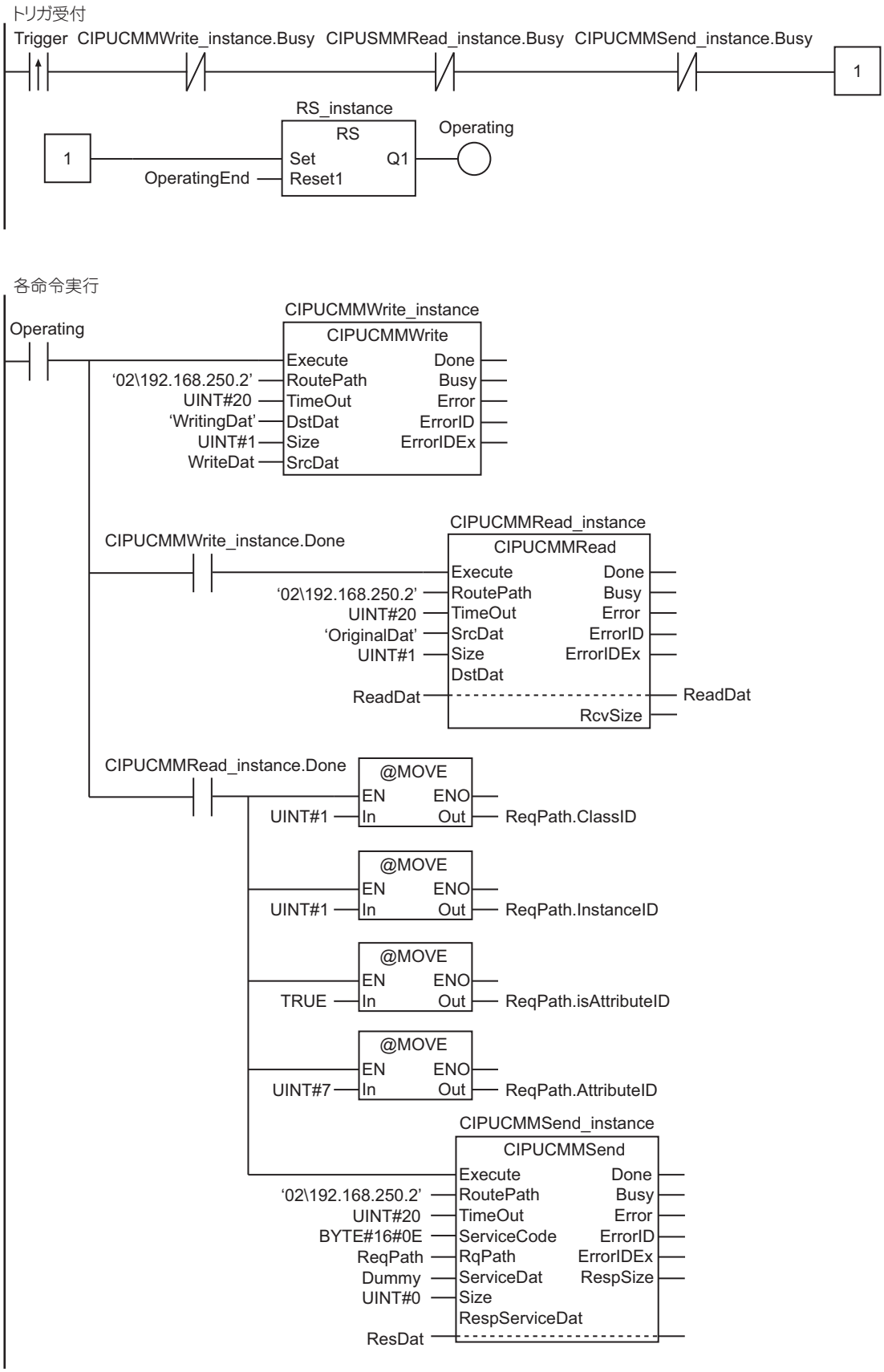


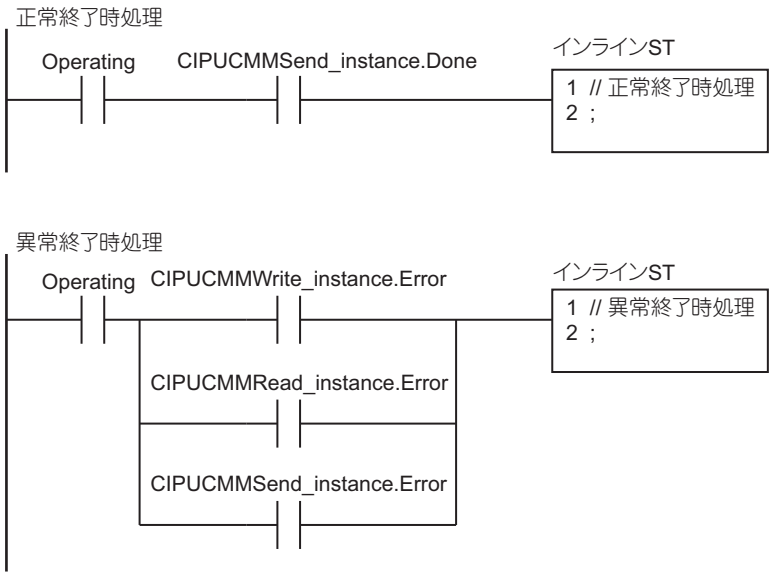
LD

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
Trigger
CIPUCMMWrite_instance.Busy
CIPUSMMRead_instance.Busy
CIPUCMMSend_instance.Busy
1
RS_instance
RS
Set
Reset1
Q1
Operating
1
OperatingEnd
各命令実行
Operating
CIPUCMMWrite_instance
CIPUCMMWrite
Execute
RoutePath
TimeOut
DstDat
Size
SrcDat
Done
Busy
Error
ErrorID
ErrorIDEx
'02\192.168.250.2'
UINT#20
'WritingDat'
UINT#1
WriteDat
CIPUCMMWrite_instance.Done
CIPUCMMRead_instance
CIPUCMMRead
Execute
RoutePath
TimeOut
SrcDat
Size
DstDat
ReadDat
Done
Busy
Error
ErrorID
ErrorIDEx
RcvSize
'02\192.168.250.2'
UINT#20
'OriginalDat'
UINT#1
ReadDat
CIPUCMMRead_instance.Done
@MOVE
EN
ENO
In
Out
UINT#1
ReqPath.ClassID
@MOVE
UINT#1
ReqPath.InstanceID
@MOVE
TRUE
ReqPath.isAttributeID
@MOVE
UINT#7
ReqPath.AttributeID
CIPUCMMSend_instance
CIPUCMMSend
Execute
RoutePath
TimeOut
ServiceCode
RqPath
ServiceDat
Size
RespServiceDat
Done
Busy
Error
ErrorID
ErrorIDEx
RespSize
'02\192.168.250.2'
UINT#20
BYTE#16#0E
ReqPath
Dummy
UINT#0
ResDat



ST

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;
```

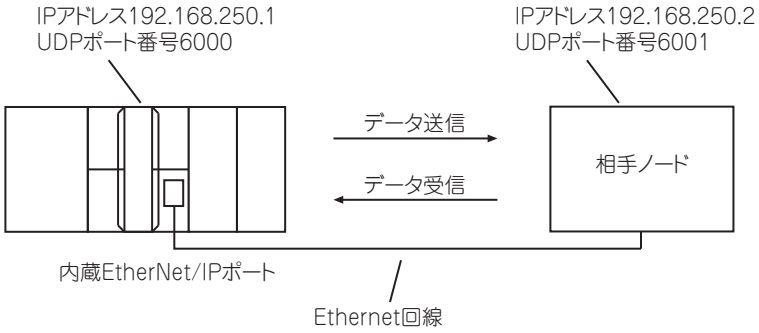
SktUDPCreate

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 SktUDPCreate Execute / SrcUdpPort Done / Busy / Error / ErrorID / Socket	SktUDPCreate_instance(Execute, SrcUdpPort, Done, Busy, Error, ErrorID, Socket);

サンプルプログラム

NJ/NX/NY シリーズ CPU ユニットと、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。



NJ/NX/NY シリーズ CPU ユニット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;
```

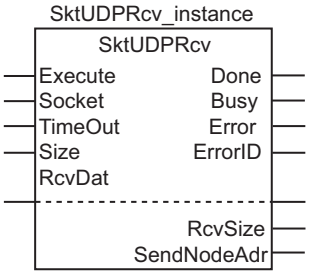
```
    0 :                                 // 正常終了
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;

    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。

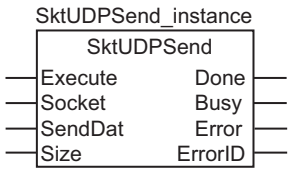
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktUDPRcv	UDP ソケット受信	FB		SktUDPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize, SendNodeAdr);

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.1-334)」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。

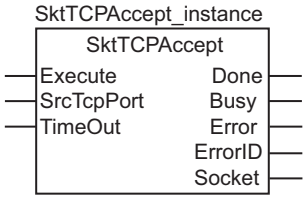
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktUDPSend	UDP ソケット送信	FB		SktUDPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.1-334)」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。

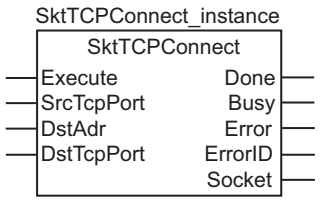
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPAccept	TCP ソケットアクセプト	FB		SktTCPAccept_instance(Execute, SrcTcpPort, TimeOut, Done, Busy, Error, ErrorID, Socket);

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.1-343)」を、参照してください。

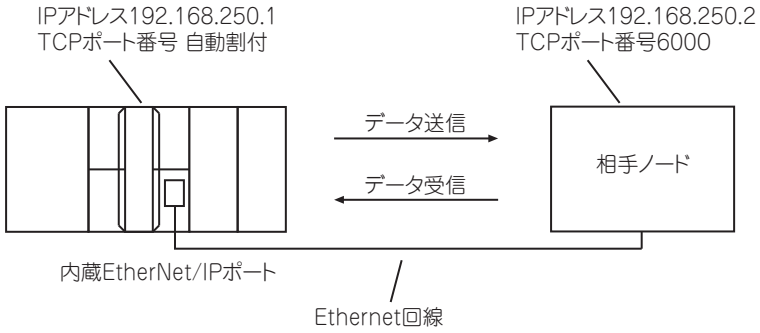
SktTCPConnect

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);

サンプルプログラム

NJ/NX/NY シリーズ CPU ユニットと、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。



NJ/NX/NY シリーズ CPU ユニット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;

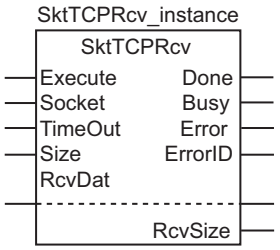
    0 :                                 // 正常終了
        DoTCP  :=FALSE;
        Trigger:=FALSE;

    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。

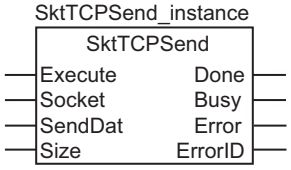
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPRcv	TCP ソケット受信	FB		SktTCPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID,RcvSize);

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.1-343)」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。

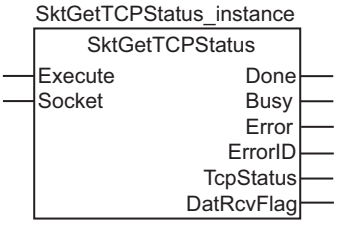
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTCPSend	TCP ソケット送信	FB		SktTCPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.1-343)」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。

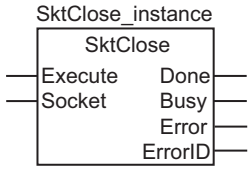
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktGetTCPStatus	TCP ソケットのステータス読出	FB		SktGetTCPStatus_instance(Execute, Socket, Done, Busy, Error, ErrorID, TcpStatus, DatRcvFlag);

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.1-343)」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。

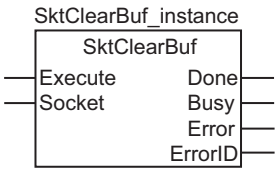
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktClose	TCP/UDP ソケットクローズ	FB		SktClose_instance(Execute, Socket, Done, Busy, Error, ErrorID);

サンプルプログラム

SktUDPCreate 命令の「サンプルプログラム (P.1-334)」、または SktTCPConnect 命令の「サンプルプログラム (P.1-343)」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。

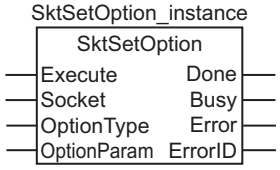
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktClearBuf	TCP/UDP ソケット受信バッファクリア	FB		SktClearBuf_instance(Execute, Socket, Done, Busy, Error, ErrorID);

サンプルプログラム

SktTCPConnect 命令の「サンプルプログラム (P.1-343)」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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.12 以降、かつ Sysmac Studio Ver.1.16 以降で、使用可能です。
形 NX1P2 CPU ユニットでは、CPU ユニット Ver.1.14 以降、かつ Sysmac Studio Ver.1.18 以降で、使用可能です。

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;
```

SktTLSConnect

接続済みの TCP コネクションを使用して、TLS セッションを確立します。

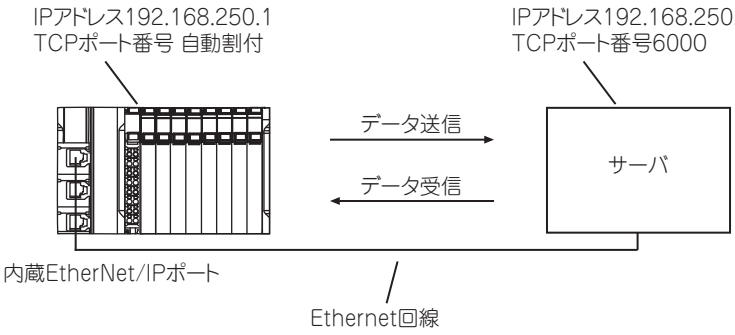
命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTLSConnect	TLS セッション確立	FB	SktTLSConnect_instance SktTLSConnect Execute / Socket / TLSSessionName Done / Busy / Error / ErrorID / Handle	SktTLSConnect_instance (Execute, Socket, TLSSessionName, Done, Busy, Error, ErrorID, Handle) ;

バージョン関連情報

本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

サンプルプログラム

形 NX102 CPU ユニットまたは形 NX1P2 CPU ユニットの内蔵 EtherNet/IP の通信ポート 1 と、サーバとの間で、セキュアソケット通信を使用して、データの送受信を行います。



ユーザプログラム

サンプルプログラムを実行する前に、セキュアソケット設定コマンドを使用し、セキュアソケット設定を CPU ユニットに設定してください。本サンプルプログラムは、ソケットサービス命令とセキュアソケットサービス命令を順に実行します。
命令を実行したときに異常が発生した場合は、異常を検出した命令以降の命令は実行されません。

処理の手順は、以下のとおりです。

1 **SktTCPConnect 命令で、サーバの TCP ポートにコネクト要求を行います。**

2 **SktTLSConnect 命令で、サーバへセキュアソケット通信の確立要求を行います。**

3 **SktTLSClearBuf 命令で、バッファのクリアを行います。**

4 **SktTLSWrite 命令で、データ送信要求を行います。送信するデータは tlsSendDat[]に格納したデータとします。**

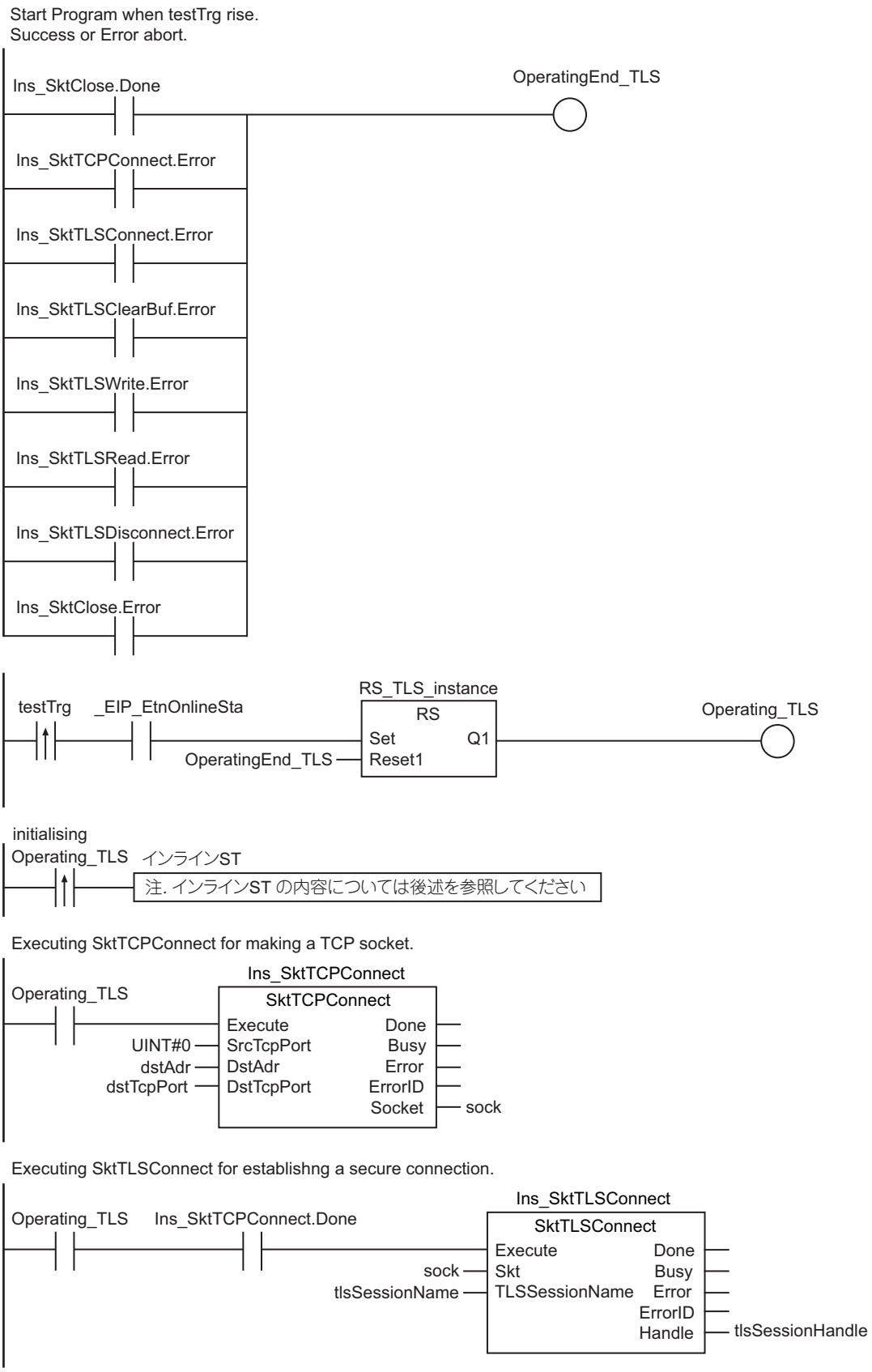
5 **SktTLSRead 命令で、データ受信要求を行います。受信したデータは tlsRcvDat[]に格納します。**

6 **SktTLSDisconnect 命令で、セキュアソケット通信の終了を行います。**

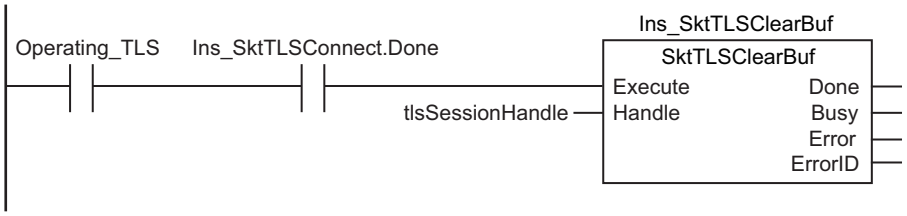
7 **SktClose 命令で、ソケットをクローズします。**

- **LD**

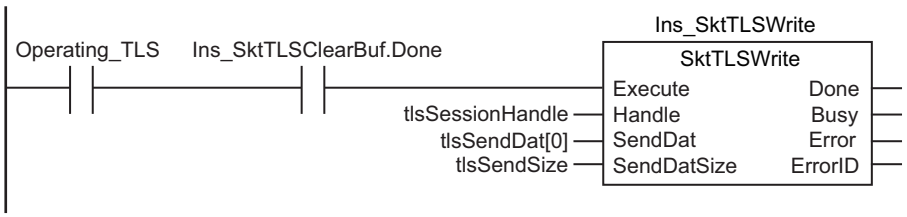
内部変数	名称	データ型	初期値	コメント
	testTrg	BOOL	FALSE	実行条件
	Operating_TLS	BOOL		処理中
	OperatingEnd_TLS	BOOL		状態遷移
	dstAdr	STRING[256]		送信先 IP アドレス、またはホスト名
	dstTcpPort	UINT		送信先ポート番号
	sock	_sSOCKET	(Handle := 0, SrcAdr := (PortNo := 0, IpAdr := ''), DstAdr := (PortNo := 0, IpAdr := ''))	ソケット
	tlsSessionName	STRING[17]		TLS セッション名
	tlsSessionHandle	DWORD		TLS セッションハンドル
	tlsSendDat	ARRAY[0..1024] OF BYTE	[1025(16#0)]	送信データ
	tlsRcvDat	ARRAY[0..1024] OF BYTE	[1025(16#0)]	受信データ
	tlsSendSize	UINT		送信するデータサイズ
	tlsRcvSize	UINT		受信できる最大データサイズ
	rcvedSize	UINT	0	受信したデータサイズ
	timeOutTime	UINT	10	タイムアウト時間
	Ins_SktTCPConnect	SktTCPConnect		
	Ins_SktTLSConnect	SktTLSConnect		
	Ins_SktTLSClearBuf	SktTLSClearBuf		
	Ins_SktTLSWrite	SktTLSWrite		
	Ins_SktTLSRead	SktTLSRead		
	Ins_SktTLSDisconnect	SktTLSDisconnect		
	Ins_SktClose	SktClose		
	RS_TLS_instance	RS		


Start Program when testTrg rise.
Success or Error abort.
Ins_SktClose.Done
OperatingEnd_TLS
Ins_SktTCPConnect.Error
Ins_SktTLSConnect.Error
Ins_SktTLSClearBuf.Error
Ins_SktTLSWrite.Error
Ins_SktTLSRead.Error
Ins_SktTLSDisconnect.Error
Ins_SktClose.Error
testTrg
_EIP_EtnOnlineSta
RS_TLS_instance
RS
Set
Reset1
Q1
OperatingEnd_TLS
Operating_TLS
initialising
Operating_TLS
インラインST
注. インラインST の内容については後述を参照してください
Executing SktTCPConnect for making a TCP socket.
Ins_SktTCPConnect
SktTCPConnect
Operating_TLS
Execute
Done
UINT#0
SrcTcpPort
Busy
dstAdr
DstAdr
Error
dstTcpPort
DstTcpPort
ErrorID
Socket
sock
Executing SktTLSConnect for establishng a secure connection.
Ins_SktTLSConnect
SktTLSConnect
Operating_TLS
Ins_SktTCPConnect.Done
Execute
Done
sock
Skt
Busy
tlsSessionName
TLSSessionName
Error
ErrorID
Handle
tlsSessionHandle

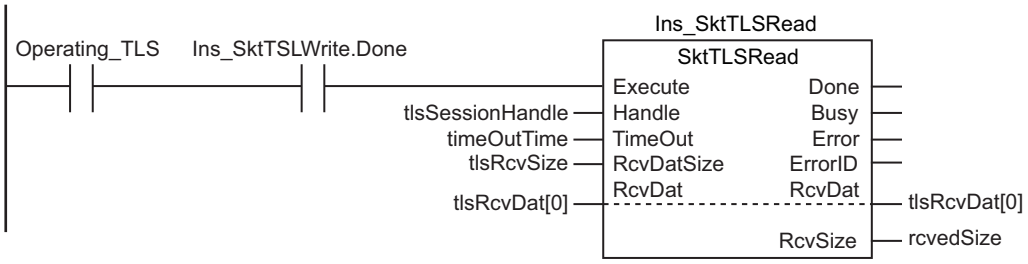
Execute SktTLSClearBuf for clearing the buffer.



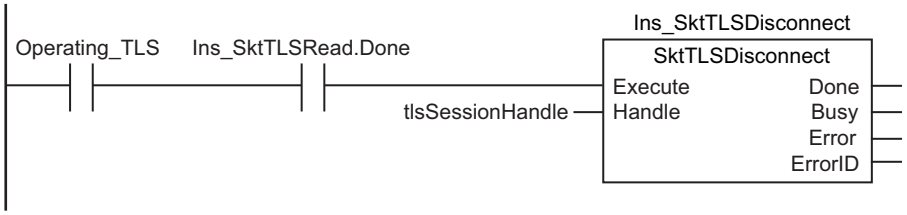
Execute SktTLSWrite for sending data to the server.



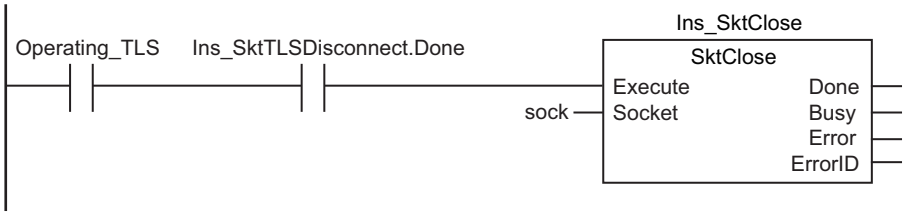
Execute SktTLSWrite for receiving data from the server.



Execute SktTLSDisconnect for disconnection the secure connection.



Executing SktClose for closing the TCP socket.



■ インライン ST の内容

```
//Configuring settings for connecting to the server.
//"tlsSessionName" must be equal to a name written in TLS setting file.
dstAdr         :='192.168.250.2';
dstTcpPort     :=6000;
tlsSessionName :='TLSSession0';

tlsSendSize    := SizeOfAry(In:=tlsSendDat);
tlsRcvSize     := SizeOfAry(In:=tlsRcvDat);
```

● ST

内部変数	名称	データ型	初期値	コメント
	testTrg	BOOL	FALSE	実行条件
	DoTLS	BOOL		処理中
	mainStage	UINT	1	状態遷移
	dstAdr	STRING[256]		送信先 IP アドレス、または ホスト名
	dstTcpPort	UINT		送信先ポート番号
	sock	_sSOCKET	(Handle := 0, SrcAdr := (PortNo := 0, IpAdr := ''), DstAdr := (PortNo := 0, IpAdr := ''))	ソケット
	tlsSessionName	STRING[17]		TLS セッション名
	tlsSessionHandle	DWORD		TLS セッションハンドル
	tlsSendDat	ARRAY[0..1024] OF BYTE	[1025(16#0)]	送信データ
	tlsRcvDat	ARRAY[0..1024] OF BYTE	[1025(16#0)]	受信データ
	tlsRcvSize	UINT	0	受信したデータサイズ
	timeOutTime	UINT	10	タイムアウト時間
	Ins_SktTCPConnect	SktTCPConnect		
	Ins_SktTLSConnect	SktTLSConnect		
	Ins_SktTLSClearBuf	SktTLSClearBuf		
	Ins_SktTLSWrite	SktTLSWrite		
	Ins_SktTLSRead	SktTLSRead		
	Ins_SktTLSDisconnect	SktTLSDisconnect		
	Ins_SktClose	SktClose		

```
//Start Program when testTrg rise.
IF ( (testTrg=TRUE) AND (DoTLS = FALSE) AND (_EIP_EtnOnlineSta =TRUE ) )THEN
    DoTLS:=TRUE;
    mainStage := 1;

    Ins_SktTCPConnect( Execute:=FALSE ); //Initilise instance
    Ins_SktTLSConnect( Execute:=FALSE ); //Initilise instance
    Ins_SktTLSClearBuf( Execute:=FALSE ); //Initilise instance
    Ins_SktTLSWrite( //Initilise instance
        Execute:=FALSE,
        SendDat:=tlsSendDat[0] ); //dummy
    Ins_SktTLSRead( //Initilise instance
        Execute:=FALSE,
        RcvDat:=tlsRcvDat[0] ); //dummy
    Ins_SktTLSDisconnect( Execute:=FALSE ); //Initilise instance
    Ins_SktClose( Execute:=FALSE ); //Initilise instance

    //Configuring settings for connecting to the server
    //"tlsSessionName" must be equal to a name written in TLS setting file.
    dstAdr :='192.168.250.2';
    dstTcpPort :=6000;
```

```
    tlsSessionName :='TLSSession0';

END_IF;

IF(DoTLS=TRUE) THEN
    CASE mainStage OF
    1://SktTCPConnect
        Ins_SktTCPConnect(
            Execute :=TRUE,
            SrcTcpPort :=UINT#0, //own TCP port number(auto allocation)
            DstAdr :=dstAdr, //target IP Address or host name
            DstTcpPort :=dstTcpPort, //target TCP port number
            Socket =>sock ); //socket variable
        IF ( Ins_SktTCPConnect.Done=TRUE )THEN
            mainStage:=2; //Success
        ELSIF( Ins_SktTCPConnect.Error=TRUE )THEN
            mainStage:=101; //Error
    END_IF;

    2://SktTLSConnect
    Ins_SktTLSConnect(
        Execute :=TRUE,
        Socket :=sock, //socket variable
        TLSSessionName :=tlsSessionName, //TLS session name
        Handle =>tlsSessionHandle ); //TLS session handle
    IF ( Ins_SktTLSConnect.Done=TRUE )THEN
        mainStage:=3; //Success
    ELSIF( Ins_SktTLSConnect.Error=TRUE )THEN
        mainStage:=102; //Error
    END_IF;

    3://SktTLSClearBuf
        Ins_SktTLSClearBuf(
            Execute :=TRUE,
            Handle :=tlsSessionHandle); //TLS session handle
        IF( Ins_SktTLSClearBuf.Done=TRUE )THEN
            mainStage:=4; //Success
        ELSIF( Ins_SktTCPConnect.Error=TRUE )THEN
            mainStage:=103; //Error
        END_IF;

    4: //SktTLSWrite
        Ins_SktTLSWrite(
            Execute :=TRUE,
            Handle :=tlsSessionHandle, //TLS session handle
            SendDat :=tlsSendDat[0], //sending data
            SendDatSize :=SizeOfAry( In:=tlsSendDat ) ); //sending data size
        IF( Ins_SktTLSWrite.Done=TRUE )THEN
            mainStage:=5; //Success
        ELSIF( Ins_SktTLSWrite.Error=TRUE )THEN
            mainStage:=104; //Error
```

```
        END_IF;

    5://SktTLSRead
        Ins_SktTLSRead(
            Execute :=TRUE,
            Handle :=tlsSessionHandle, //TLS session handle
            TimeOut :=timeOutTime, //timeout time
            RcvDatSize :=SizeOfAry( In:=tlsRcvDat ), //receive buffer size
            RcvDat :=tlsRcvDat[0], //receiving data
            RcvSize =>tlsRcvSize ); //receiving data size
        IF( Ins_SktTLSRead.Done=TRUE )THEN
            mainStage:=6; //Success
        ELSIF( Ins_SktTLSRead.Error=TRUE )THEN
            mainStage:=105; //Error
        END_IF;

    6://SktTLSDisconnect
        Ins_SktTLSDisconnect(
            Execute :=TRUE,
            Handle :=tlsSessionHandle ); //TLS session handle
        IF( Ins_SktTLSDisconnect.Done=TRUE )THEN
            mainStage:=7; //Success
        ELSIF( Ins_SktTLSDisconnect.Error=TRUE )THEN
            mainStage:=106; //Error
        END_IF;

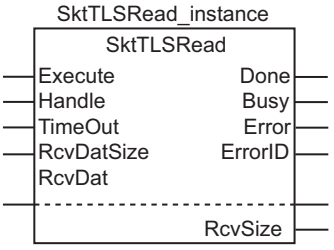
    7://SktClose
        Ins_SktClose(
            Execute :=TRUE,
            Socket :=sock ); //socket variable
        IF( Ins_SktClose.Done=TRUE )THEN
            mainStage:=0; //Success
        ELSIF(Ins_SktClose.Error=TRUE )THEN
            mainStage:=107; //Error
        END_IF;

    0://Success
        DoTLS := FALSE;
        testTrg := FALSE;

    ELSE //Error abort
        DoTLS := FALSE;
        testTrg := FALSE;
    END_CASE;
END_IF;
```

SktTLSRead

EtherNet/IP の、指定した TLS セッション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
SktTLSRead	TLS 受信	FB		SktTLSRead_instance (Execute, Handle, TimeOut, RcvDatSize, RcvDat, Done, Busy, Error, ErrorID, RcvSize);

バージョン関連情報

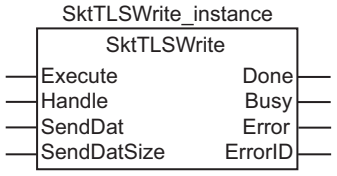
本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

サンプルプログラム

SktTLSConnect 命令の「サンプルプログラム (P.1-357)」を参照してください。

SktTLSWrite

EtherNet/IP の、指定した TLS セッション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
SktTLSWrite	TLS 送信	FB		SktTLSWrite_instance (Execute, Handle, SendDat, SendDatSize, Done, Busy, Error, ErrorID) ;

バージョン関連情報

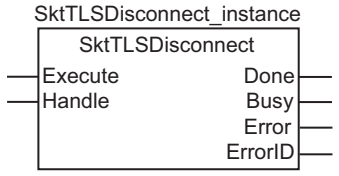
本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

サンプルプログラム

SktTLSConnect 命令の「サンプルプログラム (P.1-357)」を参照してください。

SktTLSDisconnect

EtherNet/IP の、指定した TLS セッ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
SktTLSDisconnect	TLS セッション切断	FB		SktTLSDisconnect_instance (Execute, Handle, Done, Busy, Error, ErrorID) ;

バージョン関連情報

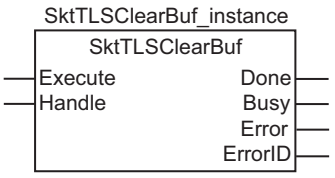
本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

サンプルプログラム

SktTLSConnect 命令の「サンプルプログラム (P.1-357)」を参照してください。

SktTLSClearBuf

EtherNet/IP の、指定した TLS セッション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
SktTLSClearBuf	TLS セッション受信バッファクリア	FB		SktTLSClearBuf_instance (Execute, Handle, Done, Busy, Error, ErrorID) ;

バージョン関連情報

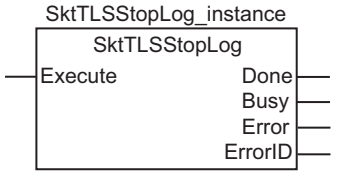
本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

サンプルプログラム

SktTLSConnect 命令の「サンプルプログラム (P.1-357)」を参照してください。

SktTLSStopLog

セキュアソケット通信ログの出力を停止します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SktTLSStopLog	セキュアソケット通信ログ停止	FB		SktTLSStopLog (Execute, Done, Busy, Error, ErrorID) ;

バージョン関連情報

本命令は、以下の CPU ユニットで使用可能です。
- 形 NX502 CPU ユニット
- 形 NX102-□□00 の CPU ユニット Ver.1.46 以降、形 NX102-□□20 の CPU ユニット Ver.1.37 以降、かつ Sysmac Studio Ver.1.46 以降
- 形 NX1P2 CPU ユニットの CPU ユニット Ver.1.46 以降、かつ Sysmac Studio Ver.1.46 以降

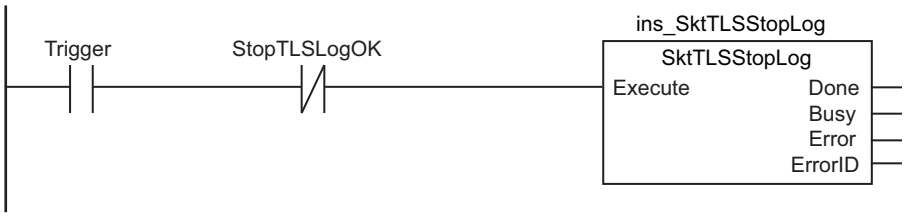
サンプルプログラム

トリガが TRUE になったときに、セキュアソケット通信ログの出力を停止します。

LD

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	セキュアソケット通信ログの停止処理を開始するトリガ
	StopTLSLogOK	BOOL	FALSE	ログ停止が完了したことを示す変数
	ins_SktTLSStopLog	SktTLSStopLog		

Start Program when Trigger rise.



If SktTLSStopLog is success, set stopTLSlogOK to TRUE.



ST

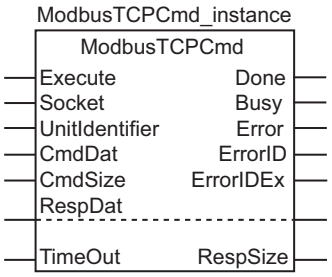
内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL	FALSE	セキュアソケット通信ログの停止処理を開始するトリガ
	DoStop	BOOL	FALSE	この変数が TRUE の時に、SktTLSLogStop 命令を実行する
	StopTLSLogOK	BOOL	FALSE	ログ停止が完了したことを示す変数
	ins_SktTLSStopLog	SktTLSStopLog		

```
//Start Program when Trigger rise.
IF ( (Trigger=TRUE) AND (StopTLSLogOK=FALSE) ) THEN
    DoStop := TRUE;
    ins_SktTLSStopLog( Execute:=FALSE );
END_IF;

IF (DoStop=TRUE) THEN
    //Execute sktTLSStopLog for stopping TLS log.
    ins_SktTLSStopLog( Execute:=TRUE );
    Trigger := FALSE;
    IF (ins_SktTLSStopLog.Done=TRUE) THEN
        //Success
        StopTLSLogOK := TRUE;
        DoStop := FALSE;
    END_IF;
    IF (ins_SktTLSStopLog.Error=TRUE) THEN
        //Error abort
        DoStop := FALSE;
    END_IF;
END_IF;
```

ModbusTCPCmd

Modbus TCP プロトコルで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
ModbusTCP Cmd	Modbus TCP 汎用コマンド送信	FB		ModbusTCPCmd_instance(Execute, Socket, UnitIdentifier, CmdDat, CmdSize, RespDat, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

バージョン関連情報

本命令は、形 NX502 CPU ユニットおよび形 NX102 CPU ユニットで使用可能です。

サンプルプログラム

CPU ユニットは、形 NX502 または形 NX102 で IP アドレスが 192.168.250.1 とします。
通信相手の Modbus TCP スレーブ機器（192.168.250.10 ポート 502）に対して、「Trigger」が ON になると、バッファをクリアし、その後 Modbus コマンドを送信します。
通信相手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。
汎用コマンドを使用し、変数読み出しを行います。
Modbus Messaging on TCP/IP implementation guide V1.0b にて、TCP-NODELAY オプションが推奨されているため、本サンプルプログラムでも、SktTCPConnect の後で、オプションの指定をしています。

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL		実行条件
	DoModbusTrigger	BOOL		処理中
	Nodelay	BOOL		NoDelay 設定
	Stage	INT		状態遷移
	Socket	_sSOCKET		ソケット
	ModbusCmdDat	ARRAY[0..4] OF BYTE		コマンドデータ
	ModbusDatSize	UINT		コマンドデータサイズ
	ModbusRespDat	ARRAY[0..253] OF BYTE		レスポンスデータ
	ModbusRespSize	UINT		レスポンスデータサイズ
	SktTCPConnect_instance	SktTCPConnect		
	SktSetOption_instance	SktTSetOption		
	SktClearBuf_instance	SktClearBuf		
	ModbusTCPCmd_instance	ModbusTCPCmd		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP1_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 (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
    DoModbusTrigger:=TRUE;
    Nodelay:=TRUE;

    SktTCPConnect_instance(Execute:= FALSE);
    SktSetOption_instance(
        Execute:=FALSE,
        Socket:=Socket,
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);

    SktClearBuf_instance(
        Execute:=FALSE,
        Socket:=Socket);
    ModbusTCPCmd_instance(
        Execute:=FALSE,
        Socket:=Socket,
        CmdDat:=ModbusCmdDat[0],
        CmdSize:=ModbusDatSize,
        RespDat:=ModbusRespDat[0]);

    SktClose_instance(
        Execute:=FALSE,
        Socket:=Socket);

    Stage:=1; // 初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1:  // ソケット接続要求
        SktTCPConnect_instance(
            Execute:=TRUE,
            SrcTcpPort:=UINT#502,
            DstAdr:='192.168.250.10',
            DstTcpPort:=UINT#502,
            Socket=>Socket);
        IF (SktTCPConnect_instance.Done=TRUE) THEN
            Stage:=2; // ソケット接続 正常終了
        ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
            Stage:=99; // ソケット接続 異常終了
        END_IF;

    2:  //TCP-NODELAY オプション要求
        SktSetOption_instance(
```

```
        Execute:=TRUE,
        Socket:=Socket,
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);
    IF (SktSetOption_instance.Done=TRUE) THEN
        Stage:=3; // オプション設定 正常終了
    ELSIF (SktSetOption_instance.Error=TRUE) THEN
        Stage:=99; // オプション設定 異常終了
    END_IF;

3:  // バッファクリア要求
    SktClearBuf_instance(
        Execute:=TRUE,
        Socket:=Socket);
    IF (SktClearBuf_instance.Done=TRUE) THEN
        Stage:=4; // バッファクリア 正常終了
    ELSIF (SktClearBuf_instance.Error=TRUE) THEN
        Stage:=99; // バッファクリア 異常終了
    END_IF;

4:  //Modbus Cmd 送信要求
    ModbusCmdDat[0]:=BYTE#16#03; // ファンクションコード（変数読み出し）
    ModbusCmdDat[1]:=BYTE#16#00; // 読み出し開始アドレス(H)
    ModbusCmdDat[2]:=BYTE#16#20; // 読み出し開始アドレス(L)
    ModbusCmdDat[3]:=BYTE#16#00; // データ数(H)
    ModbusCmdDat[4]:=BYTE#16#01; // データ数(L)
    ModbusDatSize:=5;
    ModbusTCPCmd_instance(
        Execute:=TRUE,
        Socket:=Socket,
        CmdDat:=ModbusCmdDat[0],
        CmdSize:=ModbusDatSize,
        RespDat:=ModbusRespDat[0],
        RespSize=>ModbusRespSize);
    IF (ModbusTCPCmd_instance.Done=TRUE) THEN
        Stage:=5; //ModbusTCPWrite 命令 正常終了
    ELSIF (ModbusTCPCmd_instance.Error=TRUE) THEN
        Stage:=99; //ModbusTCPWrite 命令 異常終了
    END_IF;

5:  // ソケットクローズ要求
    SktClose_instance(
        Execute:=TRUE,
        Socket:=Socket);
    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=6; // ソケットクローズ 正常終了
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=99; // ソケットクローズ 異常終了
    END_IF;

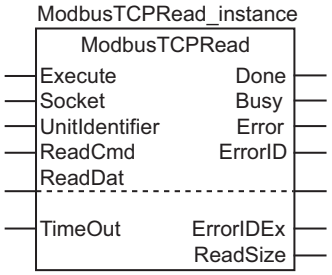
6:  //ModbusTCPRead 命令正常終了後の処理
```

```
        Trigger:=FALSE;
        DoModbusTrigger:=FALSE;

    99:  // 異常処理
        Trigger:=FALSE;
        DoModbusTrigger:=FALSE;
    END_CASE;
END_IF;
```

ModbusTCPRead

Modbus TCP プロトコルで、読み出しコマンドを送信して要求した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
ModbusTCP Read	Modbus TCP Read コマンド送信	FB		ModbusTCPRead_instance(Execute, Socket, UnitIdentifier, ReadCmd, ReadDat, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, ReadSize);

バージョン関連情報

本命令は、形 NX502 CPU ユニットおよび形 NX102 CPU ユニットで使用可能です。

サンプルプログラム

CPU ユニットは、形 NX502 または形 NX102 で IP アドレスが 192.168.250.1 とします。
通信相手の Modbus TCP スレーブ機器（192.168.250.10 ポート 502）に対して、「Trigger」が ON になると、バッファをクリアし、その後 Modbus コマンドを送信します。
通信相手の読み出し開始アドレス 19 から 1 つのコイルのステータスを読み出します。Read コマンドを使用し、変数読み出しを行います。
Modbus Messaging on TCP/IP implementation guide V1.0b にて、TCP-NODELAY オプションが推奨されているため、本サンプルプログラムでも、SktTCPConnect の後で、オプションの指定をしています。

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL		実行条件
	DoModbusTrigger	BOOL		処理中
	Nodelay	BOOL		NoDelay 設定
	Stage	INT		状態遷移
	Socket	_sSOCKET		ソケット
	ModbusReadCmd	_sMODBUS_READ		読み出しコマンド
	ModbusRespDat	BOOL		読み出しデータ
	ModbusReadSize	UINT		読み出しデータサイズ
	SktTCPConnect_instance	SktTCPConnect		
	SktSetOption_instance	SktTSetOption		
	SktClearBuf_instance	SktClearBuf		
	ModbusTCPRead_instance	ModbusTCPRead		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP1_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 (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
    DoModbusTrigger:=TRUE;
    Nodelay:=TRUE;

    SktTCPConnect_instance(Execute:= FALSE);
    SktSetOption_instance(
        Execute:=FALSE,
        Socket:=Socket,
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);

    SktClearBuf_instance(
        Execute:=FALSE,
        Socket:=Socket);
    ModbusTCPRead_instance(
        Execute:=FALSE,
        Socket:=Socket,
        ReadCmd:=ModbusReadCmd,
        ReadDat:=ModbusReadDat);

    SktClose_instance(
        Execute:=FALSE,
        Socket:=Socket);

    Stage:=1; // 初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1: // ソケット接続要求
        SktTCPConnect_instance(
            Execute:=TRUE,
            SrcTcPPort:=UINT#502,
            DstAdr:='192.168.250.10',
            DstTcpPort:=UINT#502,
            Socket=>Socket);
        IF (SktTCPConnect_instance.Done=TRUE) THEN
            Stage:=2; // ソケット接続 正常終了
        ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
            Stage:=99; // ソケット接続 異常終了
        END_IF;

    2: //TCP-NODELAY オプション要求
        SktSetOption_instance(
            Execute:=TRUE,
            Socket:=Socket,
```

```
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);

    IF (SktSetOption_instance.Done=TRUE) THEN
        Stage:=3; // オプション設定 正常終了
    ELSIF (SktSetOption_instance.Error=TRUE) THEN
        Stage:=99; // オプション設定 異常終了
    END_IF;

3: // バッファクリア要求
    SktClearBuf_instance(
        Execute:=TRUE,
        Socket:=Socket);

    IF (SktClearBuf_instance.Done=TRUE) THEN
        Stage:=4; // バッファクリア 正常終了
    ELSIF (SktClearBuf_instance.Error=TRUE) THEN
        Stage:=99; // バッファクリア 異常終了
    END_IF;

4: //Modbus Read 要求
    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

    ModbusTCPRead_instance(
        Execute:=TRUE,
        Socket:=Socket,
        ReadCmd:=ModbusReadCmd,
        ReadDat:=ModbusReadDat,
        ReadSize=>ModbusReadSize);

    IF (ModbusTCPRead_instance.Done=TRUE) THEN
        Stage:=5; //ModbusTCPWrite 命令 正常終了
    ELSIF (ModbusTCPRead_instance.Error=TRUE) THEN
        Stage:=99; //ModbusTCPWrite 命令 異常終了
    END_IF;

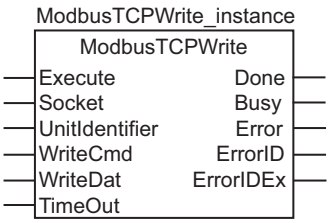
5: // ソケットクローズ要求
    SktClose_instance(
        Execute:=TRUE,
        Socket:=Socket);
    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=6; // ソケットクローズ 正常終了
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=99; // ソケットクローズ 異常終了
    END_IF;

6: //ModbusTCPRead 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;
```

ModbusTCPWrite

Modbus TCP プロトコルで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
ModbusTCP Write	Modbus TCP Write コマンド送信	FB		ModbusTCPWrite_instance(Execute, Socket, UnitIdentifier, WriteCmd, WriteDat, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx);

バージョン関連情報

本命令は、形 NX502 CPU ユニットおよび形 NX102 CPU ユニットで使用可能です。

サンプルプログラム

CPU ユニットは、形 NX502 または形 NX102 で IP アドレスが 192.168.250.1 とします。
通信相手の Modbus TCP スレーブ機器（192.168.250.10 ポート 502）に対して、「Trigger」が ON になると、バッファをクリアし、その後 Modbus コマンドを送信します。
通信相手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。Write コマンドを使用し、変数書き込みを行います。
Modbus Messaging on TCP/IP implementation guide V1.0b にて、TCP-NODELAY オプションが推奨されているため、本サンプルプログラムでも、SktTCPConnect のあとで、オプションの指定をしています。

ST

内部変数	名称	データ型	初期値	コメント
	Trigger	BOOL		実行条件
	DoModbusTrigger	BOOL		処理中
	Nodelay	BOOL		NoDelay 設定
	Stage	INT		状態遷移
	Socket	_sSOCKET		ソケット
	ModbusWriteCmd	_sMODBUS_WRITE		書き込みコマンド
	ModbusWriteDat	BOOL		書き込みデータ
	SktTCPConnect_instance	SktTCPConnect		
	SktSetOption_instance	SktTSetOption		
	SktClearBuf_instance	SktClearBuf		
	ModbusTCPWrite_instance	ModbusTCPWrite		
	SktClose_instance	SktClose		

外部変数	名称	データ型	コンスタント	コメント
	_EIP1_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 (DoModbusTrigger=FALSE) AND (_EIP1_EtnOnlineSta=TRUE) THEN
    DoModbusTrigger:=TRUE;
    Nodelay:=TRUE;

    SktTCPConnect_instance(Execute:= FALSE);
    SktSetOption_instance(
        Execute:=FALSE,
        Socket:=Socket,
        OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
        OptionParam:=Nodelay);

    SktClearBuf_instance(
        Execute:=FALSE,
        Socket:=Socket);
    ModbusTCPWrite_instance(
        Execute:=FALSE,
        Socket:=Socket,
        WriteCmd:=ModbusWriteCmd,
        WriteDat:=ModbusWriteDat);

    SktClose_instance(
        Execute:=FALSE,
        Socket:=Socket);

    Stage:=1; // 初期化完了
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1: // ソケット接続要求
        SktTCPConnect_instance(
            Execute:=TRUE,
            SrcTcpPort:=UINT#502,
            DstAdr:='192.168.250.10',
            DstTcpPort:=UINT#502,
            Socket=>Socket);
        IF (SktTCPConnect_instance.Done=TRUE) THEN
            Stage:=2; // ソケット接続 正常終了
        ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
            Stage:=99; // ソケット接続 異常終了
        END_IF;

    2: //TCP-NODELAY オプション要求
        SktSetOption_instance(
            Execute:=TRUE,
            Socket:=Socket,
            OptionType:=_eSKT_OPTION_TYPE#_TCP_NODELAY,
            OptionParam:=Nodelay);
        IF (SktSetOption_instance.Done=TRUE) THEN
```

```
            Stage:=3; // オプション設定 正常終了
        ELSIF (SktSetOption_instance.Error=TRUE) THEN
            Stage:=99; // オプション設定 異常終了
        END_IF;

    3: // バッファクリア要求
        SktClearBuf_instance(
            Execute:=TRUE,
            Socket:=Socket);
        IF (SktClearBuf_instance.Done=TRUE) THEN
            Stage:=4; // バッファクリア 正常終了
        ELSIF (SktClearBuf_instance.Error=TRUE) THEN
            Stage:=99; // バッファクリア 異常終了
        END_IF;

    4: //Modbus Write 要求
        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

        ModbusTCPWrite_instance(
            Execute:=TRUE,
            Socket:=Socket,
            WriteCmd:=ModbusWriteCmd,
            WriteDat:=ModbusWriteDat);

        IF (ModbusTCPWrite_instance.Done=TRUE) THEN
            Stage:=5; //ModbusTCPWrite 命令 正常終了
        ELSIF (ModbusTCPWrite_instance.Error=TRUE) THEN
            Stage:=99; //ModbusTCPWrite 命令 異常終了
        END_IF;

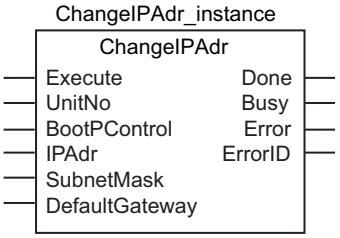
    5: // ソケットクローズ要求
        SktClose_instance(
            Execute:=TRUE,
            Socket:=Socket);
        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=6; // ソケットクローズ 正常終了
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=99; // ソケットクローズ 異常終了
        END_IF;

    6: //ModbusTCPWrite 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;
```

ChangeIPAdr

CPU ユニットの内蔵 EtherNet/IP ポート、または EtherNet/IP ユニットの 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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

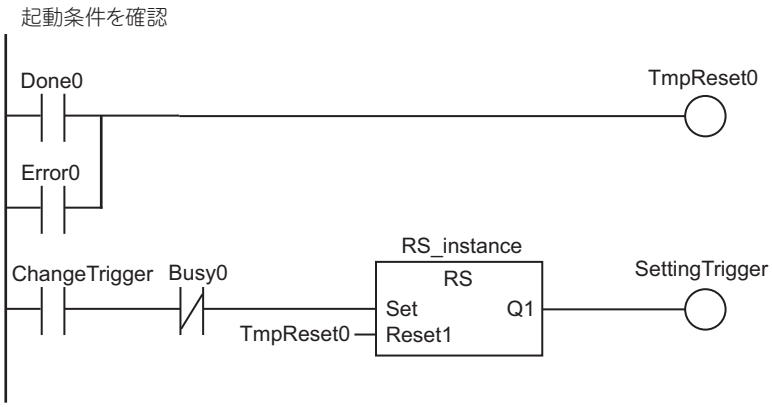
サンプルプログラム

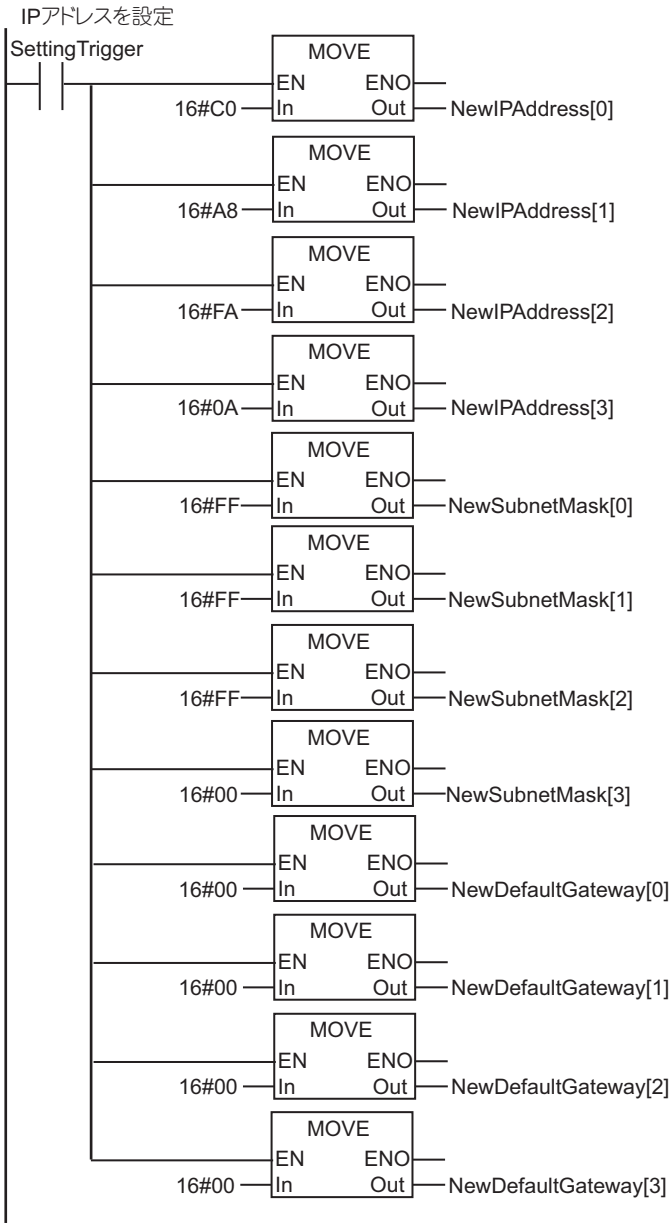
内蔵 EtherNet/IP ポートの IP アドレスを、以下の IP アドレスの固定設定に変更します。

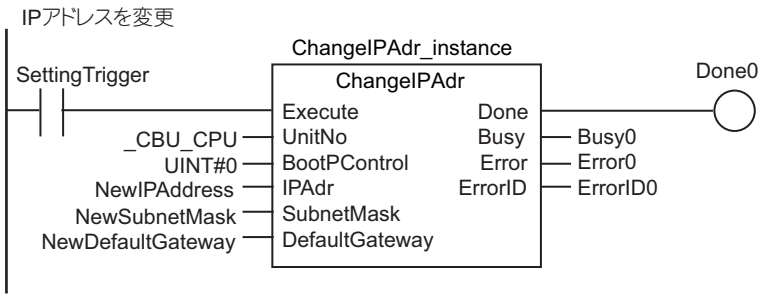
項目	内容
IP アドレス	192.168.250.10
サブネットマスク	255.255.255.0
デフォルトゲートウェイ	0.0.0.0

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 ユニットの内蔵 EtherNet/IP ポートまたは EtherNet/IP ユニットの FTP のログイン名とパスワードを変更します。

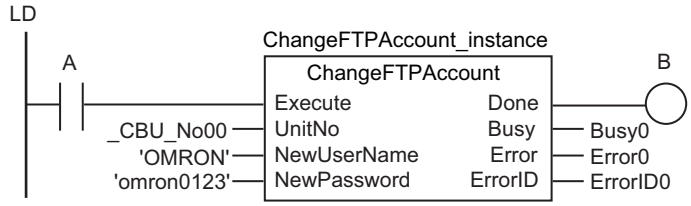
命令	名称	FB/FUN	グラフィック表現	ST 表現
ChangeFTP Account	FTP アカウント変更	FB	ChangeFTPAccount_instance ChangeFTPAccount Execute / UnitNo / NewUserName / NewPassword Done / Busy / Error / ErrorID	ChangeFTPAccount_instance(Execute, UnitNo, NewUserName, NewPassword, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

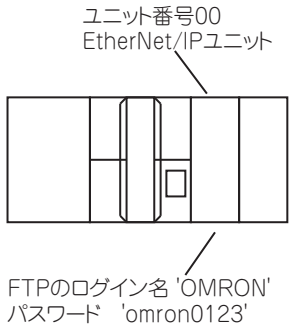
記述例

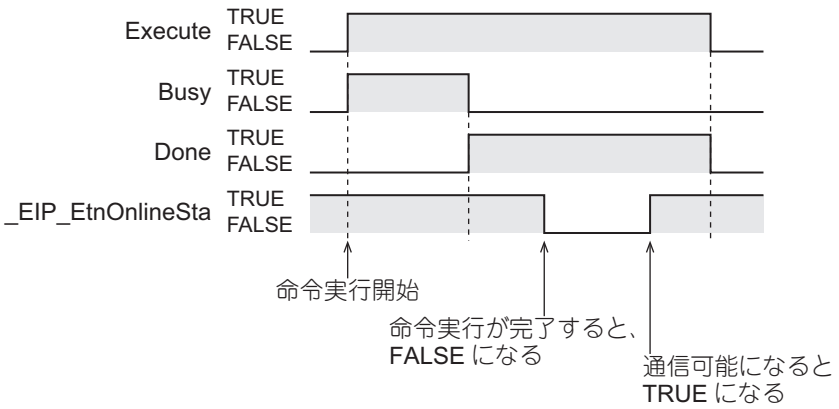
ユニット番号 00 の EtherNet/IP ユニットの、FTP のログイン名とパスワードを変更する場合の記述例を以下に示します。
「Execute」=A を表示器などから ON にすることで、FTP のログイン名を'OMRON'に、パスワードを'omron0123'に変更します。




```
ChangeFTPAccount_instance(A,_CBU_No00,'OMRON','omron0123',B,Busy0,Error0,ErrorID0);
```

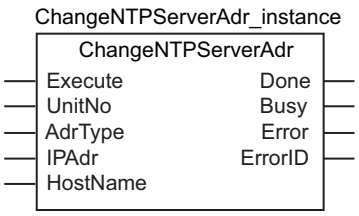
ユニット番号00のEtherNet/IPユニットの、FTPのログイン名を'OMRON'に、
パスワードを'omron0123'に変更します。




Execute
TRUE
FALSE
Busy
TRUE
FALSE
Done
TRUE
FALSE
_EIP_EtnOnlineSta
TRUE
FALSE
命令実行開始
命令実行が完了すると、
FALSE になる
通信可能になると
TRUE になる

ChangeNTPServerAdr

CPU ユニットの内蔵 EtherNet/IP ポートまたは EtherNet/IP ユニットの NTP サーバの宛先を変更します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ChangeNTPServerAdr	NTP サーバ宛先変更	FB		ChangeNTPServerAdr_instance(Execute, UnitNo, AdrType, IPAdr, HostName, Done, Busy, Error, ErrorID);

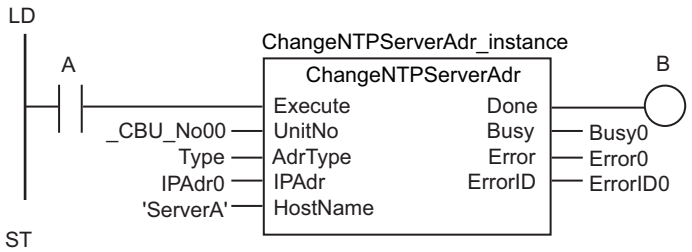
バージョン関連情報

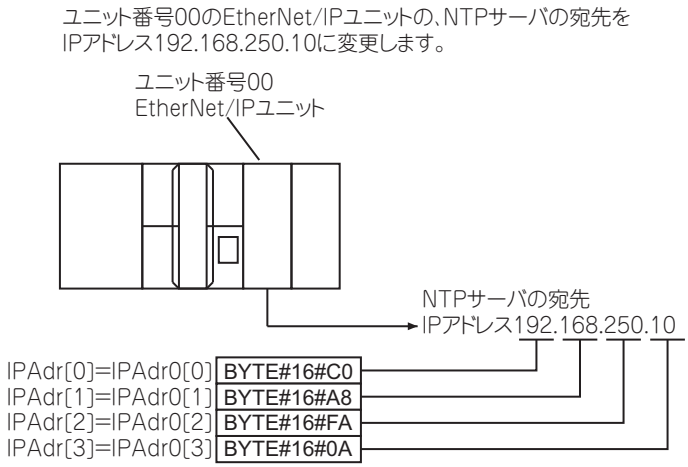
本命令は、CPU ユニット Ver.1.02 以降、かつ Sysmac Studio Ver.1.03 以降で、使用可能です。

記述例

ユニット番号 00 の EtherNet/IP ユニットの、NTP サーバの宛先を変更する場合の記述例を以下に示します。

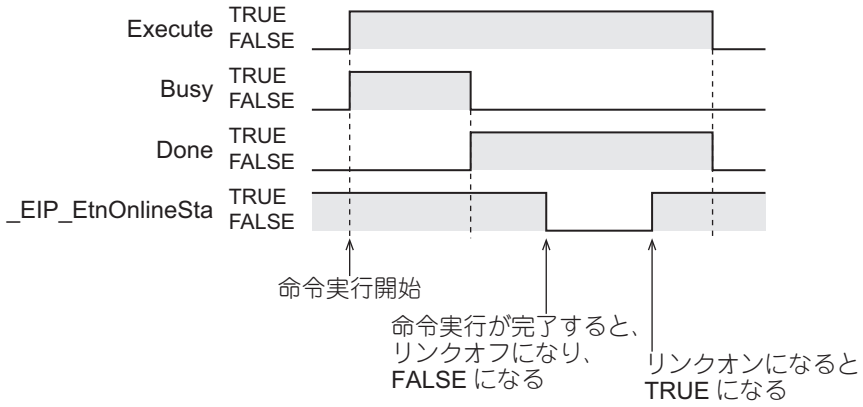
「Execute」=A を表示器などから ON にすることで、NTP サーバの宛先を変更します。
たとえば、IPAdr0[0]=BYTE#16#C0、IPAdr0[1]=BYTE#16#A8、IPAdr0[2]=BYTE#16#FA、IPAdr0[3]=BYTE#16#0A とします。
「AdrType」=Type が TRUE の場合、NTP サーバの宛先を IP アドレス 192.168.250.10 に変更します。
「AdrType」=Type が FALSE の場合、NTP サーバの宛先をホスト名 ServerA に変更します。





システム定義変数「_EIP_EtnOnlineSta」、「_EIP1_EtnOnlineSta」、「_EIP2_EtnOnlineSta」により、リンクオン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」、「_EIP2_EtnOnlineSta」についても同様になります。

「Busy」が TRUE から FALSE になると、リンクオフになり、「_EIP_EtnOnlineSta」は、FALSE になります。その後、リンクオンになると、「_EIP_EtnOnlineSta」は、FALSE から TRUE になります。



FTPGetFileList

FTP サーバ内のファイルのリストを取得します。

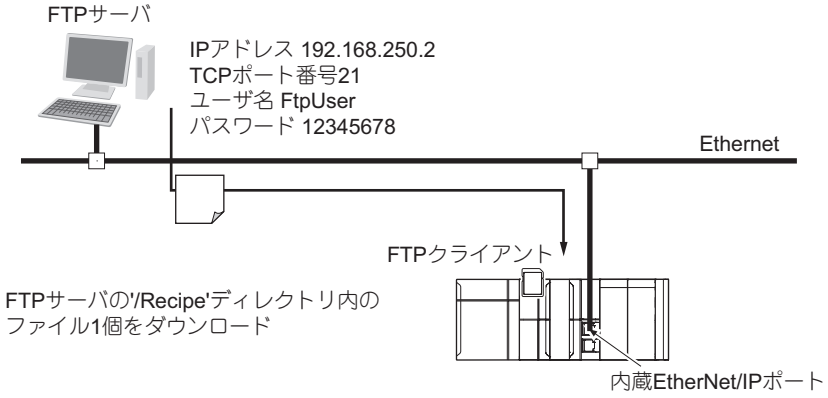
命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPGetFileList	FTP サーバのファイルリスト取得	FB	FTPGetFileList_instance FTPGetFileList Execute / ConnectSvr / SvrDirName / GetFileNum / SortOrder / ExecOption / RetryCfg / Cancel / FileList Done / Busy / CommandCanceled / Error / ErrorID / ErrorIDEx / StoredNum	FTPGetFileList_instance(Execute, ConnectSvr, SvrDirName, GetFileNum, SortOrder, ExecOption, RetryCfg, Cancel, FileList, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, StoredNum);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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[]

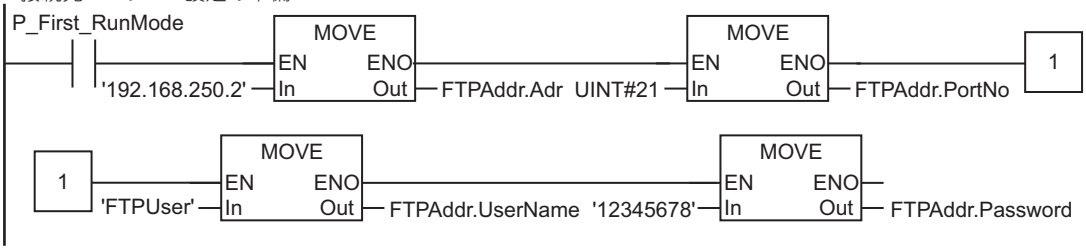
2 **FTPGetFile 命令を使用して、1 で取得したファイルリストの中から、名前昇順に並べたうちの最後のファイルをダウンロードします。**
ファイルの格納先は、SD メモリカードのルートディレクトリです。

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。

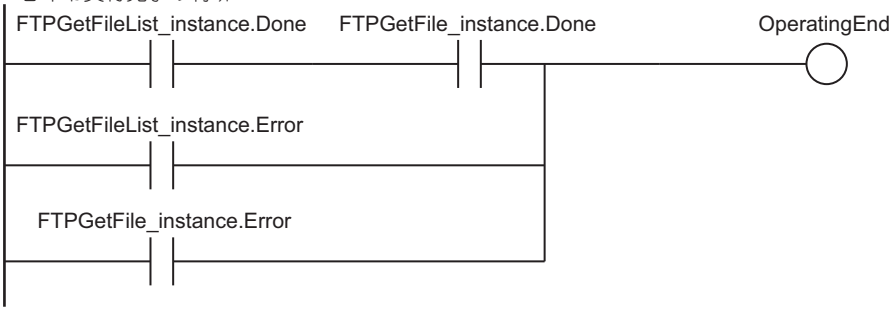
LD

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
	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

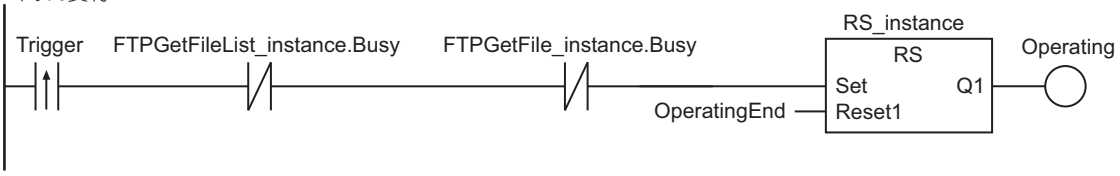
接続先FTPサーバ設定の準備



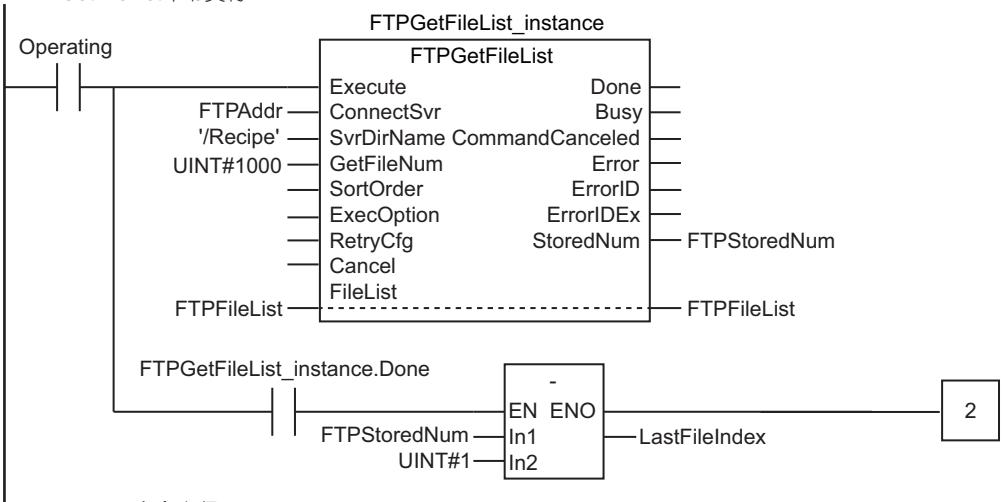
各命令実行完了の判断



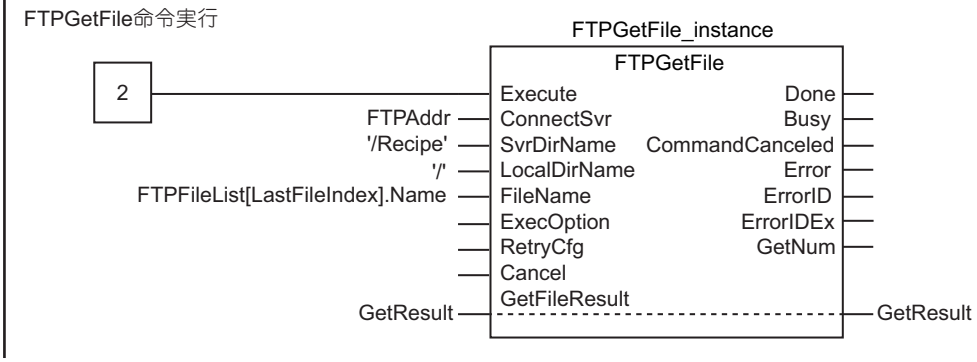
トリガ受付

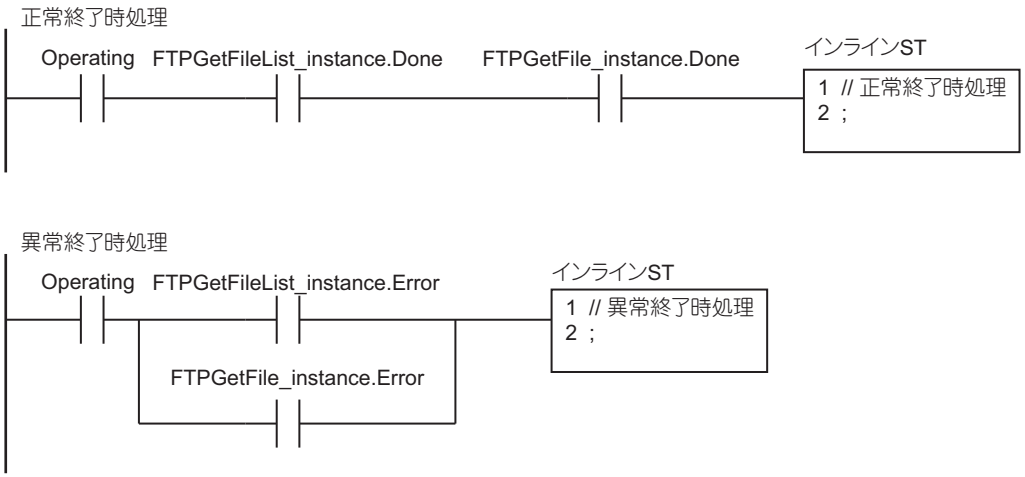


FTPGetFileList命令実行



FTPGetFile命令実行





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
	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

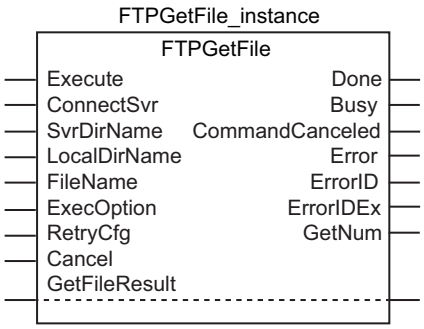
```
// 接続先 FTP サーバ設定の準備
IF P_First_RunMode THEN
   FTPAddr.Adr := '192.168.250.2';  // アドレス
   FTPAddr.PortNo := UINT#21;  // ポート番号
   FTPAddr.UserName := 'FtpUser';  // ユーザー名
   FTPAddr.Password := '12345678';  // パスワード
END_IF;
```

```
// 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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

サンプルプログラム

FTPGetFileList 命令の、「サンプルプログラム (P.1-389)」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。

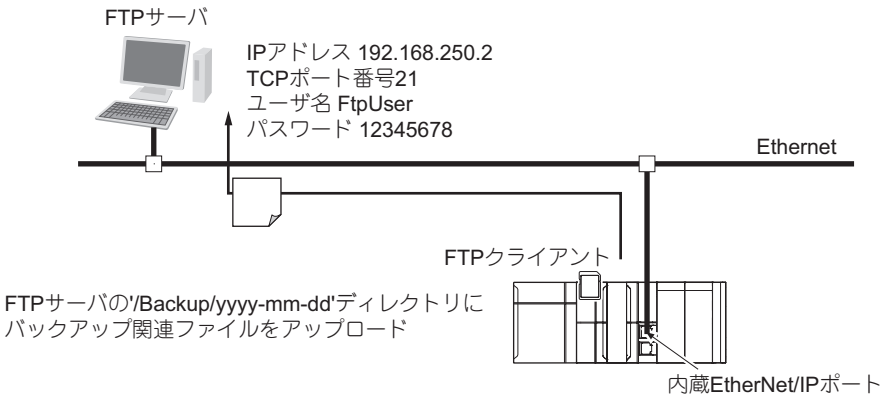
命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPPutFile	FTP サーバへのファイルアップロード	FB	FTPPutFile_instance FTPPutFile Execute / ConnectSvr / SvrDirName / LocalDirName / FileName / ExecOption / RetryCfg / Cancel / PutFileResult Done / Busy / CommandCanceled / Error / ErrorID / ErrorIDEx / PutNum	FTPPutFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, PutFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, PutNum);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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。

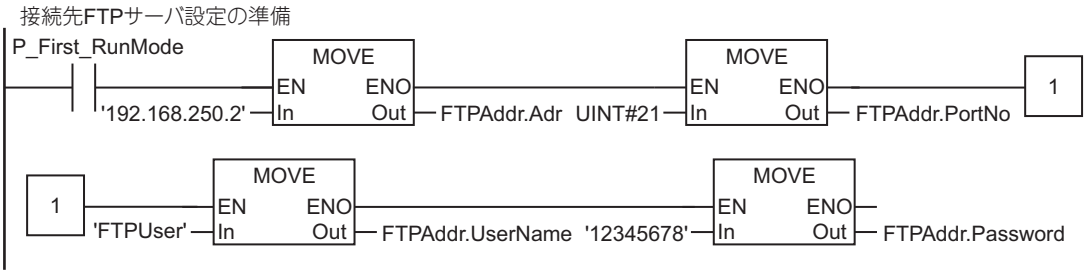
1 **BackupToMemoryCard 命令を使用して、コントローラのバックアップ関連ファイルを、SD メモリカードのルートディレクトリに保存します。**

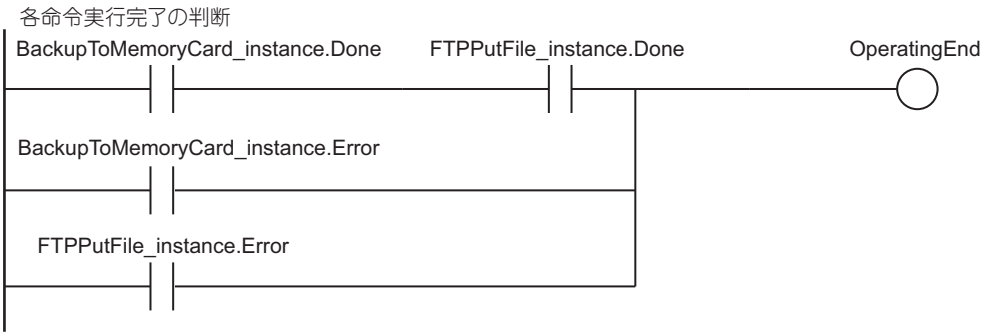
2 **FTPPutFile 命令を使用して、バックアップ関連ファイルを FTP サーバの'/Backup/yyyy-mm-dd'ディレクトリにアップロードします。転送ファイル名は、ワイルドカード指定で'*.*'とします。**

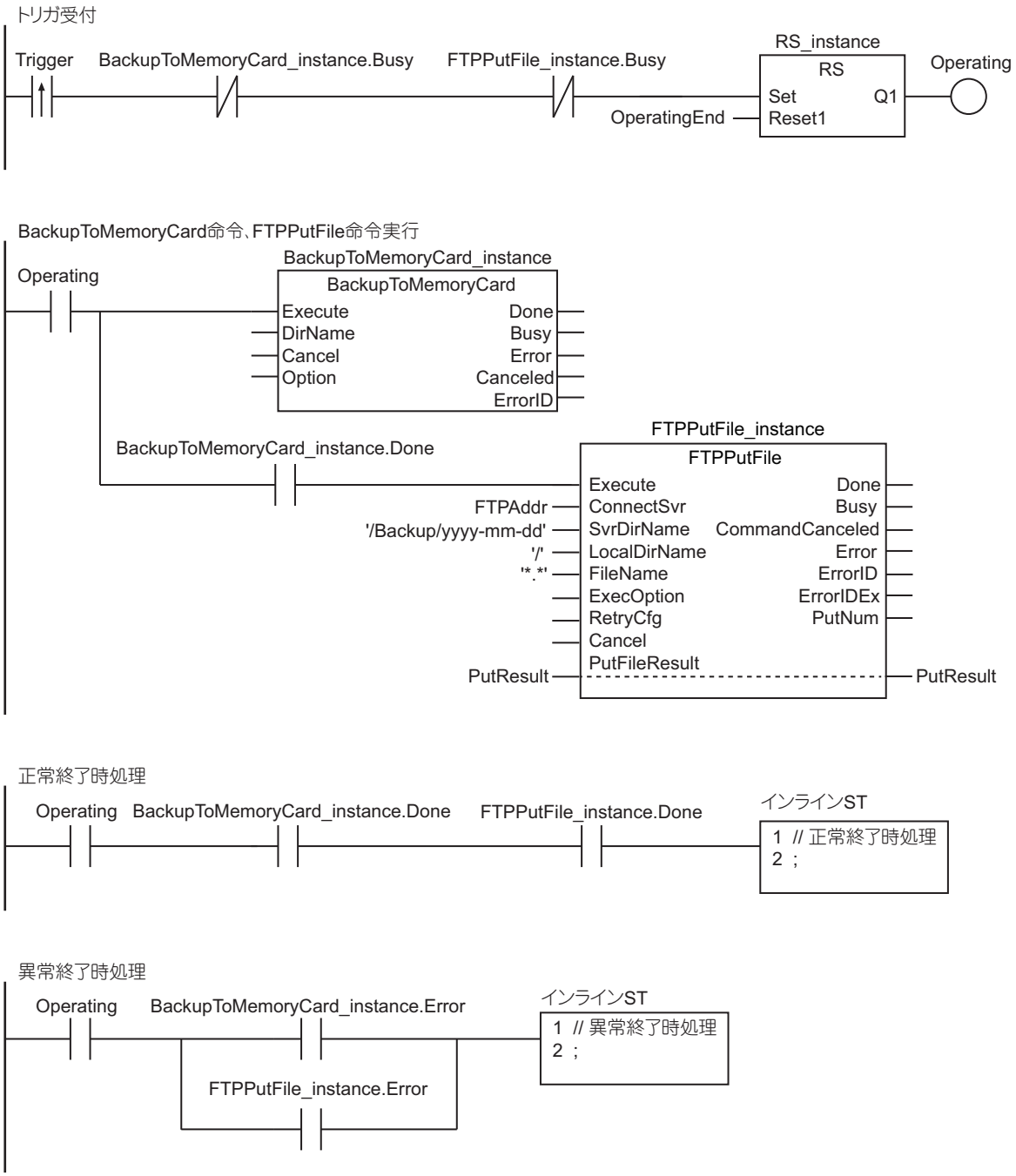
3 **すべての処理が正常に終了した場合は正常終了時処理を実行します。**
異常が発生した場合は、異常終了時処理を実行します。

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
	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







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

内部変数	名称	データ型	初期値	コメント
	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
	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。

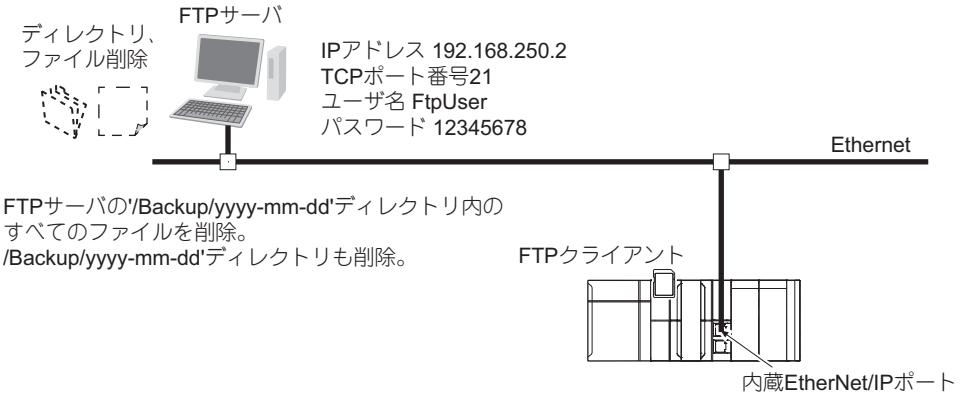
命令	名称	FB/FUN	グラフィック表現	ST 表現
FTPRemoveFile	FTP サーバのファイル削除	FB	FTPRemoveFile_instance FTPRemoveFile Execute / ConnectSvr / SvrDirName / FileName / ExecOption / RetryCfg / Cancel / RemoveFileResult Done / Busy / CommandCanceled / Error / ErrorID / ErrorIDEx / RemoveNum	FTPRemoveFile_instance(Execute, ConnectSvr, SvrDirName, FileName, ExecOption, RetryCfg, Cancel, RemoveFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, RemoveNum);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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。

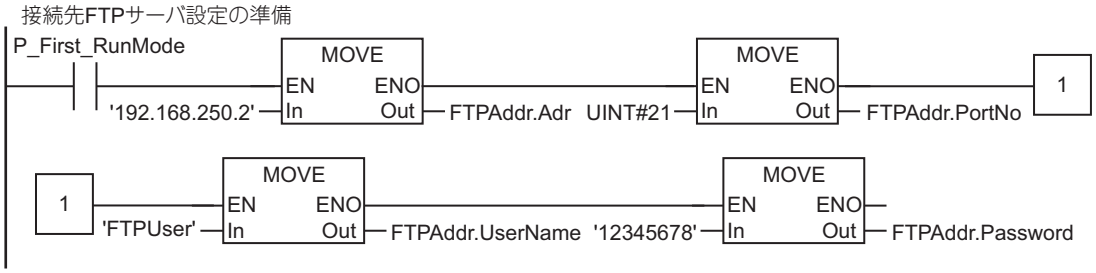
1 FTPRemoveFile 命令を使用して、FTP サーバの'/Backup/yyyy-mm-dd' ディレクトリ内のすべてのファイルを削除します。削除ファイル名は、ワイルドカード指定で'*.*'とします。

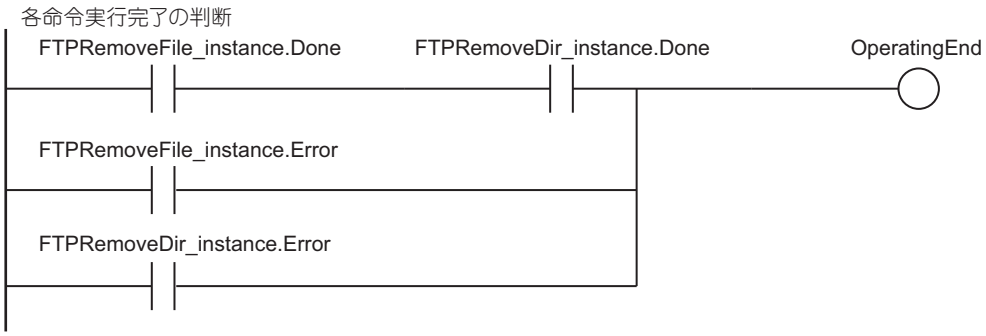
2 **FTPRemoveDir 命令を使用して、FTP サーバの'/Backup/yyyy-mm-dd' ディレクトリを削除します。**

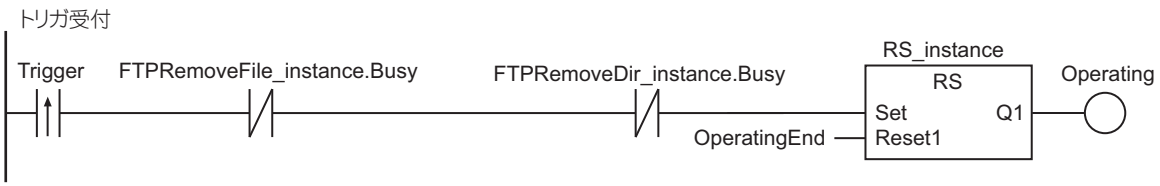
3 **すべての処理が正常に終了した場合は正常終了時処理を実行します。異常が発生した場合は異常終了時処理を実行します。**

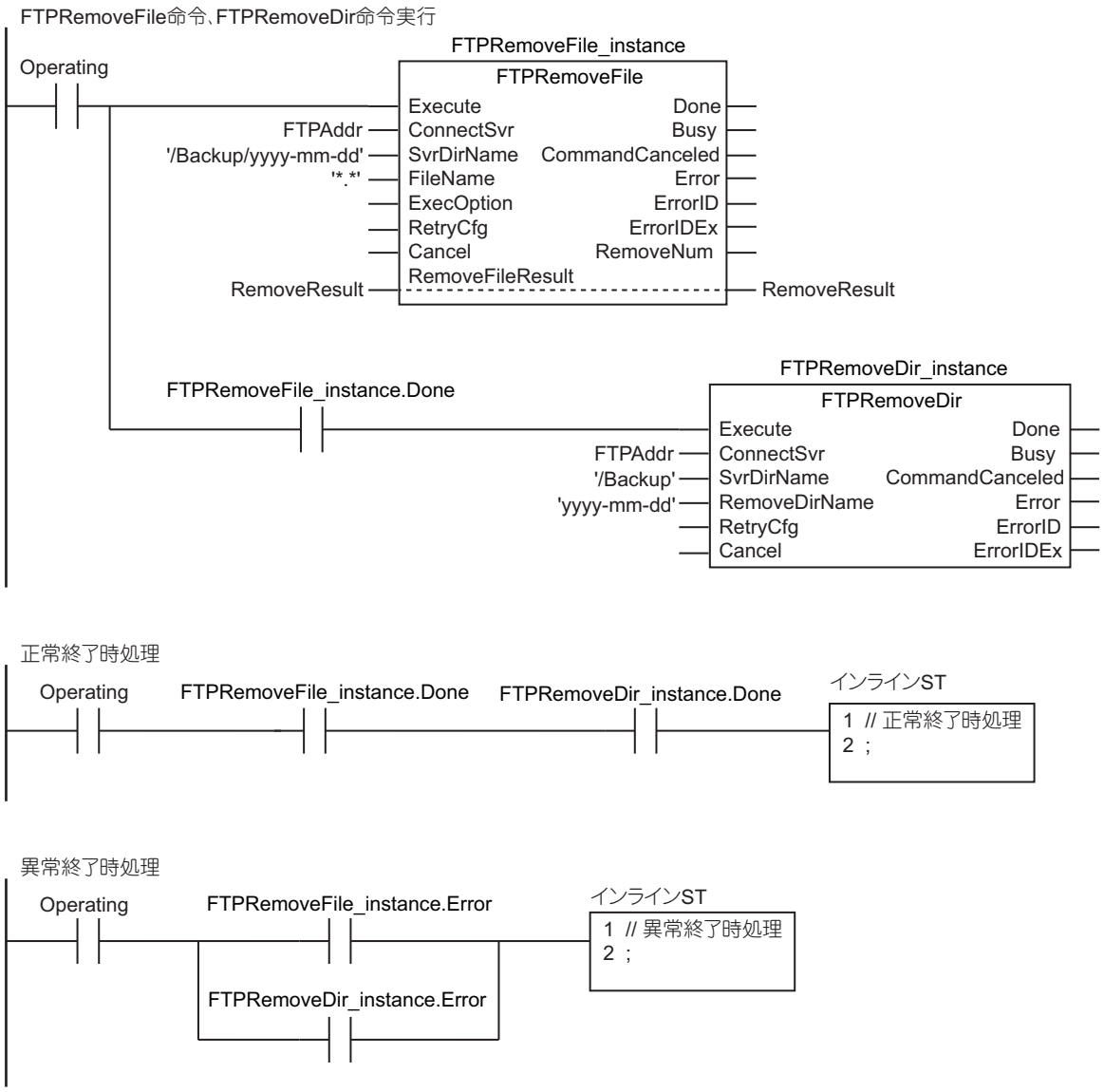
LD

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









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

内部変数	名称	データ型	初期値	コメント
	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;

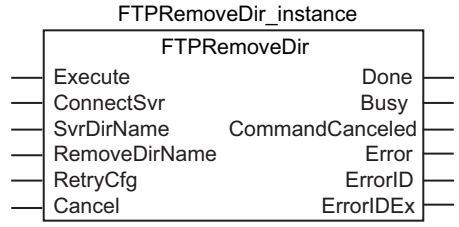
// トリガ受付
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (FTPRemoveFile_instance.Busy = FALSE) AND
      (FTPRemoveDir_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   FTPRemoveFile_instance( // インスタンス初期化
      Execute          := FALSE,
      ConnectSvr       := FTPAddr,
      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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

サンプルプログラム

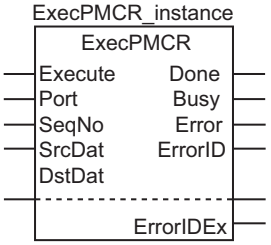
FTPRemoveFile 命令の、「サンプルプログラム (P.1-401)」を参照してください。

シリアル通信命令

命令	名称	ページ
ExecPMCR	プロトコルマクロ	P.1-408
SerialSend	シリアルコミュニケーションユニット シリアルポート出力	P.1-415
SerialRcv / SerialRcvNoClear	シリアルコミュニケーションユニット シリアルポート入力／受信バッファ非消去	P.1-419
SendCmd	コマンド送信	P.1-424
NX_SerialSend	無手順データの送信	P.1-430
NX_SerialRcv	無手順データの受信	P.1-435
NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	P.1-440
NX_ModbusRtuRead	Modbus RTU Read コマンド送信	P.1-443
NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	P.1-446
NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	P.1-449
NX_SerialSigRead	シリアル制御信号の読出	P.1-453
NX_SerialStatusRead	シリアルポートステータスの読出	P.1-454
NX_SerialBufClear	バッファクリア	P.1-455

ExecPMCR

シリアルコミュニケーションユニットに登録された送受信シーケンス（プロトコルデータ）の実行を依頼します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
ExecPMCR	プロトコルマクロ	FB		ExecPMCR_instance(Execute, Port, SeqNo, SrcDat, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx);

使用上の注意

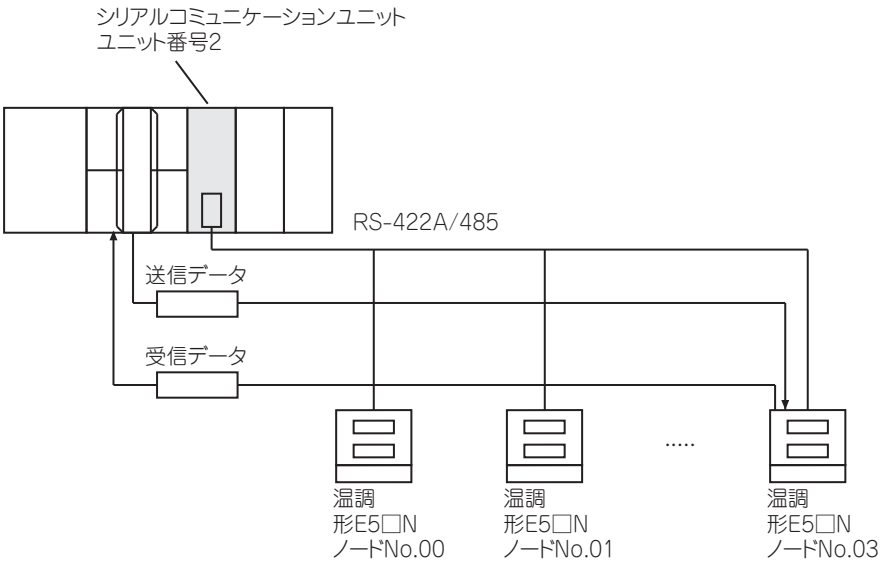
本命令は、NX シリーズ CPU ユニットでは使用できません。

サンプルプログラム

CJ シリーズのシリアルコミュニケーションユニットを使って、オムロン製温調器とのデータ送受信を行います。送受信プロトコルは、CompoWay/F 親局のシーケンス No.610（変数エリア読出）で、温調器の現在値を読み出します。
コントローラからは、送信データ配列 SendData[]の内容を、送信します。
温調器から受信したデータは、受信データ配列 RecvData[]に格納します。

主な通信仕様を、以下に示します。

項目	内容
使用ユニット	シリアルコミュニケーションユニット
ユニット番号	2
ポート番号	1(RS422/485)
送受信シーケンス番号	610（変数エリア読出）
相手先ノード No.	3
読出内容	現在値



シーケンス No.610（変数エリア読出）の、送受信データの割付内容は、以下のとおりです。

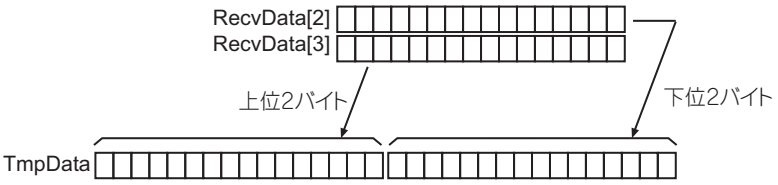
送信データ　WORD型配列

SendData[0]	送信データチャネル数	
SendData[1]	空き	ノードNo.
SendData[2]	変数種別	
SendData[3]	読出開始アドレス	
SendData[4]	要素数	

受信データ　WORD型配列

RecvData[0]	受信データチャネル数
RecvData[1]	レスポンスコード
RecvData[2]	受信データ
RecvData[3]	

受信に成功したら、温調器の現在値の下位 2 バイト(RecvData[2])と、上位 2 バイト(RecvData[3])を組み合わせて、「TmpData」に代入します。



送信データ SendData[]と受信データ RecvData[]

この例での送信データ SendData[]と受信データ RecvData[]の内容は、以下のとおりとします。

● 送信データ　WORD 型配列

変数	項目	この例の内容	値
SendData[0]	送信データチャネル数	SendData[0] ～ SendData[4]の 5 チャネル（ワード）を送信します。	WORD#16#0005
SendData[1]	ノード No.	ノード No.3 とします。	WORD#16#0003
SendData[2]	変数種別+読出開始アドレスの上位 1 バイト	現在値読出は、変数種別が BYTE#16#C0、読出開始アドレスが WORD#16#00 です。	WORD#16#C000
SendData[3]	読出開始アドレスの下位 1 バイト+固定値 BYTE#16#00		WORD#16#0000
SendData[4]	要素数	要素数は 1 とします。	WORD#16#0001

● 受信データ　WORD 型配列

変数	項目	この例の内容	値
RecvData[0]	受信データチャネル数	RecvData[0] ～ RecvData[3]の4チャネル（ワード）を受信します。	WORD#16#0004
RecvData[1]	レスポンスコード	正常終了なら WORD#16#0000 が返ります。	
RecvData[2]	受信データ	温調器の現在値の、下位 2 バイトが返ります。	
RecvData[3]		温調器の現在値の、上位 2 バイトが返ります。	

グローバル変数の定義

● グローバル変数

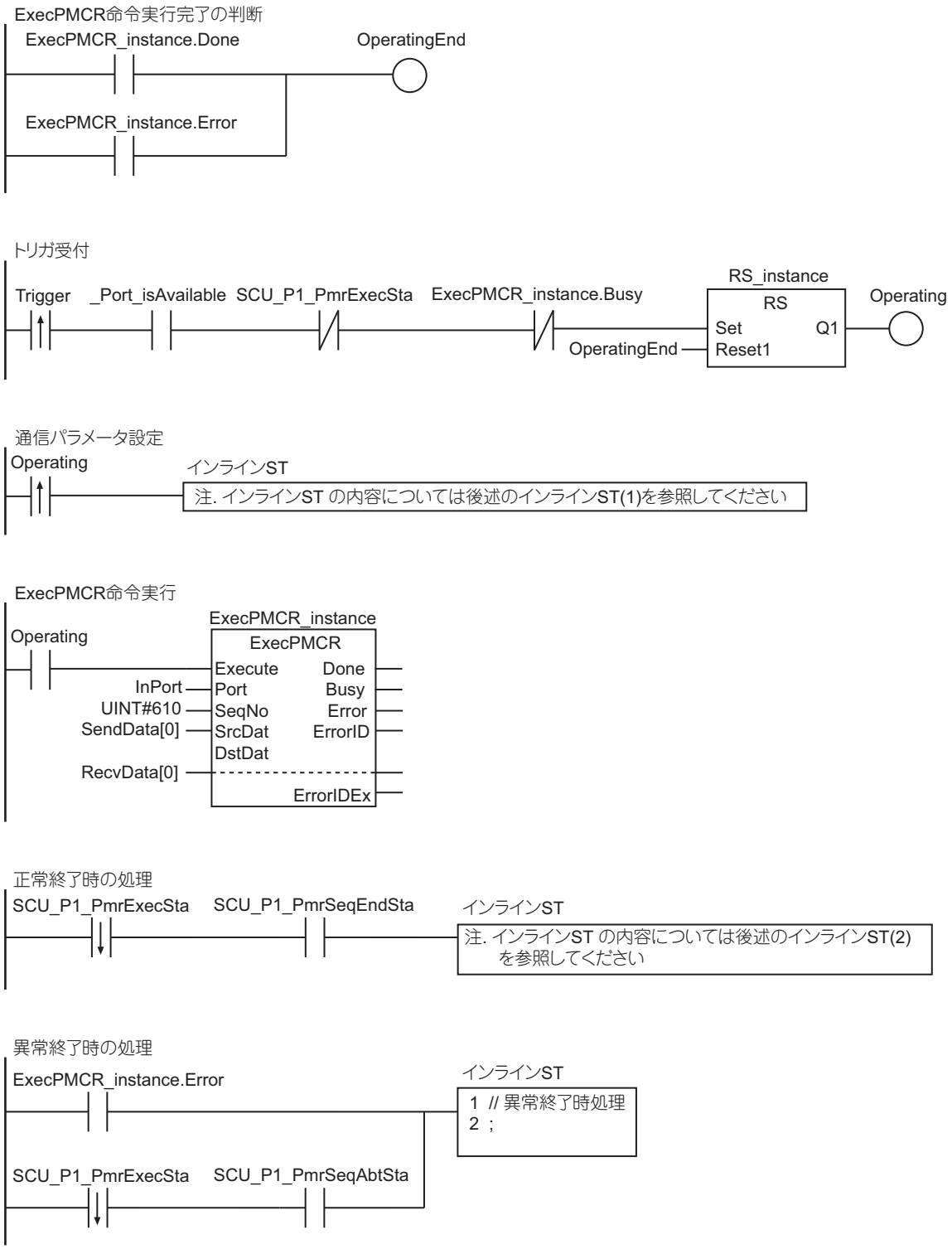
名称	データ型	割付先 *1	コメント
SCU_P1_PmrSeqEndSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqEndSta	シーケンス End 終了フラグ
SCU_P1_PmrSeqAbtSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqAbtSta	シーケンス Abort 終了フラグ
SCU_P1_PmrExecSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrExecSta	プロトコルマクロ実行中フラグ

*1.　シリアルコミュニケーションユニットをラック番号 0 のスロット番号 0 に装着した場合の割付先です。

LD

内部変数	名称	データ型	初期値	割付先	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
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)		☐	ポート指定
	SendData	ARRAY[0..4] OF WORD	[5(16#0)]		☐	送信データ
	RecvData	ARRAY[0..3] OF WORD	[4(16#0)]	%D200	☑	受信データ
	TmpData	DINT	0		☐	現在値
	RS_instance	RS			☐	
	ExecPMCR_instance	ExecPMCR			☐	

外部変数	名称	データ型	コメント
	SCU_P1_PmrSeqEndSta	BOOL	シーケンス End 終了フラグ
	SCU_P1_PmrSeqAbtSta	BOOL	シーケンス Abort 終了フラグ
	SCU_P1_PmrExecSta	BOOL	プロトコルマクロ実行中フラグ
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ



● インライン ST(1) の内容

```
InPort.UnitNo          :=_CBU_No02;    // シリアルコミュニケーションユニット　ユニット番号 2
InPort.PhysicPortNo    :=USINT#1;      // ポート番号 1
SendData[0]            :=WORD#16#0005;
SendData[1]            :=WORD#16#0003;
SendData[2]            :=WORD#16#C000;
SendData[3]            :=WORD#16#0000;
```

```
SendData[4]             :=WORD#16#0001;
RecvData[0]             :=WORD#16#0004;
```

● インライン ST(2) の内容

```
// 正常終了時処理
TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(
        RecvData[3]), 16) OR WORD_TO_DWORD(RecvData[2]) );
```

ST

内部変数	名称	データ型	初期値	割付先	保持	コメント
	State	INT	0		☐	遷移状態
	Trigger	BOOL	FALSE		☐	実行条件
	LastTrigger	BOOL	FALSE		☐	前タスク周期の Trigger の値
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)		☐	ポート指定
	SendData	ARRAY[0..4] OF WORD	[5(16#0)]		☐	送信データ
	RecvData	ARRAY[0..3] OF WORD	[4(16#0)]	%D200	☑	受信データ
	End_ExecPMCR	BOOL	FALSE		☐	ExecPMCR 命令実行終了
	TmpData	DINT	0		☐	現在値
	RS_instance	RS			☐	
	ExecPMCR_instance	ExecPMCR			☐	
	F_TRIG_instance	F_TRIG			☐	

外部変数	名称	データ型	コメント
	SCU_P1_PmrSeqEndSta	BOOL	シーケンス End 終了フラグ
	SCU_P1_PmrSeqAbtSta	BOOL	シーケンス Abort 終了フラグ
	SCU_P1_PmrExecSta	BOOL	プロトコルマクロ実行中フラグ
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ

```
// トリガ受付
IF (State=INT#0) THEN
    IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE) AND (SCU_P1_PmrExecSta<
>TRUE)
        AND (ExecPMCR_instance.Busy<>TRUE) ) THEN
      State:=INT#1;
    END_IF;
END_IF;
LastTrigger:=Trigger;
```

```
// 通信パラメータ設定と ExecPMCR 命令初期化
IF (State=INT#1) THEN
    InPort.UnitNo          :=_CBU_No02;    // シリアルコミュニケーションユニット　ユニット番号 2
    InPort.PhysicPortNo    :=USINT#1;      // ポート番号 1
    SendData[0]            :=WORD#16#0005;
    SendData[1]            :=WORD#16#0003;
    SendData[2]            :=WORD#16#C000;
    SendData[3]            :=WORD#16#0000;
    SendData[4]            :=WORD#16#0001;
    RecvData[0]            :=WORD#16#0004;
    ExecPMCR_instance(
        Execute :=FALSE,          // ExecPMCR 命令初期化
        SrcDat  :=SendData[0],    // ダミー
        DstDat  :=RecvData[0]);
     State:=INT#2;
END_IF;
// ExecPMCR 命令実行
IF (State=INT#2) THEN
    ExecPMCR_instance(
    Execute :=TRUE,
    Port    :=InPort,
    SeqNo   :=UINT#610,
    SrcDat  :=SendData[0],
    DstDat  :=RecvData[0]);

    F_TRIG_instance(SCU_P1_PmrExecSta, End_ExecPMCR);

    IF (End_ExecPMCR=TRUE) THEN
        End_ExecPMCR:=FALSE;
        State:=INT#3;
    END_IF;

    IF (ExecPMCR_instance.Error=TRUE) THEN
        State:=INT#5;
    END_IF;
END_IF;

// ExecPMCR 命令実行終了確認
IF (State=INT#3) THEN
    IF (SCU_P1_PmrSeqEndSta=TRUE) THEN
        State:=INT#4;
    END_IF;
    IF (SCU_P1_PmrSeqAbtSta=TRUE) THEN
        State:=INT#5;
    END_IF;
END_IF;

IF (State=INT#4) THEN
    // 正常終了時処理
    TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(RecvData[3]), 16)
```

```
                OR WORD_TO_DWORD(RecvData[2]));
    State:=INT#0;
END_IF;

IF (State=INT#5) THEN
    // 異常終了時処理
    State:=INT#0;
END_IF;
```

SerialSend

シリアルコミュニケーション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。

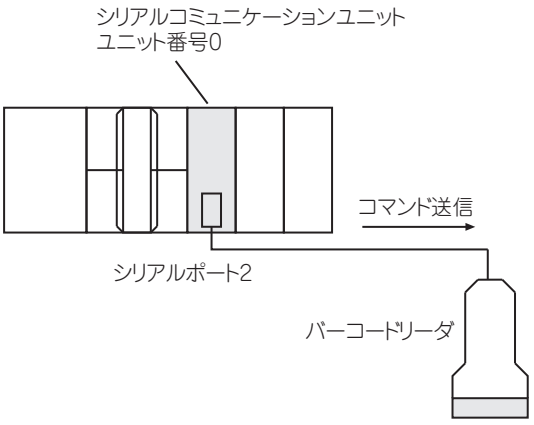
命令	名称	FB/FUN	グラフィック表現	ST 表現
SerialSend	シリアルコミュニケーションユニット シリアルポート出力	FB	SerialSend_instance SerialSend Execute / Done Port / Busy SrcDat / Error SendSize / ErrorID ErrorIDEx	SerialSend_instance(Execute, Port, SrcDat, SendSize, Done, Busy, Error, ErrorID, ErrorIDEx);

使用上の注意

本命令は、NX シリーズ CPU ユニットでは使用できません。

サンプルプログラム

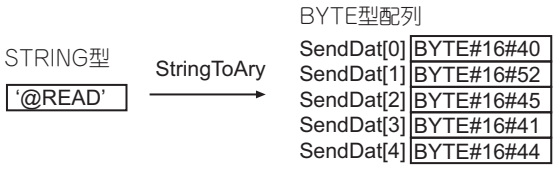
CJ シリーズのシリアルコミュニケーションユニット（ユニット番号 0、デバイス名'Barcode'）の、シリアルポート 2 に接続したバーコードリーダに、無手順コマンドを送信します。送信するコマンドは、シーン番号取得コマンド'@READ'です。
送信データは、配列変数 SendDat[]の中身です。スタートコードはなし、エンドコードは 16#OD(CR)とします。



シリアルコミュニケーションユニットの設定は、以下のとおりとします。

項目名	設定値
ポート 2: 任意設定の有無	任意設定
ポート 2: シリアル通信モード	無手順
ポート 2: データ長	8 ビット
ポート 2: ストップビット	1 ビット
ポート 2: パリティ	なし
ポート 2: 伝送速度	38400bps
ポート 2: 無手順エンドコード	D
ポート 2: 無手順スタートコード有無	なし
ポート 2: 無手順エンドコード有無	あり（エンドコードを任意設定）

SendDat[]には、文字列'@READ'を 1 文字ずつに分解して、その文字コードを各配列要素に格納しておきます。したがって、SendDat[0]には BYTE#16#40(@)、SendData[1]には BYTE#16#52(R)を格納します。その処理には StringToAry 命令を使います。



グローバル変数の定義

● グローバル変数

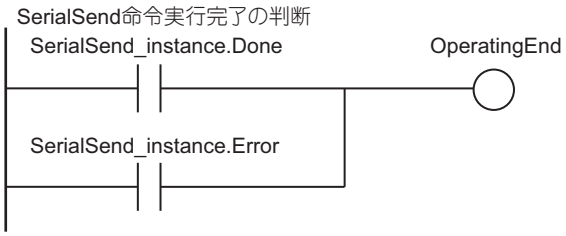
名称	データ型	割付先 *1	コメント
Barcode_P2_NopSerialSendExecSta	BOOL	IOBus://rack#0/slot#0/P2_NopSta/P2_NopSerialSendExecSta	SerialSend 命令実行中フラグ

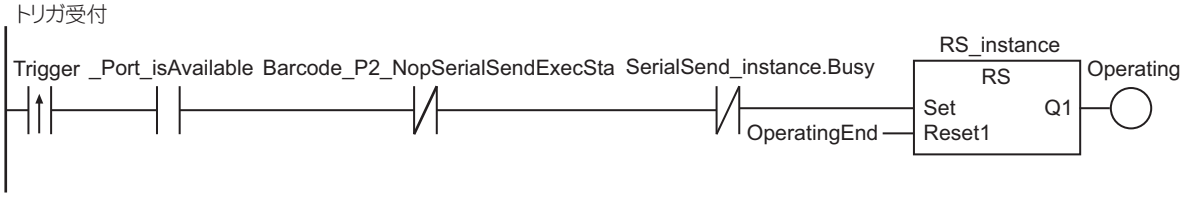
*1. シリアルコミュニケーションユニットをラック番号 0 のスロット番号 0 に装着した場合の割付先です。

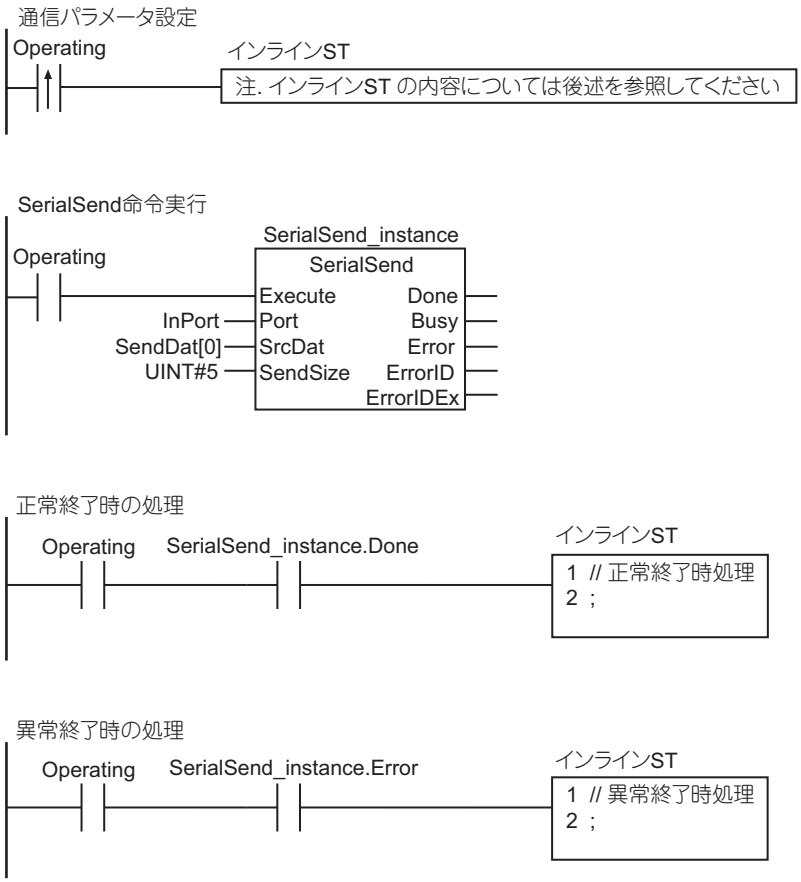
LD

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
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	ポート指定
	SendDat	ARRAY[0..4] OF BYTE	[5(16#0)]	送信データ
	RS_instance	RS		
	SerialSend_instance	SerialSend		

外部変数	名称	データ型	コメント
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ
	Barcode_P2_NopSerialSendExecSta	BOOL	SerialSend 命令実行中フラグ







● インライン ST の内容

```
StringToAry(In:='@READ', AryOut:=SendDat[0]);    // SendDat[]準備
InPort.UnitNo       :=_CBU_No00;                 // シリアルコミュニケーションユニット　ユニット番号 0
InPort.PhysicPortNo:=USINT#2;                    // シリアルポート 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
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	ポート指定
	SendDat	ARRAY[0..4] OF BYTE	[5(16#0)]	送信データ
	SerialSend_instance	SerialSend		

外部変数	名称	データ型	コメント
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ
	Barcode_P2_NopSerialSendExecSta	BOOL	SerialSend 命令実行中フラグ

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
      AND (Barcode_P2_NopSerialSendExecSta=FALSE) AND (SerialSend_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// 通信パラメータ設定と SerialSend 命令初期化
IF (OperatingStart=TRUE) THEN
    SerialSend_instance(
        Execute:=FALSE,
        SrcDat :=SendDat[0]);
    StringToAry(In:='@READ', AryOut:=SendDat[0]);
    InPort.UnitNo      :=_CBU_No00;          // シリアルコミュニケーションユニット　ユニット番号 0
    InPort.PhysicPortNo:=USINT#2;            // シリアルポート 2
    OperatingStart     :=FALSE;
END_IF;

// SerialSend 命令実行
IF (Operating=TRUE) THEN
    SerialSend_instance(
        Execute :=TRUE,
        Port    :=InPort,       // ポート指定
        SrcDat  :=SendDat[0],   // 送信データ
        SendSize:=UINT#5);      // 送信データサイズ

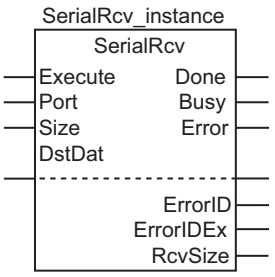
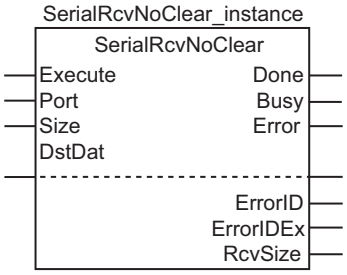
    IF (SerialSend_instance.Done=TRUE) THEN
        // 正常終了時処理
        Operating:=FALSE;
    END_IF;

    IF (SerialSend_instance.Error=TRUE) THEN
        // 異常終了時処理
        Operating:=FALSE;
   END_IF;
END_IF;
```

SerialRcv / SerialRcvNoClear

シリアルコミュニケーションユニットのシリアルポートから、無手順の受信データを読み出します。

SerialRcv　　：読出後、受信バッファはオールクリアされます。
SerialRcvNoClear　：読出後、受信バッファはオールクリアされません。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SerialRcv	シリアルコミュニケーションユニット シリアルポート入力	FB		SerialRcv_instance(Execute, Port, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);
SerialRcvNoClear	シリアルコミュニケーションユニット シリアルポート入力受信バッファ非消去	FB		SerialRcvNoClear_instance(Execute, Port, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

使用上の注意

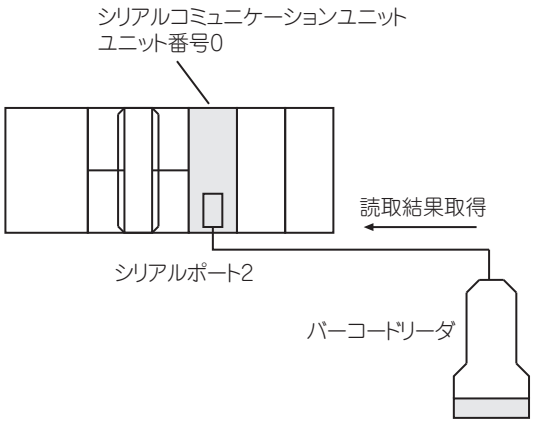
本命令は、NX シリーズ CPU ユニットでは使用できません。

バージョン関連情報

SerialRcvNoClear 命令は、CPU ユニット Ver.1.03 以降、Sysmac Studio Ver.1.04 以降,シリアルコミュニケーションユニット Ver.2.1 以降の組み合わせで、使用可能です。

サンプルプログラム

CJ シリーズのシリアルコミュニケーションユニット（ユニット番号 0、デバイス名'Barcode'）の、シリアルポート 2 に接続したバーコードリーダの読取結果を取得します。
受信データは、配列変数 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。



シリアルコミュニケーションユニットの設定は、以下のとおりとします。

項目名	設定値
ポート 2: 任意設定の有無	任意設定
ポート 2: シリアル通信モード	無手順
ポート 2: データ長	8 ビット
ポート 2: ストップビット	1 ビット
ポート 2: パリティ	なし
ポート 2: 伝送速度	38400bps
ポート 2: 無手順エンドコード	D
ポート 2: 無手順スタートコード有無	なし
ポート 2: 無手順エンドコード有無	あり（エンドコードを任意設定）

RecvDat[]に格納されるデータは、バーコードの数値を 1 文字ずつ文字コードで表現したビット列です。RecvDat[]の配列の 1 要素が、バーコードの 1 文字に対応します。それを、まず AryToString 命令を使って、文字列「RecvStringDat」に変換します。さらにそれを STRING_TO_ULINT 命令を使って、ULINT 型整数「Code」に変換します。



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
Barcode_P2_NopRcvCompleteSta	BOOL	IOBus://rack#0/slot#0/P2_NopSta/P2_NopRcvCompleteSta	受信完了フラグ

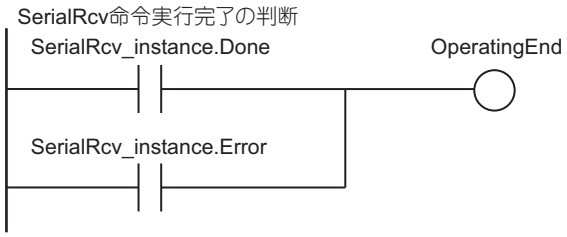
*1. シリアルコミュニケーションユニットをラック番号 0 のスロット番号 0 に装着した場合の割付先です。

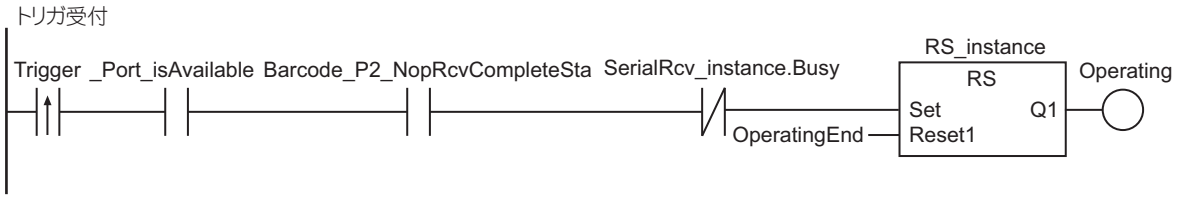
LD

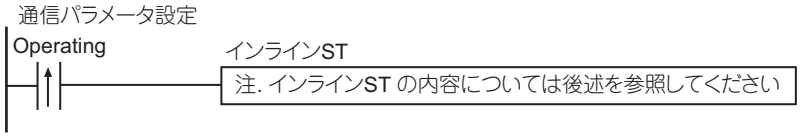
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件

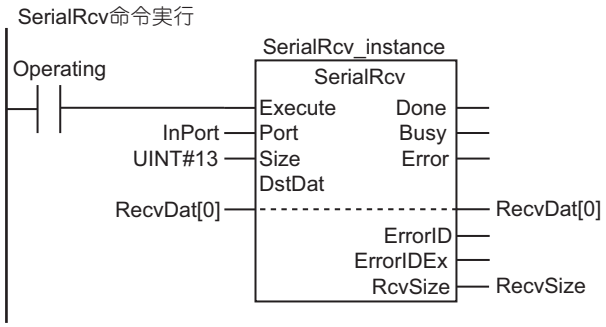
内部変数	名称	データ型	初期値	コメント
	Operating	BOOL	FALSE	処理中
	InPort	_sPORT	(UnitNo:=_CBU_No00 , PhysicPortNo:=0)	ポート指定
	RecvDat	ARRAY[0..12] OF BYTE	[13(16#0)]	受信データ
	RecvSize	UINT	0	受信データサイズ
	RecvStringDat	STRING[255]	''	バーコード（文字列）
	Code	ULINT	0	バーコード（整数）
	RS_instance	RS		
	SerialRcv_instance	SerialRcv		

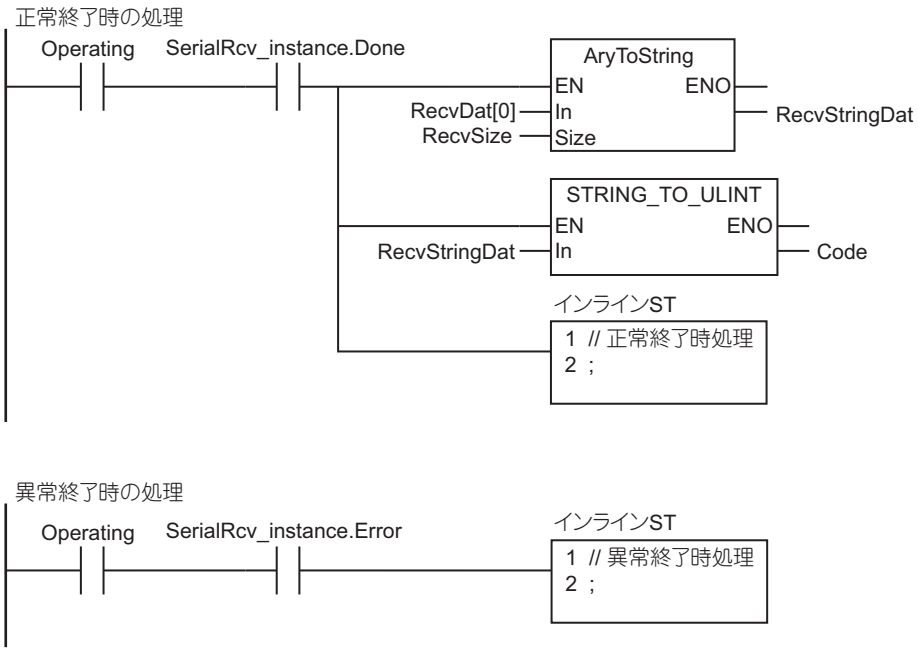
外部変数	名称	データ型	コメント
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ
	Barcode_P2_NopRcvCompleteSta	BOOL	受信完了フラグ











● インライン ST の内容

```
InPort.UnitNo       :=_CBU_No00;    // シリアルコミュニケーションユニット　ユニット番号 0
InPort.PhysicPortNo:=USINT#2;       // シリアルポート 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
	InPort	_sPORT	(UnitNo:=_CBU_No00 , PhysicPortNo:=0)	ポート指定
	RecvDat	ARRAY[0..12] OF BYTE	[13(16#0)]	受信データ
	RecvSize	UINT	0	受信データサイズ
	RecvStringDat	STRING[255]	''	バーコード（文字列）
	Code	ULINT	0	バーコード(整数)
	SerialRcv_instance	SerialRcv		

外部変数	名称	データ型	コメント
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ
	Barcode_P2_NopRcvCompleteSta	BOOL	受信完了フラグ

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
      AND (Barcode_P2_NopRcvCompleteSta=TRUE) AND (SerialRcv_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
```

```
END_IF;
LastTrigger:=Trigger;

// 通信パラメータ設定と SerialRcv 命令初期化
IF (OperatingStart=TRUE) THEN
    SerialRcv_instance(
        Execute:=FALSE,       // インスタンス初期化
        Port    :=InPort,     // ポート指定
        Size    :=UINT#13,    // 受信データサイズ
        DstDat :=RecvDat[0], // 受信データ
        RcvSize =>RecvSize); // 実際に受信したデータサイズ
    InPort.UnitNo       :=_CBU_No00; // シリアルコミュニケーションユニット　ユニット番号 0
    InPort.PhysicPortNo:=USINT#2;   // シリアルポート 2
    OperatingStart      :=FALSE;
END_IF;

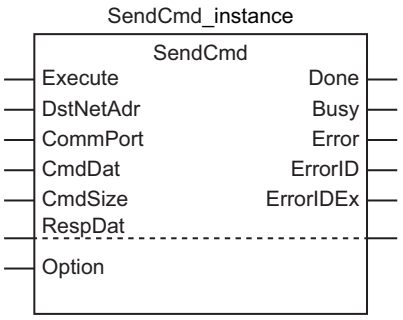
// SerialRcv 命令実行
IF (Operating=TRUE) THEN
    SerialRcv_instance(
        Execute:=TRUE,
        Port    :=InPort,
        Size    :=UINT#13,
        DstDat :=RecvDat[0],
        RcvSize =>RecvSize);

    IF (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(RecvStringDat);              // 文字列を整数に変換
        Operating     :=FALSE;
    END_IF;
    IF (SerialRcv_instance.Error=TRUE) THEN
        // 異常終了時処理
        Operating:=FALSE;
    END_IF;
END_IF;
```

SendCmd

CJ シリーズユニットに対してコマンドを送信します。
- シリアルコミュニケーションユニットに対して、シリアルゲートウェイ機能でコマンドを送信。
- DeviceNet ユニットや CompoNet マスタユニットに対して、Explicit コマンドを送信。
- EtherNet/IP ユニット（形 CJ1W-EIP21S）に対して、ソケットサービス機能のコマンドを送信。

ソケットサービス機能の詳細については、『CJ シリーズ EtherNet/IP ユニット ユーザーズマニュアル NJ シリーズ接続編(SBCD-355)』を参照してください。

命令	名称	FB/FUN	グラフィック表現	ST 表現
SendCmd	コマンド送信	FB		SendCmd_instance(Execute, DstNetAdr, CommPort, CmdDat, CmdSize, RespDat, Option, Done, Busy, Error, ErrorID, ErrorIDEx);

使用上の注意

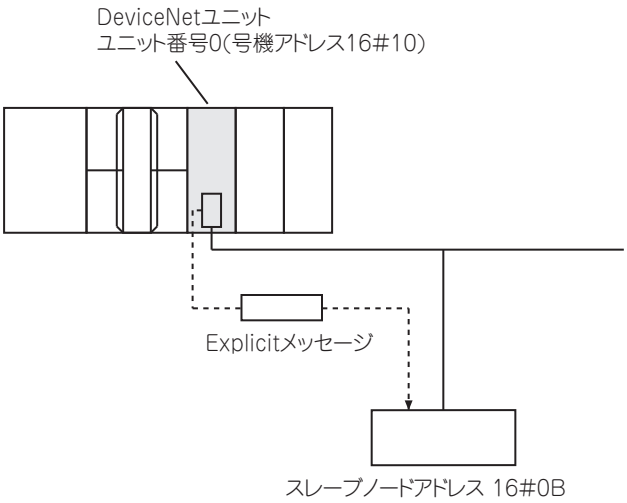
本命令は、NX シリーズ CPU ユニットでは使用できません。

サンプルプログラム

DeviceNet ユニット経由で、Explicit メッセージを発行します。号機アドレス 16#10 の DeviceNet を経由して、ノードアドレス 16#0B のスレーブから、ベンダ ID を読み出します。

主な通信仕様を、以下に示します。

項目	内容
DeviceNet ユニットの号機アドレス	16#10
スレーブノードアドレス	16#0B
サービスコード	16#0E
クラス ID	1
インスタンス ID	1
アトリビュート ID	1
タイムアウト時間	2.0s
再送回数	2



コマンド配列 SendDat[]とレスポンス格納配列 RecvDat[]

コマンド配列 SendDat[]およびレスポンス格納配列 RecvDat[]の内容は、以下のとおりです。

● コマンド配列　BYTE 型配列

配列要素	項目	この例の内容	値
SendDat[0]	コマンドコード	Explicit メッセージ送信のコマンドコードは16#2801 です。	BYTE#16#28
SendDat[1]			BYTE#16#01
SendDat[2]	スレーブノードアドレス	ノードアドレス 16#0B とします。	BYTE#16#0B
SendDat[3]	サービスコード	指定したアトリビュートの値読出(Get Arrtibute Single)のサービスコードは 16#0E です。	BYTE#16#0E
SendDat[4]	クラス ID	Identity オブジェクトのクラス ID は、16#0001 です。	BYTE#16#00
SendDat[5]			BYTE#16#01
SendDat[6]	インスタンス ID	−	BYTE#16#00
SendDat[7]			BYTE#16#01
SendDat[8]	アトリビュート ID	ベンダ ID のアトリビュート ID(Vendor ID)は、16#01 です。	BYTE#16#01

● レスポンス格納配列　BYTE 型配列

配列要素	項目	この例の内容
RecvDat[0]	コマンドコード	Explicit メッセージ送信のコマンドコードは 16#2801 です。
RecvDat[1]		
RecvDat[2]	終了コード	正常終了の場合、16#0000 です。
RecvDat[3]		
RecvDat[4]	スレーブノードアドレス以降の受信バイト数	4 バイトです。
RecvDat[5]		
RecvDat[6]	スレーブノードアドレス	正常終了の場合、ノードアドレス 16#0B となります。
RecvDat[7]	サービスコード	正常終了のサービスコードは 16#8E です。
RecvDat[8]	ベンダ ID	スレーブのベンダ ID です。
RecvDat[9]		

グローバル変数の定義

● グローバル変数

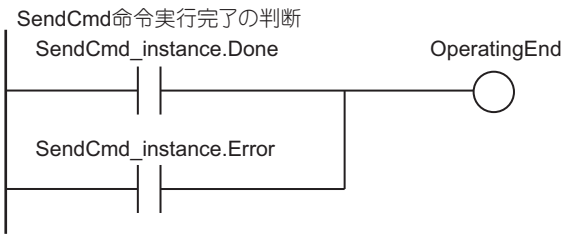
名称	データ型	割付先 *1	コメント
DeviceNet_OnlineSta	BOOL	IOBus://rack#0/slot#0/Unit2Sta/OnlineSta	オンライン

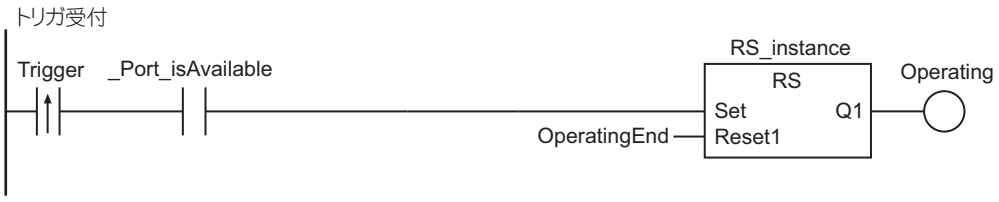
*1. シリアルコミュニケーションユニットをラック番号 0 のスロット番号 0 に装着した場合の割付先です。

LD

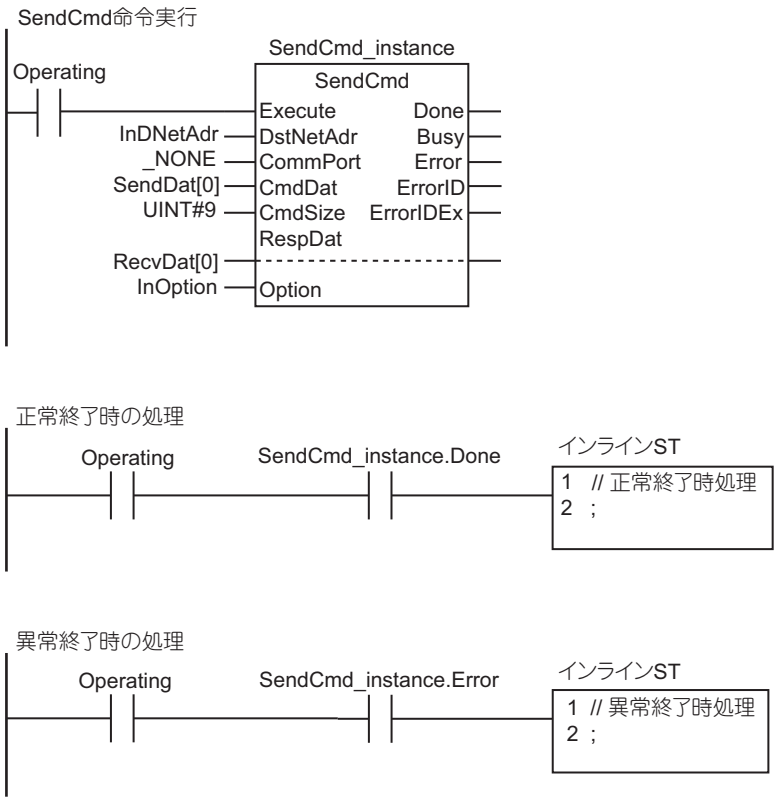
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	InDNetAdr	_sDNET_ADR	(NetNo:=0, NodeNo:=0, UnitNo:=16#0)	送信先ネットワークアドレス
	InOption	_sRESPONSE	(isNonResp:=FALSE, TimeOut:=0, Retry:=0)	レスポンス
	SendDat	ARRAY[0..8] OF BYTE	[9(16#0)]	送信データ
	RecvDat	ARRAY[0..9] OF BYTE	[10(16#0)]	受信データ
	RS_instance	RS		
	SendCmd_instance	SendCmd		

外部変数	名称	データ型	コメント
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ









● インライン ST の内容

```
InDNetAdr.NetNo      :=USINT#0;      // ネットワークアドレス設定
InDNetAdr.NodeNo     :=USINT#0;
InDNetAdr.UnitNo     :=BYTE#16#10;
InOption.isNonResp   :=FALSE;        // レスポンス設定
InOption.TimeOut     :=UINT#20;
InOption.Retry       :=USINT#2;
SendDat[0]           :=BYTE#16#28;   // コマンド配列設定
SendDat[1]           :=BYTE#16#01;
SendDat[2]           :=BYTE#16#0B;
SendDat[3]           :=BYTE#16#0E;
SendDat[4]           :=BYTE#16#00;
SendDat[5]           :=BYTE#16#01;
SendDat[6]           :=BYTE#16#00;
SendDat[7]           :=BYTE#16#01;
SendDat[8]           :=BYTE#16#01;
```

ST

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
	InDNetAdr	_sDNET_ADR	(NetNo:=0, NodeNo:=0, UnitNo:=16#0)	送信先ネットワークアドレス

内部変数	名称	データ型	初期値	コメント
	InOption	_sRESPONSE	(isNonResp:=FALSE, TimeOut:=0, Retry:=0)	レスポンス
	SendDat	ARRAY[0..8] OF BYTE	[9(16#0)]	送信データ
	RecvDat	ARRAY[0..9] OF BYTE	[10(16#0)]	受信データ
	SendCmd_instance	SendCmd		

外部変数	名称	データ型	コメント
	DeviceNet_OnlineSta	BOOL	オンライン
	_Port_isAvailable	BOOL	ネットワーク通信命令実行可フラグ

```
// Trigger の立ち上がり検出
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
      AND (DeviceNet_OnlineSta=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// 通信パラメータ設定と SendCmd 命令初期化
IF (OperatingStart=TRUE) THEN
    SendCmd_instance(
        Execute  :=FALSE,
        DstNetAdr:=InDNetAdr,
        CommPort :=_NONE,
        CmdDat   :=SendDat[0],
        CmdSize  :=UINT#9,
        RespDat  :=RecvDat[0],
        Option   :=InOption);
    InDNetAdr.NetNo      :=USINT#0;      //  ネットワークアドレス設定
    InDNetAdr.NodeNo     :=USINT#0;
    InDNetAdr.UnitNo     :=BYTE#16#10;
    InOption.isNonResp   :=FALSE;        //  レスポンス設定
    InOption.TimeOut     :=UINT#20;
    InOption.Retry       :=USINT#2;
    SendDat[0]           :=BYTE#16#28;   //  コマンド配列設定
    SendDat[1]           :=BYTE#16#01;
    SendDat[2]           :=BYTE#16#0B;
    SendDat[3]           :=BYTE#16#0E;
    SendDat[4]           :=BYTE#16#00;
    SendDat[5]           :=BYTE#16#01;
    SendDat[6]           :=BYTE#16#00;
    SendDat[7]           :=BYTE#16#01;
    SendDat[8]           :=BYTE#16#01;
    OperatingStart       :=FALSE;
END_IF;
// SendCmd 命令実行
IF (Operating=TRUE) THEN
```

```
    SendCmd_instance(
        Execute  :=TRUE,
        DstNetAdr:=InDNetAdr,
        CommPort :=_NONE,
        CmdDat   :=SendDat[0],
        CmdSize  :=UINT#9,
        RespDat  :=RecvDat[0],
        Option   :=InOption);

    IF (SendCmd_instance.Done=TRUE) THEN
        // 正常終了時処理
        Operating:=FALSE;
    END_IF;

    IF (SendCmd_instance.Error=TRUE) THEN
        // 異常終了時処理
        Operating:=FALSE;
    END_IF;
END_IF;
```

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およびオプションボードのシリアルポートから、無手順でデータを送信します。

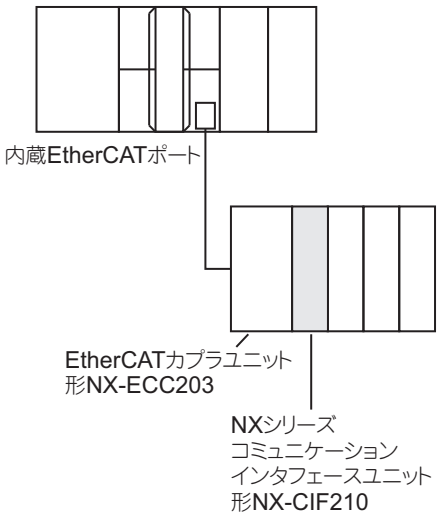
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialSend	無手順データの送信	FB	NX_SerialSend_instance NX_SerialSend Execute / DevicePort / SendDat / SendSize / SendCfg / Option / Abort Done / Busy / CommandAborted / Error / ErrorID	NX_SerialSend_instance(Execute, DevicePort, SendDat, SendSize, SendCfg, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

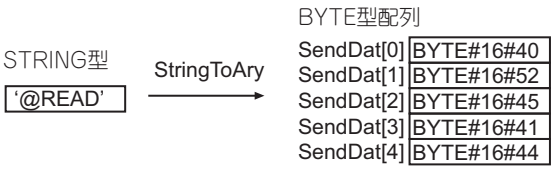
サンプルプログラム

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
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

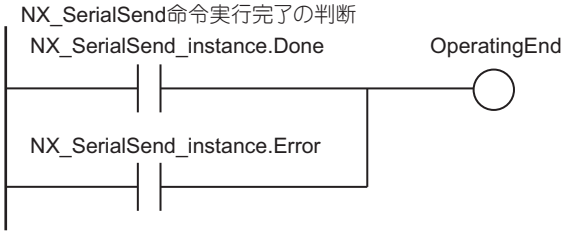
名称	データ型	割付先	コメント
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT:// node#1/NX Unit I/O Data Active Status 125	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

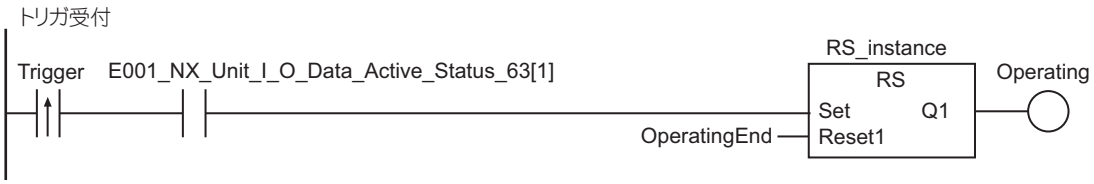
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

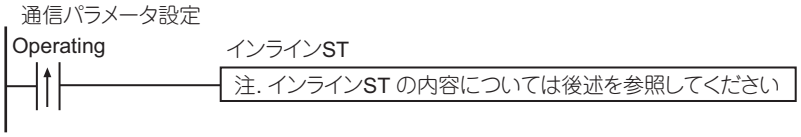
LD

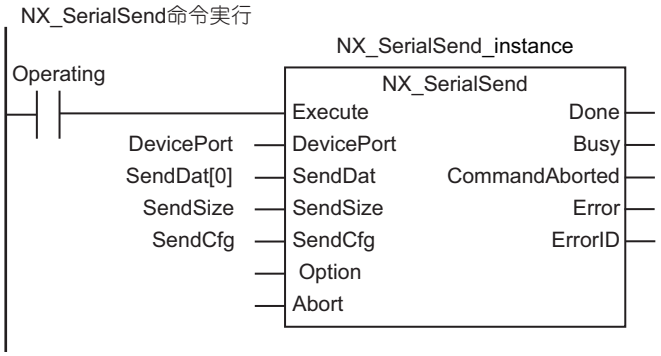
内部変数	名称	データ型	初期値	コメント
	OperationEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	SendDat	ARRAY [0..5] OF BYTE	[6(16#0)]	送信データ
	SendSize	UINT	0	送信データサイズ
	RS_instance	RS		
	NX_SerialSend_instance	NX_SerialSend		
	SendCfg	_sSERIAL_CFG		
	StartTrig	_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)

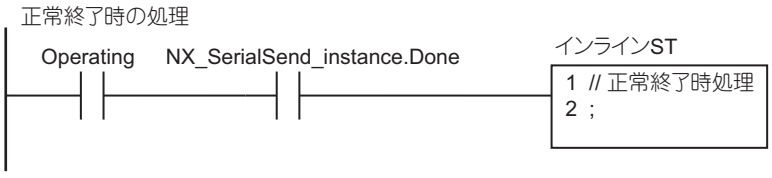
外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数

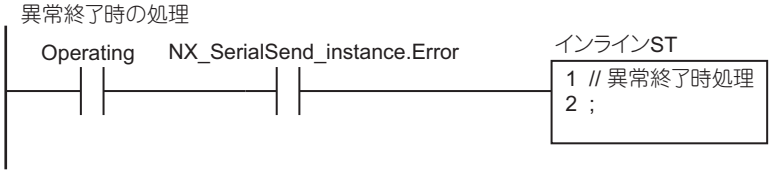












● インライン ST の内容

```
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およびオプションボードのシリアルポートから、無手順でデータを読み出します。

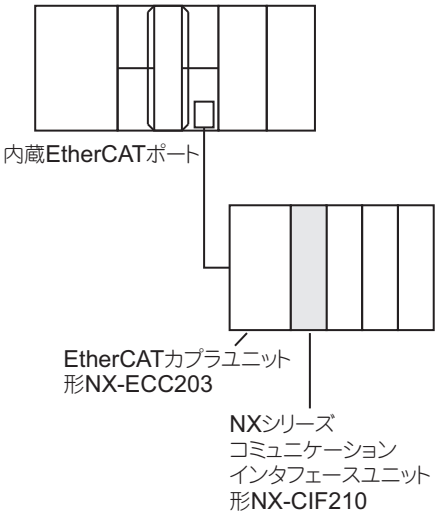
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialRcv	無手順データの受信	FB	NX_SerialRcv_instance NX_SerialRcv Execute / DevicePort / RcvDat / Size / RcvCfg / Option / Abort Done / Busy / RcvDat / CommandAborted / Error / ErrorID / RcvSize	NX_SerialRcv_instance(Execute, DevicePort, RcvDat, Size, RcvCfg, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, RcvSize);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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

項目名	設定値
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

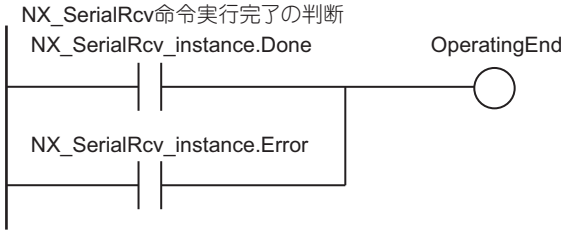
名称	データ型	割付先	コメント
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT:// node#1/NX Unit I/O Data Active Status 125	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

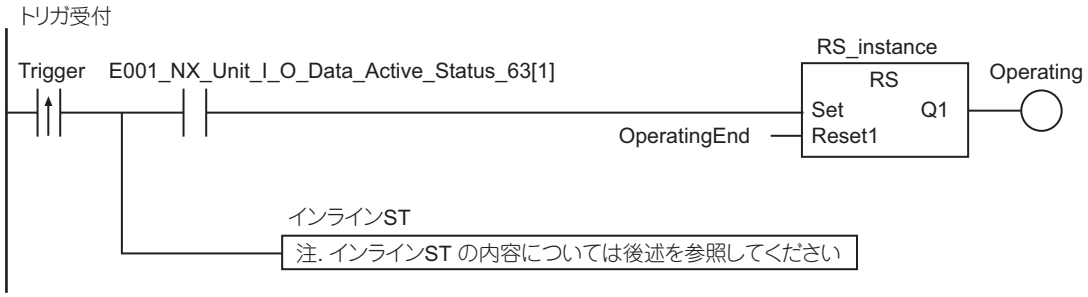
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

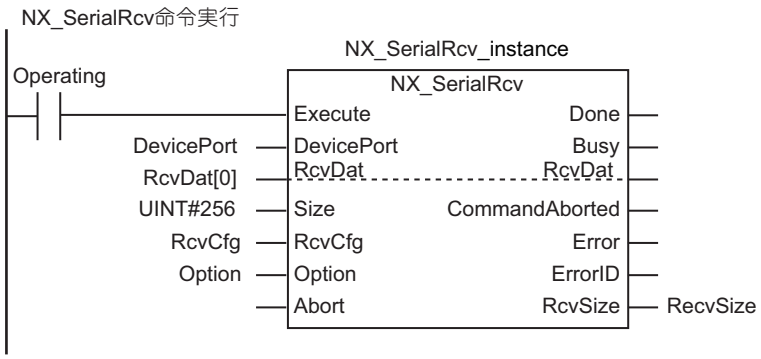
LD

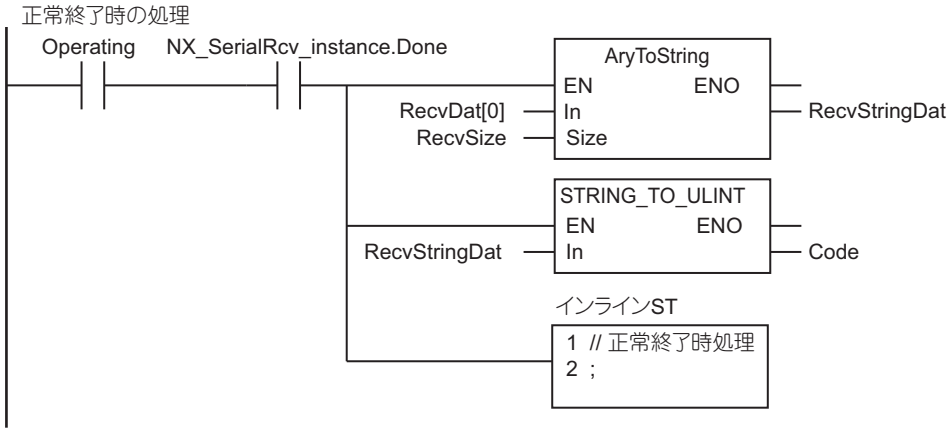
内部変数	名称	データ型	初期値	コメント
	OperationEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	DevicePort	_sDEVICE_PORT		ポート指定
	RecvDat	ARRAY [0..255] of BYTE	[256(16#0)]	受信データ
	RecvSize	UINT	0	受信データサイズ
	RecvStringDat	STRING[257]	''	
	Code	ULINT	0	バーコード（整数）
	RS_instance	RS		
	NX_SerialRcv_instance	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	

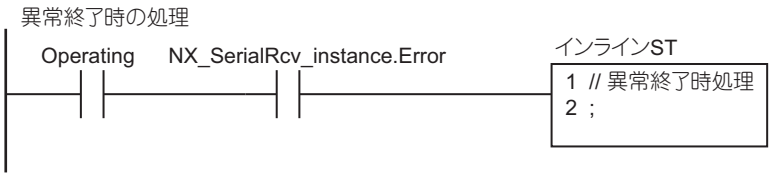
外部変数	名称	データ型	コメント
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• NX ユニット 63 台分の I/O データの使用可否 • 該当のユニット番号が「1」の場合、E001_NX_Unit_I_O_Data_Active_Status_63[1]を利用
	N1_Node_location_information	_sNXUNIT_ID	形 NX-CIF210 指定するデバイス変数











● 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）
	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およびオプションボード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。

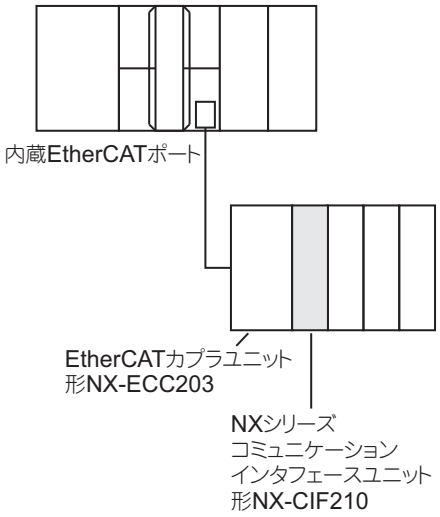
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuCmd	Modbus RTU 汎用コマンド送信	FB	NX_ModbusRtuCmd_instance NX_ModbusRtuCmd Execute / Done DevicePort / Busy SlaveAdr / CommandAborted CmdDat / Error CmdSize / ErrorID RespDat / RespDat Option / ErrorIDEx Abort / RespSize	NX_ModbusRtuCmd_instance(Execute, DevicePort, SlaveAdr, CmdDat, CmdSize, RespDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, RespSize);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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およびオプションボード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。

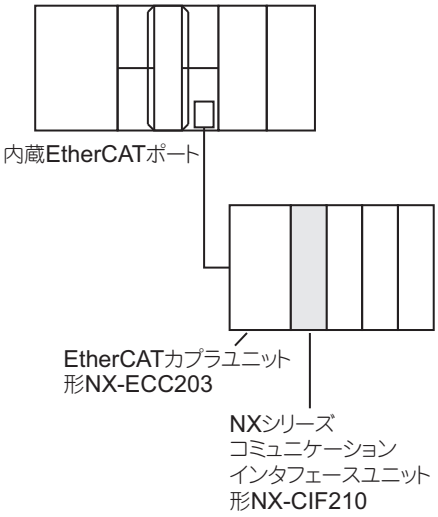
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuRead	Modbus RTU Read コマンド送信	FB	NX_ModbusRtuRead_instance NX_ModbusRtuRead Execute / Done DevicePort / Busy SlaveAdr / CommandAborted ReadCmd / Error ReadDat / ReadDat Option / ErrorID Abort / ErrorIDEx ReadSize	NX_ModbusRtuRead_instance(Execute, DevicePort, SlaveAdr, ReadCmd, ReadDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, ReadSize);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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およびオプションボード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。

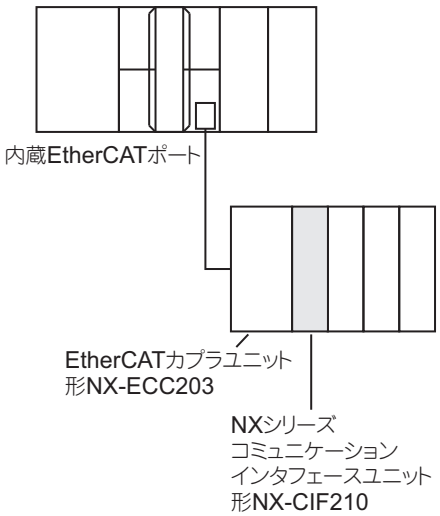
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_ModbusRtuWrite	Modbus RTU Write コマンド送信	FB	NX_ModbusRtuWrite_instance NX_ModbusRtuWrite Execute / DevicePort / SlaveAdr / WriteCmd / WriteDat / Option / Abort Done / Busy / CommandAborted / Error / ErrorID / ErrorIDEx	NX_ModbusRtuWrite_instance(Execute, DevicePort, SlaveAdr, WriteCmd, WriteDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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およびオプションボード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。

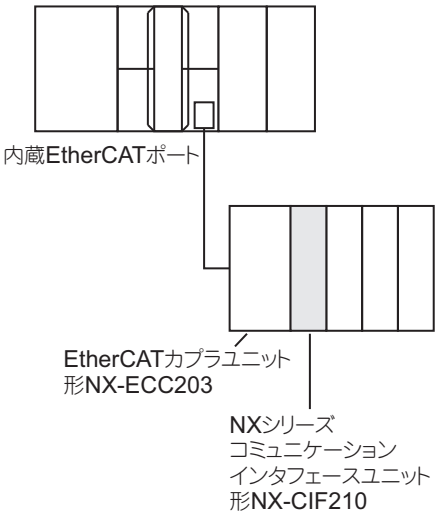
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialSigCtl	シリアル制御信号の ON/OFF 切り替え	FB	NX_SerialSigCtl_instance NX_SerialSigCtl Execute / DevicePort / Kind / Sig / TimeOut Done / Busy / Error / ErrorID	NX_SerialSigCtl_instance(Execute, DevicePort, Kind, Sig, TimeOut, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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

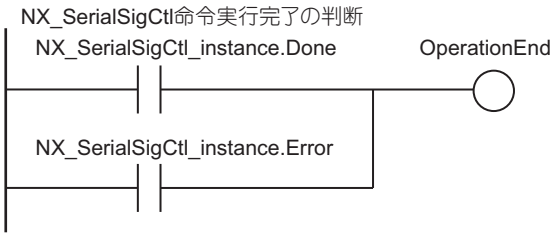
名称	データ型	割付先	コメント
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

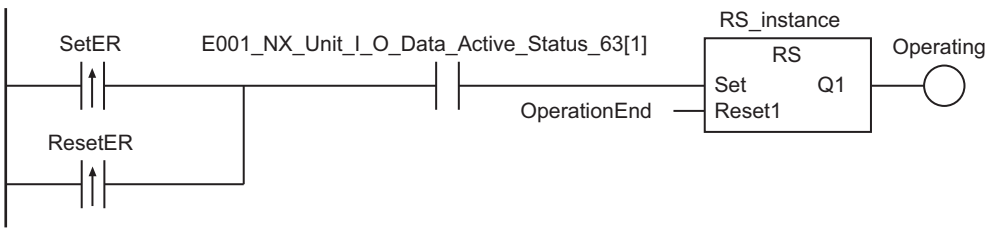
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

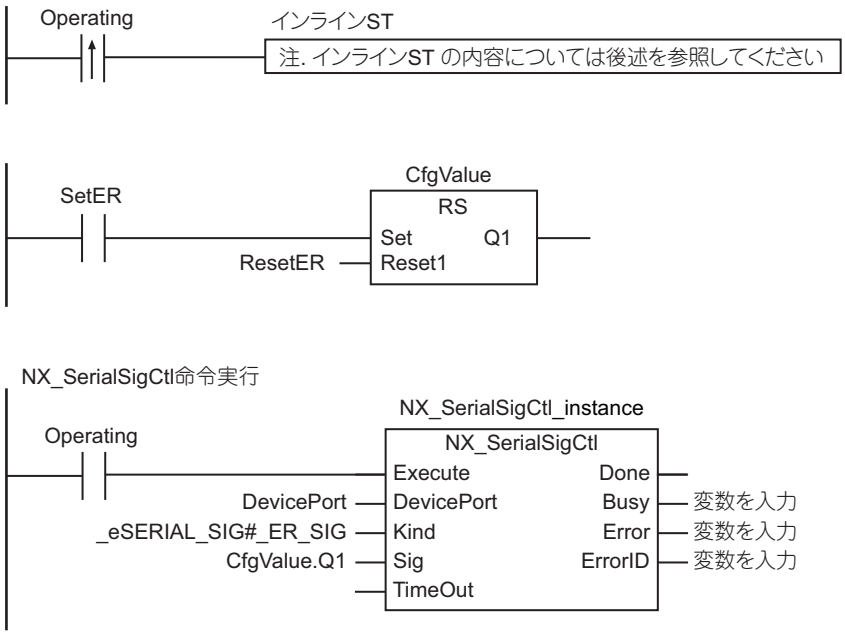
LD

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
	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
	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_SerialSigRead

オプションボードのシリアルポートの CS 信号または DR 信号を読み出します。

命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialSigRead	シリアル制御信号の読出	FB	NX_SerialSigRead_instance NX_SerialSigRead Execute / DevicePort / Kind / TimeOut Done / Busy / Error / ErrorID / Sig	NX_SerialSigRead_instance(Execute, DevicePort, Kind, TimeOut, Done, Busy, Error, ErrorID, Sig);

使用上の注意

本命令は、形 NX1P2 CPU ユニット用のオプションボードにだけ使用することができます。

サンプルプログラム

NX_SerialSigCtl 命令の、「サンプルプログラム (P.1-449)」を参照してください。

NX_SerialStatusRead

オプションボードのシリアルポー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
NX_SerialStatusRead	シリアルポートステータスの読出	FB	NX_SerialStatusRead_instance NX_SerialStatusRead Execute / DevicePort / TimeOut Done / Busy / Error / ErrorID / PortStatus	NX_SerialStatusRead_instance(Execute, DevicePort, TimeOut, Done, Busy, Error, ErrorID, PortStatus);

使用上の注意

本命令は、形 NX1P2 CPU ユニット用のオプションボードにだけ使用することができます。

サンプルプログラム

NX_SerialSigCtl 命令の、「サンプルプログラム (P.1-449)」を参照してください。

NX_SerialBufClear

送受信バッファをクリアします。

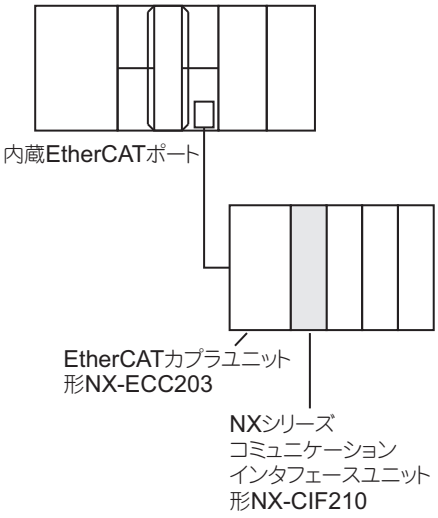
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_SerialBufClear	バッファクリア	FB	NX_SerialBufClear_instance NX_SerialBufClear Execute / Done DevicePort / Busy BufKind / Error TimeOut / ErrorID	NX_SerialBufClear_instance(Execute, DevicePort, BufKind, TimeOut, Done, Busy, Error, ErrorID);

バージョン関連情報

本命令は、CPU ユニット Ver.1.11 以降、かつ Sysmac Studio Ver.1.15 以降で、使用可能です。

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

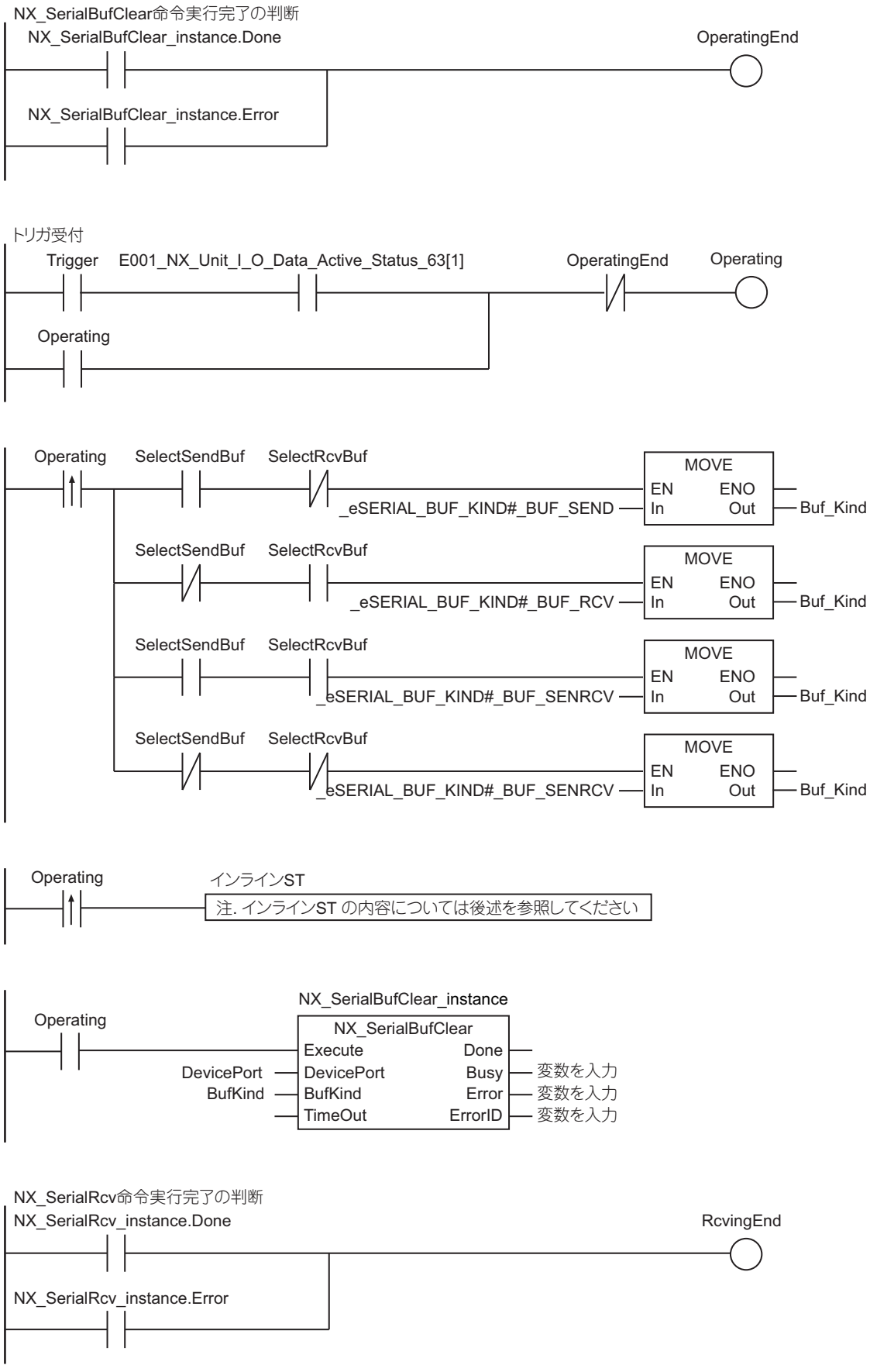
名称	データ型	割付先	コメント
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	NX ユニット 63 台分の I/O データの使用可否
N1_Node_location_information	_sNXUNIT_ID	−	形 NX-CIF210 指定するデバイス変数 *1

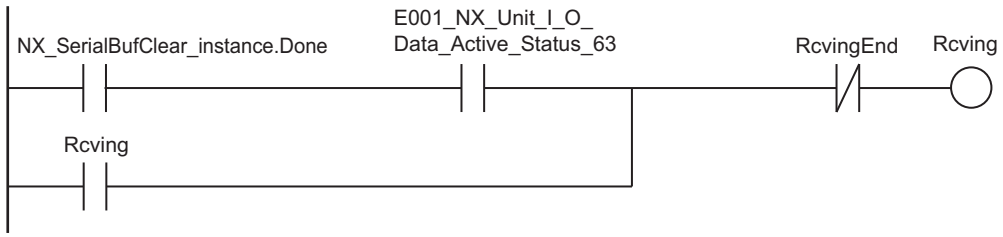
*1. Sysmac Studio で、NX シリーズスレーブターミナルのユニットを右クリックして［ノード位置ポート表示］を選択し、デバイス変数を設定します。詳細は、『Sysmac Studio Version 1 オペレーションマニュアル(SBCA-470)』を参照してください。

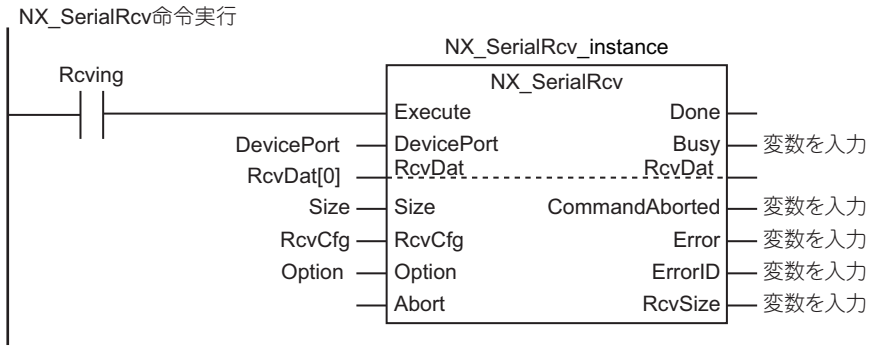
LD

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
	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


NX_SerialBufClear命令実行完了の判断
NX_SerialBufClear_instance.Done
OperatingEnd
NX_SerialBufClear_instance.Error
トリガ受付
Trigger
E001_NX_Unit_I_O_Data_Active_Status_63[1]
OperatingEnd
Operating
Operating
Operating
SelectSendBuf
SelectRcvBuf
MOVE
EN
ENO
In
Out
_eSERIAL_BUF_KIND#_BUF_SEND
Buf_Kind
SelectSendBuf
SelectRcvBuf
_eSERIAL_BUF_KIND#_BUF_RCV
SelectSendBuf
SelectRcvBuf
_eSERIAL_BUF_KIND#_BUF_SENRCV
SelectSendBuf
SelectRcvBuf
_eSERIAL_BUF_KIND#_BUF_SENRCV
Operating
インラインST
注. インラインST の内容については後述を参照してください
NX_SerialBufClear_instance
Operating
NX_SerialBufClear
Execute
Done
DevicePort
DevicePort
Busy
変数を入力
BufKind
BufKind
Error
変数を入力
TimeOut
ErrorID
変数を入力
NX_SerialRcv命令実行完了の判断
NX_SerialRcv_instance.Done
RcvingEnd
NX_SerialRcv_instance.Error





● 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
	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);
```

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.1-462
FileReadVar	変数ファイル読出	P.1-465
FileOpen	ファイルオープン	P.1-468
FileClose	ファイルクローズ	P.1-469
FileSeek	ファイルシーク	P.1-470
FileRead	ファイル読出	P.1-471
FileWrite	ファイル書込	P.1-476
FileGets	文字列読出	P.1-481
FilePuts	文字列書込	P.1-486
FileCopy	ファイルコピー	P.1-493
FileRemove	ファイル削除	P.1-497
FileRename	ファイル名変更	P.1-500
DirCreate	ディレクトリ作成	P.1-503
DirRemove	ディレクトリ削除	P.1-504
BackupToMemoryCard	SD メモリカードバックアップ	P.1-505

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 FileWriteVar Execute / Done FileName / Busy WriteVar / Error OverWrite / ErrorID	FileWriteVar_instance(Execute, FileName, WriteVar, OverWrite, Done, Busy, Error, ErrorID);

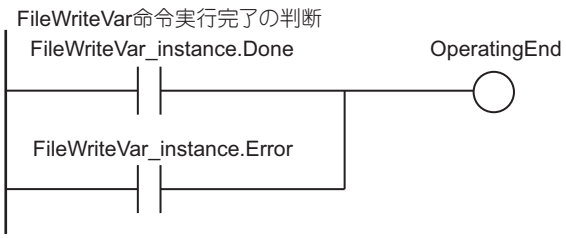
サンプルプログラム

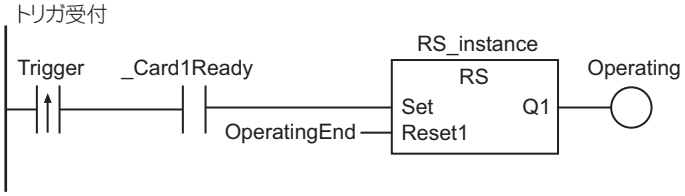
配列変数 Var1[]の全体を、ファイル'File1.dat'に書き込みます。

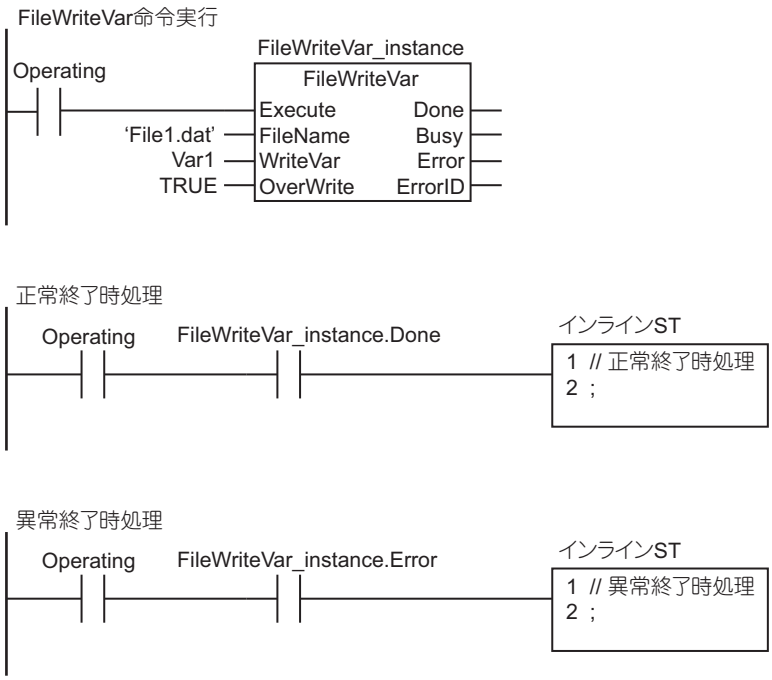
LD

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







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
```

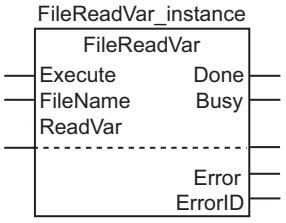
```
IF (Operating=TRUE) THEN
    FileWriteVar_instance(
        Execute  :=TRUE,
        FileName :='File1.dat',  // 指定ファイル名
        WriteVar :=Var1,         // 指定変数
        OverWrite:=TRUE);        // 上書許可

    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);

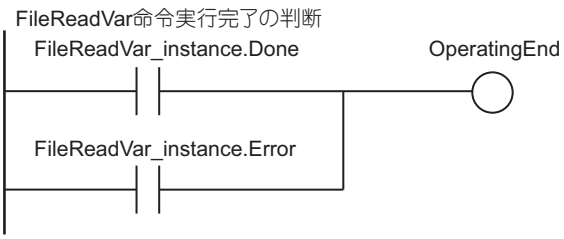
サンプルプログラム

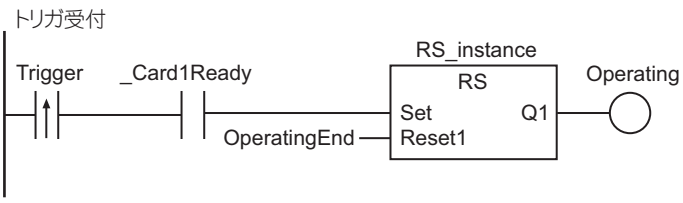
ファイル'File1.dat'の内容を読み込み、配列変数「Var1」に格納します。

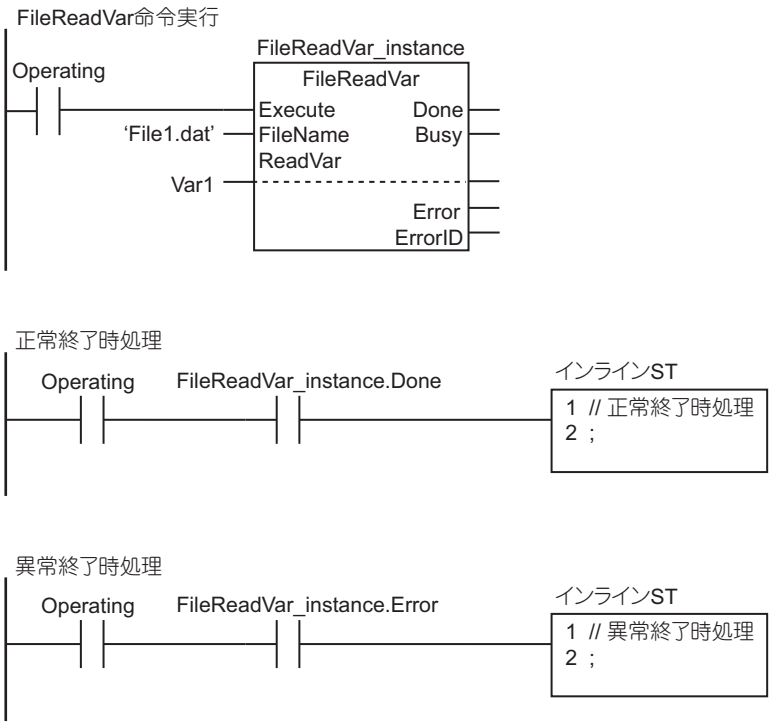
LD

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;
```

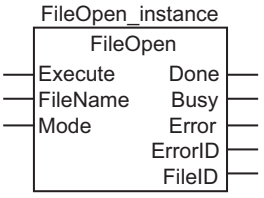
```
// FileReadVar 命令実行
IF (Operating=TRUE) THEN
    FileReadVar_instance(
        Execute :=TRUE,
        FileName:='File1.dat',     // 指定ファイル名
        ReadVar :=Var1);           // 読出先変数

    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。

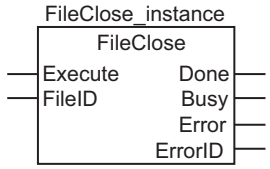
命令	名称	FB/FUN	グラフィック表現	ST 表現
FileOpen	ファイルオープン	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.1-471)」、FileWrite 命令の「サンプルプログラム (P.1-476)」、FileGets 命令の「サンプルプログラム (P.1-481)」、FilePuts 命令の「サンプルプログラム (P.1-486)」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。

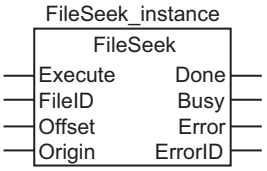
命令	名称	FB/FUN	グラフィック表現	ST 表現
FileClose	ファイルクローズ	FB		FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.1-471)」、FileWrite 命令の「サンプルプログラム (P.1-476)」、FileGets 命令の「サンプルプログラム (P.1-481)」、FilePuts 命令の「サンプルプログラム (P.1-486)」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。

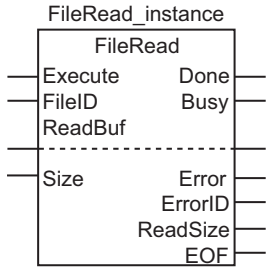
命令	名称	FB/FUN	グラフィック表現	ST 表現
FileSeek	ファイルシーク	FB		FileSeek_instance(Execute, FileID, Offset, Origin, Done, Busy, Error, ErrorID);

サンプルプログラム

FileRead 命令の「サンプルプログラム (P.1-471)」、FileWrite 命令の「サンプルプログラム (P.1-476)」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);

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。**

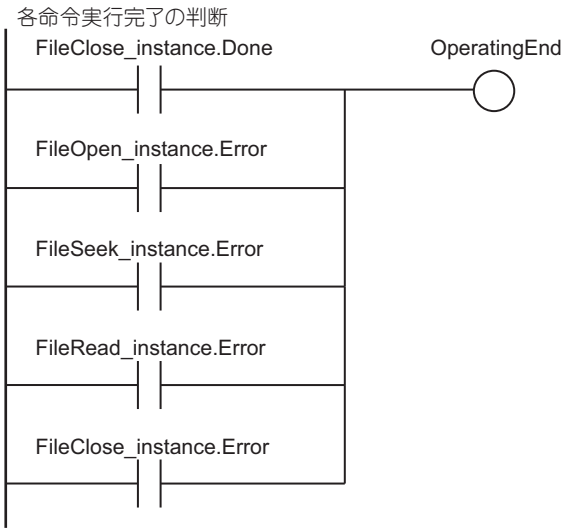
3 **FileRead 命令を使って、ファイル位置指示子の位置から 4 バイト分のデータを読み出し、配列変数 InDat[]に格納します。**

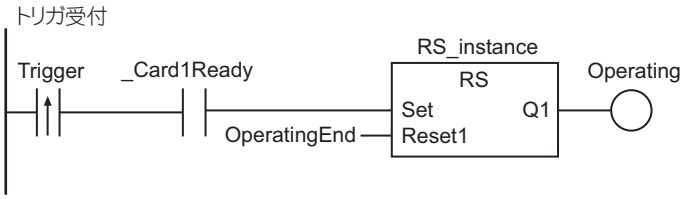
4 **FileClose 命令を使って、'ABC.bin'をクローズします。**

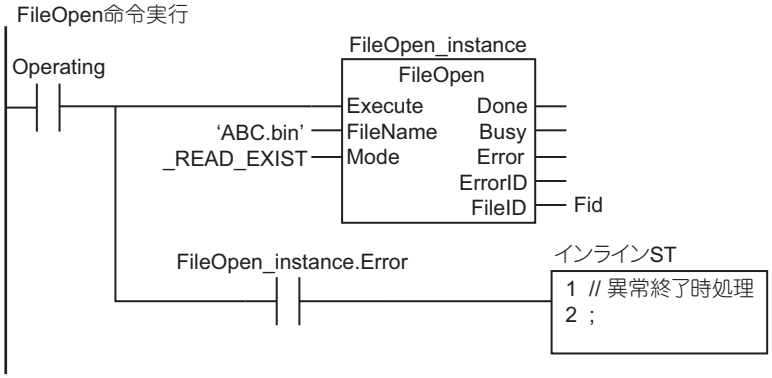
LD

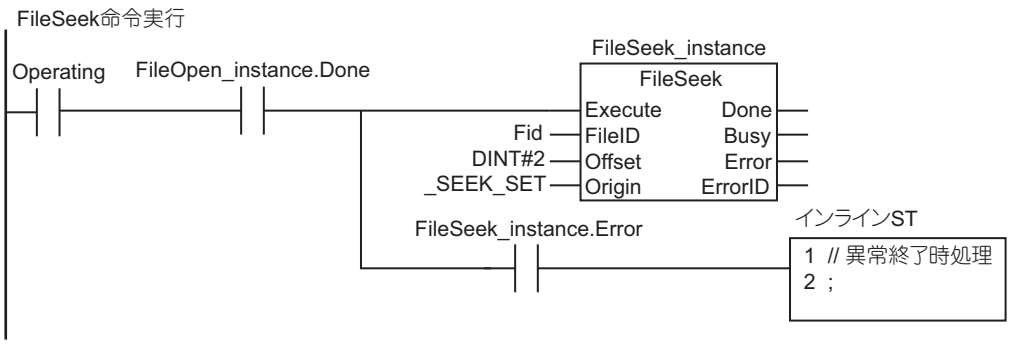
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Fid	DWORD	16#0	ファイル ID
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	読出データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

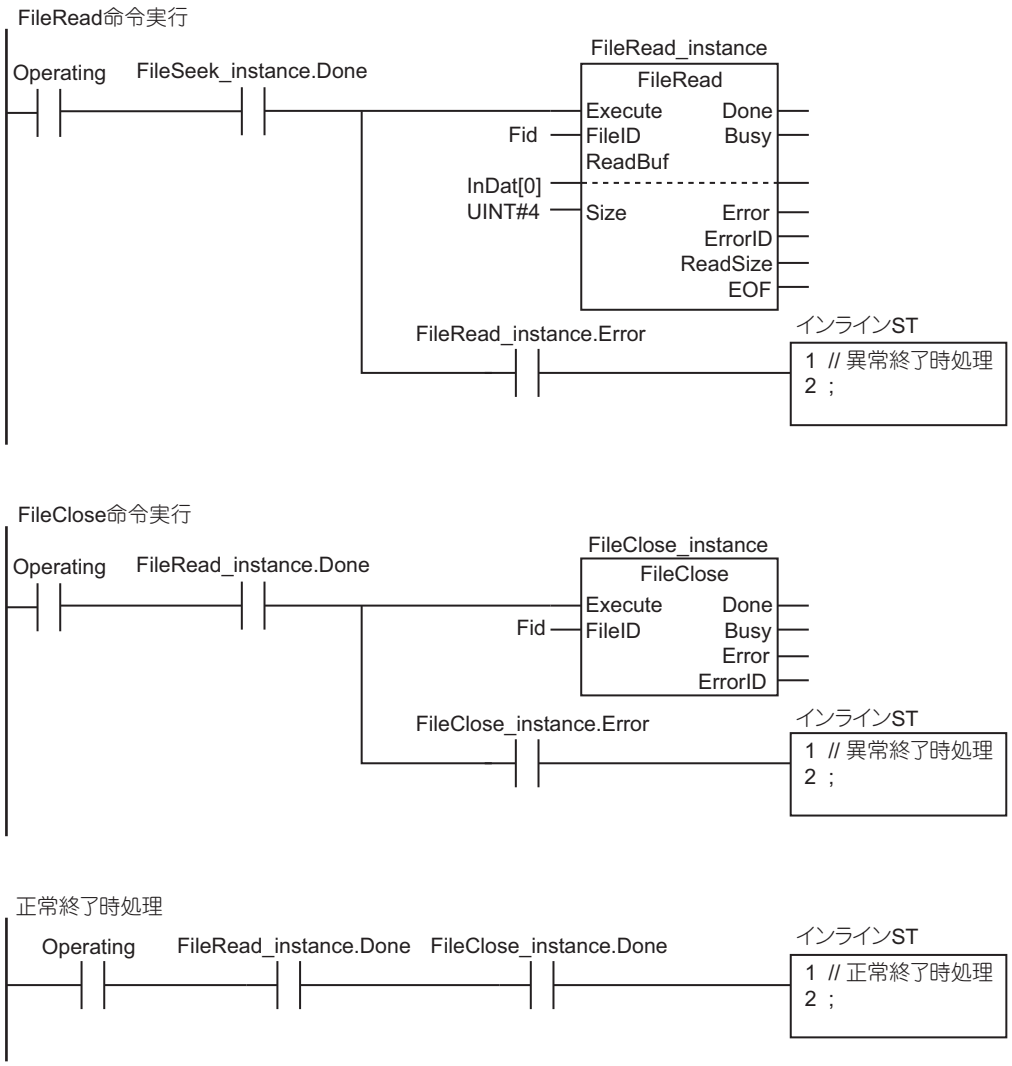
外部変数	名称	データ型	コメント
	_Card1Ready	BOOL	SD メモリカード使用可フラグ


各命令実行完了の判断
FileClose_instance.Done
OperatingEnd
FileOpen_instance.Error
FileSeek_instance.Error
FileRead_instance.Error
FileClose_instance.Error


トリガ受付
Trigger
_Card1Ready
RS_instance
RS
Set
Q1
Reset1
OperatingEnd
Operating


FileOpen命令実行
Operating
FileOpen_instance
FileOpen
Execute
Done
'ABC.bin'
FileName
Busy
_READ_EXIST
Mode
Error
ErrorID
FileID
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
Done
Fid
FileID
Busy
DINT#2
Offset
Error
_SEEK_SET
Origin
ErrorID
FileSeek_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

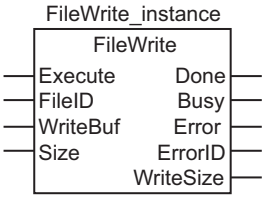
        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;           // 正常終了
        END_IF;

        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);

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。**

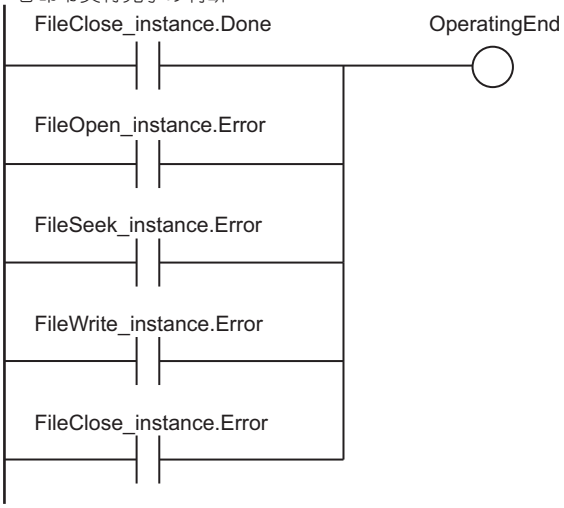
4 **FileClose 命令を使って、'ABC.bin'をクローズします。**

LD

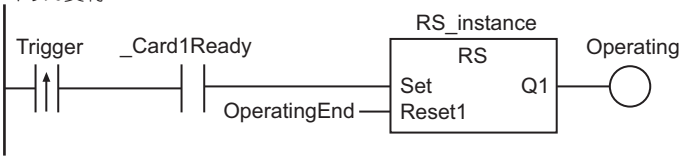
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Fid	DWORD	16#0	ファイル ID
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	書込データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	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

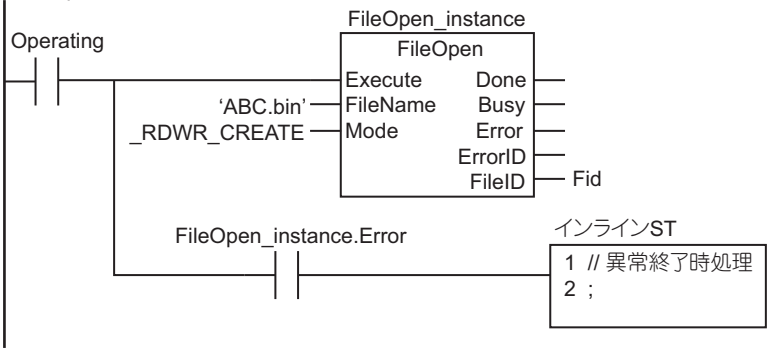
各命令実行完了の判断



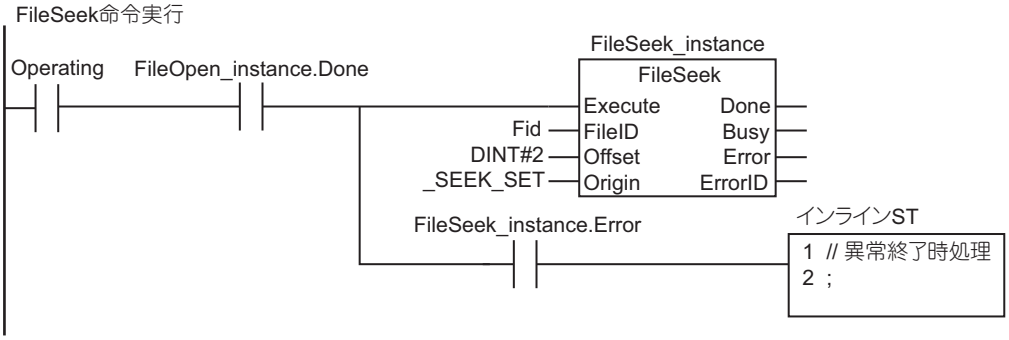
トリガ受付

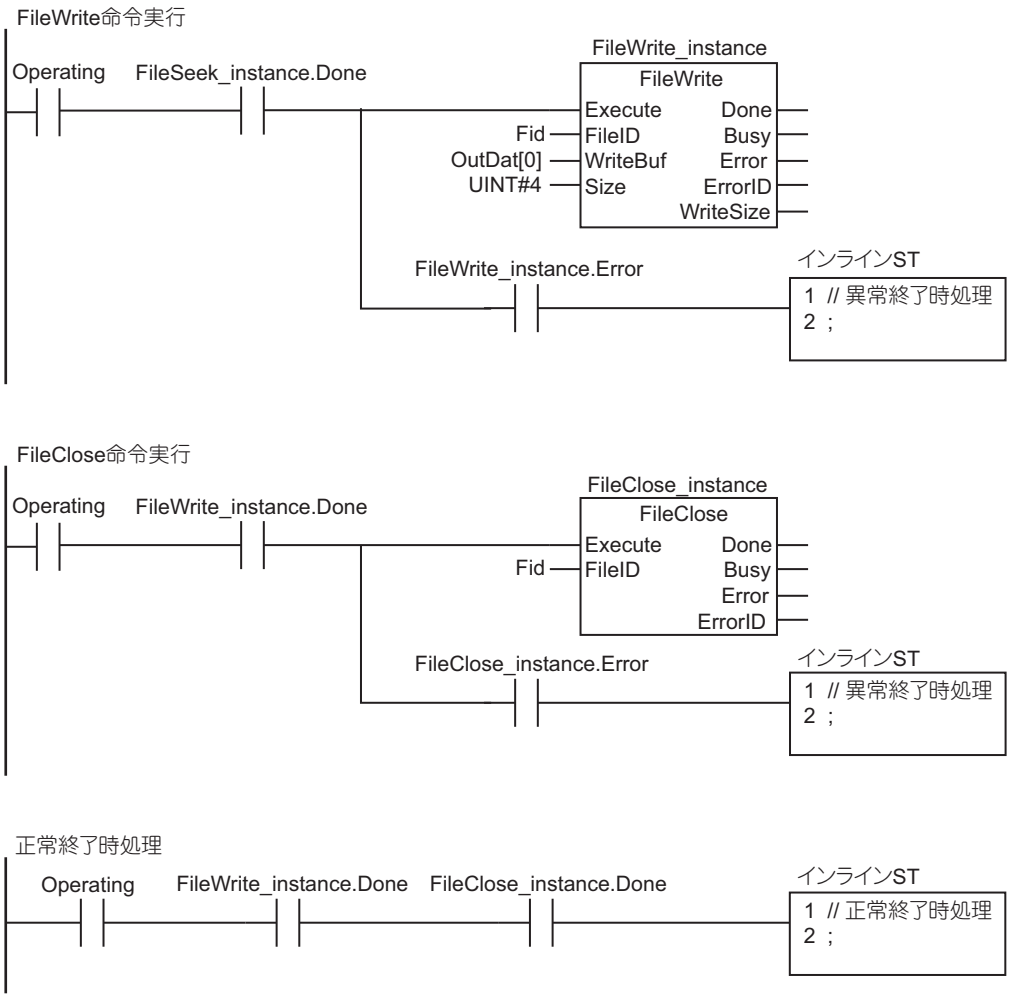


FileOpen命令実行



FileSeek命令実行





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 FileGets Execute / FileID / TrimLF Done / Busy / Error / ErrorID / Out / EOF	FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

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。**

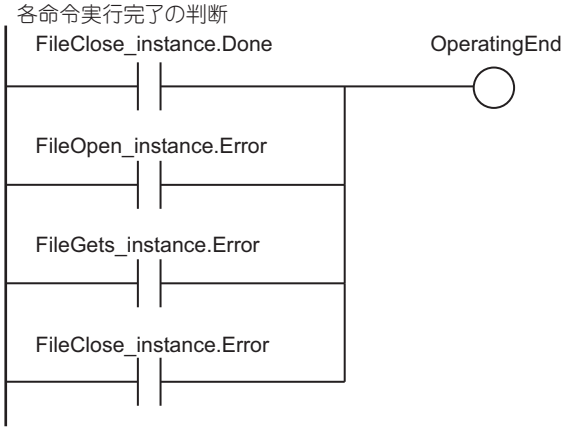
LD

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

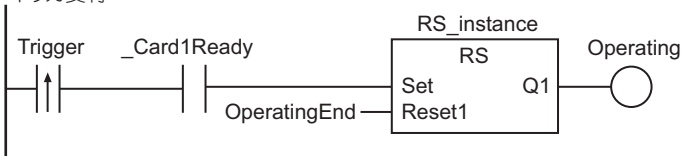
内部変数	名称	データ型	初期値	コメント
	Index	INT	0	InDat[]の要素インデックス
	Fid	DWORD	16#0	ファイル ID
	InDat	ARRAY[0..999] OF INT	[1000(0)]	整数データ
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

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

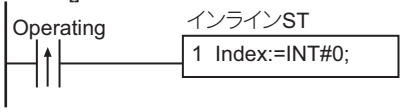
各命令実行完了の判断



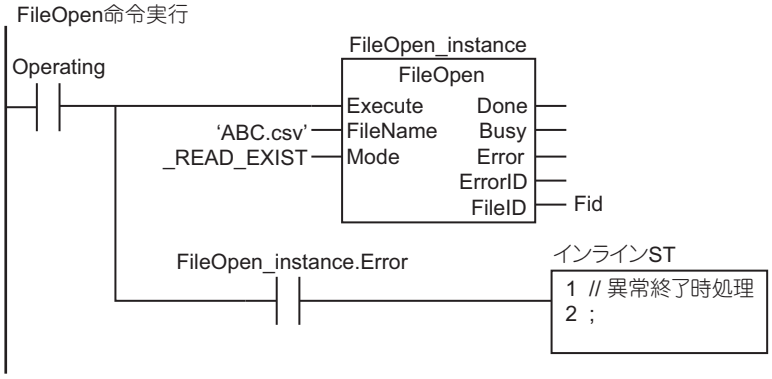
トリガ受付

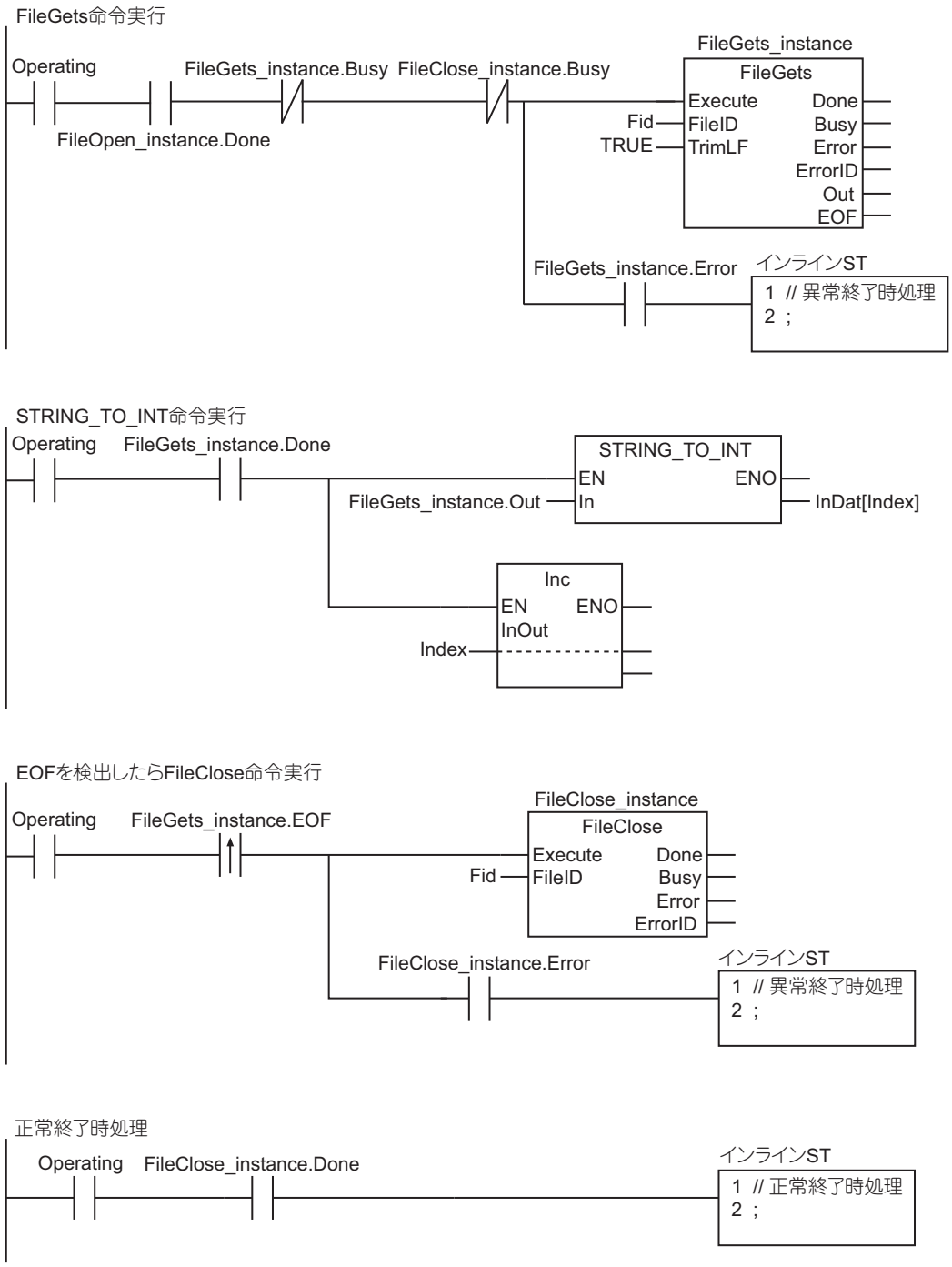


InDat[]の要素インデックス初期化



FileOpen命令実行





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

内部変数	名称	データ型	初期値	コメント
	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

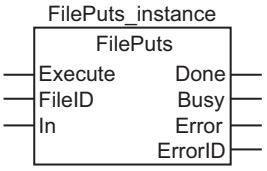
        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;   // 正常終了
        END_IF;

        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(Execute, FileID, In, Done, Busy, Error, ErrorID);

サンプルプログラム

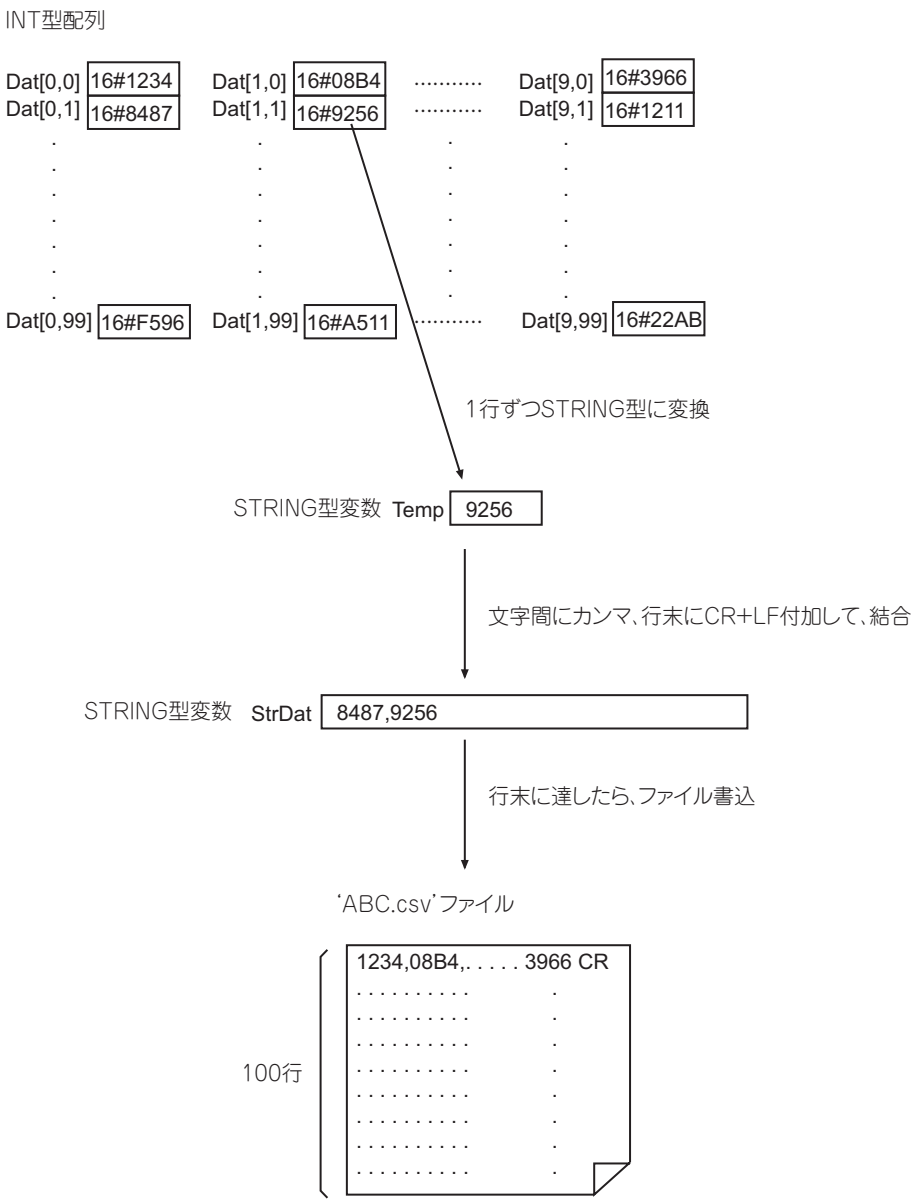
INT 型の配列変数 Dat[0..9,0..99]の内容を、100 行の CSV ファイル形式で、ファイル'ABC.csv'に格納します。
1 行には、数値の文字列が 10 個並び、それらの間にはカンマを挿入します。行末には、改行コード CR + LF を付加します。
手順は以下のとおりです。

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。**

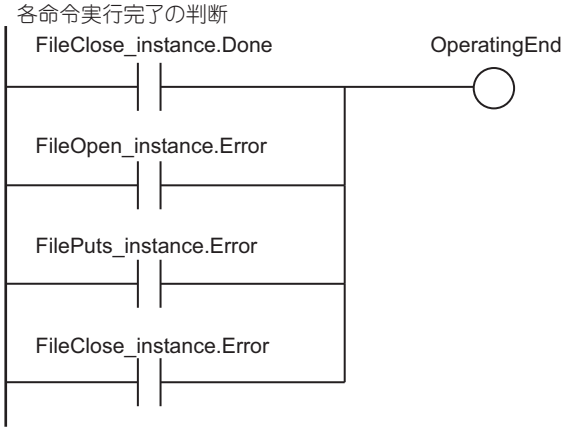


LD

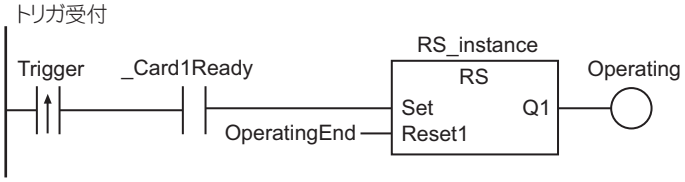
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	Index0	INT	0	列インデックス
	Index1	INT	0	行インデックス
	Fid	DWORD	16#0	ファイル ID
	StrDat	STRING[255]	''	文字列データ
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	数値データ
	Temp	STRING[255]	''	テンポラリ
	RS_instance	RS		
	FileOpen_instance	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

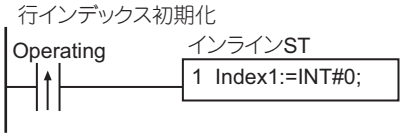
各命令実行完了の判断



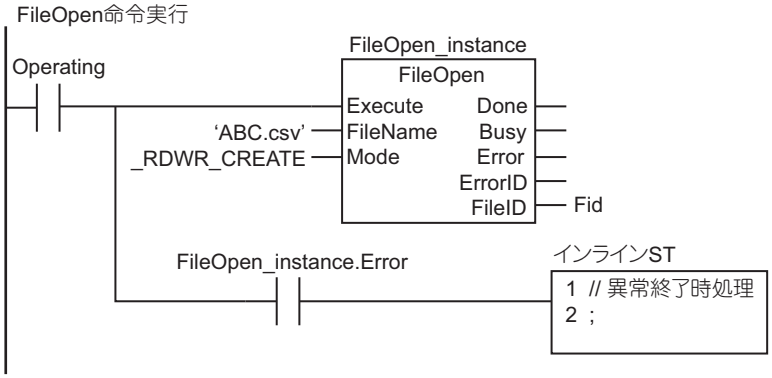
トリガ受付



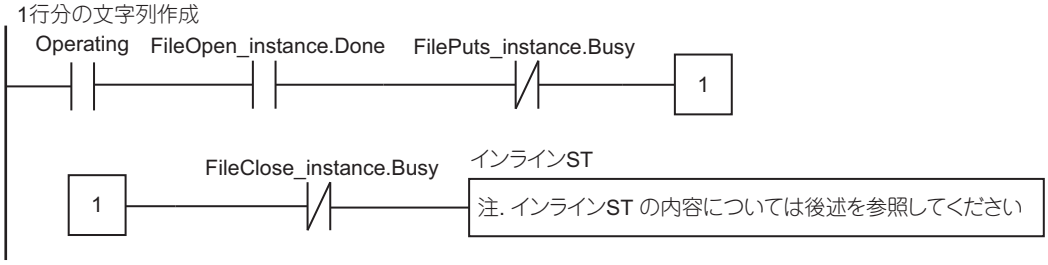
行インデックス初期化

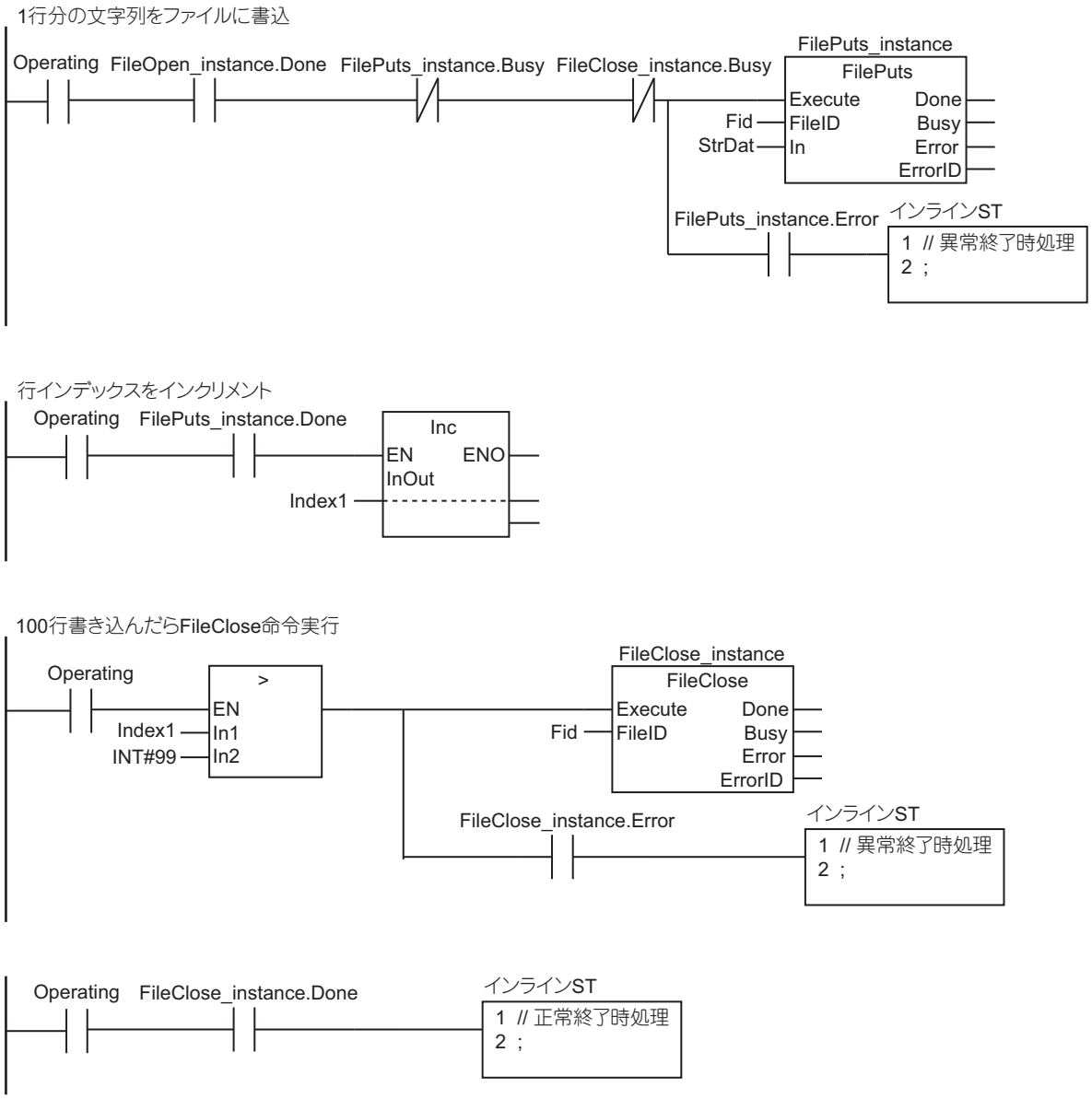


FileOpen命令実行



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 FileCopy Execute / Done SrcFileName / Busy DstFileName / Error OverWrite / ErrorID	FileCopy_instance(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

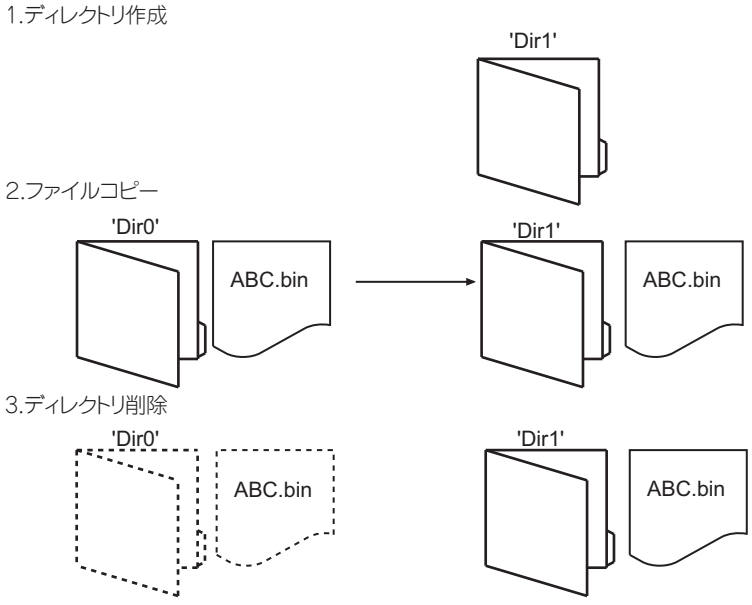
サンプルプログラム

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。**

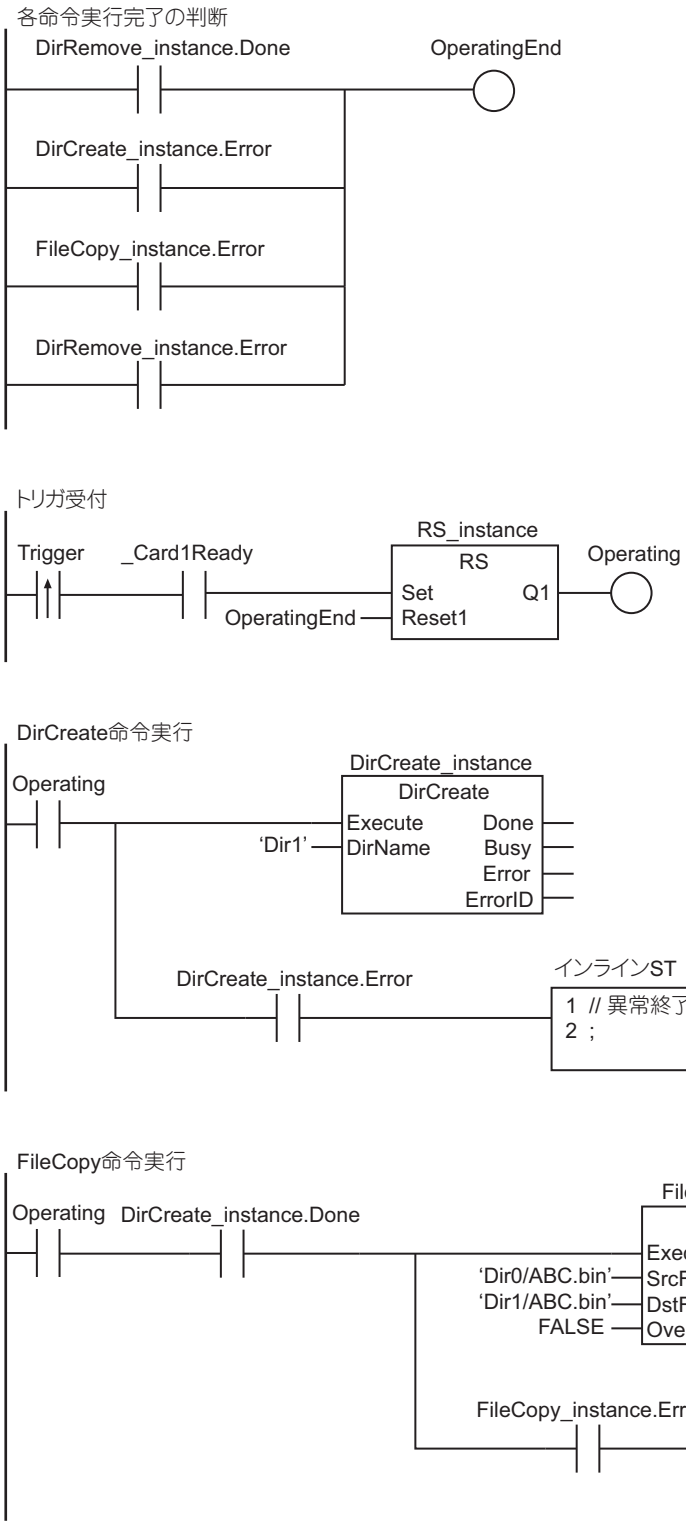


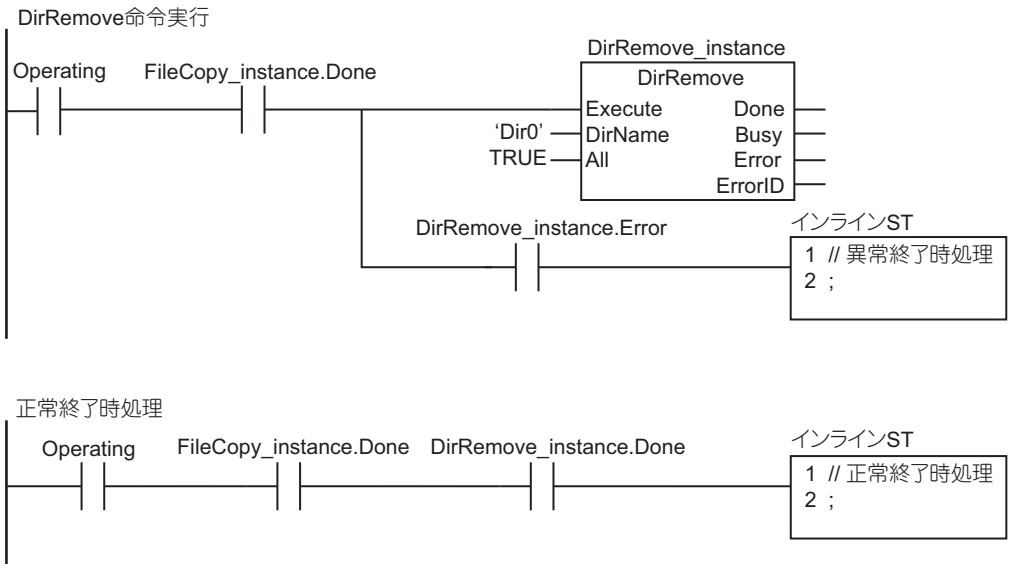
LD

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		

内部変数	名称	データ型	初期値	コメント
	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





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		
	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

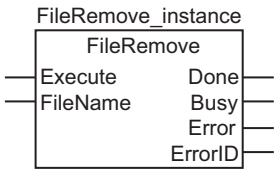
        IF (DirRemove_instance.Done=TRUE) THEN
            Operating:=FALSE;               // 正常終了
        END_IF;

        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(Execute, FileName, Done, Busy, Error, ErrorID);

サンプルプログラム

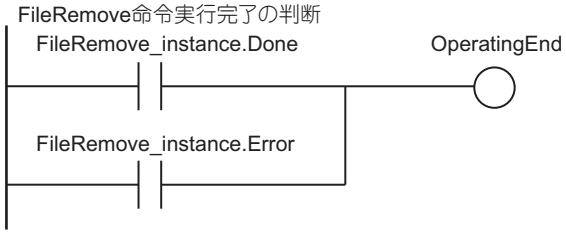
SD メモリカード内の、ファイル'ABC.bin'を削除します。

LD

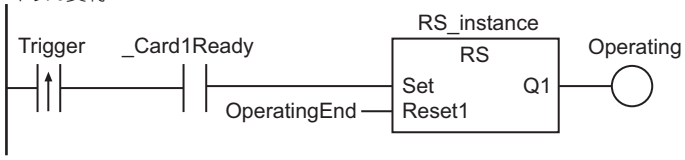
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	FileRemove_instance	FileRemove		

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

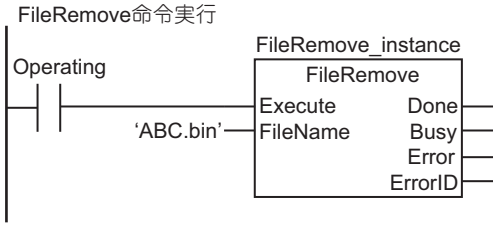
FileRemove命令実行完了の判断

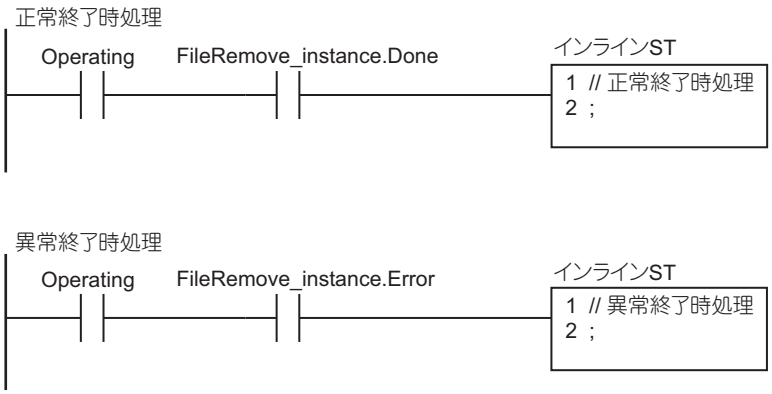


トリガ受付



FileRemove命令実行





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 FileRename Execute / Done FileName / Busy NewName / Error OverWrite / ErrorID	FileRename_instance(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

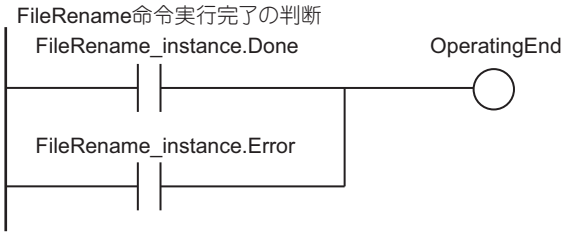
サンプルプログラム

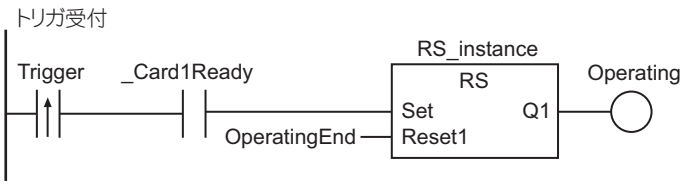
SD メモリカード内の、ファイル'ABC.bin'の名称を、'DEF.bin'に変更します。

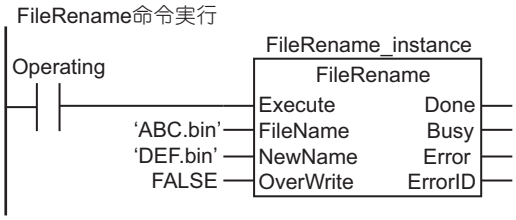
LD

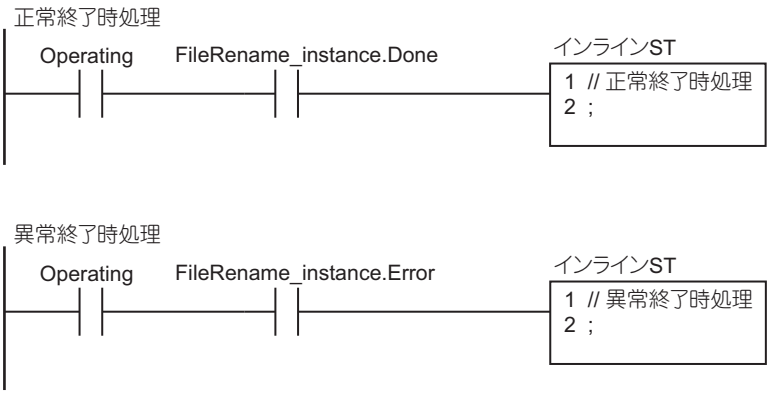
内部変数	名称	データ型	初期値	コメント
	OperatingEnd	BOOL	FALSE	処理終了
	Trigger	BOOL	FALSE	実行条件
	Operating	BOOL	FALSE	処理中
	RS_instance	RS		
	FileRename_instance	FileRename		

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;

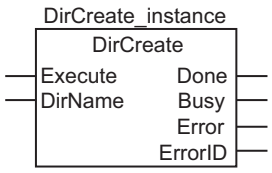
// FileRename 命令実行
IF (Operating=TRUE) THEN
    FileRename_instance(
        Execute  :=TRUE,
        FileName :='ABC.bin',     // 元ファイル名
        NewName  :='DEF.bin',     // 変更後ファイル名
        OverWrite:=FALSE);        // 上書禁止

    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。

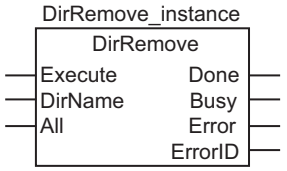
命令	名称	FB/FUN	グラフィック表現	ST 表現
DirCreate	ディレクトリ作成	FB		DirCreate_instance(Execute, DirName, Done, Busy, Error, ErrorID);

サンプルプログラム

FileCopy 命令の「サンプルプログラム (P.1-493)」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。

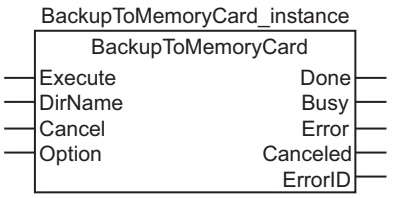
命令	名称	FB/FUN	グラフィック表現	ST 表現
DirRemove	ディレクトリ削除	FB		DirRemove_instance(Execute, DirName, All, Done, Busy, Error, ErrorID);

サンプルプログラム

FileCopy 命令の「サンプルプログラム (P.1-493)」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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

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

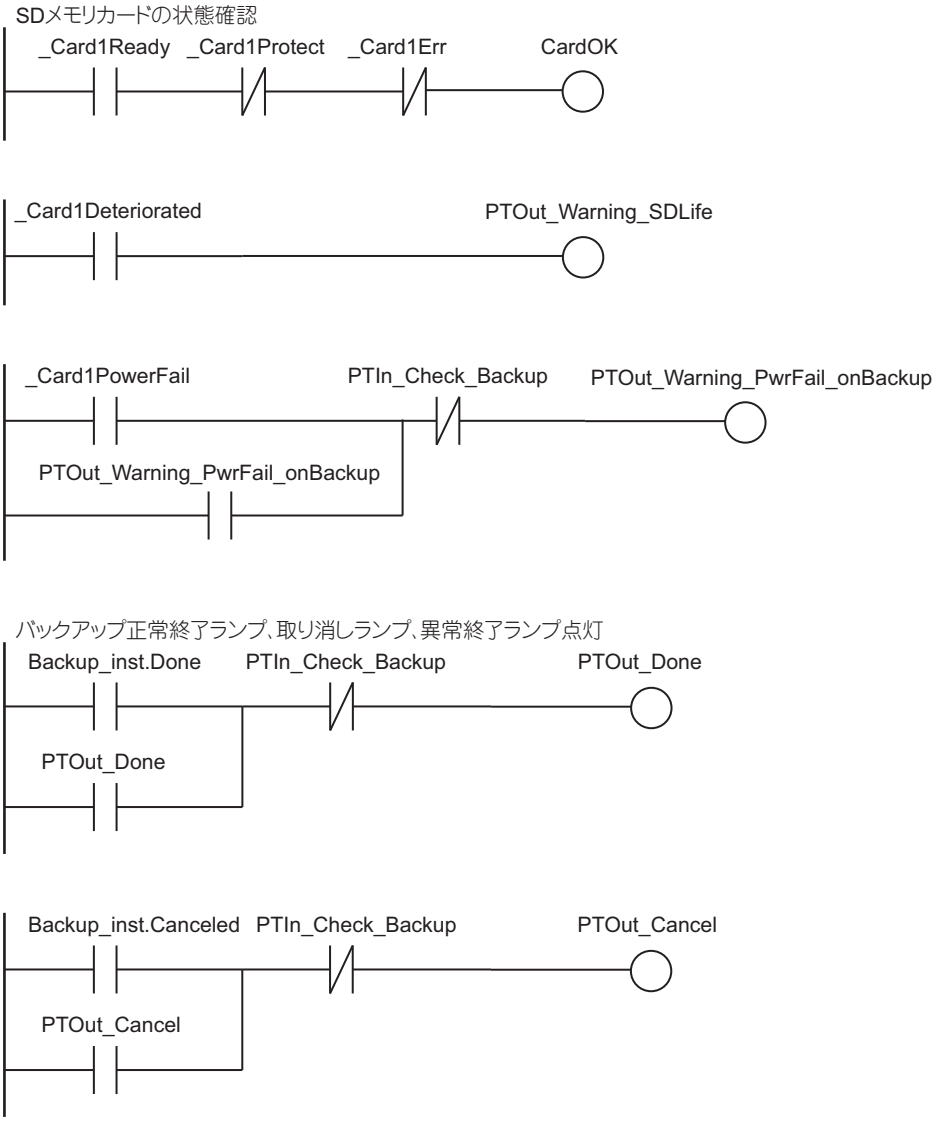
名称	データ型	初期値	コメント
PTOut_Warning_SDLife	BOOL	FALSE	SD メモリカード寿命警告ランプ出力
PTOut_Warning_PwrFail_onBackup	BOOL	FALSE	SD メモリカード電源断ランプ出力
PTOut_Done	BOOL	FALSE	バックアップ正常終了ランプ出力
PTOut_Cancel	BOOL	FALSE	バックアップ取り消しランプ出力
PTOut_Error	BOOL	FALSE	バックアップ異常終了ランプ出力
PTIn_Check_Backup	BOOL	FALSE	ランプ消灯ボタン入力
PTIn_Cancel	BOOL	FALSE	取り消しボタン入力

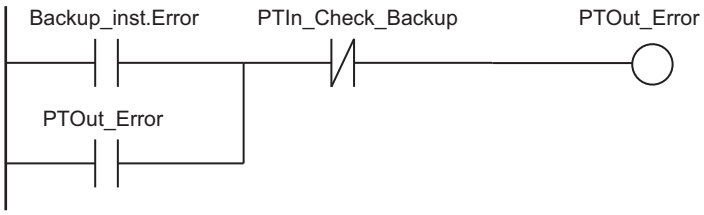
LD

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

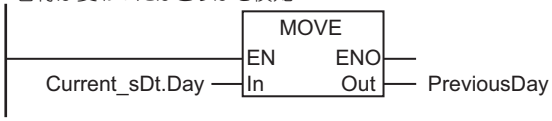
外部変数	名称	データ型	コンスタント	コメント
	_Card1Ready	BOOL	☑	SD メモリカード使用可フラグ
	_Card1Protect	BOOL	☑	SD メモリカードライトプロテクトフラグ
	_Card1Err	BOOL	☑	SD メモリカードエラーフラグ
	_Card1Deteriorated	BOOL	☑	SD メモリカード寿命警告フラグ
	_Card1PowerFail	BOOL	☐	SD メモリカードアクセス中電源断フラグ
	_BackupBusy	BOOL	☑	バックアップ関連機能実行中フラグ

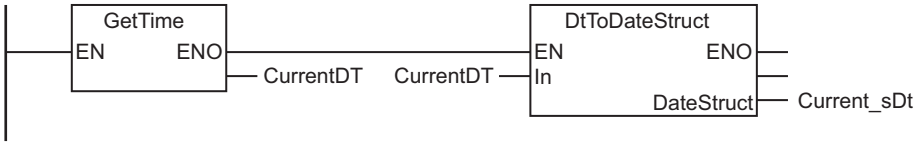
外部変数	名称	データ型	コンスタント	コメント
	PTOut_Warning_SDLife	BOOL	☐	SD メモリカード寿命警告ランプ出力
	PTOut_Warning_PwrFail_onBackup	BOOL	☐	SD メモリカード電源断ランプ出力
	PTOut_Done	BOOL	☐	バックアップ正常終了ランプ出力
	PTOut_Cancel	BOOL	☐	バックアップ取り消しランプ出力
	PTOut_Error	BOOL	☐	バックアップ異常終了ランプ出力
	PTIn_Check_Backup	BOOL	☐	ランプ消灯ボタン入力
	PTIn_Cancel	BOOL	☐	取り消しボタン入力

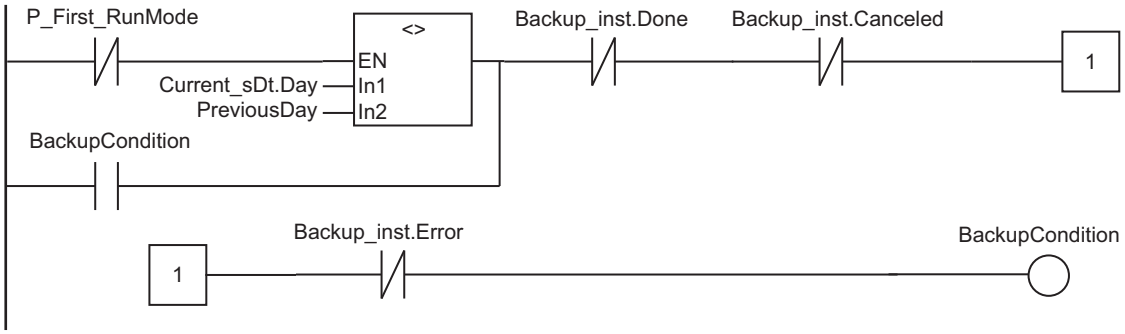




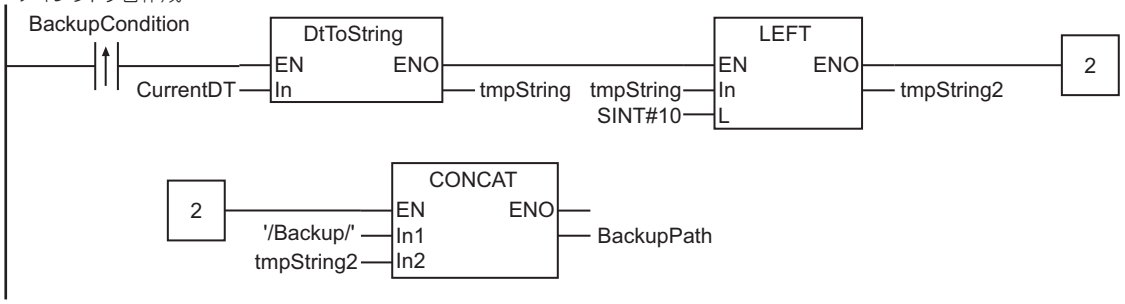
日付が変わったかどうかを検知



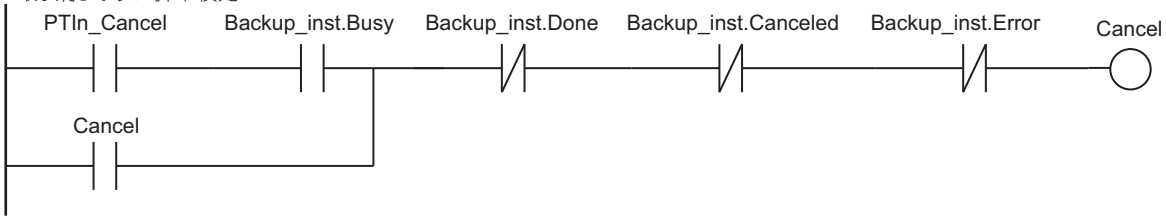


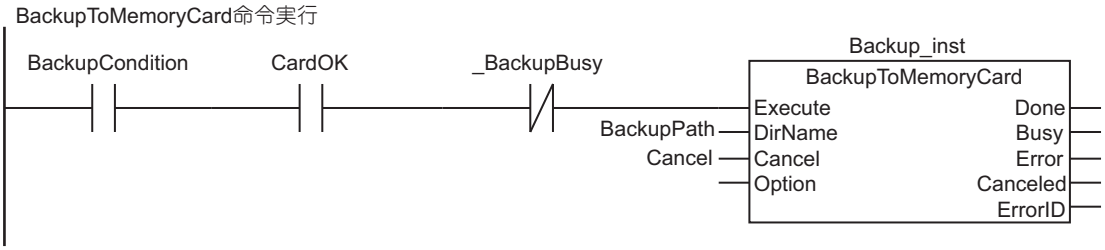


ディレクトリ名作成



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.1-514
NX_AryDOutTimeStamp	タイムスタンプデジタル出力配列書込	P.1-517

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。

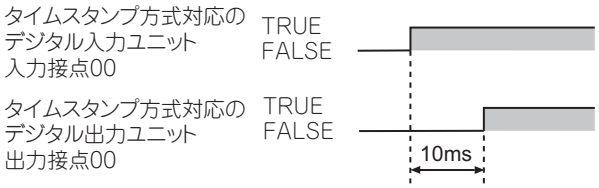
命令	名称	FB/FUN	グラフィック表現	ST 表現
NX_DOutTimeStamp	タイムスタンプデジタル出力書込	FB	NX_DOutTimeStamp_instance NX_DOutTimeStamp Enable SetDOut SetTimeStamp SyncOutTime DOut TimeStamp ENO	NX_DOutTimeStamp_instance(Enable, SetDOut, SetTimeStamp, SyncOutTime, DOut, TimeStamp);

バージョン関連情報

本命令は、CPU ユニット Ver.1.06 以降、かつ Sysmac Studio Ver.1.07 以降で、使用可能です。

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

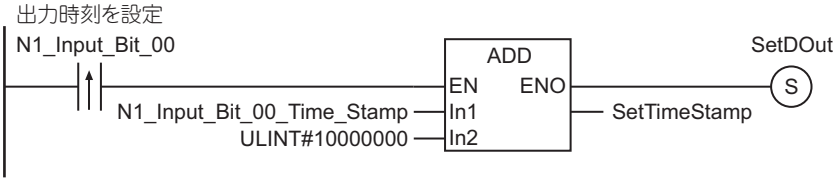
I/O マップの設定は、以下のとおりです。

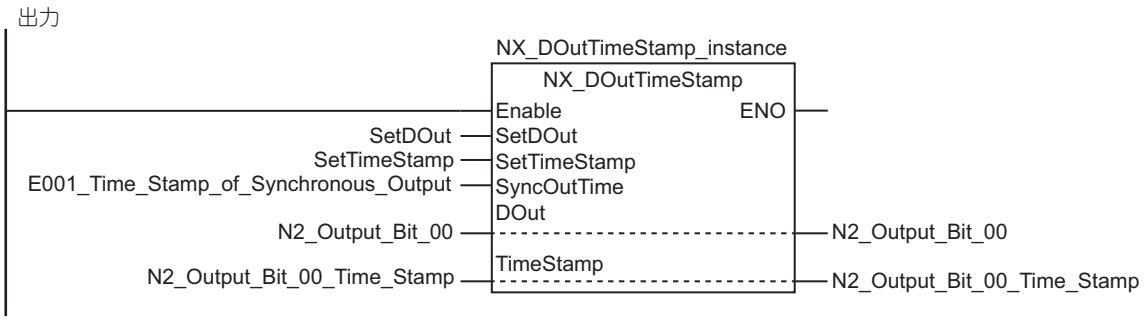
位置	ポート	説明	R/W	データ型	変数	変数種別
Node1	Time Stamp of Synchronous Output	接続された NX ユニットが同期出力したタイミングの時刻情報が格納されます。（単位:ns）	R	ULINT	E001_Time_Stamp_of_Synchronous_Output	グローバル変数
Unit1	Input Bit 00	入力接点 00	R	BOOL	N1_Input_Bit_00	グローバル変数
Unit1	Input Bit 00 Time Stamp	入力接点 00 の変化時刻	R	ULINT	N1_Input_Bit_00_Time_Stamp	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

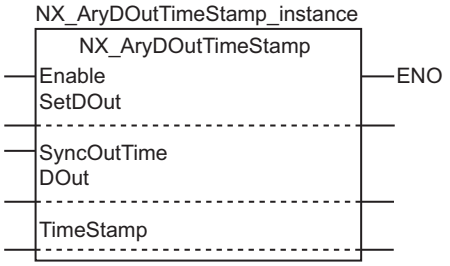
```
// 実行トリガ入力
R_TRIG_instance( N1_Input_Bit_00, SetEN);

// 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);

バージョン関連情報

本命令は、CPU ユニット Ver.1.06 以降、かつ Sysmac Studio Ver.1.07 以降で、使用可能です。

サンプルプログラム

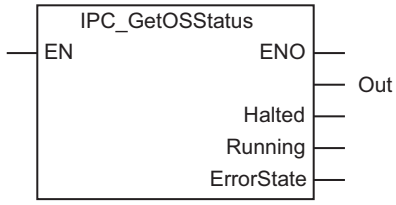
サンプルプログラムは、『』の、MC_DigitalCamSwitch 命令を参照してください。

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.1-520
IPC_RebootOS	OS 再起動	P.1-522
IPC_Shutdown	シャットダウン	P.1-524

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。

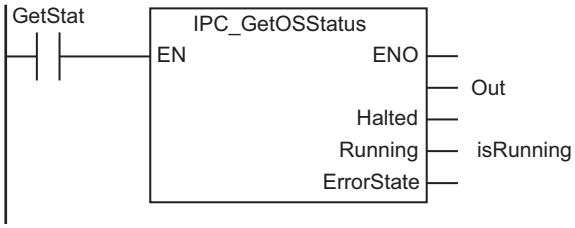
命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_GetOSStatus	OS ステータス読み出し	FUN		IPC_GetOSStatus(Halted, Running, ErrorState);

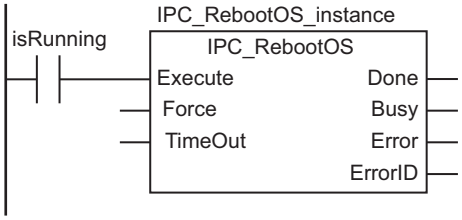
サンプルプログラム

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

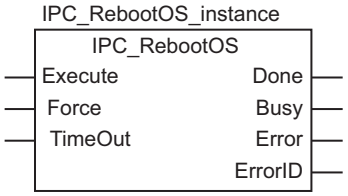
内部変数	名称	データ型	初期値	コメント
	GetStat	BOOL	FALSE	OS ステータス読み出し
	Out	BOOL	FALSE	

内部変数	名称	データ型	初期値	コメント
	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。

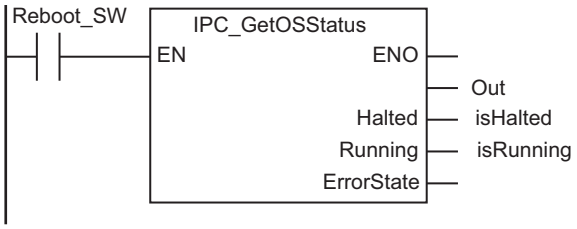
命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_RebootOS	OS 再起動	FB		IPC_RebootOS_instance(Execute, Force, TimeOut, Done, Busy, Error, ErrorID);

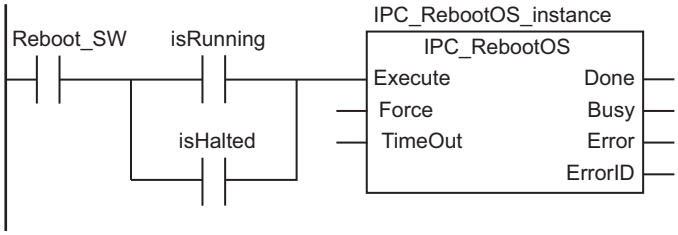
サンプルプログラム

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。

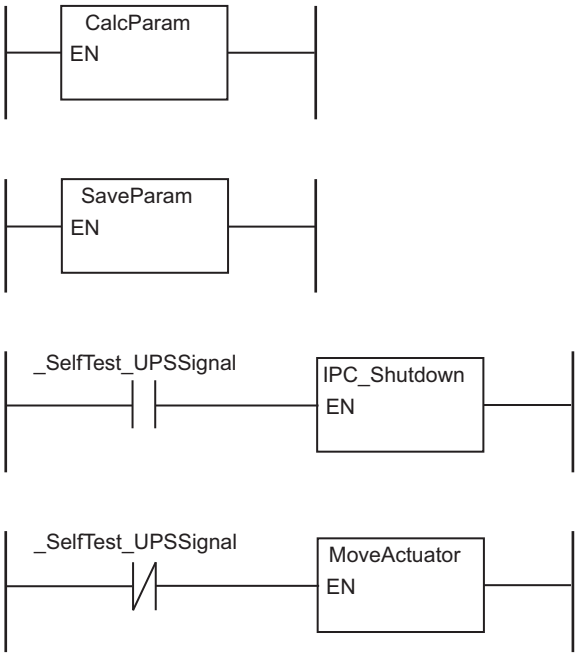
命令	名称	FB/FUN	グラフィック表現	ST 表現
IPC_Shutdown	シャットダウン	FUN	IPC_Shutdown EN — Out	IPC_Shutdown(EN);

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
ChkRange	範囲型変数チェック	P.1-528
GetMyTaskStatus	自タスク状態読出	P.1-530
GetMyTaskInterval	自タスク設定周期読出	P.1-532
Task_IsActive	タスク実行中判定	P.1-534
Lock / Unlock	タスク間排他ロック／タスク間排他ロック解除	P.1-535
ActEventTask	イベントタスク起動	P.1-539
GetPrgHashCode	プログラムハッシュコード取得	P.1-542

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。

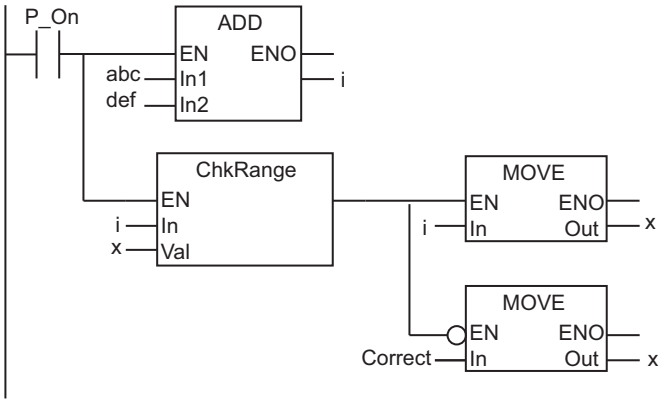
命令	名称	FB/FUN	グラフィック表現	ST 表現
ChkRange	範囲型変数チェック	FUN		Out:=ChkRange(In, Val);

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

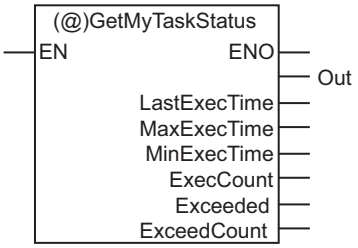
名称	データ型	初期値
i	INT	0
abc	INT	0
def	INT	0
Chk	BOOL	FALSE
x	INT(10..99)	10
Correct	INT	0

```
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。

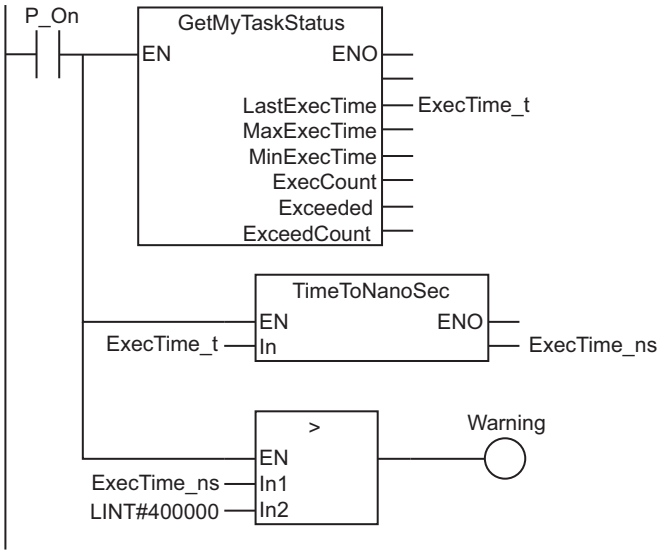
命令	名称	FB/FUN	グラフィック表現	ST 表現
GetMyTaskStatus	自タスク状態読出	FUN		GetMyTaskStatus(LastExecTime, MaxExecTime, MinExecTime, ExecCount, Exceeded, ExceedCount);

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）

名称	データ型	初期値	コメント
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();

バージョン関連情報

本命令は、CPU ユニット Ver.1.08 以降、かつ Sysmac Studio Ver.1.09 以降で、使用可能です。

サンプルプログラム

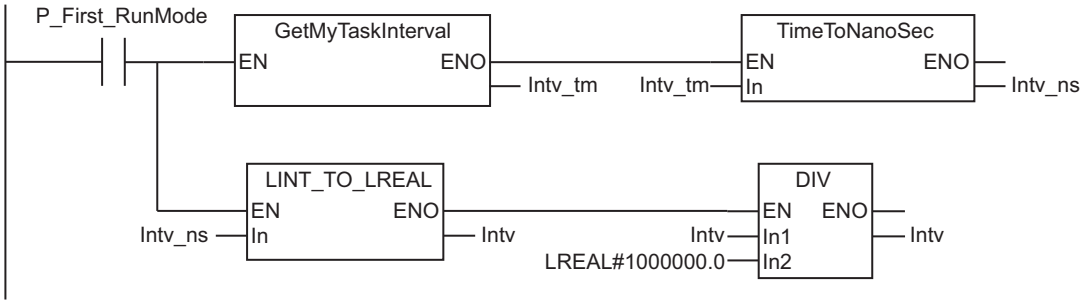
運転開始後、最初に本プログラムを実行するときに、自タスクのタスク周期を読み出します。その後、読み出したタスク周期を、TIME 型から LREAL 型の ms 単位に変換します。
タスク周期ごとの軸の目標位置を算出するためなどに使用するサンプルプログラムです。

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。

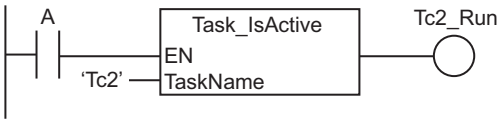
命令	名称	FB/FUN	グラフィック表現	ST 表現
Task_IsActive	タスク実行中判定	FUN	(@)Task_IsActive EN TaskName Out	Out:=Task_IsActive(TaskName);

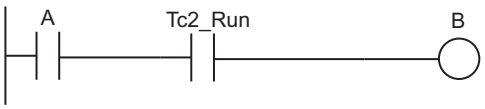
サンプルプログラム

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;
```

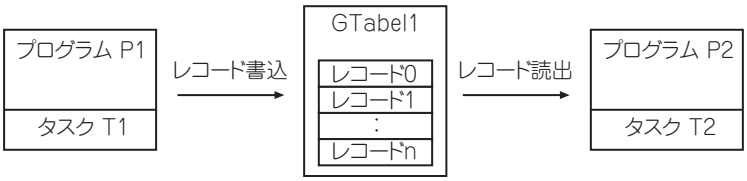
Lock / Unlock

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);

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

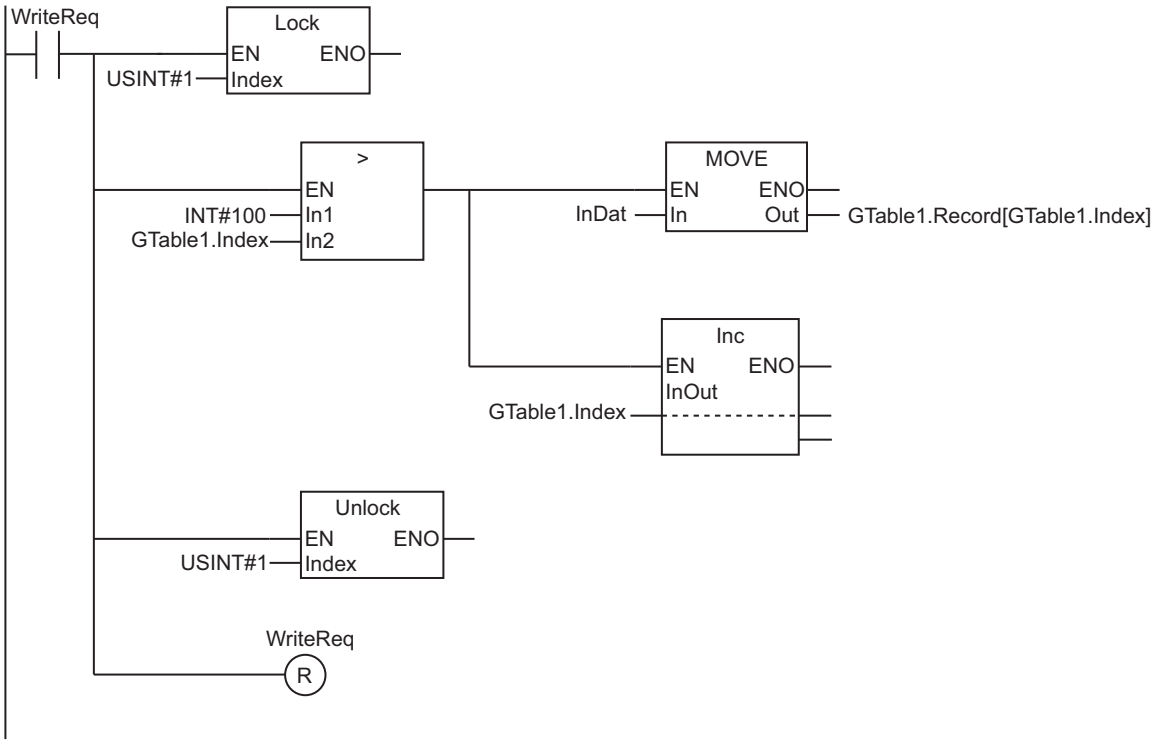
名称	データ型	初期値	コメント
GTable1	USERTABLE	(Index:=0, Record:=[100(0.0)])	レコード格納構造体

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;

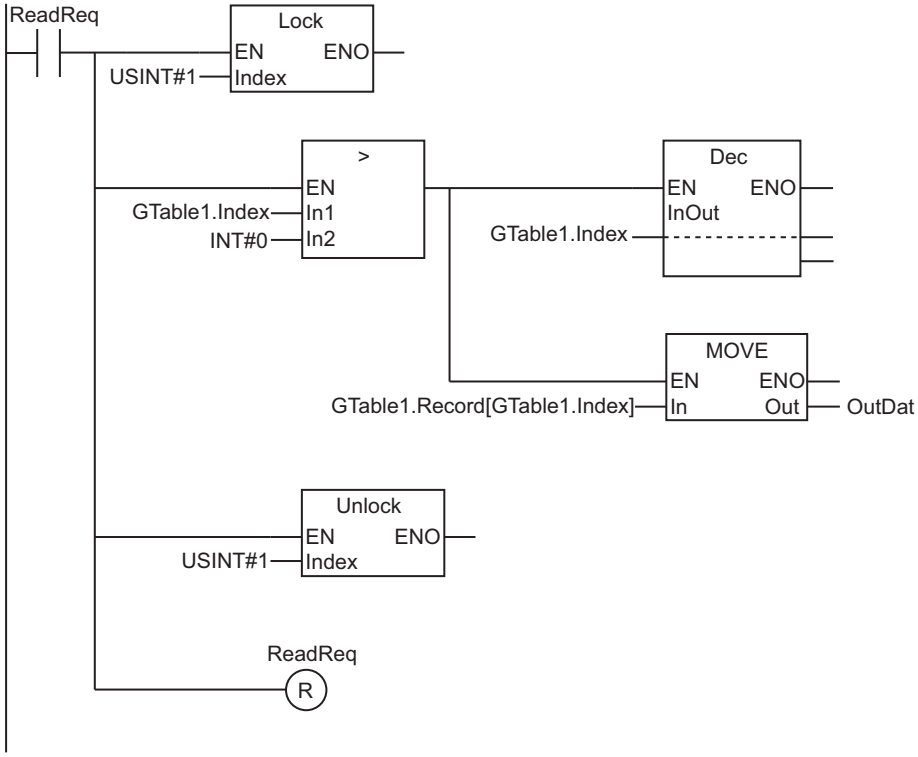
    END_IF;
```

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 EN ENO TaskName Out	ActEventTask(TaskName);

バージョン関連情報

本命令は、CPU ユニット Ver.1.03 以降、かつ Sysmac Studio Ver.1.04 以降で、使用可能です。

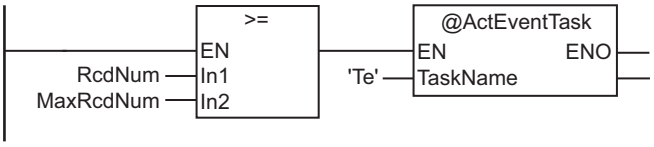
サンプルプログラム

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
MaxRcdNum	INT	100
met	BOOL	FALSE

```
IF (RcdNum>=MaxRcdNum) THEN
   IF (met=FALSE) THEN
      ActEventTask('Te');
      met:=TRUE;
   END_IF;
ELSE
```

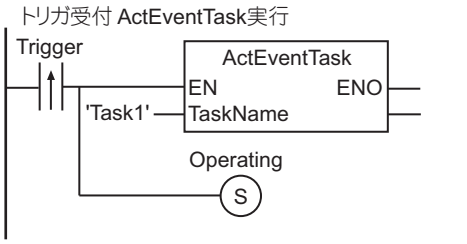
```
    met:=FALSE;
END_IF;
```

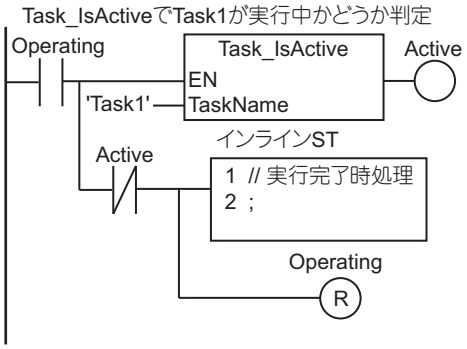
イベントタスクの実行完了を確認してから、つぎの処理を行う例

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
```

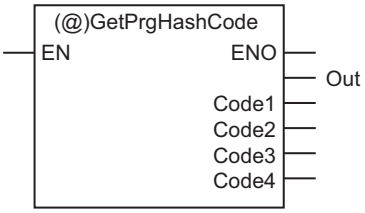
```
    Active:=Task_IsActive('Task1');

    IF (Active=FALSE) THEN        // Task1 実行完了
       Operating:=FALSE;
    END_IF;

END_IF;
```

GetPrgHashCode

ユーザプログラムに対応するプログラムのハッシュ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
GetPrgHashCode	プログラムハッシュコード取得	FUN		Out := GetPrgHashCode(Code1, Code2, Code3, Code4);

バージョン関連情報

本命令は、以下の CPU ユニットで使用可能です。
- 形 NX701 CPU ユニットは、CPU ユニット Ver.1.32 以降、かつ Sysmac Studio Ver.1.53 以降
- 形 NX502 CPU ユニットは、CPU ユニット Ver.1.60 以降、かつ Sysmac Studio Ver.1.54 以降
- NJ シリーズ、形 NX102 および形 NX1P2 の CPU ユニット は、CPU ユニット Ver.1.50 以降、かつ Sysmac Studio Ver.1.52 以降

サンプルプログラム

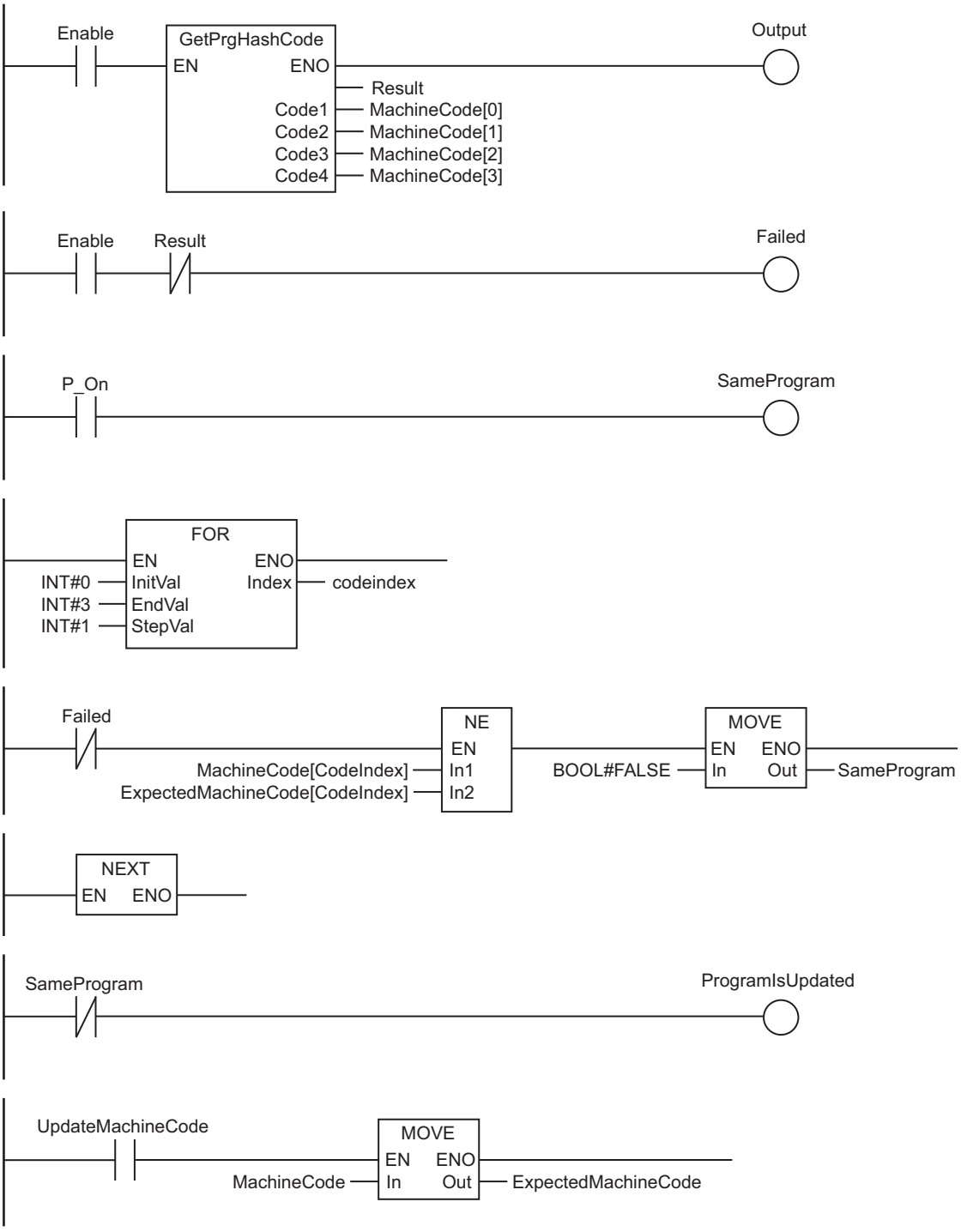
プログラムハッシュコードの変化を検出します。プログラムハッシュコードが期待値と異なる場合に「ProgramIsUpdated」を TRUE にします。

LD

「Enable」を TRUE にすると、プログラムハッシュコード取得命令を用いて「MachineCode[0]」から「MachineCode[3]」にプログラムハッシュコードを取得します。取得したプログラムハッシュコードと期待値「ExpectedMachineCode[0]」から「ExpectedMachineCode[3]」のプログラムハッシュコードを比較し、異なる場合は「ProgramIsUpdated」を TRUE にします。
取得したプログラムハッシュコードと期待値が同じ場合は「ProgramIsUpdated」を FALSE にします。
プログラムハッシュコードの取得に失敗した場合は「Failed」を TRUE にし、「MachineCode」と「ExpectedMachineCode」の比較を行いません。
「UpdateMachineCode」を TRUE にすることで、期待値「ExpectedMachineCode」を現在のプログラムハッシュコードで更新します。

内部変数	名称	データ型	初期値	コメント
	Enable	BOOL		
	Output	BOOL		
	Result	BOOL		
	ExpectedMachineCode	ARRAY[0..3] OF DWORD		
	MachineCode	ARRAY[0..3] OF DWORD		
	CodeIndex	INT		
	SameProgram	BOOL		
	ProgramIsUpdated	BOOL		

内部変数	名称	データ型	初期値	コメント
	UpdateMachineCode	BOOL		
	Failed	BOOL		



ST

「Enable」を TRUE にすると、プログラムハッシュコード取得命令を用いて「MachineCode[0]」から「MachineCode[3]」にプログラムハッシュコードを取得します。取得したプログラムハッシュコードと期待値「ExpectedMachineCode[0]」から「ExpectedMachineCode[3]」のプログラムハッシュコードを比較し、異なる場合は「ProgramIsUpdated」を TRUE にします。

取得したプログラムハッシュコードと期待値が同じ場合は「ProgramIsUpdated」を FALSE にします。
プログラムハッシュコードの取得に失敗した場合は「Failed」を TRUE にし、「MachineCode」と「ExpectedMachineCode」の比較を行いません。
「UpdateMachineCode」を TRUE にすることで、期待値「ExpectedMachineCode」を現在のプログラムハッシュコードで更新します。

内部変数	名称	データ型	初期値	コメント
	Enable	BOOL		
	Output	BOOL		
	Result	BOOL		
	ExpectedMachineCode	ARRAY[0..3] OF DWORD		
	MachineCode	ARRAY[0..3] OF DWORD		
	CodeIndex	INT		
	SameProgram	BOOL		
	ProgramIsUpdated	BOOL		
	UpdateMachineCode	BOOL		
	Failed	BOOL		

```
Result := GetPrgHashCode(
    MachineCode[0],
    MachineCode[1],
    MachineCode[2],
    MachineCode[3]
    );

Failed :=NOT(Result);

IF Failed THEN
    RETURN;
END_IF;

SameProgram :=TRUE;

IF NOT(Failed) THEN
    FOR CodeIndex := INT#0 TO INT#3 BY INT#1 DO
        IF NOT(MachineCode[CodeIndex] = ExpectedMachineCode[CodeIndex]) THEN
            SameProgram := FALSE;
        END_IF;
    END_FOR;
END_IF;

ProgramIsUpdated := NOT(SameProgram);

IF UpdateMachineCode THEN
    ExpectedMachineCode := MachineCode;
END_IF;
```

I

索引

索引

Numerics

- (減算)....1-11
-OU (減算オーバーフローチェック付き)....1-12
/ (除算)....1-16
* (乗算)....1-13
*OU (乗算 オーバーフローチェック付き)....1-15
+ (加算)....1-8
+OU (加算 オーバーフローチェック付き)....1-10
2 バイト結合....1-62
4 バイト結合....1-63

A

AC_StepProgram (ステッププログラム)....1-194
ACOS (COS^{-1} 演算)....1-18
ActEventTask (イベントタスク起動)....1-539
ADD (加算)....1-8
ADD_DT_TIME (日付時刻と時間の加算)....1-107
ADD_TIME (時間加算)....1-105
ADD_TOD_TIME (時刻と時間の加算)....1-106
AddDelimiter (デリミタ付加文字列書込)....1-90
AddOU (加算オーバーフローチェック付き))....1-10
ArySHL (配列の N 要素左シフト)....1-42
ArySHR (配列の N 要素右シフト)....1-42
ASIN (SIN^{-1} 演算)....1-18
ATAN (TAN^{-1} 演算)....1-18

B

BackupToMemoryCard (SD メモリカードバックアップ)....1-505
BCDsToBin (符号付き BCD→符号付き整数変換)....1-22
BinToBCDs_** (符号付き整数→BCD 変換グループ)....1-23
BinToGray_** (BIN コード→グレイコード変換グループ)....1-60
BIN コード→ グレイコード変換グループ....1-60

C

ChangeFTPAccount (FTP アカウント変更)....1-385
ChangeIPAdr (IP アドレス変更)....1-381
ChangeNTPServerAdr (NTP サーバ宛先変更)....1-387
ChkLeapYear (うるう年判別)....1-131
ChkRange (範囲型変数チェック)....1-528
CIP Class3 コネクション確立....1-315
CIP Class3 コネクションの切断....1-325
CIPClose (CIP Class3 コネクションの切断)....1-325
CIPOpen (CIP Class3 コネクション確立)....1-315
CIPRead (変数読出 Class3 Explicit メッセージ)....1-322
CIPSend (任意 Explicit メッセージ送信 Class3)....1-324
CIPUCMMRead (変数読出 UCMM Explicit メッセージ)....1-326
CIPUCMMSend (任意 Explicit メッセージ送信 UCMM)....1-328
CIPUCMMWrite (変数書込 UCMM Explicit メッセージ)....1-327
CIPWrite (変数書込 Class3 Explicit メッセージ)....1-323
CoE SDO 読出....1-256
CONCAT_ DATE_TOD (日付と時刻の結合)....1-116
COS (COS 演算)....1-17
COS^{-1} 演算....1-18
COS 演算....1-17

D

DateToSec (日付→秒変換)....1-122
DaysToMonth (日数→月変換)....1-134
Dec (デクリメント)....1-20
DirCreate (ディレクトリ作成)....1-503
DirRemove (ディレクトリ削除)....1-504
DIV (除算)....1-16
DIVTIME (時間除算)....1-115
DT_TO_DATE (日付時刻からの日付切出)....1-118
DT_TO_TOD (日付時刻からの時刻切出)....1-117
DtToSec (日付時刻→秒変換)....1-121

E

EC_ChangeEnableSetting (EtherCAT スレーブ有効／無効切替)....1-274
EC_CoESDORead (CoE SDO 読出)....1-256
EC_ConnectSlave (EtherCAT スレーブ再加入)....1-273
EC_CopyMon (EtherCAT パケット転送)....1-266
EC_DisconnectSlave (EtherCAT スレーブ離脱))....1-267
EC_SaveMon (EtherCAT パケット保存)....1-265
EC_StartMon (EtherCAT パケットモニタ機能開始)1-259
EC_StopMon (EtherCAT パケットモニタ機能停止)1-264
EnumToNum (列挙型→整数変換)....1-36
EtherCAT 異常解除....1-223
EtherCAT 異常ステータス取得....1-224
EtherCAT スレーブ再加入....1-273
EtherCAT スレーブ有効／無効切替....1-274
EtherCAT スレーブ離脱....1-267
EtherCAT パケット転送....1-266
EtherCAT パケット保存....1-265
EtherCAT パケットモニタ機能開始....1-259
EtherCAT パケットモニタ機能停止....1-264
ExecPMCR (プロトコルマクロ)....1-408

F

FileClose (ファイルクローズ)....1-469
FileCopy (ファイルコピー)....1-493
FileGets (文字列読出)....1-481
FileOpen (ファイルオープン)....1-468
FilePuts (文字列書込)....1-486
FileRead (ファイル読出)....1-471
FileReadVar (変数ファイル読出)....1-465
FileRemove (ファイル削除)....1-497
FileRename (ファイル名変更)....1-500

FileSeek（ファイルシーク）.....1-470
FileWrite（ファイル書込）.....1-476
FileWriteVar（変数ファイル書込）.....1-462
FTPGetFileList（FTP サーバのファイル一覧取得）.1-389
FTPGetFile（FTP サーバからのファイルダウンロード）.....1-395
FTPPu t File（FTP サーバへのファイルアップロード）.....1-396
FTPRemoveDir（FTP サーバのディレクトリ削除）.1-406
FTPRemoveFile（FTP サーバのファイル削除）.....1-401
FTP アカウント変更.....1-385
FTP サーバからのファイルダウンロード.....1-395
FTP サーバのディレクトリ削除.....1-406
FTP サーバのファイル削除.....1-401
FTP サーバのファイルリスト取得.....1-389
FTP サーバへのファイルアップロード.....1-396

G

GetDayOfWeek（曜日取得）.....1-135
GetDaysOfMonth（月の日数取得）.....1-132
GetECError（EtherCAT 異常ステータス取得）.....1-224
GetMCError（モーション制御異常ステータス取得）..1-222
GetMyTaskInterval（自タスク設定周期読出）.....1-532
GetMyTaskStatus（自タスク状態読出）.....1-530
GetNXUnitError（NX ユニット異常ステータス取得）.1-225
GetPrgHashCode（プログラムハッシュコード取得）1-542
GetTime（時刻取得）.....1-120
GetTraceStatus（データトレースステータス読出）...1-209
Gray（グレイコード変換）.....1-50
GrayToBin_**（グレイコード→BIN コード変換グループ）..1-60

I

Inc（インクリメント）.....1-20
IO-Link デバイスオブジェクト書込.....1-308
IO-Link デバイスオブジェクト読出.....1-304
IOL_ReadObj（IO-Link デバイスオブジェクト読出）1-304
IOL_WriteObj（IO-Link デバイスオブジェクト書込）1-308
IPC_GetOSStatus(OS ステータス読み出し).....1-520
IPC_RebootOS(OS 再起動).....1-522
IPC_Shutdown(シャットダウン).....1-524
IP アドレス変更.....1-381

L

LEFT（文字列・左からの取出）.....1-86
LimitAlarm_**（上下限警報グループ）.....1-178
LimitAlarmDv_**（上下限偏差警報グループ）.....1-179
LimitAlarmDvStbySeq_**（待機シーケンス付き上下限偏差警報グループ）.....1-180
LN（自然対数演算）.....1-19
Lock（タスク間排他ロック）.....1-535
LOG（常用対数演算）.....1-19
LOWER_BOUND （配列開始番号取得）.....1-65
LrealToFormatString（実数 LREAL→文字列変換書式付き）.....1-30

M

Modbus RTU Read コマンド送信.....1-443
Modbus RTU Write コマンド送信.....1-446
Modbus RTU 汎用コマンド送信.....1-440
Modbus TCP Read コマンド送信.....1-374
Modbus TCP Write コマンド送信.....1-378
Modbus TCP 汎用コマンド送信.....1-370
ModbusTCPCmd（Modbus TCP 汎用コマンド送信）.....1-370
ModbusTCPRead （Modbus TCP Read コマンド送信）.....1-374
ModbusTCPWrite （Modbus TCP Write コマンド送信）.....1-378
MovingAverage（移動平均）.....1-55
MUL（乗算）.....1-13
MulOU（乗算 オーバーフローチェック付き）.....1-15
MULTIME（時間乗算）.....1-114

N

NanoSecToTime（ナノ秒→時間変換）.....1-129
NSHLC（N ビットデータ N ビット左シフト CY 付き）.1-45
NSHRC（N ビットデータ N ビット右シフト CY 付き）.1-45
NTP サーバ宛先変更.....1-387
NumToDecString（固定長 10 進文字列変換）.....1-58
NumToEnum（整数→列挙型変換）.....1-38
NumToHexString（固定長 16 進文字列変換）.....1-58
NX_AryDOutTimeStamp(タイムスタンプデジタル出力配列書込).....1-517
NX_ChangeWriteMode(NX ユニット書込モード変更).....1-231
NX_DOutTimeStamp(タイムスタンプデジタル出力書込)...1-514
NX_ModbusRtuCmd（Modbus RTU 汎用コマンド送信）.....1-440
NX_ModbusRtuRead（Modbus RTU Read コマンド送信）.....1-443
NX_ModbusRtuWrite（Modbus RTU Write コマンド送信）.....1-446
NX_ReadObj(NX オブジェクト読出).....1-299
NX_ReadTotalPowerOnTime（NX ユニット累積通電時間読出）.....1-235
NX_SaveParam(NX ユニットパラメータ保存).....1-232
NX_SerialBufClear（バッファクリア）.....1-455
NX_SerialRcv（無手順データの受信）.....1-435
NX_SerialSend（無手順データの送信）.....1-430
NX_SerialSigCtl（シリアル制御信号の ON/OFF 切り替え）.....1-449
NX_SerialSigRead（シリアル制御信号の読出）.....1-453
NX_SerialStatusRead（シリアルポートステータスの読出）.....1-454
NX_WriteObj(NX オブジェクト書込).....1-289
NX オブジェクト書込.....1-289
NX オブジェクト読出.....1-299
NX ユニット異常ステータス取得.....1-225
NX ユニット書込モード変更.....1-231
NX ユニットパラメータ保存.....1-232
NX ユニットリスタート.....1-230

NX ユニット累積通電時間読出.....1-235
N ビットデータ N ビット左シフト CY 付き.....1-45
N ビットデータ N ビット右シフト CY 付き.....1-45
N ビット左回転.....1-46
N ビット左シフト.....1-43
N ビット右回転.....1-46
N ビット右シフト.....1-43

O

OS 再起動.....1-522
OS ステータス読み出し.....1-520

P

PackDword (4 バイト結合).....1-63
PackWord (2 バイト結合).....1-62
PIDAT (オートチューニング付き PID 演算).....1-146
PIDAT_HeatCool.....1-152
PLC_ReadTotalPowerOnTime（PLC 累積通電時間読出）.....1-233
PLC 異常解除.....1-215
PLC 累積通電時間読出.....1-233
PrgStart（プログラム実行指令）.....1-240
PrgStatus (プログラム状態読出).....1-252
PrgStop (プログラム停止指令).....1-241
PWLApprox (折れ線近似変換（折れ線チェックあり))1-51
PWLApproxNoLineChk (折れ線近似変換（折れ線チェックなし)).....1-51
PWLDatChk（折れ線データチェック).....1-53

R

RealToFormatString (実数 REAL→文字列変換書式付き).....1-26
RecMax (レコード最大値検索).....1-83
RecMin (レコード最小値検索).....1-83
RecSort (レコードソート).....1-79
ResetAlarm (ユーザ異常解除).....1-214
ResetECError (EtherCAT 異常解除).....1-223
ResetMCError (モーション制御異常解除).....1-218
ResetPLCError (PLC 異常解除).....1-215
ResetUnit (ユニットリスタート).....1-227
RestartNXUnit（NX ユニットリスタート）.....1-230
RIGHT (文字列・右からの取出).....1-86
ROL (N ビット左回転).....1-46
ROR (N ビット右回転).....1-46

S

ScaleTrans(スケ-ル変換).....1-192
SD メモリカードバックアップ.....1-505
SecToDate (秒→日付変換).....1-125
SecToDt (秒→日付時刻変換).....1-124
SecToTime (秒→時間変換).....1-130
SecToTod (秒→時刻変換).....1-126
SendCmd (コマンド送信).....1-424
SerialRcv (シリアルコミュニケーションユニット シリアルポート入力).....1-419
SerialRcv N oClear (シリアルコミュニケーションユニット シリアルポート入力受信バッファ非消去).....1-419
SetAlarm (ユーザ異常発生).....1-211
SetTime (時計補正).....1-119
SHL (N ビット左シフト).....1-43
SHR (N ビット右シフト).....1-43
SIN (SIN 演算).....1-17
SIN^{-1} 演算.....1-18
SIN 演算.....1-17
SktClearBuf (TCP/UDP ソケット受信バッファクリア).....1-353
SktClose (TCP/UDP ソケットクローズ).....1-352
SktGetTCPStatus (TCP ソケットのステータス読出).....1-351
SktSetOption(TCP ソケットオプション設定).....1-354
SktTCPAccept (TCP ソケットアクセプト).....1-342
SktTCPConnect (TCP ソケットコネクト).....1-343
SktTCPRcv (TCP ソケット受信).....1-349
SktTCPSend (TCP ソケット送信).....1-350
SktTLSClearBuf(TLS セッション受信バッファクリア).....1-367
SktTLSConnect(TLS セッション確立).....1-357
SktTLSDisconnect(TLS セッション切断).....1-366
SktTLSRead(TLS 受信).....1-364
SktTLSStopLog(セキュアソケット通信ログ停止).....1-368
SktTLSWrite(TLS 送信).....1-365
SktUDPCreate (UDP ソケットクリエイト).....1-334
SktUDPRcv (UDP ソケット受信).....1-340
SktUDPSend (UDP ソケット送信).....1-341
StackFIFO (先入れ先出し).....1-77
StackIns (スタックデータ挿入).....1-78
StackLIFO (後入れ先出し).....1-77
StackPush (スタックデータ格納).....1-70
STRING_TO_** (文字列→実数変換グループ).....1-34
SUB (減算).....1-11
SUB_DATE_DATE (日付減算).....1-111
SUB_DT_DT (日付時刻減算).....1-112
SUB_DT_TIME (日付時刻と時間の減算).....1-113
SUB_TIME (時間減算).....1-108
SUB_TOD_TIME (時刻と時間の減算).....1-109
SUB_TOD_TOD (時刻減算).....1-110
SubDelimiter（文字列読出デリミタ削除）.....1-97
SubOU (減算オーバーフローチェック付き).....1-12

T

TAN (TAN 演算).....1-17
TAN^{-1} 演算.....1-18
TAN 演算.....1-17
Task_IsActive (タスク実行中判定).....1-534
TCP/UDP ソケットクローズ.....1-352
TCP/UDP ソケット受信バッファクリア.....1-353
TCP ソケットアクセプト.....1-342
TCP ソケットオプション設定.....1-354
TCP ソケットコネクト.....1-343
TCP ソケット受信.....1-349
TCP ソケット送信.....1-350
TCP ソケットのステータス読出.....1-351
TimeProportionalOut（時分割比例出力）.....1-165

TimeToNanoSec (時間→ナノ秒変換).....1-127
TimeToSec (時間→秒変換).....1-128
TLS 受信.....1-364
TLS セッション確立.....1-357
TLS セッション受信バッファクリア.....1-367
TLS セッション切断.....1-366
TLS 送信.....1-365
TodToSec (時刻→秒変換).....1-123
ToLCase (文字列・小文字変換).....1-88
TON (オンディレータイマ).....1-4
ToUCase (文字列・大文字変換).....1-88
TraceSamp (データトレースサンプリング).....1-206
TraceTrig (データトレーストリガ条件成立).....1-208
TrimL (文字列・左トリム).....1-89
TrimR (文字列・右トリム).....1-89
TruncDt (日付時刻切り捨て).....1-139
TruncTime (時間切り捨て).....1-136
TruncTod (時刻切り捨て).....1-142

U

UDP ソケットクリエイト.....1-334
UDP ソケット受信.....1-340
UDP ソケット送信.....1-341
Unlock (タスク間排他ロック解除).....1-535
UPPER_BOUND （配列終了番号取得）.....1-65

あ

後入れ先出し.....1-77

い

移動平均.....1-55
イベントタスク起動.....1-539
インクリメント.....1-20

う

うるう年判別.....1-131

お

折れ線近似変換（折れ線チェックあり）.....1-51
折れ線近似変換（折れ線チェックなし）.....1-51
折れ線データチェック.....1-53
オンディレータイマ.....1-4
オートチューニング付き PID 演算.....1-146
オートチューニング付き加熱冷却 PID 演算.....1-152

か

加算.....1-8
加算（オーバーフローチェック付き）.....1-10

く

グレイコード→BIN コード変換グループ.....1-60
グレイコード変換.....1-50

け

減算.....1-11
減算（オーバーフローチェック付き）.....1-12

こ

固定長 10 進文字列変換.....1-58
固定長 16 進文字列変換.....1-58
コマンド送信.....1-424

さ

先入れ先出し.....1-77

し

時間加算.....1-105
時間切り捨て.....1-136
時間減算.....1-108
時間乗算.....1-114
時間除算.....1-115
時間→ナノ秒変換.....1-127
時間→秒変換.....1-128
時刻切り捨て.....1-142
時刻減算.....1-110
時刻取得.....1-120
時刻と時間の加算.....1-106
時刻と時間の減算.....1-109
時刻→秒変換.....1-123
自然対数演算.....1-19
自タスク状態読出.....1-530
自タスク設定周期読出.....1-532
実数(LREAL）→文字列変換書式付き.....1-30
実数(REAL）→文字列変換書式付き.....1-26
時分割比例出力.....1-165
シャットダウン.....1-524
上下限警報グループ.....1-178
上下限偏差警報グループ.....1-179
乗算.....1-13
乗算（オーバーフローチェック付き）.....1-15
常用対数演算.....1-19
除算.....1-16
SerialSend (シリアルコミュニケーションユニット シリアルポート出力).....1-415
シリアルコミュニケーションユニット シリアルポート出力.....1-415
シリアルコミュニケーションユニット シリアルポート入力.....1-419
シリアルコミュニケーションユニット シリアルポート入力受信バッファ非消去.....1-419
シリアル制御信号の ON/OFF 切り替え.....1-449
シリアル制御信号の読出.....1-453
シリアルポートステータスの読出.....1-454

す

スケール変換.....1-192
スタックデータ格納.....1-70
スタックデータ挿入.....1-78

ステッププログラム....1-194

せ

整数→列挙型変換....1-38
セキュアソケット通信ログ停止....1-368

た

待機シーケンス付き上下限偏差警報グループ....1-180
タイムスタンプデジタル出力書込....1-514
タイムスタンプデジタル出力配列書込....1-517
タスク間排他ロック....1-535
タスク間排他ロック解除....1-535
タスク実行中判定....1-534

つ

月の日数取得....1-132

て

ディレクトリ削除....1-504
ディレクトリ作成....1-503
デクリメント....1-20
デリミタ付加文字列書込....1-90
データトレースサンプリング....1-206
データトレースステータス読出....1-209
データトレーストリガ条件成立....1-208

と

時計補正....1-119

な

ナノ秒→時間変換....1-129

に

日数→月変換....1-134
任意 Explicit メッセージ送信(Class3)....1-324
任意 Explicit メッセージ送信(UCMM)....1-328

は

配列開始番号取得....1-65
配列終了番号取得....1-65
配列の N 要素左シフト....1-42
配列の N 要素右シフト....1-42
バッファクリア....1-455
範囲型変数チェック....1-528

ひ

日付減算....1-111
日付時刻からの時刻切出....1-117
日付時刻からの日付切出....1-118
日付時刻切り捨て....1-139
日付時刻減算....1-112
日付時刻と時間の加算....1-107
日付時刻と時間の減算....1-113
日付時刻→秒変換....1-121
日付と時刻の結合....1-116
日付→秒変換....1-122
秒→時間変換....1-130
秒→時刻変換....1-126
秒→日付時刻変換....1-124
秒→日付変換....1-125

ふ

ファイルオープン....1-468
ファイル書込....1-476
ファイルクローズ....1-469
ファイルコピー....1-493
ファイル削除....1-497
ファイルシーク....1-470
ファイル名変更....1-500
ファイル読出....1-471
符号付き BCD→符号付き整数変換....1-22
符号付き整数→BCD 変換グループ....1-23
プログラム実行指令....1-240
プログラム状態読出....1-252
プログラム停止指令....1-241
プログラムハッシュコード取得....1-542
プロトコルマクロ....1-408

へ

変数書込 (Class3 Explicit メッセージ)....1-323
変数書込 (UCMM Explicit メッセージ)....1-327
変数ファイル書込....1-462
変数ファイル読出....1-465
変数読出 (Class3 Explicit メッセージ)....1-322
変数読出 (UCMM Explicit メッセージ)....1-326

む

無手順データの受信....1-435
無手順データの送信....1-430

も

文字列・ 大文字変換....1-88
文字列書込....1-486
文字列・ 小文字変換....1-88
文字列→実数変換グループ....1-34
文字列・ 左からの取出....1-86
文字列・ 左トリム....1-89
文字列・ 右からの取出....1-86
文字列・ 右トリム....1-89
文字列読出....1-481
文字列読出デリミタ削除....1-97
モーション制御異常解除....1-218
モーション制御異常ステータス取得....1-222

ゆ

ユニットリスタート....1-227
ユーザ異常解除....1-214

ユーザ異常発生..... 1-211

よ

曜日取得..... 1-135

れ

レコード最小値検索..... 1-83
レコード最大値検索..... 1-83
レコードソート..... 1-79
列挙型→整数変換..... 1-36

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**

本誌には主に機種のご選定に必要な内容を掲載しており、ご使用上の注意事項等を掲載していない製品も含まれています。
本誌に注意事項等の掲載のない製品につきましては、ユーザーズマニュアル掲載のご使用上の注意事項等、ご使用の際に必要な内容を必ずお読みください。

●本誌に記載の標準価格はあくまで参考であり、確定されたユーザ購入価格を表示したものではありません。本誌に記載の標準価格には消費税が含まれておりません。
●本誌にオープン価格の記載がある商品については、標準価格を決めていません。
●本誌に記載されているアプリケーション事例は参考用ですので、ご採用に際しては機器・装置の機能や安全性をご確認の上、ご使用ください。
●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。
●規格認証/適合対象機種などの最新情報につきましては、
当社Webサイト（www.fa.omron.co.jp）の「規格認証/適合」をご覧ください。

オムロン商品のご用命は

カタログ番号 SBCA-488B　2024年10月現在

©OMRON Corporation 2021-2024 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