BA-CDS35

CODESYS

スタートアップ

マニュアル

MG CO., LTD.
www.mgco.jp

NM-7290-C 3版

(このページは空白です)

はじめに

本マニュアルは、Programming Automation Controller(以下、「本製品」および「コントローラ」)に搭載のIEC61131-3準拠のソフトロジックについて説明しています。

ご使用になる前に本書をよくお読み頂き、正しくお使い下さい。

なお、本マニュアルは、IEC61131-3仕様を理解されていることを前提に作成されています。用語については、それぞれの文献を参照して下さい。

使用者

本製品の使用者はオートメーションに十分熟知した電気機器の専門技術者でPLCプログラムの知識が十分にあるものとします。

マニュアルについて

本マニュアルに記載されている記号、および共通注意事項は以下のとおりです。

記号説明

警告

取扱いを誤った場合に危険な状況が起こりえて死亡または重傷を受ける可能性が想定されることを示しています。

注意

取扱いを誤った場合に危険な状況が起こりえて中程度の傷害や軽傷を受ける可能性が想定される場合および物的損害だけの発生が想定されることを示しています。この注意に記載した事項でも状況により重大な結果に結びつく可能性があります。

補足

操作時のヒント、追加情報や補足事項を記載しています。

いずれも重要な内容を記載していますので厳守してください。

本マニュアルは必要なときに読めるよう大切に保管すると共に必ず最終ユーザまでお届けいただくようお願いいたします。

安全上のご注意

(ご使用の前に必ずお読みください)

本製品のご使用の際には本マニュアルおよび関連マニュアルをよくお読みいただき安全に対しての十分な注意と配慮、および正しい取扱いをしていただくようお願いいたします。

設計上の注意事項

警 告

- フィールドバスを含むネットワークが交信異常になったときの動作状態についてはそのネットワークに関連するマニュアルを参照してください。誤出力や誤動作により事故の恐れがあります。
- インターネット経由の外部機器からの不正アクセスに対してコントローラの安全を保つ必要があるときはユーザによる対策を盛り込んでください。
- 運転中のユーザアプリケーションやデータを変更するときは常時システム全体が安全側に働くようにユーザアプリケーション上でインターロック回路を構成してください。またユーザアプリケーションの変更、パラメータ変更や運転状態の変更を行うときは関連するマニュアルを熟読し十分に安全を確認してから行ってください。
- FLAGSエリアのユーザ使用可能領域以外の領域に対するデータ書込みを行うとコントローラが誤動作する危険性があります。

取付け上の注意事項

注 意

- 本製品や使用するIOカードはそれぞれに用意されたマニュアルに記載されている環境にて使用してください。それ以外の環境で使用すると感電、火災、誤動作、製品の損傷あるいは劣化の原因になります。
- 本製品やIOカードの着脱は必ずシステムで使用している外部供給電源を全相遮断してから行ってください。全相遮断しないと製品の損傷の恐れがあります。
- 本製品やIOカードの導電部分や電子部品には直接触らないでください。製品の誤動作や故障の原因になり

ます。

配線上の注意事項

注 意

- 外部接続用コネクタはメーカ指定の工具で正しく圧着、圧接またはハンダ付けをしてください。接続が不完全な場合は短絡、火災、誤動作の原因になります。
- 本製品やIOカードに接続する通信ケーブルや電源ケーブルはダクトに納めるかクランプにより固定処理を行ってください。ケーブルがダクトに納められなかったりクランプによる固定処理をされないとケーブルのふらつき、移動や不注意の引っ張りなどによる製品やケーブルの破損あるいはケーブルの接続不良による誤動作の原因となります。

保守時の注意事項

注 意

- 製品の分解や改造はしないでください。故障、誤動作、ケガ、火災の原因になります。
- カードの着脱は必ずシステムで使用している外部供給電源を全相遮断してから行ってください。全相遮断しないとカードの故障や誤動作の原因になります。
- 通電中に端子に触れないでください。誤動作の原因になります。
- 清掃、端子ネジの増し締めは必ずシステムで使用している外部供給電源を全相遮断してから行ってください。全相遮断しないとカードの故障や誤動作の原因になります。
- カードに触れる前には必ず接地された金属などに触れて人体などに帯電している静電気を放電してください。静電気を放電しないとカードの故障や誤動作の原因になります。

運転時の注意事項

注 意

- 運転中のユーザアプリケーション変更、データ変更や運転状態の変更を行うときは十分に安全を確認してから行ってください。ユーザアプリケーション変更、データ変更、運転状態の変更を誤るとシステムの誤動作や機械の破損や事故の原因になります。

製品の適用について

1. 本製品をご使用にあたり万一本製品に故障・不具合などが発生したとしても重大な事故にいたらない用途であり、故障・不具合発生時にはバックアップやフェールセーフ機能が本製品の外部でシステム的に実施されていることを使用の条件とさせていただきます。
2. 本製品は一般工業などへの用途を対象とした汎用品として設計・製作されています。
   依って以下のような機器やシステムなどの特殊用途への適用を除外させていただきます。万一使用された場合は弊社として製品の品質、性能、安全に関る一切の責任(債務不履行責任、瑕疵担保責任、品質保証責任、不法行為責任、製造物責任を含むがそれらに限定されない)を負わないものとさせていただきます。
   - 各電力会社の原子力発電所およびその他発電所向けなどの公共への影響が大きい用途
   - 鉄道各社および官公庁などの特別な品質保証体制の構築を弊社にご要求になる用途
   - 航空宇宙、医療、鉄道、燃焼・燃料装置、乗用移動体、有人搬送装置、娯楽機械、安全機械など生命、身体や財産に大きな影響が予測される用途

注 意

- 本書の内容に関しては、改良のために予告なしに仕様等変更することがありますのでご了承ください。
- 本書の内容の一部または全部を無断で複写、複製、転載することを禁じます。
- 本書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付きのことがありましたら、お手数ですが巻末記載宛てまでご連絡ください。

目 次

1.IEC61131-3

国際規格IEC61131は、PLCのハードウェアからプログラミングシステムまでを包含する規格です。

1. 一般情報
2. 装置への要求事項および試験
3. プログラミング言語
4. ユーザガイドライン
5. メッセージングサービス仕様

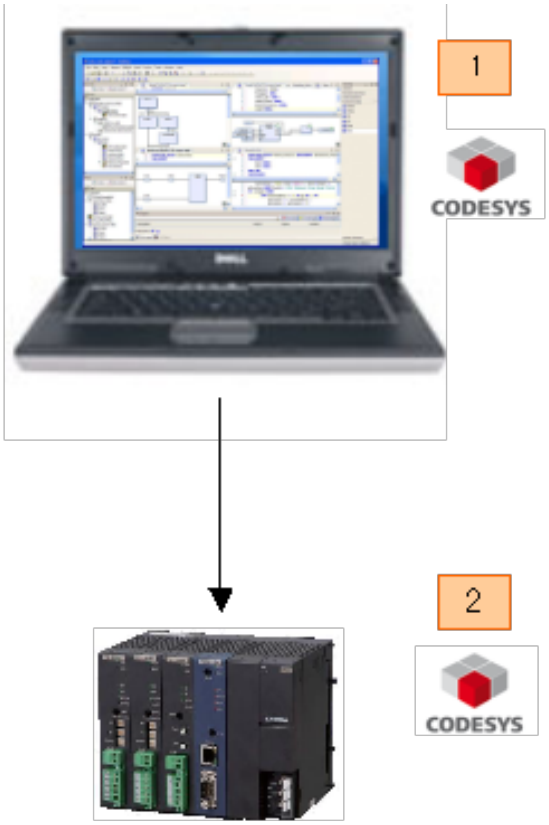
このような構成で、IEC61131-3は、その第３部を指しています。

(このページは空白です)

2.CODESYS 開発環境

2.1.プログラミングツールCODESYSについて

「CODESYS」は国際規格IEC61131-3に準拠したプログラミングツール(プログラミングシステム)です。このツールではプログラミングエディタ、HMI開発環境、オンラインデバッグ機能が統合されています。



1	統合開発環境 (IDE System)	プログラミングエディタ コンパイラ, HMIエディタ, デバイス設定, オンラインデバッグ
2	ターゲット (Target Runtime System)	ソフトロジック フィールドバス (Modbus, BACnet, LonWorks など)

2.2.動作環境

CODESYS IDE (Automation Platform) の動作には次の環境を必要とします。

リソース	必要条件
OS	Windows 7/8 (32 / 64 ビット) Windows 10 (32 / 64 ビット) *1
RAM	1GB(32ビット) または 2GB(64ビット)以上
ハードディスク空き容量	1GB 以上
画面解像度	1024 x 768 以上
CPU	1GHz Pentium 以上

*1) CODESYS V3.5SP8 以降が対応しています。

2.3.インストール

CODESYSで開発を行うためには次のソフトウェアをインストールする必要があります。

- 開発環境はエディタなどを統合したCODESYS IDE (CODESYS IDEのインストール)
- 機種別の設定や機能をまとめて提供するパッケージ (PACKAGEのインストール)
- 別途提供されたライブラリ(LIBRARYのインストール) (必須ではありません。機種ごとに必要なライブラリはパッケージで提供されていますので、別途提供されるライブラリに関しては必要に応じてインストールしてください。)

CODESYS IDE のインストール

インストーラを実行し、ガイダンスに従いインストールを進めます。

(V3.5 SP12 以前のインストーラファイル名の例)

32bit版 Setup_CODESYSV35SP12Patch7.exe
64bit版 Setup_CODESYSV35SP12Patch7_x64.exe

(V3.5 SP13 以降のインストーラファイル名の例)

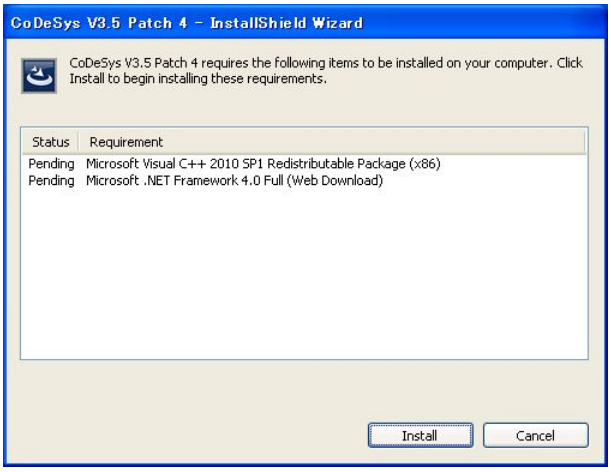
32bit版 CODESYS 3.5.13.20.exe

64bit版 CODESYS 64 3.5.13.20.exe

補 足

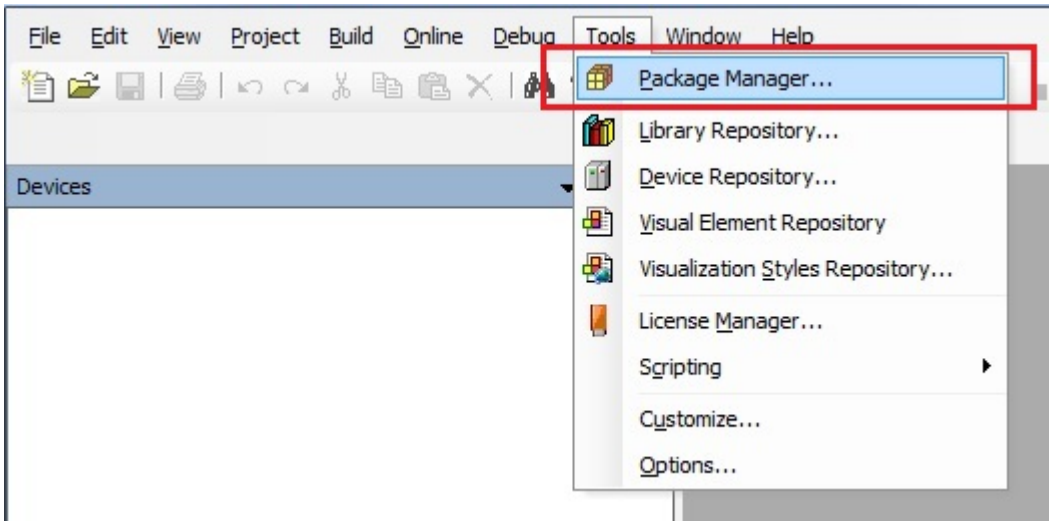
- 32bit版OSでは、32bit版 CODESYS のみが使用でき、64bit版OSでは、32bit/64bit両方のCODESYSが使用できます。
- 画面例は、お使いの CODESYS のバージョンと異なる場合があります。お使いのバージョンに適宜に読み替えて読み進めてください。

次のダイアログが表示される場合があります。この画面ではソフトウェアの動作に必要な追加のソフトウェアをインストールします。「Install」を押してインストールを進めてください。

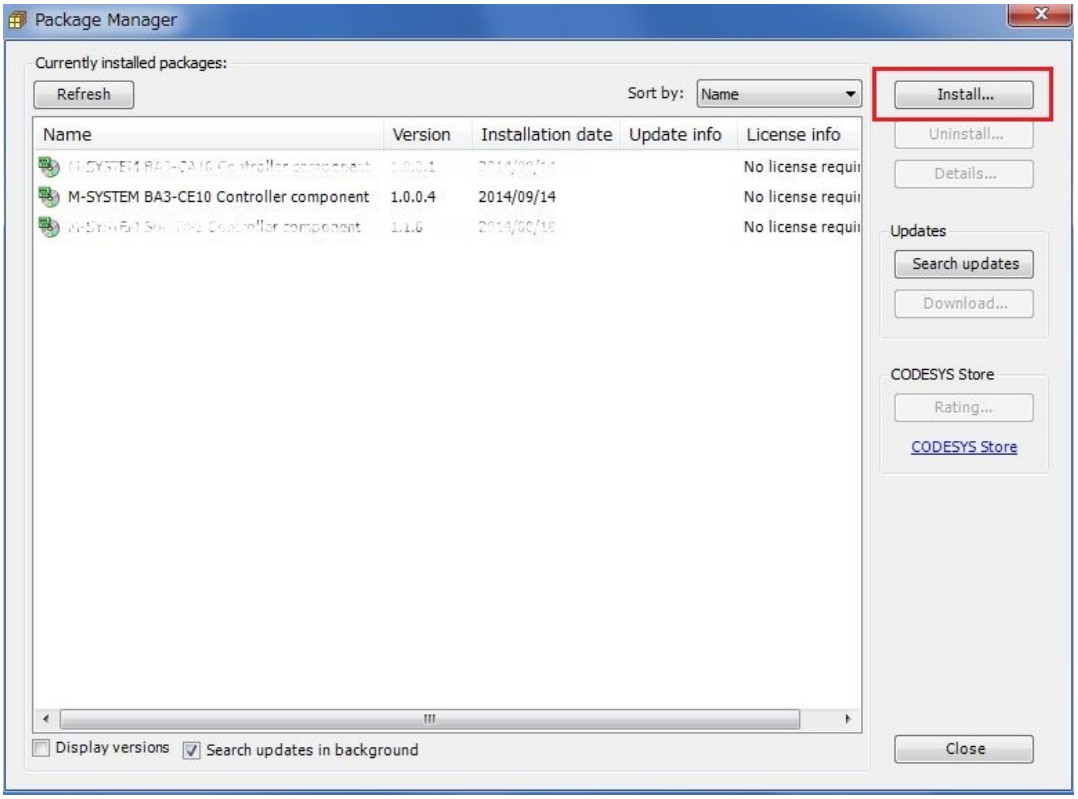


PACKAGEのインストール

CODESYS IDE のメニュー [Tools] [Package Manager...] を選択し、ガイダンスに従いインストールを進めます。



[Package Manager] ダイアログの [Install ...] ボタンを押すとファイル選択ダイアログが表示されます。インストールするパッケージ ファイルを選択してインストールを進めてください。

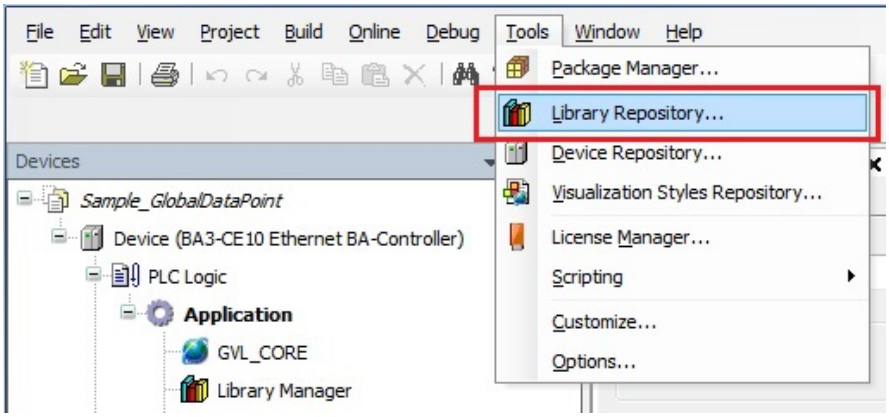


補 足

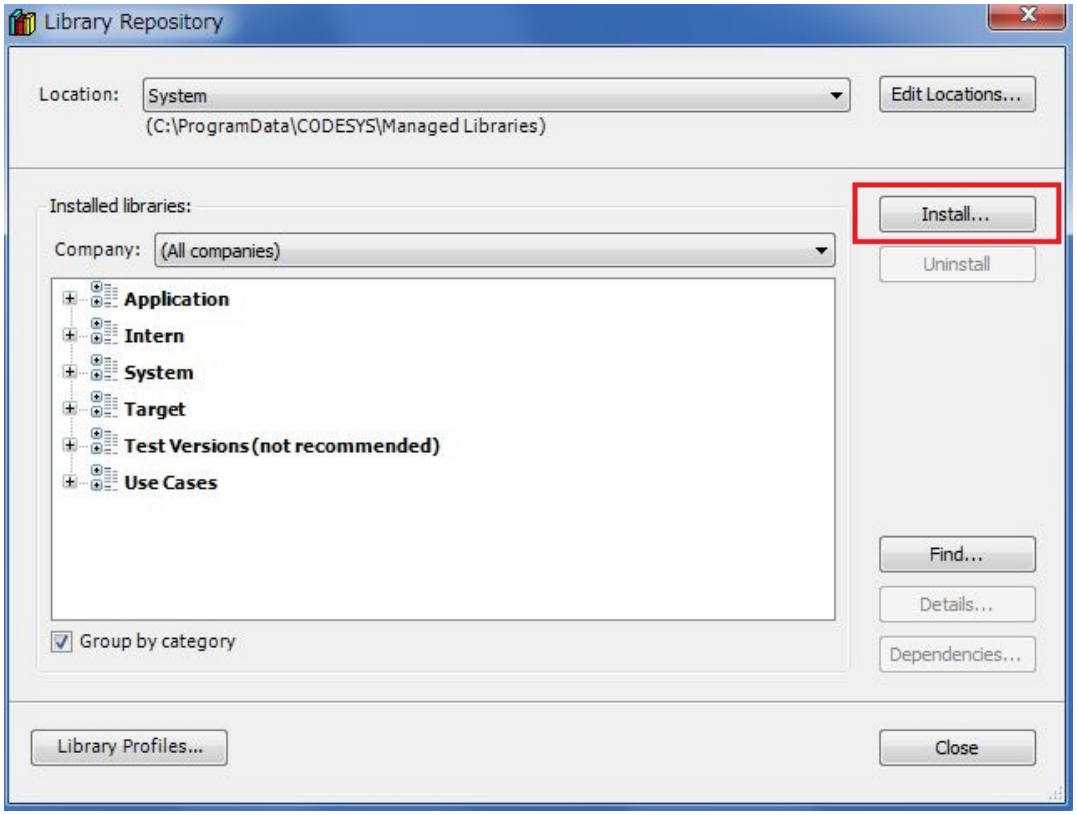
このソフトウェアには .NET 4.5 が必要です。古い状態の PC (古いバージョンのOSをお使いの場合や Windows Update の行われていない状態の場合) は complete setupを必要とする場合があります。その場合は、他の既にインストール済みのアプリケーションに悪影響を与える可能性がありますのでその判断は自己責任でお願いします。

LIBRARYのインストール

CODESYS IDE のメニュー [Tools] [Library Repository...] を選択し、ガイダンスに従いインストールを進めます。



[Library Repository] ダイアログの [Install ...] ボタンを押すとファイル選択ダイアログが表示されます。インストールするライブラリファイルを選択してインストールを進めてください。



2.4.アンインストール

ここではCODESYS IDE 本体やCODESYS IDE にインストールされているソフトウェアのアンインストール方法を説明します。

- CODESYS IDE 本体(CODESYS IDEのアンインストール)
- 機種別の設定や機能をまとめて提供するパッケージ (PACKAGEのアンインストール)
- 別途提供されたライブラリ(LIBRARYのアンインストール)

CODESYS IDE のアンインストール

インストーラを実行し、インストールウィザードの最初のダイアログにあるオプション「Remove all installed features」を選択し、「Next」を押します。あるいはWindowsの「プログラムの追加と削除」からもアンインストールを行えます。

PACKAGEのアンインストール

CODESYS IDE のメニュー [Tools] [Package Manager...] を選択します。ここで表示されるダイアログの現在インストールされているパッケージ一覧からアンインストールしたいパッケージを選択し [Uninstall...] ボタンを押すと開始されます。

LIBRARYのアンインストール

CODESYS IDE のメニュー [Tools] [Library Repository...] を選択します。ここで表示されるダイアログの現在インストールされているライブラリー覧からアンインストールしたいライブラリを選択し [Uninstall...] ボタンを押すと開始されます。表示されているライブラリー覧は [Company] や [Group by category] でフィルタリングされています。目的のライブラリが見つからない場合は適切なフィルタリングを選択してください。

2.5.スタートと画面説明

ここでは次の項目について説明しています。

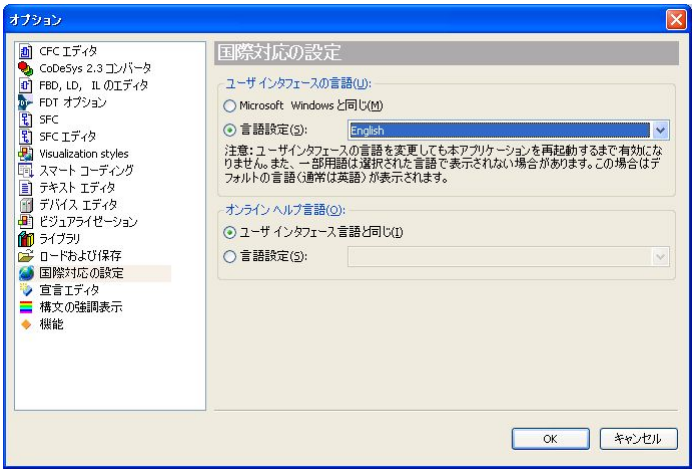
CODESYSの起動

Windowsのスタートメニュー(CODESYS 3.5SP12 Patch7 64bit版 の場合)

[CODESYS] → [CODESYS 64] → [CODESYS V3.5 SP12 Patch 7]

を選ぶか、インストール時にデスクトップに作成された以下のショートカットからも起動できます。

CODESYSを初めて起動した際の設定



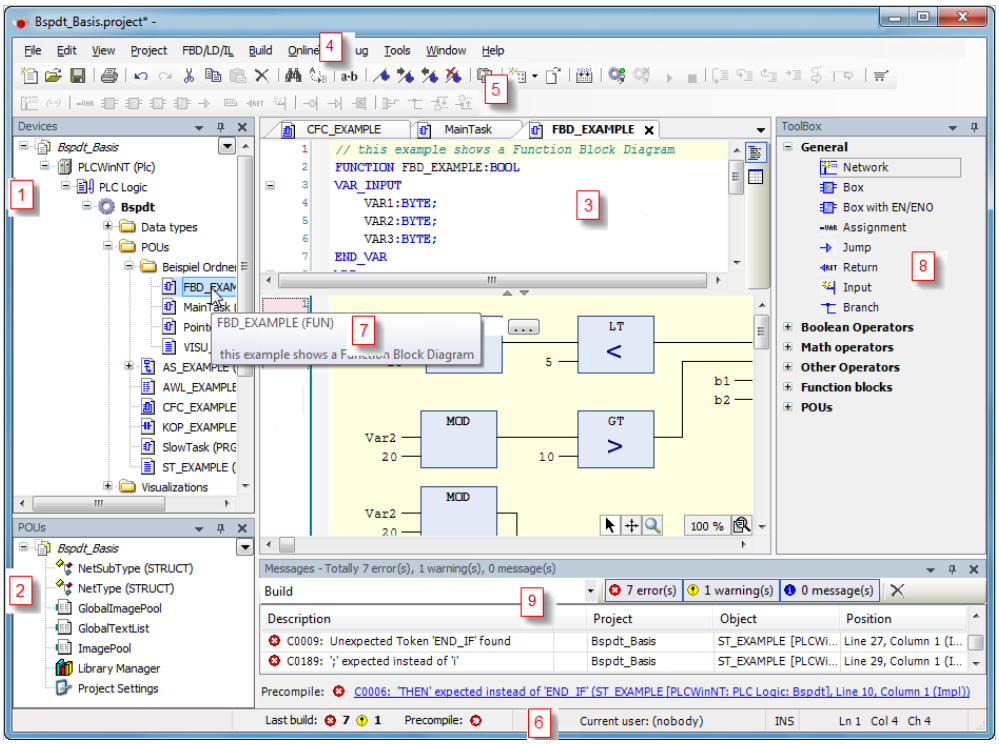
現時点の日本語表示では、まれにダイアログの日本語表示で文字が途中までしか表示されなかったり、押しボタンが表示されない場合があります。そのためユーザインタフェースの言語設定を「English」にされることをお勧めします。

変更方法は[ツール] → [オプション] → [国際対応の設定] の [ユーザインタフェースの言語]

言語設定「English」を選択します。

CODESYS再起動でユーザインタフェースの表示が「英語」に替わります。

CODESYSの画面構成



1	デバイス ツリー
2	POUツリー
3	エディタ ウィンドウ
4	メニュー バー
5	ツール バー
6	ステータス行
7	コメントのツールチップ表示
8	ツール ボックス
9	メッセージ ウィンドウ

Deviceビュー(Deviceツリー)

デバイス(コントローラ)で定義されている全てのリソースが表示されます。

代表的な表示内容:

「プロジェクト」: 例 MySample

「デバイス」: 例 Device

「リソース」: 例 PLC Logic

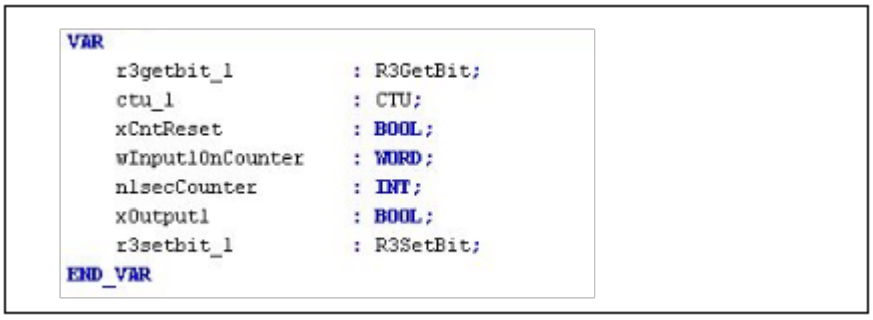
「アプリケーション」: 例 Application

- グローバル変数変数定義 例 GVL_PLC
- 使用ライブラリ定義
- POU(プログラム、ファンクションブロック、ファンクション) 例 PLC_PRG
- DUT(構造体、列挙など)
- タスク定義

プログラムウィンドウの構成(「変数宣言部」、「プログラム本体部」)

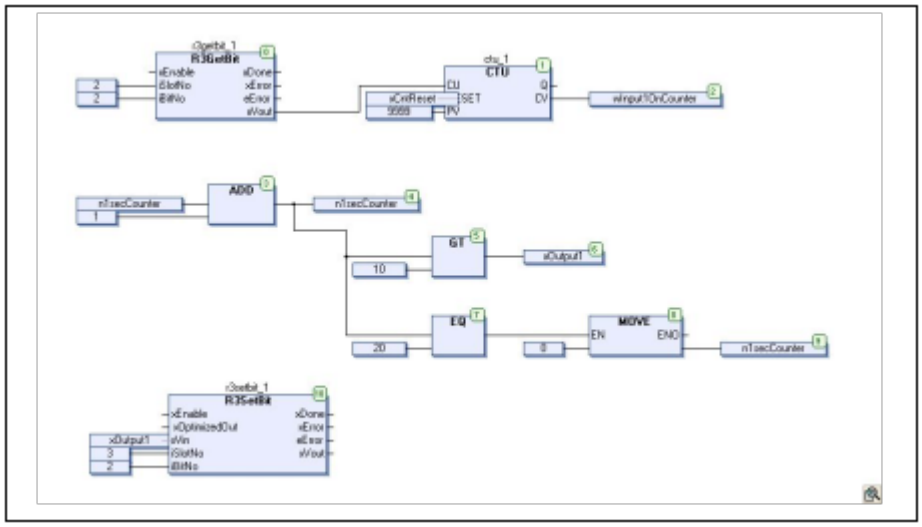
プログラムを作成する際に使用します。プログラムの作成では処理を記述する「プログラム本体部」とプログラム内で使用される変数を宣言する「変数宣言部」という2つのウィンドウを使用して行います。

「変数宣言部」の例



「プログラム本体部」の例

CFC言語



ST言語

```
(* Network 1 *)
r3getbit_1(iSlotNo:=2, iBitNo:=1);
ctu_1(CU:=r3getbit_1.xVout, RESET:=xCntReset, PV:=9999, CV=>wInput1OnCounter);
(* Network 2 *)
nlsecCounter := nlsecCounter + 1;
(* Network 3 *)
IF nlsecCounter > 10 THEN
    xOutput1 := TRUE;
ELSE
    xOutput1 := FALSE;
END_IF
(* Network 4 *)
IF nlsecCounter = 20 THEN
    nlsecCounter := 0;
END_IF
(* Network 5 *)
// r3setbit_1(iSlotNo:=3, iBitNo:=1, xVin:=xOutput1);
r3setbit_1.iSlotNo := 3;            // Slot:3, Ch:2
r3setbit_1.iBitNo := 1;
r3setbit_1.xVin:=xOutput1;
r3setbit_1();
```

オンライン モード情報

オンラインモードの情報は、画面の最下位に配置されていますステータスバーに表示されます。

表示される情報は:

[RUN] ：プログラム実行中

[STOP] ：プログラム停止中

[HALT ON BP] ：プログラムがブレイクポイントで停止中

Program loaded ：プログラムはデバイスにロード済み

Program unchanged	: デバイス内のプログラムはプログラムツールのものと一致しています
Program modified(Online Change)	: デバイス内のプログラムはプログラミングツールと異なるためオンライン変更が必要です
Program modified(Full download)	: デバイス内のプログラムはプログラミングツールと異なるため完全なダウンロードが必要です

(このページは空白です)

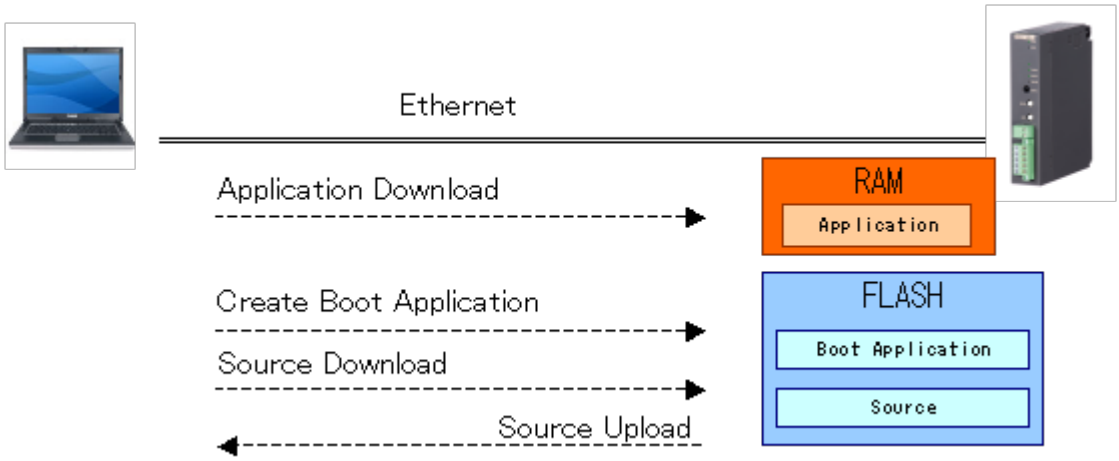
3.プログラミング

3.1.コントローラを使用する前に行っていただきたいこと

コントローラによりRTC (リアルタイムクロック)機能を搭載している機種があります。該当の機種を**「はじめて使用する」**場合や**「長期間未稼働のものを使用再開する」**場合など、日付時刻情報が正確ではない可能性がある場合には、使用開始前に再度日付時刻の設定を行うことをお勧めします。

3.2.プログラミングツールとの接続

プログラミングツールがインストールされたPCとは、本コントローラのEthernetポートで接続します。
プログラミングツールで接続する前に、pingコマンド等により通信が可能かどうか事前に確認されることをお勧めします。



3.3.アプリケーション

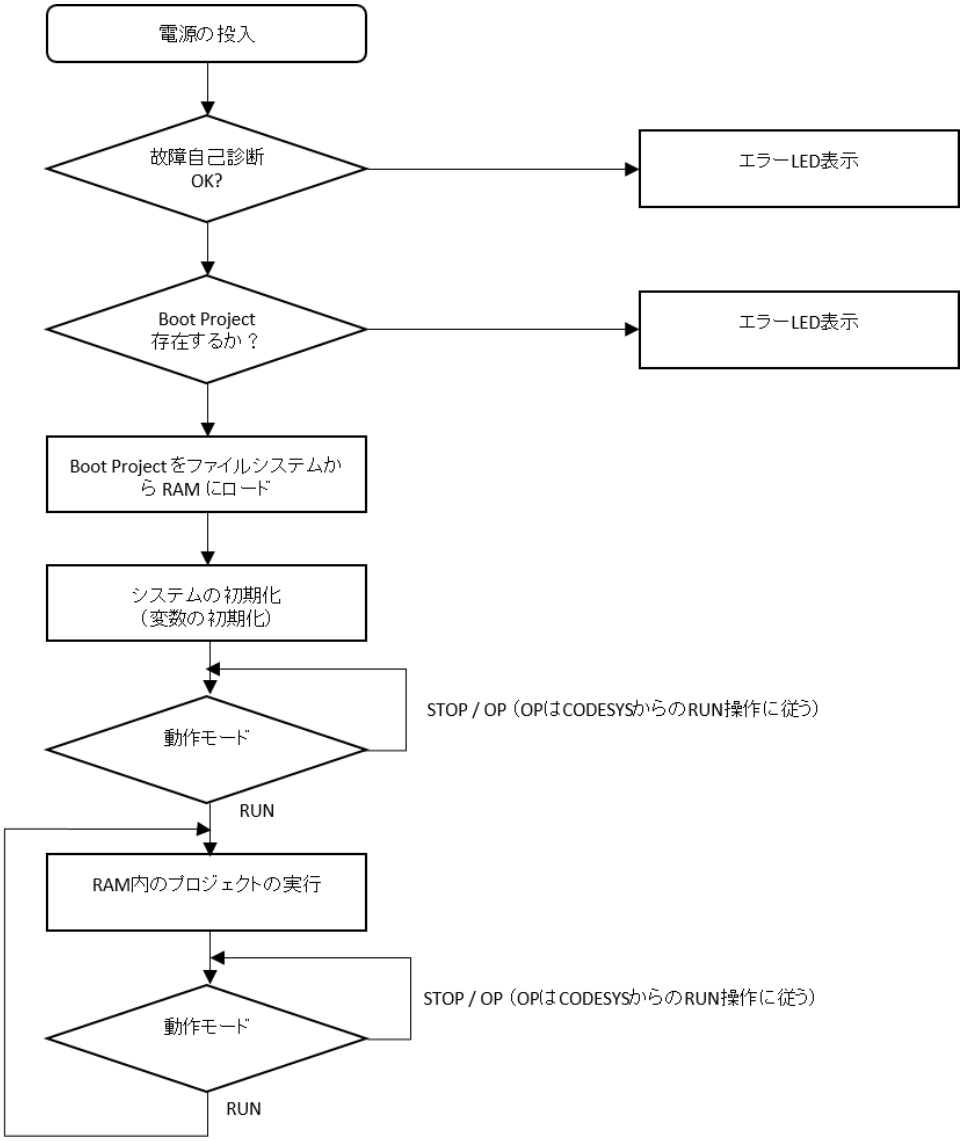
ここでは次の項目についての説明を記述します。

- アプリケーションの実行
- アプリケーションのサイクル
- ブートアプリケーション

アプリケーションの実行

電源供給またはハードウェアリセットを行うとコントローラが起動します。

起動は、最初に自己診断を行い異常が検知されなければファイルシステムに格納されているユーザプロジェクトがRAMに転送されます。その後ユーザプロジェクトは、RETAIN, PERSISTENT以外の変数が初期化され「RUN」モードに切り替わります。



アプリケーションのサイクル

サイクルタイムはアプリケーション(IECタスク) が開始されてから次の開始までの時間です。アプリケーションにループ処理がある場合や時間待ち合わせファンクションの呼び出しがある場合には、動作時間およびサイクルタイムがその分だけ長くなります。

アプリケーションの実行中は、入出力データやタイマ値(ファンクションブロックTONなど)の更新が行われません。これらの値の更新は、アプリケーションの最後で行われます。よってアプリケーションの処理中にイベント待ちやループ処理中のタイムアウト検知にタイマを使用することはできません。

ブートアプリケーションの起動

ブートアプリケーションとして登録されたアプリケーションは、電源供給またはハードウェアリセットを行うとコントローラが起動した後に自動的にファイルシステムからRAMに読み込み込まれ実行されます。

- ブートアプリケーションの登録：コントローラにログイン後のメニュー [Online] [Create boot application] で現在のアプリケーションをブートアプリケーションとして登録することができます。
- ブートアプリケーションの解除：コントローラにログイン後のメニュー [Online] [Reset origin] を実行すると解除されます。

3.4.オンラインコマンドと保持変数

オンラインコマンド実行後の各変数の状態を以下に示します。

×＝値が保持されます －＝値は初期化されます

オンラインコマンド	オンラインコマンド実行後の状態					
	VAR	VAR RETAIN	VAR PERSISTENT	Application	Boot Application	Downloaded Source
Download	－	－	×	Download	×	×
Online Change	×	×	×	Update	×	×
STOP	×	×	×	×	×	×
Reboot PAC	－	×	×	Loading boot application	×	×
Reset warm	－	×	×	×	×	×
Reset cold	－	－	×	×	×	×
Reset origin	－	－	－	－	－	×

(このページは空白です)

4.プログラミング要素

ここでは、プログラミングに必要となる要素について説明します。

4.1.POU (Program Organization Unit)

プログラム構成ユニットと訳されるプログラムの最小単位を指します。これには次の3種類があります。

名称	記号	キーワード
ファンクション	FUN (Function)	FUNCTION
ファンクションブロック	FB (Function Block)	FUNCTION_BLOCK
プログラム	PROG (Program)	PROGRAM

「**ファンクション**」と「**ファンクションブロック**」との主な違いは、「**ファンクション**」では全ての入力パラメータが等しければ返される結果は常に同じとなります(内部状態の記憶なし)。これに対して「**ファンクションブロック**」では内部状態を記憶(インスタンス化)するので呼び出し毎に返す結果を変化させることができます。また、「**プログラム**」はユーザプログラム(アプリケーション)の最上位に位置する唯一のインスタンスを持つファンクションブロックのようなもので、実行権(タスクからの呼び出し)を与えることができます。

POUの典型的な例(プログラミングツールでは個別のウィンドウで表示されます)

```
FUNCTION_BLOCK  MyFB
    VAR_INPUT
        bEN  :  BOOL;
        bReset  : BOOL;
        nCounterLimit  :  INT;
    END_VAR
    VAR_OUTPUT
```

```
        bERROR  :  BOOL;
        nCurrent  :  INT;
    END_VAR

    IF bEN=FALSE THEN
        RETURN;
    END_IF
    IF bReset = TRUE THEN
        nCurrent := 0;
    END_IF
    IF nCurrent <= nCounterLimit THEN
        nCurrent := nCurrent + 1;
    END_IF
END_FUNCTION_BLOCK
```

4.2.FUN (Function)

全ての入力パラメータが等しければ、返される結果は常に同じとなります(内部状態の記憶なし)。

種類	可否
入力パラメータ	Yes
出力パラメータ	No
入出力パラメータ	No *1
ファンクションの戻り値	Yes
ローカル変数と出力変数の保持	No

*1): 拡張機能(CODESYS では使用可能)

4.3.FB (Function Block)

内部状態を記憶(インスタンス化)するので呼び出し毎に返す結果を変化させることができます。

種類	可否
入力パラメータ	Yes
出力パラメータ	Yes
入出力パラメータ	Yes
ファンクションの戻り値	No
ローカル変数と出力変数の保持	Yes

4.4.DUT (Data Unit Type)

データユニットタイプは、ユーザ定義の新しいデータ型を定義できます。基本型や既に定義されたユーザ型から新しいユーザ型を定義することができます。

種類	キーワード
配列	<Array_Name> : ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>
構造体	TYPE <structurename>: STRUCT <declaration of variables 1> ... <declaration of variables n> END_STRUCT END_TYPE
共用体	TYPE <structurename>: UNION <declaration of variables 1> ... <declaration of variables n> END_UNION END_TYPE
列挙	TYPE <identifier> : (<enum_0> ,<enum_1>, ...,<enum_n>) |<base data type>; END_TYPE
参照	<identifier> : REFERENCE TO <data type>
ポインタ	<identifier>: POINTER TO <object>
範囲型	TYPE <name> : <Int type> (<ug>..<og>) END_TYPE

4.5.VAR (Variable)

変数の種類と属性を以下に示します。

種類	キーワード
入力パラメータ	VAR_INPUT
出力パラメータ	VAR_OUTPUT
入出力パラメータ	VAR_IN_OUT
ローカル	VAR
グローバル	VAR_GLOBAL
CONSTANT	VAR CONSTANT / VAR_GLOBAL CONSTANT
RETAIN(不揮発指定)	VAR RETAIN
PERSISTENT(永続指定)	VAR_GLOBAL PERSISTENT RETAIN
静的変数	VAR_STAT
一時変数	VAR_TEMP

4.6.Direct I/O (Variable)

入出力カードやメモリに直接(絶対番地)変数を割り当てる方法があります。

種類	記号	宣言例
入力	%I	xInput1 : BOOL AT %IX0.0; bInput2 : BYTE AT %IB1; wInput3 : WORD AT %IW2;
出力	%Q	xOutput1 : BOOL AT %QX0.0; bOutput2 : BYTE AT %QB1; wOutput3 : WORD AT %QW2;
メモリ	%M	xFlag1 : BOOL AT %MX0.0; bFlag2 : BYTE AT %MB1; wFlag3 : WORD AT %MW2;

補 足

- **弊社コントローラでは特別な場合を除いてこの割り当て方法(Direct I/O)を使用する必要はありません。**
- 弊社コントローラでは、入出力カードやメモリにアクセスする専用命令が用意されています。それら専用命令を使用すると複数バイトのデータアクセスをより安全に行うことができます。

4.7.リテラル

数値などの常数を直接表記する場合はリテラルを用います。リテラルは、数字、文字列、時刻の表記に必要です。

また、デバックなどで値を入力するときにも、リテラルと同様の表記を使用する必要があります。

リテラルは、次のように表記します。

[データ型 #][データ]

- 数字リテラル
- 文字列リテラル
- 持続リテラル
- 日付時刻リテラル

数字リテラル

使用できる数字リテラルを次の表に示します:

型	例
整数リテラル	-12 0 123_456 +986
実数リテラル	-12.0 0.0 0.4560 3.14159_26
指数付実数リテラル	-1.34E-12 -1.34e-12 1.0E+6
2 進リテラル	INT#2#1111_1111
8 進 リテラル	INT#8#377
16 進リテラル	INT#16#FF SINT#16#ff
ブール FALSE と TRUE	FALSE TRUE
ブール 0 と 1	0

型	例
	1

INT やBOOL だけでなく、変数ワークシートで使用するリテラルも、キーワード(データ型)なしで使用できます。

例：

INT#16#FF なら 16#ff を使えます。

BOOL#FALSE なら FALSE を使えます。

文字列リテラル

文字列リテラルは、2つの単引用符で囲まれたゼロまたはそれ以上の連続する文字です。WSTRING (UNICODE) 文字列リテラルでは、2つの2重引用符で囲みます。

型	例
空文字列	''
空白付文字列	' '
空でない文字列	'Hello'
空でないWSTRING文字列	"こんにちは"

持続リテラル

持続時間データは、時間、分、秒、ミリ秒とその組合せで表せます。

型	例
短い接頭語	T#14ms t#14ms t#12m18s3.5ms T#25h_15m t#25h_15m
長い接頭語	TIME#14ms time#14ms TIME#25h_15m time#25h_15m

日付時刻リテラル

型	例
日付	DATE#1996-01-24 date#1996-01-24 D#1996-01-24 d#1996-01-24
時刻	TIME_OF_DAY#15:36:55.36 time_of_day#15:36:55.36 TOD#15:36:55.36 tod#15:36:55.36
日付と時刻	DATE_AND_TIME#1996-01-24-15:36:55.36 date_and_time#1996-01-24-15:36:55.36 DT#1996-01-24-15:36:55.36 dt#1996-01-24-15:36:55.36

4.8.データ型

ここでは、次の項目について説明しています。

基本データ型

型	説明	データ範囲
BOOL	ブール	TRUE, FALSE
SINT	単精度（8ビット）整数	-128 ～ +127
USINT	符号なし単精度（8ビット）整数	0 ～ +255
INT	（16ビット）整数	-32,768 ～ +32,767
UINT	符号なし（16ビット）整数	0 ～ +65,535
DINT	倍精度（32ビット）整数	-2,147,483,648 ～ +2,147,483,647
UDINT	符号なし倍精度（32ビット）整数	0 ～ +4294967295
LINT	64ビット整数	-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807
ULINT	符号なし64ビット整数	0 ～ 18,446,744,073,709,551,615
TIME	持続時間	4,294,976,295ms ～ 4,294,976,295s
BYTE	長さ8のビット列（バイト）	16#00 ～ 16#FF
WORD	長さ16のビット列（ワード）	16#0000 ～ 16#FFFF
DWORD	長さ32のビット列（ダブルワード）	16#00000000 ～ 16#FFFFFFFF
LWORD	長さ64のビット列（クワッドワード）	16#0000000000000000 ～ 16#FFFFFFFFFFFFFFFF
REAL	単精度実数（32ビット浮動小数） （IEEE 754）	±0.0000001 ～ ±9,999,999 有効桁数7桁 （*1）

型	説明	データ範囲
LREAL	倍精度実数（64ビット浮動小数） （IEEE 754）	±0.000000000000001 ～ ±999,999,999,999,999 有効桁数15桁 （*1）
STRING	文字列	最大255文字 （初期80文字）
WSTRING	UNICODE文字列	最大255文字 （初期80文字）
Arrays	配列	
Structures	構造体	
Union	共用体	

*1: 演算結果は誤差(桁落ち、小数点位置の違う2値の演算による有効桁数による情報落ち)が発生することに留意する必要があります。
例えば、REAL型の次の演算では 1.1 + 2.2 = 3.3 を期待するが結果は 3.3000002 となります。

Arrays：配列

配列は、要素となるデータ型の1～3次元までの配列を提供します。この配列はPOUやグローバル変数リストの中の宣言部で定義できます。

構文：

<配列名> : ARRAY [<ll1>..<ul1>, <ll2>..<ul2>, <ll3>..<ul3>] OF <要素データ型>
ll1, ll2, ll3 は各次元の下限値で、ul1, ul2, ul3 は各次元の上限値を整数で指定します。

例：

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

配列を初期化する例：

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];
arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)];   (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];   (* short
for 0,0,4,4,4,4,2,3 *)
```

構造体の配列を初期化する例：

```
Structure definition:
```

```
        TYPE STRUCT1
        STRUCT
              p1:int;
              p2:int;
              p3:dword;
        END_STRUCT
        END_TYPE
    Array initialization:
        ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),
        (p1:=2,p2:=0,p3:=299),(p1:=14,p2:=5,p3:=112)];
```

配列の部分的な初期化の例：

```
    arr1 : ARRAY [1..10] OF INT := [1,2];
```

Structures：構造体

構造体は、プロジェクト内にDUT オブジェクトとして定義できます。

構文：

TYPE <structurename>:
STRUCT

<変数1の宣言>
...
<変数nの宣言>

END_STRUCT
END_TYPE

構造体名 Polygonline の初期化の例：

```
    TYPE Polygonline:
    STRUCT
    Start:ARRAY [1..2] OF INT;
        Point1:ARRAY [1..2] OF INT;
```

```
        Point2:ARRAY [1..2] OF INT;
        Point3:ARRAY [1..2] OF INT;
        Point4:ARRAY [1..2] OF INT;
        End:ARRAY [1..2] OF INT;
    END_STRUCT
    END_TYPE
```

構造体の初期化の例：

```
    Poly_1:polygonline := ( Start:=[3,3], Point1:=[5,2], Point2:=[7,3],
    Point3:=[8,5], Point4:=[5,7], End:= [3,5]);
```

構造体メンバーのアクセス:

次の構文を使用して構造体のメンバーにアクセスできます:

<構造体名>.<メンバー名>

次は構造体`polygonline` のメンバー `start` をアクセスする例です。

```
    Poly_1.Start
```

構造体内でのBit アクセス

データ型 BIT は、構造体内の宣言にだけ使用できます。これは構造体における1ビットのメモリ空間を消費する名前の付いた単一ビットを宣言できます。

TYPE <構造体名>:
STRUCT

<BIT名 bit1> : BIT;
<BIT名 bit2> : BIT;
<BIT名 bit3> : BIT;
...
<BIT名 bitn> : BIT;

END_STRUCT
END_TYPE

次の構文を使用することで構造体のBITメンバーにアクセスできます:

<構造体名>.<BIT名>

補 足

BIT変数のポインタ指定と参照指定は使用できません。また、BIT変数の配列は許されていません。

5.プログラミング言語

IEC61131-3では、5つのプログラミング言語の定義、表記と要素を規定しています。

ここでは、次のプログラミング言語について説明します。

- コンティニアス ファンクション チャート「グラフィック」(言語としてはFBD)

  CFC (Continuous Function Chart)

- ファンクション ブロック ダイアグラム「グラフィック」

  FBD (Function Block Diagram)

- 命令リスト「テキスト」

  IL (Instruction List)

- ラダー ダイアグラム「グラフィック」

  LD (Ladder Logic Diagram)

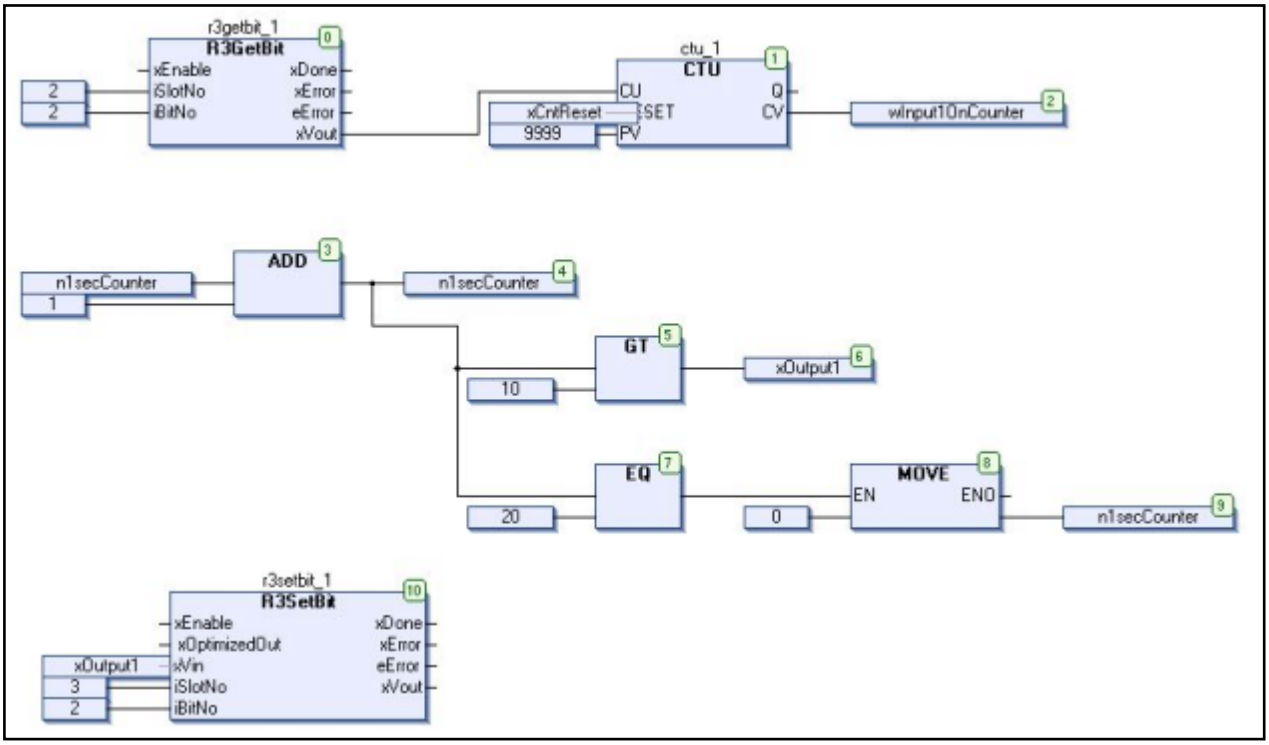
- シーケンシャル ファンクション チャート「グラフィック」

  SFC (Sequential Function Chart)

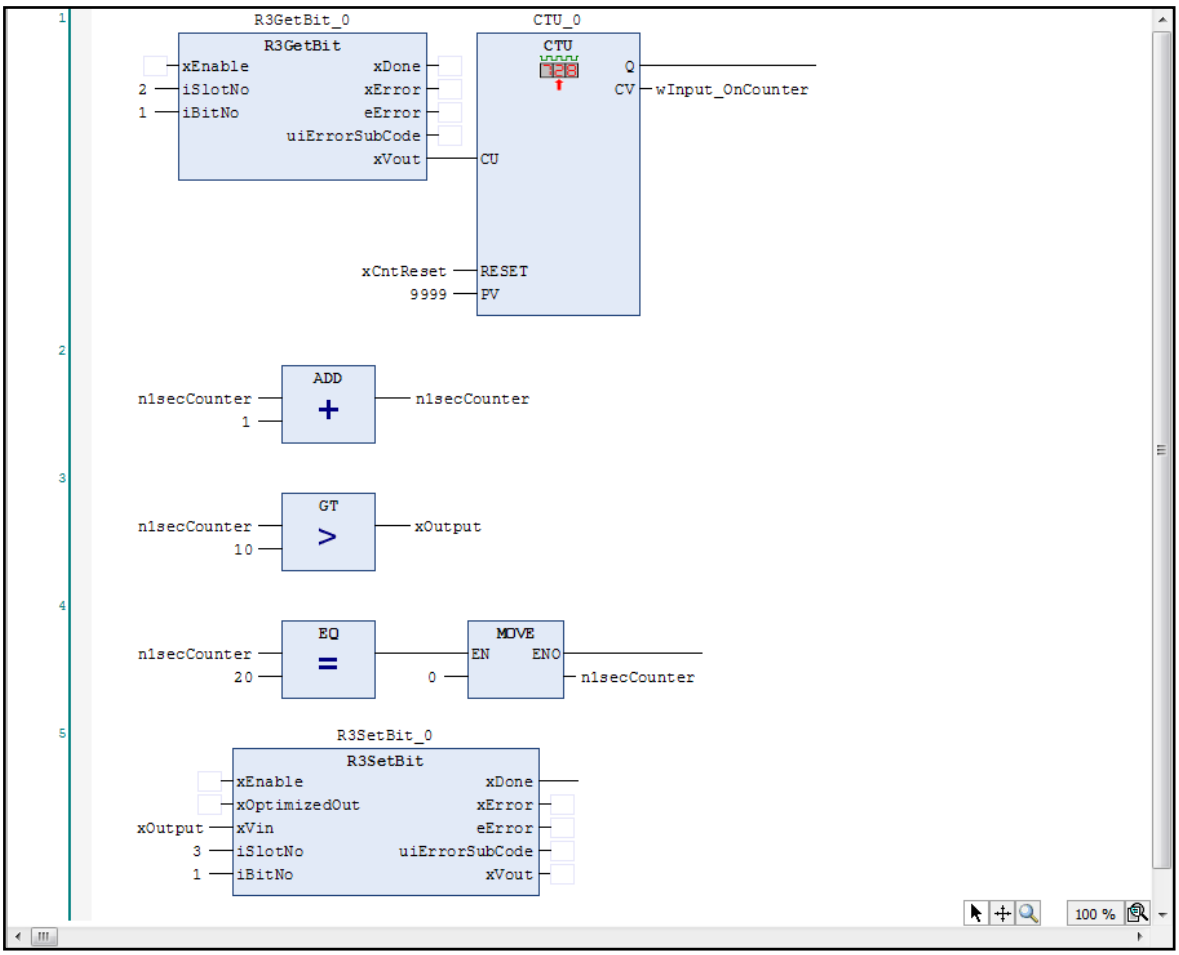
- 構造化テキスト「テキスト」

  ST (Structured Text)

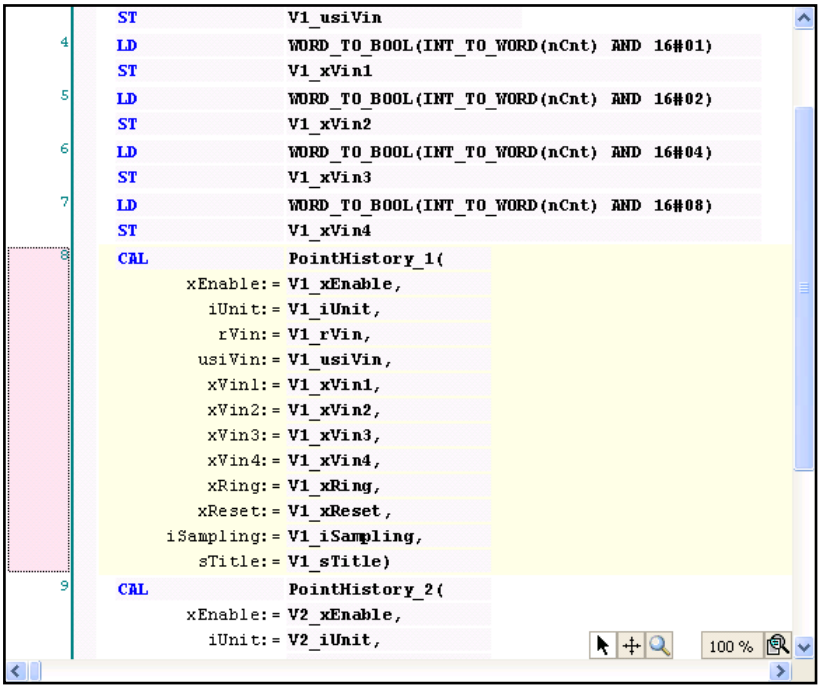
5.1.CFC (Continuous Function Chart)



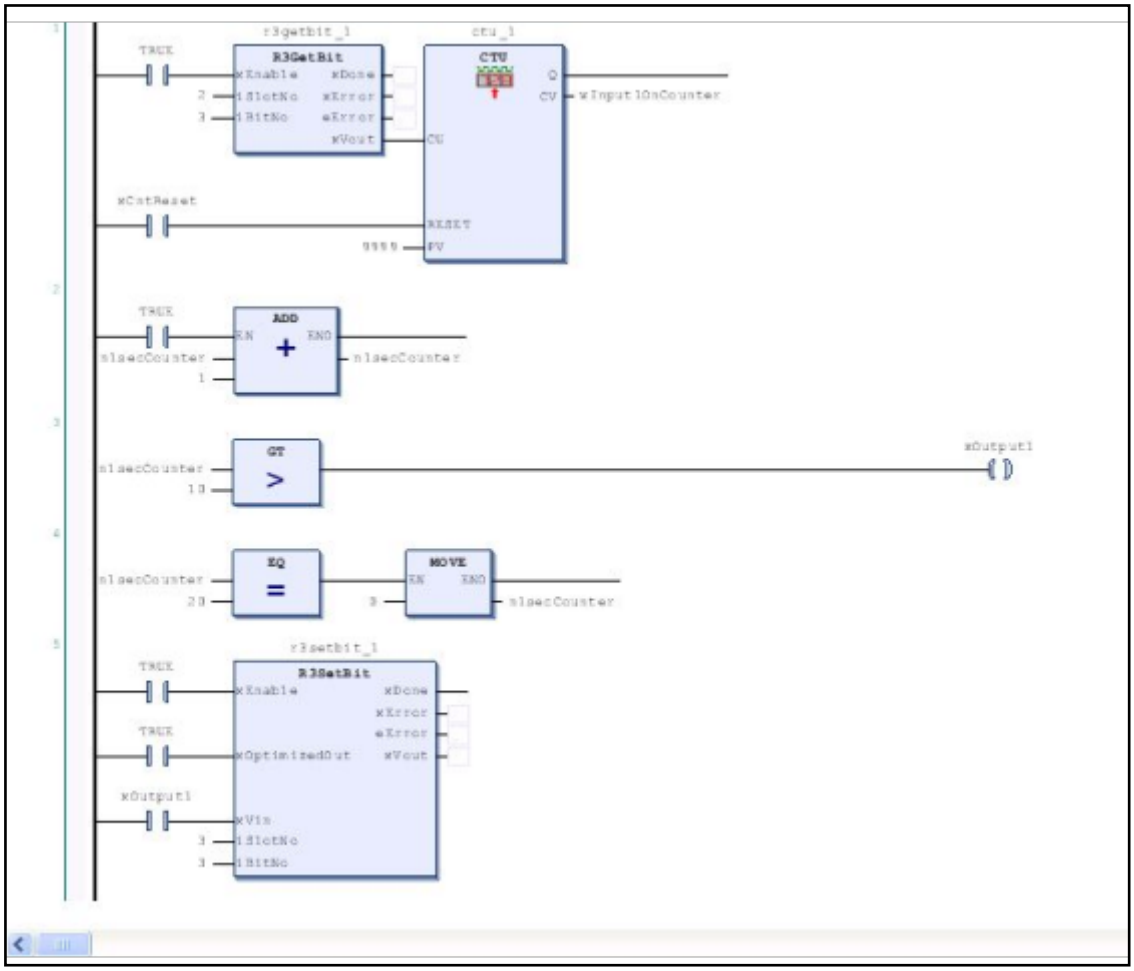
5.2.FBD (Function Block Diagram)



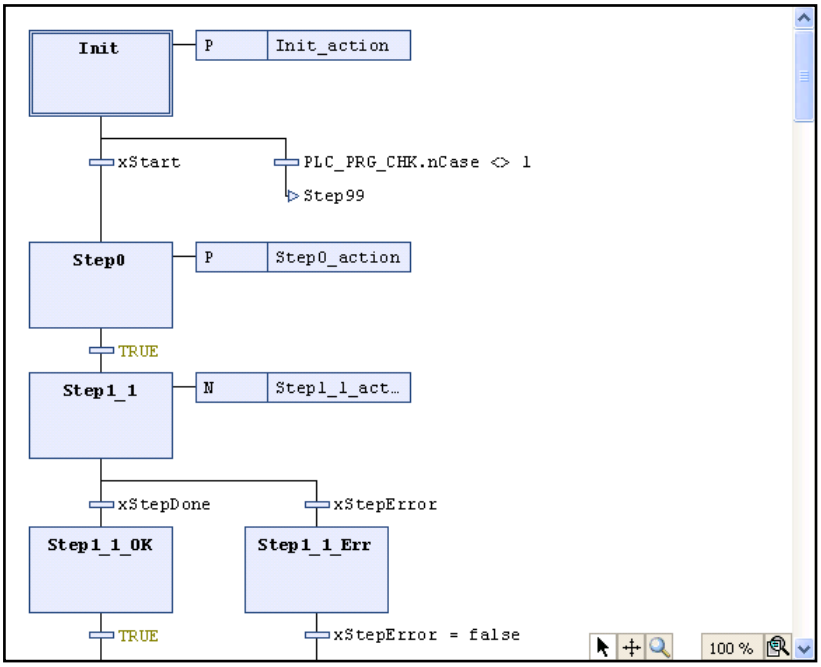
5.3.IL (Instruction List)



5.4.LD (Ladder Logic Diagram)



5.5.SFC (Sequential Function Chart)



5.6.ST (Structured Text)

```
(* Network 1 *)
r3getbit_1(iSlotNo:=2, iBitNo:=1);
ctu_1(CU:=r3getbit_1.xVout, RESET:=xCntReset, PV:=9999, CV=>wInput1OnCounter);
(* Network 2 *)
n1secCounter := n1secCounter + 1;
(* Network 3 *)
IF n1secCounter > 10 THEN
    x0Output1 := TRUE;
ELSE
    x0Output1 := FALSE;
END_IF
(* Network 4 *)
IF n1secCounter = 20 THEN
    n1secCounter := 0;
END_IF
(* Network 5 *)
// r3setbit_1(iSlotNo:=3, iBitNo:=1, xVin:=x0Output1);
r3setbit_1.iSlotNo := 3;            // Slot:3, Ch:2
r3setbit_1.iBitNo := 1;
r3setbit_1.xVin:=x0Output1;
r3setbit_1();
```

(このページは空白です)

6.プロジェクトの作成と実行

本章では、サンプルプロジェクトの作成を通してプログラミングからデバッグの手順を説明しています。

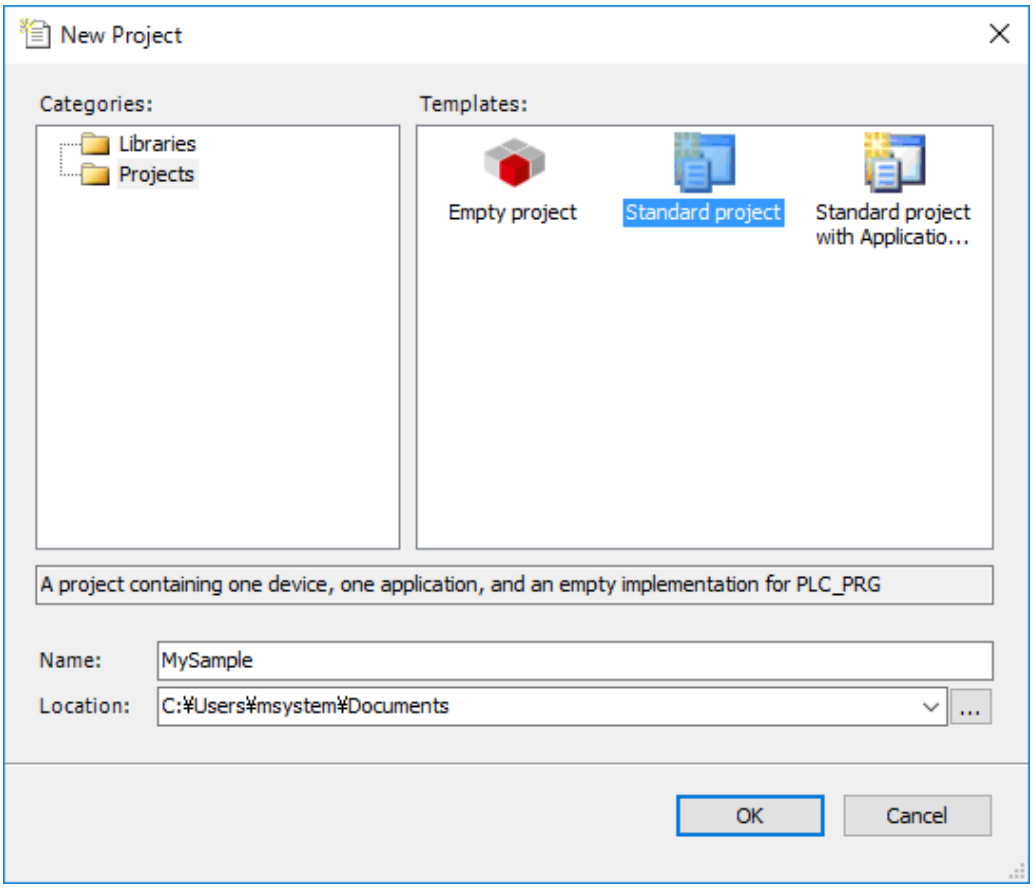
手順

- プロジェクトの作成
- プログラミング
- ターゲットへの接続とアプリケーションのダウンロード
- デバッグ
- ターゲットとの接続を終了
- HMIの作成

6.1.プロジェクトの作成

新規プロジェクトの作成

操作：[File] → [New Project...]



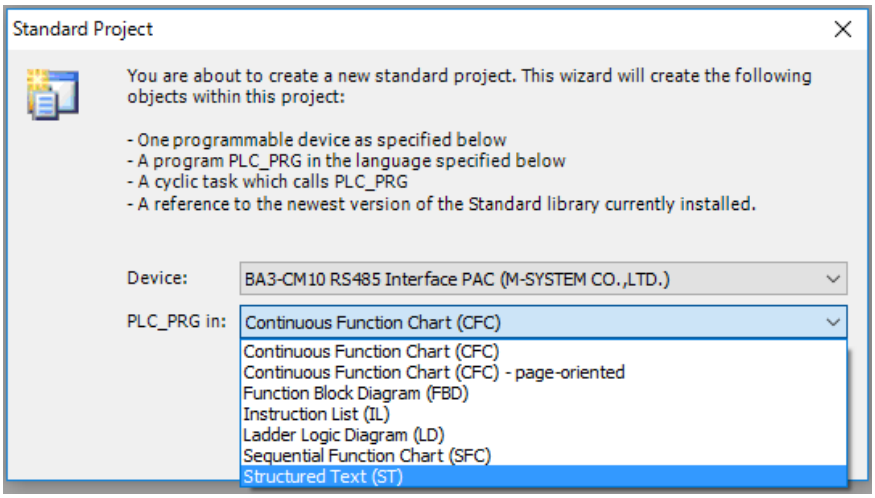
Templates: 作成するプロジェクトに合うテンプレートを選択します。

例 : [Standard project] を選択します。

Name: プロジェクトの名称を入力します。

例 : MySample

Location: 作成されるプロジェクトファイルを格納するためのフォルダを指定します。



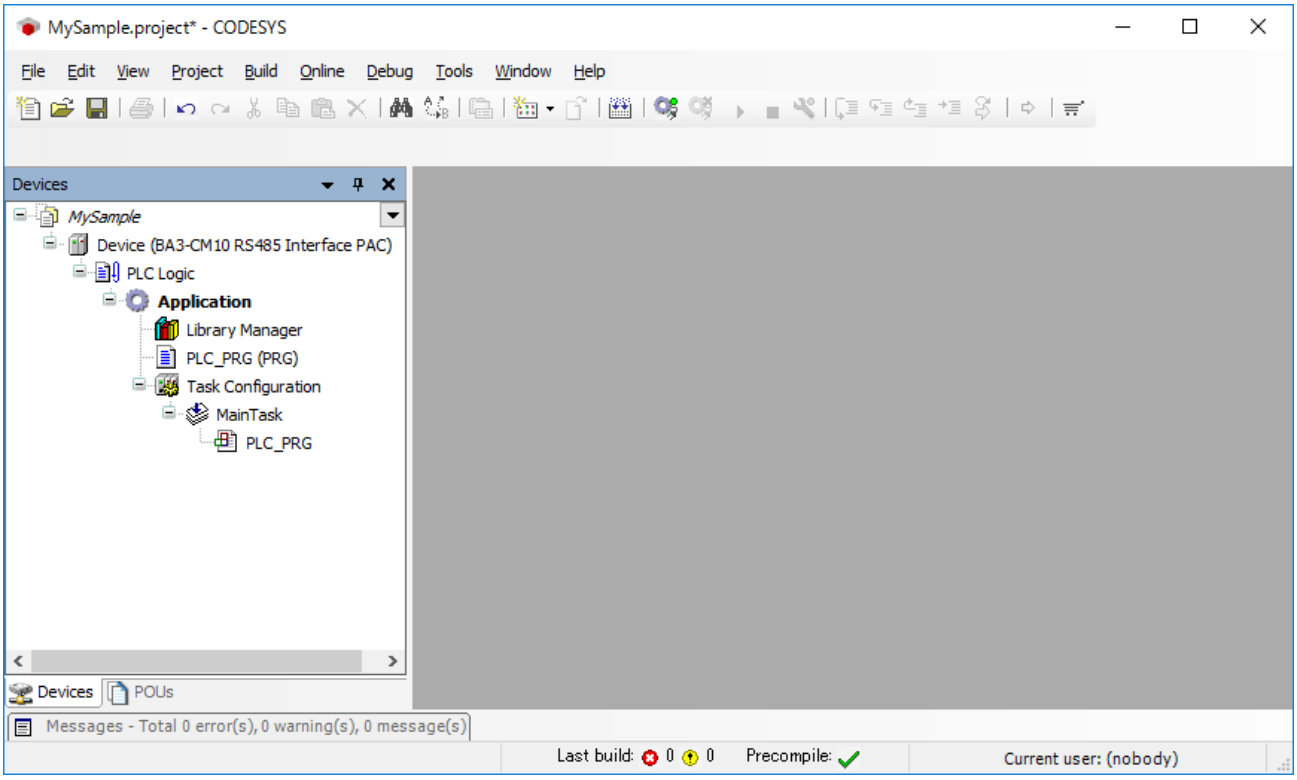
Device: 作成するプロジェクトを実行するターゲットを選択します。

例 : [BA3-CM10 RS485 Interface PAC (M-SYSTEM CO.,LTD.)] を選択します。

PLC_PRG in: 作成するプロジェクトに作成される最初のPOUの記述に使用するIECプログラミング言語を選択します。

例 : [Structured Text (ST)] を選択します。

プロジェクトが作成され各ウィンドウが表示されます。「Deviceツリー」では、次のような表示が確認できます。プロジェクト名は「Device」,「POUs」ウィンドウにおいてツリーのルート名 (例 : MySample) として表示されます。

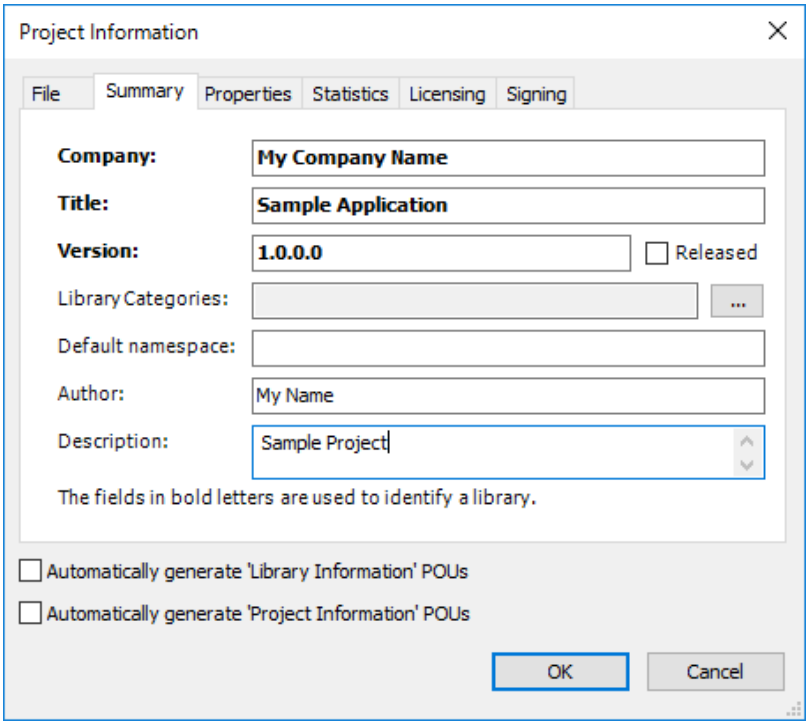


「POUs」タブは、POU管理やプロジェクト設定に使用します。

「Device」タブは、ターゲット(例 : "Device (BA3-CM10 RS485 Interface PAC)") に属するコンポーネントをツリー構造で管理します。このツリーでは、プログラム「PLC_PRG」とそのタスク設定の「Main Task」が含まれています。そして、このプロジェクトで使用するライブラリを管理する「Library Manager」もツリーで管理します。初期状態はI/O設定に必要なライブラリ「IoStandard」とIEC61131-3に従った全てのファンクションとファンクションブロックを提供するライブラリ「Standard」が読み込まれています。

プロジェクト情報の設定

操作 : [Project] → [Project Information...]



ここではプロジェクトの作成者やタイトルなどの情報を記録します。

Company：作成者の会社名など 例：My Company Name

Title：タイトル 例：Sample Application

Version：バージョン 例：1.0.0.0

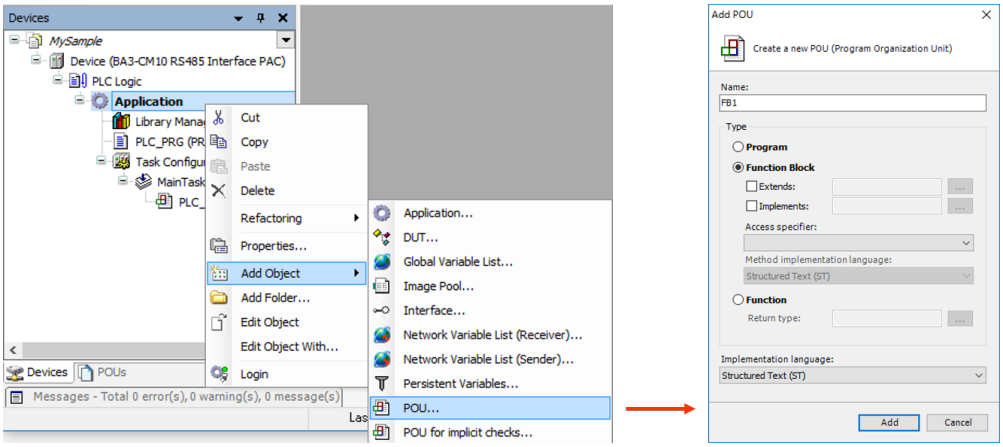
Author：作成者 例：My Name

Description：説明 例：Sample Project

6.2.プログラミング

POUの作成

操作：デバイスツリー内の Application を選択した状態で [Project] → [Add Object] → [POU...]



ここでは

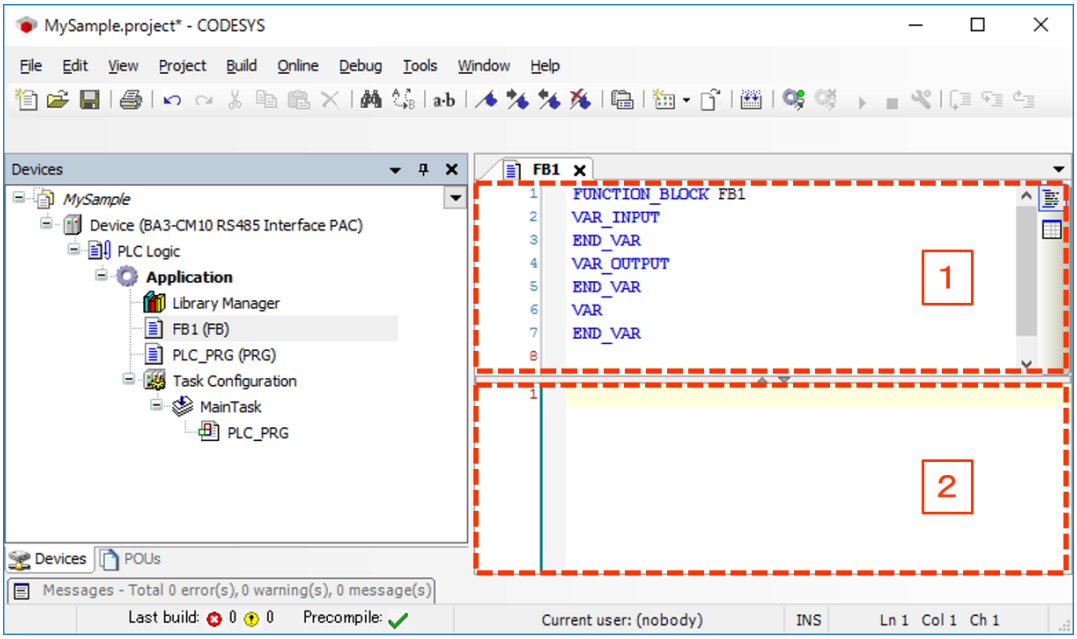
Name: “FB1”

Type: [Function Block] を選択

Implementation Language: [Structured Text (ST)] を選択

[Add] ボタンを押します。

ファンクションブロック"FB1"のプログラミング用エディタが開きます。このエディタは、変数宣言部とプログラム本体部とに上下画面分割されています。



1	変数宣言部	各種変数の定義や起動時に設定する初期値の定義を記述する場所です。 POUタイプと名前("FUNCTION_BLOCK FB1")、変数宣言のためのキーワード"VAR_INPUT"や"END_VAR"を表示しています。

2	プログラム 本体部	IECプログラミング言語によるプログラム コードを記述する場所です。 初期状態では空で、行番号として「1」という数字のみが表示されています。

エディタの変数宣言部でVAR_INPUTの後ろにカーソルを置き、エンターキーを押します。新しい行が挿入されますので、整数型(INT型)で「in」を入力します。同じ操作でVAR_OUTPUTには整数型(INT型)で「out」、VARには、初期値を2とする整数型(INT型)で「ivar」を宣言します。

```
FUNCTION_BLOCK FB1
VAR_INPUT
    in: INT;
END_VAR
VAR_OUTPUT
    out: INT;
END_VAR
VAR
    ivar: INT:=2;
END_VAR
```

プログラム本体部は、下記のステートメントを入力します。

```
out := in + ivar;
```

これで、入力変数"in"に定数ivar=2を加えた結果を出力変数"out"に出力するファンクションブロックの"FB1"が定義されました。

PLC_PRGの変数宣言

操作：「Device」ペイン内のツリーから「PLC_PRG」をダブルクリックする

エディタの変数宣言部に整数型(INT型)で「ivar」と先で定義したファンクションブロック(FB1)のインスタンスを「fbinst」として宣言します。

```
PROGRAM PLC_PRG
VAR
    ivar: INT;
    fbinst: FB1;
END_VAR
```

なお、このように変数宣言部で1つ1つの変数を宣言する代わりに、次に示す「自動宣言」機能を使用することができます。

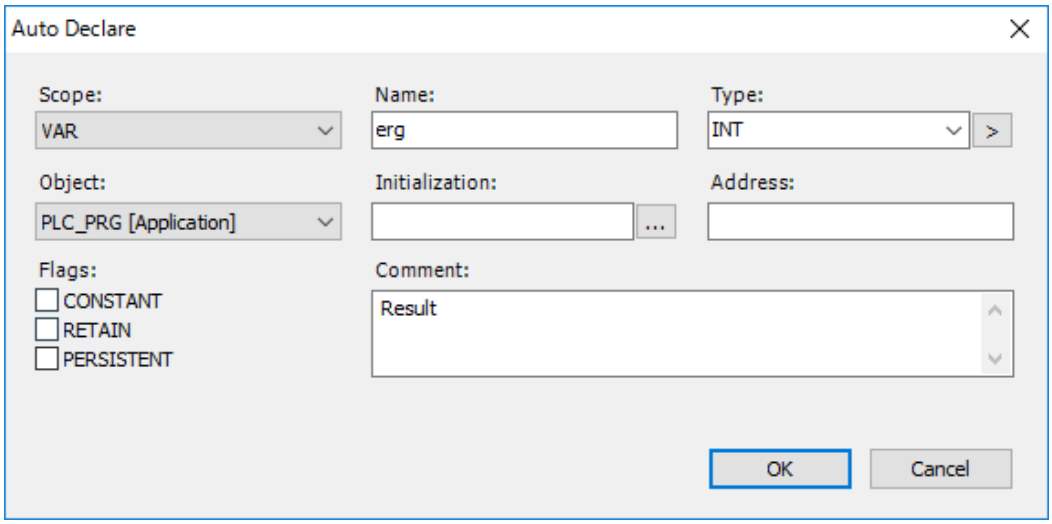
変数の「自動宣言」機能

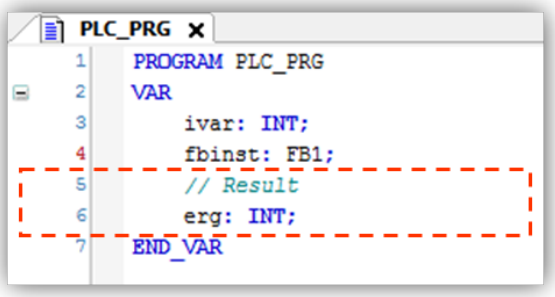
エディタのプログラム本体部は下記のステートメントを入力します。

```
ivar := ivar + 1;
fbinst(in:=11, out=>erg);
```

ここで、まだ宣言されていない変数がプログラムで使用されると、変数宣言「入力アシスタント」ダイアログが自動的に開かれます。内容を確認して[OK]を押すと、未定義であった変数(この例では「erg」)の宣言が、変数宣言部に追加されます。

(この未宣言変数の「入力アシスタント」自動表示は、オプションの指定により抑止できます)

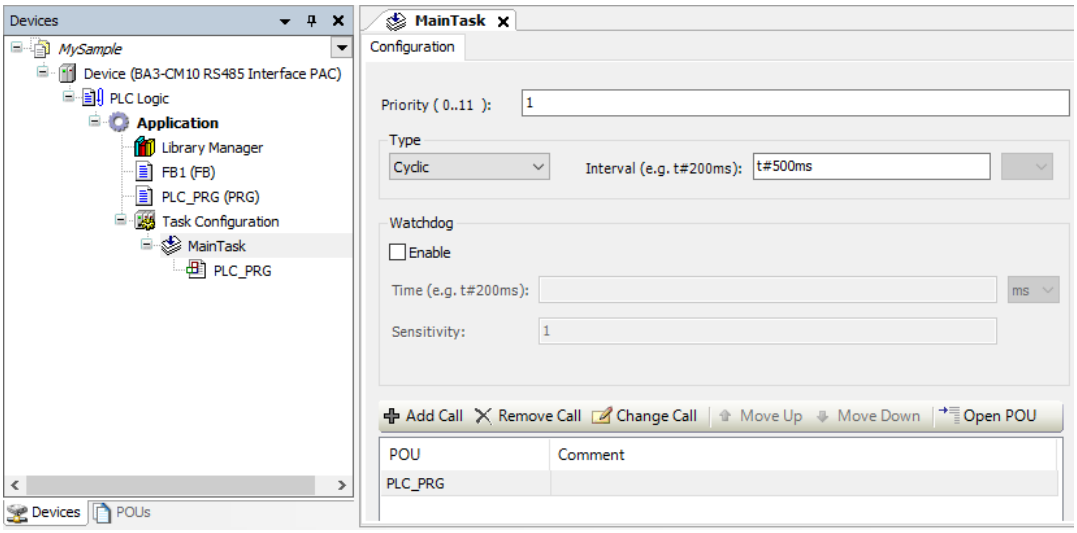




タスク設定

操作：「Device」ペイン内のツリーから「MainTask」をダブルクリックする

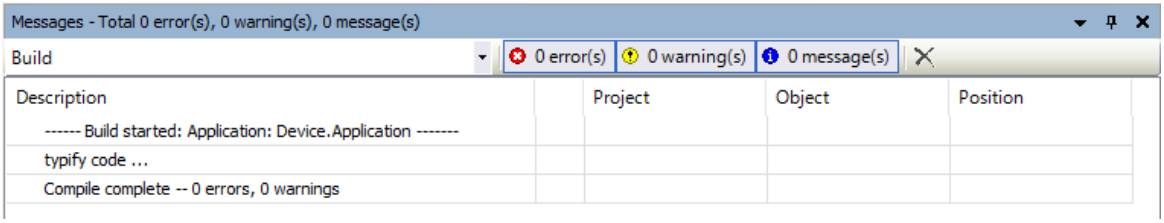
このタスク設定画面では、既にCALL一覧に登録されているPOU(この例ではPLC_PRG)の実行周期を指定します。初期設定では、500ms周期で呼び出されます。



これでプログラム作成が終わりました。次は、プログラムのエラーチェックとコントローラに転送するイメージを作成するコンパイルを行います。

ビルド(コンパイル)

操作：「Build」から「Build」を選択



正常にビルドが終わると上記のように0 errors, 0 warningsと表示されます。

もし、エラーや警告が検出された場合は、出現場所がこのメッセージウィンドウに表示されます。エラーと警告が無くなるまで、問題個所の修正とビルド作業を繰り返す必要があります。

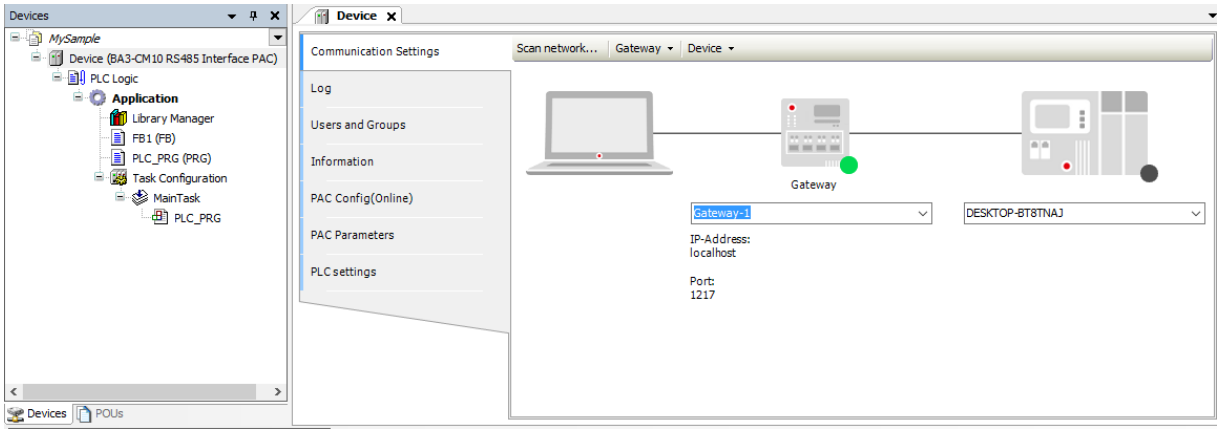
補 足

インスタンスの追加または削除を含む変更をOnline Change(実行中に変更後継続実行)で行うと問題が発生する場合があります。インスタンスの追加または削除を行った後のBuild(明示、暗黙的に関わらず)の前にはCleanを行うようにしてください。

6.3.ターゲットへの接続とアプリケーションのダウンロード

ターゲットとの接続

操作：Deviceツリーから「Device」を選択



この通信設定画面で接続に使用するゲートウェイと接続先のターゲット(コントローラ)を設定します。ここでは、現在接続できる(PCが接続されているネットワークに接続されているターゲット)ターゲットを検出します。

手順：

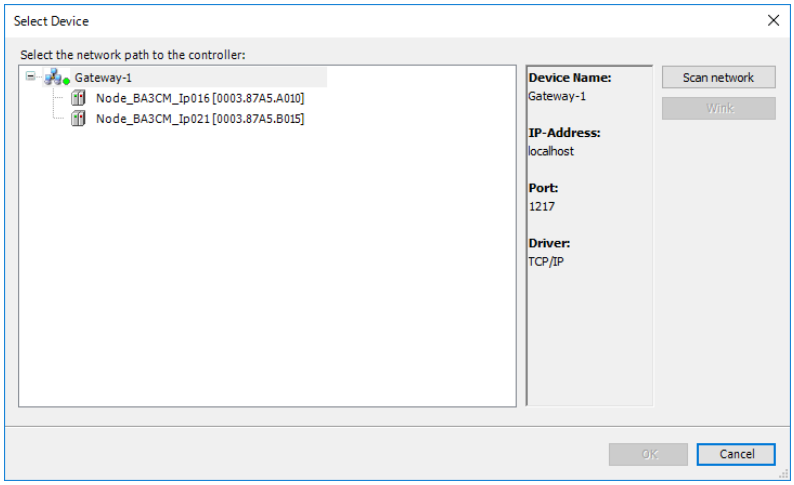
1. 画面左上に表示されている「Scan network」をクリックします。

   ネットワーク上に存在する接続可能なコントローラが検出されリストアップされます。

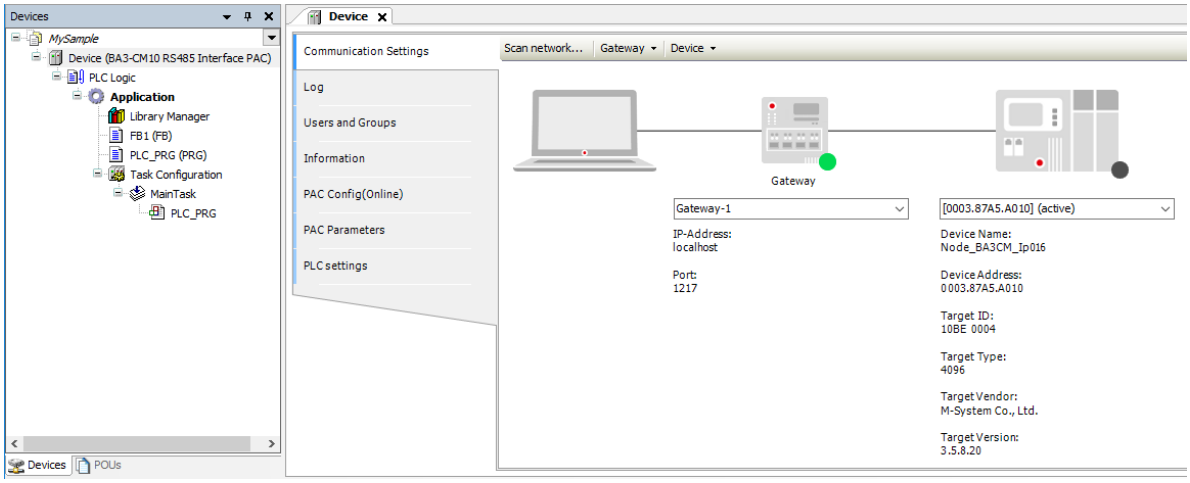
2. 接続するターゲットを選択します。

リストアップされたコントローラからコントローラを1台選び「OK」ボタンを押します。

下記の例では2台のターゲットが検出されています。その中から1つを選択して「OK」を押します。



もし、「Scan network」で1台もコントローラが検出されない場合は、ネットワークの設定を確認する必要があります。接続対象のコントローラとPCとが同一のネットワークに存在するようにネットワークパラメータ(IPアドレスなど)が適切かどうか確認してください。



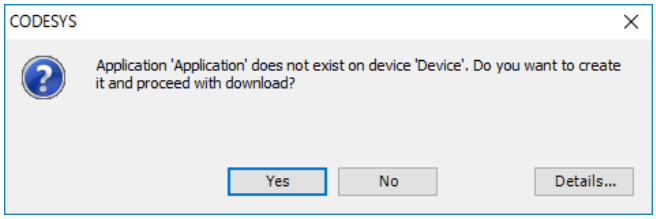
検出されたコントローラは次の書式で表示されます。

機種	表示形式1
BA3-CB10	Node_BA3CB_IpNNN NNN: IP Address (Rotary SW)
BA3-CM10	Node_BA3CM_IpNNN NNN: IP Address (Rotary SW)

機種	表示形式1
BA3-CM20	Node_BA3CM2_IpNNN NNN: IP Address (Rotary SW)

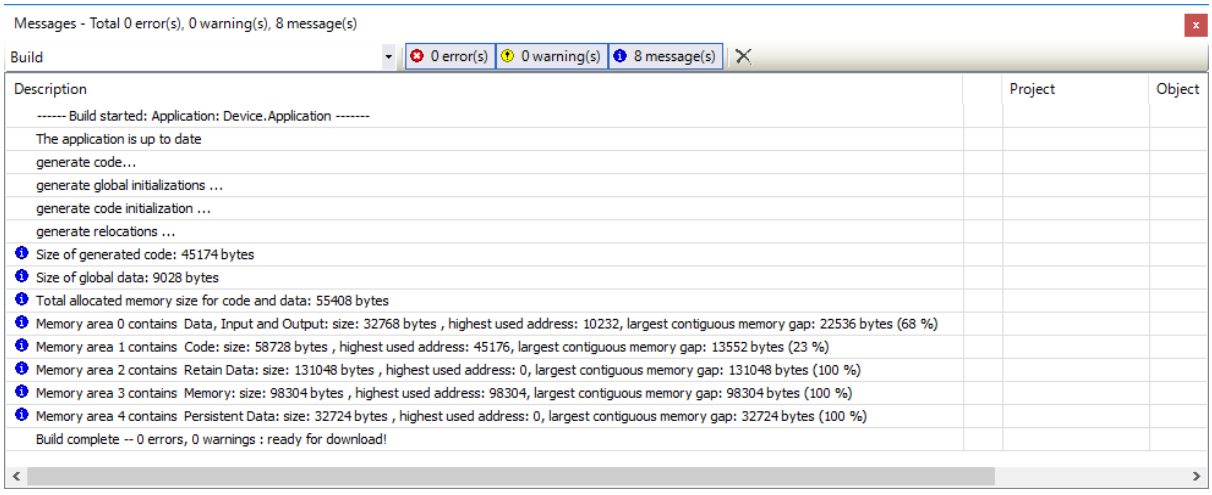
ターゲットへの接続(ログイン)とアプリケーションの転送

操作 : [Online] → [Login], またはツールバー : 



このダイアログは、接続先のターゲットにアプリケーションがない場合に表示されます。「Yes」ボタンを押してプログラムのダウンロード(PCからターゲットへプログラム転送)を行います。

プログラミングツールは、ターゲットにログインすると現在状態、現在値をリアルタイムに表示するオンラインモードとなります。



ダウンロードされたアプリケーションが必要としているメモリ容量や空きメモリ状況が表示されます。

ここで表示される gap は連続したメモリの空き容量です。

アプリケーションのダウンロードに必要な空きメモリは、分断されたメモリの合計ではなく、この連続したメモリの空き容量です。何度もダウンロードを行うと次のダウンロードに必要な連続した空きメモリ容量が確保できなくなる場合があります。

そのような場合のダウンロードは失敗します。連続した空きメモリを確保するためにはアプリケーションの再ビルドを行い完全ダウンロードを実施する必要があります。

手順は

1. ログイン状態ならばログアウトします。
2. メニュー「Build」から「Clean all」を選択します。
3. 同じメニューから「Build」を選択します。
4. 再度ターゲットにログインします(アプリケーションのダウンロードを問うダイアログが表示されるので「Yes」とします)。

補 足

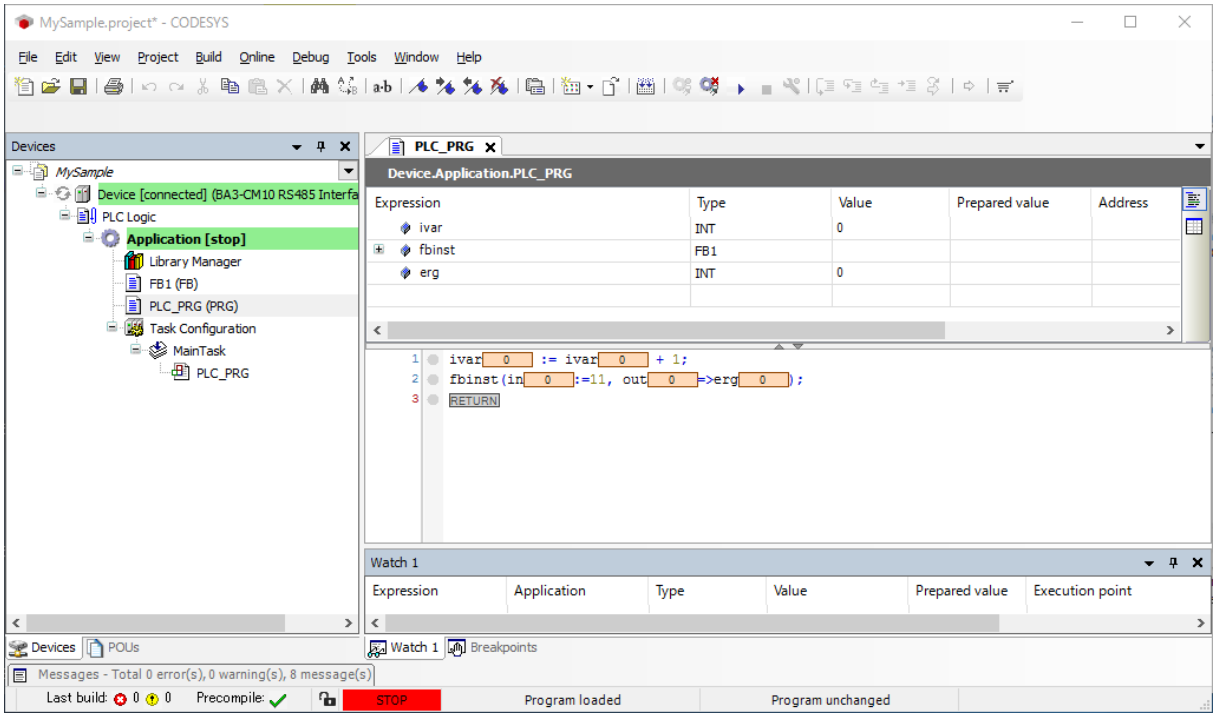
「コード」、「データ」、「RETAIN」、「PERSISTENT」の各領域の使用容量は「Build」から「Generate code」を実行することで事前に知ることもできます。

ターゲット内のアプリケーションを実行する

操作：アプリケーションの実行: [Debug] → [Start], アプリケーションの停止: [Debug] → [Stop]

アプリケーションを新規にダウンロードした直後の状態は「STOP」です。

アプリケーションの実行には状態を「RUN」に移行する必要があります。



ターゲット(コントローラ)にブートアプリケーションを作成する

ブートアプリケーションとは、コントローラに電源が投入され自動的に起動されるアプリケーションのことです。

ここでは、オンライン(ログイン)状態でのブートアプリケーション作成の手順を示します。

手順：[Online] → [Create boot application]

この操作で接続中の現在アクティブ化されているアプリケーションをブートアプリケーションとしてターゲットに登録します。

注 意

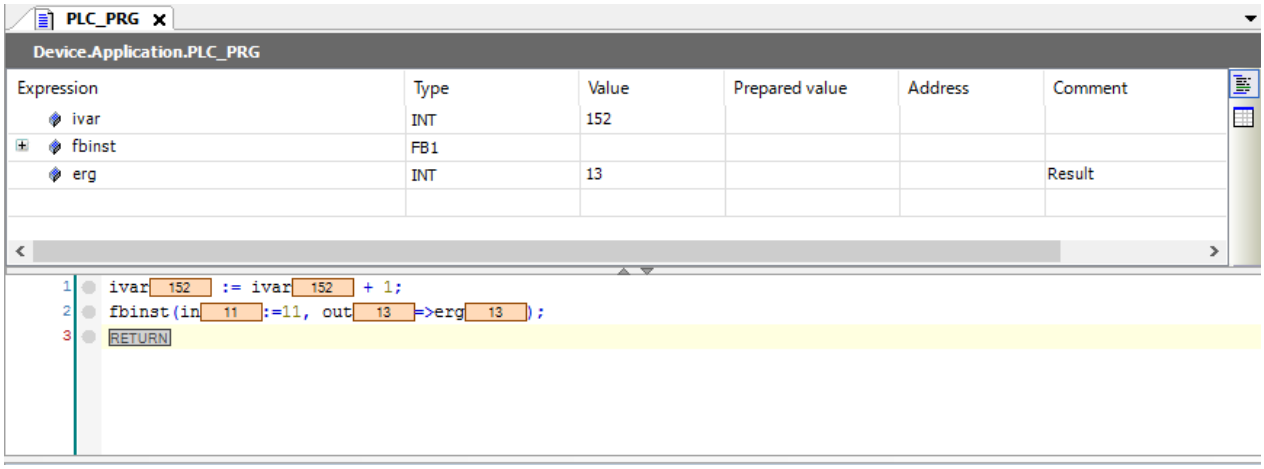
- ログインなどで転送されたアプリケーションは、ブートアプリケーションではありません。そのため次回電源再投入では以前作成したブートアプリケーションが起動されます。別途オプション設定によりログインなどのダウンロードで自動的にブートアプリケーションを作成するように指定できます(Applicationプロパティの[Boot application]タブ)。
- [online]モードのメニュー[Online] [Create boot application]で作成されるアプリケーションはファイルシステム(フラッシュメモリ)に格納されます。この書き込みには数十秒から数分時間を必要としますので、その間は書き込み処理中であることを示すERR LEDを点灯状態にします。このERR LEDが点灯している間は、ファイルの破損を防ぐために「リセット」や「電源のOFF」を行わないようにしてください。もし不慮の事態でファイルが破損した場合、次回電源投入かリセット時にファイルシステムは初期化(すべてのファイルが削除)されます。

6.4.デバッグ

オンラインモードで変数の現在値モニタ

オンラインモードは、変数の現在値やファンクション、ファンクションブロックの入出力パラメータの現在値をリアルタイムに表示します。

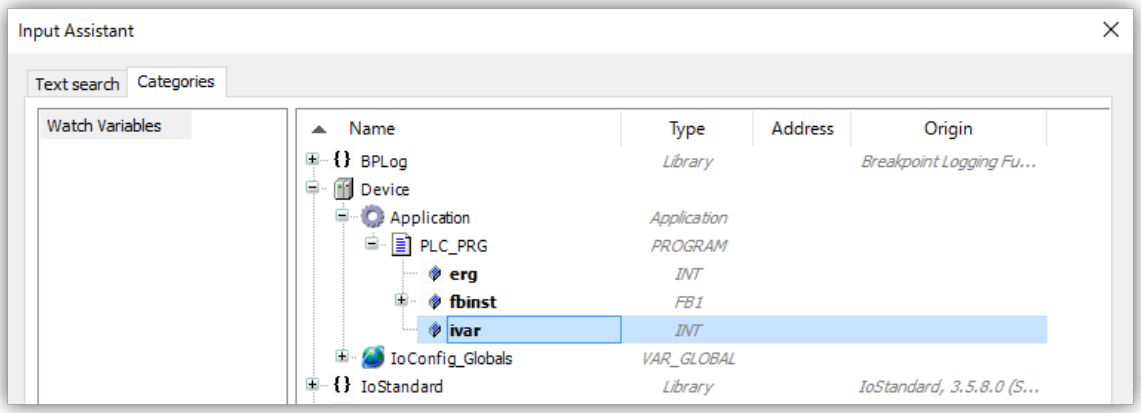
下図のようにプログラム本体部では、各変数にオレンジ色の四角の表示に現在の値がリアルタイムで表示されます。上側の変数宣言部では、プログラムに含まれる変数の一覧を見ることができます。また、特定の変数をまとめてモニタができるウォッチリスト機能もあります。

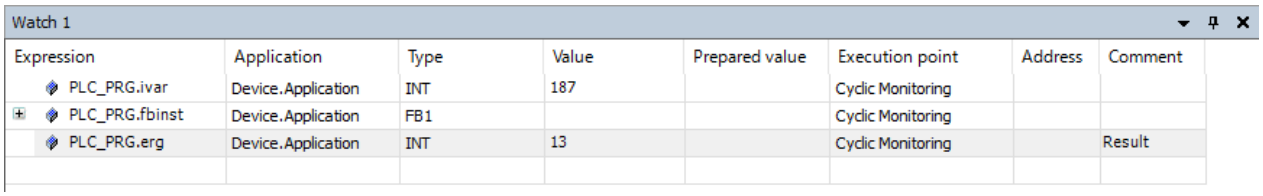


ウォッチウィンドウ

操作：[View] → [Watch] [Watch 1] を選択

画面下にウォッチウィンドウが開くので、ウォッチする変数式(例 Device.Application.PLC_PRG.ivar)を入力します。変数式の入力で [...] か[F2] キーを押すと入力アシスタントを利用できます。





"FB1"のようなファンクションブロックは、下図のようにメンバー変数がツリー表示されます。

Watch 1

Expression	Application	Type	Value	Prepared value	Execution point	Address	Comment
PLC_PRG.ivar	Device.Application	INT	296		Cyclic Monitoring		
PLC_PRG.fbinst	Device.Application	FB1			Cyclic Monitoring		
in		INT	11		Cyclic Monitoring		
out		INT	13		Cyclic Monitoring		
ivar		INT	2		Cyclic Monitoring		
PLC_PRG.erg	Device.Application	INT	13		Cyclic Monitoring		Result

オンラインモードで変数の現在値を設定変更

オンラインモードでは、必要に応じて変数値を書き替えることができます。

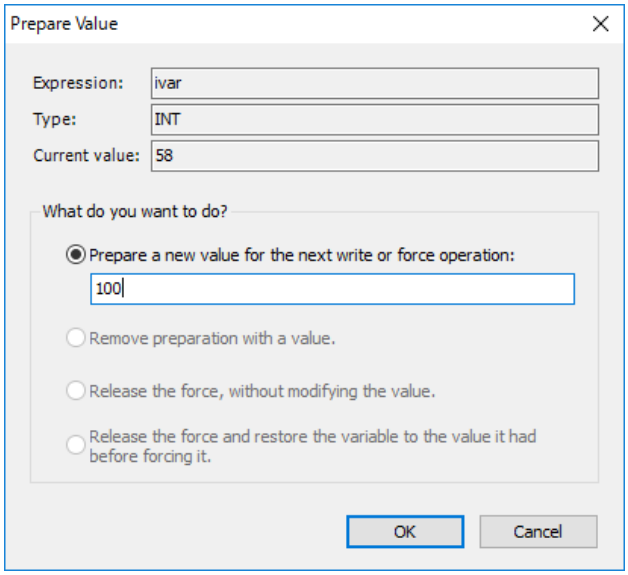
書き替えには、次の方法があります。

種類	手順	効果
値書き込み	メニュー「Debug」「Write values」 あるいは <Ctrl>+<F7>	次の実行周期の最初に指定の値を1度だけ書き込みます。
値の強制	設定： メニュー「Debug」「Force values」 または <F7> 解除： メニュー「Debug」「Unforce values」 または <Alt>+<F7>	実行周期の最初と最後で指定の値を毎周期書き込みます。 実行周期では次順で処理されます。 1．強制値書き込み 2．プログラムコードの実行 3．強制値書き込み

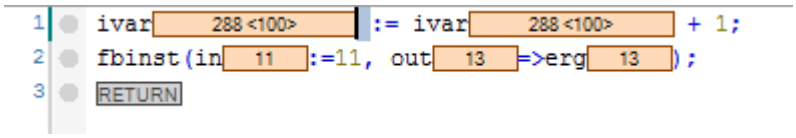
注 意

「値書き込み」や「値の強制」は実行中のプログラム動作に予想外の影響を与えることがあります。使用する場合は動作の影響範囲や安全に十分な配慮をしてください。

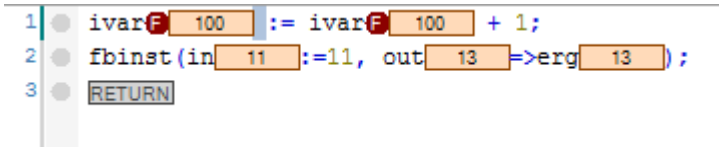
「値書き込み」、「値の強制」には「Prepared value」欄に書き込む値を事前に準備する必要があります。



「prepared value」欄に値を準備出来たら、「値書き込み」の場合は<Ctrl>+<F7> あるいは「値の強制」の場合は <F7> を押します。



下図は「値の強制」を行った場合の表示です。「Value」欄には、現在強制中であることを示す(F)が現在値の左に表示されています。



プログラムコードを任意の位置で停止

オンラインモードでは、必要に応じて停止ポイント(ブレークポイント)を設定できます。

プログラムコードの実行がブレークポイントに達するとプログラムの実行が一時的に停止されます。

停止状態からプログラムを再開する方法には、次の選択ができます。

1. 実行の再開

   「Run」

2. ステップ実行

「Step Over」: 現在命令の次の命令に進む

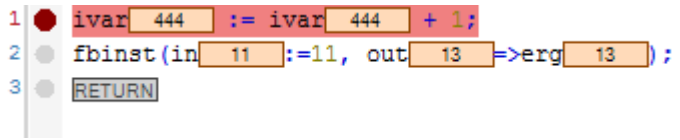
「Step Into」: 現在命令の内部に移動(命令のコードが表示可能な場合)

「Step Out」: 現在のファンクションあるいはファンクションブロックから呼び出すもとに戻る

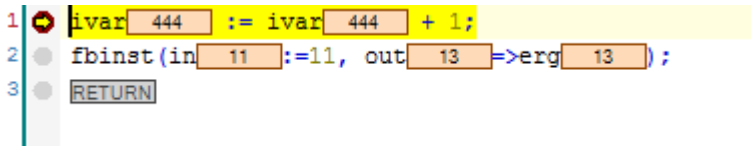
3. カーソル位置まで実行

「Run to Cursor」: 現在停止位置からカーソル位置の命令まで実行する

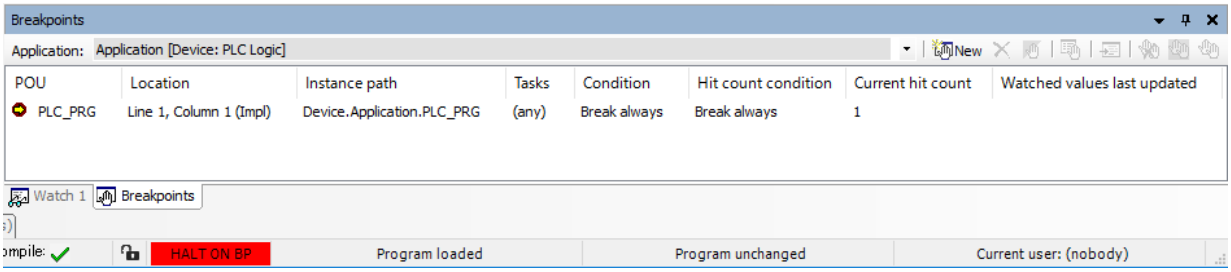
下図は、停止したい行(ここでは行番号1)にブレークポイントを設定した時の表示です。ブレークポイントが設定された行の先頭には ● が表示されます。



実行がブレークポイントに到達するとプログラムが一時停止し、シンボルが ➡ 表示に替わり該当行が黄色く反転表示されます。



「View」メニューの「Breakpoints」では、プログラム中のブレークポイントの一覧を見ることができます。



6.5.ターゲットとの接続を終了

ターゲットとの接続を終了(ログアウト)

操作 : [Online] → [Logout], またはツールバー : 

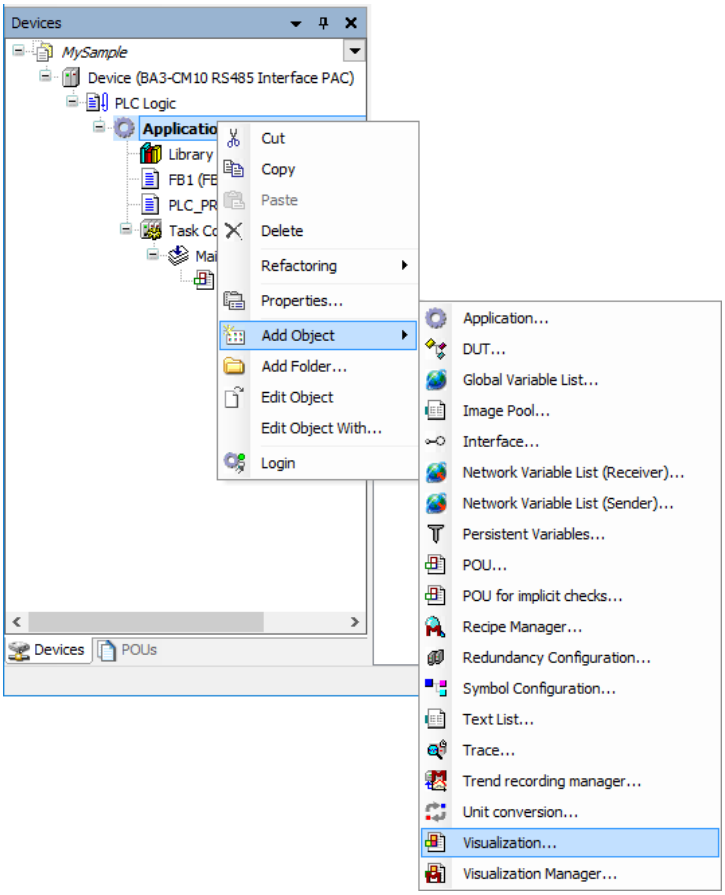
デバッグを終了してプログラムの修正や追加作業を行うには、ターゲットとの接続を一旦終了する必要があります。

6.6.HMI

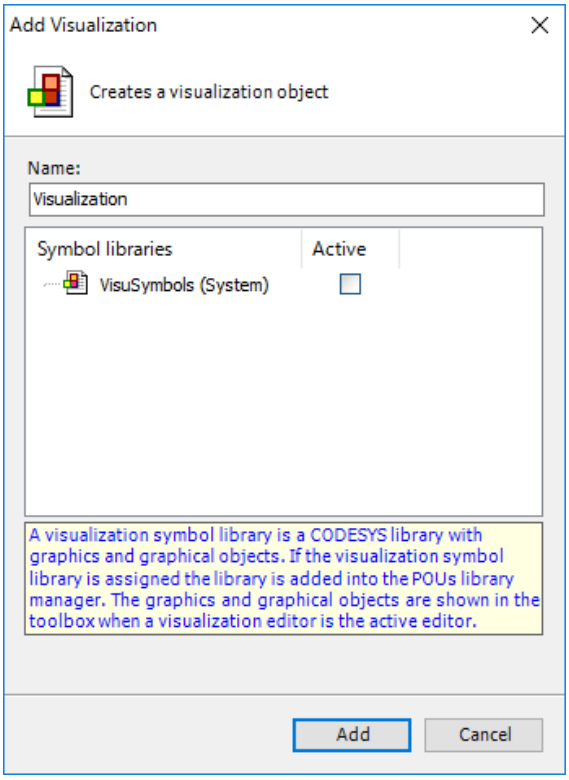
CODESYS IDEには、ユーザインタフェースの「Visualization」を作成する機能があります。このVisualization機能は、アプリケーションで使用している全ての内部変数をアクセスでき、Visualization画面から変数値の表示、書き換えができます。

HMIコンポーネントの追加

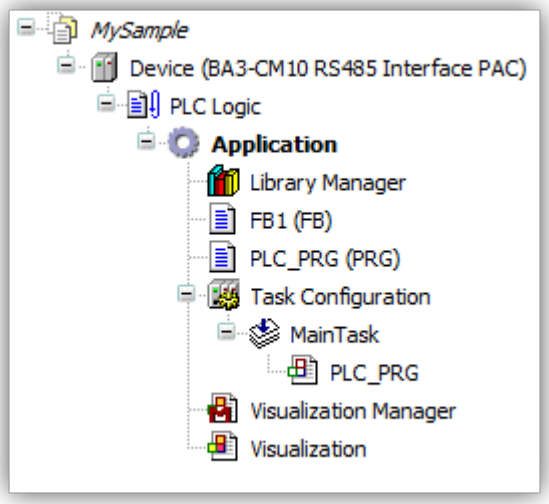
操作：デバイスツリー内のApplicationを選択した状態で [Project] → [Add Object] → [Visualization...]



下図ダイアログは、CODESYS提供のシンボルを使用したい場合にチェックを付けます。



下図のようにVisualizationが追加されます。



Visualizationの種類

CODESYSが提供するVisualizationには次の3種類があります。

- CODESYS HMI

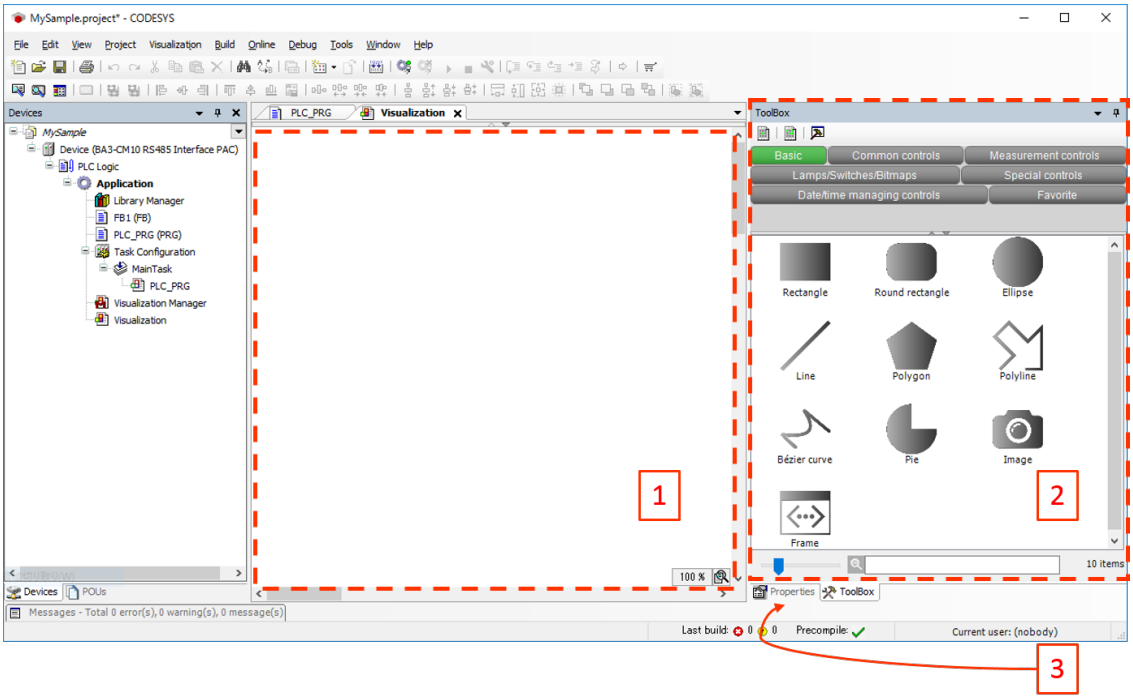
  HMI 表示先は、Windows PC (CODESYS IDE の Online monitor で表示されます)
- CODESYS Target Visualization

  HMI表示先は、コントローラ自体(ディスプレイを持つコントローラの場合)
- CODESYS Web Visualization

  HMI表示先は、HTML 5 CanvasElement の利用できるWeb Browser

Visualizationエディタ画面の説明

Visualizationエディタの画面



1	画面エディタ	実際の表示イメージにあらかじめ用意されている画面コンポーネント(部品)を配置する場所です。 画面の作成は、右側のツールボックスより部品をドラッグしてここに配置します。
2	ツールボックス	用意されている各種画面コンポーネントが一覧で表示されます。
3	プロパティ	エディタ画面に配置された部品のプロパティが一覧で表示され設定値の表示と変更ができます。

プログラムの修正

これから作成するVisualization画面で表示を変化させるために先で作成したプログラムを次のように修正します。変数にはBOOL型のflgを追加し、プログラムはivarの結果を0～99に制限する演算と、flg値をivar値が50以上でTRUE、それ以外でFALSEとする演算を追加します。このプログラムが実行されるとivarは初期値0から1づつ加算され99になると次に0になります。結果flg値はFALSE→TRUEを繰り返すことになります。

変数宣言部

```
PROGRAM PLC_PRG
VAR
    ivar: INT;
    fbinst: FB1;
    // Result
    erg: INT;
    flg: BOOL;
END_VAR
```

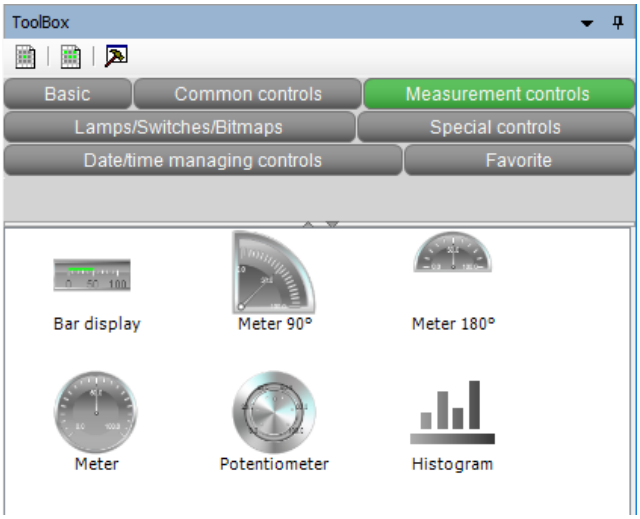
プログラム本体部

```
ivar := (ivar + 1) MOD 100;
flg := ivar >= 50;
fbinst(in:=11, out=>erg);
```

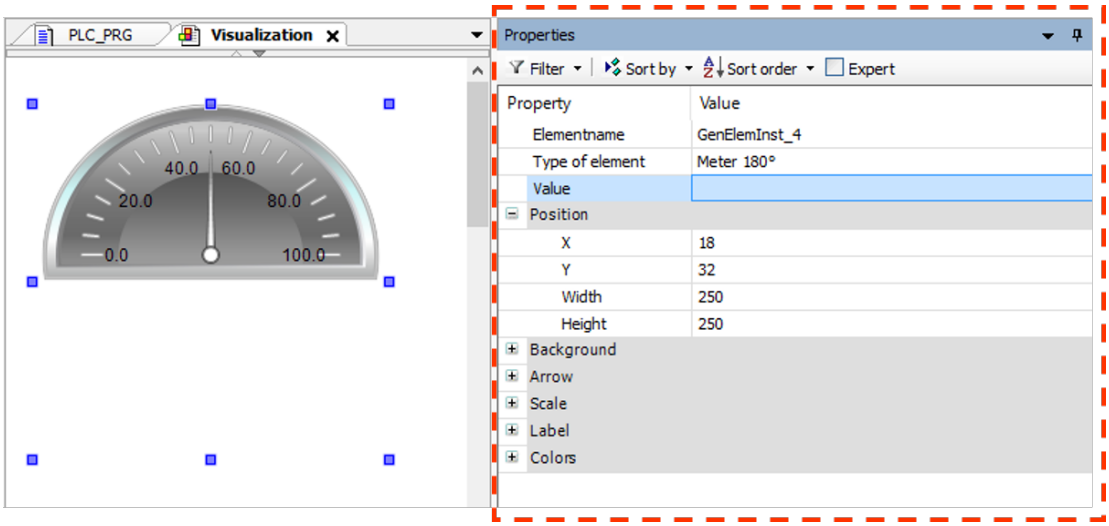
画面のデザイン

操作：デバイスツリー内のVisualizationをダブルクリックします。

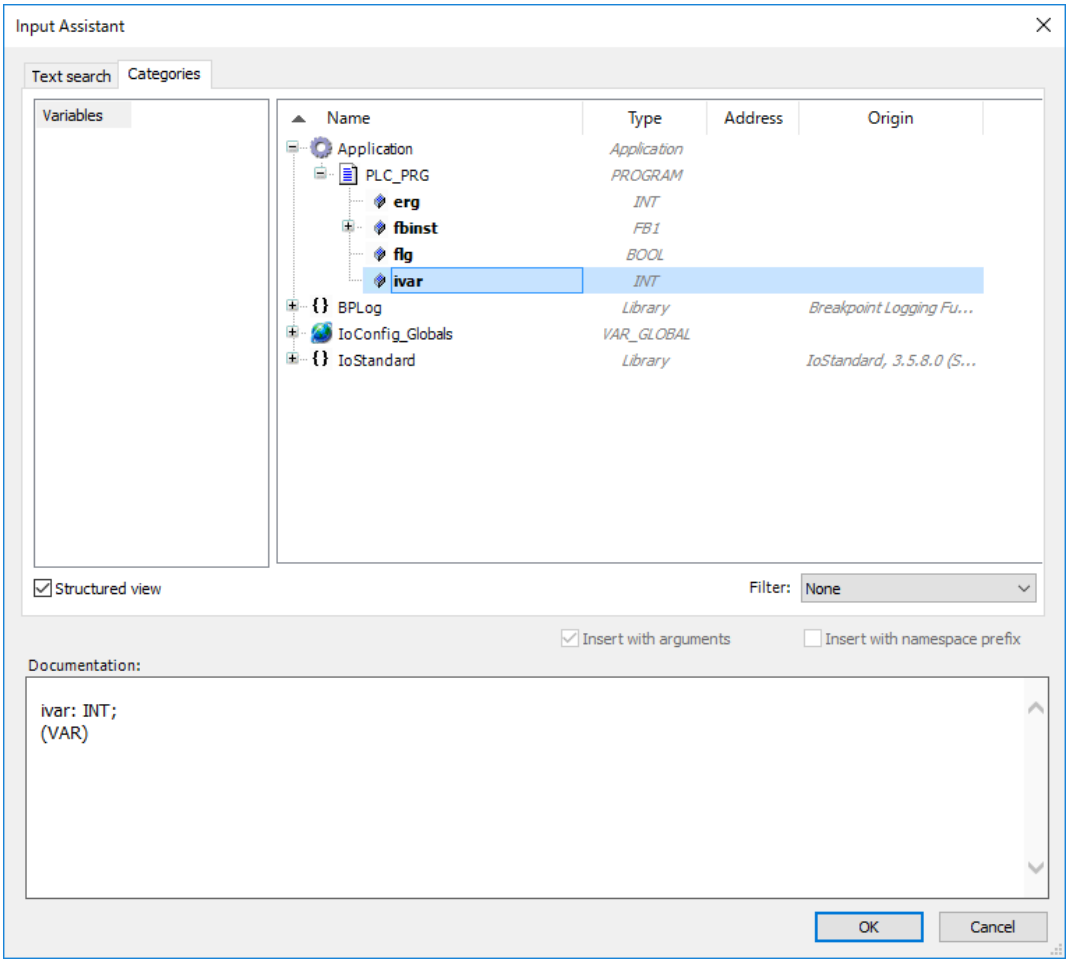
右側のツールボックスから「Meter 180°」をドラッグして左側の画面エディタに配置します。



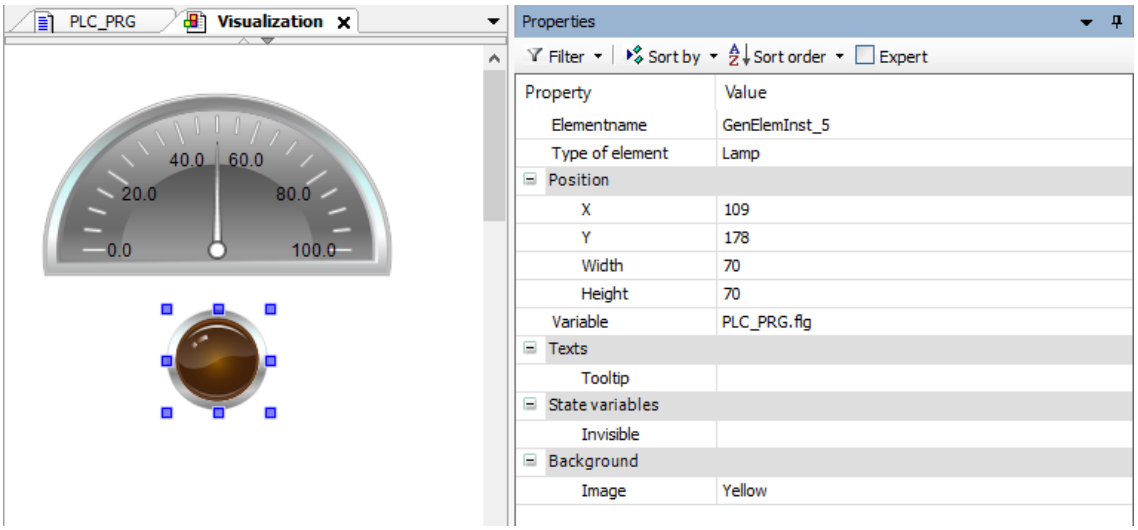
メータ部品が配置されるとプロパティが表示されます。このメータと変数ivarを関連付けます。画面に配置したメータをクリックすると右側にプロパティー覧が表示されます。



メータの針が示す値は、プロパティ「Value」で指定できます。「Value」の値欄をクリックして表示される[...]ボタンか[F2]キーを押して「入力アシスタント」から関連付ける変数を選択します。下図でApplication.PLC_PRG.ivarを選択し「OK」を押します。



同様に、ツールボックスの「Lamps/Switches/Bitmaps」グループから、LEDランプ型コンポーネント「Lamp1」を画面エディタに配置し、明滅状態を指定するプロパティ「Value」を変数PLC_PRG.flgに関連付けします。

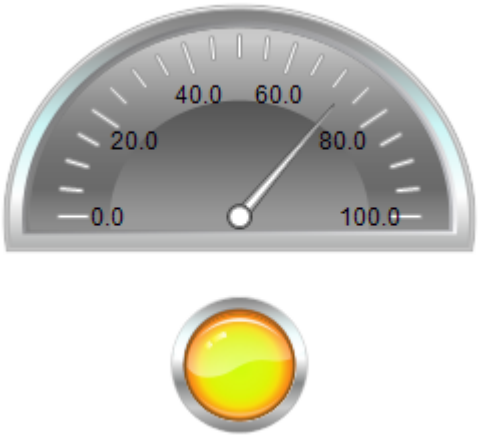


実行

操作 : [Online] → [Login], またはツールバー :

Visualizationの無い場合と同様に、ターゲットにログインした後に実行します。もし、前の章で設定したブレークポイントが残っている場合は、実行の前に全て解除しておいてください。

実行するとCODESYS IDE 内のVisualization画面表示が実行状態(オンライン)に移行します。



メータの針は、関連付けられた変数ivar値に従い0から99の範囲を繰り返し表示します。また、ランプの色は、関連付けられた変数flg値に従いFALSE(暗い黄色)、TRUE(明るい黄色)と明滅することが確認できます。

7.ライブラリ

ライブラリは、プログラムを再利用する最も有効な方法です。自身の作成したファンクションやファンクションブロックを他のアプリケーションで使用したり、他の開発者に提供することを容易にします。

ここでは、ユーザライブラリの作成方法から既存のライブラリの説明を行います。

7.1.ユーザライブラリ

ライブラリの作成は、テンプレートを使用する場合と新規(空)で作成する方法とがあります。

テンプレートを利用した場合は、多くのモジュールが含まれているので完成後には不要なモジュールを削除されることをお勧めします。

CODESYS CAAのガイドラインに準拠したライブラリの作成には次のテンプレートを使用します。

[CODESYS container library], [CODESYS interface library], [CODESYS library]

この中で[External CODESYS library] は使用しません。

この章では、ライブラリとしての最小限の要素を持つライブラリを新規(空)から作成する方法を説明します。

詳細についてはオンラインヘルプの「Guidelines for creating libraries」を参照してください。

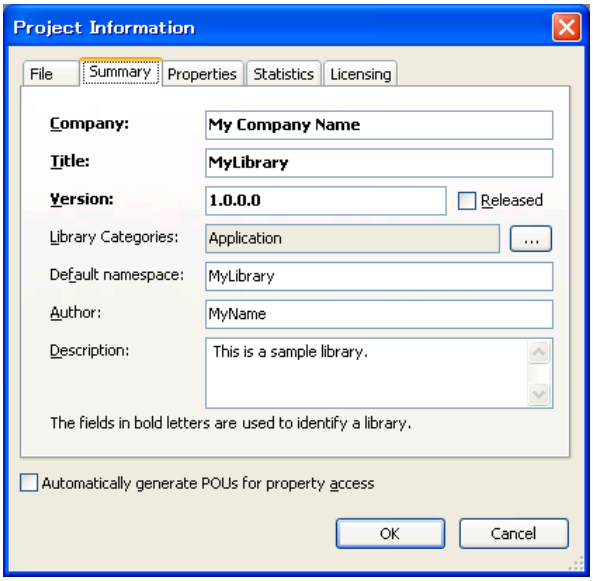
ユーザライブラリの作成

メニュー [File] [New Project...] のダイアログで [Categories]=Library, [Templates]=Empty library を選択します。

ライブラリのプロジェクト情報を設定

左側に表示される「Deviceツリー」のタブ[POUs]を選択しておきます。

メニュー [Project] [Project Information...]



Company: <My Company Name>

Title: <MyLibrary>

Version: <1.0.0.0>

Library Categories: 次に説明します。

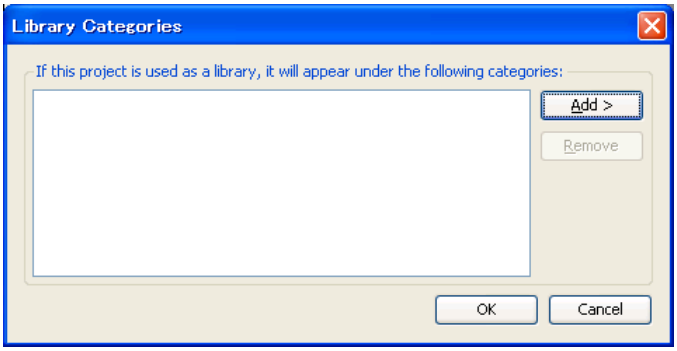
Default namespace: <MyLibrary>

Author: <MyName>

Description: <This is a sample library.>

これらの項目を入力します。

[Library categories]の入力は、右端に位置する [...] ボタンを押します。

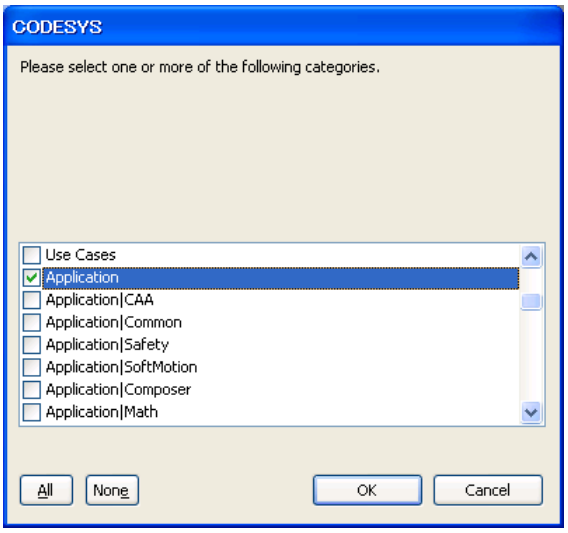


[Add>] [From Description File...]

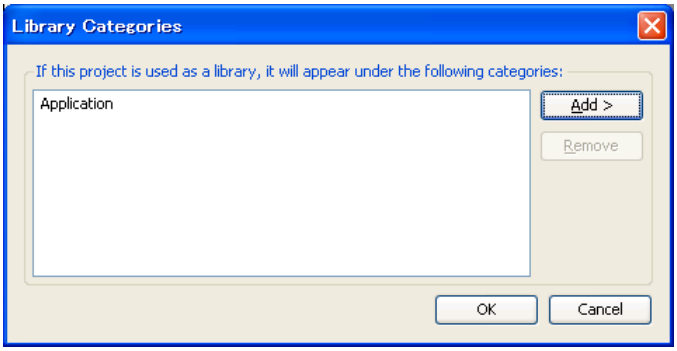
[Windows 10の場合]

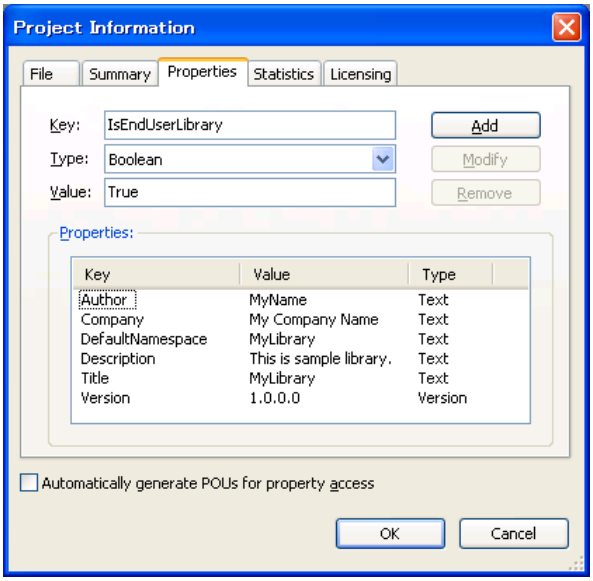
C:\Program Files (x86)\m-system\CODESYS V3 Tools\CODESYS\Templates\Library_Template

LibraryCategoryBase.libcat.xml



全選択状態なので[None]を押し一旦解除しておき、リストの[Application]だけを選択する。





ここで次のプロパティを手動で追加します。

Key: IsEndUserLibrary

Type: Boolean

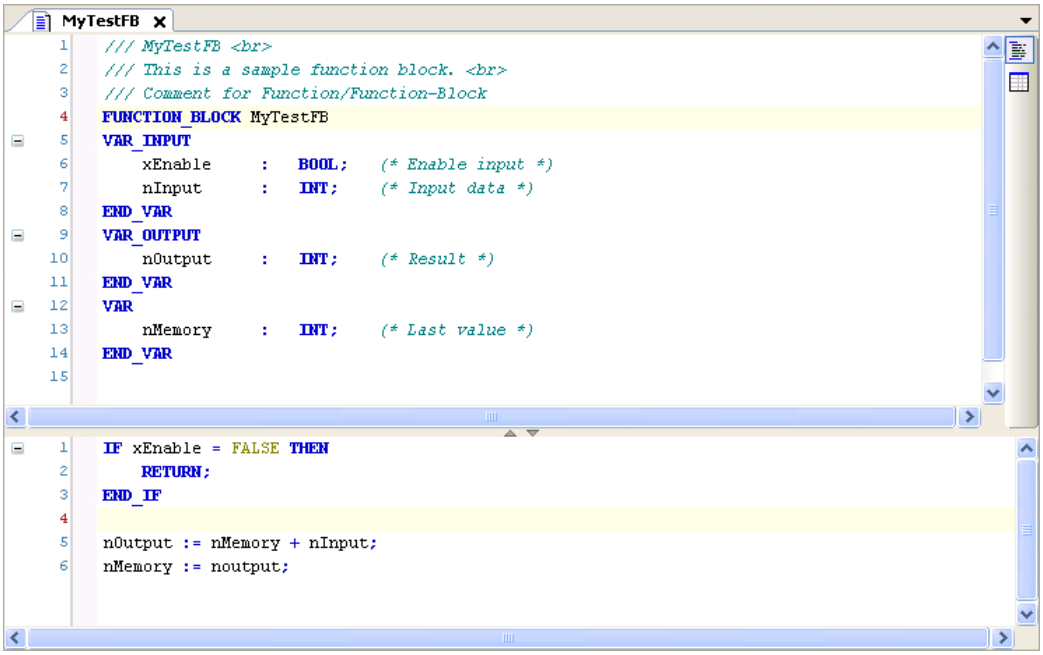
Value: True

これらを入力後に追加[Add]ボタンを押します。

ライブラリに自身のオブジェクトを追加

左側に表示される「Deviceツリー」の最初の項目(ファイル名と同じ文字)を選択します。

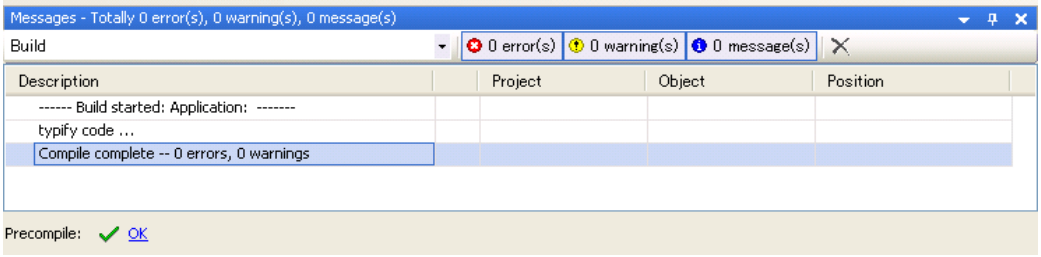
メニュー[Project][Add Object] で表示されるオブジェクトからDUTやPOUなどライブラリにしたいプログラムを追加していきます。



ライブラリのエラー確認

ライブラリの確認は[Build] [Build] ではなく[Build] [Check all Pool Objects] で行います。

もしエラーが検出されたなら修正し解消してください。



ライブラリの種類

ライブラリは、作成時に使用しているライブラリプロジェクトのまま使用、配布することもできますが、ソースコードを公開しない「コンパイル済みライブラリ」を作成することができます。

ライブラリには、次の形式があります。

- ライブラリプロジェクト (*.library)

  ソースコード参照可能ですが、プロテクトオプションで制御可能

- コンパイル済みライブラリ(*.compiled-library)

  ソースコード参照不可

「コンパイル済みライブラリ」の作成は、開発環境でライブラリプロジェクト(*.library)を開きメニュー [File] [Save Project As Compiled Library...] を押します。

ここで作成された「コンパイル済みライブラリ」は、自動的にリポジトリへの登録は行われていません。

このライブラリを使用するには、次の「リポジトリ登録」作業を行う必要があります。

ライブラリの公開(リポジトリ登録)

作成したライブラリや配布されたライブラリをアプリケーションから使用するためには、ライブラリ・リポジトリへの登録が必要となります。

ライブラリ・リポジトリへの登録は、前述のどちらのライブラリ形式でも可能です。

- **外部から配布されたライブラリをリポジトリに登録するには**

  メニュー [Tool] [Library Repository...] で表示されるダイアログの [Install...] を押します。

  登録したいライブラリ(*.library あるいは *.compiled-library)を選択します。

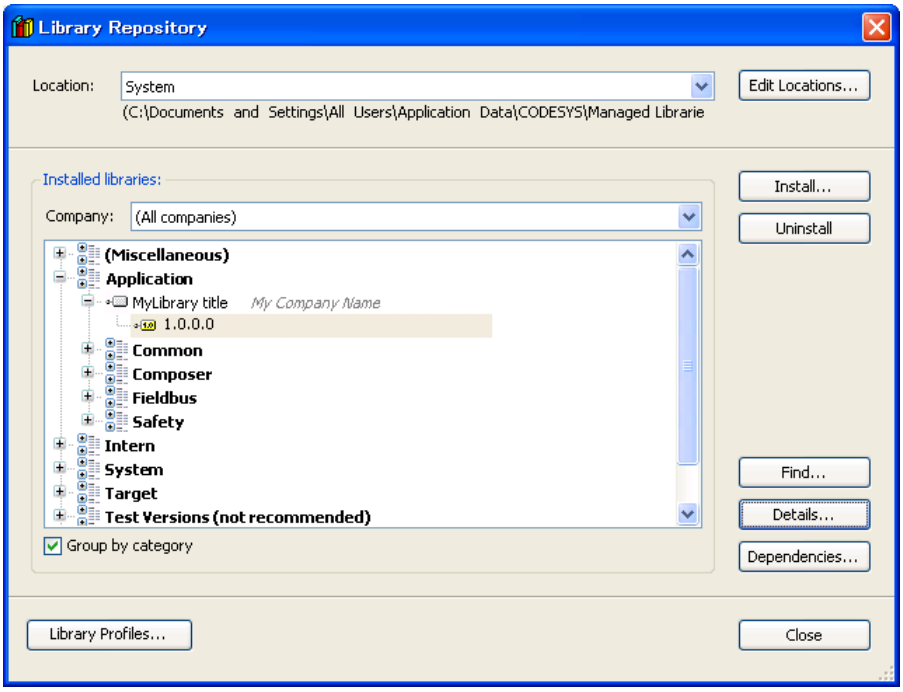
- **開発環境で現在開いているライブラリプロジェクトを登録するには**

  メニュー [File] [Save Project And Install Into Library Repository] を実行します。

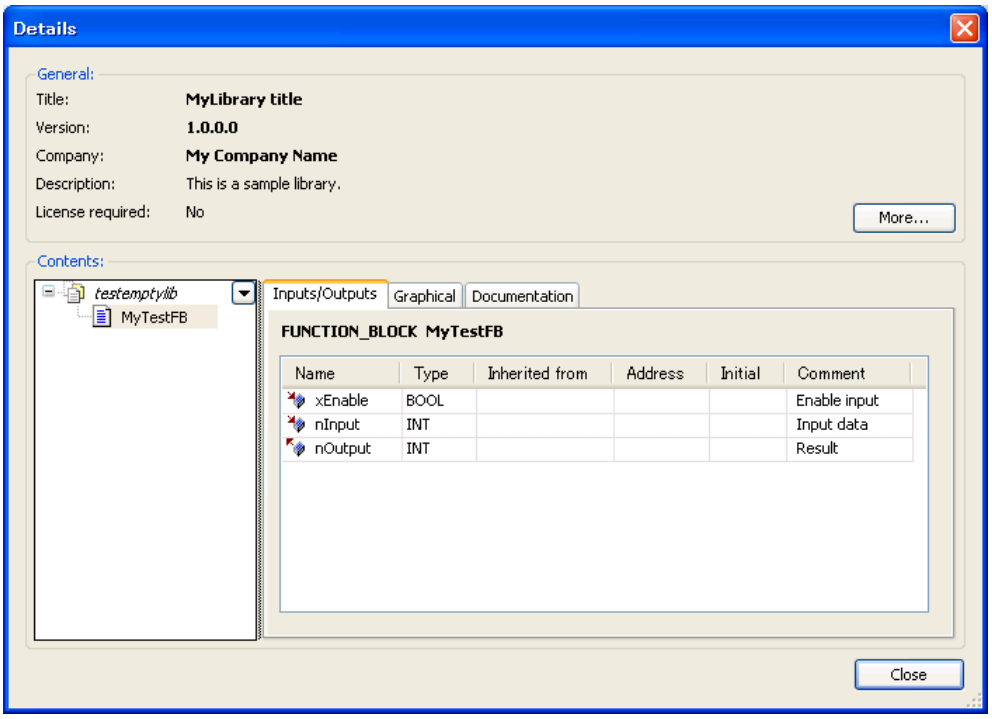
公開されているライブラリの確認

登録済みライブラリの情報は、「ライブラリマネージャ」か「ライブラリ・リポジトリ」画面で確認できます。

ここでは「ライブラリ・リポジトリ」画面での操作を紹介します。



先ほど作成したライブラリを登録した場合は、[Application]の下に[MyLibrary title]が登録されているのが確認できます。このライブラリの詳細情報を表示するためにバージョン[1.0.0.0]を選択し[Details...]ボタンを押します。

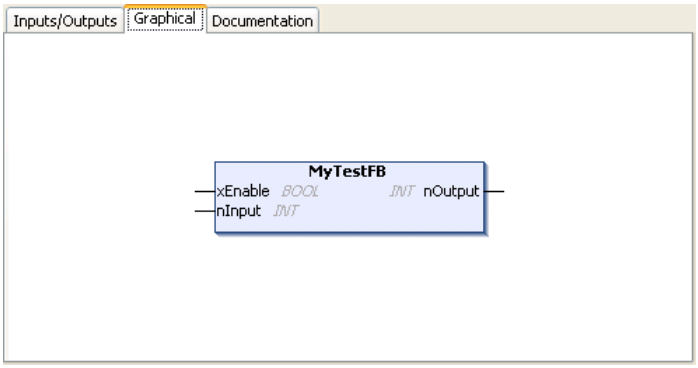


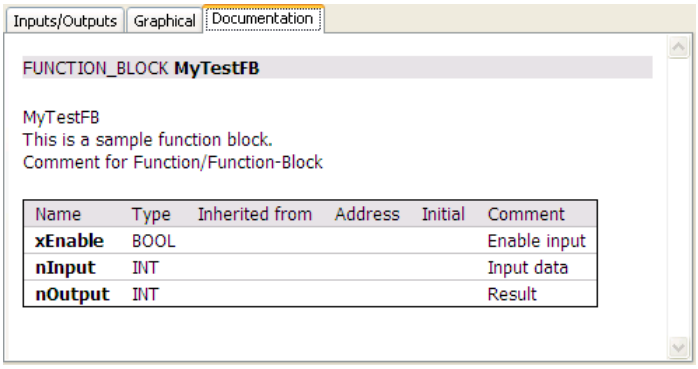
このライブラリの詳細情報が表示されます。

ツリー内には、ライブラリ内で公開されているDUTやPOUなどが表示されます。

ここでは、先で作成したファンクション・ブロック[MyTestFB]が確認できます。

ツリー内でファンクションブロック名を選択すると、その入力出力定義やグラフィックエディタでの表示や説明を確認できます。





ここでは、ファンクション・ブロックを作成する際に記述したコメントが説明文の一部として現れます。プログラムする際には、この自動ドキュメント化機能を意識したコメントをプログラムと併せて記述することで個別ドキュメントを別途用意する工数の削減とプログラムとドキュメントの一元管理が容易となります。

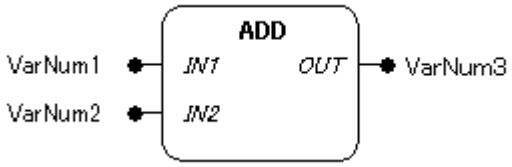
7.2.演算子

分類	命令	機能
算術演算	ADD	加算
	MUL	乗算
	SUB	減算
	DIV	除算後の商
	MOD	除算後の余り
	MOVE	転送
	INDEXOF *1	インデックスの取得
	SIZEOF *1	変数の占有サイズの取得
論理演算	AND	論理積
	OR	論理和
	XOR	ビットデータ排他的論理和
	NOT	ビットデータ反転
ビットシフト演算	SHL	左ビットシフト
	SHR	右ビットシフト
	ROL	左ビットローテーション
	ROR	右ビットローテーション
データ選択	SEL	データ選択
	MAX	最大値選択
	MIN	最小値選択
	LIMIT	上下限制限
	MUX	マルチプレクサ
比較演算	GT	IN1 ＞ IN2
	LT	IN1 ＜ IN2
	LE	IN1 ≦ IN2
	GE	IN1 ≧ IN2
	EQ	IN1 ＝ IN2
	NE	IN1 ≠ IN2
アドレス演算	ADR *1	アドレス取得
	BITADR *1	DWORD中のビットオフセット取得

分類	命令	機能
数値演算	ABS	絶対値
	SQRT	平方根
	LN	自然対数
	LOG	常用対数
	EXP	e の指数累乗
	SIN	サイン(入力はラジアン)
	COS	コサイン(入力はラジアン)
	TAN	タンジェント(入力はラジアン)
	ASIN	アークサイン(結果はラジアン)
	ACOS	アークコサイン(結果はラジアン)
	ATAN	アークタンジェント(結果はラジアン)
	EXPT	べき乗

*1) IEC61131-3 非準拠

ADD:加算



機能

入力パラメータに接続された値の加算結果(IN1 + IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME, TIME_OF_DAY (TOD), DATE, DATE_AND_TIME (DT)

次の組み合わせも可能です: TIME+TIME=TIME, TOD+TIME=TOD, DT+TIME=DT

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	被加数	
IN2 (VarNum2)	加数	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarNum3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

FB/LDエディタにおけるADDオペレータは入力数を拡張できます。

(ILの例)

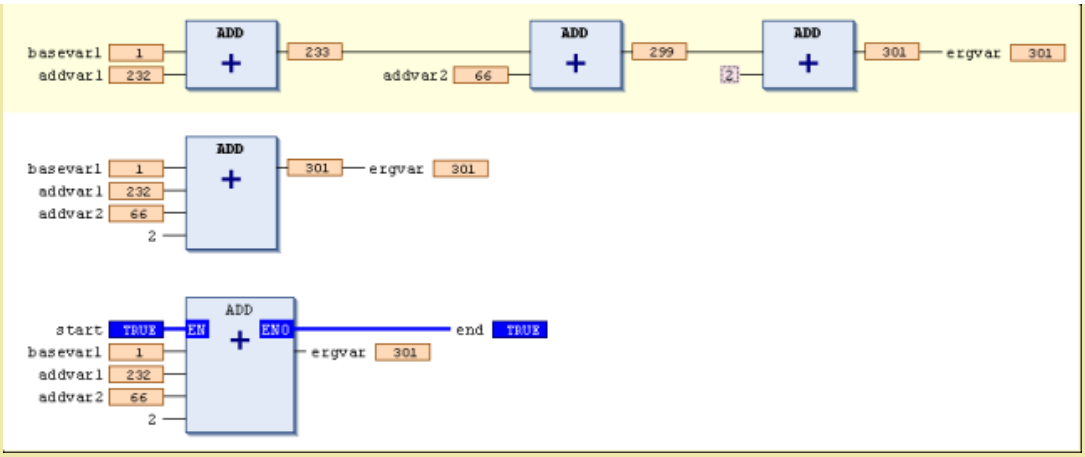
```
LD   7
ADD  2
ADD  4
ADD  7
ST   iVar
```

(STの例)

```
var1 := 7+2+4+7;
```

(FBDの例)

1.連続させたADD, 2.拡張したADD, 3.EN/ENO付のADD



MUL:乗算

機能

入力パラメータに接続された値の乗算結果(IN1 × IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

TIME型変数は整数型変数と乗算できます。

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	被乗数	
IN2 (VarNum2)	乗数	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarNum3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

FB/LDエディタにおけるMULオペレータは入力数を拡張できます。

(ILの例)

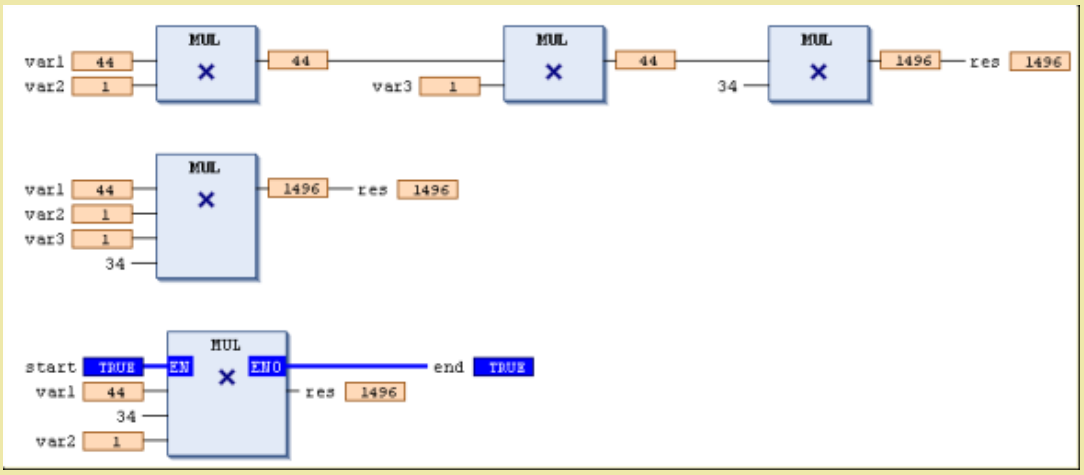
```
LD    7
MUL   2,
      4,
      7
ST    Var1
```

(STの例)

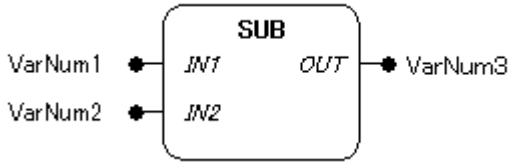
```
var1 := 7*2*4*7;
```

(FBDの例)

1.連続させたMUL, 2.拡張したMUL, 3.EN/ENO付のMUL



SUB:減算



機能

入力パラメータに接続された値の減算結果(IN1 - IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME, TIME_OF_DAY (TOD), DATE, DATE_AND_TIME (DT)
次の組み合わせも可能です: TIME-TIME=TIME, DATE-DATE=TIME, TOD-TIME=TOD, TOD-TOD=TIME, DT-TIME=DT, DT-DT=TIME
負のTIME値は未定義と見なされます。

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	被減数	
IN2 (VarNum2)	減数	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarNum3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

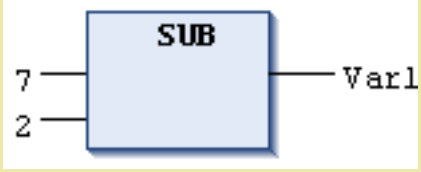
(ILの例)

```
LD   7
SUB  2
ST   Varl
```

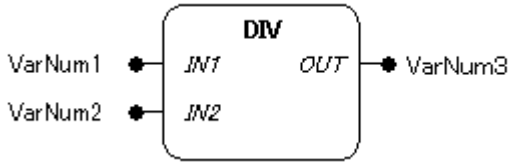
(STの例)

```
var1 := 7-2;
```

(FBDの例)



DIV:除算



機能

入力パラメータに接続された値の除算結果(IN1 ÷ IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

TIME変数は整数変数で除算できます。

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	被除数	
IN2 (VarNum2)	除数	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarNum3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結:果Var1 の値は 4)

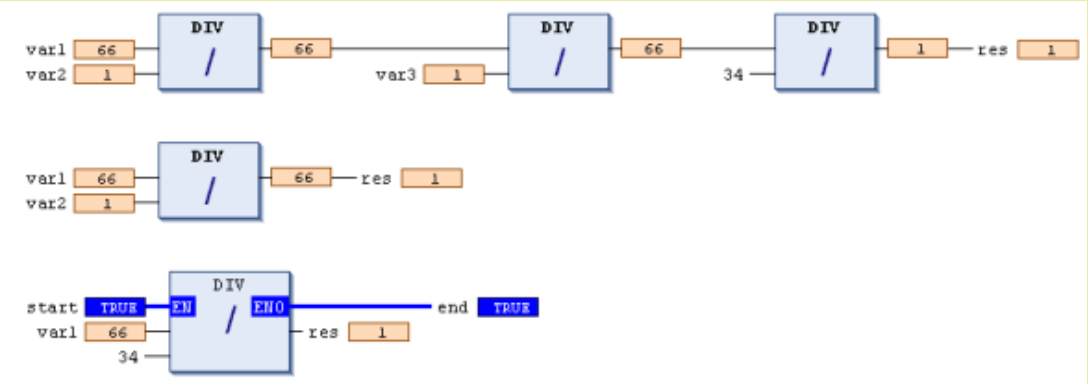
```
LD      8
DIV     2
ST      Var1
```

(STの例)

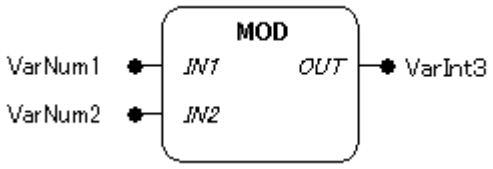
```
var1 := 8/2;
```

(FBDの例)

1.連続させたDIV, 2.単体でのDIV, 3.EN/ENO付のDIV



MOD : 除算の余り



機能

入力パラメータに接続された値を除算した余り(IN1 MOD IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	被除数	
IN2 (VarNum2)	除数	

出力パラメータ	説明	備考
OUT (VarInt3)	結果	整数データ型

(ILの例) (結果: Var1 の値は 1)

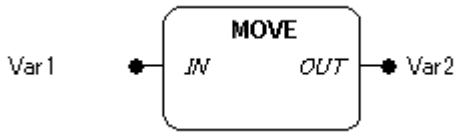
```
LD    9
MOD   2
ST    Var1
```

(STの例)

```
Var1 := 9 MOD 2;
```

(FBDの例)

MOVE:転送



機能

入力パラメータに接続された値を出力パラメータに接続された変数に転送(IN = OUT)します。

パラメータで使用可能なデータ型

全てのデータ型が使用できます。

パラメータ

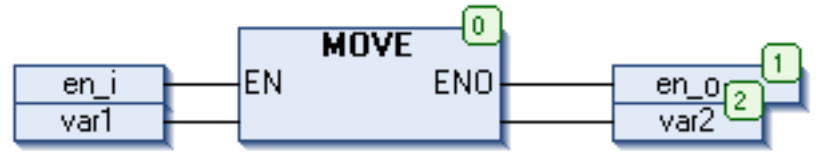
入力パラメータ	説明	備考
IN (Var1)	入力	

出力パラメータ	説明	備考
OUT (Var2)	出力	INと同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

EN/ENO付のCFC 接続例：

en_i が TRUE の場合に var1 の値が var2 に代入されます。



(ILの例) (結果： var2 が var1 の値になります)

```
LD      var1
MOVE
ST      var2
```

次も同じ結果となります:

```
LD      var1
ST      var2
```

(STの例)

```
ivar2 := MOVE(ivar1);
```

(次も同じ結果となります: ivar2 := ivar1;)

INDEXOF : インデックスの取得

機能

与えられたオブジェクトのPOUインデックスを返します。

この演算子は使用されなくなります。代替えとしてADR演算子を使用してください。

SIZEOF : 変数の占有サイズの取得

機能

与えられた変数が占有する記憶領域のサイズをバイト単位で返します。

パラメータ

入力パラメータ	説明	備考
-		

出力パラメータ	説明	備考
占有バイトサイズ	出力（返される数によりデータ型が決定）	返されるデータ型
	0 <= size of x < 256	USINT
	256 <= size of x < 65536	UINT
	65536 <= size of x < 4294967296	UDINT
	4294967296 <= size of x	ULINT

(ILの例) (結果 10)

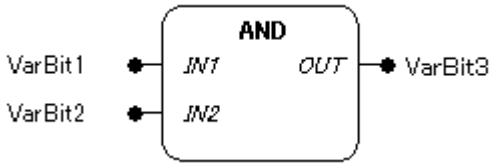
```
arr1:ARRAY[0..4] OF INT;
Var1:INT;
```

```
LD      arr1
SIZEOF
ST      Var1
```

(STの例)

```
var1 := SIZEOF(arr1);  (* 等価 var1:=USINT#10; *)
```

AND:論理積



機能

入力パラメータに接続された値の論理積結果(IN1 AND IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BOOL, BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN1 (VarBit1)	入力１	
IN2 (VarBit2)	入力２	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBit3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果 Var1 の値は 2#1000_0010)

```
Var1:BYTE;
```

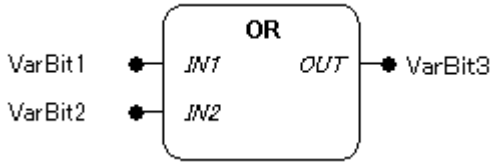
```
LD      2#1001_0011
AND     2#1000_1010
ST      var1
```

(STの例)

```
var1 := 2#1001_0011 AND 2#1000_1010
```

(FBDの例)

OR:論理和



機能

入力パラメータに接続された値の論理和結果(IN1 OR IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BOOL, BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN1 (VarBit1)	入力1	
IN2 (VarBit2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBit3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果 var1 の値は 2#1001_1011)

```
    var1 :BYTE;
```

```
LD      2#1001_0011
OR      2#1000_1010
ST      Var1
```

(STの例)

```
Var1 := 2#1001_0011 OR 2#1000_1010;
```

(FBDの例)

XOR:排他的論理和

機能

入力パラメータに接続された値の排他的論理和結果(IN1 XOR IN2 = OUT)を出力します。

パラメータで使用可能なデータ型

BOOL, BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN1 (VarBit1)	入力1	
IN2 (VarBit2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBit3)	結果	IN1と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果 var1 の値は 2#0001_1001)

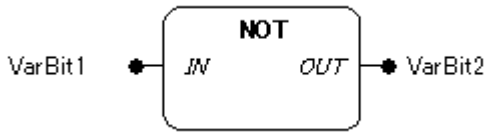
```
LD      2#1001_0011
XOR     2#1000_1010
ST      var1
```

(STの例)

```
Var1 := 2#1001_0011 XOR 2#1000_1010;
```

(FBDの例)

NOT:ビットデータ反転



機能

入力パラメータに接続された値のビット単位の反転(NOT:1の補数)結果(IN NOT = OUT)を出力します。

パラメータで使用可能なデータ型

BOOL, BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN (VarBit1)	入力	

出力パラメータ	説明	備考
OUT (VarBit2)	結果	INと同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果 Var1 の値は 2#0110_1100)

```
Var1 :BYTE;
```

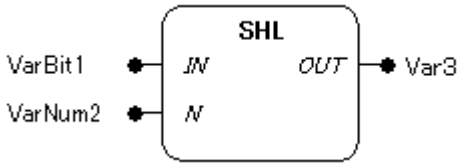
```
LD    2#1001_0011
NOT
ST    var1
```

(STの例)

```
Var1 := NOT 2#1001_0011 ;
```

(FBDの例)

SHL : 左ビットシフト



機能

入力パラメータINに接続した値を、Nに接続したビット数だけ左シフトした結果を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN (VarBit1)	シフトされるデータ	ここに与えるデータ型のサイズが演算の最大ビット数となり、 ここに定数を与えた場合その値が格納できる最小データ型の サイズで演算されます。
N (VarNum2)	シフトするビット数	0から入力パラメータINのデータ型の最大ビット数までが有効であり、 負の値を与えるとシフト方向が逆になります。

出力パラメータ	説明	備考
OUT (Var3)	結果	このデータ型のサイズはシフト演算に影響しません。 INデータ型が符号付きであれば符号付きのデータ型とする

解 説

左シフト命令では指定のNビットだけ左へシフトすると共に右のビットに0を補充します。

データ型がWORDの場合

2#0000_0000_0100_1011の2ビット左シフトの結果は2#0000_0001_0010_1100

2#0000_0000_0100_1011の4ビット左シフトの結果は2#0000_0100_1011_0000

データ型がBYTEの場合

2#0100_1011の2ビット左シフトの結果は2#0010_1100

2#0100_1011の4ビット左シフトの結果は2#1011_0000

(ILの例)

```
LD    in_byte
SHL   2
ST    erg_byte
```

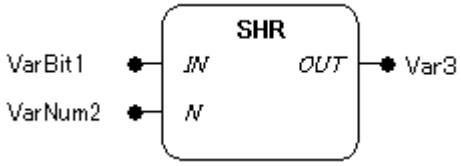
(STの例)

次の例は入力変数(BYTEまたはWORD)のデータ型に対応したerg_byteとerg_wordのそれぞれの結果を16進で表しています。

```
PROGRAM shl_st
VAR
     in_byte : BYTE:=16#45; (* 2#01000101 )
     in_word : WORD:=16#0045; (* 2#0000000001000101 )
     erg_byte : BYTE;
     erg_word : WORD;
     n: BYTE :=2;
END_VAR
erg_byte:=SHL(in_byte,n); (* 結果 16#14, 2#00010100 *)
erg_word:=SHL(in_word,n); (* 結果 16#0114, 2#0000000100010100 *)
```

(FBDの例)

SHR:右ビットシフト



機能

入力パラメータINに接続した値を、Nに接続したビット数だけ右シフトした結果を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD

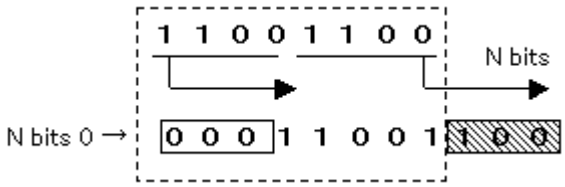
パラメータ

入力パラメータ	説明	備考
IN (VarBit1)	シフトされるデータ	ここに与えるデータ型のサイズが演算の最大ビット数となり、 ここに定数を与えた場合その値が格納できる最小データ型の サイズで演算されます。
N (VarNum2)	シフトするビット数	0から入力パラメータINのデータ型の最大ビット数までが有効であり、 負の値を与えるとシフト方向が逆になります。

出力パラメータ	説明	備考
OUT (Var3)	結果	このデータ型のサイズはシフト演算に影響しません。 INデータ型が符号付きであれば符号付きのデータ型とする

解 説

右シフト命令では指定のNビットだけ右へシフトすると共に左のビットに0を補充します。



データ型がWORDの場合

2#0100_1011_0000_0000の2ビット右シフトの結果は2#0001_0010_1100_0000

2#0100_1011_0000_0000の4ビット右シフトの結果は2#0000_0100_1011_0000

データ型がBYTEの場合

2#0100_1011の2ビット右シフトの結果は2#0001_0010

2#0100_1011の4ビット右シフトの結果は2#0000_0100

(ILの例)

```
LD    in_byte
SHR   2
ST    erg_byte
```

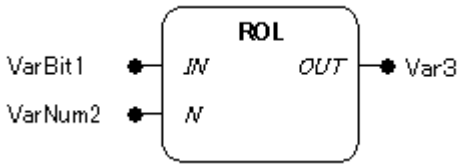
(STの例)

次の例は入力変数(BYTEまたはWORD)のデータ型に対応したerg_byteとerg_wordのそれぞれの結果を16進で表しています。

```
PROGRAM shr_st
VAR
     in_byte : BYTE:=16#45; (* 2#01000101 )
     in_word : WORD:=16#0045; (* 2#0000000001000101 )
     erg_byte : BYTE;
     erg_word : WORD;
     n: BYTE :=2;
END_VAR
erg_byte:=SHR(in_byte,n); (* 結果 16#11, 2#00010001 *)
erg_word:=SHR(in_word,n); (* 結果 16#0011, 2#0000000000010001 *)
```

(FBDの例)

ROL:左ビットローテーション



機能

入力パラメータINに接続した値を、Nに接続したビット数だけ左ローテーションした結果を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN (VarBit1)	ローテーションされるデータ	ここに与えるデータ型のサイズが演算の最大ビット数となり、ここに定数を与えた場合その値が格納できる最小データ型のサイズで演算されます。
N (VarNum2)	ローテーションするビット数	0から入力パラメータINのデータ型の最大ビット数までが有効であり、負の値を与えるとシフト方向が逆になります。

出力パラメータ	説明	備考
OUT (Var3)	結果	このデータ型のサイズはシフト演算に影響しません。 INデータ型が符号付きであれば符号付きのデータ型とする

解 説

左ローテーション命令では指定のNビットだけ左へローテーションします。

データ型がWORDの場合

2#0100_1011_0000_0000の2ビット左ローテーションの結果は2#0010_1100_0000_0001

2#0100_1011_0000_0000の4ビット左ローテーションの結果は2#1011_0000_0000_0100

データ型がBYTEの場合

2#0100_1011の2ビット左ローテーションの結果は2#0010_1101

2#0100_1011の4ビット左ローテーションの結果は2#1011_0100

(ILの例)

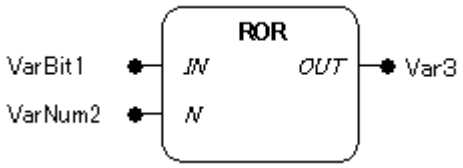
```
LD      in_byte
ROL     n
ST      erg_byte
```

(STの例)

```
PROGRAM rol_st
VAR

     in_byte : BYTE:=16#45;
     in_word : WORD:=16#45;
     erg_byte : BYTE;
     erg_word : WORD;
     n: BYTE :=2;

END_VAR
erg_byte:=ROL(in_byte,n); (* 結果 16#15 *)
erg_word:=ROL(in_word,n); (* 結果 16#0114 *)
```

(FBDの例)

ROR:右ビットローテーション



機能

入力パラメータINに接続した値を、Nに接続したビット数だけ右ローテーションした結果を出力します。

パラメータで使用可能なデータ型

BYTE, WORD, DWORD, LWORD

パラメータ

入力パラメータ	説明	備考
IN (VarBit1)	ローテーションされるデータ	ここに与えるデータ型のサイズが演算の最大ビット数となり、

入力パラメータ	説明	備考
		ここに定数を与えた場合その値が格納できる最小データ型のサイズで演算されます。
N (VarNum2)	ローテーションするビット数	0から入力パラメータINのデータ型の最大ビット数までが有効であり、負の値を与えるとシフト方向が逆になります。

出力パラメータ	説明	備考
OUT (Var3)	結果	このデータ型のサイズはシフト演算に影響しません。 INデータ型が符号付きであれば符号付きのデータ型とする

解 説

右ローテーション命令では指定のNビットだけ右へローテーションします。

データ型がWORDの場合

2#0000_0000_0100_1011の2ビット右ローテーションの結果は2#1100_0000_0001_0010

2#0000_0000_0100_1011の4ビット右ローテーションの結果は2#1011_0000_0000_0100

データ型がBYTEの場合

2#0100_1011の2ビット右ローテーションの結果は2#1101_0010

2#0100_1011の4ビット右ローテーションの結果は2#1011_0100

(ILの例)

```
LD      in_byte
ROR     n
ST      erg_byte
```

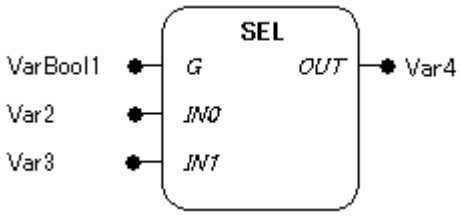
(STの例)

```
PROGRAM ror_st
VAR

     in_byte : BYTE:=16#45;
```

```
        in_word : WORD:=16#45;
        erg_byte : BYTE;
        erg_word : WORD;
        n: BYTE :=2;
    END_VAR
    erg_byte:=ROR(in_byte,n); (* 結果 16#51 *)
    erg_word:=ROR(in_word,n); (* 結果 16#4011 *)
```

(FBDの例)

SEL:データ選択



機能

選択入力 G の値に従い、2つの入力 IN0、IN1 のどちらかを OUT に出力します。選択入力 G が FALSE なら IN0 が出力され TRUE なら IN1 が出力されます。

パラメータで使用可能なデータ型

IN0, IN1, OUT：全ての型
G：BOOL型

パラメータ

入力パラメータ	説明	備考
G (VarBool1)	選択入力	

入力パラメータ	説明	備考
IN0 (Var2)	G = FALSEのとき出力する値	
IN1 (Var3)	G = TRUEのとき出力する値	IN0 と同じデータ型

出力パラメータ	説明	備考
OUT (Var4)	結果	IN0 と同じデータ型

入力 IN0、IN1 と出力 OUT は、同じデータ型でなければなりません。

解 説

G ＝ FALSE のとき OUT ← IN0

G ＝ TRUE のとき OUT ← IN1

(ILの例)

```
LD   TRUE
SEL  3,4     (* IN0 = 3, IN1 =4 *)
ST   Var1    (* 結果は 4 *)
LD   FALSE
SEL  3,4
ST   Var1    (* 結果は 3 *)
```

(STの例)

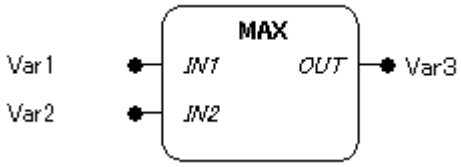
```
Var1:=SEL(TRUE,3,4);  (* 結果は 4 *)
```

(FBDの例)

注 意

N0がTRUEの場合はIN0の計算は処理されません。またN0がFALSEの場合はIN1の計算は処理されません。

MAX:最大値選択



機能

入力パラメータに接続された値のうちの最も大きな値を出力します。

パラメータで使用可能なデータ型

全ての型

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1 と同じデータ型

出力パラメータ	説明	備考
OUT (Var3)	結果	IN1 と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果は 90)

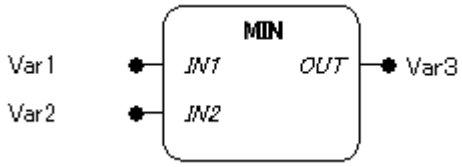
```
LD    90
MAX   30
MAX   40
MAX   77
ST    Var1
```

(STの例)

```
Var1:=MAX(30,40);          (* 結果 40 *)
Var1:=MAX(40,MAX(90,30));  (* 結果 90 *)
```

(FBDの例)

MIN:最小値選択



機能

入力パラメータに接続された値のうち最も小さな値を出力します。

パラメータで使用可能なデータ型

全ての型

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1 と同じデータ型

出力パラメータ	説明	備考
OUT (Var3)	結果	IN1 と同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

(ILの例) (結果は 30)

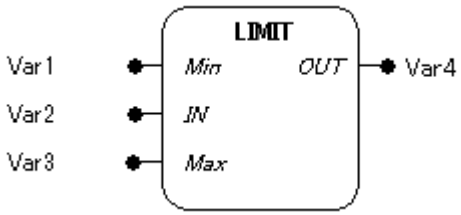
```
LD      90
MIN     30
MIN     40
MIN     77
ST      Var1
```

(STの例)

```
Var1:=MIN(90,30);           (* 結果 30 *)
Var1:=MIN(MIN(90,30),40);   (* 結果 30 *);
```

(FBDの例)

LIMIT:上下限制限



機能

入力パラメータ IN の値を、入力パラメータ Min(下限値)と Max(上限値)により範囲を制限します。

パラメータで使用可能なデータ型

全ての型

パラメータ

入力パラメータ	説明	備考
Min (Var1)	下限値	INと同じデータ型
IN (Var2)	入力値	
Max (Var3)	上限値	INと同じデータ型

出力パラメータ	説明	備考
OUT (Var4)	結果	INと同じデータ型

全てのパラメータは、同じデータ型でなければなりません。

解 説

入力パラメータ IN の値を以下のように制限します。

Min ≦ IN ≦ Max の場合 OUT = IN.

IN ＜ Min の場合 OUT = Min.

IN ＞ Max の場合 OUT = Max.

OUT := LIMIT(Min, IN, Max) とOUT := MIN (MAX (IN, Min), Max) は同じ意味です。

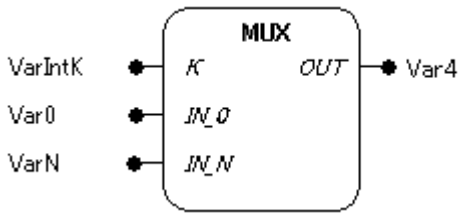
(ILの例) (結果 80)

```
LD      90
LIMIT   30  ,
        80
ST      Var1
```

(STの例)

```
Var1 := LIMIT(30,90,80);  (* 結果 80 *)
```

MUX : マルチプレクサ



機能

入力パラメータ IN_0 から IN_N の値を入力パラメータ K により選択し出力します。

パラメータで使用可能なデータ型

K：BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT, UDINT

IN_0, IN_N, OUT：全ての型

パラメータ

入力パラメータ	説明	備考
K (VarIntK)	選択番号	0 ～ N
IN_0 (Var0)	入力0	
IN_N (VarN)	入力N	IN_0 と同じデータ型

出力パラメータ	説明	備考
OUT (Var4)	結果	IN_0 と同じデータ型

(ILの例) (結果 30)

```
LD    0
MUX   30 ,
      40 ,
      50 ,
      60 ,
      70 ,
      80
ST    Var1
```

(STの例)

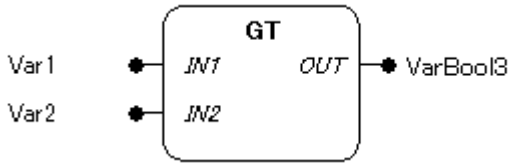
```
Var1:=MUX(0,30,40,50,60,70,80);  (* 結果30 *)
```

注意

入力パラメータKとKにより選択されている入力パラメータ以外の入力パラメータ式は、実行時間を節約するために処

理されません！ しかしシミュレーションモードだけは、すべての式が実行されます。

GT：比較＞(Grater Than)



機能

入力パラメータの比較結果(IN1 > IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2：全ての基本型

OUT：BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBool3)	結果	TRUE: IN1 ＞ IN2 FALSE:IN1 ≦ IN2

(ILの例) (結果は FALSE)

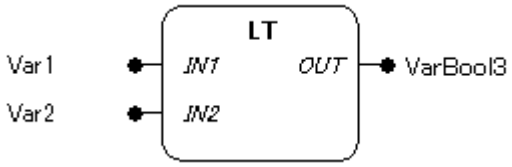
```
LD      20
GT      30
ST      Var1
```

(STの例)

```
Var1 := 20 > 30;
```

(FBDの例)

LT:比較＜(Less Than)



機能

入力パラメータの比較結果(IN1 < IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2: 全ての基本型

OUT: BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (Var3Bool)	結果	TRUE: IN1 ＜ IN2 FALSE:IN1 ≧ IN2

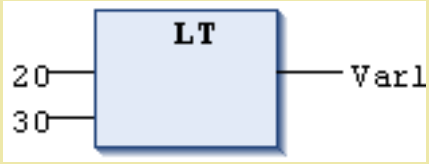
(ILの例)(結果は TRUE)

```
LD    20
LT    30
ST    Var1
```

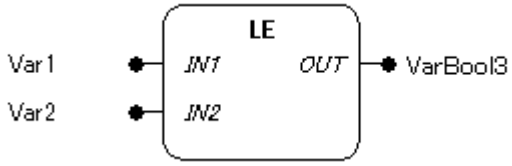
(STの例)

```
Var1 := 20 < 30;
```

(FBDの例)



LE: 比較≦(Less or Equal)



機能

入力パラメータの比較結果(IN1 ≦ IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2: 全ての基本型

OUT: BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBool3)	結果	TRUE: IN1 ≦ IN2 FALSE:IN1 ＞ IN2

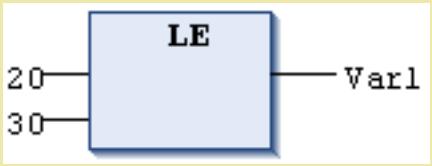
(ILの例) (結果は TRUE)

```
LD    20
LE    30
ST    Var1
```

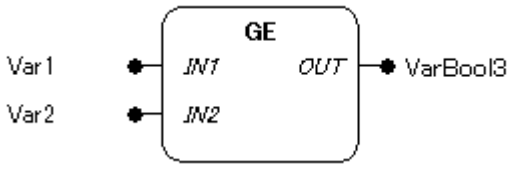
(STの例)

```
Var1 := 20 <= 30;
```

(FBDの例)



GE:比較≧(Grater or Equal)



機能

入力パラメータの比較結果(IN1 ≧ IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2: 全ての基本型

OUT: BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBool3)	結果	TRUE: IN1 ≧ IN2 FALSE:IN1 ＜ IN2

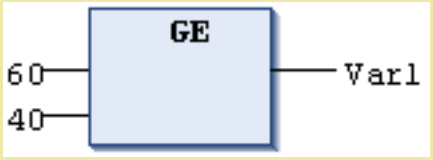
(ILの例) (結果は TRUE)

```
LD    60
GE    40
ST    Var1
```

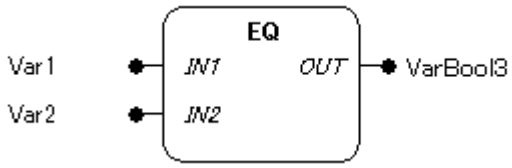
(STの例)

```
Var1 := 60 >= 40;
```

(FBDの例)



EQ: 比較＝(Equal)



機能

入力パラメータの比較結果(IN1 = IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2: 全ての基本型

OUT: BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力1	
IN2 (Var2)	入力2	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBool3)	結果	TRUE: IN1 = IN2 FALSE:IN1 ≠ IN2

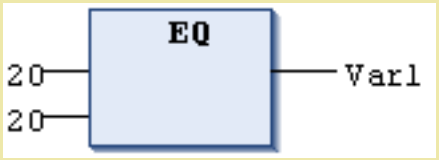
(ILの例) (結果は TRUE)

```
LD    40
EQ    40
ST    Var1
```

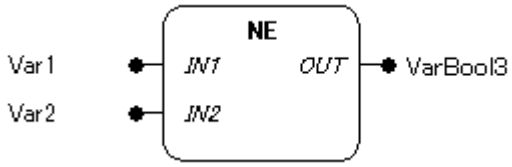
(STの例)

```
Var1 := 40 = 40;
```

(FBDの例)



NE：比較≠(Not Equal)



機能

入力パラメータの比較結果(IN1≠IN2)が成立ならTRUE、それ以外はFALSEを出力します。

パラメータで使用可能なデータ型

IN1, IN2：全ての基本型

OUT：BOOL

パラメータ

入力パラメータ	説明	備考
IN1 (Var1)	入力１	
IN2 (Var2)	入力２	IN1と同じデータ型

出力パラメータ	説明	備考
OUT (VarBool3)	結果	TRUE: IN1 ≠ IN2 FALSE:IN1 ＝ IN2

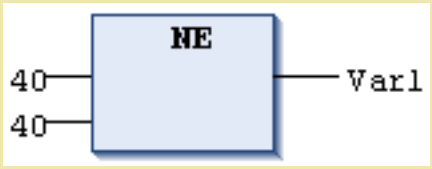
(ILの例) (結果は FALSE)

```
LD      40
NE      40
ST      Var1
```

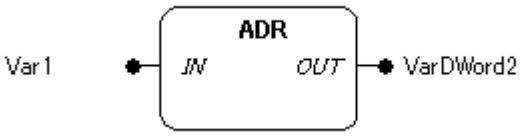
(STの例)

```
Var1 := 40 <> 40;
```

(FBDの例)



ADR:アドレス取得



機能

入力パラメータに接続された変数、プログラム、ファンクション、ファンクションブロック名やメソッド名を指定でき、それぞれのオブジェクトが配置されたメモリアドレスをDWORD型で返します。

パラメータで使用可能なデータ型

IN: 変数、プログラム、ファンクション、ファンクションブロック名やメソッド名

OUT: DWORD

パラメータ

入力パラメータ	説明	備考
IN (Var1)	変数、インスタンス	

出力パラメータ	説明	備考
OUT (VarDWord2)	結果	POINTER

(STの例)

```
VAR

     pt:POINTER TO INT;   (* ポインタptの宣言 *)
```

```
        var_int1:INT := 5;  (* 変数 var_int1, var_int2 の宣言 *)
        var_int2:INT;
    END_VAR
    pt := ADR(var_int1); (* var_int1のアドレスがポインタptに代入されます *)
    var_int2:= pt^;      (*ポインタptの参照によりvar_int1の値5がvar_int2に代入されます *)
```

注 意

- オンライン変更の場合には、変数がメモリ内の別の場所に移動される可能性があります。

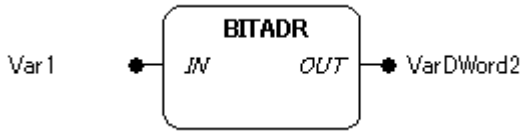
  (コピーが必要な場合、オンライン変更時に表示があります)

  ポインタ変数は、そのような後に無効なメモリを指すことがあります。

  この問題を回避するには、サイクルを超えて保持しないよう各サイクルでポインタの新しい値を取得します。

- 関数やメソッドのPOINTER TO変数は、呼び元の関数に返したり、グローバル変数に渡すべきではありません。

BITADR：ビットオフセット取得



機能

入力パラメータに接続された変数のメモリ領域(セグメント)内に配置された位置のビットオフセットをDWORD型で返します。オフセット値はターゲット設定のバイトアドレス指定がされているかどうかに依存します。返される値のDWORD上位4ビットはメモリ領域を示しています。

パラメータで使用可能なデータ型

IN: BOOL型変数

OUT: DWORD

パラメータ

入力パラメータ	説明	備考
IN (Var1)	BOOL型変数	

出力パラメータ	説明	備考
OUT (VarDWord2)	結果	ビットオフセット

補足

DWORD上位4ビットで返されるメモリ領域:

Memory:16#40000000

Input:16#80000000

Output:16#C0000000

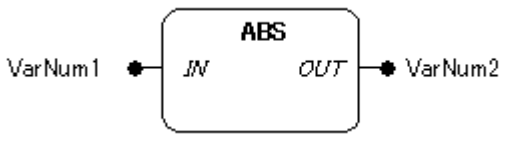
(STの例)

```
VAR
     var1 AT %IX2.3:BOOL;
     bitoffset: DWORD;
END_VAR

bitoffset:=BITADR(var1); (* 結果 BYTE Addresssing=TRUE: 16#80000013 *)
                         (* BYTE Addressing=FALSE: 16#80000023 *)
```

注意

オンライン変更後の値は変更があるかもしれません。アドレスのポインタとして使用している場合は注意が必要となります。

ABS:絶対値



機能

入力パラメータに接続された値 IN の絶対値(｜IN｜ = OUT)を計算します。

パラメータで使用可能なデータ型

IN, OUT: 数値型

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	

出力パラメータ	説明	備考
OUT (VarNum2)	結果	INと同じデータ型

(ILの例) (結果 2)

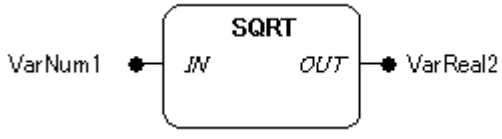
```
LD    -2
ABS
ST    i
```

(STの例)

```
i:=ABS(-2);
```

(FBDの例)

SQRT: 平方根(Square Root)



機能

入力パラメータに接続された値 IN の平方根を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

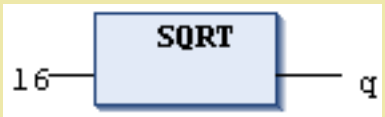
(ILの例) (結果q の値は 4)

```
LD      16
SQRT
ST      q
```

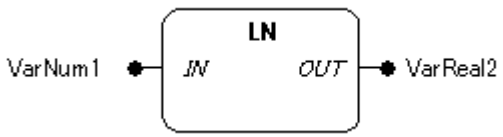
(STの例)

```
q:=SQRT(16);
```

(FBDの例)



LN: 自然対数(Natural Logarithm)



機能

入力パラメータに接続された値 IN の自然対数(底＝e)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

(ILの例) (結果 q は 3.80666)

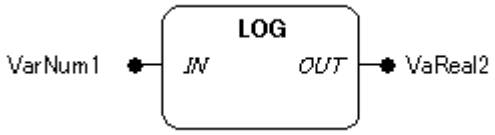
```
LD    45
LN
ST    q
```

(STの例)

```
q:=LN(45);
```

(FBDの例)

LOG: 常用対数(Logerithm)



機能

入力パラメータに接続された値 IN の常用対数(底＝10)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

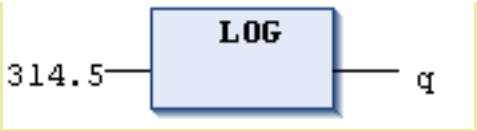
(ILの例) (結果 q は 2.49762)

```
LD      314.5
LOG
ST      q
```

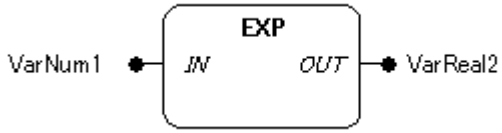
(STの例)

```
q:=LOG(314.5);
```

(FBDの例)



EXP：e の指数累乗(Exponential)



機能

入力パラメータに接続された値 IN の自然指数関数を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	ベースeの指数

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

(ILの例) (結果 q は 7.389056099)

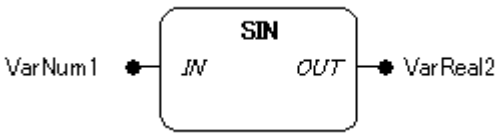
```
LD    2
EXP
ST    q
```

(STの例)

```
q:=EXP(2);
```

(FBDの例)

SIN : サイン(Sine)



機能

入力パラメータに接続された値 IN (ラジアン単位)のサイン(正弦)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	ラジアン

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

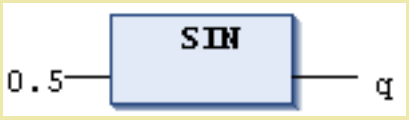
(ILの例) (結果 q は 0.479426)

```
LD    0.5
SIN
ST    q
```

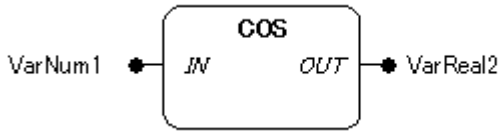
(STの例)

```
q:=SIN(0.5);
```

(FBDの例)



COS:コサイン(Cosine)



機能

入力パラメータに接続された値 IN (ラジアン単位)のコサイン(余弦)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	ラジアン

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

(ILの例) (結果 q は 0.877583)

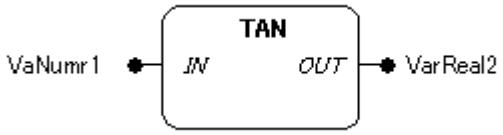
```
LD      0.5
COS
ST      q
```

(STの例)

```
q:=COS(0.5);
```

(FBDの例)

TAN:タンジェント(Tangent)



機能

入力パラメータに接続された値 IN (ラジアン単位)のタンジェント(接線)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	ラジアン

出力パラメータ	説明	備考
OUT (VarReal2)	結果	

(ILの例) (結果 q は 0.546302)

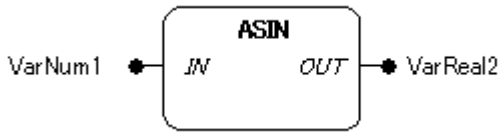
```
LD      0.5
TAN
ST      q
```

(STの例)

```
q:=TAN(0.5);
```

(FBDの例)

ASIN:アークサイン(Arc Sine)



機能

入力パラメータに接続された値 IN のアークサインの主値(ラジアン単位)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	-1.0 ～ 1.0

出力パラメータ	説明	備考
OUT (VarReal2)	結果	ラジアン

(ILの例) (結果 q は 0.523599)

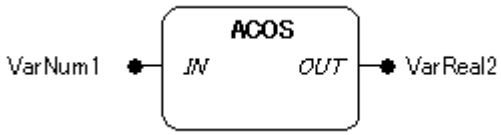
```
LD    0.5
ASIN
ST    q
```

(STの例)

```
q:=ASIN(0.5);
```

(FBDの例)

ACOS:アークコサイン(Arc Cosine)



機能

入力パラメータに接続された値 IN のアークコサインの主値(ラジアン単位)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	-1.0 ～ 1.0

出力パラメータ	説明	備考
OUT (VarReal2)	結果	ラジアン

(ILの例) (結果 q は 1.0472)

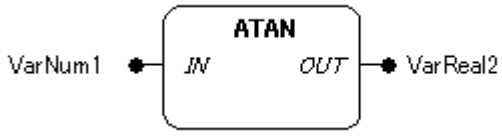
```
LD      0.5
ACOS
ST      q
```

(STの例)

```
q:=ACOS(0.5);
```

(FBDの例)

ATAN：アークタンジェント(Arc Tangent)



機能

入力パラメータに接続された値 IN のアークタンジェントの主値(ラジアン単位)を計算します。

パラメータで使用可能なデータ型

IN: 数値型

OUT: REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN (VarNum1)	入力	-1.0 ～ 1.0

出力パラメータ	説明	備考
OUT (VarReal2)	結果	ラジアン

(ILの例) (結果 q は 0.463648)

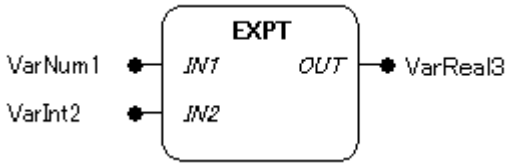
```
LD      0.5
ATAN
ST      q
```

(STの例)

```
q:=ATAN(0.5);
```

(FBDの例)

EXPT: べき乗算



機能

入力パラメータ IN1 に接続された値を IN2 に接続された整数値によるべき乗算結果を出力します。

パラメータで使用可能なデータ型

IN1, IN2: 数値型

OUT：REAL, LREAL

パラメータ

入力パラメータ	説明	備考
IN1 (VarNum1)	ベース	
IN2 (VarInt2)	指数	

出力パラメータ	説明	備考
OUT (VarReal3)	結果	

(ILの例) (結果 49)

```
LD      7
EXPT    2
ST      Varl
```

(STの例)

```
Var1 := EXPT(7,2);
```

(FBDの例)

7.3.呼び出し演算子

ファンクションブロックの呼び出しで使用されます。

分類	命令	機能
呼び出し演算	CAL	IL言語ファンクションブロック呼び出し

CAL:呼び出し

機能

IL言語で指定されたファンクション・ブロックのインスタンスを呼び出します。

(ILの例)

```
CAL     SRinst  (
      SET1:= VarBool1  ,
     RESET:= VarBool2  )
LD      SRinst.Q1
ST      VarBool3
```

7.4.型変換演算子

大きな型から小さな型への変換は、明示的な型変換が必要となります。例えばDINTからINTや、WORDからBYTEへの変換です。

サイズや負号の異なるデータ型へ変換する場合は、情報を失う恐れがありますので注意が必要です。

ある基本型から別の基本型へ変換するには、次の構文を使用します。

<変換前の型>_TO_<変換後の型>

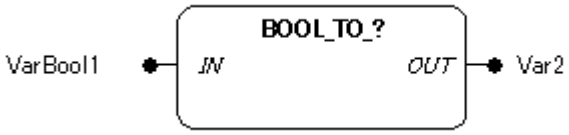
例：DINT_TO_INT

また、～TO_STRING変換の結果は、左詰めの文字列が生成されます。この際に変換後の文字列変数定義が変換前の変数定義よりも短い場合は結果文字列の右側(終端側)が切られます。

分類	命令	機能
BOOL型変換演算子	BOOL_TO_?	BOOL型の変換
	?_TO_BOOL	BOOL型への変換
数値型変換演算子	SINT_TO_? INT_TO_? DINT_TO_? LINT_TO_?	整数型の変換
	?_TO_SINT ?_TO_INT ?_TO_DINT ?_TO_LINT	整数型への変換
	BYTE_TO_? WORD_TO_? DWORD_TO_? LWORD_TO_?	ビット列型の変換
	?_TO_BYTE ?_TO_WORD ?_TO_DWORD ?_TO_LWORD	ビット列型への変換

分類	命令	機能
	USINT_TO_? UINT_TO_? UDINT_TO_? ULINT_TO_?	符号なし数値型の変換
	?_TO_USINT ?_TO_UINT ?_TO_UDINT ?_TO_ULINT	符号なし数値型への変換
実数型変換演算子	REAL_TO_?	実数の変換
	?_TO_REAL	実数型への変換
BCD変換関数	BCD_TO_BYTE BCD_TO_WORD BCD_TO_DWORD BCD_TO_INT	BCD値の変換
	BYTE_TO_BCD WORD_TO_BCD DWORD_TO_BCD INT_TO_BCD	BCD値への変換
時間型変換演算子	TIME_TO TIME_OF_DAY	時間型の変換
日付型変換演算子	DATE_TO DT_TO	日付型の変換
実数の整数変換演算子	TRUNC TRUNC_INT	小数点以下の値の切り捨て

BOOL_TO_? 変換



機能

BOOL データ型の入力値を他のデータ型へ変換します。

構文は

BOOL_TO_<データ型>

数値型への変換は、入力パラメータが TRUE の時の結果は 1、FALSE の時の結果は 0 が返ります。
文字列型への変換は、入力パラメータが TRUE の時の結果は 'TRUE'、FALSE の時の結果は 'FALSE' が返ります。

パラメータで使用可能なデータ型

IN: BOOL
OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarBool1)	入力	

出力変数	解説	備考
OUT (Var2)	結果	指定したデータ型

解説

入力値が FALSE のとき、出力値は 0 に、入力値が TRUE のとき、出力値は 1 に変換されます。

(ILの例)

```
LD          TRUE
BOOL_TO_INT
ST          i
```
(*結果は 1 *)

```
LD          TRUE
BOOL_TO_STRI...
ST          str
```
(* 結果は 'TRUE' *)

```
LD          TRUE
BOOL_TO_TIME
ST          t
```
(* 結果は T#1ms *)

```
LD          TRUE
BOOL_TO_TOD
ST          tof
```
(* 結果は TOD#00:00:00.001 *)

```
LD          FALSE
BOOL_TO_DATE
ST          dandt
```
(* 結果は D#1970-01-01 *)

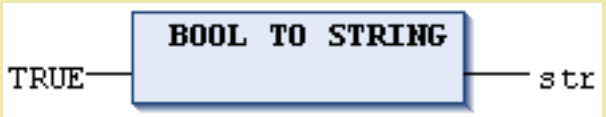
```
LD          TRUE
BOOL_TO_DT
ST          dandt
```
(* 結果は DT#1970-01-01-00:00:01 *)

(STの例)

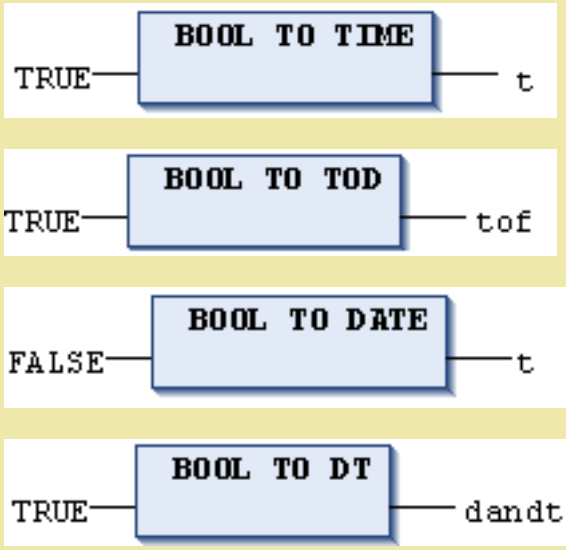
```
i:=BOOL_TO_INT(TRUE);        (* 結果は 1 *)
str:=BOOL_TO_STRING(TRUE);   (* 結果は "TRUE" *)
t:=BOOL_TO_TIME(TRUE);       (* 結果は T#1ms *)
tof:=BOOL_TO_TOD(TRUE);      (* 結果は TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);    (* 結果は D#1970 *)
dandt:=BOOL_TO_DT(TRUE);     (* 結果は DT#1970-01-01-00:00:01 *)
```

(FBDの例)

(* 結果は 1 *)



(* 結果は 'TRUE' *)

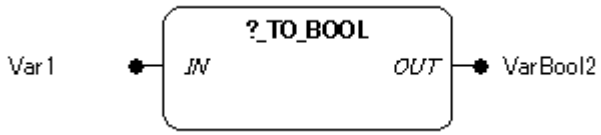



```
(* 結果は T#1ms *)

(* 結果は TOD#00:00:00.001 *)

(* 結果は D#1970-01-01 *)

(* 結果は DT#1970-01-01-00:00:01
*)
```

補 足

STRING変換は文字列変換ファンクションの項を参照して下さい。

?_TO_BOOL 変換



機能

他のデータ型の入力値を BOOL データ型へ変換します。

構文は

<データ型>_TO_BOOL

パラメータで使用可能なデータ型

IN: 数値型, STRING

OUT: BOOL

パラメータ

入力変数	解説	備考
IN (Var1)	入力	指定したデータ型

出力変数	解説	備考
OUT (VarBool2)	結果	

解 説

入力が0以外の場合、変換結果はTRUE、入力が0と等しい場合はFALSE になります。

(ILの例)

```
LD              213
BYTE_TO_BOOL
ST              b
```
(* 結果はTRUE *)

```
LD              0
INT_TO_BOOL
ST              b
```
(*結果は FALSE *)

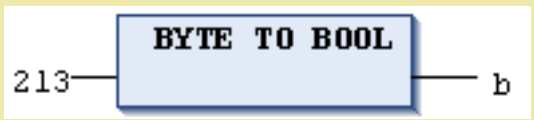
```
LD              T#5ms
TIME_TO_BOOL
ST              b
```
(*結果は TRUE *)

```
LD              'TRUE'
STRING_TO_BOOL
ST              b
```
(*結果は TRUE *)

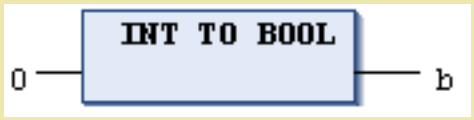
(STの例)

```
b := BYTE_TO_BOOL(2#11010101);  (*結果は TRUE *)
b := INT_TO_BOOL(0);            (*結果は FALSE *)
b := TIME_TO_BOOL(T#5ms);       (*結果は TRUE *)
b := STRING_TO_BOOL('TRUE');    (*結果は TRUE *)
```

(FBDの例)



(*結果は TRUE *)



(*結果は FALSE *)

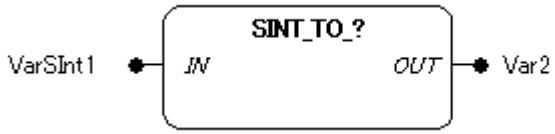
(*結果は TRUE *)

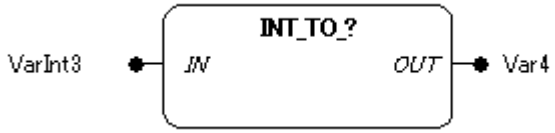
(*結果は TRUE *)

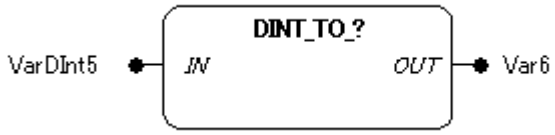
補 足

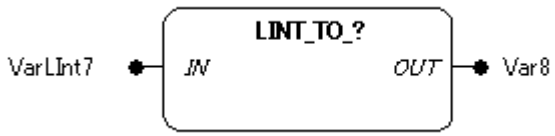
STRING変換は、文字列変換ファンクションの項を参照して下さい。

SINT_TO_? / INT_TO_? / DINT_TO_? / LINT_TO_? 変換









機能

整数型のデータ型入力値を他のデータ型へ変換します。

構文は

SINT_TO_<データ型>
INT_TO_<データ型>
DINT_TO_<データ型>
LINT_TO_<データ型>

パラメータで使用可能なデータ型

IN: SINT, INT, DINT, LINT
OUT: 数値型

パラメータ

入力変数	解説	備考
IN (VarSInt1, VarInt3, VarDInt5, VarLInt7)	入力	指定したデータ型

出力変数	解説	備考
OUT (Var2, Var4, Var6, Var8)	結果	

解 説

出力データ型が入力データ型より大きいときは、符号適合拡張が行われます。(例:SINT#-1 → DINT#-1)
小さいデータ型へ変換を行う場合、下位データビットに合わせるため、情報を失う恐れがあります。
符号無し型(USINT、UINT、UDINT)へ変換する場合、出力は常に正の値となります。

(STの例)

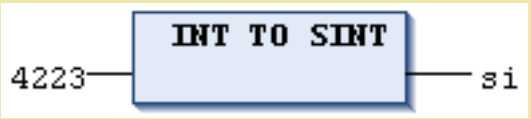
```
si := INT_TO_SINT(4223); (* 結果127 *)
```

ここで整数値 4223 (16進表記で16#107f) をSINT 変数に保存したならば127 (16進表記で16#7f)が現れます。

(ILの例)

```
LD      4223
INT_TO_SINT
ST      si
```

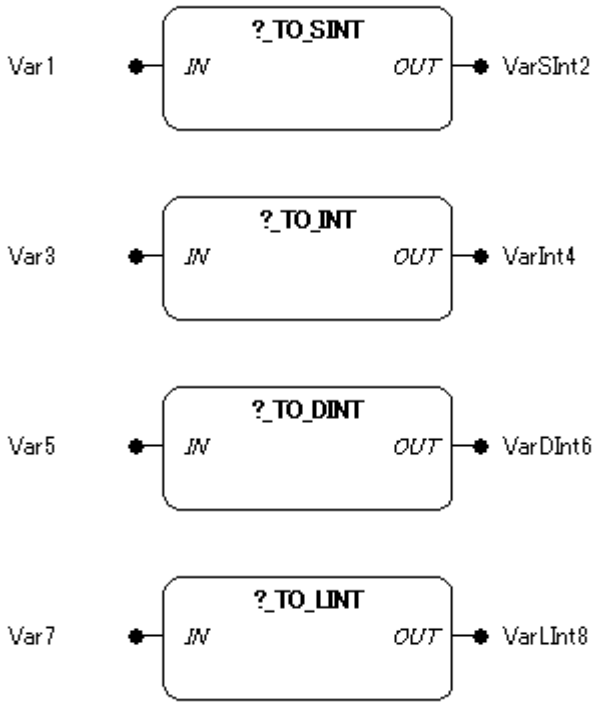
(FBDの例)



補 足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

?_TO_SINT / ?_TO_INT / ?_TO_DINT / ?_TO_LINT 変換



機能

他のデータ型入力値を整数型のデータ型へ変換します。

構文は

<データ型>_TO_SINT

<データ型>_TO_INT

<データ型>_TO_DINT

<データ型>_TO_LINT

パラメータで使用可能なデータ型

IN: 数値型, STRING

OUT: SINT, INT, DINT, LINT

パラメータ

入力変数	解説	備考
IN (Var1, Var3, Var5, Var7)	入力	指定したデータ型

出力変数	解説	備考
OUT (VarSInt2, VarInt4, VarDInt6, VarLInt8)	結果	

解 説

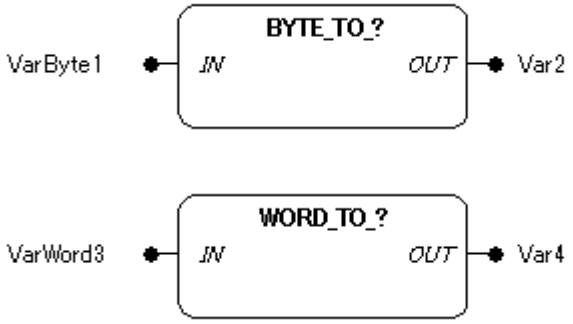
出力データ型が入力データ型より大きいときは、符号適合拡張が行われます。(例 : 16#FF → -1)

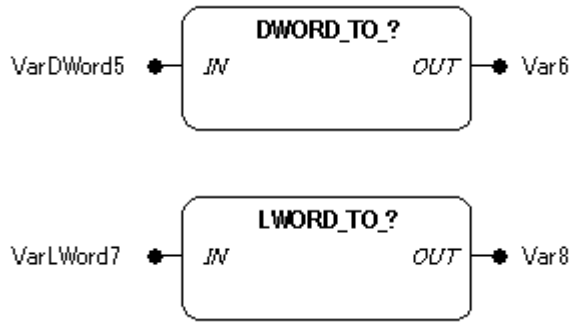
小さいデータ型へ変換を行う場合は、下位データビットに合わせるために情報を失う恐れがあります。

補 足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

BYTE_TO_? / WORD_TO_? / DWORD_TO_? / LWORD_TO_? 変換





機能

ビットストリーム型のデータ型入力値を他のデータ型へ変換します。

構文は

BYTE_TO_<データ型>
WORD_TO_<データ型>
DWORD_TO_<データ型>
LWORD_TO_<データ型>

パラメータで使用可能なデータ型

IN: BYTE, WORD, DWORD, LWORD
OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarByte1, VarWord3, VarDWord5, VarLWord7)	入力	

出力変数	解説	備考
OUT (Var2, Var4, Var6, Var8)	結果	指定したデータ型

解説

出力データ型が入力データ型より大きいときは、符号適合拡張が行われます。(例 : 16#FF → -1)
小さいデータ型へ変換を行う場合、下位データビットに合わせるため、情報を失う恐れがあります。
出力データ型のMSB (最上位ビット) がセットされている場合は、負の出力値へ変換されます。

(IL の例)

```
LD          16#80   (* 16#80 をアキュムレータにロードします *)
BYTE_TO_INT         (* BYTE を INT に変換します *)
ST          Var2    (* -128 を Var2 に格納します *)


LD           16#7FFF  (* 16#7FFF をアキュムレータにロードします *)
WORD_TO_UINT          (* WORD を UINT に変換します *)
ST           Var4     (* 32767 を Var4 に格納します *)


LD            16#FFFFFF7F  (* 16#FFFFFF7F をアキュムレータにロードします *)
DWORD_TO_BYTE              (* DWORD を BYTE に変換します *)
ST            Var6         (* 16#7F を Var6 に格納します *)
```

(STの例)

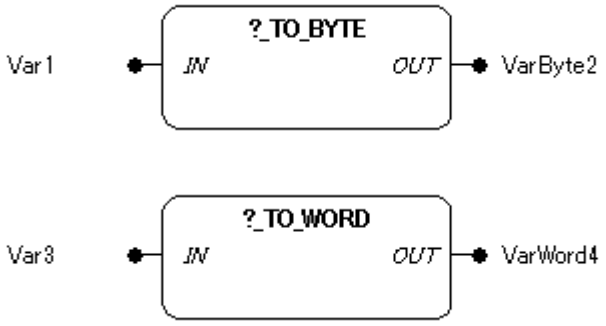
```
iVar := BYTE_TO_INT(16#80);  (* 結果 -128 *)
```

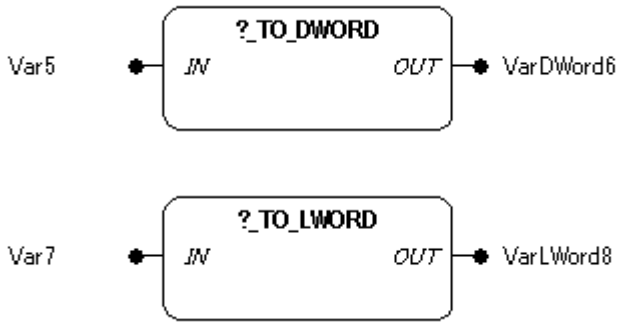
ここで整数値 128 (16進表記で16#80) をINT 変数に保存したならば-128 (16進表記で16#ff80)が現れます。

補 足

BOOL、REAL、BCD、STRING変換は各ファンクションの項を参照して下さい。

?_TO_BYTE / ?_TO_WORD / ?_TO_DWORD / ?_TO_LWORD 変換





機能

他のデータ型入力値をビットストリーム型のデータ型へ変換します。

構文は

<データ型>_TO_BYTE
<データ型>_TO_WORD
<データ型>_TO_DWORD
<データ型>_TO_LWORD

パラメータで使用可能なデータ型

IN: 数値型, STRING
OUT: BYTE, WORD, DWORD, LWORD

パラメータ

入力変数	解説	備考
IN (Var1, Var3, Var5, Var7)	入力	指定したデータ型

出力変数	解説	備考
OUT (VarByte2, VarWord4, VarDWord6, VarLWord8)	結果	

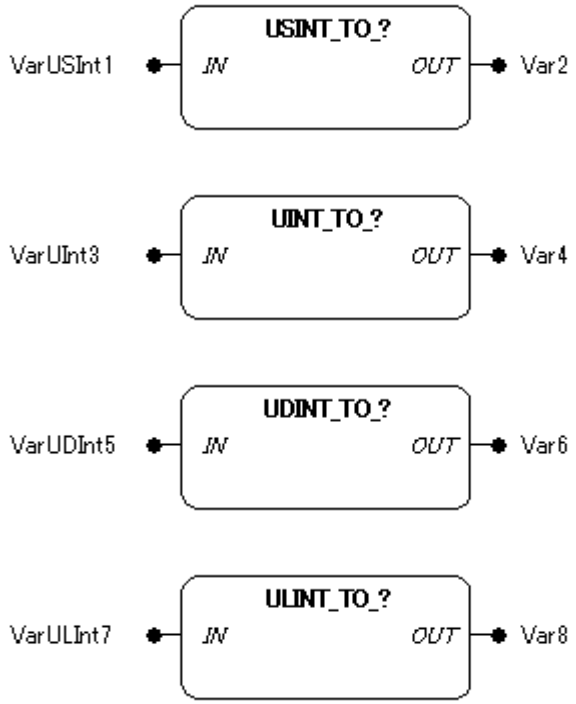
解説

小さいデータ型へ変換を行う場合、下位データビットに合わせるため、情報を失う恐れがあります。

補足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

USINT_TO_? / UINT_TO_? / UDINT_TO_? / ULINT_TO_? 変換



機能

符号無し整数型のデータ型入力値を他のデータ型へ変換します。

構文は

USINT_TO_<データ型>
UINT_TO_<データ型>
UDINT_TO_<データ型>
ULINT_TO_<データ型>

パラメータで使用可能なデータ型

IN: USINT, UINT, UDINT, ULINT
OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarUSInt1, VarUInt3, VarUDInt5, VarULInt7)	入力	

出力変数	解説	備考
OUT (Var2, Var4, Var6, Var8)	結果	指定したデータ型

解 説

小さいデータ型へ変換を行う場合は、下位データビットに合わせるために情報を失う恐れがあります。
出力データ型のMSB (最上位ビット) がセットされている場合は、負の出力値へ変換されます。

(STの例)

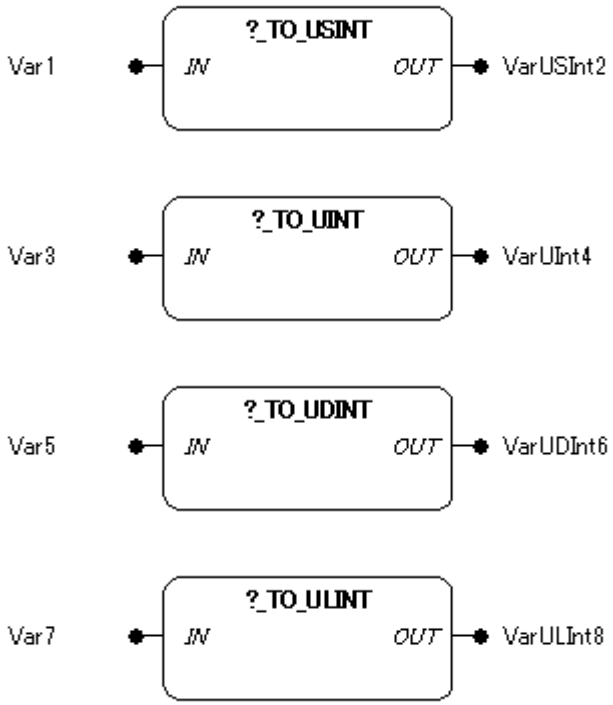
```
siVar := USINT_TO_SINT(254);  (* 結果 -2 *)
```

ここで整数値 254 (16進表記で16#fe) をSINT 変数に保存したならば-2 (16進表記で16#fe)が現れます。

補 足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

?_TO_USINT / ?_TO_UINT / ?_TO_UDINT / ?_TO_ULINT 変換



機能

他のデータ型入力値を符号無し整数型のデータ型へ変換します。

構文は

<データ型>_TO_USINT
<データ型>_TO_UINT
<データ型>_TO_UDINT
<データ型>_TO_ULINT

パラメータで使用可能なデータ型

IN: 数値型, STRING
OUT: USINT, UINT, UDINT, ULINT

パラメータ

入力変数	解説	備考
IN (Var1, Var3, Var5, Var7)	入力	指定したデータ型

出力変数	解説	備考
OUT (VarUSInt2, VarUInt4, VarUDInt6, VarULInt8)	結果	

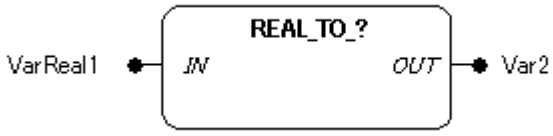
解 説

出力は、常に正の値となります。
小さいデータ型へ変換を行う場合は、下位データビットに合わせるために情報を失う恐れがあります。

補 足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

REAL_TO_? 変換



機能

REAL データ型の入力値を 他のデータ型の出力値に変換します。

構文は

REAL_TO_<データ型>

パラメータで使用可能なデータ型

IN: REAL

OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarReal1)	入力	

出力変数	解説	備考
OUT (Var2)	結果	指定したデータ型

解 説

REAL 入力値は小数点第1位で四捨五入された後、他のデータ型に変換されます。

小さいデータ型へ変換を行う場合、下位データビットに合わせるため、情報を失う恐れがあります。

(STの例)

```
i := REAL_TO_INT(1.5);   (* 結果は 2 *)
j := REAL_TO_INT(1.4);   (* 結果は 1 *)
i := REAL_TO_INT(-1.5);  (* 結果は -2 *)
j := REAL_TO_INT(-1.4);  (* 結果は -1 *)
```

(ILの例)

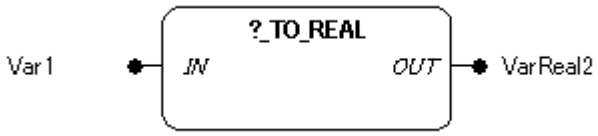
```
LD          2.7
REAL_TO_INT
ST          i
```

(FBDの例)

補 足

BOOL、REAL、BCD、STRING変換は各ファンクションの項を参照して下さい。

?_TO_REAL 変換



機能

他のデータ型の入力値を REAL データ型の出力値に変換します。

構文は

<データ型>_TO_REAL

パラメータ

入力変数	解説	備考
IN (Var1)	入力	指定したデータ型

出力変数	解説	備考
OUT (VarReal2)	結果	

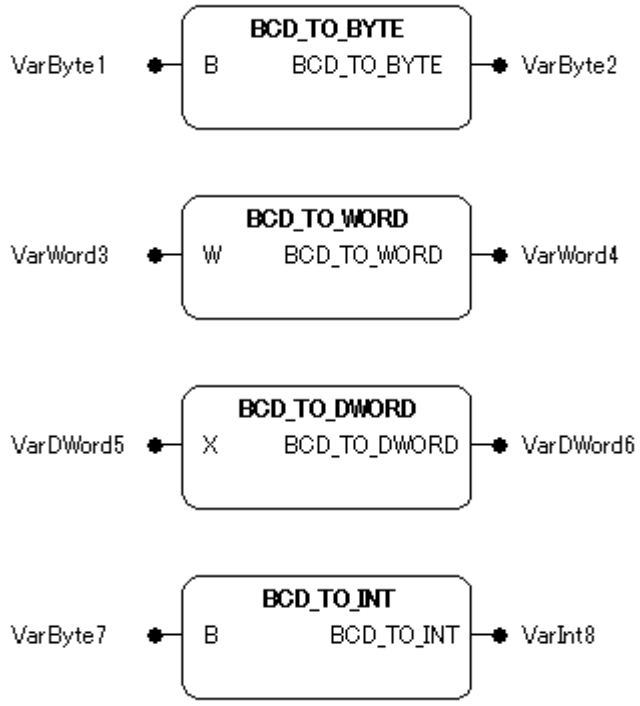
(STの例)

```
VarReal2 := INT_TO_REAL(-32768);  (* 結果は -3.2768000E+04 *)
```

補 足

BOOL、REAL、BCD、STRING変換は、各ファンクションの項を参照して下さい。

BCD_TO_BYTE / BCD_TO_WORD / BCD_TO_DWORD / BCD_TO_INT 変換 【FUN】



機能

BCD データ型の入力値 (バイナリコードの 10 進数) を他のデータ型の戻り値に変換します。

構文は

BCD_TO_BYTE

BCD_TO_WORD

BCD_TO_DWORD

BCD_TO_INT

パラメータで使用可能なデータ型

B: BYTE

W: WORD

X: DWORD

パラメータ

入力変数	解説	備考
B (VarByte1, VarByte7)	入力	16#00～16#99
W (VarWord3)	入力	16#0000～16#9999
X (VarDWord5)	入力	16#00000000～16#99999999

戻り値	解説	備考
BCD_TO_BYTE	結果	BYTEデータ型
BCD_TO_WORD	結果	WORDデータ型
BCD_TO_DWORD	結果	DWORDデータ型
BCD_TO_INT	結果	BCD書式でなければ -1 を返します

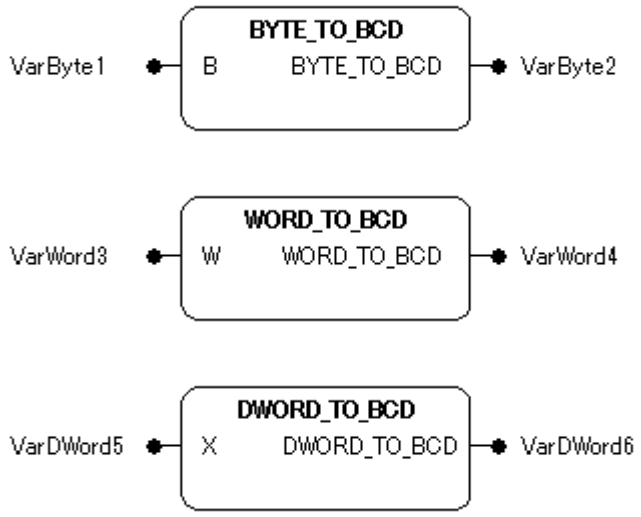
(ST の例)

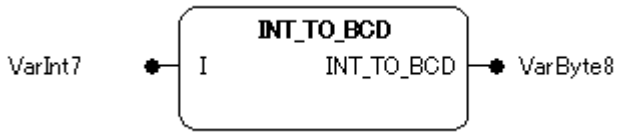
```
i:=BCD_TO_INT(73);   (* 結果は 49 *)
k:=BCD_TO_INT(151);  (* 結果は 97 *)
l:=BCD_TO_INT(15);   (* 値がBCDではないので、結果は -1 *)
```

補 足

- 不正な入力値 (使用可能な値は 0 から 9 まで) のときは、出力値は -1 に変換されます。たとえば、入力値が 16#0A0B のとき、出力値は -1 になります。
- 出力データ型の有効範囲を超えている値の場合は、オーバーフローになることを考慮する必要があります。

BYTE_TO_BCD / WORD_TO_BCD / DWORD_TO_BCD / INT_TO_BCD 変換 【FUN】





機能

他のデータ型の入力値を BCD データ型 (バイナリコードの 10 進数) の出力値に変換します。

構文は

BYTE_TO_BCD
WORD_TO_BCD
DWORD_TO_BCD
INT_TO_BCD

パラメータで使用可能なデータ型

B: BYTE
W: WORD
X: DWORD
I: INT

パラメータ

入力変数	解説	備考
B (VarByte1)	入力	0 ～ 99
W (VarWord3)	入力	0 ～ 9999
X (VarDWord5)	入力	0 ～ 99999999
I (VarInt7)	入力	0 ～ 99

戻り値	解説	備考
BYTE_TOBCD	結果	
WORD_TOBCD	結果	
DWORD_TOBCD	結果	
INT_TO_BCD	結果	0 ～ 99以外は 255 を返します

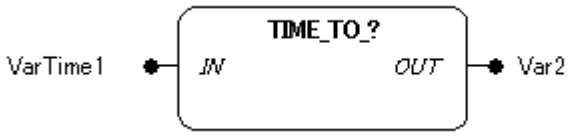
(STの例)

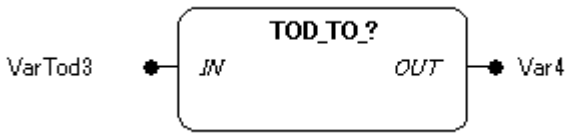
```
i:=INT_TO_BCD(49);   (* 結果は 73 *)
k:=INT_TO_BCD(97);   (* 結果は 151 *)
l:=INT_TO_BCD(100);  (* エラーなので 255 *)
```

補 足

- 負の入力値は、常に16#FFFFFFFF へ変換されます。たとえば、入力値が-128 のとき出力値は16#FFFFFFFF になります。
- 出力データ型 (BCD) の有効範囲を超えた入力値の場合は、出力値は常に16#FFFFFFFF へ変換されます。BCD 値の最大有効範囲は99999999 です。

TIME_TO_? / TIME_OF_DAY_? 変換





機能

TIME あるいはTIME_OF_DAY データ型入力値 IN を他のデータ型の出力値に変換します。

構文は

TIME_TO_<データ型>
TOD_TO_<データ型>

パラメータで使用可能なデータ型

IN: TIME, TOD
OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarTime1)	入力	TIME データ型

入力変数	解説	備考
IN (VarTod3)	入力	TIME_OF_DAY

出力変数	解説	備考
OUT (Var2, Var4)	結果	指定したデータ型

解 説

時間は内部でDWORD型にミリ秒で格納され、TIME_OF_DAY 変数では12:00A.M.を開始とする値が格納されます。STRING データ型への変換は時間コンスタント値が返されます。大きな値をもつ入力変数から小さなデータ型へ変換を行うと情報を失う恐れがあります。

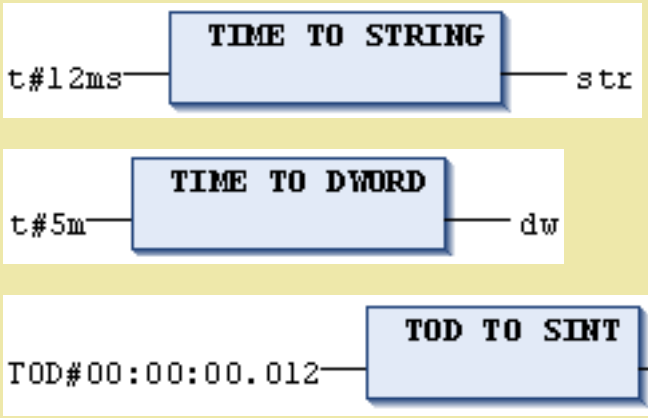
(ILの例)

```
LD          T#12ms
TIME_TO_STTI...
ST          str          (* 結果は 'T#12ms' *)

LD          T#300000ms
TIME_TO_DWORD
ST          dw           (*結果は 300000 *)

LD          TOD#00:00:00.012
TIME_TO_SINT
ST          si           (*結果は 12 *)
```

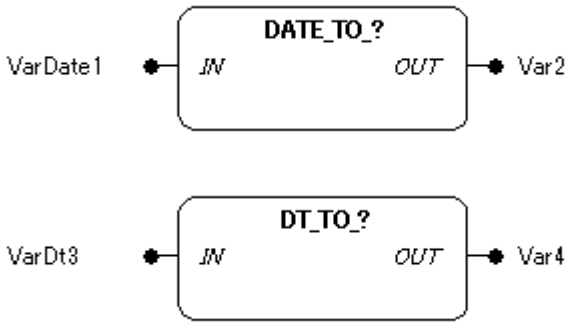
(STの例)

```
str :=TIME_TO_STRING(T#12ms);        (*結果は T#12ms *)
dw:=TIME_TO_DWORD(T#5m);             (*結果は 300000 *)
si:=TOD_TO_SINT(TOD#00:00:00.012);   (*結果は 12 *)
```

(FBDの例)



DATE_TO_? / DATE_AND_TIME_TO_? 変換



機能

DATE あるいは DATE_AND_TIME データ型入力値を 他のデータ型の出力値に変換します。

構文は

DATE_TO_<データ型>

DT_TO_<データ型>

パラメータで使用可能なデータ型

IN: DATE, DATE_AND_TIME

OUT: 数値型, STRING

パラメータ

入力変数	解説	備考
IN (VarDate1)	入力	DATE データ型
IN (VarDt3)	入力	DATE_AND_TIME

出力変数	解説	備考
OUT (Var2, Var4)	結果	指定したデータ型

解 説

時刻は、内部でDWORD型に1970/1/1からの通算秒とする値が格納されます。STRING データ型への変換は時刻コンスタント値が返されます。大きな値をもつ入力変数から小さなデータ型へ変換を行うと情報を失う恐れがあります。

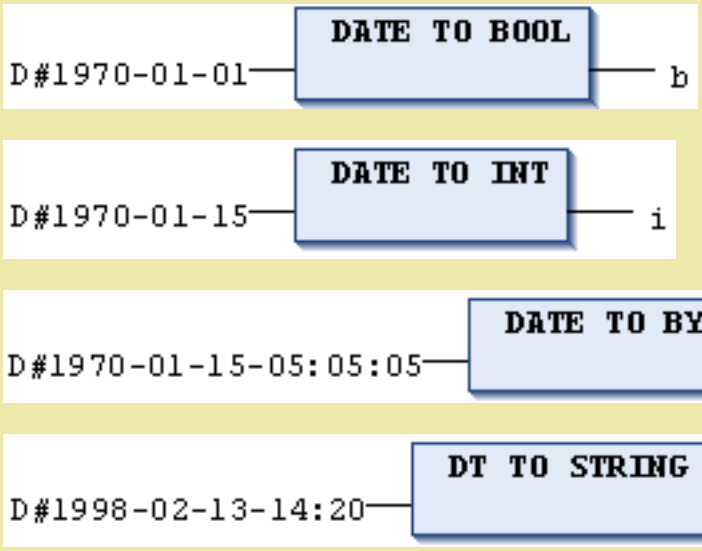
(ILの例)

```
LD          D#1970-01-01
DATE_TO_BOOL
ST          b                (* 結果は FALSE *)

LD          D#1970-01-01
DATE_TO_INT
ST          i                (*結果は 29952 *)

LD          D#1970-01-15-05:05:.
DATE_TO_BYTE
ST          byt              (*結果は 129 *)

LD          D#1998-02-13-14:20
DATE_TO_STRI...
ST          str              ( *結果は 'DT#1998-02-13-14:20' *)
```

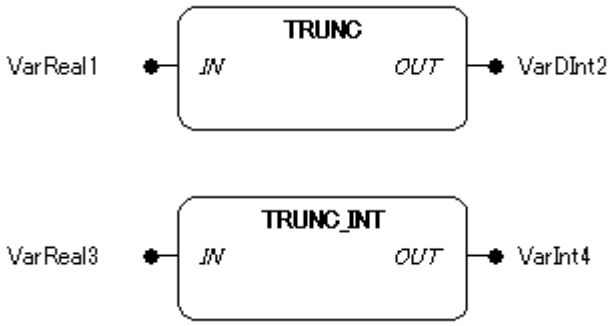
(STの例)

```
b :=DATE_TO_BOOL(D#1970-01-01);          (*結果は FALSE *)
i :=DATE_TO_INT(D#1970-01-15);           (*結果は 29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-
05:05:05);                               (*結果は 129 *)
str:=DT_TO_STRING(DT#1998-02-13-         (*結果は 'DT#1998-02-13-14:20'
14:20);                                  *)
```

(FBDの例)



TRUNC / TRUNC_INT 変換



機能

TRUNCはREAL 値から小数点以下の値は切り捨ててDINT 値へ変換します。

TRUNC_INTはREAL 値から小数点以下の値は切り捨ててINT 値へ変換します。

パラメータ

入力変数	解説	備考
IN (VarReal, VarReal3)	入力	REAL データ型

出力変数	解説	備考
OUT (VarDInt2)	結果	DINT データ型
OUT (VarInt4)	結果	INT データ型

(ILの例)

```
LD      1.9
TRUNC
ST      diVar

LD      1.9
TRUNC_INT
ST      iVar
```

(STの例)

```
diVar:=TRUNC(1.9);   (* 結果は 1 *)
diVar:=TRUNC(-1.4);  (* 結果は -1 *)


iVar:=TRUNC_INT(1.9);   (* 結果は 1 *)
iVar:=TRUNC_INT(-1.4);  (* 結果は -1 *)
```

7.5.文字列操作ファンクション

分類	命令	機能
文字列操作	LEN	文字列の長さ
	LEFT	左端からの文字列抽出
	RIGHT	右端からの文字列抽出
	MID	中間からの文字列抽出
	CONCAT	文字列の連結
	INSERT	文字の挿入
	DELETE	文字列の削除
	REPLACE	文字の置換
	FIND	文字の検索

LEN:文字列長さ【FUN】

機能

文字列の長さを返します。

パラメータで使用可能なデータ型

STR: STRING

LEN: INT

パラメータ

入力変数	解説	備考
STR ('conveyor')	入力	

出力変数	解説	備考
LEN (VarInt2)	結果	

解 説

入力パラメータSTRに接続された文字列の長さを返します。

(ILの例) (結果は "4")

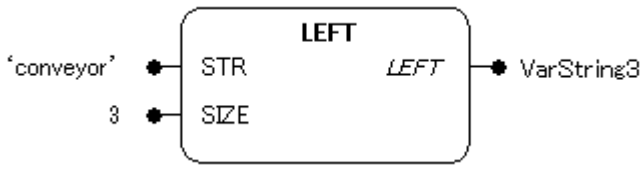
```
LD      'SUSI'
LEN
ST      VarINT1
```

(STの例)

```
VarINT1 := LEN ('SUSI');
```

(FBDの例)

LEFT:左文字列抽出 [FUN]



機能

文字列入力の左から指定長さの文字列を抽出します。

パラメータで使用可能なデータ型

STR: STRING

SIZE: INT

LEFT: STRING

パラメータ

入力パラメータ	説明	備考
STR ('conveyor')	文字列入力	
SIZE (3)	左から抽出する長さ(文字数)	0 以上

出力パラメータ	説明	備考
LEFT (VarString3)	結果 ('con')	

解 説

入力 STR に接続された文字列の左端から入力 SIZE に接続された長さの文字列を抽出し返します。

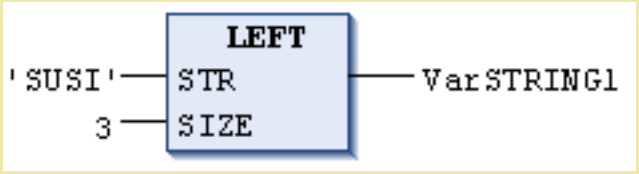
(ILの例) (結果は 'SUS')

```
LD      'SUSI'
LEFT    3
ST      VarSTRING1
```

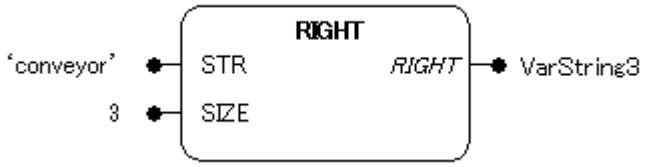
(STの例)

```
VarSTRING1 := LEFT ('SUSI',3);
```

(FBDの例)



RIGHT:右文字列抽出 [FUN]



機能

文字列入力の右から指定長さの文字列を抽出します。

パラメータで使用可能なデータ型

STR: STRING

SIZE: INT

RIGHT: STRING

パラメータ

入力パラメータ	説明	備考
STR ('conveyor')	文字列入力	
SIZE (3)	右から抽出する長さ(文字数)	1 以上

出力パラメータ	説明	備考
RIGHT (VarString3)	結果 ('yor')	

解 説

入力 STR に接続された文字列の右端から入力 SIZE に接続された長さの文字列を抽出し返します。

(ILの例) (結果は 'USI')

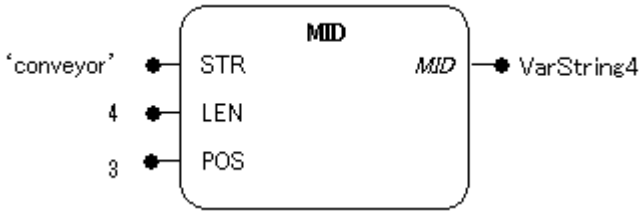
```
LD      'SUSI'
RIGHT   3
ST      VarSTRING1
```

(STの例)

```
VarSTRING1 := RIGHT ('SUSI',3);
```

(FBDの例)

MID:中間文字列抽出 [FUN]



機能

文字列入力の中間位置から、指定長さの文字列を抽出します。

パラメータで使用可能なデータ型

STR: STRING

LEN, POS: INT

MID: STRING

パラメータ

入力パラメータ	説明	備考
STR ('conveyor')	文字列入力	
LEN (4)	抽出する長さ(文字数)	0 以上
POS (3)	STRの先頭を1とした抽出開始位置	

出力パラメータ	説明	備考
MID (VarString4)	結果 ('nvey')	

解 説

入力 STR に接続された文字列の入力 POS で指定される中間位置から、入力 LEN で指定される長さの文字列を抽出し結果として返します。

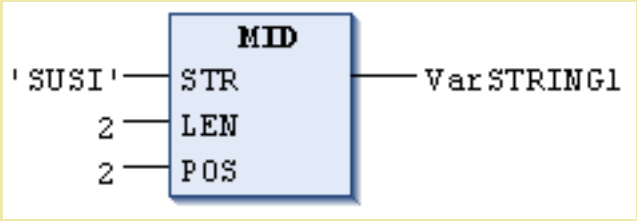
(ILの例) (結果は 'US')

```
LD      'SUSI'
MID     2,
        2
ST      VarSTRING1
```

(STの例)

```
VarSTRING1 := MID ('SUSI',2,2);
```

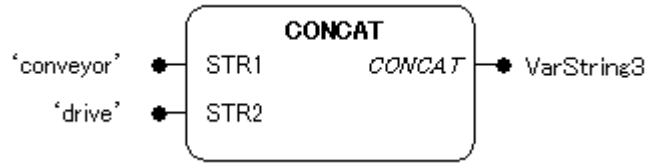
(FBDの例)



補 足

- POS は 0 を指定することはできません。文字列の最初の位置は 1 です。

CONCAT:文字列連結 [FUN]



機能

文字列の連結をします。

パラメータで使用可能なデータ型

STR1, STR2: STRING

CONCAT: STRING

パラメータ

入力パラメータ	説明	備考
STR1 ('conveyor')	文字列1入力	
STR2 ('drive')	文字列2入力	

出力パラメータ	説明	備考
CONCAT (VarString3)	結果 ('conveyordrive')	

解 説

STR1入力文字列の後尾にSTR2入力文字列を付加し2つの文字列を結合した文字列を返します。

入力文字列と結果として返される文字列の最大は255文字となります。

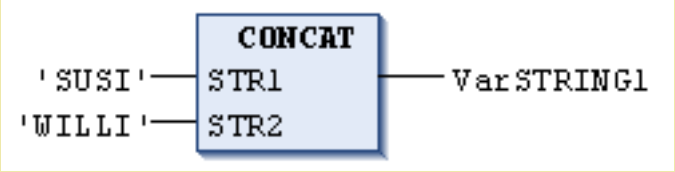
(ILの例) (結果は 'SUSIWILLI')

```
LD      'SUSI'
CONCAT  'WILLI'
ST      VarSTRING1
```

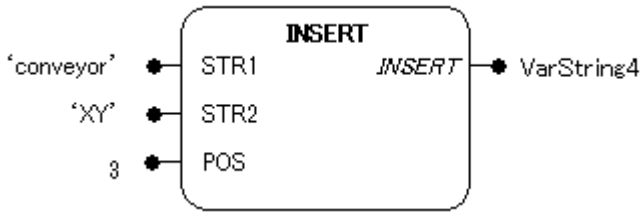
(STの例)

```
VarSTRING1 := CONCAT ('SUSI','WILLI');
```

(FBDの例)



INSERT:文字列挿入 [FUN]



機能

文字列の挿入をします。

パラメータで使用可能なデータ型

STR1, STR2: STRING

POS: INT

INSERT: STRING

パラメータ

入力パラメータ	説明	備考
STR1 ('conveyor')	挿入される文字列	
STR2 ('XY')	挿入する文字列	
POS (3)	STRの先頭を1とした位置	この位置の後に挿入

出力パラメータ	説明	備考
INSERT (VarString4)	結果 ('conXYveyor')	

解説

与えられた文字列 STR1 の指定位置へ文字列 STR2 を挿入します。

STR1 の文字位置 POS の後ろにSTR2 が挿入されます。

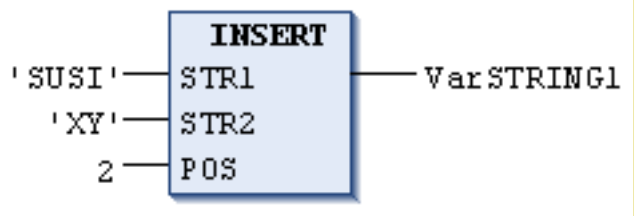
(ILの例) (結果は 'SUXYSI')

```
LD      'SUSI'
INSERT  'XY',
        2
ST      VarSTRING1
```

(STの例)

```
VarSTRING1 := INSERT ('SUSI','XY',2);
```

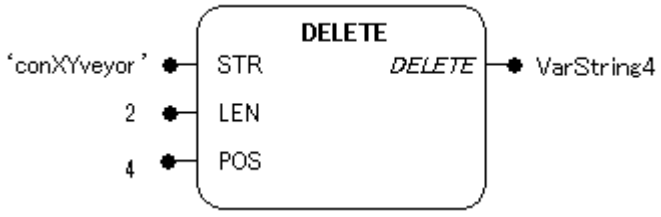
(FBDの例)



補 足

- POS は 0 を指定することはできません。文字列の最初の位置は 1 です。
- 文字列を他の文字列の前に挿入したいときは、ファンクション CONCAT を使用してください。

DELETE:文字列削除 [FUN]



機能

文字列の削除をします。

パラメータで使用可能なデータ型

STR: STRING

LEN, POS: INT

DELETE: STRING

パラメータ

入力パラメータ	説明	備考
STR ('conXYveyor')	削除箇所を含む文字列	
LEN (2)	削除する文字数	0 以上
POS (4)	STRの先頭を1とした削除開始位置	この位置から削除

出力パラメータ	説明	備考
DELETE (VarString4)	結果 ('conveyor')	

解 説

与えられた文字列 STR の文字位置 POS で始まる文字数 LEN 部分が削除されます。

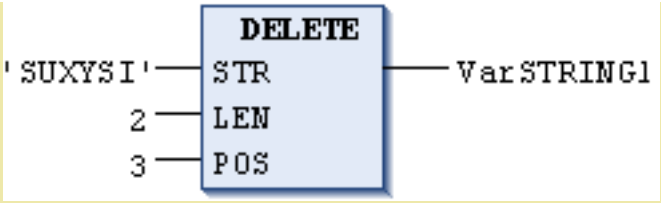
(ILの例) (結果は 'SUSI')

```
LD      'SUXYSI'
DELETE  2,
        3
ST      VarSTRING1
```

(STの例)

```
Var1 := DELETE ('SUXYSI',2,3);
```

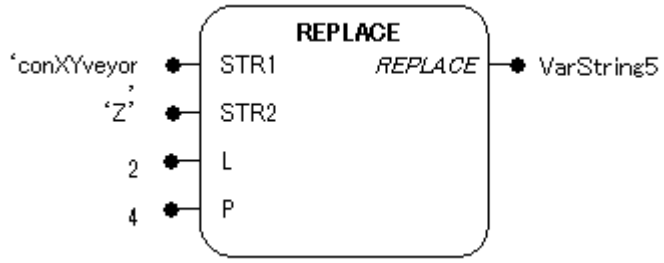
(FBDの例)



補 足

- POS は 0 を指定することはできません。文字列の最初の位置は 1 です。

REPLACE：文字列置換 [FUN]



機能

文字列の置換を行います。

パラメータで使用可能なデータ型

STR1, STR2：STRING

L, P：INT

REPLACE：STRING

パラメータ

入力パラメータ	説明	備考
STR1 ('conXYveyor')	置換箇所を含む文字列	
STR2 ('Z')	置換後の文字列	
L (2)	置換する文字数	0 以上
P (4)	STR1の先頭を1とした置換位置	この位置から L 文字置換

出力パラメータ	説明	備考
REPLACE (VarString5)	結果 ('conZveyor')	

解 説

文字列中のある文字列を他のものと置き換えます。

文字列 STR1 の文字位置 P から始まる文字数 L 分を STR2 に置き換えます。

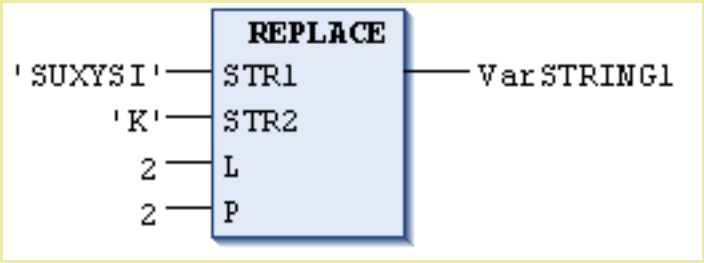
(ILの例) (結果は 'SKYSI')

```
LD      'SUXYSI'
REPLACE 'K'  ,
        2    ,
        2
ST      VarSTRING1
```

(STの例)

```
VarSTRING1 := REPLACE ('SUXYSI','K',2,2);
```

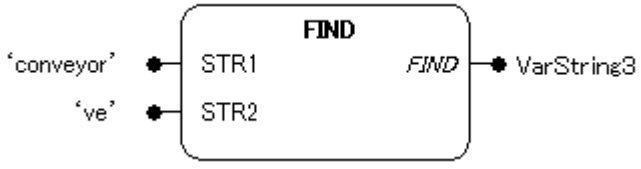
(FBDの例)



補 足

- P は 0 を指定することはできません。文字列の最初の位置は 1 です。

FIND:文字列検索 [FUN]



機能

文字列の検索をします。

パラメータで使用可能なデータ型

STR1, STR2: STRING

FIND: INT

パラメータ

入力パラメータ	説明	備考
STR1 ('conveyor')	文字列1	
STR2 ('ve')	STR1内で検索する文字列2	

出力パラメータ	説明	備考
FIND (VarString3)	結果 (4)	STR1の先頭は1

解説

与えられた文字列 STR1 内で文字列 STR2 の位置を検出します。

STR1 の中で STR2 が最初に現れる位置を結果として返します。

STR1 内にSTR2 が無ければ結果に0を返します。

(ILの例) (結果は '4')

```
LD      'abcdef'
FIND    'de'
ST      VarSTRING1
```

(STの例)

```
arINT1 := FIND ('abcdef','de');
```

(FBDの例)

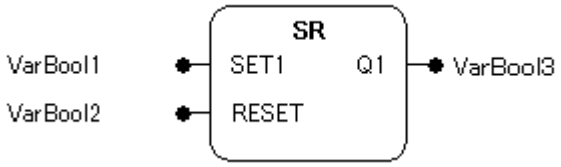
補 足

- 文字列の最初の位置は1です。

7.6.標準ファンクションブロック

分類	命令	機能
バイステーブル	SR	セット優先ラッチ
	RS	リセット優先ラッチ
カウンタ	CTU	アップカウンタ
	CTD	ダウンカウンタ
	CTUD	アップダウンカウンタ
タイマ	TON	オンディレイタイマ
	TOF	オフディレイタイマ
	TP	パルス幅出力
エッジ検出型	R_TRIG	立ち上がり検出
	F_TRIG	立ち下がり検出

SR：セット優先ラッチ [FB]



機能

SET1が優先するラッチです。SET1とRESETの両方の信号がTRUEならば出力Q1はTRUEになります。

パラメータで使用可能なデータ型

SET1, RESET：BOOL

Q1：BOOL

パラメータ

入力パラメータ	説明	備考
SET1 (VarBool1)	入力	
RESET (VarBool2)	リセット入力	

出力パラメータ	説明	備考
Q1 (VarBool3)	結果	

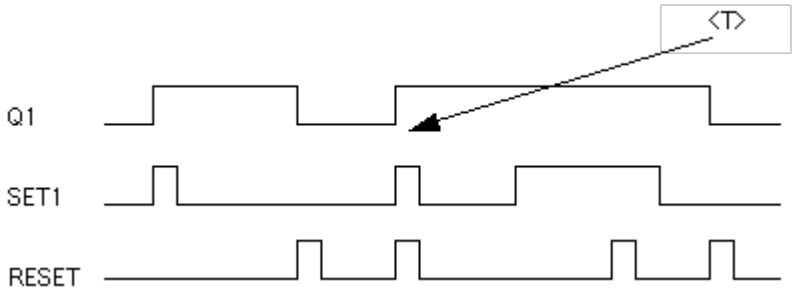
解 説

出力Q1のラッチを行います。

入力SET1 = TRUEの場合に出力Q1はセットされTRUEとなり、その後SET1がFALSEになってもQ1はTRUE状態が残ります。入力RESET = TRUEの場合にQ1がリセットされFALSEとなります。

SET1とRESETの両方の入力がTRUEの場合はSET1が優先されて出力Q1はセットされます。

初めてこのファンクションブロックが呼び出される際のQ1はFALSEです。



<T>: 同時の場合はセットが優先

宣言の例:

SRInst : SR ;

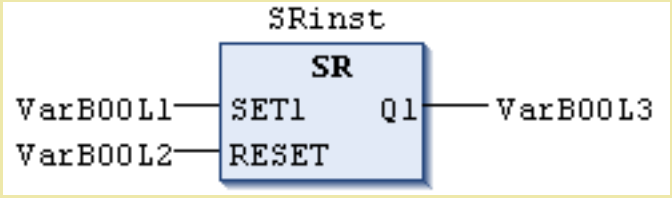
(ILの例)

```
CAL     SRinst  (
    SET1:= VarBool1 ,
  RESET:= VarBool2 )
LD      SRinst.Q1
ST      VarBool3
```

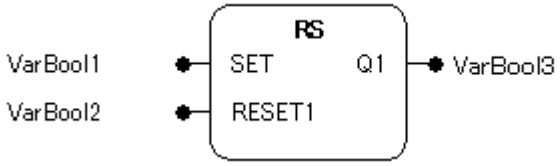
(STの例)

```
SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2 );
VarBOOL3 := SRInst.Q1 ;
```

(FBDの例)



RS:リセット優先ラッチ [FB]



機能

RESET1が優先するラッチです。SETとRESET1の両方の信号がTRUEならば出力Q1はFALSEになります。

パラメータで使用可能なデータ型

SET, RESET1: BOOL

Q1: BOOL

パラメータ

入力パラメータ	説明	備考
SET (VarBool1)	入力	
RESET1 (VarBool2)	リセット入力	

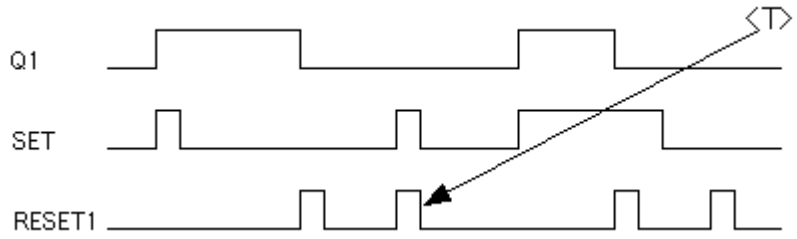
出力パラメータ	説明	備考
Q1 (VarBool3)	結果	

解説

出力Q1のラッチを行います。

入力SET = TRUEの場合に出力Q1はセットされTRUEとなり、その後SETがFALSEになっても、Q1はTRUE状態が残ります。入力RESET1 = TRUEの場合にQ1はリセットされFALSEとなります。

SETとRESET1の両方の入力がTRUEの場合はRESET1が優先されて出力Q1はリセットされます。
初めてこのファンクションブロックが呼び出される際のQ1はFALSEです。



<T>: 同時の場合はリセットが優先

宣言の例:

RSInst : RS ;

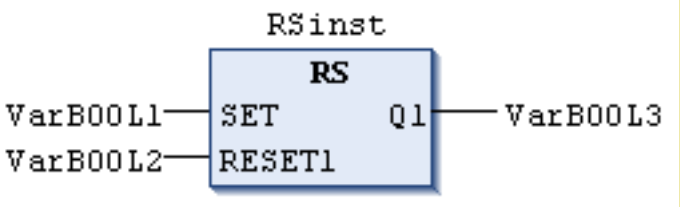
(ILの例)

```
CAL     RSinst  (
    SET:= VarBool1  ,
RESET1:= VarBool2  )
LD      RSinst.Q1
ST      VarBool3
```

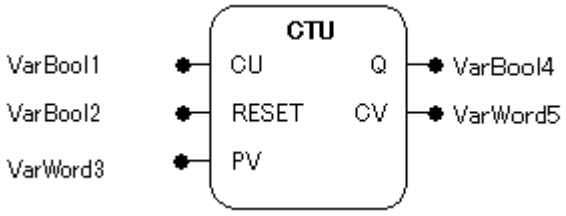
(STの例)

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1 ;
```

(FBDの例)



CTU : アップカウンタ [FB]



機能

カウント値をカウントアップしプリセット値(最大値)に達したことを知らせるアップカウンタです。

パラメータで使用可能なデータ型

CU, RESET : BOOL

PV : WORD

Q : BOOL

CV : WORD

パラメータ

入力パラメータ	説明	備考
CU (VarBool1)	カウントトリガ入力	立ち上がりで1加算
RESET (VarBool2)	リセット入力	
PV (VarWord3)	目標値	

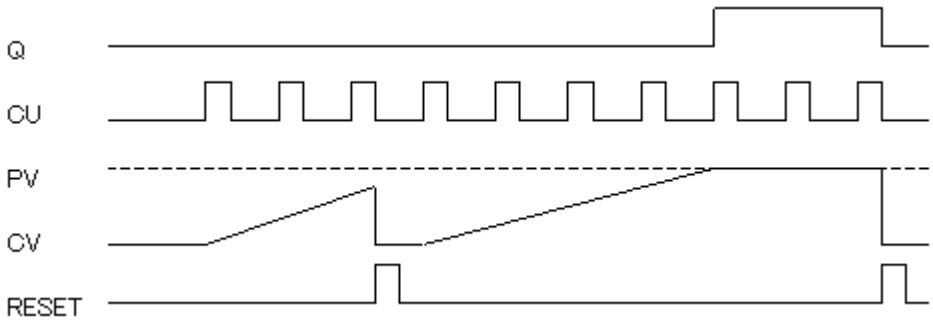
出力パラメータ	説明	備考
Q (VarBool4)	結果 (CV >= PV の場合 TRUE)	
CV (VarWord5)	カウント値	

解説

カウント値CVのカウントアップを行います。

RESET = FALSEの場合は入力CUの立ち上がりエッジでCVを1増加します。CVがプリセット値PVに到達すると出力Q = TRUEが出力され、このファンクションブロックはカウントを停止します。

RESET = TRUEの場合は出力Q = FALSE、カウンタCV = 0で初期化されます。



宣言の例:

CTUInst : CTU ;

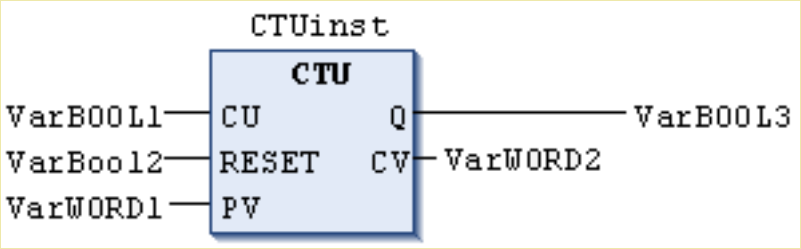
(ILの例)

```
CAL     CTUinst(
        CU:= VarBOOL1,
     Reset:= VarBOOL2,
        PV:= VarWORD1,
        CV=> VarWORD2)
LD      CTUinst.Q
ST      VarBOOL3
```

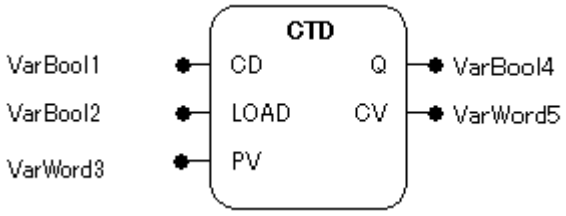
(STの例)

```
CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarWORD1);
VarBOOL3 := CTUInst.Q ;
VarWORD2 := CTUInst.CV;
```

(FBDの例)



CTD : ダウンカウンタ [FB]



機能

プリセット値からカウントダウンしてカウント値が0に達したことを知らせるダウンカウンタです

パラメータで使用可能なデータ型

CD, LOAD: BOOL

PV: WORD

Q: BOOL

CV: WORD

パラメータ

入力パラメータ	説明	備考
CD (VarBool1)	カウントトリガ入力	立ち上がりで1減算
LOAD (VarBool2)	ロード入力	
PV (VarWord3)	開始値	

出力パラメータ	説明	備考
Q (VarBool4)	結果 (CV = 0 の場合 TRUE)	
CV (VarWord5)	カウント値	

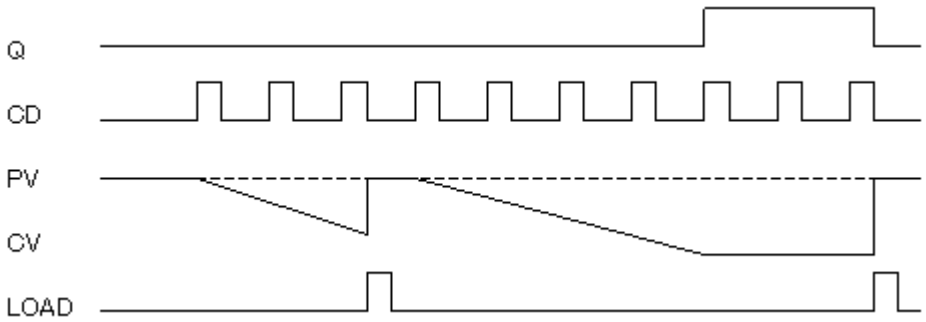
解説

カウント値CVのカウントダウンを行います。

LOAD = FALSE の場合は入力CDの立ち上がりエッジでCV値を1減少させます。CV = 0に到達すると出力 Q = TRUEを出力し、このファンクションブロックはカウントを停止します。

LOAD = TRUEの場合は、出力Q = FALSE、カウンタCV = PV で初期化されます。

初期状態でCV = 0の時もQ = TRUEとなります。



宣言の例:

CTDInst : CTD ;

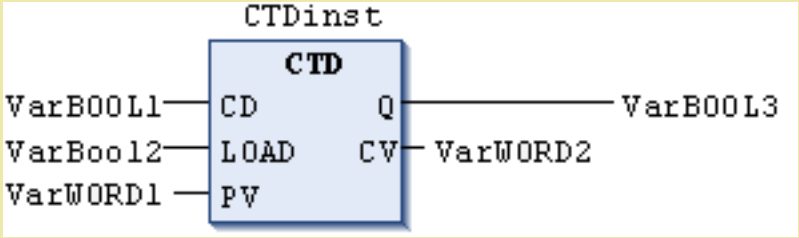
(ILの例)

```
CAL     CTDinst(
        CD:= VarBOOL1,
      LOAD:= VarBOOL2,
        PV:= VarWORD1,
        CV=> VarWORD2)
LD      CTDinst.Q
ST      VarBOOL3
```

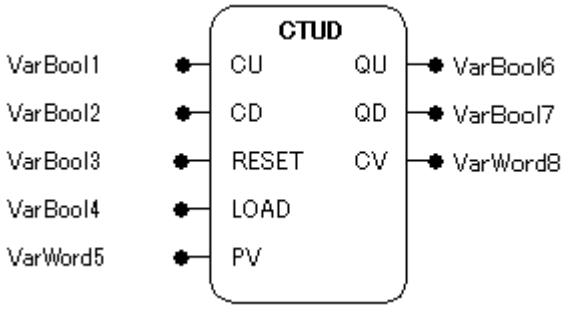
(STの例)

```
CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarWORD1);
VarBOOL3 := CTDInst.Q ;
VarWORD2 := CTDInst.CV;
```

(FBDの例)



CTUD : アップダウンカウンタ [FB]



機能

CUでカウントアップ、CDでカウントダウンし、カウント値が0あるいはプリセット値に達したことを知らせるカウンタです。

パラメータで使用可能なデータ型

CU, CD, RESET, LOAD: BOOL

PV: WORD

QU, QD: BOOL

CV: WORD

パラメータ

入力パラメータ	説明	備考
CU (VarBool1)	カウントトリガ入力	立ち上がりで1加算
CD (VarBool2)	カウントトリガ入力	立ち下がりで1減算
RESET (VarBool3)	リセット入力	
LOAD (VarBool4)	ロード入力	
PV (VarWord5)	カウンタ最大値	

出力パラメータ	説明	備考
QU (VarBool6)	結果 (加算してCV >= PV の場合 TRUE)	
QD (VarBool7)	結果 (減算してCV = 0 の場合 TRUE)	
CV (VarWord8)	カウント値	

解 説

入力CUの立ち上がりエッジでCVが1増加しCV = PVに達すると出力QU = TRUEが出力されます。
入力CDの立ち下がりエッジでCVが1減少しCV = 0に達すると出力 QD = TRUEが出力されます。
RESET = TRUE の場合はCV = 0、QU = FALSE、(QD = TRUE)で初期化されます。
LOAD = TRUE の場合はCV = PV、QD = FALSE、(QU = TRUE)で初期化されます。
RESETおよびLOADがFALSEでなければカウントは行われません。

宣言の例:

CTUDInst : CUTD ;

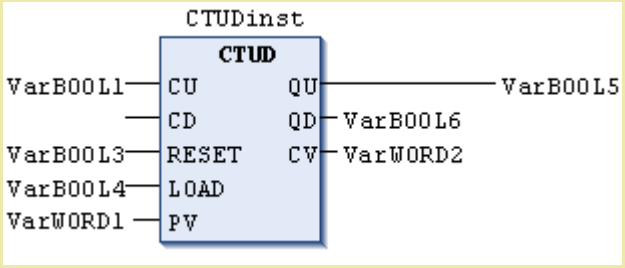
(ILの例)

```
CAL     CTUDinst(
        CU:= VarBOOL1,
      RESET:= VarBOOL3,
       LOAD:= VarBOOL4,
         PV:= VarWORD1,
        QD=> VarBOOL6,
        CV=> VarWORD2)
LD      CTUDinst.QU
ST      VarBOOL5
```

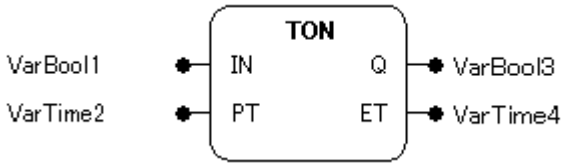
(STの例)

```
CTUDInst(CU := VarBOOL1, CD:= VarBOOL2, RESET := VarBOOL3,
LOAD:=VarBOOL4 , PV:= VarWORD1);
VarBOOL5 := CTUDInst.QU ;
VarBOOL6 := CTUDInst.QD ;
VarWORD2 := CTUDInst.CV;
```

(FBDの例)



TON : オンディレイタイマ [FB]



概要

入力がTRUE となってから指定の時間が経過した後に出力をTRUE にするオンディレイタイマです。

パラメータで使用可能なデータ型

IN: BOOL

PT: TIME

Q: BOOL

ET: TIME

パラメータ

入力パラメータ	説明	備考
IN (VarBool1)	開始入力	立ち上がりでET=0
PT (VarTime2)	プリセット時間	

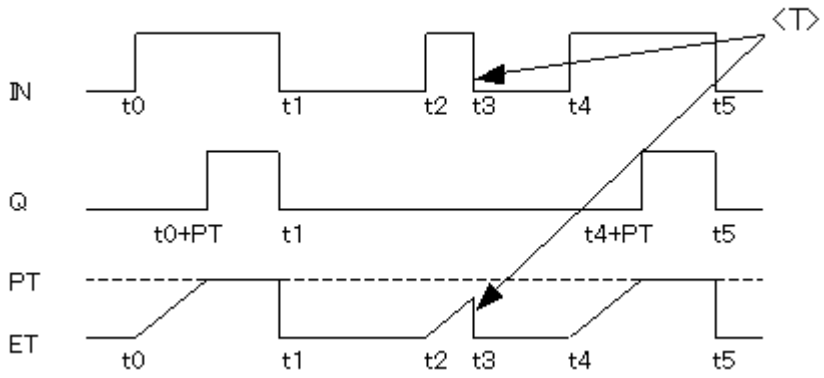
出力パラメータ	説明	備考
Q (VarBool3)	結果	IN＝TRUEかつ ET≧PTの時 TRUE
ET (VarTime4)	経過時間	

解説

入力がOFFからONになった後に決められた時間の間出力をONすることを遅延します。

入力 IN がFALSEからTRUEに変化すると経過時間ETのカウントを開始します。経過時間ETが遅延用プリセット時間PTに達すると出力Q = TRUEが出力され、この出力は入力IN = TRUEの間保持されます。

入力INがFALSEに戻ると出力Q = FALSEが出力され、経過時間ET = 0に初期化されます。



<T>: 遅延時間に達する前に入力がFALSEとなった場合のQはTRUEにならない

宣言の例:

TONInst : TON ;

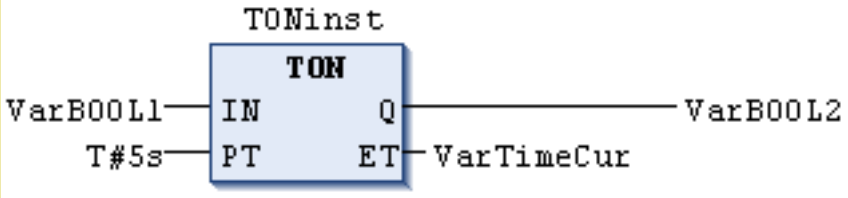
(ILの例)

```
CAL     TONinst (
        IN:= VarBOOL1 ,
        PT:= T#5s ,
        ET=> VarTimeCur )
LD      TONinst.Q
ST      VarBOOL2
```

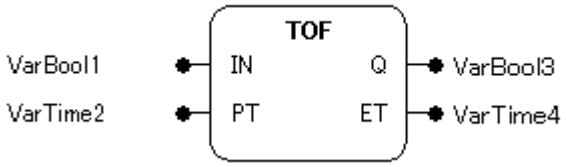
(STの例):

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

(FBDの例)



TOF : オフディレイタイマ [FB]



機能

入力がFALSEとなってから指定の時間が経過するまで出力をFALSEとしないオフディレイタイマです。

パラメータで使用可能なデータ型

IN: BOOL

PT: TIME

Q: BOOL

ET: TIME

パラメータ

入力パラメータ	説明	備考
IN (VarBool1)	開始入力	立ち上がりでET=0
PT (VarTime2)	プリセット時間	

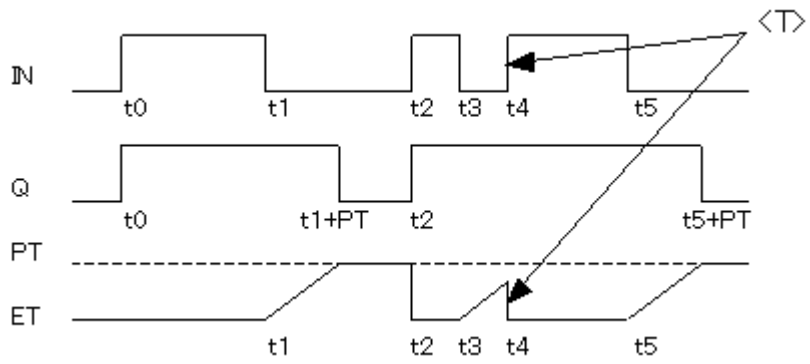
出力パラメータ	説明	備考
Q (VarBool3)	結果	IN＝FALSEかつ ET≧PTの時 FALSE
ET (VarTime4)	経過時間	

解説

入力がONからOFFになった後に決められた時間の間出力をOFFすることを遅延します。

入力 IN がTRUEからFALSEに変化すると経過時間ETのカウントを開始します。経過時間ETが遅延用プリセット時間PTに達すると出力Q = FALSEが出力され、この出力は入力IN = FALSEの間保持されます。

入力INがTRUEに戻ると出力Q = TRUEが出力され、経過時間ET = 0に初期化されます。



<T>: 遅延時間に達する前に入力がTRUEとなった場合のQはFALSEにならない

宣言の例:

TOFInst : TOF ;

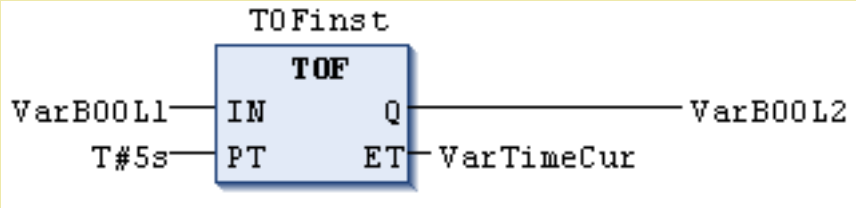
(ILの例)

```
CAL     TOFinst (
        IN:= VarBOOL1 ,
        PT:= T#5s ,
        ET=> VarTimeCur )
LD      TOFinst.Q
ST      VarBOOL2
```

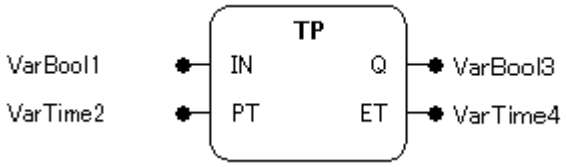
(STの例)

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TOFInst.Q;
```

(FBDの例)



TP : パルス幅出力 [FB]



機能

指定の持続時間を持ったパルスを発生するタイマです。

パラメータで使用可能なデータ型

IN: BOOL

PT: TIME

Q: BOOL

ET: TIME

パラメータ

入力パラメータ	説明	備考
IN (VarBool1)	開始入力	立ち上がりでET=0
PT (VarTime2)	プリセット時間	

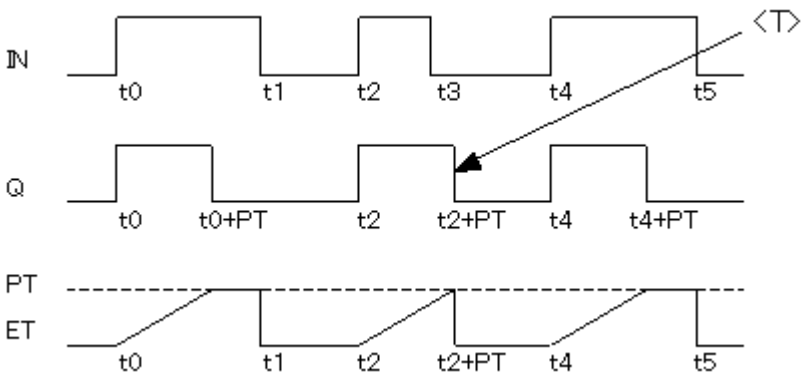
出力パラメータ	説明	備考
Q (VarBool3)	結果	IN＝TRUE後ET＜PTの間TRUE
ET (VarTime4)	経過時間	

解説

パルスを作成します。

入力INがFALSEからTRUEに変化すると出力Qにパルス用プリセット時間PTの長さでパルスが作成されます。PT時間経過前のパルス持続中に入力INが変化しても出力Qのパルス持続には影響しません。

すでに経過した時間は経過時間ETに表示されます。



<T>: 入力の立ち上がりでスタートし、入力の変化に関わらずPT幅のパルスを出力

宣言の例:

TPInst : TP ;

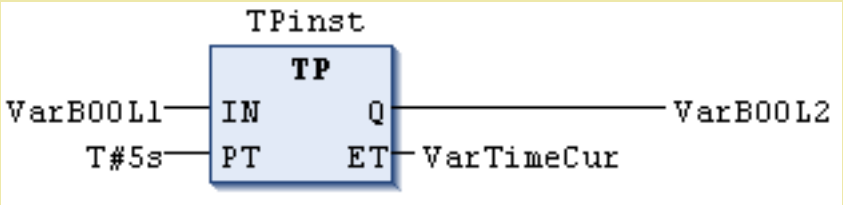
(ILの例)

```
CAL     TPinst  (
        IN:= VarBOOL1   ,
        PT:= T#5s       ,
        ET=> VarTimeCur )
LD      TPinst.Q
ST      VarBOOL2
```

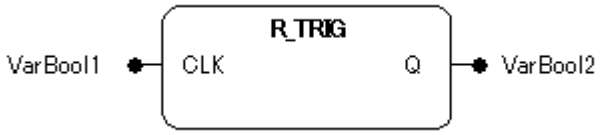
(STの例)

```
TPInst(IN:= VarBOOL1, PT:= T#5s);
VarBOOL2 := TPInst.Q;
```

(FBDの例)



R_TRIG: 立ち上がりエッジ検出 [FB]



機能

立ち上がりエッジ(微分)を検出します。エッジを検出したときに単一のパルスを発生します。

パラメータで使用可能なデータ型

CLK: BOOL

Q: BOOL

パラメータ

入力パラメータ	説明	備考
CLK (VarBool1)	入力	

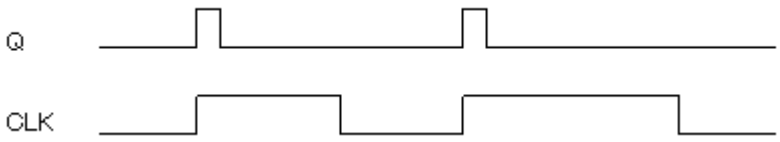
出力パラメータ	説明	備考
Q (VarBool2)	結果	

解説

立ち上がりエッジを検出します。

入力CLKで立ち上がりエッジが検出されると出力QはFALSEからTRUEに変化します。Qはファンクションブロックの次の実行(通常プログラムの1周期)までTRUEの状態を維持します。

初めてファンクションブロックが呼び出される場合は最初のエッジが検出されるまでのQはFALSEとなります。



宣言の例:

RTRIGInst : R_TRIG ;

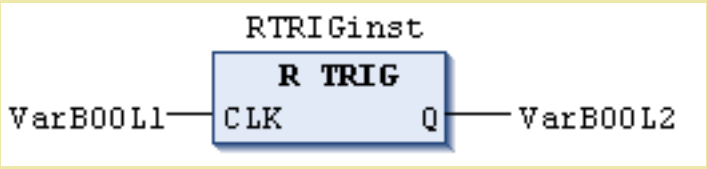
(ILの例)

```
CAL     RTRIGinst (
        CLK:= VarBool1 )
LD      RTRIGinst.Q
ST      VarBool2
```

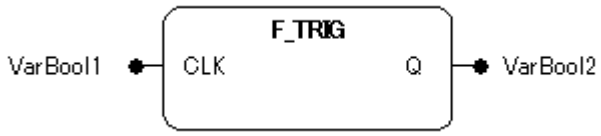
(STの例)

```
RTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := RTRIGInst.Q;
```

(FBDの例)



F_TRIG: 立ち下がりエッジ検出 [FB]



機能

立ち下がりエッジ(微分)を検出します。エッジを検出したときに単一のパルスを発生します。

パラメータで使用可能なデータ型

CLK: BOOL

Q: BOOL

パラメータ

入力パラメータ	説明	備考
CLK (VarBool1)	入力	

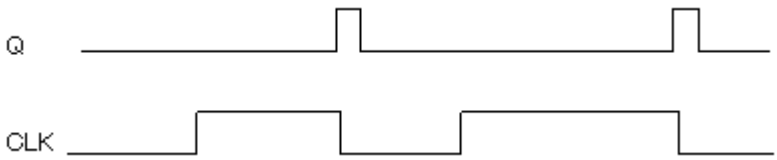
出力パラメータ	説明	備考
Q (VarBool2)	結果	

解 説

立ち下がりエッジを検出します。

入力CLKで立ち下がりエッジが検出されると出力QはFALSEからTRUEに変化します。Qはファンクションブロックの次の実行(通常プログラムの1周期)までTRUEの状態を維持します。

初めてファンクションブロックが呼び出される場合は最初のエッジが検出されるまでのQはFALSEとなります。



宣言の例:

FTRIGInst : F_TRIG ;

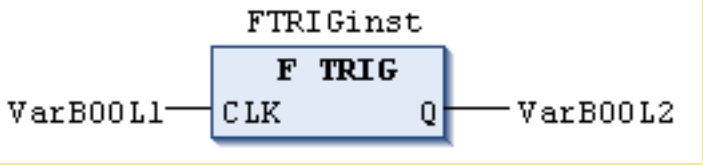
(ILの例)

```
CAL     FTRIGinst (
    CLK:= VarBOOL1 )
LD      FTRIGinst.Q
ST      VarBOOL2
```

(STの例)

```
FTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := FTRIGInst.Q;
```

(FBDの例)


FTRIGinst
F TRIG
VarBOOL1
CLK
Q
VarBOOL2

索 引

(このページは空白です)