EcoStruxure Machine Expert

プログラミングガイド

06/2019

EIO0000002854JP.00

www.schneider-electric.com

Schneider Electric

本書の情報には本書に記載された製品についての一般的説明および性能の技術特性が含まれます。本書は、お客様の特定の用途に対する本製品の適合性または信頼性を確約するために作成されたものではありません。お客様またはインテグレーター様は自らの責任で、関連する特定の用途またはその使用に関する本製品のリスク分析、評価、および試験を完全かつ適切に行なってください。シュナイダーエレクトリック社あるいは系列会社は、本書に記載された情報の誤用に対して一切の責任を負いかねますので、あらかじめご了承ください。本書の内容について改善点や修正点の提案がある場合、また何らかの誤りを発見した場合には、弊社までご連絡ください。

本製品を設置して使用する際には、関連する州、地域、地区の安全規定をすべて順守する必要があります。安全のため、また、記録されたシステムデータの適合性を確保するため、部品の修理は製造業者にお任せください。

装置を技術的な安全要件がある用途に使用する場合、関連する指示に従ってください。

シュナイダーエレクトリックのハードウェア製品には必ず、シュナイダーエレクトリック製のソフトウェアまたは承認された ソフトウェアをご使用ください。この指示に従わない場合、人的損害、物的損害、また不適切な動作が生じる可能性があります。

この情報に従わない場合、人的損害や装置の損傷を招くおそれがあります。

目次

安全に関する使用上の注意

重要情報

お断り

本書をよくお読みいただき、装置の正しい取り扱いと機能を十分ご理解いただいた上で、設置、操作、保守を行ってください。本書および装置には以下の表示が使われています。これらは潜在的な危険を警告したり、手順を明確化あるいは簡素化する情報について注意を呼びかけるものです。

この記号が「危険」または「警告」安全ラベルに追加されると、電気的な危険が存在し、指示に従わないと人身傷害の危険があることを示します。

安全警告記号です。人的傷害の危険性があることを警告します。
この記号の後に記載された安全に関する情報に従って、人的傷害や死亡の危険性を回避してください。

⚠ 危険

危険は、危険が生じる可能性のある状況を示します。回避しないと、死亡や重傷を**招きます**。

⚠ 警告

警告は、危険が生じる可能性のある状況を示します。回避しないと、死亡や重傷を**招くおそれ**があります。

⚠ 注意

注意は、危険が生じる可能性のある状況を示します。回避しないと、軽傷を**招くおそれ**があります。

注記

この表示は、指示に従わないと物的損害を負う可能性があることを示します。

以下の点に注意してください。

電気装置の設置、操作、サービス、および保守は有資格者のみが行うことができます。定められた範囲外の使用によって生じた結果については、シュナイダーエレクトリックは一切の責任を負いかねます。

有資格者とは、電気装置の構造および操作ならびに設置に関する技術と知識を持ち、関連する危険性を認識して回避するための安全トレーニングを受けた人を指します。

本書について

概要

本書の適用範囲

本書では、EcoStruxure Machine Expert のユーザーインターフェイスと機能を説明します。詳細については、EcoStruxure Machine Expert オンラインヘルプにあるその他のドキュメントを参照してください。

有効性に関する注意

本書は、EcoStruxure™ Machine Expert V1.1 のリリース時に更新されました。

関連ドキュメント

ドキュメントのタイトル	型式番号
EcoStruxure Machine Expert Introduction	EIO0000002836 (ENG); EIO0000002837 (FRE); EIO0000002838 (GER); EIO0000002840 (SPA); EIO0000002839 (ITA); EIO0000002841 (CHS)
EcoStruxure Machine Expert Menu Commands Online Help	EIO0000002860 (ENG); EIO0000002861 (FRE); EIO0000002862 (GER); EIO0000002864 (SPA); EIO0000002863 (ITA); EIO0000002865 (CHS)
EcoStruxure Machine Expert Compatibility and Migration User Guide	EIO0000002842 (ENG); EIO0000002843 (FRE); EIO0000002844 (GER); EIO0000002846 (SPA); EIO0000002845 (ITA); EIO0000002847 (CHS)
EcoStruxure Machine Expert ファンクションおよびライブラリーユーザーガイド (EcoStruxure Machine Expert Functions and Libraries User Guide)	EIO0000002829 (ENG); EIO0000002830 (FRE); EIO0000002831 (GER); EIO0000002833 (SPA); EIO0000002832 (ITA); EIO0000002834 (CHS)
EcoStruxure Machine Expert Controller Assistant User Guide	EIO0000001671 (ENG); EIO0000001672 (FRE); EIO0000001673 (GER); EIO0000001675 (SPA); EIO0000001674 (ITA); EIO0000001676 (CHS)
EcoStruxure Machine Expert Device Type Manager (DTM) User Guide	EIO0000003047 (ENG); EIO0000003048 (FRE); EIO0000003049 (GER); EIO0000003051 (SPA); EIO0000003050 (ITA); EIO0000003052 (CHS)
EcoStruxure Machine Expert TwidoEmulationSupport Library Guide	EIO0000001692 (ENG); EIO0000001693 (FRE); EIO0000001694 (GER); EIO0000001696 (SPA); EIO0000001695 (ITA); EIO0000001697 (CHS)

ドキュメントのタイトル	型式番号
EcoStruxure Machine Expert Network Variable Configuration SE_NetVarUdp Library Guide	EIO0000002974 (ENG); EIO0000002975 (FRE); EIO0000002976 (GER); EIO0000002978 (SPA); EIO0000002977 (ITA); EIO0000002979 (CHS)

これらのテクニカルパブリケーション、およびその他の技術情報は、当社のウェブサイト *www.schneider-electric.com/en/download* からダウンロードしていただけます。

製品関連情報

⚠ 警告

制御不能

- 制御手法の設計者は制御パスの障害モードが発生するおそれを考慮する必要があり、特定の重要制御機能については、パス障害の最中および終了後に安全な状態を実現するための方策を準備しておく必要があります。重要制御機能の例としては、緊急停止、オーバートラベル停止、停電、および再起動があります。
- 重要な制御機能に対しては、別のまたは冗長性のある制御パスを用意してください。
- システム制御パスには、データ通信が含まれることがあります。予期しないデータの転送遅れや障害について考慮する必要があります。
- あらゆる事故防止規制および地域の安全性ガイドライン [1] を遵守してください。
- 運用を開始する前に、各実装について、正しく動作するかどうかを個別に十分にテストする必要があります。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

[1] 詳細は、NEMA ICS 1.1 (最新版)、"Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control"、および NEMA ICS 7.1 (最新版)、"Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems"、または該当地域での同等のガイドラインを参照してください。

⚠ 警告

装置の意図しない動作

- 本装置には、シュナイダーエレクトリック認定のソフトウェアのみ使用してください。
- ハードウェアの設定を変更した場合は、必ずアプリケーションプログラムも更新してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

規格から派生した用語

技術用語、専門用語、シンボル、本書の記述、また本製品での表示は、国際規格用語および定義に由来しています。

安全機能システム、ドライブ、一般オートメーションにおいて、用語は、*安全性*、*安全機能*、*安全状態*、*異常*、*異常リセット*、*誤動作*、*障害*、*エラー*、*エラーメッセージ*、*危険*等を含みますが、それに限定されません。

特に以下の規格が含まれます。

規格	詳細
IEC 61131-2:2007	プログラマブルコントローラー、第 2 部 : 機器要件、およびテスト
ISO 13849-1:2015	機械の安全性 - 制御システムの安全関連部品 設計の一般原則
EN 61496-1: 2013	機械の安全性 - 電気感光性保護機器 第 1 部 : 一般要件、およびテスト
ISO 12100: 2010	機械類の安全性 - 設計の一般原則 - リスク評価とリスク低減
EN 60204-1: 2006	機械の安全性 - 装置の電気機器 - 第 1 部 : 一般要件

規格	詳細
ISO 14119: 2013	機械類の安全性 - ガードと共同するインターロック装置 - 設計、および選択のための原則
ISO 13850:2015	機械類の安全性 - 非常停止 - 設計原則
IEC 62061: 2015	機械類の安全性 - 安全関連の電気・電子・プログラマブル電子制御システムの機能安全
IEC 61508-1: 2010	電気 / 電子 / プログラマブル電子安全関連システムの機能安全 : 一般要件
IEC 61508-2: 2010	電気 / 電子 / プログラマブル電子安全関連システムの機能安全 : 電気 / 電子 / プログラマブル電子安全関連システムの要件。
IEC 61508-3: 2010	電気 / 電子 / プログラマブル電子安全関連システムの機能安全 : ソフトウェア要件
IEC 61784-3:2016	産業用通信ネットワーク - プロファイル - 第 3 部 : 機能安全フィールドバス - 一般規則およびプロファイルの定義
2006/42/EC	機械指令
2014/30/EU	電磁両立性指令
2014/35/EU	低電圧指令

本書で使われている用語には下記の規格も含まれています。

規格	詳細
IEC 60034 シリーズ	回転電気機械
IEC 61800 シリーズ	可変速電気駆動システム
IEC 61158 シリーズ	計測制御用デジタルデータ通信 – 産業制御システム用のフィールドバス

*動作領域*は特定の危険性記述と併せて使われる場合があり、*機械指令* (*2006/42/EC*) と *ISO 12100:2010* の*危険区域*と*危険地帯*に定義されています。

注記：前述の規格は、本書記載の特定の機器には適用されない場合があります。本書に記載されている製品の適用規格についての詳細は製品の特徴が記載された表を参照してください。

第 I 部
概要

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
1	EcoStruxure Machine Expert Logic Builder の概要	23
2	EcoStruxure Machine Expert Logic Builder ユーザーインターフェイス	27
3	基本概念	45

第 1 章
EcoStruxure Machine Expert Logic Builder の概要

この章について

この章には次の項目が含まれています。

EcoStruxure Machine Expert Logic Builder とは？

一般概要

Logic Builder では EcoStruxure Machine Expert プロジェクトの設定およびプログラミング環境が構築できます。

プロジェクトのさまざまな要素を別々のビューに表示でき、これらのビューは EcoStruxure Machine Expert のユーザーインターフェイスやデスクトップ上に自由に配置できます。このビュー構造を使用すると、ハードウェアとソフトウェアの要素をドラッグ & ドロップでプロジェクトに追加できます。プロジェクトのコンテンツを作成するためのメインの設定ダイアログボックスは、Logic Builder 画面の中央に表示されます。

簡単な設定とプログラミングに加えて、Logic Builder は強力な診断と保守機能も提供します。

EcoStruxure Machine Expert Logic Builder でできること

プロジェクトの設定とプログラミング

Logic Builder を使って、EcoStruxure Machine Expert のプロジェクトにロジックをプログラミングしたり、デバイスを追加したりできます。

設計をスムーズに行えるよう、次の機能が用意されています。

- **コントローラー、HMI & iPC、デバイス & モジュール、その他**のハードウェアカタログビューが別々に用意されているので、ドラッグ & ドロップで簡単にハードウェアデバイスをプロジェクトに追加できます。デバイステンプレートやファンクションテンプレートも使用できます。
- **変数、アセット、マクロ、ツールボックス、ライブラリー** のソフトウェアカタログビューが別々に用意されているので、ドラッグ & ドロップで簡単にさまざまなソフトウェアの要素をプロジェクトに追加できます。例えば、**アセット**ビューを使用して、ファンクションブロックや POU の作成および管理ができます。

EcoStruxure Machine Expert は作業中の内容に関係するビューのみを表示するパースペクティブ (*41* ページ) を、ハードウェア設定、ソフトウェア設定、およびオンラインモード用にそれぞれ用意しています。パースペクティブはデフォルト設定のまま使用することもできますが、最もよく使用するビューでオリジナルのパースペクティブを作成することもできます。

プロジェクトのビルド

Logic Builder は、EcoStruxure Machine Expert プロジェクトをビルドするさまざまな方法 (**ビルド**、**すべてのビルド**、または**すべてクリーン**など) を提供します。

コントローラーとの通信

Logic Builder は、Ethernet ネットワーク上で利用可能なコントローラーを検知するスキャン機能を提供します。コントローラーとの通信で使われるさまざまなプロトコルに対応しています。

通信の確立後、アプリケーションをコントローラーへアップロードしたりコントローラーからダウンロードすることが可能です。コントローラーでアプリケーションを開始、および停止できます。

オンライン機能と監視

Logic Builder のオンラインと監視機能を使用して以下のタスクが可能です。

- プログラムコードと**ウォッチ**ビューの値のオンライン監視
- オンラインでの変更操作
- オンラインでのトレースの設定
- オンラインでトレースの監視
- オンラインモードでの診断およびテストのために内蔵ビジュアライゼーションを使用して機器と通信
- コントローラーやデバイスのステータスの読み取り
- デバッグ機能を使用した潜在的なプログラミングロジックエラーの検出

第 2 章
EcoStruxure Machine Expert Logic Builder ユーザーインターフェイス

この章について

この章には次の項目が含まれています。

EcoStruxure Machine Expert Logic Builder 画面の要素

概要

Logic Builder は、以下で構成されています。

- メニューとツールバー
- **ナビゲーター** ビュー
- **カタログ** ビュー
- メインエディターウィンドウ

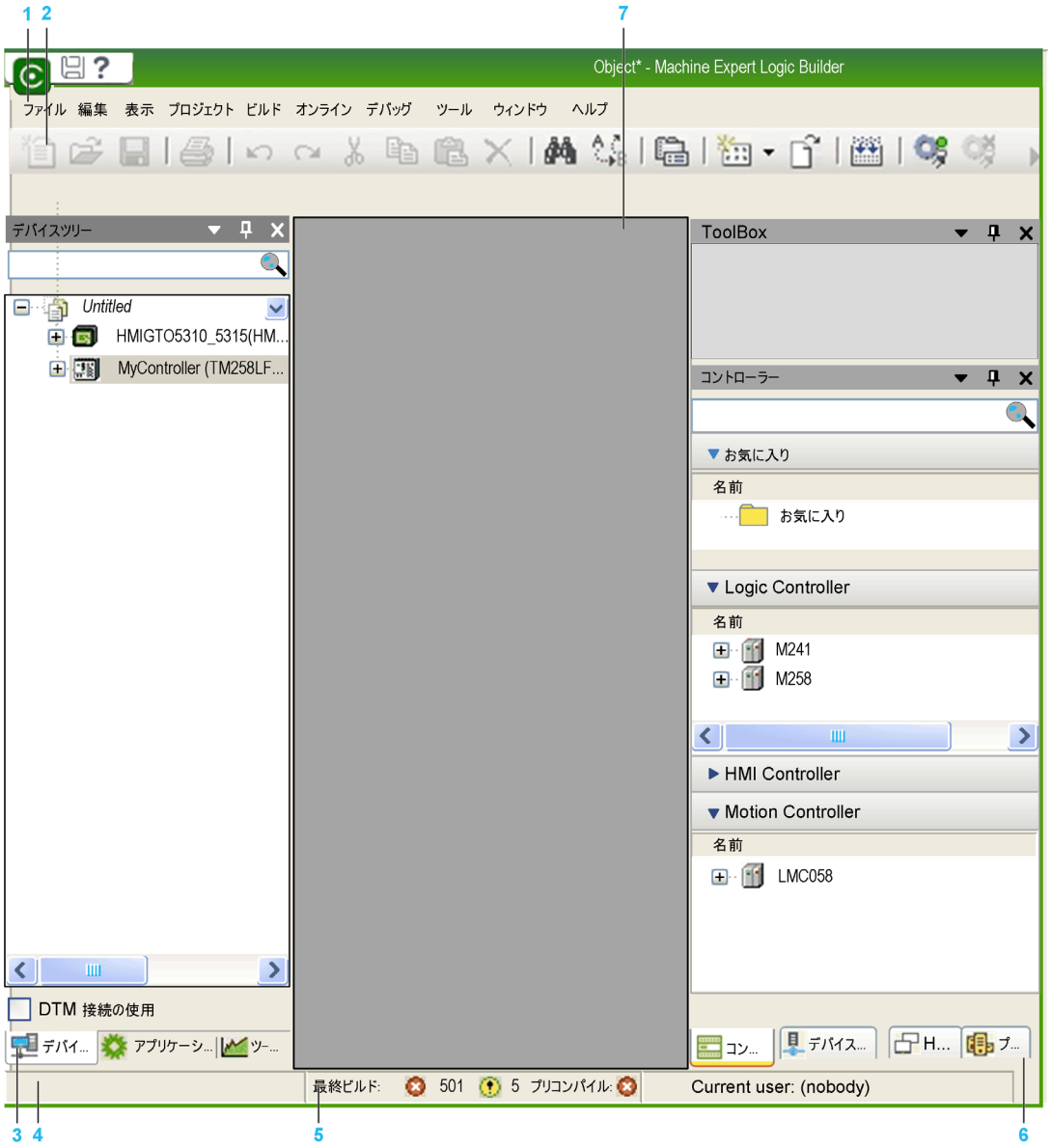
Logic Builder を開くとデフォルトの画面レイアウトが表示されます。本書ではデフォルト位置について説明します。

*ユーザーインターフェイスのカスタマイズ*の章 (*40* ページ) にある通り、レイアウトを自由に変更できます。**カスタマイズ**ダイアログボックスで現在の設定を確認し変更できます。デフォルト設定では**ツール**メニューにあります。

ビューの移動、ドッキング / ドッキング解除、ウィンドウのサイズ変更や閉じるを行うことで、いつでもビューとウィンドウを並べ替えることができます。並べ替えた位置はプロジェクトと共に保存されます。プロジェクトを再度開くと、プロジェクトを保存した時の位置でレイアウトが表示されます。
ビューの位置は別にパースペクティブ (*41* ページ) に保存されます。

デフォルトの Logic Builder 画面

Logic Builder 画面のメニュー、バー、およびビューのデフォルト位置



1 メニューバー
2 ツールバー
3 マルチタブ**ナビゲーター**: **デバイスツリー**、**ツールツリー**、**アプリケーションツリー**、**ファンクションツリー**
4 **メッセージ**ビュー
5 情報とステータスバー
6 マルチタブカタログビュー: ハードウェアカタログ: **コントローラー**、**HMI & iPC**、**デバイス & モジュール**、**その他** ソフトウェアカタログ: **変数、アセット。マクロ**、**ツールボックス**、**ライブラリー**
7 マルチタブのエディタービュー

デフォルトコンポーネント

Logic Builder 画面は以下のコンポーネントがデフォルトで表示されます。

コンポーネント	詳細
メニューバー	**ツール → カスタマイズ**ダイアログボックスで選択したコマンドのメニューを表示します。
ツールバー	**ツール → カスタマイズ**ダイアログボックスで選択したツールを実行するボタンが表示されます。

コンポーネント	詳細
マルチタブの**ナビゲーター**	タブになっている次の**ナビゲーター**を使うと、プロジェクト内のさまざまなオブジェクトをツリー構造で切り替え表示できます。 ● **デバイスツリー** ● **アプリケーションツリー** ● **ツールツリー** ● **ファンクションツリー** 詳細については、*マルチタブのナビゲーター* (*33* ページ) の章を参照してください。
メッセージビュー	プリコンパイル、コンパイル、ビルド、ダウンロードなどの操作に対するメッセージを表示します。詳細は、**メッセージ** ビューコマンドの説明を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
情報とステータスバー	次の情報を表示します。 ● 現在のユーザーに関する情報。 ● エディターが開いている場合、編集モード、現在の位置に関する情報。 詳細は、この章の*情報とステータスバー*のセクションを参照してください。
マルチタブ**カタログ**ビュー	**カタログ**ビューは、ハードウェアやソフトウェアのオブジェクトがリストされたさまざまなタブで構成されています。 ● **ハードウェアカタログ** ❍ **コントローラー** ❍ **HMI & iPC** ❍ **デバイス & モジュール** ❍ **その他** ● **ソフトウェアカタログ** ❍ **変数** ❍ **アセット** ❍ **マクロ** ❍ **ツールバー** ❍ **ライブラリー** 詳細については、*マルチタブカタログビュー* (*39* ページ) の章を参照してください。
マルチタブエディターウィンドウ	それぞれのエディターでオブジェクトを作成するために使用します。言語エディター (ST エディター、CFC エディター) の場合、通常は言語エディターが下に、宣言エディターが上になるウィンドウが表示されます。 他のエディターの場合、ダイアログボックスが表示されます (例 : タスクエディター、デバイスエディター)。POU の名前やリソースオブジェクトがこのビューのタイトルバーに表示されます。**オブジェクトの編集**コマンドを実行して、オフラインまたはオンラインモードでオブジェクトをエディターウィンドウに表示できます。

情報とステータスバー

Logic Builder 画面の下端にあるバーには、3 種類の情報が表示されます。

- ログインしているユーザーに関する情報。
- エディターウィンドウで作業している場合 : カーソルの位置と編集モードのステータス。
- オンラインモードの場合 : プログラムステータス。

ログインしているユーザーに関する情報。

各プロジェクトには、ユーザーおよびアクセス管理の設定があります (**プロジェクト → ユーザー管理 → 権限 ...** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*)。ログインしているユーザーの名前はステータスバーに表示されます。

エディターウィンドウのカーソルの位置

カーソルの位置は、エディターウィンドウの左端または上端から数えます。

略語	詳細
Ln	カーソルがある行
Col	カーソルがある列 (列には 1 つのスペース、文字または数字が含まれます。)

略語	詳細
Ch	文字数 (ここでは、1 文字とは単一の文字または数字だけではなく、例えば、4 つの列を含む 1 つのタブもあり得ます。)

フィールドをダブルクリックして **Go To Line** ダイアログボックスを開きます。ここでカーソルの位置を入力できます。

編集モードのステータスは次の略語で表示されます。

略語	詳細
INS	挿入モード
OVR	上書きモード

このフィールドをダブルクリックして設定を切り替えます。

次のプログラムステータスが表示されます。

テキスト	詳細
Program loaded	プログラムがデバイスに読み込まれました。
Program unchanged	デバイスのプログラムはプログラミングシステムのものと一致します。
Program modified (Online Change)	デバイスのプログラムがプログラミングシステムのものと異なります。 オンラインでの変更が必要です。
Program modified (Full download)	デバイスのプログラムがプログラミングシステムのものと異なります。 フルダウンロードが必要です。

オンラインモードの情報

デバイス上のアプリケーションステータス。

テキスト	背景色	詳細
RUN	緑	プログラム実行中。
STOP	赤	プログラム停止。
HALT ON BP	赤	プログラムがブレークポイントで停止。
次のステータスは、コントローラーがサイクルに依存しない監視 (デバイスディスクリプションの設定による) に対応している場合にのみ表示されます。		
IN CYCLE	白	1 サイクル内で監視対象の式の値が読み込まれることを示します。
OUT OF CYCLE	赤	1 サイクル内では監視対象の変数の値を取得できないことを示します。

ウォッチウィンドウとエディターのオンラインビュー

ウォッチウィンドウとオンラインエディタービューでは、POU の監視ビューやウォッチ式のユーザー定義リストを表示します。

ウィンドウ、ビュー、エディターウィンドウ

Logic Builder には 2 種類のウィンドウがあります。

- 1 つは、EcoStruxure Machine Expert ウィンドウの任意の位置にドッキングしたり、EcoStruxure Machine Expert ウィンドウとは独立してドッキング解除した状態で画面に配置することができます。また、EcoStruxure Machine Expert ウィンドウの枠内にタブとして非表示にすることもできます (*ユーザーインターフェイスのカスタマイズ*の章 (*40* ページ) を参照してください)。これらのウィンドウは、プロジェクトの 1 つのオブジェクトに依存しない情報を表示します (例えば、**メッセージ**ビューや**デバイスツリー**)。**表示** メニュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* からアクセスできます。ほとんどのビューには、ウィンドウ内のソート、表示、検索ボタンを備えたツールバーが含まれています。
- もう 1 つのウィンドウは、特定のプロジェクトオブジェクトを表示または編集する際に開きます。それらはマルチタブのエディターウィンドウに表示されます。非表示にしたり EcoStruxure Machine Expert ウィンドウからドッキングを解除することはできません。**ウィンドウ**メニューからアクセスできます。

ウィンドウの切り替え

EcoStruxure Machine Expert では、開いているビューとエディターを切り替えることができます。開いているビューとエディターを切り替えるには、**Ctrl** キーと **Tab** キーを同時に押します。開いているビューとエディターのリストが表示されます。**Ctrl** キーが押されている間、リストは表示されたままになります。特定のビューまたはウィンドウを選択するには、**Tab** キーまたは**矢印**キーを同時に押します。

マルチタブのナビゲーター

概要

マルチタブ**ナビゲーター**は、Logic Builder 画面にデフォルトで表示されています。

デフォルトで次のナビゲータが使用できます。

- **デバイスツリー**: アプリケーションを実行するデバイスの管理ができます。
- **アプリケーションツリー**: プロジェクト特有、およびグローバルの POU やタスクをひとつのビューで管理できます。
- **ツールツリー**: プロジェクト特有、およびグローバルライブラリーやその他の要素をひとつのビューで管理できます。
- **ファンクションツリー**: 必要に応じて、コントローラーのコンテンツをグループ化することができます。

表示メニューから各ビューにアクセスできます。

ナビゲーターに要素を追加

ナビゲーターのルートノードはプログラミング可能なデバイスを表します。このルートの下にさらに要素を挿入できます。

ナビゲーターのノードに要素を追加するには、Logic Builder 画面の右側にあるハードウェアまたはソフトウェアカタログでデバイスまたはオブジェクトを選択し、**ナビゲーター** (例えば、**デバイスツリー**) にドラッグ & ドロップします。選択されたデバイスまたはオブジェクトがあるノードは自動的に展開し太字で表示されます。選択したデバイスまたはオブジェクトが挿入できない他のノードはグレー表示されます。デバイスまたはオブジェクトを適当なノードにドロップすると、自動的に挿入されます。通信マネージャーなど、デバイスやオブジェクトにさらに必要なものがあれば、それらは自動的に挿入されます。

また、ツリーのノードを選択することもできます。選択したデバイスまたはオブジェクトにオブジェクトを追加することができる場合、緑色のプラスボタンが表示されます。このプラスボタンをクリックすると、挿入可能な要素を含むメニューが開きます。

オブジェクトやデバイスを追加するには、**ナビゲーター**でノードを右クリックし**オブジェクトの追加**または、**デバイスの追加**コマンドを実行してもできます。挿入できるデバイスタイプは**ナビゲーター**で選択されているオブジェクトに依存します。例えば、PROFIBUS DP スレーブのモジュールは、適切なスレーブデバイスを挿入する前に挿入することはできません。ローカルシステムに正しく設置され、ツリー内の位置と合っているデバイスにのみ挿入できることに注意してください。

オブジェクトの配置変更

オブジェクトの配置を変更するには、**編集**メニューのクリップボードコマンド (**切り取り、コピー、貼り付け、削除**) を使用します。または、マウスボタン (コピーの場合は、同時に CTRL キーも) を押しながら、選択したオブジェクトをマウスでドラッグすることもできます。コピーと貼り付け機能を使用してデバイスを追加すると、新しいデバイスに同じ名前が付けられ、名前の後の番号が増えていきます。

デバイスのバージョンの更新

ナビゲーターにすでに挿入されているデバイスは、バージョンアップしたり別のデバイスに変換することができます。

それぞれのコマンドの説明を参照してください。

- **デバイスの更新** コマンド (*62* ページ)
- **デバイスの変換** コマンド (*63* ページ)

デバイスツリーの説明

デバイスツリーの各オブジェクトは特定の (対象) ハードウェアを示します。

例 : コントローラー、フィールドバスノード、バスカップラー、ドライブ、I/O モジュール

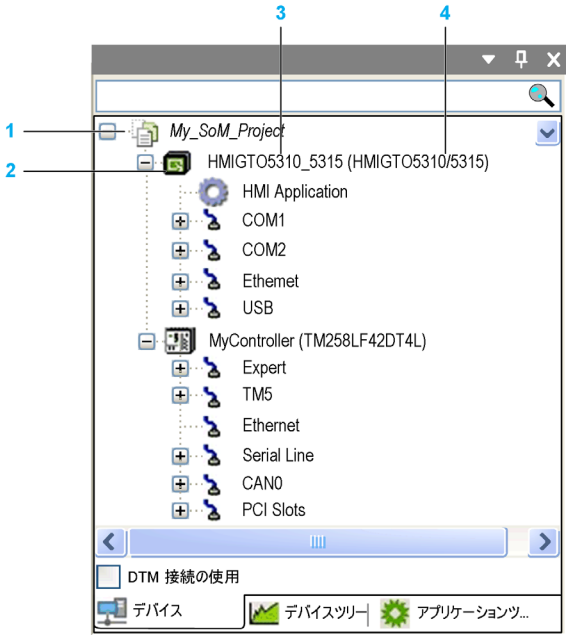
デバイスやサブデバイスは**デバイスツリー**で管理します。コントローラーで実行するアプリケーションに必要な他のオブジェクトは、別の**ナビゲーター**にグループ分けされます。

- ツリーのルートノードは次のアイコンです : < プロジェクト名 >
- コントローラーの構成は、**デバイスツリー**での機器の配置によって設定できます。特定のデバイスやタスクパラメーターの設定は、それぞれのエディターダイアログで行います。*タスク設定* (*192* ページ) の章も参照してください。

ハードウェア構成は、**デバイスツリー**内でデバイスオブジェクトの配置によって表示されるので、コントローラーやその下層にあるフィールドバスなどの複雑なネットワークシステムを設定できます。

- DTM (デバイスタイプマネージャー) で設定したデバイスをプロジェクトに追加するには、**デバイスツリー**の下部にある **DTM 接続の使用**チェックボックスを有効にします。するとツリーのルートノードに **FdtConnections** というノードが追加されます。**FdtConnections** ノードの下には接続マネージャーが自動的に挿入されます。このノードに適切な DTM デバイスを追加できます。
  詳細は、デバイスタイプマネージャー (DTM) ユーザーガイド *(EcoStruxure Machine Expert, Device Type Manager (DTM), User Guide 参照)* を参照してください。
- この章の *ナビゲーターに要素を追加*にある推奨事項を考慮してください。

デバイスツリーの例 :



1 ルートノード
2 アプリケーションでプログラミング可能なデバイス
3 シンボルデバイス名
4 デバイス記述ファイルで定義されたデバイス名

- **デバイスツリー**の各項目は、シンボル、シンボル名 (編集可) とデバイスタイプ (= デバイス記述で表示されたデバイス名) を表示します。
- デバイスはプログラミング可能または設定可能です。デバイスタイプによってツリーに追加できる位置や、挿入できるリソースが決まります。
- 製造元やタイプに関わらず、1 つのプロジェクトに複数のプログラミング可能なデバイスを設定できます (マルチリソース、マルチデバイス、ネットワーキング)。
- デバイスダイアログ (デバイスエディター) で、通信、パラメーター、I/O マッピングに関するデバイスを設定します。デバイスエディターを開くには、**デバイスツリー**のデバイスノードをダブルクリックします (デバイスエディター (*81* ページ) の説明を参照してください)。
- オンラインモードでは、デバイスステータスはデバイスエントリーのアイコンで表示されます。
  - コントローラーが接続され、アプリケーションを実行中、デバイスを動作中、データを送受信中です。デバイスエディターの **PLC 設定** (*110* ページ) ビューにある**停止中に IO を更新**オプションは切り替え可能です。
  - コントローラーが接続され停止 (STOP) しています。デバイスエディターの **PLC 設定** (*110* ページ) ビューにある**停止中に IO を更新**オプションは無効です。
  - コントローラーが接続され、有効なアプリケーションを実行中、診断情報が利用可能です。
  - デバイスは準備モードでまだ動作していません。診断情報が利用可能です。
  - デバイスはデータの送受信をしていません。バスエラーが検知されたか、設定されていない、またはシミュレーションモードです (**シミュレーション**コマンドの説明を参照してください)。
  - デバイスがデモモードで 30 分間動作中です。この時間を経過すると、デモモードが終了し、フィールドバスはデータの送受信を停止します。

-  デバイスは設定されていますが、使用可能な状態ではありません。データの送受信はされていません。例えば、CANopen デバイスは起動中および動作前の状態です。
-  待機状態：フィールドバスマスターは、他のマスターが稼働状態のため、データを送信していません。
-  デバイスがデバイスリポジトリーにありません。**デバイスリポジトリー**ダイアログボックスでのデバイスの配置または削除に関する詳細は、**デバイスリポジトリー** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。

- 接続されているデバイスとアプリケーションの名前は緑色で表示されます。
- シミュレーションモード (**シミュレーション**コマンドの説明を参照してください) で実行されているデバイス名は斜体で表示されます。
- 診断の追加情報は、デバイスエディターの **ステータス** ビュー (*120* ページ) に表示されます。

またアクティブなアプリケーションは、デフォルトで既に有効に設定されている、シミュレーションを使って実行できます。そのため、アプリケーションの実際の動きをテストするための実機は必要ありません (ハードウェアに依存しないアプリケーションに限る)。シミュレーションモード *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* に切り替えると、**デバイスツリー**の項目が斜体で表示され、アプリケーションにログインできます。

コントローラーに実際のアプリケーションをロードせずに、オンライン設定モード (*オンライン設定モード (EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の章を参照してください) でコントローラーに接続することもできます。これは、実際のアプリケーションプログラムをビルドしてロードする前にコントローラーの I/O にアクセスしテストできるため、I/O システムの初期起動に役立ちます。

プロジェクトを開くときのデバイス参照の変換については *Compatibility and Migration User Guide* を参照してください。

デバイスツリーでのオブジェクトの配置を設定

デバイス / オブジェクトの配置

デバイスまたはオブジェクトを**デバイスツリー**に追加するには、Logic Builder 画面の右側にあるハードウェアカタログのデバイスまたはオブジェクトを選択し、**デバイスツリー**にドラッグします。選択されたデバイスまたはオブジェクトがあるノードは展開し太字で表示されます。選択したデバイスまたはオブジェクトが挿入できない他のノードはグレー表示されます。デバイスまたはオブジェクトを適当なノードにドロップすると、自動的に挿入されます。

また、ツリーのノードを選択することもできます。選択したデバイスまたはオブジェクトにオブジェクトを追加することができる場合、緑色のプラスボタンが表示されます。プラスボタンをクリックすると、挿入可能な要素を表示するメニューが開きます。

あるいは、**デバイスツリー**でノードを右クリックし**オブジェクトの追加**または**デバイスの追加**コマンドを実行して、オブジェクトまたはデバイスを追加することもできます。挿入できるデバイスタイプは**デバイスツリー**で選択されているオブジェクトに依存します。例えば、PROFIBUS DP スレーブのモジュールは、スレーブデバイスを設置するまで追加できません。プログラミングが不可能なデバイスの下にアプリケーションを挿入することはできません。

ローカルシステムに正しくインストールされ、ツリー内の位置と合っているデバイスにのみ挿入できることに注意してください。

オブジェクトの配置変更

オブジェクトの配置を変更するには、**編集**メニューのクリップボードコマンド (**切り取り、コピー、貼り付け、削除**) を使用します。または、マウスボタン (コピーの場合は、同時に CTRL キーも) を押しながら、選択したオブジェクトをマウスで描画することもできます。**貼り付け** コマンドを使用する際、貼り付けするオブジェクトが、選択されている項目の下または上に挿入できる場合は、**貼り付け位置を選択**ダイアログボックスが開きます。これにより挿入位置を指定することができます。コピーと貼り付け機能を使用してデバイスを追加すると、新しいデバイスに同じ名前が付けられ、名前の後の番号が増えていきます。

デバイスのバージョンの更新

すでに**デバイスツリー**に挿入されているデバイスは、同じタイプの別のバージョン、または別のタイプのデバイス (デバイスの更新) に置き換えることができます。そうすることで、それぞれのデバイスの下にあるツリー設定を長期にわたって維持することができます。

ルートノードにデバイスを追加

デバイスのみをルートノード < プロジェクト名 > の下レベルに直接配置することができます。**オブジェクトの追加**ダイアログボックスから**テキストリスト**オブジェクトなどのオブジェクトを選択した場合、**アプリケーションツリー**の**グローバル**ノードに追加されます。

サブノード

デバイスはツリーにノードとして挿入されます。デバイス記述ファイルで定義されている場合はサブノードが自動的に挿入されます。プログラミングがサブノードはプログラミング可能なデバイスである可能性もあります。

デバイスオブジェクトの下にデバイスを挿入

デバイスオブジェクトの下にさらにデバイスを挿入することができます。ローカルシステムに設置されている場合は、ハードウェアカタログまたは、**オブジェクトの追加**または**デバイスの追加**ダイアログボックスに表示されます。デバイスオブジェクトはツリーの中で上から下にソートされます。ツリーの構造によっては最初にプログラミング可能なデバイスが配置され、次にアルファベット順にソートされたデバイスが続きます。

アプリケーションツリーの説明

アプリケーションオブジェクト、タスク設定、およびタスクオブジェクトは**アプリケーションツリー**で管理します。

デバイスのプログラミングに必要なオブジェクト (アプリケーション、テキストリストなど) は**アプリケーションツリー**で管理します。プログラミングが不可能なデバイス (設定のみ) はプログラミングオブジェクトとして指定できません。デバイスエディターのパラメーターダイアログでデバイスパラメーターの値を編集できます。

特定の POU やグローバル変数リストのようなプログラミングオブジェクトは、定義によって、**アプリケーションツリー**で二通りの管理ができます。

- **グローバル**ノードのサブノードとして定義されている場合は、すべてのデバイスからアクセスすることができます。
- **アプリケーション**ノードのサブノードとして定義されている場合は、この**アプリケーション**ノードで定義されているデバイスからのみアクセスすることができます。

アプリケーションオブジェクトは**アプリケーションツリー**にのみ挿入できます。

各アプリケーションには、**DUT**、**GVL** やビジュアライゼーションオブジェクトのような追加のプログラミングオブジェクトを挿入できます。アプリケーションの下にタスクを挿入します。このタスクでは、それぞれのプログラム呼び出しを定義する必要があります。(**アプリケーションツリー**の**グローバル**ノードやデバイス特有の POU からの POU のインスタンス)。アプリケーションは、それぞれのデバイスエディターの **I/O マッピング** ビューで定義されるとお考えください (*123* ページ)。

ツールツリーの説明

ライブラリーは**ツールツリー**で管理します。単に設定可能なデバイスには、そのようなプログラミングオブジェクトを割り当てることはできません。デバイスエディターのパラメーターダイアログでデバイスパラメーターの値を編集できます。

ライブラリーマネージャーのようなプログラミングオブジェクトは、定義によって、**ツールツリー**で二通りの管理ができます。

- **グローバル**ノードのサブノードとして定義されている場合は、すべてのデバイスからアクセスすることができます。
- **アプリケーション**ノードのサブノードとして定義されている場合は、この**アプリケーション**ノードで定義されているデバイスからのみアクセスすることができます。

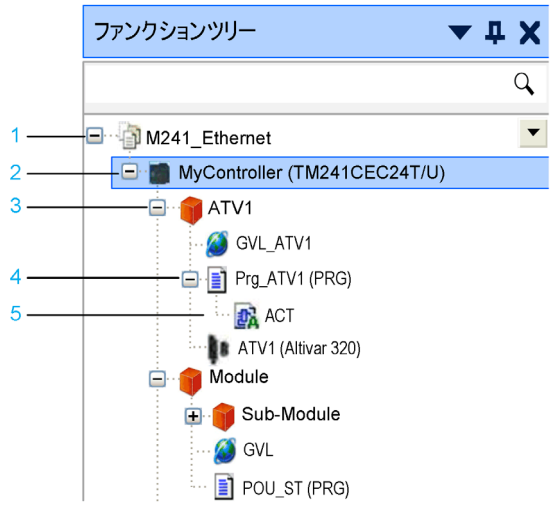
ファンクションツリー

概要

ファンクションツリーは、**デバイスツリー**に**ファンクションモデル**のノードがあるコントローラーでのみ使用できます。IEC コードやデバイスなど複数のオブジェクトをグループ化し、1 つのファンクションにリンクすることができます。一度作成すると、このファンクションを再利用できます。このようにモジュール化することで開発の再利用が容易になりプロジェクトの見方をを改善することができます。**ファンクションツリー**をエクスポート / インポートしたり他のプロジェクトで再利用することができます。

ファンクションツリーの説明

ファンクションツリーの例 :



1 ルートノード : 開いているプロジェクトの名前
2 コントローラーノード : **デバイスツリー**に**ファンクションモデル**のノードがあるコントローラーのみを表示
3 ファンクションモジュール : **ファンクションツリー**を構成する機能
4 搭載オブジェクト : ファンクションモジュールに含まれているオブジェクト
5 子オブジェクト : 含まれているオブジェクトの子オブジェクト

コントローラーの選択

次の手順で**ファンクションツリー** のコントローラーを選択します。

手順	手順内容	結果
1	**ファンクションツリー**でルートノードを右クリックし、**コントローラーの選択**コマンドを実行します。	**デバイスツリー**で選択されたコントローラー用に新規サブノード**ファンクションモデル**が挿入されます。
2	**コントローラーの選択**ダイアログボックスで**ファンクションツリー**に追加したいコントローラーを選択し **OK** をクリックします。	選択されたコントローラー用に、**ファンクションツリー**のルートノードの下に新規コントローラーのノードが追加されます。

ノードの追加

必要に応じて、コントローラーの内容をグループ化するために**ファンクションツリー** でコントローラーノードの下にサブノードを作成できます。

ノード	詳細	作成方法
ファンクションモジュール	ファンクションモジュールとはアプリケーションの機能を実行することを目的としたプログラム要素のグループです。 ファンクションモジュールノードは、**ファンクションツリー**で階層構造を作成します。わかりやすい構造を作成するには、デフォルト名を変更し各ファンクションモジュールに内容のわかる名前を割り当てます。	親ノード (例えばコントローラーノード) を選択して緑色のプラスボタンをクリックします。

ノード	詳細	作成方法
搭載オブジェクト	搭載オブジェクトはコントローラーのコンテンツを示す他のナビゲーター (**デバイスツリー**、**アプリケーションツリー**、**ツールツリー**) のノードです。 次の点に注意してください。 ● 1つのファンクションモジュールには1つのオブジェクトのみ含めることが可能です。 ● 同じコントローラーのファンクションモジュールにのみオブジェクトを含めることが可能です。 ● ファンクションテンプレート (*723* ページ) で許可されているオブジェクトのみ含めることが可能です。	ファンクションモジュールノードを右クリックし、コンテキストメニューから**オブジェクトの選択**コマンドを実行します。**オブジェクトの選択**ダイアログボックスぁら添付したいノードを選択し **OK** をクリックします。
子オブジェクト	搭載オブジェクトの子オブジェクト	子オブジェクトは**ファンクションツリー**で表示されます。

ノードの削除

ファンクションツリーからノードを削除するには、右クリックのコンテキストメニューから**削除**コマンドを実行します。選択したオブジェクトに対して、オブジェクトのみ削除するか、子オブジェクトも一緒に削除、**ファンクションツリー**からのみ削除、プロジェクト全体から削除するかを尋ねられます。

子オブジェクトは**ファンクションツリー**からのみ削除することはできません。子オブジェクトを削除する場合、子オブジェクトを削除しようとすると、オブジェクトがプロジェクト全体から削除されることを確認するメッセージが表示されます。

ファンクションモジュールの再利用

同じプロジェクトまたは別のプロジェクトで再利用をするファンクションモジュールを作成した場合は、含まれているオブジェクト間の依存関係を解決できるファンクションテンプレートの使用を推奨します。**インポート / エクスポート**コマンドとコピー / 貼り付け機能も使用できますが、以下のセクションで説明するように特別なケースでのみ役に立ちます。

ファンクションテンプレートを使用したファンクションモジュールの再利用

ノードを右クリックし、コンテキストメニューから**ファンクションテンプレートとして保存**コマンドを実行するとファンクションモジュールをファンクションテンプレートとして保存できます。

ファンクションテンプレートからファンクションモジュールをインスタンス化するには、**ファンクションツリー**でノードを右クリックし、**テンプレートからファンクションを追加**コマンドを実行します。

詳細については、*ファンクションテンプレートの管理*の章 (*718* ページ) を参照してください。

インポート / エクスポートコマンドを使用したファンクションモジュールの再利用

ファンクションモジュールの再利用で**プロジェクト → エクスポート** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* および **プロジェクト → インポート** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用する際、次の点に注意してください。

条件	結果
コントローラーデバイス全体をエクスポートし、それを同じまたは別のプロジェクトにインポートする場合	ファンクションモデルが再作成されます。
ファンクションモデルのみをエクスポートしインポートする場合	搭載オブジェクトは再作成されません。

コピーと貼り付け機能を使用したファンクションモジュールの再利用

ファンクションモジュールの再利用でコピー貼り付け機能を使用する際、次の点に注意してください。

条件	結果
完全なコントローラーデバイスをコピーし、それを同じまたは別のプロジェクトに貼り付ける場合	ファンクションモデルが再作成されます。
ひとつまたは複数のファクションモジュールをコピー貼り付けする場合	添付オブジェクトは再作成されません。

ファンクションツリーで添付オブジェクトをコピー貼り付けすることはできません。

マルチタブカタログビュー

概略

マルチタブ**ハードウェアカタログ**は、Logic Builder 画面のデフォルトコンポーネントです。

以下のタブが含まれます。

- **コントローラー** : EcoStruxure Machine Expert プロジェクトに挿入可能な**ロジック**、**HMI**、および**モーション**コントローラーを含みます。
- **デバイス & モジュール** : EcoStruxure Machine Expert プロジェクトに挿入可能な **PLC コンポーネント**、**I/O モジュール**、および**通信**、**モーター制御**、**セーフティー**、および**センサー**デバイスを含みます。デバイステンプレートを使用してデバイスを挿入することもできます。
- **HMI & iPC**: EcoStruxure Machine Expert プロジェクトに挿入可能な **HMI** および **iPC** デバイスを含みます。
- **その他** : EcoStruxure Machine Expert プロジェクトに挿入可能なサードパーティーのデバイスを含みます。

個々のタブの内容はプロジェクトにによって異なります。EcoStruxure Machine Expert プロジェクトに統合されたコントローラーが、例えば、CANopen に対応していない場合は、CANopen デバイスはカタログに表示されません。

メニューの**表示 → ソフトウェアカタログ**から、**ソフトウェアカタログ**のタブ (**変数**、**アセット**、**マクロ**、**ツールボックス**、**ライブラリー**) でマルチタブカタログビューを拡張できます。

ツールバーの**ハードウェアカタログ**  および**ソフトウェアカタログ**  ボタンでカタログビューを表示または非表示にできます。

*ドラッグ & ドロップでデバイスを追加*の章 (*52* ページ) で説明されているように、要素をカタログからプロジェクトに簡単にドラッグ & ドロップできます。

カタログ内の検索

カタログビューの各タブには検索ボックスがあります。タブのサブリストで検索ボックスに入力した文字列が分析されます。開いているサブリストではエントリーが黄色にマークされます。検索文字列に対応しない項目は表示されません。閉じているサブリスト内で見つかった項目数は各サブリストのタイトルバーに太字で表示されます。

デフォルトでは、リスト内の項目の名前に対して検索が実行されます。EcoStruxure Machine Expert では、タグ付けの仕組みもサポートしています。検索文字列を**カタログ**ビュー内のどの項目にも割り当てることができます。

お気に入りリスト

カタログビューの各タブには**お気に入り**リストがあります。すばやくアクセスできるように、頻繁に使用する要素をドラッグ & ドロップでこの**お気に入り**リストに追加することができます。

デバイス & モジュールタブのデバイステンプレートからデバイスを追加

デバイス & モジュールタブの下部に**デバイステンプレート**オプションがあります。**デバイス & モジュール**タブのリストにフィールドデバイスのテンプレートを表示するには、このオプションを有効にします。*テンプレートからデバイスを追加*の章 (*708* ページ) の説明のように**デバイスツリー**に追加します。

ユーザーインターフェイスのカスタマイズ

概要

特定の構成要素の配置と設定に関するユーザーインターフェイスの外観は、次の項目によって異なります。

- メニュー、キーボード機能、およびツールバーのあらかじめセットされたデフォルト設定。EcoStruxure Machine Expert の初期設定は**カスタマイズ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* (デフォルトでは**ツール**メニューにあります) から上書きできます。現在の設定はローカルシステムに保存されます。リセット機能でいつでもデフォルト値に戻すことができます。
- それぞれの**ツール → オプション** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で定義されたエディターのプロパティ。これらを上書きすることも可能です。現在の設定はローカルシステムに保存されます。
- プロジェクト内のビューやエディターウィンドウの配置位置。現在の位置はプロジェクトと共に保存されます (下記参照)。
- 選択したパースペクティブデフォルトで**ロジック設定**パースペクティブが選択されています。詳細については、この章の*パースペクティブ*(*41* ページ) の項を参照してください。

メニューバーとツールバーの配置

メニューバーはユーザーインターフェイスの上部、ウィンドウタイトルバーとビューウィンドウの間に配置されています。ツールバーをメニューバー (固定) と同じ領域に配置できます。また、独立したウィンドウとして画面上どこにでも配置できます。

デバイスツリーなどのビューウィンドウでは、特別なツールバーが利用できます。ウィンドウ内での並べ替え、表示、検索が可能です。このツールバーは変更できません。

ウィンドウとビューの配置

ビューまたはエディターウィンドウを閉じる : 右上のクロスボタンをクリックします。

閉じているビューを開く : デフォルトで**表示**メニューからスタンダードコンポーネントのビューを開くことができます。エディターウィンドウを開くには、**プロジェクト → エディターオブジェクト**コマンドを実行するか**デバイスツリー**、**アプリケーションツリー**、または**ツールツリー**それぞれの項目をダブルクリックします。

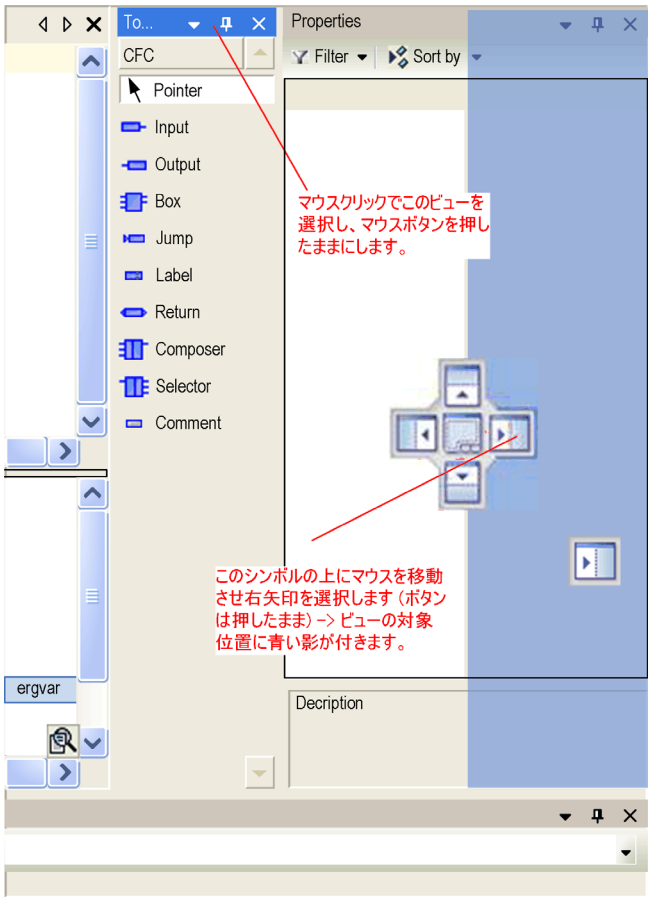
フレームウィンドウ内でビューまたはウィンドウのサイズ変更をする : ビューとビューの間の区切り線を移動します。ウィンドウの枠を移動することでデスクトップ上の独立したビューウィンドウのサイズを変更することができます。

ビューをデスクトップ上またはフレームウィンドウ内の別の位置に移動する : タイトルバー、または、タブ付きビューの場合はビューのタブをクリックし、マウスボタンを押したままビューを移動したい場所に移動します。矢印のシンボルは移動可能な場所を示します。マウスボタンを押したまま、矢印シンボル上にカーソルを移動して位置を選択します。移動先は青い影として表示されます。

新しい位置を示す矢印シンボル

矢印シンボル	詳細
	ビューは上に配置されます。
	ビューは下に配置されます。
	ビューは右に配置されます。
	ビューは左に配置されます。
	ビューはここに配置されます。ビューは現在この位置に配置されていて新しいビューはアイコンとして配置されます。

矢印シンボルによるナビゲーションの例



マウスボタンを離すと、ビューは新しい位置に配置されます。

Auto Hide ボタンのあるビューは、画面上の任意の場所に独立したウィンドウ (フローティング) として配置することができます。矢印シンボル上にドラッグしないようにビューを移動させます。この場合、ビューから **Auto Hide** ボタンはなくなります。代わりに**ウィンドウ**メニューから**ドック** と**フロート**コマンドを実行します。

ビューを非表示にする : EcoStruxure Machine Expert ウィンドウの境界線にある **Auto Hide** ボタンでビューを非表示にすることができます。ビューの右上隅にある **Auto Hide** のダウンボタンをクリックします。ビューは、最も近いフレームウィンドウの境界線にタブとして表示されます。ビューの内容は、カーソルがこのタブ上で動かされている場合にのみ表示されます。タブには、アイコンとビューの名前が表示されます。ビューのこの状態は、ドッキングボタンが **Auto Hide** に変更されたことで示されます。

ビューを表示する : 非表示だったビューを表示するには **Auto Hide** ボタンをクリックします。

ビューを表示 / 非表示にする別の方法は、デフォルトで利用可能な**ウィンドウ**メニューにある **Auto Hide** コマンドです。

ユーザーインターフェイス下部の境界にある情報とステータスバー (*29* ページ) の位置を変更することはできません。

パースペクティブ

パースペクティブは EcoStruxure Machine Expert ビューのレイアウトの保存に使用します。**メッセージ**ビューと**ウォッチ**ビューが開いているかどうか、どの位置で開いているか (ドッキングまたは独立したウィンドウ) を保存します。

デフォルトで EcoStruxure Machine Expert では、**ウィンドウ → パースペクティブの切り替え**メニュー、またはツールバーのパースペクティブテーブルで、以下の使用例に対する 4 つのパースペクティブを提供しています。

パースペクティブ名	使用例	ナビゲーター (左側)	カタログビュー (右側)	画面下部のビュー
デバイス設定	デバイスの追加、設定	● **デバイスツリー** ● **アプリケーションツリー** ● **ツールツリー**	ハードウェアカタログ ● **コントローラー** ● **デバイス & モジュール** ● **HMI & iPC** ● **その他**	**メッセージ** (**Auto Hide** モード)
ロジック設定	デバイスの追加、設定	● **デバイスツリー** ● **アプリケーションツリー** ● **ツールツリー**	ソフトウェアカタログ ● **変数** ● **アセット** ● **マクロ** ● **ツールバー** ● **ライブラリー**	**メッセージ** (**Auto Hide** モード)
CODESYS クラシック	スタンダード CoDeSys ビュー	● **デバイス** ● **POU**	ハードウェアカタログ ● **コントローラー** ● **デバイス & モジュール** ● **HMI & iPC** ● **その他**	**メッセージ** (**Auto Hide** モード)
オンライン	オンラインモード	● **デバイスツリー** ● **アプリケーションツリー** ● **ツールツリー**	ハードウェアカタログ ● **コントローラー** ● **デバイス & モジュール** ● **HMI & iPC** ● **その他**	● **メッセージ** (**Auto Hide** モード) ● **ウォッチ 1**

アプリケーションがオンラインに切り替わると、自動的に**オンライン**パースペクティブが選択されます。

カスタムパースペクティブを作成

これらのデフォルトパースペクティブに加え、必要に応じて独自のビューレイアウトを作成しパースペクティブとして保存することができます。

次の手順でカスタムパースペクティブを作成します。

手順	手順内容
1	個々の要件に応じてビューをリサイズしたり、開いたり、閉じたりします。
2	**ウィンドウ**メニューから **パースペクティブの保存** コマンドを実行し、レイアウトを新しいパースペクティブに保存します。
3	**パースペクティブの保存** ダイアログボックスでパースペクティブの名前を入力します。 **結果** : 現在のビューレイアウトが保存されます。この新しいパースペクティブは**ウィンドウ → パースペクティブの切り替え** メニューとツールバーのパースペクティブテーブルから使用できます。

パースペクティブを初期状態にリセット

パースペクティブを初期状態にリセットするには、**ウィンドウ**メニューの**現在のパースペクティブのリセット**コマンドを実行します。

パースペクティブのインポートとエクスポート

異なる場所にインストールされた EcoStruxure Machine Expert 間、または異なるユーザー間でパースペクティブを使用できるようにするには、**ツール → オプション → パースペクティブ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* でパースペクティブを XML ファイルにエクスポート、または既存の XML ファイルをインポートします。

ズーム

各エディターウィンドウにはズーム機能があります。ウィンドウの右端にあるズームボタン  をクリックしてリストを開きます。ズームレベルの 25、50、100、150、200、400% から選択するか、ズーム率を入力します。プリントアウトは常に 100% 表示です。

ユーザーインターフェイスのカスタマイズは、オフラインおよびオンラインモードで可能です。

オンラインモードのユーザーインターフェイス

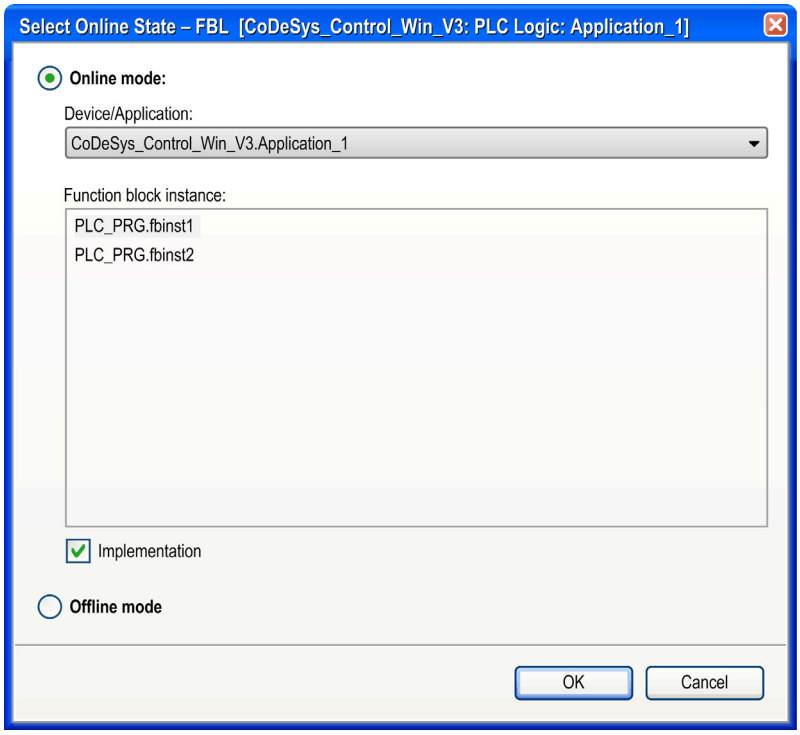
概要

プロジェクトにログインすると、オフラインモードで既に開いているオブジェクトは、オンラインモードで自動的に表示されます。パースペクティブは自動的に**オンライン** パースペクティブ (*41* ページ) に切り替わり、デフォルトで**ウォッチ**ビューが開きます。

まだ開いていないオブジェクトをオンラインモードで開くには、**アプリケーションツリー**のノードをダブルクリックするか、**プロジェクト → オブジェクトの編集**コマンドを実行します。オンラインモードでオブジェクトが開かれます。

選択されたオブジェクト (ファンクションブロックなど) のインスタンスがプロジェクトに複数ある場合は、**Select Online State < オブジェクト名 >** というダイアログボックスが表示されます。オブジェクトのインスタンスまたは基本実装を表示するかどうか、オブジェクトをオンラインモードまたはオフラインモードで表示するかどうかを選択できます。

Select Online State ダイアログボックス



デバイス / アプリケーションフィールドには、それぞれのオブジェクトが関連付けられているデバイスとアプリケーションが表示されます。

オブジェクトのオンラインビューを開くには**オンラインモード**オプションを有効にし **OK** をクリックします。オフラインビューを開くには**オフラインモード**オプションを有効にます。

オブジェクトがファンクションブロックの場合は**ファンクションブロックインスタンス**フィールドには現在アプリケーションで使用されているインスタンスのリストが表示されます。

この場合に使用できるオプションは次のとおりです。

- インスタンスの 1 つを選択し**オンライン**または**オフラインモード**を有効にします。
- または、選択されたインスタンスとは独立した **Implementation** オプションを選択すると、ファンクションブロックの基本実装ビューが開きます。**Implementation** オプションはインスタンス化されていないオブジェクトには影響しません。

特定のエディターのオンラインビューの詳細については、それぞれのエディターの説明を参照してください。

ステータスバー (*29* ページ) は、アプリケーションの現在のステータスに関する情報を表示します。

メニューとコマンド

概要

次の図はデフォルトのメニューバーです。

いくつかのコマンドはデフォルトビューでは非表示になっています。メニューにコマンドを追加するには、**ツール → カスタマイズ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で選択しメニューに挿入します。

特定のエディター専用などの特別なコマンドは、通常、そのコマンドに対応するメニューにあります。それらのコマンドは、そのエディターを開いたときのみに表示されます。例えば、SFC エディターのオブジェクトを編集するときに、**SFC** メニューがメニューバーに追加されます。

メニュー構造を編成するには、**ツール → カスタマイズ**ダイアログボックスを使用します。

メニューとコマンドの詳細については、別の EcoStruxure Machine Expert メニューコマンドオンラインヘルプ *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を参照してください。

第 3 章
基本概念

概要と基本概念

概要

EcoStruxure Machine Expert はデバイスに依存しないコントローラープログラミングシステムです。

IEC 61131-3 規格に準拠し、すべての標準プログラミング言語に対応しています。

オブジェクト指向

オブジェクト指向のアプローチは、適切なプログラミング要素や機能を使用可能とするだけではなく、EcoStruxure Machine Expert やそのプロジェクト内での構成やバージョン管理にも反映されています。共同に使用され、インスタンス化されたプログラミングユニットにより、EcoStruxure Machine Expert プロジェクトでの複数デバイスの使用ができます。

バージョン管理

複数のバージョンの EcoStruxure Machine Expert インストールし、必要なバージョンの組み合わせで作業をすることが可能です。これは、デバイス固有の異なるコンパイラーバージョンの使用に適しています。全体のバージョンを更新することなく個々の機能を追加することができます。

詳細は、*Compatibility and Migration User Guide* を参照してください。

プロジェクト構成

プロジェクト構成もオブジェクト指向です。EcoStruxure Machine Expert プロジェクトはさまざまなプログラミングオブジェクトからなるコントローラープログラムを持ち、そこには定義された対象システム (デバイス、コントローラー) でプログラム (アプリケーション) のインスタンスを実行するために必要なリソースの定義が含まれています。

プロジェクトには 2 種類のメインオブジェクトがあります。

オブジェクトタイプ	詳細
プログラミングオブジェクト (POU) (*135* ページ)	プログラム、ファンクション、ファンクションブロック、メソッド、インターフェイス、アクション、データ型、定義などが含まれます。
リソースオブジェクト (**デバイスツリー**)	デバイスオブジェクトは**デバイスツリー**で管理されます。**デバイスツリー**にオブジェクトを挿入する際には、*ナビゲーターに要素を追加*セクションにある推奨事項を考慮してください (*35* ページ)。

コード生成

統合されたコンパイラーによるコード生成と、それに続くマシンコードの使用で短時間での実行が実現されます。

コントローラーデバイスへのデータ転送

EcoStruxure Machine Expert とデバイス間のデータ転送は、ゲートウェイ (コンポーネント) を介してランタイムシステムへ行われます。アプリケーションからコントローラーにダウンロード後 EcoStruxure Machine Expert でモニターおよび制御が可能です。

サポートされているプログラミング言語

IEC 規格 IEC 61131 で言及されているプログラミング言語は、特別に採用されたエディターでサポートされています。

- FBD/LD/IL エディター (*231* ページ): ファンクションブロックダイアグラム (FBD)、ラダーロジックダイアグラム (LD)、命令リスト (IL)
- SFC エディター (*297* ページ): シーケンシャル ファンクション チャート
- ST エディター (*321* ページ): 構造化テキスト

さらに、EcoStruxure Machine Expert では IEC 規格の一部ではない CFC のプログラミング用エディターも提供しています。

- CFC エディター (*279* ページ): 連続ファンクションチャート

CFC は標準 IEC プログラミング言語の拡張版です。

CFC (ページ指向のエディター) も提供されています。個々のページに CFC 要素を配置することができます。

第 II 部
設定

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
4	デバイスの設置	49
5	デバイスの管理	51
6	共通デバイスエディターのダイアログ	79

第 4 章
デバイスの設置

他社製 Sercos デバイスの設置

概要

デバイスリポジトリー ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使うと、汎用 I/O プロファイルを持つ Sercos デバイスをプログラミングシステムに取り込めます。

この Sercos デバイスを設置するには、デバイスのベンダーが提供する SDDML (Sercos Device Description Markup Language) ファイル (Sercos デバイスのデバイス記述ファイル) が必要です。SDDML ファイルは Sercos デバイスのデバイス記述ファイルです。

Sercos デバイスには、2 種類の汎用 I/O プロファイルをもつ 2 種類のデバイスがあります。

- ブロック I/O デバイス
  ブロック I/O デバイスはバスインターフェイスと I/O モジュールで構成されるあらかじめ組み立てられたブロックです。
- モジュラー I/O デバイス
  モジュラー I/O デバイスはバスインターフェイスに接続可能な I/O モジュールです。

EcoStruxure Machine Expert への統合

次の手順に従って、他社製の Sercos デバイスをプログラミングシステムに設置します。

手順	手順内容
1	メニューバーから、**ツール → デバイスリポジトリ ...** を選択します。 **結果** : **デバイスリポジトリー**ダイアログボックスが開きます。
2	**デバイスリポジトリー** ダイアログボックスで**インストール ...** ボタンをクリックします。 **結果** : **Install Device Description** ダイアログボックスが開きます。
3	ファイルタイプに **SERCOS III I/O デバイス記述 (*.xml)** を選択して SDDML ファイルを検索します。
4	SDDML ファイルを選択し**開く**をクリックします。 **結果** : SDDML ファイルは EcoStruxure Machine Expert 互換ファイルフォーマットに変換されインポートされます。

注記：選択した SDDML ファイルに互換性がない、またはサードバーティーベンダーの Sercos デバイスが互換性のある FSP (Function Specific Profile) タイプを使用していない場合は、それに対応する診断メッセージが**メッセージ** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* に表示されます。

設置完了の確認

次の手順に従って、汎用 I/O プロファイル付き Sercos デバイスがプログラミングシステムに統合できたかどうかを確認します。

手順	手順内容
1	メニューバーから、**ツール → デバイスリポジトリ ...** を選択します。 **結果** : **デバイスリポジトリー**ダイアログボックスが開きます。
2	ツリー構造の **設置済みデバイス記述**で、**Fieldbusses → Sercos** ノードを展開します。
3	**スレーブ**サブノードを展開し、統合した Sercos バスインターフェイスがリストに表示されることを確認します。
4	**Module** サブノードを展開し、統合した Sercos I/O モジュールがリストに表示されることを確認します。

詳細については、**デバイスリポジトリー** ダイアログボックスの説明を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

第 5 章
デバイスの管理

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
5.1	ドラッグ & ドロップによるデバイスの追加	52
5.2	コンテキストメニューまたはプラスボタンによるデバイスの追加	54
5.3	デバイスの更新	62
5.4	デバイスの変換	63
5.5	プロジェクトの変換	65

5.1
ドラッグ & ドロップによるデバイスの追加

ドラッグ & ドロップによるデバイスの追加

概要

EcoStruxure Machine Expert は、EcoStruxure Machine Expert Logic Builder の右側にマルチタブカタログビューが表示されます。

2 種類のカタログビューがあります。

- **ハードウェアカタログ**
- **ソフトウェアカタログ**

デバイスツリーにデバイスを追加するには、**ハードウェアカタログ**でその項目を選択し、**デバイスツリー**にドラッグし適切なノードにドロップします。プロジェクトに自動的に追加されます。

ドラッグ & ドロップでコントローラーを追加

次の手順に従ってプロジェクトにコントローラーを追加します。

手順	手順内容
1	**ハードウェアカタログ**が開いていない場合は、EcoStruxure Machine Expert Logic Builder ツールバーの**ハードウェアカタログ**ボタン  をクリックして開きます。
2	**ハードウェアカタログ**で**コントローラ**タブを選択します。 **結果** : EcoStruxure Machine Expert プロジェクトに適切なコントローラーが**ハードウェアカタログ**に表示されます。
3	**コントローラー**タブでコントローラーを選択し、**デバイスツリー**にドラッグして適切なノードにドロップします。**デバイスツリー**内で何もないスペースどこにでもドロップできます。 **結果 デバイスツリー** にコントローラーが新規ノードとして追加されます。コントローラーのタイプによって異なるサブノードも一緒に追加されます。

ドラッグ & ドロップによる拡張デバイスの追加

次の手順に従ってコントローラーに拡張デバイスを追加します。

手順	手順内容
1	**ハードウェアカタログ**が開いていない場合は、EcoStruxure Machine Expert Logic Builder ツールバーの**ハードウェアカタログ**ボタン  をクリックして開きます。
2	**ハードウェアカタログ**の**デバイス & モジュール**タブを選択します。 **結果** : EcoStruxure Machine Expert プロジェクトに適切な拡張デバイスが**ハードウェアカタログ**に表示されます。
3	拡張デバイスを選択出力、**デバイスツリー**にドラッグして適切なサブノードにドロップします。 **注記：** EcoStruxure Machine Expert に適切なサブノートが開き、強調表示されます。 **結果：**拡張デバイスは、**デバイスツリー**にあるコントローラーのサブノードの下に追加されます。
4	拡張デバイスに通信マネージャーが必要な場合は、そのノードが自動的に**デバイスツリー**に追加されます。 拡張デバイスに使用できる複数の通信マネージャーがある場合は、ダイアログボックスが表示されるので適切な通信マネージャーを選択します。

ドラッグ & ドロップによるデバイスとモジュールの追加

次の手順に従ってコントローラーにフィールドデバイスを追加します。

手順	手順内容
1	**ハードウェアカタログ**が開いていない場合は、EcoStruxure Machine Expert Logic Builder ツールバーの**ハードウェアカタログ**ボタン  をクリックして開きます。
2	**ハードウェアカタログ**の**デバイス & モジュール**タブを選択します。 **結果** : EcoStruxure Machine Expert プロジェクトに適切なフィールドデバイスが**ハードウェアカタログ**に表示されます。
3	**デバイス & モジュール** カタログビューでフィールドデバイスを選択し、**デバイスツリー**にドラッグして適切なコントローラーのサブノードにドロップします。 **注記：** EcoStruxure Machine Expert に適切なサブノートが開き、強調表示されます。 **結果：** フィールドデバイスは、**デバイスツリー**にあるコントローラーのサブノードの下に追加されます。
4	フィールドデバイスに通信マネージャーが必要な場合は、そのノードが自動的に**デバイスツリー**に追加されます。 フィールドデバイスに使用できる複数の通信マネージャーがある場合は、ダイアログボックスが表示されるので適切な通信マネージャーを選択します。

ドラッグ & ドロップでデバイスをデバイステンプレートから追加

次の手順に従ってデバイステンプレートからデバイスを追加します。

手順	手順内容
1	**ハードウェアカタログ**が開いていない場合は、EcoStruxure Machine Expert Logic Builder ツールバーの**ハードウェアカタログ**ボタン  をクリックして開きます。
2	**ハードウェアカタログ**の**デバイス & モジュール**タブを選択します。
3	**デバイス & モジュール**タブの下部にある**デバイステンプレート**オプションを選択します。 **結果** : EcoStruxure Machine Expert プロジェクトに適切なテンプレートが**デバイス & モジュール**タブに表示されます。
4	*テンプレートからデバイスを追加*の章 (*708* ページ) の説明のように**デバイスツリー**に追加します。

ドラッグ & ドロップでデバイスをファンクションテンプレートから追加

次の手順に従ってファンクションテンプレートからデバイスを追加します。

手順	手順内容
1	ソフトウェアカタログが開いていない場合は、EcoStruxure Machine Expert Logic Builder ツールバーの**ソフトウェアカタログ**ボタン  をクリックしてソフトウェアカタログを開きます。
2	**ソフトウェアカタログ**の**マクロ**タブを選択します。 **結果** : EcoStruxure Machine Expert で使用できるファンクションテンプレートが**ソフトウェアカタログ**に表示されます。
3	**マクロ**ビューでファンクションテンプレートを選択し、**デバイスツリー**にドラッグして適切なサブノードにドロップします。 **注記：** EcoStruxure Machine Expert に適切なサブノートが開き、強調表示されます。 **結果：** ファンクションテンプレートに基づくデバイスが**デバイスツリー**に追加されます。

5.2
コンテキストメニューまたはプラスボタンによるデバイスの追加

このセクションについて

このセクションには次の項目が含まれています。

コントローラーの追加

概要

デバイスツリーにデバイスをドラッグ & ドロップする代わりに、**ツリー**の適切なノードに表示される緑色のプラスボタンをクリックします。または、**ツリー**のノードを右クリックしてコンテキストメニューから適切なデバイスを追加することもできます。**デバイスの追加**ダイアログボックスが開き、選択したノードにデバイスを追加、挿入、またはプラグインするかどうかを決定できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

コントローラーをプロジェクトに追加すると、**デバイスツリー**にいくつかのノードが自動的に追加されます。これらのサブノードは、コントローラーが提供する機能に応じたコントローラー固有のものです。

次の項では、コントローラーを追加する一般的な手順について説明します。特定のコントローラーの詳細については、そのコントローラーのプログラミングマニュアルを参照してください。

コントローラーの追加

次の手順に従って EcoStruxure Machine Expert プロジェクトにデバイスを追加します。

手順	手順内容
1	プロジェクトノードを選択し右クリックします。コンテキストメニューから**デバイスの追加 ...** コマンドを実行します。 **結果** : **デバイスの追加**ダイアログボックスが開きます。
2	**デバイスの追加**ダイアログボックスでの**メーカー**リストボックスから **Schneider Electric** を選択します。
3	プロジェクトに挿入したいコントローラーを選択します。
4	**名前**テキストボックスにデバイス名を入力して名前を変更します。 **注記：**IEC 規格に準拠した名前を選択してください。名前には特殊文字、数字から始まる名前、スペースを使用しないでください。名前は、32 文字以下にしてください。デバイス名を変更しない場合、初期設定の名前になります。 デバイスに意味のある名前をつけることで、プロジェクトの構成が容易になります。
5	**デバイスの追加**ボタンをクリックします。 **結果 :** 選択したコントローラーがプロジェクトに追加され**デバイスツリー**の新しいノードとして表示されます。**デバイスの追加**ダイアログボックスは開いたままです。次の操作を実行できます。 ● コントローラーをもう 1 台追加するには手順 3 に戻ります。 ● **閉じる**ボタンをクリックして **デバイスの追加**ダイアログボックスを閉じます。

拡張デバイスの追加

使用可能な拡張デバイス

それぞれのコントローラーで使用可能な拡張デバイスのリストは、EcoStruxure Machine Expert *概要* ドキュメントの*対応デバイス* の章を参照してください。

⚠ 警告

装置の意図しない動作

- 本装置には、シュナイダーエレクトリック認定のソフトウェアのみ使用してください。
- ハードウェアの設定を変更した場合は、必ずアプリケーションプログラムも更新してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

拡張デバイスの追加

次の手順に従ってデバイスに拡張デバイスを追加します。

手順	手順内容
1	コントローラーのノードを選択し緑色のプラスボタンをクリックするか、コントローラーのノードを右クリックしコンテキストメニューから**デバイスの追加 ...** コマンドを実行します。 **結果** : **デバイスの追加**ダイアログボックスが開きます。
2	**デバイスの追加**ダイアログボックスで、**メーカー**リストから **Schneider Electric** を選択します。
3	下の**デバイス**リストからコントローラーに追加したい拡張デバイスを選択します。
4	**名前**テキストボックスに拡張デバイス名を入力して名前を変更します。 **注記：**名前にスペースを入れないでください。拡張デバイス名の変更をしない場合、デフォルトで名前が与えられます。 拡張デバイスに意味を持つ名前をつけると、プロジェクトの構成がわかりやすくなります。
5	**デバイスの追加**ボタンをクリックします。 **結果** : 選択した拡張デバイスがプロジェクトに追加され、**デバイスツリー**でコントローラーの新しいサブノードとして表示されます。 **デバイスの追加**ダイアログボックスは開いたままです。次の操作を実行できます。 ● 拡張デバイスをさらに追加するには手順 3 に戻ります。 ● **閉じる**ボタンをクリックします。

注記：TWDNOI10M3 オブジェクト (AS-Interface Master Module) を追加すると、それに対応する **仮想 AS インターフェイスバス**フィールドバスマネージャーが自動的に挿入されます。

拡張デバイスの設定

詳細設定については、拡張デバイスの *プログラミングガイド*を参照してください。

通信マネージャーの追加

概要

通信マネージャーは、CANopen やシリアルラインなどのハードウェアバスインターフェイスを有効にしたり設定するために必要です。

2 種類の通信マネージャーがあります。

- フィールドバスマネージャー: フィールドバスデバイス (例 : CANopen スレーブ、Modbus スレーブ) の設定
- 一般通信マネージャー

以下は EcoStruxure Machine Expert で使用可能な通信マネージャーのリストです。

名前	インターフェイスの種類	詳細
ASCII マネージャー	シリアルライン	単純なデバイスでデータを送受信します。
Machine Expert-ネットワークマネージャー	● シリアルライン (最大 1 点) ● Ethernet (最大 3 点)	HMI とのデータ送受信用。
Modbus IOScanner	シリアルライン	Modbus RTU または ASCII プロトコルマネージャー、Modbus スレーブデバイスとの黙示的送受信 (I/O スキャン) を定義するために使用します。
Modbus マネージャー	シリアルライン	マスターまたはスレーブモードで Modbus RTU または用に使用します。
CANopen パフォーマンス	CAN	CANopen マネージャー、パフォーマンスコントローラー (M241、M251、M258 および LMC058) 用。
CANopen_Manager	CAN	CANopen マネージャー、PacDrive コントローラー用。
CANmotion	CAN	CANmotion マネージャー、Modicon LMC058 モーションコントローラー CAN1 ポート用のみ。
J1939_Manager	CAN	M241 / M251 コントローラーの J1939 スレーブ (Electronic Control Units (ECU)) 追加用。
Modbus TCP スレーブデバイス	Ethernet	Modbus TCP マネージャー、Ethernet ポート付きコントローラー用。
EthernetIP	Ethernet	EtherNet/IP マネージャー、Ethernet ポート付きコントローラー (M251、M258、および LMC058) 用。
Industrial Ethernet Network	Ethernet	Ethernet ポート付き M241 / M251 コントローラーの EtherNet/IP および Modbus TCP スキャナーサービス設定用。

通信マネージャーの追加

通信マネージャーはそれぞれのデバイスに自動的に追加されます。

次の手順に従って通信マネージャーを追加します。

手順	手順内容
1	**デバイスツリー**でバスインターフェイス (**シリアルライン**、**CANopen bus**、**CANbus**、**Ethernet**) を選択し、ノードの緑色のプラスボタンをクリックする、またはバスインターフェイスのノードを右クリックしてコンテキストメニューから**デバイスの追加**コマンドを実行します。 **結果** : **デバイスの追加**ダイアログボックスが開きます。
2	**デバイスの追加**ダイアログボックスでの**メーカー**リストから **Schneider Electric** を選択します。 **注記** : **メーカー**リストをクリックして、デバイスをブランド別にフィルターできます。
3	リストから**通信マネージャー**を選択します。
4	**名前**ボックスに名前を入力してデバイス名を変更します。 **注記** : 名前にはスペースを使用しないでください。デバイス名を変更しない場合、初期設定の名前になります。 デバイスに意味のある名前をつけることで、プロジェクトの構成が容易になります。

手順	手順内容
5	**デバイスの追加**ボタンをクリックします。
6	**閉じる**ボタンをクリックして**デバイスの追加**ダイアログボックスを閉じます。
7	**通信マネージャー**を設定します。

デバイスを通信マネージャーに追加

概要

デバイスツリーのフィールドデバイスマネージャー (例 : CANopen または Modbus マネージャー) を選択して緑色のプラスサインをクリックすることで、フィールドデバイスを通信マネージャーに追加することができます。または、**デバイスツリー**のフィールドデバイスマネージャーノードを右クリックし**デバイスの追加**コマンドを実行します。

デバイスが**デバイスリポジトリー** ダイアログボックスにあることが前提条件です *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

デバイスの追加

手順	手順内容
1	**デバイスツリー**のフィールドデバイスマネージャーノード (CANopen または Modbus マネージャー) を選択し緑色のプラスサインをクリックするか、フィールドデバイスマネージャーノードを右クリックし、**デバイスの追加 ...** コマンドをコンテキストメニューから実行します。 **結果** : **デバイスの追加**ダイアログボックスが開きます。
2	**デバイスの追加**ダイアログボックスでの**メーカー**リストボックスから **Schneider Electric** を選択します。 **注記** : **メーカー**リストボックスをクリックして、デバイスをブランド別にフィルターできます。
3	下のリストからデバイスを選択します。
4	**名前**テキストボックスに名前を入力してデバイス名を変更します。 **注記 :** 名前にはスペースを使用しないでください。名前の末尾にアンダースコア (_) を使用しないでください。 デバイス名を変更しない場合、初期設定の名前になります。 デバイスに意味のある名前をつけることで、プロジェクトの構成が容易になります。
5	**デバイスの追加**ボタンをクリックします。 **結果** : フィールドデバイスがフィールドデバイスマネージャーに追加されます。 **注記 : デバイスの追加**ダイアログボックスは開いたままです。 次の操作を実行できます。 ● デバイスをさらに追加するには手順 2 に戻ります。 ● **閉じる**ボタンをクリックします。

診断情報へのアクセス

CANopen 上のデバイスの診断情報を取得するには、CAA_CiA405.library を使用します。

診断設定へのアクセス (アドバンストユーザー用)

CANopen コンフィギュレーターの**サービスデータオブジェクト**タブにある、**エラーの場合は中止**、および**エラーの場合は行にジャンプ**、のオプションを使用して潜在的な設定の不一致に対処することができます。

CAN マスターのパフォーマンスを最適化するために、CAN 診断はコントローラーの CAN マスターの外部にあります。CAN 診断構造は**ライブラリーマネージャー**にある CanConfig Extern ライブラリーで定義します。

g_aNetDiagnosis 構造にはスレーブの最新診断情報があります。スレーブが設定されるたびにこの構造は更新されます。

この構造はプログラム内で以下を実行するために使用できます。
- SDO メッセージを介して設定されたスレーブの応答を監視。
- マシンまたはアプリケーションを起動許可する前に、スレーブからの中止メッセージがないかマスターを監視。

この構造は、アプリケーションのテスト、デバッグおよび試運転の間、ユーザーアプリケーション内で設定され有効である必要があります。マシンとその制御アプリケーションの試運転と検証がされた場合は、CANopen ネットワークのデータ転送量を減らすためにこのコードの実行を無効にすることができます。

ただし、アプリケーションのライフサイクル中やマシンまたはマシンが制御する処理中に、動作可能なシステムのスレーブが追加または交換された場合、診断構造をアプリケーションで有効なままにしてください。

⚠ 警告

装置の意図しない動作

- g_aNetDiagnosis データ構造をアプリケーション内で使用して、設定コマンドに対する CAN スレーブの応答を監視してください。
- いずれかの CAN スレーブから SDO 中止メッセージを受信した場合、アプリケーションが起動しないこと、機械または処理を動作可能な状態にしないことを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

CanConfig Extern ライブラリーをアプリケーションに追加後、**Net Diagnostic** 定義をアプリケーションで使用し CAN スレーブからの SDO 中止メッセージをテストします。

次のコード例は CAN 診断データ構造の使用方法を示しています。

```
IF g_aNetDiagnosis[CAN_Net_Number].ctSDOErrorCounter = 0 THEN
   (* 設定でエラーは検知されていません *)
ELSE
   (* 設定でエラーが検知されました。エラーの最新情報を取得してください。*)
   // 中止コードを送信したスレーブのノード ID
   ReadLastErrorNodeID := g_aNetDiagnosis[CAN_Net_Number].usiNodeID;
   // 中止した SDO で使用された インデックス
   ReadLastErrorIndex := g_aNetDiagnosis[CAN_Net_Number].wIndex;
   // 中止した SDO で使用された サブインデックス
   ReadLastErrorSubIndex := g_aNetDiagnosis[CAN_Net_Number].bySubIndex;
   //SDO 中止コード
   ReadLastErrorSdoAbortCode := g_aNetDiagnosis [CAN_Net_Number].udiAbortCode;
   (* 起動、他の機械や処理の動作を許可しないでください。*)
END_IF
```

注記：この例では、CAN0 ポートの CAN_Net_Number は 0 です。コントローラーが装備されている場合、CAN1 ポートは 1 です。

テンプレートからデバイスを追加

概要

デバイステンプレートを使用して新しいデバイスを追加することもできます。この手順については、*デバイステンプレートの管理* (*708* ページ) を参照してください。

5.3
デバイスの更新

デバイスの更新

概要

デバイスの更新機能を使用して**デバイスツリー**で選択したデバイスを以下に置き換えることができます。

- 同じデバイスの別のバージョン、または
- 別のタイプのデバイス

デバイスの更新

次の手順に従って EcoStruxure Machine Expert プロジェクトのデバイスを別のバージョンまたは違うデバイスに置き換えます。

手順	手順内容
1	**デバイスツリー**で置き換えたいデバイスを選択し、**プロジェクト**メニューの**デバイスの更新 ...** コマンドを実行します。 または **デバイスツリー**で置き換えたいデバイスを右クリックし、コンテキストメニューから**デバイスの更新 ...** を実行します。 **結果** : **デバイスの更新**ダイアログボックスが開きます。 または **デバイスツリー**で置き換えたいデバイスを右クリックし、コンテキストメニューから**デバイスの追加 ...** を実行します。**デバイスの追加**ダイアログボックスで**アクション : デバイスの更新**を選択します。 **結果** : **デバイスの追加**ダイアログボックスが**デバイスの更新**ダイアログボックスに変換されます。
2	**デバイス :** リストから現在のデバイスを置き換えるデバイスを選択します。 デバイスの特定のバージョンを選択するには、**すべてのバージョンを表示 (専門技術者専用)** および / または **Display outdated versions** オプションを選択します。
3	**デバイスの更新**ボタンをクリックします。 **結果 : デバイスツリー**で選択したデバイスが新しいデバイスタイプまたは新しいバージョンに置き換わります。**デバイスタイプ**で選択されたノードには新しいデバイスタイプまたは新しいバージョンが表示されます。

デバイスの更新後の影響

デバイスツリーで更新したデバイスの下にあるサブデバイスも自動的に更新されます。

デバイスタイプが同じ場合はデバイス設定は変更されません。

更新によって既存の設定に不一致が生じる場合は、次回のアプリケーションの**ビルド**で検出されます。検出された不一致はメッセージで表示されます。これは、デバイスの更新時に自動的には適切に削除されない暗黙的に追加されたライブラリーも関係します。

5.4
デバイスの変換

デバイスの変換

概要

EcoStruxure Machine Expert 4.0 以降のバージョンでは、プロジェクトで設定されたデバイスを別の互換性のあるデバイスに変換できます。EcoStruxure Machine Expert は、現在設定されているデバイスを選択したデバイスに変換し、変更を**メッセージ**ビューに表示します。

デバイスの変換コマンドは自動的にモジュールを追加または削除することがあります。これらのハードウェアの変更はアドレス設定やライブラリーにも影響します。

デバイスの変換後に意図しない動作が発生しないようにするには以下を行います。
- 新しいデバイスがプロジェクトで必要な機能や通信ポートをサポートしていることを確認します。
- アプリケーションで直接アドレスを使用しないようにします。
- デバイスを変換する前に、パソコンにプロジェクトのバックアップを実行してください。

⚠ 警告

装置の意図しない動作
- デバイスの変換後に、アプリケーションで使用されている直接アドレス (例 : %IB5) が正しく変換されていることを確認してください。
- デバイスの変換後、変更されたプロジェクトには意図された通りの設定がされ、意図された機能が提供できているかを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記

データの損失

デバイスを変換する前に、パソコンにプロジェクトのバックアップを実行してください。

上記の指示に従わないと、物的損害を負う可能性があります。

デバイスの変換

次の手順に従って互換性のあるデバイスに変換します。

手順	手順内容
1	デバイスを変換する前に**ファイル → ... としてプロジェクトを保存**コマンドを実行し、プロジェクトをパソコンにバックアップします。
2	**デバイスツリー**で変換したいデバイスを右クリックします。
3	**デバイスの変換**コマンドをコンテキストメニューから実行します。 **結果** : **デバイスの変換**ダイアログボックスが表示されます。選択したデバイスと互換性のあるデバイスリストが表示され、選択したデバイスに関する詳細情報が表示されます。
4	現在設定されているデバイスを別のデバイスに変換するのにリストから選択します。 使用できるデバイスのバージョンを表示するには**すべてのバージョンを表示 (専門技術者専用)** オプションを選択します。
5	バックアップを行っていない場合は、**キャンセル**をクリックして操作を取りやめ、操作を再度行う前にバックアップを行ってください。 **OK** をクリックして変換を開始します。 **結果** : 現在設定されているデバイスがリストから選択したデバイスに変換されます。関連するモジュールがまだ利用可能な場合は、入力した情報は維持されます。変換できなかった変更や設定は**メッセージ**ビューにリスト表示されます。
6	変更されたプロジェクトには意図された通りの設定がされ、意図された機能が提供できているかを確認してください。されていない場合は設定を変更するか、変更されていないプロジェクトファイルのバックアップを復元します。

メッセージビューに表示される変換情報

次の変換処理の情報は**メッセージ**ビューに表示されます。

- 変換前と変換後のデバイス
- 変換時に移行されなかったパラメーター
- 変換されなかったデバイス

メッセージビューで表示された情報を保存するには、クリップボードにコピー (CTRL + C) しデータファイルに貼り付け (CTRL + V) ます。

5.5
プロジェクトの変換

SoMachine Basic と Twido プロジェクトの変換

概要

EcoStruxure Machine Expert を使用すると、SoMachine Basic または TwidoSoft/TwidoSuite プロジェクト、および設定されたコントローラーを選択可能な EcoStruxure Machine Expert ロジックコントローラー (*849* ページ) に変換することができます。コントローラーとそれに対応するロジックが変換され EcoStruxure Machine Expert プロジェクトに統合されます。

変換処理では、**ファイル → SoMachine Basic プロジェクトの変換**、または**ファイル → Twido プロジェクトの変換**コマンドを実行します。**SoMachine Basic プロジェクトの変換**ダイアログボックスまたは **Twido プロジェクトの変換**ダイアログボックスが開きます。コマンドが利用できない場合は、**ツール → カスタマイズ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で、使用したコマンドをメニューに挿入することができます。

変換メカニズムをサポートしている SoMachine Basic バージョンは EcoStruxure Machine Expert のリリースノートに記載されています。サポートしている最新バージョンよりも新しいバージョンの SoMachine Basic で作成された SoMachine Basic プロジェクトを変換すると、**メッセージ** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* にメッセージが表示されます。その後、変換を続行またはキャンセルすることができます。続行するとアプリケーションは変換されますが、修正が必要なエラーが発生する可能性があります。この場合、サービスを開始する前にメッセージビューとアプリケーションの両方を確認し検証してください。

注記: SoMachine Basic または Twido プロジェクトが EcoStruxure Machine Expert に変換される前に有効であることを確認してください。

注記：パスワードで保護されたプロジェクトは変換できません。

プロジェクトの変換後に意図しない動作をさせないために、変換後のコントローラーがプロジェクトで必要な機能や通信ポートをサポートしていることを確認してください。

⚠ 警告

装置の意図しない動作

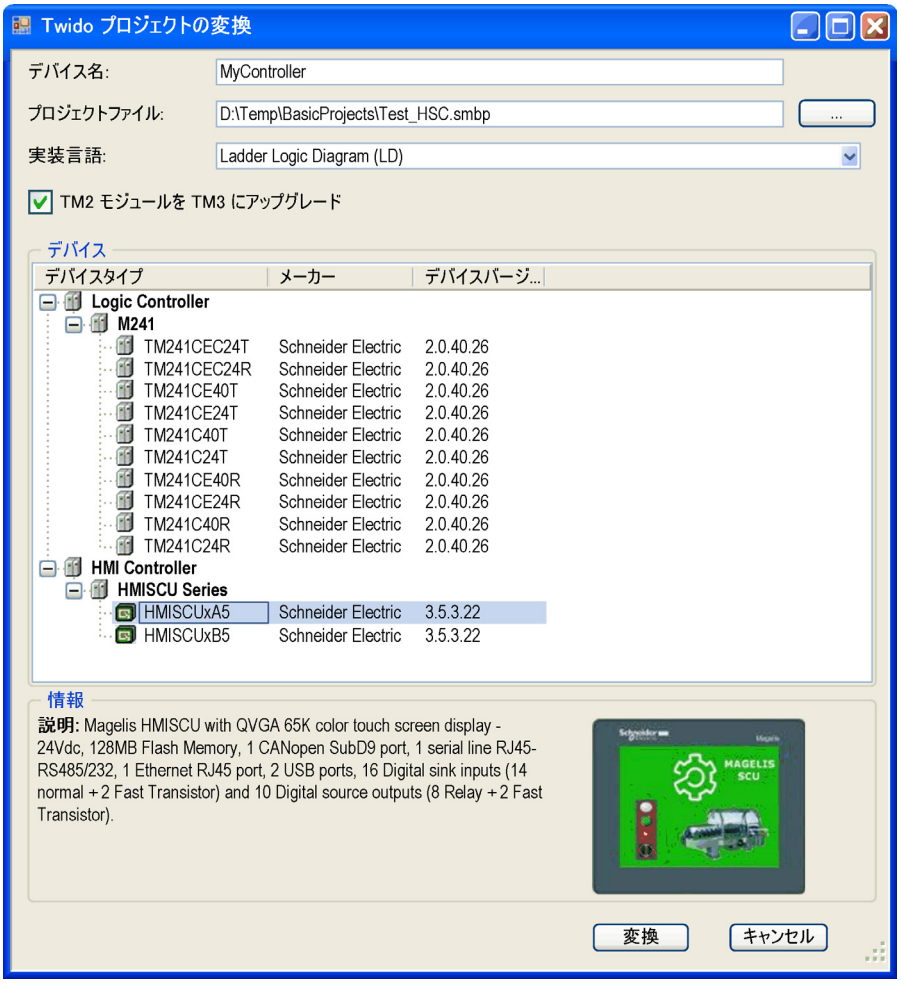
- 変換後のコントローラーのプログラムに目的の設定があり、プロジェクトの変換後に目的のファンクションが機能することを確認してください。
- 運用前に、変換されたプログラムの機能を完全にデバッグ、確認、検証してください。
- プログラムを変換する前に変換前のプログラムが有効であること、例えば、変換前のコントローラーにダウンロードできることを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：EcoStruxure Machine Expert へのプロジェクトのインポートに関する詳細、アドバイス、重要な安全情報については、*Compatibility and Migration User Guide (EcoStruxure Machine Expert Compatibility and Migration, User Guide 参照)* を参照してください。

SoMachine Basic または Twido プロジェクトの変換

次の手順に従って SoMachine Basic または Twido プロジェクトを変換します。

手順	手順内容
1	変換処理を開始するには、EcoStruxure Machine Expert Logic Builder (本章の*概要* に記載 (*65* ページ)) の 3 つのアクションのいずれかを実行します。 **結果** : **SoMachine Basic プロジェクトの変換**ダイアログボックス、または **Twido プロジェクトの変換**ダイアログボックスが開きます。 
2	**デバイス名**フィールドにコントローラー名を入力します。
3	SoMachine Basic または Twido プロジェクトファイルのパスを**プロジェクトファイル**ボックスに入力する、または **...** ボタンをクリックしてファイルを参照します。 **注記：**一度 SoMachine Basic または Twido プロジェクトを**プロジェクトを開く**ダイアログボックスで参照した場合、**プロジェクトファイル**フィールドにパスが自動的に入力され編集できません。
4	**記述言語**リストからロジック変換後のプログラミング言語を選択します。 以下のプログラミング言語がサポートされています。 ● ラダーダイアグラム (LD) ● ファンクションブロックダイアグラム (FBD) ● 命令リスト (IL) ● コンティニュアスファンクションチャート (CFC)
5	SoMachine Basic または Twido コントローラーを変換する**デバイス**リストから、目的のコントローラーを選択します。選択したデバイスの詳細情報は**情報**領域に表示されます。
6	**変換**をクリックして変換を開始します。 **結果** : SoMachine Basic または Twido プロジェクトは、開いている EcoStruxure Machine Expert プロジェクトに変換され取り込まれます。変換できなかった変更や設定は**メッセージ** ビューに表示されます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
7	**メッセージ**ビューの**プロジェクトの変換**カテゴリーを参照し、検出されたエラーとアラートを確認します。

手順	手順内容
8	変換されたプロジェクトに適切な設定および適切なファンクションが含まれているかを確認します。できていない場合は、設定を調節します。

オブジェクト名と変数名の IEC 互換性

EcoStruxure Machine Expert プロジェクトのオブジェクト名と変数名は IEC 規格で定義されている命名規則に準拠しなければなりません。SoMachine Basic または Twido プロジェクト内の規格に準拠していない名前は変換時に自動的に IEC 規則に適合されます。

変換された EcoStruxure Machine Expert プロジェクトの IEC 互換ではない名前を保持する場合は、**Project Settings → コンパイルオプション** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の **Allow unicode characters for identifiers** オプションを有効にします。

TwidoEmulationSupport ライブラリー

TwidoEmulationSupport ライブラリー *(EcoStruxure Machine Expert, Twido Emulation Support Library, Library Guide 参照)* には SoMachine Basic および TwidoSoft/TwidoSuite の機能を EcoStruxure Machine Expert アプリケーション内で提供するファンクションとファンクションブロックが含まれています。TwidoEmulationSupport ライブラリーは EcoStruxure Machine Expert プロジェクトに変換されたコントローラーと共に自動的に取り込まれます。

アプリケーションプログラムの変換

変換後の EcoStruxure Machine Expert プロジェクトでは、各 SoMachine Basic POU と無料の POU、および各 Twido サブルーチンとプログラムセクションごとに、別々のプログラムが作成されます。これらのプログラムに使用されるプログラミング言語は、**SoMachine Basic プロジェクトの変換** / **Twido プロジェクトの変換**ダイアログボックスで選択された**記述言語**によって決まります。グラフィックグラフセでプログラミングされた POU は例外です。それらは SFC プログラムに変換されます。詳細は、この章のグラフセ (*73* ページ) を参照してください。

アプリケーションプログラムが使用している言語オブジェクト (メモリーオブジェクト、ファンクションブロックなど) ごとに 1 つのグローバル変数が作成されます。別々のオブジェクトカテゴリー (メモリービット用、メモリーワード用など) とに別々の グローバル変数リスト (*172* ページ) が作成されます。

アプリケーションプログラムの変換でプログラムの構造に関して以下の制限があります。

- EcoStruxure Machine Expert では、別のプログラムのラベル (*264* ページ) にジャンプすることはできません。
- サブプログラムでグラフセステップを定義することはできません。
- サブプログラムでグラフセステップ (# および D# 命令) を有効または無効にすることはできません。

メモリーオブジェクトの変換

SoMachine Basic および Twido のメモリーオブジェクト用の領域は EcoStruxure Machine Expert と異なります。

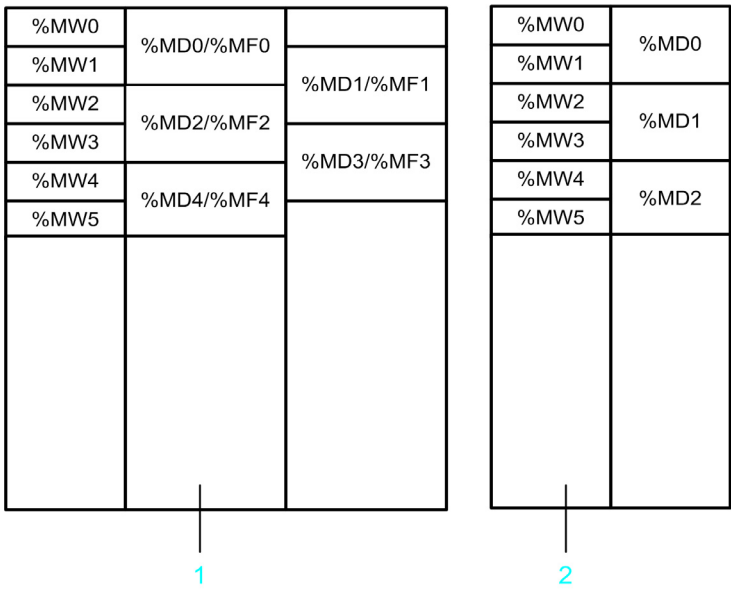
SoMachine Basic および Twido では、メモリーオブジェクトには 3 つの異なる領域があります。

領域	メモリーオブジェクト
ビットメモリー領域	ビットメモリー (%M)
ワードメモリー領域	• ワードメモリー (%MW) • ダブルワード (%MD) • 浮動小数点の値 (%MF)
定数領域	• ワード型定数 (%KW) • ダブルワード (%KD) • 浮動小数点の値 (%KF)

EcoStruxure Machine Expert では、メモリーオブジェクトにはワードメモリー領域のみがあります。

領域	メモリーオブジェクトは以下を含みます。
ワードメモリー領域	● ワードメモリー (%MW) ● ダブルワード (%MD) ● 浮動小数点の値 浮動小数点の値には特別なアドレス形式はありません。浮動小数点の変数は、%MD アドレスにマップできます。

下図は、SoMachine Basic / Twido および EcoStruxure Machine Expert の %MD および %MF アドレスの異なるレイアウトの概要を示しています。



1 SoMachine Basic / Twido のメモリーアドレス
2 EcoStruxure Machine Expert のメモリーアドレス

メモリーオブジェクトは以下のように変換されます。

変換前メモリーオブジェクト	変換後メモリーオブジェクト	詳細
%MW	同じ %MW アドレスにマップ。 **例** %MW2 は %MW2 にマップされます。	各 %MW オブジェクトに、INT 型のグローバル変数が作成されます。
偶数アドレスの %MD および %MF	以前と同じ %MW アドレスに配置されるようにマップされます。 **例** %MD4 / %MF4 は %MD2 にマップされます。	各 %MD オブジェクトに、DINT 型のグローバル変数が作成されます。 使用される各 %MF オブジェクトに、REAL 型のグローバル変数が作成されます。
奇数アドレスの %MD および %MF	DINT 変数は奇数ワードアドレスに配置できないのでマップできません。	変換されたアプリケーションを確実にビルドするために変数が作成されます。ただし、そのような変数の作成がプログラムの全体機能に及ぼす影響を調べる必要があります。
%M	%MW 領域の固定場所にパケットビットフィールドとしてマップされます。	各 %M オブジェクトに、BOOL 型のグローバル変数が作成されます。
%KW	%MW 領域の連続アドレスにマップされます。	使用される各 %KW オブジェクトに、INT 型のグローバル変数が作成されます。

%KW、%KD、および %KF オブジェクトの関係は %MW、%MD、および %MF オブジェクトと同じです。例えば、%KD4 / %KF4 は %KW4 と同じ場所にマップされます。奇数の %KD / %KF アドレスはマップできません。

リモートアクセス

メモリーオブジェクト (%MW、%MD、%MF、および %M) は Modbus サービスを介したリモートデバイスでアクセスできます。

- 変換前のアプリケーションでリモートデバイスが %MW、%MD、または %MF オブジェクトをアクセスしている場合、このアクセスは EcoStruxure Machine Expert アプリケーションでも引き続き利用できます。
- 変換前のアプリケーションでリモートデバイスが %M オブジェクトをアクセスしている場合、このアクセスは EcoStruxure Machine Expert アプリケーションでは利用できなくなります。

立上がり接点および立下り接点の処理

立上がり接点および立下り接点は次のように変換されます。

1. 接尾辞 _Rise/_Fall をもつグローバル変数が作成されます (例えば、立上がり接点 %M1 には M1_Rise)。
2. この変数は、SystemFunctions プログラムの R_TRIG/F_TRIG インスタンスを介して設定されます。

接点の検出はコントローラーサイクルの始めに行われます。

FALLING/RISING 命令は直接 R_TRIG/F_TRIG インスタンスに変換されます。

接点の検出は元のアプリケーションと同じ実行シーケンスの場所で行われます。

ファンクションブロックの変換

SoMachine Basic / Twido の以下のファンクションブロックに対して、TwidoEmulationSupport ライブラリーには互換性のある機能をもつファンクションブロックがあります。

SoMachine Basic / Twido ファンクションブロック	TwidoEmulationSupport ライブラリー ファンクションブロック
タイマー %TM	FB_Timer
カウンター %C	FB_Counter
レジスター %R	FB_FiFo / FB_LiFo
ドラム %DR	FB_Drum
シフトビットレジスター %SBR	FB_ShiftBitRegister
ステップカウンター %SC	FB_StepCounter
スケジュール %SCH	FB_ScheduleBlock
PID	FB_PID
交換 / メッセージ %MSG	FB_EXCH
高速カウンター %HSC / %VFC	本章の *高速カウンター (FC) と高速カウンター (HSC) (Twido : 超高速カウンター) およびパルス出力の変換* (*71* ページ) に変換方法が記述されています。
高速カウンター %FC	
PLS パルス出力 %PLS	
PWM パルス出力 %PWM	
PTO ファンクションブロック %PTO、%MC_xxx_PTO	
周波数発生器 %FREQGEN	
通信ファンクションブロック READ_VAR, WRITE_VAR, WRITE_READ_VAR および SEND_RECV_MSG	FB_ReadVar, FB_WriteVar, FB_WriteReadVar および FB_SendRecvMsg
SMS ファンクションブロック SEND_RECV_SMS	変換されません。
MC_MotionTask_PTO	
ドライブファンクションブロック %MC_xxx_ATV	
%DATALOG	

ファンクションブロックの変換については次の点に注意してください。

- TwidoEmulationSupport ライブラリーには、高速カウンター (HSC)、高速カウンター (FC)、およびパルス出力などのハードウェアに関連するファンクションブロックはありません。これらは、プラットフォーム固有の HSC および PTO_PWM ライブラリーのファンクションブロックを通して制御される必要があります。これらのファンクションブロックは変換前のファンクションブロックと互換性がありません。要約すると、コントローラーハードウェアリソースに基づくファンクションが変換前のプログラムにある場合は、完全な変換はできません。詳細については *高速カウンター (FC) と高速カウンター (HSC) (Twido: 超高速カウンター) およびパルス出力の変換* (*71* ページ) を参照してください。
- SoMachine Basic / Twido では、EXCHx 命令と %MSGx ファンクションブロックによるメッセージ機能があります。EcoStruxure Machine Expert アプリケーションでは、この機能は 1 つのファンクションブロック FB_EXCH によって実行されます。
- SoMachine Basic / Twido では、特定のファンクションブロックは特別なダイアログボックスで設定できます。この設定データーは、専用パラメーターによって TwidoEmulationSupport ライブラリーのファンクションブロックに提供されます。
- ラング に複数のファンクションブロックがある場合、コンバーターはラングを複数のロジックネットワークに分割することがあります。

ネットワークオブジェクトブジェクトの変換

この表は変換でサポートされているネットワークオブジェクトの種類を示しています。

ネットワークオブジェクト	オブジェクト機能	対応
%QWE	入力アセンブリ (EtherNet/IP)	あり
%IWE	出力アセンブリ (EtherNet/IP)	あり
%QWM	入力レジスター (Modbus TCP)	あり
%IWM	出力レジスター (Modbus TCP)	あり
%IN	デジタル入力 (IO scanner)	Serial IO scanner のみ
%QN	デジタル出力 (IO scanner)	Serial IO scanner のみ
%IWN	入力レジスター (IO scanner)	Serial IO scanner のみ
%QWN	出力レジスター (IO scanner)	Serial IO scanner のみ
%IWNS	(IO scanner 診断)	Serial IO scanner のみ

システム変数の変換

以下のシステムビットおよびシステムワードが変換されます。

システムビット / ワード	詳細
%S0	コールドスタート後、最初のサイクルで 1 に設定されます。 **注記：**このシステムビットに書き込むことでコールドスタートをトリガーすることはできません。
%S1	ウォームスタートの後、最初のサイクルで 1 に設定されます。 **注記：**このシステムビットに書き込むことでウォームスタートをトリガーすることはできません。
%S4	時間ベース 10 ms のパルス
%S5	時間ベース 100 ms のパルス
%S6	時間ベース 1 s のパルス
%S7	時間ベース 1 min のパルス
%S13	コントローラー起動後、最初のサイクルで 1 に設定されます。
%S18	算術オーバーフローが発生した場合、1 に設定されます。 **注記：**このフラグは TwidoEmulationSupport ライブラリーが提供しているので、このライブラリーが提供しているファンクションによりのみ設定されます。
%S21 、%S22	書き込みのみ。これらの変数は読み込みできません。
%S113	シリアルライン 1 の Modbus Serial IOScanner を停止します。
%S114	シリアルライン 2 の Modbus Serial IOScanner を停止します。
%SW63. . . 65	MSG ブロック 1...3 のエラーコード。
%SW114	スケジュールブロックのフラグを有効にします。

その他のシステム変数の変換はサポートされていません。変換前のアプリケーションプログラムで使用されているシステム変数がサポートされていない場合は、**メッセージ** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**プロジェクトの変換**カテゴリーでメッセージが生成されます。

保持動作の変換

SoMachine Basic / Twido の変数とファンクションブロックは保持変数です。予期せぬコントローラーのシャットダウン後もコントローラーの通常の電源サイクル後も、それらの値と状態を保持します。

この保持動作は変換中に保存されません。EcoStruxure Machine Expert では、通常の変数やファンクションブロックに変換されるので、それらは、予期せぬコントローラーのシャットダウンやコントローラーの通常の電源サイクル中に初期化されます。EcoStruxure Machine Expert アプリケーションで変数を保持する必要がある場合は、この属性キーワード (*496* ページ) を手動で定義します。

アニメーションテーブルの変換

アニメーションテーブルの管理は、変換前と変換後のアプリケーションで異なります。

- SoMachine Basic / Twido では、名前別に複数のアニメーションリストを定義することができます。各アニメーションリストには、アニメーション化されるオブジェクトを複数含めることができます。各変数に対して、**トレース**オプションを選択できます。
- EcoStruxure Machine Expert には、4 つの定義済み ウォッチリスト (*386* ページ) (**ウォッチ 1**... **ウォッチ 4**) があります。各ウォッチリストにはアニメーション化される複数の変数を含めることができます。1 つのウォッチリストには異なるコントローラーの変数を含めることができます。
  SoMachine Basic / Twido で **トレース** オプションが選択された変数には、EcoStruxure Machine Expert がトレースオブジェクトを作成します。これらの変数は、トレースエディター (*415* ページ) で表示されます。

変換処理で変換前のアニメーションテーブルの項目はウォッチリスト **ウォッチ 1** の最後に追加されます。

シンボルの変換

SoMachine Basic / Twido プロジェクトで定義されたシンボルは EcoStruxure Machine Expert プロジェクトに自動的に移されます。

シンボル名には以下の制限があります。

条件	結果
シンボル名が EcoStruxure Machine Expert の命名規則にそぐわない場合。	シンボル名が変更されます。
シンボル名が EcoStruxure Machine Expert のキーワードと同じ場合。	シンボル名が変更されます。
言語オブジェクト用に変数が作成されない場合。	シンボル名は破棄されます。
シンボルがアプリケーションプログラムで使用されていない場合。	シンボル名は破棄されることがあります。

変更が必要であったシンボルのリストは、**メッセージ**ビューを参照してください。

高速カウンター (FC) と高速カウンター (HSC) (Twido: 超高速カウンター) およびパルス出力の変換

EcoStruxure Machine Expert で提供されるファンクションブロックと SoMachine Basic / Twido で提供されるファンクションブロックは異なります。しかしながら、高速カウンター (FC)、高速カウンター (HSC)、およびパルス出力は可能な限り変換されます。次のセクションでは適用される制限の概要を説明します。

一般的な制限

以下の一般的な制限があります。

制限事項	対処方法
変換された高速カウンター (HSC)、およびパルス出力で使用される入出力は、変換前のアプリケーションの入出力と異なる場合があります。	変換されたコントローラーの配線ではこれを考慮してください。 入力と出力の再割り当ては**メッセージ** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* にレポートされます。

制限事項	対処方法
SoMachine Basic コントローラーは選択した変換前のコントローラーとは異なる数のカウンターとパルス出力に対応している場合があります。変換機能では、変換後のコントローラーが対応しているカウンターとパルス出力のみを変換します。	アプリケーションを手動で調整してください。

%FC、%HSC / %VFC、%PLS および %PWM **の変換に関する制限事項**

SoMachine Basic / Twido アプリケーションで使用されている各 %FC、%HSC、%VFC、%PLS および %PWM ファンクションブロックに対して、1 つのプログラムが EcoStruxure Machine Expert で作成されます。アプリケーションのニーズに応じてこの基本的な実装を改善することができます。

以下の制限があります。

制限事項	対処方法
ファンクションブロックのパラメーターへのアクセスは、SoMachine Basic および EcoStruxure Machine Expert では異なる方法で実行されます。 SoMachine Basic では、ファンクションブロックのパラメーターはアプリケーションプログラムから直接アクセスします。例 : %HSC.P = 100 EcoStruxure Machine Expert では、コントローラー固有のファンクションブロック (例 : EXPERTSetParam) を使用してパラメーターにアクセスします。	変換前のアプリケーションがファンクションブロックのパラメーターにアクセスしている場合は、変換後のアプリケーションをそれに応じて拡張する必要があります。
EcoStruxure Machine Expert のカウンターの動作は、プリセット値が設定されている場合、SoMachine Basic / Twido と異なります。 Twido の場合 ● 減算カウンタは、0 に達するとカウントを続けます。 ● 加算カウンタは、プリセット値に達するとカウントを続けます。 EcoStruxure Machine Expert の場合 ● 減算カウンタは、0 に達するとカウントを停止します。 ● 加算カウンタは、プリセット値に達するとカウントを最初から開始します。	アプリケーションを手動で調整してください。
SoMachine Basic ファンクションブロックの以下のパラメーターは EcoStruxure Machine Expert に変換できません。 ファンクションブロック %PLS: ● 出力パラメーター D [Done] ● パラメーター R [Duty Cycle] ファンクションブロック %PWM: ● パラメーター R [Duty Cycle] ファンクションブロック %HSC: ● 出力パラメーター U [Counting Direction]	アプリケーションを手動で調整してください。

PTO ファンクションブロックの変換に関する制限事項 %PTO および %MC_xxxx

M241 の場合

EcoStruxure Machine Expert の M241 コントローラー用 PTO ファンクションブロックは SoMachine Basic の PTO ファンクションブロックと互換性があります。PTO ファンクションブロックの変換に制限はありません。唯一の例外は MC_MotionTask_PTO ファンクションブロックです。MC_MotionTask_PTO は変換されません。

HMISCU の場合

EcoStruxure Machine Expert に提供されている HMISCU コントローラーの PTO ファンクションブロックは SoMachine Basic に提供されている PTO ファンクションブロックと互換性がありません。PTO ファンクションブロックは変換されません。

ファンクションブロックの変換に関する制限事項 %FREQGEN

周波数発生器のファンクションブロック %FREQGEN は M241 および HMISCU コンバーターの両方に対して制限なく変換されます。

ループ要素 (FOR / ENDFOR) の変換

変換の設定先言語はループに対応していません。そのため、FOR ループはラベル要素とジャンプ要素を使って機能的に同等な論理ネットワークのシーケンスに分割されます。

条件付き要素 (IF / ELSE / ENDIF) の変換

変換の設定先言語は条件記述 (別の目的ですでに使用されている EN / ENO を除く) に対応していません。そのため、IF 構造はラベル要素とジャンプ要素を使って機能的に同等な論理ネットワークのシーケンスに分割されます。

グラフセプログラムの変換

グラフセプログラムはテキスト形式またはグラフィック形式で記述することができます。

グラフセ型	詳細	対応ソフト
テキスト	グラフセ状態の定義、有効化、無効化には、さまざまな IL および LD のプログラミング要素が利用できます。	● TwidoSoft/TwidoSuite ● SoMachine Basic
グラフィック	ステップ、移行、および分岐のレイアウトをグラフィック形式で描画できます。	SoMachine Basic V1.4 移行のバージョンのみ。

テキスト形式のグラフセの変換

EcoStruxure Machine Expert のプログラミング言語はグラフセを使ったプログラミングに対応していません。

そのため変換されたグラフセアプリケーションにはグラフセ管理を実装させる追加の言語要素が含まれています。

付加的要素	詳細
フォルダー**グラフセ**	このフォルダーにはグラフセ状態マシンの管理に使用される以下の言語要素が含まれます。
データ構造 GRAFCET_STATES	このデータ構造には、各グラフセ状態につき 1 ビットあります。初期状態では、この要素は TRUE に設定され、それ以外では FALSE に設定されます。
グローバル変数リスト**グラフセ 変数**	このグローバル変数リストには以下の変数が含まれます。 ● 各グラフセ状態に対する 1 ビットを含む STATES 変数 1 つ。各ビットは、対応するグラフセ状態 (%Xi オブジェクト) の現在値をあらわします。 ● 各グラフセ状態に対する 1 ビットを含む ACTIVATE_STATES 変数 1 つ。このビットが TRUE の場合、グラフセ状態は次のサイクルで有効にされます。 ● 各グラフセ状態に対する 1 ビットを含む DEACTIVATE_STATES 変数 1 つ。このビットが TRUE の場合、グラフセ状態は次のサイクルで無効にされます。
グラフセ プログラム	このプログラムはグラフセ状態マシンを実装します。グラフセステップの有効化、および無効化のロジックを含みます。 このプログラムは以下のアクションを含みます。 ● Init はグラフセステップを初期状態にします。これは、アプリケーションプログラムでシステムビット %S21 が設定されると実行されます。 ● Reset はグラフセステップを FALSE にリセットします。これは、アプリケーションプログラムでシステムビット %S22 が設定されると実行されます。

アプリケーションプログラムのグラフセ命令は次のように変換されます。

- 各グラフセステップの開始にはステップ名のラベルが付けられます。
  グラフセステップ内の最初の記述は、ステップが有効かどうかをチェックします。有効でない場合、次のグラフセステップのラベルへジャンプします。
- %Xi へのアクセスは STATES.Xi 変数へのアクセスに変換されます。

- Grafcet を有効にする #i 命令は i 状態を有効にするビットと現在の状態を無効にするビットをセットします。
- Grafcet を無効にする #Di 命令は i 状態を無効にするビットと現在の状態を無効にするビットをセットします。

このセクションの情報を考慮すると、変換したグラフセプログラムを拡張することができます。

グラフィックグラフセの変換

グラフィックグラフセは EcoStruxure Machine Expert で提供されているプログラミング言語 SFC に似ています。この理由からグラフィックグラフセ POU は可能な限り SFC プログラムに変換されます。

次にグラフィックグラフセと SFC の相違点を示します。

グラフィックグラフセ	SFC	詳細
任意の数の初期ステップをもつことができます。	1 つの初期ステップが必要です。	グラフィックグラフセ POU に複数の初期ステップがある場合、変換機能は SFC に複数の初期ステップを作成します。これは、変換後のアプリケーションでエラーが検出されずビルドできないという影響があります。 変換後のプログラムを慎重に調整してください。
代替分岐の複数のステップを有効にすることが可能です。	1 つの代替分岐のステップのみを有効にすることが可能です。	変換後のプログラムが予想通りに動作していることを慎重に確認してください。
ステップの出力移行はステップ実行直後に評価されます。	SFC プログラムの移行はすべてのステップが実行されてから評価されます。	変換後のプログラムが予想通りに動作していることを慎重に確認してください。
ステップ、移行、および分岐のレイアウトは自由にできます。	ステップ、移行、および分岐のレイアウトは制限されています。	グラフィックレイアウトは可能な限り SFC に変換されます。変換中に発生した非互換性は**メッセージ**ビューに表示されます。 ステップアクションと移行セクションは完全に変換されます。 必要に応じて作成された SFC を完成させてください。

グラフィックグラフセ POU はシステムビット %S21 をセットすることで初期化できます。SoMachine Basic プロジェクトでこのビットがセットされると、変換機能では暗黙的な変数 SFCInit を有効にし SFC プログラムの初期化に使用します。

TM2 拡張モジュールの TM3 拡張モジュールへの変換

Twido コントローラーでは、TM2 拡張モジュールのみ使用できます。M221 および M241 ロジックコントローラーでは TM3 モジュールだけでなく TM2 モジュールも扱えますが、TM3 モジュールを使用することが推奨されています。Twido プロジェクトで使用された TM2 モジュールを EcoStruxure Machine Expert プロジェクト用の TM3 モジュールに変換するために、**TM2 モジュールを TM3 にアップグレード**がデフォルトで選択されています。

TM2 拡張モジュールは、表示のリストのように TM3 拡張モジュールに変換されます。

変換前の TM2 拡張モジュール	変換後の TM3 拡張モジュール	詳細
TM2DDI8DT	TM3DI8	–
TM2DAI8DT	TM3DI8A	–
TM2DDO8UT	TM3DQ8U	–
TM2DDO8TT	TM3DQ8T	–
TM2DRA8RT	TM3DQ8R	–
TM2DDI16DT	TM3DI16	–
TM2DDI16DK	TM3DI16K	–
TM2DRA16RT	TM3DQ16R	–
TM2DDO16UK	TM3DQ16UK	–
TM2DDO16TK	TM3DQ16TK	–

変換前の TM2 拡張モジュール	変換後の TM3 拡張モジュール	詳細
TM2DDI32DK	TM3DI32K	–
TM2DDO32UK	TM3DQ32UK	–
TM2DDO32TK	TM3DQ32TK	–
TM2DMM8DRT	TM3DM8R	–
TM2DMM24DRF	TM3DM24R	–
TM2AMI2HT	TM3AI2H	–
TM2AMI4LT	TM3TI4	変換後の温度モジュールの動作は、変換前のモジュールと異なる場合があります。変換後のモジュールを慎重に確認してください。
TM2AMI8HT	TM3AI8	–
TM2ARI8HT	–	TM2 モジュールの TM2ARI8HT、TM2ARI8LRJ、および TM2ARI8LT は、TM3 拡張モジュールに対応されていないため変換されません。このモジュールを 2 つの TM3TI4 モジュールに置き換えできます。
TM2AMO1HT	TM3AQ2	対象の TM3 拡張モジュールには、元の TM2 モジュールより多くの I/O チャンネルがあります。
TM2AVO2HT		–
TM2AMM3HT	TM3TM3	–
TM2ALM3LT		変換後の温度モジュールの動作は、変換前のモジュールと異なる場合があります。変換後のモジュールを慎重に確認してください。
TM2AMI2LT	TM3TI4	対象の TM3 拡張モジュールには、元の TM2 モジュールより多くの I/O チャンネルがあります。 変換後の温度モジュールの動作は、変換前のモジュールと異なる場合があります。変換後のモジュールを慎重に確認してください。
TM2AMM6HT	TM3AM6	–
TM2ARI8LRJ	–	TM2 モジュールの TM2ARI8HT、TM2ARI8LRJ、および TM2ARI8LT は、TM3 拡張モジュールに対応されていないため変換されません。このモジュールを 2 つの TM3TI4 モジュールに置き換えできます。
TM2ARI8LT	–	TM2 モジュールの TM2ARI8HT、TM2ARI8LRJ、および TM2ARI8LT は、TM3 拡張モジュールに対応されていないため変換されません。このモジュールを 2 つの TM3TI4 モジュールに置き換えできます。

注記：EcoStruxure Machine Expert プロジェクトで TM2 および TM3 拡張モジュールをご使用の場合ツリー構造での位置に注意してください。ツリー構造で TM3 ノードが TM2 ノードの下にある場合、検知された**ビルド**エラーとして**メッセージ**ビューに表示されます。

Modbus Serial IOScanner の変換

コントローラーのプラットフォームに違いがあるため、また変換されたプログラムが正常に機能することに依存するコントローラー機器では特に、変換処理の結果を確認してください。変換中にエラーやアラートが検知されたかどうかに関わらず、機械や処理の全体を徹底的にテストし検証することが不可欠です。

⚠ 警告
装置の意図しない動作 ● 変換後のコントローラーのプログラムに目的の設定があり、プロジェクトの変換後に目的のファンクションが機能することを確認してください。 ● 運用前に、変換されたプログラムの機能を完全にデバッグ、確認、検証してください。 ● プログラムを変換する前に変換元のプログラムが有効であること、例えば、変換元のコントローラーにダウンロードできることを確認してください。 **上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。**

設定

IOScanner 設定はすべて変換されます。

- デバイスは **Generic Modbus Slave** デバイスに変換されます。変換前のデバイスタイプは保持されません。
- デバイス設定はすべて変換されます。初期化リクエスト、チャンネル設定、および変数のリセットを含みます。

ファンクションブロック

Modbus IOScanner (MC_xxx_ATV) 上の d.Altivar ドライブを制御するファンクションブロックは変換されません。

ステータス処理

IOScanner のステータス処理は SoMachine Basic と EcoStruxure Machine Expert で異なるため、これらの機能は部分的な変換のみになります。アプリケーションが IOScanner のステータス情報を使用している場合、このロジックが動作するかどうか確認してください。

IOScanner ステータス	詳細
デバイスステータス (%IWNSx)	SoMachine Basic と EcoStruxure Machine Expert はいずれもスレーブデバイスのステータス情報を提供しますが、ステータス値は異なります。ステータスロジックは部分的に変換されます。
チャンネルステータス (%IWNSx.y)	EcoStruxure Machine Expert はチャンネルごとのステータス情報は提供しません。チャンネルステータスはデバイスステータスに変換されます。
システムワードおよびシステムビット	
%S110/%S111 (IOScanner リセット)	変換されません。
%S113/%S114 (IOScanner 停止)	変換されます。
%SW210/%SW211 (IOScanner ステータス)	変換されません。

Modbus TCP IO Scanner の変換

Modbus TCP IO Scanner の設定は変換されません。

I/O の即時アクセス

デジタルローカル I/O チャンネルへの即時アクセスをする SoMachine Basic の READ_IMM_IN 命令および WRITE_IMM_OUT 命令は変換されません。

M241 コントローラーの場合、PLCSystem ライブラリーが提供する GetImmediateFastInput ファンクションおよび PhysicalWriteFastOutputs ファンクションを使用できますが、以下の違いに注意してください。

READ_IMM_IN および WRITE_IMM_OUT 命令 (M221 コントローラー)	GetImmediateFastInput および PhysicalWriteFastOutputs ファンクション (M241 コントローラー)
すべてのローカル入力およびローカル出力へのアクセス。	高速入力および高速出力へのアクセスのみ。

READ_IMM_IN および WRITE_IMM_OUT 命令 (M221 コントローラー)	GetImmediateFastInput および PhysicalWriteFastOutputs ファンクション (M241 コントローラー)
WRITE_IMM_OUT は 1 ビットの書き込みを行います (読み込み機能と似ています)。 WRITE_IMM_OUT はエラーコードを返します。	PhysicalWriteFastOutputs は同時に高速出力に書き込みます。 PhysicalWriteFastOutputs はどの出力に実際書き込まれたかの情報のみを返します。
READ_IMM_IN と GetImmediateFastInput のエラーコードは異なります。	
READ_IMM_IN は入力オブジェクト (%I0.x) の更新を行います。	GetImmediateFastInput は読み込み値のみを返しますが、入力チャンネルの更新は行いません。

注記：HMISCU コントローラーには同等の機能はありません。

Twido 通信機能

以下の Twido の通信機能は変換されません。

- AS インターフェイス
- CANopen
- リモートリンク

これらの通信機能を Twido アプリケーションで使用している場合は、EcoStruxure Machine Expert アプリケーションを手動で設定する必要があります。

変換中に、EcoStruxure Machine Expert アプリケーションのビルドが成功するように、各 I/O オブジェクトに対して変数が 1 つ作成されます。これらの変数は別々のグローバル変数リストにまとめられます。これは置換する変数の識別に役立ちます。

メッセージビューで表示する検出されたエラーおよびアラート

変換中にエラーやアラートが検出されると、メッセージボックスが開き検出されたエラーとアラートの数を表示します。詳細は、**メッセージ** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**プロジェクトの変換**カテゴリーを参照してください。各項目を進行に検証して、アプリケーションを修正する必要があるかどうかを確認してください。

⚠ 警告

装置の意図しない動作

- 変換後のコントローラーのプログラムに目的の設定があり、プロジェクトの変換後に目的のファンクションが機能することを確認してください。
- 運用前に、変換されたプログラムの機能を完全にデバッグ、確認、検証してください。
- プログラムを変換する前に変換元のプログラムが有効であること、例えば、変換元のコントローラーにダウンロードできることを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

- **警告**メッセージは、変換処理でアプリケーションの機能におそらく影響を与えない調整が行われたことを示します。
- **エラー**メッセージは、アプリケーションの一部が変換されなかったことを示します。この場合、変換後のアプリケーションで同じ機能を保持するには、手動でアプリケーションを調整する必要があります。
- アプリケーションプログラムが変換できない機能を使用している場合、変換機能では、サポートしていない言語オブジェクトに対して変数が作成されます。これにより、アプリケーションを正常にコンパイルすることができます。ただし、変換後にサポートされていない機能をしてください。

メッセージビューで表示された情報を保存するには、クリップボードにコピー (CTRL + C) しデータファイルに貼り付け (CTRL + V) ます。

第 6 章
共通デバイスエディターのダイアログ

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
6.1	デバイス設定	80
6.2	I/O マッピング	122

6.1
デバイス設定

このセクションについて

このセクションには次の項目が含まれています。

デバイスエディターについての一般情報

概要

デバイスエディターは**デバイスツリー**で管理されるデバイス設定用パラメーターを提供します。

次の手順に従って特有のデバイスのデバイスエディターを開きます。

- **デバイスツリー**でデバイスをダブルクリックします。または、
- **デバイスツリー**でデバイスを選択し、コンテキストメニュー、または**プロジェクト**メニューから**オブジェクトの編集**コマンドを実行します。

ツール → オプション → デバイスエディターダイアログボックスでは、設定ビューを非表示にすることができます (*103* ページ)。

この章では、メインデバイスエディターについて説明します。バス固有の設定ダイアログについては別に説明します。

デバイスエディターダイアログ

メインダイアログのタイトルにはデバイス名が含まれます。例 : **MyPlc**

デバイスタイプに応じて、デバイスエディターは以下のタブを提供します。

タブ	詳細
コントローラーの選択モードの通信設定 (*83* ページ)	プログラミングシステムとプログラマブルデバイス (コントローラー) 間の接続設定。表示されるタブは、**ツール → オプション → デバイスエディター** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**通信ページ**のパラメーターで選択されたモードによって異なります。
シンプルモードの通信設定 (*98* ページ)	
クラシックモードの通信設定 (*100* ページ)	
設定 (*103* ページ)	デバイスパラメーターの表示または設定。
パラメーター (*104* ページ)	PacDrive コントローラーのデバイスパラメーターの表示または設定。
アプリケーション (*105* ページ)	コントローラーで実行中のアプリケーションのリスト。*プログラム* の章 (*131* ページ) の説明を参照してください。
同期ファイル (*106* ページ)	アプリケーションのダウンロードでコントローラーにダウンロードされたファイルのリスト
ファイル (*107* ページ)	ホストとコントローラー間のファイル転送の設定。
ログ (*108* ページ)	コントローラーログファイルの表示。
PLC 設定 (*110* ページ)	以下の設定。 ● I/O 処理用アプリケーション ● 停止状態の I/O 動作 ● バス周期オプション
ユーザーとグループ (*112* ページ)	ランタイム中にデバイスにアクセスするユーザーの管理。
アクセス権 (*116* ページ)	ランタイムオブジェクトおよびファイルの、特定ユーザーグループ用アクセス権設定。
タスクの配置 (*119* ページ)	定義されたタスクに割り当てられた入力および出力の表示。トラブルシューティングに使用されます。
ステータス (*120* ページ)	デバイス固有のステータスと診断メッセージ
情報 (*121* ページ)	デバイスの一般情報 (例えば、名前、製造元、バージョンなど)。
I/O マッピング (*122* ページ)	プロジェクト (アプリケーション) 上の I/O デバイスの入力および出力チャンネルのマッピング。

タブ	詳細
OPC UA サーバー設定	コントローラーのコンパクトフラッシュ (CF) カード上のサーバー設定ファイル ServerConfig.ini を編集するための多様な機能があります。 OPC UA サーバー設定では以下を行います。 ● 新規サーバー証明書のデフォルトプロパティの定義 ● 既存の証明書の表示および削除 ● サンプリングレートの表示および追加 ● サーバー証明書のエクスポートおよびインポート ● さまざまな設定の有効化と管理 この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。。

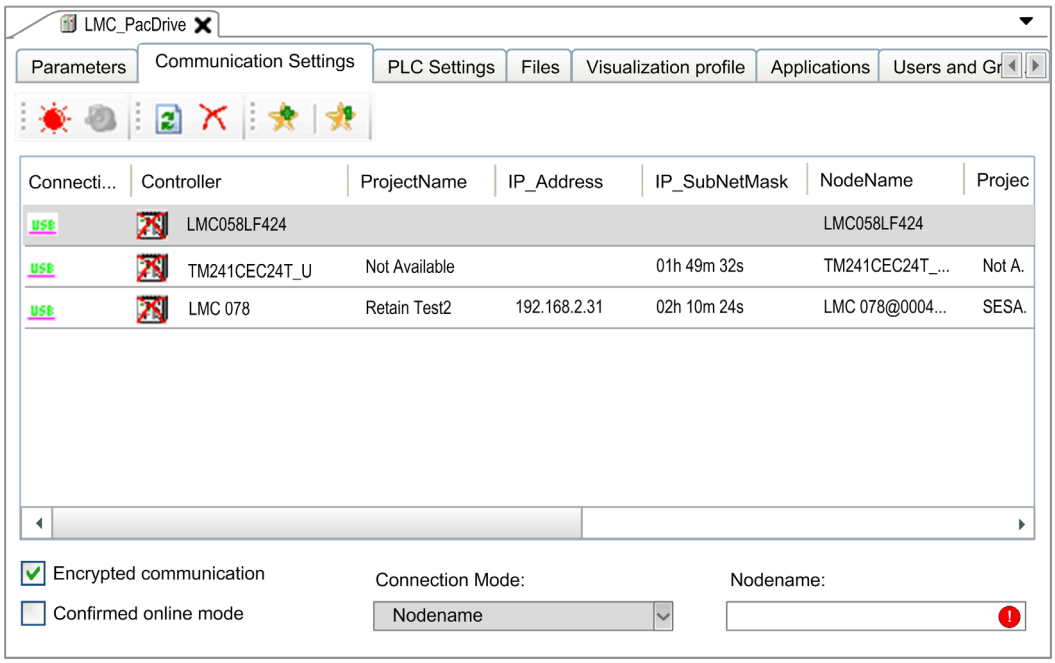
コントローラーの選択モードの通信設定

概略

コントローラーの選択モードの**通信設定**タブは、**コントローラーの選択モードが ツール → オプション → デバイスエディター** ダイアログボックスの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **通信ページ**のパラメーターで選択されているときに表示されます。このタブでは、Ethernet ネットワークをスキャンして利用可能なコントローラーをリスト表示する Network Device Identification サービスへのアクセスを提供します。デバイス (この章では*コントローラー*) とプログラミングシステム間の通信用パラメーターを設定できます。

このコントローラーのリストは EcoStruxure Machine Expert の要求に対する応答を送信したネットワーク内のコントローラーが含まれています。選択したコントローラーがこのリストに含まれないことがあります。これにはいくつかの原因がありえます。原因と適切な解決策については、*コントローラーへのアクセス - トラブルシューティングとよくある質問* (*743* ページ) を参照してください。

コントローラー選択モードにある**通信設定**



通信設定タブには次の要素があります。

- ツールバーのボタン
- 利用可能なコントローラーの情報リスト
- タブ下部にオプション、リスト、テキストボックス

ツールバーのボタンの説明

ツールバーでは以下のボタンが利用できます。

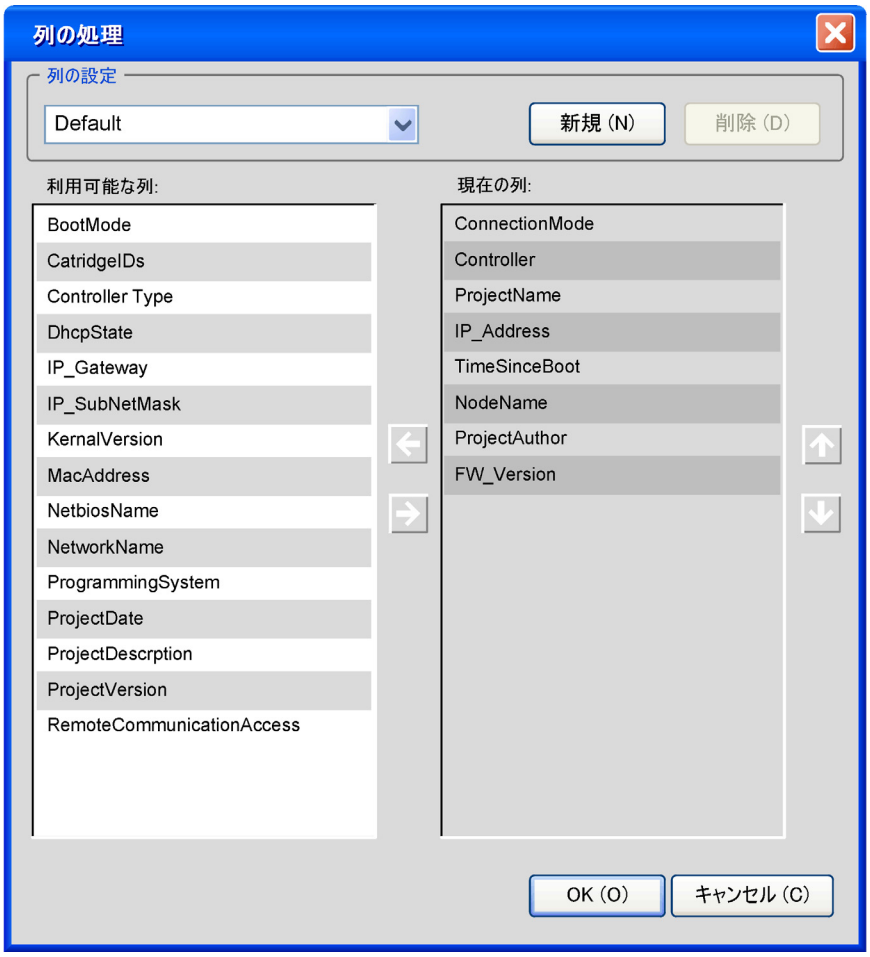
ボタン	詳細
光学的	このボタンをクリックすると選択されたコントローラーが光学的信号を示します。制御 LED がすばやく点滅します。多数のコントローラーを使用している場合に各コントローラーを識別できます。ファンクションは、2 回目のクリックで停止または約 30 秒後に自動的に停止します。 **注記**：光学的信号はこの機能をサポートしているコントローラーだけが発行できます。
光学的と音響的	このボタンをクリックすると選択されたコントローラーが光学的信号と音響信号を示します。ビープ音がなり、制御 LED がすばやく点滅します。多数のコントローラーを使用している場合に各コントローラーを識別できます。 ファンクションは、2 回目のクリックで停止または約 30 秒後に自動的に停止します。 **注記**：光学的と音響的の信号はこの機能をサポートしているコントローラーだけが発行できます。

ボタン	詳細
更新	このボタンをクリックするとコントローラーのリストを更新します。ネットワーク内のコントローラーに要求が送信されます。この要求に応答したコントローラーは、現在値とともにリストされます。 既存のコントローラーのエントリーは新しい要求ごとに更新されます。 新しい要求に対して応答のなかった既存コントローラーは削除されません。それらは、コントローラーのアイコンに赤い十字マークがつき無効と表示されます **更新**ボタンは、リストを右クリックしたときに表示されるコンテキストメニューの**リストのリフレッシュ**コマンドに対応しています。選択されたコントローラーの情報を更新するために、**コントローラーのリフレッシュ**コマンドがコンテキストメニューで提供されています。このコマンドは選択したコントローラーの詳細情報を要求します。 **注記：コントローラーのリフレッシュ**コマンドは他のことローラーの情報を更新することもできます。
リストから有効でない コントローラーを削除	ネットワークスキャンに応答しなかったコントローラーはリストで無効とマークされます。コントローラーのアイコンの赤い十字マークで示されます。このボタンをクリックすると、リストから無効なコントローラーを削除します。 **注記：**コントローラーはそうでない場合でも無効とマークすることができます。 コントローラーを右クリックすると表示されるコンテキストメニューには、コントローラーを削除するための 2 つのコマンドがあります。 ● **リストから選択したコントローラーを削除**コマンドを使用すると、選択されたコントローラーのみを削除することができます。 ● **リストからすべてのコントローラーを削除**コマンドを使用するとリストからすべてのコントローラーを同時に削除することができます。
新規お気に入り ... および **お気に入り 0**	**お気に入り**を使用して、コントローラーの選択を個人的な要件に合わせることができます。ネットワーク上の多くのコントローラーの追跡ができます。 **お気に入り**は、固有の識別子によってコントローラーのコレクションを表します。 お気に入りボタン (例 : お気に入り **0**) をクリックして、選択または選択を解除します。お気に入りを選択していない場合は、検知されたコントローラーはすべて表示されます。 コンテキストメニューから**お気に入り**にアクセスすることも可能です。コンテキストメニューは、リスト内のコントローラーを右クリックすると開きます。 ツールバーのお気に入りボタンにカーソルを移動すると、ツールチップに関連するコントローラーが表示されます。

コントローラーリスト

デバイスエディターの**通信設定**タブの中央にあるコントローラーリストには、ネットワークスキャンに対する応答を送信したコントローラーが表示されます。各コントローラーの情報が複数の列に表示されます。個々の要求に応じて、コントローラーリストに表示された列を調整することができます。

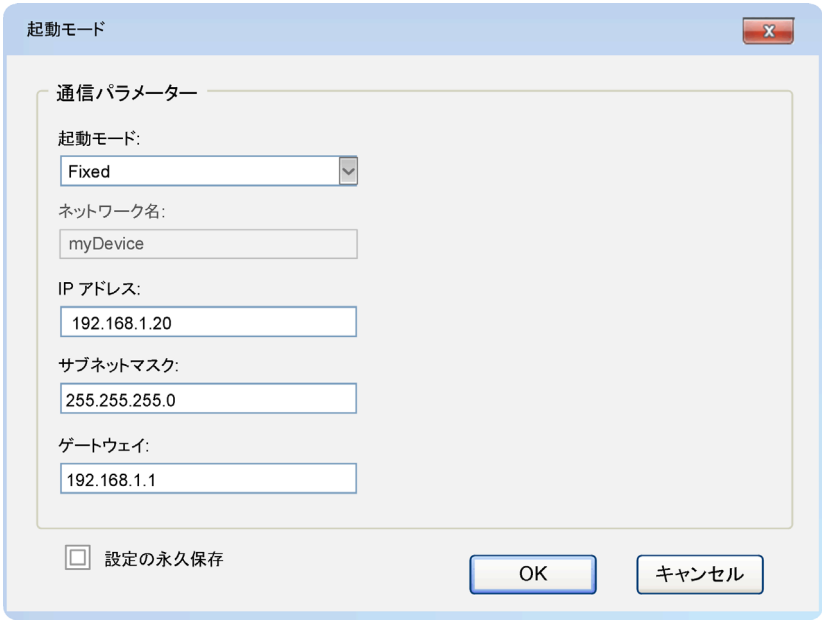
これを行うには、列のヘッダーを右クリックして**列の処理**ダイアログボックスを開きます。



このテーブルの独自のレイアウトを作成します。**新規**をクリックしてレイアウトの名前を入力します。横向きの矢印ボタンをクリックして、**利用可能な列**リストから**現在の列**リストへ、またその逆に列を移動します。上矢印と下矢印ボタンをクリックして、**現在の列**リストの順番を変更します。

通信の設定

下記の手順に従って、プログラミングシステムとコントローラー間の通信用パラメーターを設定します。

手順	手順内容
1	コントローラーリストからコントローラーを選択します。
2	コントローラーを右クリックして**通信設定の編集 ...** をコンテキストメニューから実行します。 **結果：T 通信設定の編集 ...** ダイアログボックスが開き、コントローラーの現在の設定値が表示されます。  **注記：**多くのコントローラーは、コントローラーの通信パラメーターの変更を防ぐためのパラメーター (**RemoteAccess** など) を提供しています。

手順	手順内容
3	通信パラメーターを設定します。 ● **起動モード** ❍ **固定** : 固定 IP アドレスは以下で入力された値 (**IP アドレス**、**サブネットマスク**、**ゲートウェイ**). に従って使用されます。 ❍ **BOOTP**: IP アドレスは、BOOTP (bootstrap protocol、ブートストラッププロトコル) によって動的に受信されます。下の値は無視されます。 ❍ **DHCP**: IP アドレスは、DHCP (dynamic host configuration protocol、ダイナミック ホスト コンフィギュレーション プロトコル) によって動的に受信されます。下の値は無視されます。 **注記：**すべてのデバイスが BOOTP および / または DHCP をサポートしているわけではありません。 ● **IP アドレス** IP アドレスを設定するときは、次の警告メッセージを参照してください。 このボックスはコントローラーの IP アドレスを含んでいます。ピリオドで区切られた 4 組の数字 (範囲 : 0 ~ 255) で構成される固有のアドレスです。IP アドレスはこの (サブ) ネットワーク内で固有にしてください。 ● **サブネットマスク** サブネットマスクは、コントローラーが属するネットワークセグメントを指定します。ピリオドで区切られた 4 組の数字 (範囲 : 0 ~ 255) で構成される固有のアドレスです。通常、標準サブネット番号には 0 または 255 のみが使用されます。ただし、他の数値も使用可能です。サブネットマスクの値は、通常、ネットワーク内のすべてのコントローラーで同じです。 ● **ゲートウェイ** ゲートウェイアドレスはコントローラーと同じネットワークにあるローカル IP ルーターのアドレスです。IP ルーターはローカルネットワーク外のあて先にデータを渡します。ピリオドで区切られた 4 組の数字 (範囲 : 0 ~ 255) で構成される固有のアドレスです。ゲートウェイの値は、通常、ネットワーク内のすべてのコントローラーで同じです。 ● 再起動後も通信設定をコントローラーに保存するには、**設定の永久保存**オプションを有効にします。
4	**OK** をクリックしてコントローラーに設定を転送します。

ネットワークの各デバイスには固有のアドレスが必要なため、IP アドレスは慎重に管理してください。複数のデバイスに同じ IP アドレスを設定すると、ネットワークおよび関連する機器が意図しない動作をする可能性があります。

⚠ 警告

装置の意図しない動作

- すべてのデバイスが固有のアドレスであることを確認してください。
- システム管理者から IP アドレスを取得してください。
- システムを動作させる前に、デバイスの IP アドレスが固有であることを確認してください。
- ネットワーク上の他の機器に同じ IP アドレスを割り当てないでください。
- Ethernet 通信を含むアプリケーションを複製した後は、IP アドレスを固有のアドレスに更新してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

お気に入りの管理

次の手順に従ってコントローラーリストのお気に入りを管理します。

手順	手順内容
1	コントローラーリストからコントローラーを選択します。
2	コントローラーを右クリックし、次のいずれかのコマンドを選択します。 ● **新規お気に入り**を選択して新規お気に入りグループを作成します。 ● **お気に入り n** を選択して以下を行います。 ❍ 選択したコントローラーをこのお気に入りのリストに追加します。 ❍ 選択したコントローラーをこのお気に入りのリストから削除します。 ❍ すべてのコントローラーをこのお気に入りのリストから削除します。 ❍ お気に入りの選択 ❍ お気に入りの名前の変更 ❍ お気に入りの削除

暗号化通信オプション

暗号化通信オプションを選択すると、コントローラーへの通信は暗号化されます。

注記：
コントローラーとの暗号化通信を実行するには、次の前提条件を満たしている必要があります。
- コントローラーが TLS (Transport Layer Security) をサポートしていること。
- コントローラーに証明書が発行されていること。

TLS のサポートに関する情報は、各コントローラーについては、*プログラミングガイド*を参照してください。をご覧ください。

暗号化された通信を使用してログインを試みる際、以下の状況が考えられます。

条件	結果	コメント
コントローラーが TLS をサポートしていない (前提条件 1 が満たされていない) 場合	コントローラーにログインを試みると、TLS をサポートしていないことを示すメッセージが表示されます。	ログインは拒否されます。
コントローラーに発行されていない (前提条件 2 が満たされていない) 場合	コントローラーにログインを試みると、通信の暗号化が正しく行われなかったことを示すメッセージが表示されます。	ログインは拒否されます。
上記の前提条件が両方とも満たされている場合	コントローラーへの初回ログイン時に、EcoStruxure Machine Expert を起動する PC が所在する**コントローラ証明書**のローカルストアーに、(非信頼の) コントローラーの証明書をインストールするよう求めるメッセージが表示されます。	**OK** をクリックして承認すると、 ● ログインに成功し、必要に応じてユーザーパスワードが提供されます。 ● コントローラーへの通信は暗号化されます。 ● メッセージは表示されなくなります。 **キャンセル**をクリックすると、 ● ログインは拒否されます。 ● メッセージは、ログインを試みるたびに表示されます。

オンラインモード確認済み オプション

オンラインモード確認済み オプションを有効にすると EcoStruxure Machine Expert は以下のオンラインコマンドのいずれかが選択されたときに、確認を要求するメッセージを表示します。**値の強制、ログイン、複数ダウンロード、強制リストの開放、シングルサイクル、開始、停止、値の書き込み**。**オンラインモード確認済み**オプションを無効にして、このメッセージの表示を削除するには、このオプションのチェックをクリアします。

固有のデバイス名 (NodeNames) の指定

NodeName という用語はデバイス名という用語の同義語として使用されます。ノード名は、ネットワークスキャン後にコントローラーを識別するためにも使用されるため、IP アドレスと同様に慎重に管理をし、各ノード名はネットワーク内で固有であることを確認してください。複数のデバイスに同じノード名が割り当てられていると、ネットワークや関連する機器が予期せぬ動作をする可能性があります。

⚠ 警告

装置の意図しない動作

- すべてのデバイスに固有のノード名が割り当てられていることを確認してください。
- システムを動作させる前に、デバイスのノード名が固有であることを確認してください。
- ネットワーク上の他の機器に同じノード名を割り当てないでください。
- Ethernet 通信を含むアプリケーションを複製した後は、固有のノード名に変更してください。
- M241 および M251 コントローラーなど、自動的にノード名を作成しないデバイスには固有のノード名を作成してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

コントローラーのタイプによって、**NodeName** (デバイス名) の自動作成の手順は異なります。固有の名前を作成するために、IP アドレスを使用するコントローラーもあれば Ethernet アダプターの MAC アドレスを使用するコントローラーもあります。この場合、名前を変更する必要はありません。

以下のように固有のデバイス名 (**NodeName**) を割り当てることもできます。

手順	手順内容
1	リスト内のコントローラーを右クリックし、コンテキストメニューから**デバイス名の変更 ...** コマンドを実行します。 **結果** : **デバイス名の変更**ダイアログボックスが開きます。
2	**デバイス名の変更**ダイアログボックスで、固有のデバイス名を**新規**ボックスに入力します。
3	**OK** ボタンをクリックして確定します。 **結果** : 入力したデバイス名はコントローラーに割り当てられ、リストの **NodeName** 列に表示されます。 **注記 :** デバイス名と **NodeName** は同義語です。

接続モードの指定

通信設定タブの左下にある**接続モード**リストでは、**アドレス**フィールドに入力する接続アドレスのフォーマットを選択することができます。

次のフォーマットがサポートされています。

- 自動 (*89* ページ)
- ノード名 (*90* ページ)
- IP アドレス (*90* ページ)
- IP アドレス (高速 TCP) (*90* ページ)
- NAT 経由のノード名 (リモート TCP) (*91* ページ) (NAT = network address translation、ネットワークアドレス変換)
- NAT 経由の IP アドレス (リモート TCP) (*92* ページ)
- ゲートウェイ経由のノード名 (*93* ページ)
- ゲートウェイ経由の IP アドレス (*95* ページ)
- モデム経由のノード名 (*97* ページ)
- IP Address (PacDriveM のみ) (*90* ページ) (Controller Assistant のようなサービスツールでのみ利用可能)

注記 : 接続モードの変更後は、選択したコントローラーにアクセスするためのログイン手順を 2 回行う必要があることがあります。

接続モード 自動

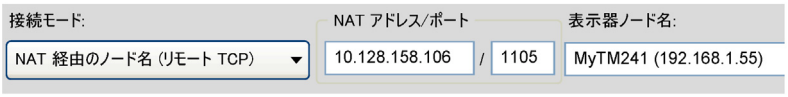
自動を**接続モード**リストから選択すると、ノード名、IP アドレス、または接続 URL (uniform resource locator、ユニフォームリソースロケーター) を選択して**アドレス**を指定することができます。

注記 : アドレスの先頭または末尾にスペースは使用しないでください。

別の**接続モード**を選択して**アドレス**をこのモード用に指定してある場合は、**接続モード → 自動**に切り替えても指定した**アドレス**をボックスで使用できます。

例

接続モード → NAT 経由のノード名 (リモート TCP) が選択され、アドレスとノード名が指定されています。



接続モード → 自動に切り替えると、情報は、接頭辞が enodename3:// で始まる URL に変換されます。

接続モードで IP アドレスが入力されている場合、情報は、接頭辞で始まる URL に変換されます。**接続モード → IP Address** には、接頭辞 etcp3:// が使われます。**接続モード → IP Address (Fast TCP)** には、接頭辞 etcp4:// が使われます。例 : etcp4://<IpAddress>

注記 : Controller Assistant と診断ツールでは IP アドレスに接頭辞 etcp2:// を追加できます。これは PacDrive M のコントローラーのみで可能です。

接続モードでノード名が入力されている場合 (例 : **接続モード → Nodename** が選択されている場合)、情報は、接頭辞が enodename3:// で始まる URL に変換されます。例、enodename3://<Nodename>。

接続モード → ノード名

Nodename オプションを**接続モード**リストで選択した場合、コントローラーのノード名を入力して**アドレス**を指定することができます。テキストボックスは、コントローラーリストのコントローラーをダブルクリックすると自動的に入力されます。

例 : **ノード名 :** MyM238 (10.128.158.106)

選択したコントローラーがノード名を提供しない場合は、**接続モード**が自動的に **IP アドレス**に切り替わり、リストからの IP アドレス**アドレス**ボックスに入力されます。

接続モード → IP アドレス

IP アドレスオプションを**接続モード**リストで選択した場合、コントローラーの IP アドレスを入力して**アドレス**を指定することができます。ボックスは、コントローラーリストのコントローラーをダブルクリックすると自動的に入力されます。

例 : **IP アドレス :** 190.201.100.100

選択したコントローラーが IP アドレスを提供しない場合は、**接続モード**が自動的に**ノード名**に切り替わり、リストからのノード名が**アドレス**ボックスに入力されます。

注記 : IP アドレスは <Number>.<Number>.<Number>.<Number> の形式で入力してください。

接続モード → IP アドレス (高速 TCP)

IP アドレス (高速 TCP) オプションを**接続モード**リストから選択すると、TCP プロトコルを使用してコントローラーに接続できます。コントローラーの**表示器の IP アドレス / ポート** をそれぞれ入力します。ネットワークアドレス変換 (NAT) を使用している場合は、**ポート**のデフォルト設定 **11740** を適用できます。

例 : **IP アドレス :** 190.201.100.100

注記 : IP アドレスは <Number>.<Number>.<Number>.<Number> の形式で入力してください。

コントローラーがリストに表示されない場合は、**テスト**ボタンをクリックします。コントローラーがネットワークスキャンに応答すると、そのエントリーがコントローラーのリストに追加されます。このエントリーは、最初の列に **TCP** アイコンが表示されます。

注記 : ご使用のコントローラー専用の この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。で **IP アドレス (高速 TCP)** 接続モードに対応しているかご確認ください。

接続モード → IP アドレス (PacDriveM のみ)

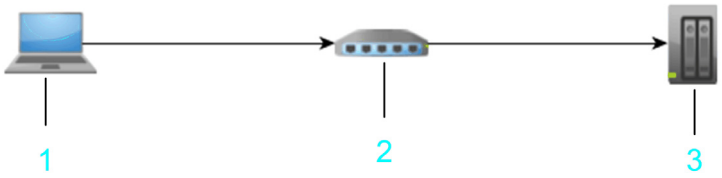
IP アドレス (PacDriveM のみ) オプションを**接続モード**リストで選択した場合、コントローラーの IP アドレスを入力して**アドレス**を指定することができます。ボックスは、コントローラーリストの PacDrive M コントローラーをダブルクリックすると自動的に入力されます。

例 : **IP アドレス :** 190. 201. 100. 100

注記 : IP アドレスは <Number>. <Number>. <Number>. <Number> の形式で入力してください。

接続モード → NAT 経由のノード名 (リモート TCP)

NAT 経由のノード名 (リモート TCP) オプションを**接続モード**リストで選択した場合、ネットワーク内の NAT ルーターの背後にあるコントローラーのアドレスを指定できます。コントローラーのノード名、および NAT ルーターの IP アドレス、またはホスト名とポートを入力します。



1 PC
2 NAT ルーター
3 ターゲットデバイス

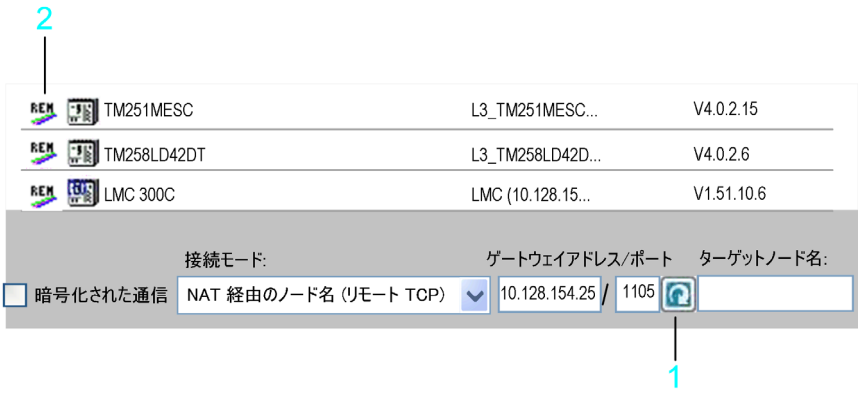
例 : **NAT アドレス / ポート :** 10. 128. 158. 106/1105 **表示器ノード名 :** MyM238 (10. 128. 158. 106)

注記 : 有効な IP アドレス (形式 <Number>. <Number>. <Number>. <Number>) または **NAT アドレス**の有効なホスト名を入力します。
使用する NAT ルーターのポートを入力します。それ以外はデフォルトポート **1105** が使用されます。

入力した情報は、TCP ブロックドライバーを使用してリモート TCP ブリッジを作成する URL として解釈され、ローカルゲートウェイの指定されたノード名のコントローラーをスキャンして接続します。

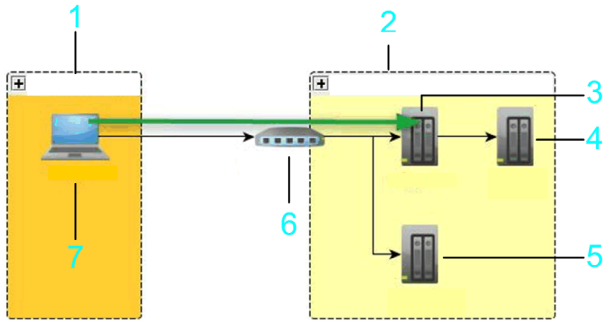
注記 : NAT ルーターは、対象コントローラーにも設置できます。それを使用してコントローラーへの TCP ブリッジを作成することができます。

リモートコントローラー (ブリッジコントローラー) を使用してリモートネットワークをスキャンすることもできます。これをするには、**NAT アドレス / ポート**を入力し **NAT アドレス / ポート**テキストフィールドの右側のリフレッシュボタンをクリックします。リモートネットワークスキャンに応答したコントローラーはコントローラーリストに表示されます。このらエントリーは、最初の列に **REM** アイコンが表示されます。リストに詳細情報を表示するには、各コントローラーを右クリックし**コントローラーのリフレッシュ**コマンドを実行します。コントローラーがこの機能に対応している場合、詳細情報がリストに表示されます。各コントローラーについては、*プログラミングガイド*を参照してください。



1 リフレッシュボタン
2 **REM** アイコン

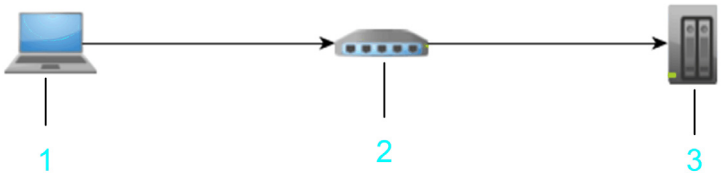
次の例では、**ブリッジコントローラー**、**コントローラー 2**、および**コントローラ 3** がスキャンされます。



1 ローカルサブネット
2 リモートサブネット
3 ブリッジコントローラー
4 コントローラー 3
5 コントローラー 2
6 NAT ルーター
7 PC

接続モード → NAT 経由の IP アドレス (リモート TCP)

NAT 経由の IP アドレス (リモート TCP) (NAT = network address translation、ネットワークアドレス変換) オプションプを**接続モード**リストで選択した場合、ネットワーク内の NAT ルーターの背後にあるコントローラーのアドレスを指定できます。コントローラーの IP アドレス、および NAT ルーターの IP アドレス、またはホスト名とポートを入力します。



1 PC
2 NAT ルーター
3 ターゲットデバイス

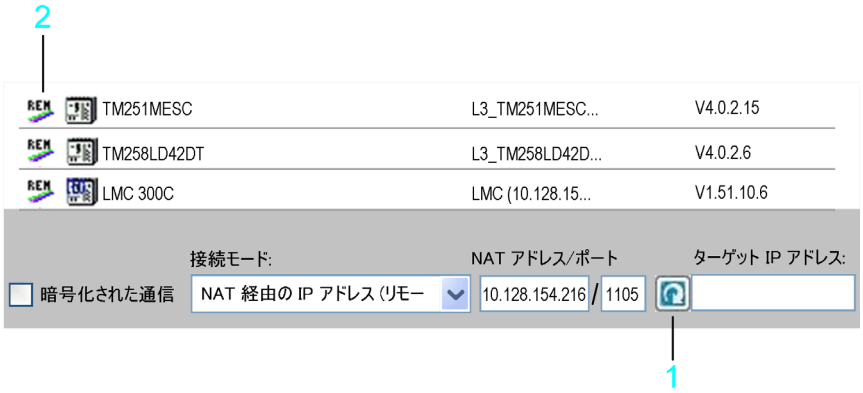
例 : **NAT アドレス / ポート :** 10.128.154.206/1105 **表示器の IP アドレス :** 192.168.1.55

注記 : 有効な IP アドレス (形式 <Number>.<Number>.<Number>.<Number>) または **NAT アドレス**の有効なホスト名を入力します。
使用する NAT ルーターのポートを入力します。それ以外はデフォルトポート **1105** が使用されます。
対象 IP アドレスに有効な IP アドレス (形式 <Number>.<Number>.<Number>.<Number>) を入力します。

入力した情報は、TCP ブロックドライバーを使用してリモート TCP ブリッジを作成する URL として解釈され、ローカルゲートウェイの指定されたノード名のコントローラーをスキャンして接続します。IP アドレスは、ノード名 (例えば、MyController (10.128.154.207)) またはゲートウェイのスキャンされた各デバイスのサービスを呼び出すことで検索されます。

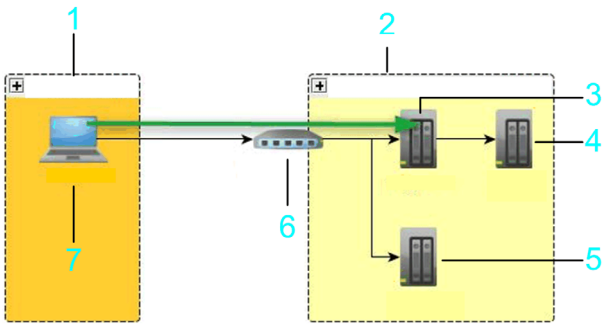
注記 : NAT ルーターは、対象コントローラーにも設置できます。それを使用してコントローラーへの TCP ブリッジを作成することができます。

リモートコントローラー (ブリッジコントローラー) を使用してリモートネットワークをスキャンすることもできます。これをするには、**NAT アドレス / ポート**を入力し **NAT アドレス / ポート**テキストフィールドの右側のリフレッシュボタンをクリックします。リモートネットワークスキャンに応答したコントローラーはコントローラーリストに表示されます。このらエントリーは、最初の列に **REM** アイコンが表示されます。リストに詳細情報を表示するには、各コントローラーを右クリックし**コントローラーのリフレッシュ**コマンドを実行します。コントローラーがこの機能に対応している場合、詳細情報がリストに表示されます。各コントローラーについては、*プログラミングガイド*を参照してください。



1 リフレッシュボタン
2 **REM** アイコン

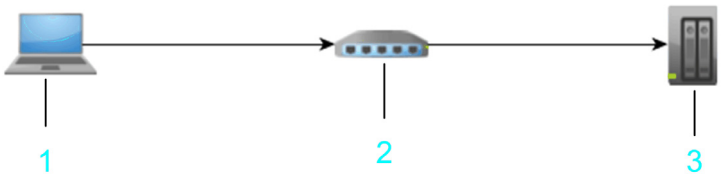
次の例では、**ブリッジコントローラー**、**コントローラー 2**、および**コントローラ 3** がスキャンされます。



1 ローカルサブネット
2 リモートサブネット
3 ブリッジコントローラー
4 コントローラー 3
5 コントローラー 2
6 NAT ルーター
7 PC

接続モード → ゲートウェイ経由のノード名

ゲートウェイ経由のノード名オプションを**接続モード**リストで選択した場合、ネットワーク内の EcoStruxure Machine Expert ゲートウェイルーターの背後、または近くにあるコントローラーのアドレスを指定できます。コントローラーのノード名、および EcoStruxure Machine Expert ゲートウェイルーターの IP アドレス、またはホスト名とポートを入力します。



1 PC / HMI
2 EcoStruxure Machine Expert ゲートウェイがインストールされた PC / HMI / デバイス
3 ターゲットデバイス

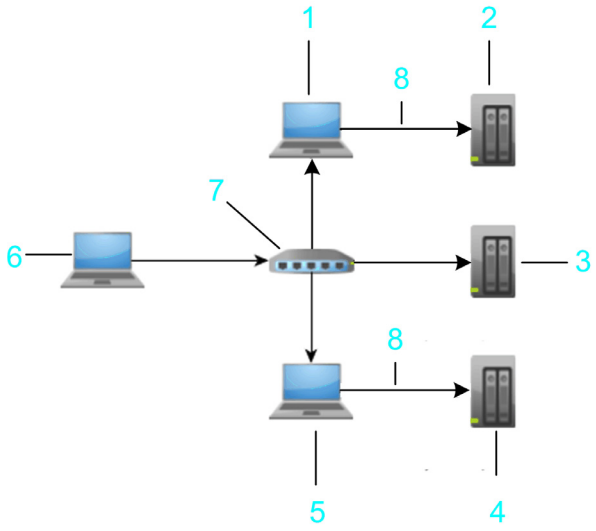
例 : **ゲートウェイアドレス / ポート :** 10. 128. 156. 28/1217 **表示器ノード名 :** MyPLC

注記 : 有効な IP アドレス (形式 <Number>. <Number>. <Number>. <Number>) または**ゲートウェイアドレス / ポート** の有効なホスト名を入力します。
使用するゲートウェイルーターのポートを入力します。入力しない場合は、デフォルトの EcoStruxure Machine Expert ゲートウェイポート **1217** が使用されます。
先頭または末尾にスペースは使用しないでください。また **対象ノード名**のボックスにはコンマを使用しないでください。

入力した情報は URL として解釈されます。入力したノード名でゲートウェイに直接接続しているデバイスをスキャンします。直接接続しているとは、EcoStruxure Machine Expert ゲートウェイトポロジーでは、それ自身がルートノードであるかルートノードの子ノードであることを意味します。

注記：EcoStruxure Machine Expert ゲートウェイは HMI、送信先の PC、またはローカル PC に設置することができ、EcoStruxure Machine Expert ネットワークの背後にあるサブネットに存在する固有のノード名をもたないデバイスに接続することができます。

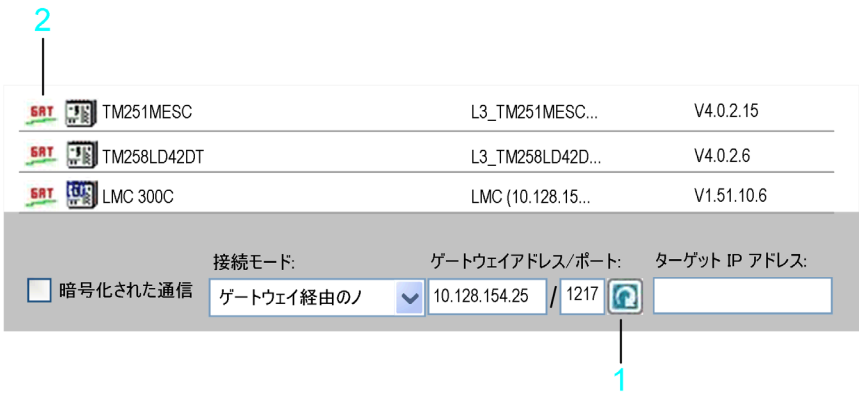
下図は、ホップ PC2 (図の項目 5) (EcoStruxure Machine Expert ゲートウェイがインストールされている必要あり) のアドレスを使用して PCI から対象コントローラー 3 (図の項目 4) への接続を許可する例を示しています。



1 ホップ PC 1
2 対象コントローラー 1: MyNotUniqueNodename
3 対象コントローラー 2: MyNotUniqueNodename
4 対象コントローラー 3: MyNotUniqueNodename
5 ホップ PC 2
6 PC / HMI
7 ルーター
8 Ethernet

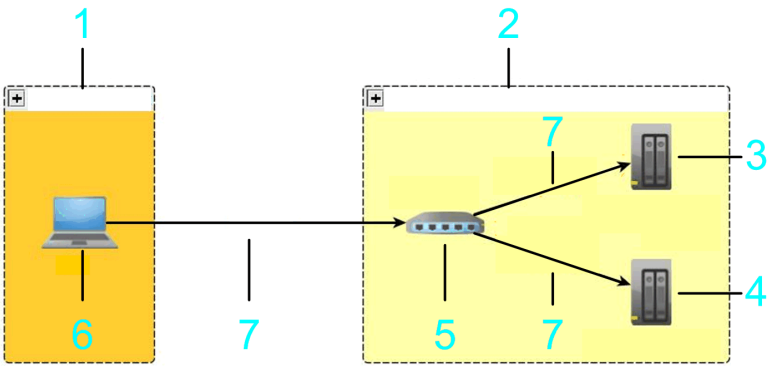
特定のコントローラーとの接続確立を検証するには、**ゲートウェイアドレス / ポート**を入力し**テスト**ボタンをクリックします。コントローラーがネットワークスキャンに応答すると、そのエントリーがコントローラーのリストに追加されます。このエントリーは、最初の列に **GAT** アイコンが表示されます。

特定のゲートウェイをスキャンしてコントローラーを検知するには、**ゲートウェイアドレス / ポート**を入力し**ゲートウェイアドレス / ポート**テキストフィールド右側のリフレッシュボタンをクリックします。ゲートウェイスキャンに応答したコントローラーはコントローラーリストに表示されます。このらエントリーは、最初の列に **GAT** アイコンが表示されます。リストに詳細情報を表示するには、各コントローラーを右クリックし**コントローラーのリフレッシュ**コマンドを実行します。コントローラーがこの機能に対応している場合、詳細情報がリストに表示されます。各コントローラーについては、*プログラミングガイド*を参照してください。



1 リフレッシュボタン
2 **GAT** アイコン

スキャンするゲートウェイは、ローカルまたはリモートサブネット内の PC または HMI に配置することができます。次の例では、ブリッジ**対象コントローラー 1** および**対象コントローラー 2** がスキャンされます。

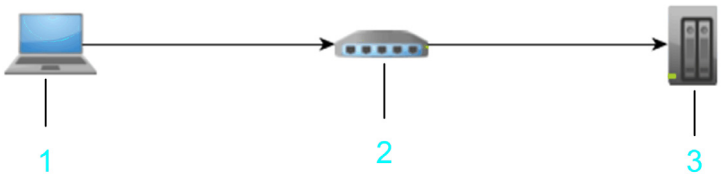


1 ローカルサブネット
2 リモートサブネット
3 対象コントローラー 1:
4 対象コントローラー 2:
5 ゲートウェイ
6 PC
7 Ethernet

ゲートウェイを使用してリストされているデバイスに接続できます。

接続モード → ゲートウェイ経由の IP アドレス

ゲートウェイ経由の IP アドレスオプションを**接続モード**リストで選択した場合、ネットワーク内の EcoStruxure Machine Expert ゲートウェイルーターの背後、または近くにあるコントローラーのアドレスを指定できます。コントローラーの IP アドレス、および EcoStruxure Machine Expert ゲートウェイルーターの IP アドレス、またはホスト名とポートを入力します。



1 PC / HMI
2 EcoStruxure Machine Expert ゲートウェイがインストールされた PC / HMI / デバイス
3 ターゲットデバイス

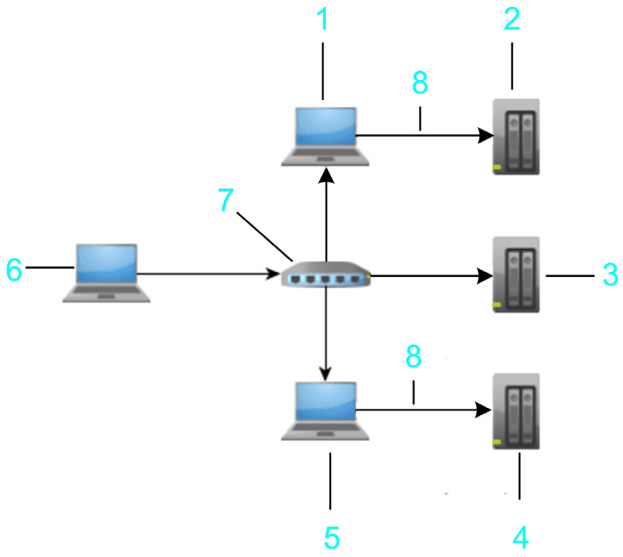
例 : **ゲートウェイアドレス / ポート :** 10.128.156.28/1217 **表示器の IP アドレス :** 10.128.156.222

注記 : 有効な IP アドレス (形式 <Number>.<Number>.<Number>.<Number>) または**ゲートウェイアドレス / ポート** の有効なホスト名を入力します。
使用するゲートウェイルーターのポートを入力します。入力しない場合は、デフォルトの EcoStruxure Machine Expert ゲートウェイポート **1217** が使用されます。
対象 IP アドレスに有効な IP アドレス (形式 <Number>.<Number>.<Number>.<Number>) を入力します。

入力した情報は URL として解釈されます。入力した IP アドレスでゲートウェイをスキャンしデバイスを検索します。IP アドレスは、ノード名 (例えば、MyController (10.128.154.207)) またはゲートウェイのスキャンされた各デバイスのサービスを呼び出すことで検索されます。

注記 : EcoStruxure Machine Expert ゲートウェイは、HMI、送信先の PC、またはローカル PC に配置することができます。EcoStruxure Machine Expert ネットワークの背後にあるサブネットに存在する固有のノード名をもたないデバイスに接続することができます。

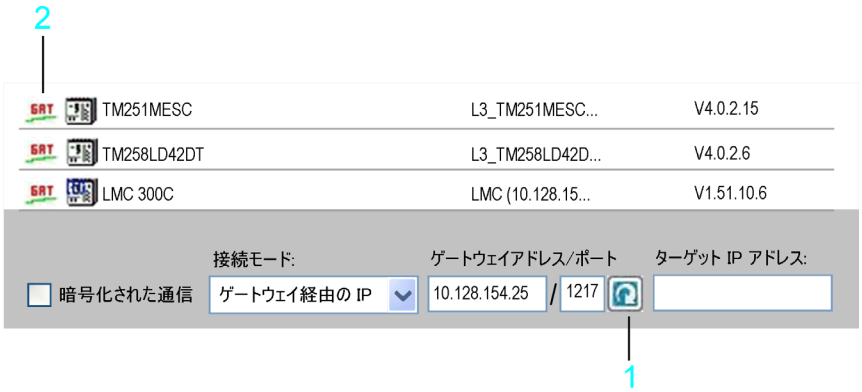
下図は、ホップ PC2 (図の項目 5) (EcoStruxure Machine Expert ゲートウェイがインストールされている必要あり) から対象コントローラー 3 (図の項目 4) への接続を許可する例を示しています。



1 ホップ PC 1
2 対象コントローラー 1: 10.128.156.20
3 対象コントローラー 2: 10.128.156.20
4 対象コントローラー 3: 10.128.156.20
5 ホップ PC 2
6 PC
7 ルーター
8 Ethernet

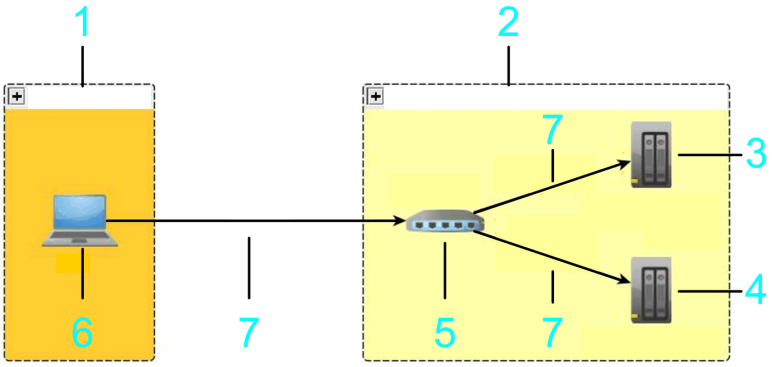
特定のコントローラーとの接続確立を検証するには、**ゲートウェイアドレス / ポート**を入力し**テスト**ボタンをクリックします。コントローラーがネットワークスキャンに応答すると、そのエントリーがコントローラーのリストに追加されます。このエントリーは、最初の列に **GAT** アイコンが表示されます。

特定のゲートウェイをスキャンしてコントローラーを検知するには、**ゲートウェイアドレス / ポート**を入力し**ゲートウェイアドレス / ポート**テキストフィールド右側のリフレッシュボタンをクリックします。ゲートウェイスキャンに応答したコントローラーはコントローラーリストに表示されます。このらエントリーは、最初の列に **GAT** アイコンが表示されます。リストに詳細情報を表示するには、各コントローラーを右クリックし**コントローラーのリフレッシュ**コマンドを実行します。コントローラーがこの機能に対応している場合、詳細情報がリストに表示されます。各コントローラーについては、*プログラミングガイド*を参照してください。



1 リフレッシュボタン
2 **GAT** アイコン

スキャンするゲートウェイは、ローカルまたはリモートサブネット内の PC または HMI に配置することができます。次の例では、ブリッジ**対象コントローラー 1** および**対象コントローラー 2** がスキャンされます。

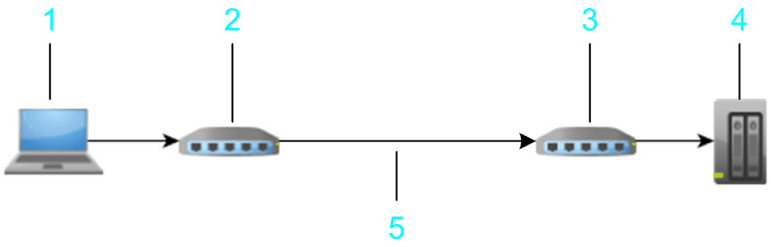


1 ローカルサブネット
2 リモートサブネット
3 対象コントローラー 1:
4 対象コントローラー 2:
5 ゲートウェイ
6 PC
7 Ethernet

ゲートウェイを使用してリストされているデバイスに接続できます。

接続モード → モデム経由のノード名

モデム経由のノード名オプションを**接続モード**リストで選択した場合、モデムラインの背後にあるコントローラーを指定できます。



1 PC
2 PC / MODEM
3 対象モデム
4 ターゲットデバイス
5 電話回線

モデム接続を確立するには、**モデム → 接続**ボタンをクリックします。**モデム設定**ダイアログボックスで、対象モデムの**電話番号**を入力し通信設定を設定します。**OK** をクリックして確認しモデムへの接続を確立します。

EcoStruxure Machine Expert ゲートウェイが停止して再起動すると、ローカルゲートウェイの接続はすべて終了します。EcoStruxure Machine Expert は再起動前に確認が必要なメッセージを表示します。

モデムへの接続の確立が成功すると、**モデム** ボタンが**接続**から**切断**に切り替わります。モデムに接続されたコントローラーがスキャンされ、コントローラーリストが更新されます。コントローラーリストから項目をダブルクリックするか**対象ノード名**ボックスにノード名を入力して特定のコントローラーに接続することができます。

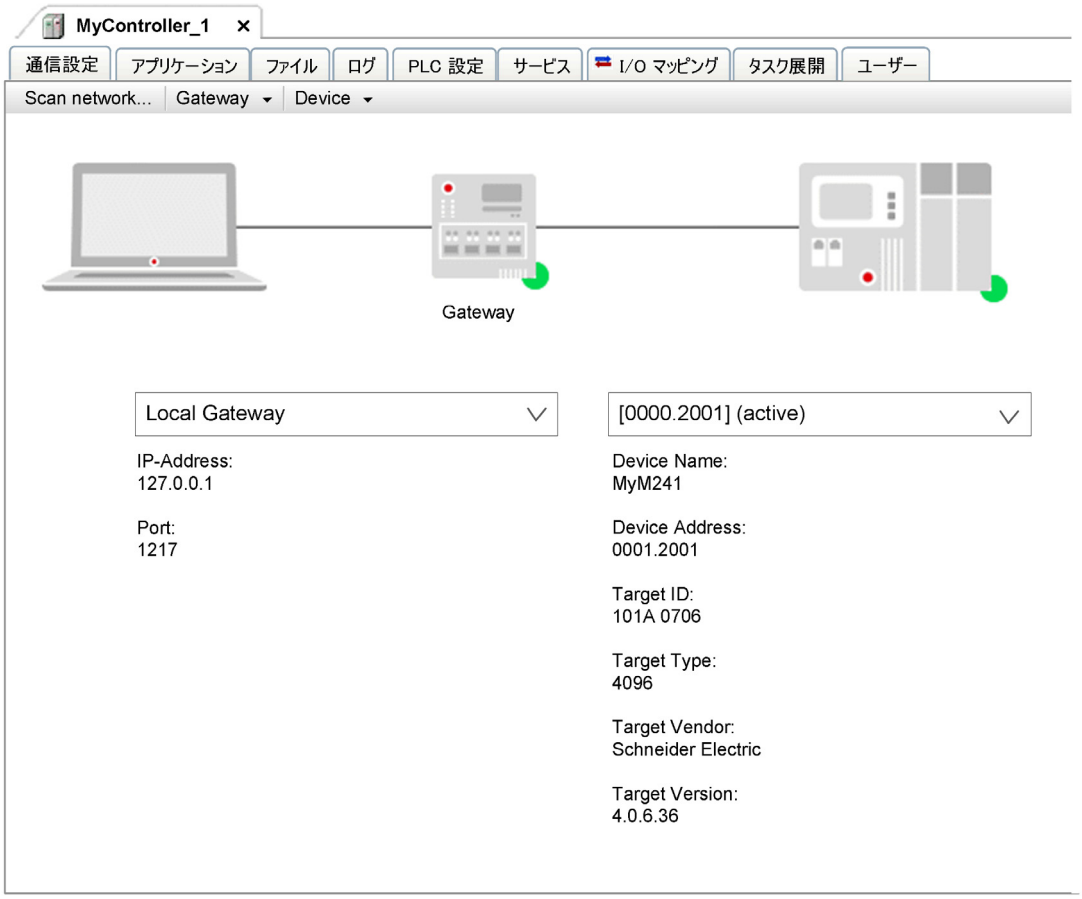
モデム → 切断ボタンをクリックしてモデム接続を終了し EcoStruxure Machine Expert ゲートウェイを停止、再起動します。Ethernet ネットワークに接続されたコントローラーがスキャンされ、コントローラーリストが更新されます。

シンプルモードの通信設定

概要

シンプルモードの**通信設定**タブは、**シンプルモード**が **ツール → オプション → デバイスエディター** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**通信ページ**のパラメーターで選択されているときに表示されます。デバイスとプログラミングシステムの通信パラメーターを設定するためのグラフィックビューを提供します。

シンプルモードの**通信設定**タブには、プログラミングデバイス、現在のゲートウェイ、および対象デバイスの図が接続ステータスとともに表示されます。



選択フィールドでゲートウェイと対象デバイスを選択します。選択できるエントリーのリストは **Manage gateways** および **Manage favorite devices** のパラメーターによって異なります。

対象デバイスを別の方法で入力できます。

- IP アドレス、例 : **192.168.101.109**
- デバイスアドレス、例 : **[056D]**
- デバイス名、例 : **MyDevice**

注記：デバイス名でデバイスを検索する場合、ネットワーク内で一意のデバイス名が必要です。

ゲートウェイシンボルの右下のステータスビュレットは通信状態を示します。

色指定	詳細
赤	接続が確立できません。
緑	接続が確立されました。
黒	接続ステートが未定義です。

注記：一部の通信プロトコルでは、ゲートウェイの定期的な確認ができません。よって、ステータスが表示されません。

対象デバイスのステータスビュレットをクリックして、デバイスのネットワークスキャンを開始します。ただし、ゲートウェイの検索が開始していない場合にのみ可能です。

要素の説明

シンプルモードの**通信設定**タブの要素 :

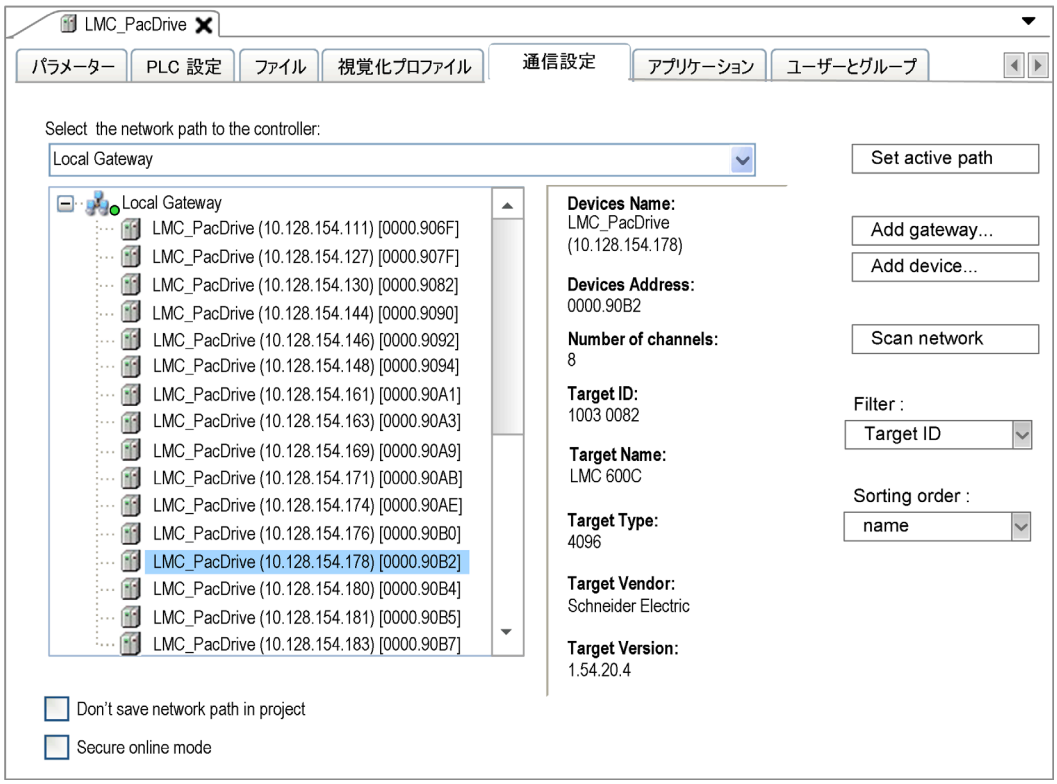
要素		詳細
Scan network... ボタン		**Select Device** ダイアログボックスを開き、設定されたゲートウェイと関連デバイスをリストします。
ゲートウェイリスト		–
	新規ゲートウェイの追加 ...	**ゲートウェイ**ダイアログボックスを開き、新規ゲートウェイを追加します *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
	ゲートウェイの管理 ...	**Manage gateways** ダイアログボックスを開き、すべてのゲートウェイの概要を表示します。このダイアログボックスでゲートウェイを追加、または削除できます。ボタンを使用してゲートウェイのエントリーの順番を変更できます。
	Configure local gateway...	**ゲートウェイ設定** ダイアログボックスを開きます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。ローカルゲートウェイのブロックドライバーを設定できます。
デバイスリスト		–
	Add current device to favorites	定義されたデバイスをお気に入りのデバイスリストに追加します。
	Manage favorite devices...	お気に入りのデバイスリストを表示するダイアログボックスを開きます。このダイアログボックスでデバイスを追加、削除、またはエントリーの順番を変更できます。リストの一番上にあるデバイスがデフォルトデバイスになります。
	Rename active device...	デバイスの名前を変更するためのダイアログボックスを開きます。
	Wink active device	ログイン中に接続しているコントローラーが点滅します。
	エコーサービスの送信	EcoStruxure Machine Expert は ping ツールに似たエコーサービスがあります。 ネットワーク接続の良否を確認するために、EcoStruxure Machine Expert 5 つのエコーデータパケットをコントローラーに送信します。これらのパケットに連続して追加されるユーザーデータの量は、コントローラーの通信バッファーサイズによって異なります。 平均往復遅延時間と、接続を通じてエコーされたユーザーデータの量を示す結果メッセージが表示されます。
	Store communication settings in project	このオプションを有効にすると、別のコンピューターでプロジェクトを開く場合でも、EcoStruxure Machine Expert は通信設定を自動的に復元します。 このオプションが選択されていない場合、設定はご使用のコンピューターのローカルの EcoStruxure Machine Expert オプションに保存されます。その場合、別のコンピューターでプロジェクトを使用する場合は設定しなおす必要があります。
	オンラインモード確認済み	このオプションを有効にすると、次のいずれかのオンラインコマンドを実行するたびに EcoStruxure Machine Expert はユーザーに確認を求めます。**値の強制、値の書き込み、複数ダウンロード、強制リストの下方、シングルサイクル、開始、停止**。
	ネットワークスキャンを対象 ID でフィルター	このオプションを有効にすると、プロジェクトで設定されたデバイス設定されたデバイスと同じ対象 ID を持つデバイスのみがリストに表示されます。

クラシックモードの通信設定

概要

クラシックモードの**通信設定**タブは、**クラシックモード**が **ツール → オプション → デバイスエディター** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**通信ページ**のパラメーターで選択されているときに表示されます。デバイスとプログラミングシステムの通信パラメーターを設定できます。

クラシックモードの**通信設定**タブはパラメーターを設定するためのツリー構成を提供します。



このタブは 2 つの部分に分かれています。

- 左側には設定されているゲートウェイのチャンネルがツリー構成で表示されます。
- 右側には対応するデータと情報が表示されます。

ツリー構成の説明

最初のプロジェクトをローカルシステムに作成する時、ローカル**ゲートウェイ**は既にツリー内でノードとして表示されています。システムの開始時にゲートウェイも自動的に開始します。

このゲートウェイの設定はウィンドウの右部分に表示されます。

例 :

デバイス名 : Gateway-1

ポート : 1217

IP アドレス : 127.0.0.1

ドライバー : TCP/IP

ゲートウェイシンボルの右下のステータスビュレットは接続状態を示します。

色指定	詳細
赤	接続が確立できません。
緑	接続が確立されました。
黒	接続ステートが未定義です。

注記 : 一部の通信プロトコルでは、ゲートウェイの定期的な確認ができません。よって、ステータスが表示されません。

このゲートウェイを介して接続可能なデバイスが、**ゲートウェイ**ノード (**+/-** サインで開閉) の下に展開され表示されます。デバイスの前には シンボルが付いています。プロジェクトで設定されているデバイスの対象 ID と異なる ID のデバイスは灰色のフォントで表示されます。使用可能なデバイスの最新リストを取得するには、**ネットワークをスキャン**ボタンを使用します。

デバイスノードはシンボル、ノード名、およびノードアドレスで構成されています。ウィンドウの右部分には、**デバイス名** , **デバイスアドレス**、**チャネル数**、**ターゲット ID**, **ターゲット名**、**ターゲットタイプ**、**ターゲットメーカー**、および**ターゲットバージョン**が表示されます。

ツリー構造でチャンネルを選択すると **Select the network path to the controller** フィールドに自動的にゲートウェイチャンネルが指定されます。

注記：チャネル数 パラメーターには選択されたコントローラーが対応しているチャンネル数が表示されます。使用中のチャンネル数をオンラインでモニターすることはできません。
チャンネルとはクライアント (診断、Logic Builder、WebVisu、OPC、および HMI など) への接続です。通信サービスによって、クライアントが複数のチャンネルを短期間に占有することがあります。コントローラーが対応しているチャンネルがすべて使用されている場合、Logic Builder は、**接続がデバイスにより拒否されました。すべての通信チャンネルは既に使用されています** というメッセージを表示します。

フィルターと並び替え機能

タブの右側の選択ボックスで、ツリーに表示されているゲートウェイとデバイスノードのフィルター、および並び替えができます。

- **フィルター** : ツリー構造で表示されているデバイスを、プロジェクトで設定されているデバイスの **ターゲット ID** とマッチするデバイスのみに減らすことができます。
- **並べ替え順序** : ツリー構造で表示されているデバイスを、**名前**、または**ノードアドレス**でアルファベット順または昇順で並べ替えることができます。

ボタンとコマンドの説明

通信設定を変更するには、次のボタン、またはコマンドを使用します。

ボタン / コマンド	詳細
有効なパスを設定	このコマンドは選択されている通信チャンネルをコントローラーの有効なパスとして設定します。**有効なパスを設定**コマンドの説明を参照してください。ツリー構造でダブルクリックをしても同じことができます。
Add gateway...	このコマンドは**ゲートウェイ**ダイアログボックスを開きます。設定に追加するゲートウェイの設定ができます。 **ゲートウェイを追加** コマンドの説明を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
デバイスの追加 ...	**デバイスの追加**ダイアログボックスを開きます。現在選択されているゲートウェイに追加するデバイスを手動で定義することができます。(**ネットワークをスキャン** 機能を検討してください。) デバイスの追加の説明を参照してください (*54* ページ)。
Edit Gateway...	このコマンドは**ゲートウェイ**ダイアログボックスを開きます。選択されているゲートウェイ設定の編集ができます。 **ゲートウェイを編集 ...** コマンドの説明を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
Delete selected Device	このコマンドは、選択したデバイスを設定ツリーから削除します。
Scan for device by address	このコマンドは設定ツリーで指定されたアドレスをもつデバイスを検知するためにネットワークをスキャンします。検知されたデバイスは、指定したアドレスに名前を足してゲートウェイに表示されます。スキャンはツリーで選択されているゲートウェイの下のデバイスを指します。 デフォルトでは、メニューでこのコマンドを使用できません。**ツール → カスタマイズ** メニューからこのコマンドを追加できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

ボタン / コマンド	詳細
Scan for device by name	このコマンドは設定ツリーで指定された名前をもつデバイスを検知するためにネットワークをスキャンします (大文字と小文字を区別した検索)。検知されたデバイスは、指定した名前に固有のノードアドレスを足してゲートウェイに表示されます。スキャンはツリーで選択されているゲートウェイの下のデバイスを指します。 デフォルトでは、メニューでこのコマンドを使用できません。**ツール → カスタマイズ** メニューからこのコマンドを追加できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
Scan for device by IP address	このコマンドは設定ツリーで指定された IP アドレスをもつデバイスを検知するためにネットワークをスキャンします。検知されたデバイスは、指定したアドレスに名前を足してゲートウェイに表示されます。スキャンはツリーで選択されているゲートウェイの下のデバイスを指します。 デフォルトでは、メニューでこのコマンドを使用できません。**ツール → カスタマイズ** メニューからこのコマンドを追加できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
エコーサービスの送信	EcoStruxure Machine Expert は ping ツールに似たエコーサービスがあります。 ネットワーク接続の良否を確認するために、EcoStruxure Machine Expert 5 つのエコーデータパケットをコントローラーに送信します。これらのパケットに連続して追加されるユーザーデータの量は、コントローラーの通信バッファーサイズによって異なります。平均往復遅延時間と、接続を通じてエコーされたユーザーデータの量を示す結果メッセージが表示されます。
ローカルゲートウェイの設定	このコマンドはローカルゲートウェイの設定ダイアログボックスを開き、*Gateway.cfg* ファイルを手動で編集する代わりに使用できます。 **ローカルゲートウェイの設定 ...** コマンドの説明を参照してください。*(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
Scan network	このコマンドはローカルネットワークで利用できるデバイスの検索を開始します。該当するゲートウェーの設定ツリーが、それに応じて更新されます。

オプションの説明

ツリー構造の下には 2 のオプションがあります。

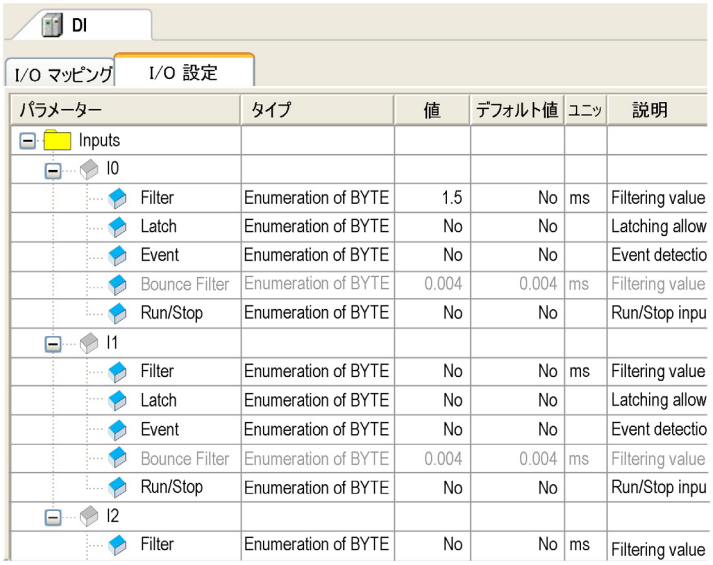
オプション	詳細
プロジェクトに通信設定を保存しないでください。	ネットワークパスの設定をプロジェクトには保存せず、ご使用のコンピューターのローカルオプション設定に保存する場合に有効にします。これによりプロジェクトを再度同じコンピューターで開くとこのパス設定が復元されます。他のシステムで使用する場合は定義しなおす必要があります。
オンラインモード確認済み	このオプションを有効にすると、次のいずれかのオンラインコマンドを実行するたびにユーザーに確認を求めます。**値の強制、複数ダウンロード、強制リストの開放、シングルサイクル、開始、停止、値の書き込み**。

設定

概要

設定ビューは **Show generic device configuration views** オプションが**ツール → オプション → デバイスエディター**ダイアログボックスで有効な場合、デバイスエディターで表示されます。**設定**ビューでは、デバイス固有のパラメーターを表示し、デバイスディスクリプションで許可されている場合はパラメーターの値が編集可能です。

デバイスエディターの**設定**ビュー



パラメーター	タイプ	値	デフォルト値	ユニッ	説明
Inputs					
I0					
Filter	Enumeration of BYTE	1.5	No	ms	Filtering value
Latch	Enumeration of BYTE	No	No		Latching allow
Event	Enumeration of BYTE	No	No		Event detectio
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering value
Run/Stop	Enumeration of BYTE	No	No		Run/Stop inpu
I1					
Filter	Enumeration of BYTE	No	No	ms	Filtering value
Latch	Enumeration of BYTE	No	No		Latching allow
Event	Enumeration of BYTE	No	No		Event detectio
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering value
Run/Stop	Enumeration of BYTE	No	No		Run/Stop inpu
I2					
Filter	Enumeration of BYTE	No	No	ms	Filtering value

このビューには以下の要素があります。

要素	詳細
パラメーター	パラメーター名、編集不可。
タイプ	パラメーターのデータ型、編集不可。
値	主にパラメーターのデフォルト値が直接またはシンボル名で表示されます。パラメーターが変更できる場合 (デバイスディスクリプションにより、変更できないパラメーターは灰色で表示されます)、表のセルをクリックして編集フレームまたは選択リストを開いて値を変更します。値がファイル指定の場合、セルをダブルクリックするとファイルを開くためのダイアログボックスが開きます。別のファイルを選択することができます。
初期値	デフォルトパラメーター値、編集不可。
単位	パラメーター値の単位。(例 : **ms** は、ミリ秒)、編集不可。
詳細	パラメーターの簡単な説明、編集不可。

パラメーター

概要

パラメータービューは、テーブルにコントローラー固有のパラメーターを表示します。デバイス説明で編集可能と定義されたパラメーターの **Value** を編集できます。この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

このビューには以下の要素があります。

要素	詳細
パラメーター	パラメーター名、編集不可。
タイプ	パラメーターのデータ型、編集不可。
値	パラメーターのデフォルト値が直接またはシンボル名で表示されます。 値を変更できる場合は、フィールドをダブルクリックして編集フレーム、選択リスト、またはファイル選択ダイアログボックスを開きます。 デバイス説明で編集可能として定義されていないパラメーターは、このリストでは灰色で表示されます。
初期値	デバイス説明で定義されたデフォルトパラメーター値、編集不可。
単位	パラメーター値の単位 (例 : **ms** は、ミリ秒)、編集不可。
詳細	デバイス説明で定義されたパラメーターの簡単な説明、編集不可。

アプリケーション

概要

デバイスエディターの**アプリケーション**ビューではコントローラーのアプリケーションのスキャンと削除を行います。アプリケーションの内容に関する情報やいくつかのアプリケーションのプロパティの詳細が表示されます。

要素の説明

アプリケーションビューでは次の要素を提供します。

要素	詳細
Applications on the PLC	このテキストボックスには、最後のスキャン (**リストのリフレッシュ**をクリック) で検知されたアプリケーション名をリスト表示します。 スキャンがまだ実行されていない、またはゲートウェイが設定されていない (*98* ページ) ためスキャンが可能でない場合は、それに応じたメッセージが表示されます。
削除 **すべて削除**	これらのボタンをクリックすると、現在リストで選択されているアプリケーション、またはすべてのアプリケーションがコントローラーから削除されます。
詳細	このボタンをクリックすると、アプリケーションオブジェクトの**プロパティ**ダイアログボックスの**情報**タブで定義された情報を表示するダイアログボックスが開きます。
コンテンツ	**表示 → プロパティ → Application build options** で、アプリケーションオブジェクト *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* に対して**アプリケーション情報のダウンロード**オプションが有効な場合、アプリケーションの内容に関する追加情報がコントローラーにロードされます。 **コンテンツ**ボタンをクリックすると、比較ビューで異なる POU が表示されます。複数回のダウンロードの後にこの情報を使用して、新しいアプリケーションのコードと、すでにコントローラーにあるコードを比較できます。これはログイン方法を決定するための詳細な情報を提供します。詳細は**ログイン**コマンドの説明を参照してください。
リストのリフレッシュ	このボタンをクリックすると、コントローラーのアプリケーションをスキャンします。リストがそれに応じて更新されます。

アプリケーションをコントローラーにロードする前の検証

アプリケーションをコントローラーにロードする前に以下の検証が行われます。

- コントローラー上のアプリケーションのリストとプロジェクトで使用可能なアプリケーションのリストが比較されます。不一致が検出された場合は、コントローラーにまだないアプリケーションをロードするためのダイアログボックス、またはコントローラーからアプリケーションを削除するダイアログボックスが表示されます。
- ロードされるアプリケーションに外部実装されている POU は、コントローラーでも利用可能かどうか検証されます。コントローラーで利用不可能な場合、**ダウンロード**オプションが選択されているとメッセージ (**unresolved reference(s)**) がメッセージボックス、および**メッセージ**ビューに生成されます。
- 読み込むアプリケーションの POU パラメーター (変数) は、コントローラーで既に使用可能なアプリケーションの同じ名前の POU パラメーターと比較 (シグネチャーの確認) されます。不一致が検出された場合、**ダウンロード**オプションが選択されているとメッセージ (**signature mismatch(es)**) がメッセージボックス、および**メッセージ**ビューに生成されます。

注記：表示 → プロパティ → Application build options で**アプリケーション情報のダウンロード**オプションが有効な場合、アプリケーションの内容に関する追加情報がコントローラーにロードされます。詳細については、前の表の**コンテンツ**ボタンの説明を参照してください。

同期ファイル

概要

デバイスエディターの**同期ファイル**ビューは、アプリケーションがダウンロードされた時に、コントローラーにダウンロードされたファイルを表示します。

- 外部ファイル。例えば、アプリケーションに追加されたファイル。
- 暗黙的ファイル。ソースコードのアーカイブファイルなど、ダウンロードの時間が設定されていて **Show implicit files for application download on the editor of a PLC** オプションが **ツール → オプション → デバイスエディター** ダイアログボックスで選択されている場合にこのビューにのみ表示されます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

要素の説明

同期ファイルタブには次の要素があります。

要素	詳細	例
ファイル名	アプリケーション内でのファイル名、または暗黙に転送されたファイル (例えば、archive.prj) の直接的な名前。 ファイル名をダブルクリックして開きます。	archive.prj
Host path	ファイルの保存場所または下の場所。 Windows Explorer で、パスをダブルクリックしてディレクトリーを開きます。	D:\Proj1\Files
Time interval	コントローラーでファイルを更新する時間の間隔。	ダウンロード後 / アプリケーションのオンライン変更。
情報	追加情報、プロジェクトにより異なる。	オブジェクト : 外部ファイル
元	元のファイルの種類	外部ファイルオブジェクト、ソースコードダウンロードプロバイダー

ファイル

概要

デバイスエディターの**ファイル**ビューでは、ホストとコントローラー間のファイル転送を行います。ローカルネットワークのディレクトリーから任意のファイルを選択し、現在接続されているランタイムシステムのファイルディレクトリーにコピーする、またその逆を行うことができます。

このビューは 2 つの部分に分かれています。

- 左側には**ホスト**のファイルが表示されます。
- 右側には**ランタイム**システムのファイルが表示されます。

要素の説明

ファイルビューでは次の要素を提供します。

要素	詳細
	**ランタイム**リストを更新します。
	ファイルをコピーできる新しいフォルダーを作成します。
	選択したファイルまたはフォルダーを削除します。
場所	ファイル転送に使用されるファイルシステムのフォルダーを指定します。 リストからエントリーを選択するか、ファイルシステムツリーを参照します。
<< >>	EcoStruxure Machine Expert とコントローラー間のファイル転送 (以下パラグラフ参照)。

ファイルツリーのディレクトリーには、次のような種類があります。

要素	詳細
	通常のファイルディレクトリー。
	コントローラーのランタイムシステムの設定で定義されたプレースホルダーファイルディレクトリー。
	コントローラーのランタイムシステムの暗黙的に作成されたプレースホルダーファイルディレクトリー。

EcoStruxure Machine Expert とコントローラー間のファイル転送

手順	手順内容	コメント
1	ファイルシステムツリーでコピーされるファイルを選択します。	複数が選択可能です。フォルダーを選択し、その中のファイルをコピーすることもできます。
2	**>>** または、**<<** ボタンをクリックしてランタイムシステムで設定されたディレクトリーに選択したファイルを転送します。	ここでの転送はコピーを意味します。ファイルがランタイムのディレクトリーにない場合は新たに作成されます。すでに存在する場合は、上書きされます。 ファイル名が既に存在し書込み禁止の場合、適切なメッセージが生成されます。

ログ

概要

デバイスエディターの**ログ**ビューでは、コントローラーのランタイムシステムで記録されたイベントが表示されます。

次の項目に関連します。

- システム起動、またはシャットダウン時のイベント (ロードされたコンポーネント、およびそのバージョン)
- アプリケーションのダウンロード、および起動プロジェクトのダウンロード
- 顧客固有のエントリー
- I/O ドライバーのログエントリー
- データサーバーのログエントリー

要素の説明

ログビューでは次の要素を提供します。

要素	詳細
Severity	ログのイベントは 4 つのカテゴリーにグループ分けされます。 ● **警告** ● **エラー** ● **例外** ● **情報** リストの上にあるバーのボタンの横には、それぞれのログ数が表示されます。ボタンをクリックしてそれぞれのエントリーを表示、または非表示にします。
タイムスタンプ	日時 例 : **13.01.2007 09:48**
詳細	イベントの説明 例 : **< アプリケーション > 開始**
コンポーネント	ここでは、特定のコンポーネントを選択し、そのコンポーネントに関連するログエントリーのみを表示します。初期設定は **<All components>** です。
ロガー	選択リストは利用可能なログを提供します。初期設定は **<Default Logger>** で、ランタイムシステムで定義されます。
	リストを更新します。
	リストを XML ファイルにエキスポートします。ファイルを保存するダイアログボックスが開きます。ファイルフィルターは、**xml-files (*.xml)** に設定されます。ログファイルは .XML の拡張子で指定した名前で選択されたディレクトリーに保存されます。
	上記でエキスポートされた XML ファイルに保存されたログエントリーを表示します。ファイルを参照するためのダイアログボックスが開きます。フィルターは、**xml-files (*.xml)** に設定されます。ログファイルを選択します。このファイルのエントリーは別のウィンドウに表示されます。
	表示されているエントリーを削除し、ログテーブルをクリアします。
Offline-Logging	このオプションは EcoStruxure Machine Expert では使用されません。
UTC Time	このオプションを有効にするとランタイムシステムのタイムスタンプをそのまま (変換なしで) 表示します。無効にするとコンピューターのローカル時刻 (オペレーティングシステムのタイムゾーンに従う) でタイムスタンプが表示されます。 **注記 :** タイムスタンプを UTC (Universal Time Coordinated、世界協定時刻) で表示するには、コントローラーの構成時刻をあらかじめ UTC 時刻に設定する必要があります (コントローラー設定の**サービス**タブも参照してください)。

トラブルシューティング

エラーが検出されて ***SOURCEPOSITION***, というテキストが表示された場合は、ダブルクリックするか、コンテキストメニューから **Show source code in editor** コマンドを実行して、それぞれの機能を開くことができます。該当する位置にカーソルが置かれます。診断するために、**ダウンロード情報ファイル**、およびエクスポートされたログファイルを含むプロジェクトのアーカイブを使用できます。保護されている機能の場合は、**The source code is not available for <function name>** というメッセージが表示されます。

PLC 設定

概要

デバイスエディターの **PLC 設定**ビューで、コントローラーの一般設定を行います。

要素の説明

PLC 設定ビューには次の要素があります。

要素	詳細
I/O 処理用アプリケーション	ここで、I/O 処理を監視するための、**デバイスツリー**でデバイスに割り当てられたアプリケーションを定義します。EcoStruxure Machine Expert では、アプリケーションは 1 つのみ使用できます。
PLC 設定領域	
停止中に IO を更新	PacDrive 用メモ : TM5 / TM7 IO には対応していますが、IO ベースの制御またはドライブには対応していません。 このオプションが有効な場合 (初期設定)、コントローラーが停止するとチャンネルの入力値および出力値が更新されます。ウォッチドッグの有効期限が切れた場合、出力は定義されたデフォルト値に設定されます。
停止時の出力に対する動作	選択リストで、停止した場合の出力チャンネルの値の処理方法を次のオプションから 1 つ定義します。 ● **現在の値を保持** 現在値は変更されません。 ● **すべての出力をデフォルトに設定** マッピングによるデフォルト値が割り当てられます。 ● **プログラムの実行** プロジェクトで有効なプログラムによって出力の動作を決定できます。ここにプログラム名を入力すると、コントローラーの停止時にそのプログラムが実行します。**...** ボタンをクリックして、**入力アシスタント**を使用できます。
常に変数を更新	バス周期タスクで I/O 変数が更新された場合のグローバル定義。この設定は、更新設定が**無効**の場合は、スレーブとモジュールの I/O 変数にのみ有効になります。 ● **無効 (タスクで使用されている場合のみ更新)**: I/O 変数は、タスクで使用されている場合にのみ更新されます。 ● **有効 1 (タスクで使用されていない場合はバス周期タスクを使用)**: I/O 変数は、他のタスクで使用されていない場合バス周期タスクで更新されます。 ● **有効 2 (バス周期タスクでは常に)**: すべての変数は、使用されているか、または、入力または出力チャンネルにマップされているかにかかわらず、バス周期タスクの周期ごとに更新されます。 このオプションは、**I/O マッピング** ビュー (*126* ページ) でデバイスごとに個別に設定できます。
バスサイクルオプション領域	
バスサイクルタスク	選択リストには、有効なアプリケーションの**タスク設定**で定義されているタスク (例 : **MAST**) が表示されます。初期設定の **MAST** は自動的に入力されます。 **<unspecified>** は、タスクがコントローラーの内部設定に応じて選択され、そのためコントローラー依存であることを意味します。

注記 : バス周期タスクを **<unspecified>** に設定することによって、アプリケーションの意図しない動作を招く可能性があります。各コントローラーについては、*プログラミングガイド*を参照してください。

⚠ 警告

装置の意図しない動作

バス周期タスクを コントローラーのバス周期タスク設定を理解せずに、**<unspecified>** に使用しないでください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

追加設定

要素	詳細
IO マッピング用強制変数の生成	この設定はデバイスで対応されている場合にのみ利用できます。各コントローラーについては、*プログラミングガイド*を参照してください。 このオプションが、**I/O マッピング**ダイアログボックスで変数に割り当てられた各 I/O チャンネルで有効な場合は、アプリケーションがビルドされるとすぐに 2 つのグローバル変数が作成されます。これらの変数は、I/O 値を強制するための HMI ビジュアライゼーションで使用できます。詳細については、*I/O マッピング*の章 (*122* ページ) を参照してください。
デバイス用診断を有効にする	CAA デバイス診断ライブラリーがプロジェクトに自動的に追加されます。各デバイスに対して、ファンクションブロックが**アプリケーションツリー**に生成されます。ファンクションブロックがすでに存在する場合は、拡張 FB (例えば、EtherCAT) が使用されるか、新たなファンクションブロックインスタンスが追加されます。このファンクションブロックには、デバイス診断用の一般的な実装が含まれます。これらのファンクションブロックインスタンスを使用することで、デバイスのステータスをアプリケーションで取得できます。また、検出されたエラーを評価することもできます。このライブラリーは、**デバイスツリー**のプログラムによる評価 (例えば、子デバイスでの検索や親デバイスへのジャンプなど) のための機能を提供します。詳細は、ライブラリーにある PDF ドキュメントの *CAA デバイス診断* を参照してください。
Show I/O warnings as errors	I/O 設定に関する警告は検出されたエラーとして出力します。

ユーザーとグループ

概要

デバイスエディターには、デバイスの**ユーザーとグループ**管理に対応しているデバイス用の**ユーザーとグループ**ビューがあります。デバイスでサポートされている場合は、そのデバイス**ユーザーとグループ**管理を表示および編集できます。その後、**アクセス権**を割り当てることで、ランタイムで特定のユーザグループがコントローラー上のオブジェクトにアクセスする権限を割り当てることができます。

プロジェクトレベルでのユーザー管理については、**プロジェクト → ユーザー管理 → 権限 ...** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を参照してください。

デバイス**ユーザーとグループ**管理はデバイス説明で事前に定義できます。

プロジェクトのユーザー管理と同様に、ユーザーは少なくとも 1 ユーザーグループのメンバーである必要があります。ユーザーグループにのみ特定のアクセス権 (*116* ページ) を割り当てることができます。

ユーザーとグループを管理するには、**管理者**としてログインする必要があります。

注記：ユーザーとグループ機能は EcoStruxure Machine Expert プロジェクトを悪意のあるのアクセスから保護するためではなく、信頼できるユーザーからの間違いを防ぐために使用されることを意図しています。
プロジェクト全体を保護したい場合は、**プロジェクト設定 → セキュリティ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で **Enable project file encryption** オプションを有効にします。
プロジェクトのコードの一部のみを保護したい場合は、そのコードをコンパイルされたライブラリー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 内に入れます。

⚠ 注意

認証されていない不正なアクセス

- コントローラーとコントローラーのネットワークを可能な限り公衆のネットワークおよびインターネットに公開しないでください。
- リモートアクセスには VPN のような追加のセキュリティーレイヤーを使用し、ファイアウォールのメカニズムをインストールしてください。
- 許可されている人のアクセスを制限してください。
- 起動時にデフォルトパスワードを変更し、頻繁に更新してください。
- 定期的かつ頻繁にこれらの手段の有効性を確認してください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

注記：セキュリティーに関連するコマンドを使用できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。ログインしている対象デバイスのオンラインの**ユーザーとグループ**管理でユーザーの追加、編集、および削除が可能です。

注記：EcoStruxure Machine Expert ソフトウェアを使用してユーザーアクセス権を確立する必要があります。コントローラーのアプリケーションを別のコントローラーのアプリケーションに*クローン*した場合、クローン先のコントローラーでユーザーアクセス権を有効にし確立する必要があります。

注記：ユーザーアクセス権が有効だがパスワードをもっていない場合、コントローラーにアクセスする唯一の方法は、SD カード、または USB メモリーキーを使用し、またはコントローラーによってはスクリプトを実行し、**ファームウェアの更新**操作を行うことです (詳細は *コントローラーアシスタントユーザーズガイド* を参照してください)。スクリプトの実行方法はコントローラーによって異なるため、使用しているコントローラーのプログラミングガイドの *SD カードを使用してファイルを転送*、または *USB メモリーキーを使用してファイルを転送*の章を参照してください。これにより、コントローラーのメモリーから既存のアプリケーションが効果的に削除され、コントローラーへのアクセスが復元されます。

ユーザーとグループビューのツールバー

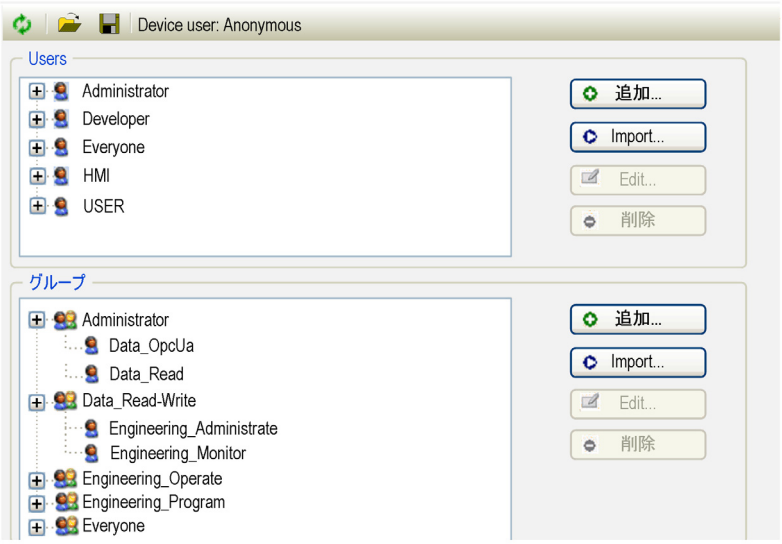
ツールバーには以下の要素があります。

要素	詳細
同期	**同期**ボタンをクリックして、エディターとコントローラーの**ユーザーとグループ**管理の同期をオンまたはオフにします。 **同期**が有効になっていない場合は、エディターにはディスクからインポートされた**ユーザーとグループ**管理設定が含まれているか、または設定がまったく含まれていません。 **同期**が有効になっている場合は、エディターに表示されているデータが接続されたコントローラーの**ユーザーとグループ**管理設定と継続的に同期されます。 デバイスと同期されていない**ユーザーとグループ**管理設定がエディターに含まれているときに、**同期**を実施すると、エディターに表示される内容を決定するように求められます。 ● **Upload from the device and overwrite the editor content**: コントローラーの**ユーザーとグループ**設定がエディターに読み込まれます。エディターの内容は上書きされます。 ● **Download the editor content to the device and overwrite the user management there**: エディターの設定がコントローラーに読み込まれます。コントローラーの内容は上書きされます。
Import from disk	**Import from disk** ボタンをクリックすると、**ユーザーとグループ**管理設定をディスクから選択しインポートするダイアログボックスが開きます。 **ユーザーとグループ**ビューでボタンをクリックすると、ファイルタイプはデフォルトで *デバイスユーザー管理* ファイル (*.dum) に設定されます。 **Import from disk** は、オフラインモードのとき、または**同期**が解除されているときに使用できます。
Export to disk	**Export to disk** ボタンをクリックするとファイルをディスクに保存するダイアログボックスが開きます。**ユーザーとグループ**管理設定は XML ファイルとして保存されます。 **ユーザーとグループ**ビューでボタンをクリックすると、ファイルタイプはデフォルトで *デバイスユーザー管理* ファイル (*.dum) に設定されます。
Device user	コントローラーにログインしているユーザー名。

ユーザーとグループ管理

ユーザーとグループ管理ダイアログの操作はプロジェクトユーザー管理の操作と同様です。

デバイスエディターの**ユーザーとグループ**ビュー



このビューは 2 つの部分に分かれています。

- 上部は、**ユーザー**のアクセス管理用です。
- 下部は、**グループ**のアクセス管理用です。

ユーザーとグループが確立される前のユーザーとグループ管理の編集または表示

コントローラーがデバイスの**ユーザーとグループ**管理に対応している場合、最初のログイオン時に次の手順に従います。

手順	手順内容	コメント
1	**デバイスツリー**でコントローラーのノードをダブルクリックします。	**結果**：デバイスエディターが開きます。
2	**ユーザーとグループ**ビューを選択します。	–
3	**同期**ボタンをクリックします  。	**結果**：デバイスの**ユーザーとグループ**管理を有効にするかどうかを確認するダイアログボックスが開きます。
4	**はい**をクリックしてダイアログボックスを確認し、デバイス**ユーザーとグループ**管理を有効にします。	**結果**：**Device user login** ダイアログボックスが開きます。
5	Administrator を**ユーザー名**と**パスワード**として入力します。	**結果**：**Password expired! Please provide a new one.** ダイアログボックスが開きます。
6	新しいパスワードを入力し **OK** をクリックして確認します。	**結果**：デバイスの**ユーザーとグループ**管理がエディタービューに表示されます。

コントローラーのユーザーとグループ管理で新規ユーザーの設定を行う

コントローラーが**ユーザーとグループ**管理に対応している場合、次の手順で新規ユーザーを追加することができます。

手順	手順内容	コメント
1	**デバイスツリー**でコントローラーのノードをダブルクリックします。	**結果**：デバイスエディターが開きます。
2	**ユーザーとグループ**ビューを選択します。	–
3	**同期**ボタン  をクリックして、**ユーザーとグループ** 管理設定をコントローラーからエディターに読み込みます。	コントローラーにログインしていない場合は、**Device User Login** ダイアログボックスが開きます。ユーザー名とパスワードを入力できます。 **結果**：コントローラーらの**ユーザーとグループ**管理設定がエディターに表示されます。
4	**ユーザーとグループ**ビューの**ユーザー**部分にある**追加**ボタンをクリックします。	**結果**：**ユーザーの追加**ダイアログボックスが開きます。
5	新しいユーザーの**名前**を入力しリストからユーザーの**デフォルトグループ**を選択します。	ユーザーを後で他のグループに割り当てられます。
6	新しいパスワードを入力、パスワードを確認、ユーザーがパスワードを変更できるかどうか、および最初のログイン時にユーザーがパスワードを変更する必要があるかどうかを指定します。	–
7	**OK** をクリックして**ユーザーの追加**ダイアログボックスを閉じます。	**結果**：新しいユーザーは、**ユーザー**部分に新しいノードとして、**グループ**部分に選択したデフォルトグループの新しいサブノードとして表示されます。

Loading a ユーザーとグループ管理の *.dum ファイルからの読込み、変更、および後のコントローラーへのダウンロード

手順	手順内容	コメント
1	**デバイスツリー**でコントローラーのノードをダブルクリックします。	**結果**：デバイスエディターが開きます。
2	**ユーザーとグループ**ビューを選択します。	–
3	**編集**ボタンをクリックし、**ユーザーとグループ**管理の *.dum ファイルを検索、**開く**をクリックして確認します。	**結果**：ファイルに保存されたユーザーとグループの設定がエディターに表示されます。
4	必要に応じて設定を調整してください。	–

手順	手順内容	コメント
5	**同期**ボタン  をクリックして、**ユーザーとグループ**管理設定をコントローラーに送信します。	適切な操作を選択するように促すダイアログボックスが表示されます。
6	**Download the editor content to the device and overwrite the user management there** オプションを選択します。	**結果** : **Device user login** ダイアログボックスが表示されます。
7	コントローラーにログインするために有効なログインデータを入力します。	ログイン後、変更がコントローラーに送信されます。 **同期**ボタン  が有効な限り、エディターでの変更は自動的にコントローラーに転送されます。

ユーザーとグループ管理設定を印刷

ユーザーとグループビューの設定を印刷するには、**ファイル**メニューから**印刷**コマンドを実行するか、**プロジェクト**メニューから**ドキュメント**コマンドを実行します。

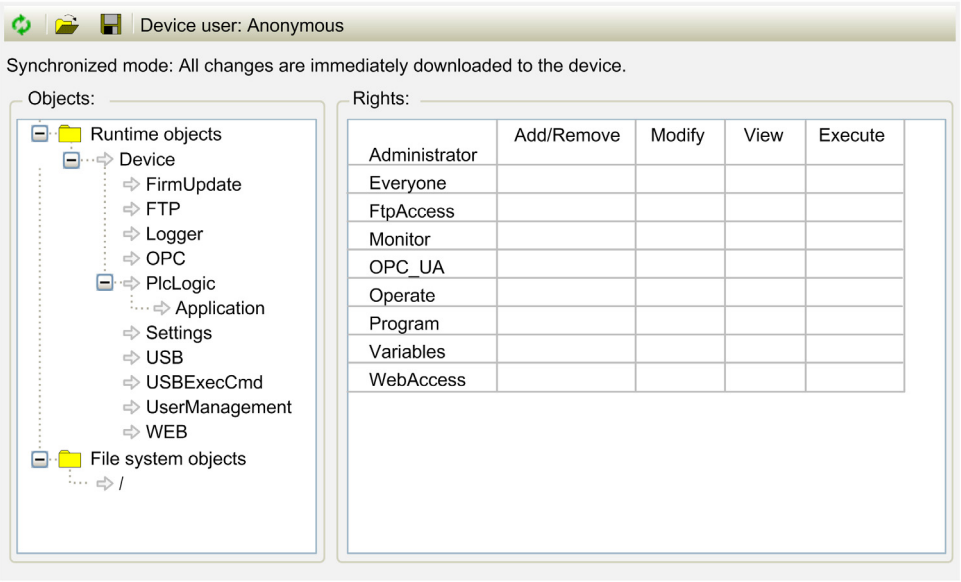
アクセス権

概略

デバイスエディターの **アクセス権**ビューで、コントローラー内のオブジェクトに対するデバイスユーザーのデバイスアクセス権を定義します。

コントローラーのデバイスエディターで**アクセス権**ビューを使用するには、**Show access rights page** オプションを**ツール → オプション → デバイスエディター** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で有効にします。さらに、**ユーザーとグループ**の管理はコントローラーで設定される必要があります。

デバイスエディターの**アクセス権**ビュー



コントローラーのユーザーとグループ管理でコントローラーオブジェクトに対するアクセス権の変更を行う

コントローラーが**ユーザーとグループ**管理に対応している場合、次の手順でアクセス権を割り当てることができます。

手順	手順内容	コメント
1	**デバイスツリー**でコントローラーのノードをダブルクリックします。	**結果** : デバイスエディターが開きます。
2	**アクセス権**ビューを選択します。	–
3	**同期**ボタン  をクリックして、**アクセス権** 管理設定をコントローラーからエディターに読み込みます。	コントローラーにログインしていない場合は、**Device User Login** ダイアログボックスが開きます。ユーザー名とパスワードを入力できます。 **結果** : コントローラーらの**アクセス権**管理設定がエディターに表示されます。
4	左側の**オブジェクト**ツリー構成でオブジェクトを選択します。	**結果** : 右側の **Rights** エリアで、選択されたオブジェクトのアクセス権が設定されたユーザーグループのテーブルに表示されます。
5	**Rights** テーブルで変更したい権利をダブルクリックします。	選択したオブジェクトに子オブジェクトがある場合は、子オブジェクトに対する権限を同時に変更するかどうかを確認するダイアログボックスが表示されます。
6	子オブジェクトの権限を変更するかどうか、**はい**または**いいえ**をクリックしてダイアログボックスを閉じます。	**結果** : 権利の変更が許可から不許可に、またはその逆に切り替えられます。 テーブルセルのシンボルはそれに応じて変更されます。 明示的に設定されている権限は、緑色または赤色の記号で表に表示されます。 親オブジェクトから継承された権限は、灰色の記号で表に表示されます。

アクセス権ビューのツールバー

ツールバーには以下の要素があります。

要素	詳細
同期	**同期**ボタンをクリックして、エディターとコントローラーの**アクセス権**管理の同期をオンまたはオフにします。 **同期**が有効になっていない場合は、エディターにはディスクからインポートされた**アクセス権**管理設定が含まれているか、または設定がまったく含まれていません。 **同期**が有効になっている場合は、エディターに表示されているデータが接続されたコントローラーの**アクセス権**管理設定と継続的に同期されます。 デバイスと同期されていない**アクセス権**管理設定がエディターに含まれているときに、**同期**を実施すると、エディターに表示される内容を決定するように求められます。 ● **Upload from the device and overwrite the editor content**: コントローラーの**アクセス権**設定がエディターに読み込まれます。エディターの内容は上書きされます。 ● **Download the editor content to the device and overwrite the user management there**: エディターの設定がコントローラーに読み込まれます。コントローラーの内容は上書きされます。
Load from disk	**Load from disk** ボタンをクリックすると、ユーザーアクセス権管理設定をディスクから選択しインポートするダイアログボックスが開きます。 **アクセス権**ビューでボタンをクリックすると、ファイルタイプはデフォルトで *デバイス権限管理* ファイル (*.drm) に設定されます。
Save to disk	**Save to disk** ボタンをクリックするとファイルをディスクに保存するダイアログボックスが開きます。 **アクセス権**ビューでボタンをクリックすると、ファイルタイプはデフォルトで *デバイス権限管理* ファイル (*.drm) に設定されます。
Device user	コントローラーにログインしているユーザー名

オブジェクトエリア

左側の**オブジェクト**ツリー構成にランタイムモードで実行が許可されているオブジェクトが表示されます。オブジェクトはオブジェクトのソースによって割り当てられます。オブジェクトグループで部分的に並び替えられます。右側の **Rights** エリアで、選択したオブジェクトのユーザーグループに対するアクセスオプションを設定できます。

オブジェクトツリー構成の最上位にフォルダーに分けられた 2 つのオブジェクトカテゴリーがあります。

- ファイルシステムオブジェクト
- ランタイムシステムオブジェクト

オブジェクトカテゴリーの下にインデントされて、さらにサブノードがあります。例えば、サブノード**デバイス**には、以下のサブノードがあります。

- **ロガー**
- **PlcLogic**
- **設定**
- **UserManagement**

Rights エリア

右側の **Rights** エリアに、選択されたオブジェクトのアクセス権がテーブルに表示されます。選択されたオブジェクトに対して可能なアクションの権限設定がすべてのユーザーグループについて表示されます。

オブジェクトに対する次のアクションが設定できます。

- **追加 / 削除**
- **変更**
- **表示**
- **Execute**

アクセス権を示すシンボル

アイコン	詳細
	アクセス (アクション) は明示的に許可されています。
	アクセス (アクション) は明示的に許可されていません。
 	アクセス権は親オブジェクトから継承されています。
	親オブジェクトも含み、アクセス権は明示的に許可にも不許可にもなっていません。アクセスできません。
アイコンなし	異なるアクセス権の複数のオブジェクトが選択されています。

アクセス権を変更するにはシンボルをクリックします。

アクセス権定義を印刷

アクセス権ビューの設定を印刷するには、**ファイル**メニューから**印刷**コマンドを実行するか、**プロジェクト**メニューから**ドキュメント**コマンドを実行します。

タスクの配置

概要

デバイスエディターの**タスクの配置**ビューには、入力 / 出力と定義されたタスクへの割り当ての表が表示されます。この情報を表示するには、プロジェクトをコンパイルしてコードを生成する必要があります。同じ入力 / 出力が別のタスクで別の優先度で更新される場合、この情報はトラブルシューティングに役立ちます。

デバイスエディターの**タスクの配置**

テーブルにはタスクが優先度順に表示されます。列のヘッダー (タスク名) をクリックするとそのタスクに割り当てられた変数のみが表示されます。最初の列 (**I/O チャンネル**) をクリックするとすべての変数を表示します。

チャンネルの I/O マッピングテーブルを開くには、入力または出力をダブルクリックします。

青い矢印はバスサイクルのタスクを示しています。

上記の例では、**usiBK5120Out AT %QB0** 変数は 2 つの異なるタスクに割り当てられています。この場合、出力はひとつのタスクでセットされ、もう 1 つのタスクで上書きされる可能性があるので、未定義の値につながります。一般的に、プログラムをデバッグするのが困難になり、機械や処理の操作に意図しない結果が生じることがあるため、複数のタスクで出力を書き込むことは推奨しません。

⚠ 警告
装置の意図しない動作 複数のタスクで出力変数に書き込みを行わないでください。 **上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。**

ステータス

概要

デバイスエディターの**ステータス**ビューでは、ステータス情報 (例 : **実行中**、**停止**) やデバイス、使用済みのカード、および内部バスシステムからの特定の診断メッセージを表示します。

情報

概要

デバイスエディターの**情報**ビューには、**デバイスツリー**で現在選択されているデバイスの一般的な情報 (**名前、メーカー、タイプ、バージョン番号、注文番号、詳細、イメージ**) が表示されます。

6.2
I/O マッピング

このセクションについて

このセクションには次の項目が含まれています。

I/O マッピング

概要

デバイスエディターの **I/O マッピング**ビューは、**< デバイスタイプ > I/O マッピング** (例 : **PROFIBUS DP I/O マッピング**) という名前がつけられます。これはコントローラーの I/O マッピングの設定に使用します。アプリケーションで使用されているプログラム変数をコントローラーの入力、出力、メモリーアドレスに割り当てます。

I/O を取り扱うアプリケーションは、**PLC 設定** ビュー (*110* ページ) で定義します。

注記 : デバイスがサポートしている場合は、アプリケーションを事前にロードしないで、I/O ハードウェアにアクセスするオンライン設定モードを使用できます。詳細については、オンライン設定モード *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。

次の章を参照してください。
- I/O マッピングダイアログの操作 (*125* ページ)
- オンラインモードの I/O マッピング (*128* ページ)
- I/O の強制用暗黙的変数 (*129* ページ)

変数の I/O マッピングの概略

I/O マッピングが現在のデバイスで設定できるかどうかはデバイスによって異なります。I/O マッピングビューは、暗黙的に作成されたデバイスのインスタンスを表示するためのみに使用されることがあります。*IEC オブジェクト* (*127* ページ) の説明を参照してください。

基本的には、変数の I/O マッピングについて以下に注意してください。
- 入力を要求する変数には、書き込みによるアクセスはできません。
- 既存の変数は 1 点の入力にのみマップできます。
- **I/O マッピング**ビューの代わりに AT 宣言 (*489* ページ)を使用してアドレスを変数に割り当てることができます。
  ただし以下に注意してください。
  - AT 宣言はローカルまたはグローバル変数でのみ使用できます。POU の入力および出力変数では使用できません。
  - I/O の強制変数 (I/O を強制するための暗黙的変数 (*129* ページ)を参照してください) を AT 宣言で生成することはできません。
  - 構造体やファンクションブロックのメンバーで AT 宣言が使用された場合は、すべてのインスタンスは同じメモリーロケーションにアクセスします。このメモリーロケーションとは、C 言語などの古典的なプログラミング言語の静的変数にあたります。
- 構造のレイアウトは対象デバイスによって決まります。
- **I/O マッピング**ビューで I/O チャンネルに割り当てられた各変数に対して、アプリケーションのビルド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 中に強制変数が作成されます。マシンの試運転にその強制変数を入力値または出量値を強制するために使用できます *I/O を強制するための暗黙的変数* (*129* ページ) の章を参照してください。

自動 I/O マッピング

自動 I/O マッピングは、I/O モジュールを備えたデバイスまたはモジュールが**デバイスツリー**に追加され、各入力および / または出力にマップされると自動的に IEC 変数が作成されます。デフォルトでこの機能は有効になっています。

プロジェクト → プロジェクト設定 → 自動 I/O マッピングダイアログボックスで、この機能を無効にし、設定することができます。

ダイアログボックスでは次の要素が表示されます。

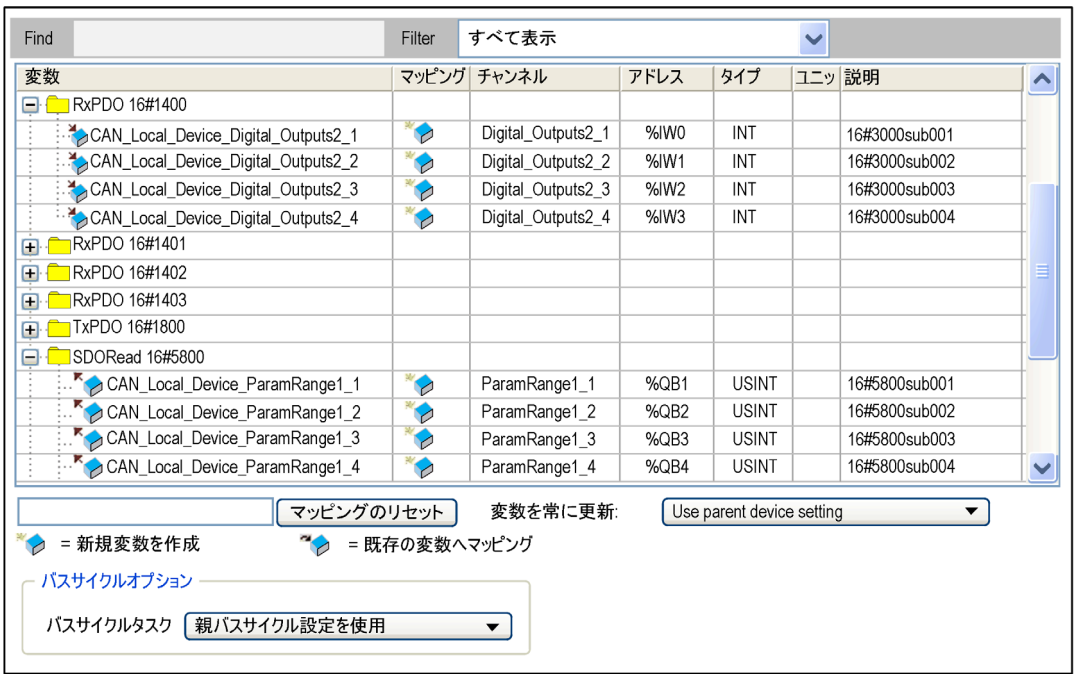
要素	詳細
I/O マッピングで変数を作成	自動 I/O マッピングを有効または無効にするオプションを選択します。
マッピング	
ビット単位	各ビットに変数を作成するにはこのオプションを選択します。
モジュールごと	個々のビットではなくモジュールごとに変数を作成するにはこのオプションを選択します。

要素	詳細
命名規則	
テキストボックス	変数名の構成部分を指定するには、# シンボルに続く次の文字を入力します。 ● #X を入力すると、入力には i、出力には q が変数名に付きます。 ● #T を入力すると、変数名に変数のデータ型のプレフィックスが付きます。異なるデータ型に使用するプレフィックスの一覧は、*識別子の名称に関する推奨事項* の章 (*483* ページ) にあります。 ● #D を入力すると変数名にデバイス名が付きます。 ● #C を入力すると**チャンネル**列で定義した名前が変数名に付きます。

I/O マッピングダイアログの操作

概要

下図はデバイスエディターの **I/O マッピング**タブを示しています。



チャンネルエリアの要素の説明

I/O マッピングタブの**チャンネル**エリアでは、デバイスによって提供されている場合、次の要素を表示します。

要素	詳細
チャンネル	デバイスの入力または出力の象徴的な名前
アドレス	チャンネルのアドレス。例 : %IW0
タイプ	入力または出力チャンネルのデータ型。例 : BOOL データ型がスタンダードではなくストラクチャーまたはデバイスディスクリプションで定義されたビットフィールドの場合、それが IEC 規格の一部である場合にのみ表示されます。デバイスディスクリプションで IEC タイプとして示されます。それ以外の場合はテーブルのエントリーは空になります。
デフォルト値	コントローラーが停止モードに設定されているときにチャンネルに割り当てられているデフォルト値。 この列は、**すべての出力をデフォルトに設定**が、デバイスエディターの **PLC 設定** ビューで (*110* ページ) **停止時の出力に対する動作**パラメーターに設定されているときのみ使用可能です。 このフィールドは、新しい変数にマッピングするとき、またはマッピングがない場合に編集できます。既存の変数にマッピングする場合は、変数の初期化値がデフォルト値として使用されます。 **注記 :** 「新規」の変数と「既存」の変数 (AT 宣言を使用したもの) が同じ出力にまマッピングされている場合は、「既存」の変数の初期値が規定値として使用されます。
単位	パラメーター値の単位。例 : **ms** は、ミリ秒。
詳細	パラメーターの簡単な説明。
現在値	オンラインモードで表示されるパラメーターの現在値

注記 : アプリケーションで使用されていない入力および出力は、オンラインモードではコントローラーで読み込めません。入力と出力が使用されていないことを示すために、灰色の背景でマークされています。灰色の線に表示される値はすべて無効です。

アドレスの変更と修正

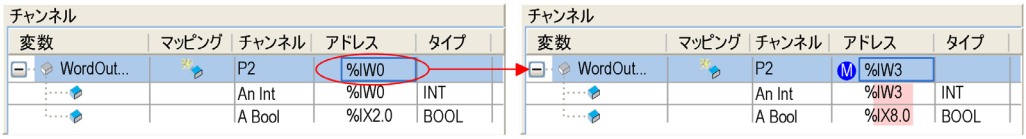
この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

このタブで表示された出力または入力のアドレスを変更または修正できます。これを使用して、与えられたハードウェア設定にアドレスを調整、またはアドレス値をモジュールの順番が変わっても保持するようにします。デフォルトで、アドレス値の自動調整を行う起因となります。

デバイスディスクリプションによっては、入力または出力のアドレスのみを変更できますが、そのサブルメント (ビットチャンネル) は変更できないことを考慮してください。よって、このマッピングテーブルでサブツリーで表示されている入力または出力は、最上位のエントリーのアドレスフィールドのみを編集できます (下図参照 : 最初の行のアドレスフィールドのみ開くことができます)。

アドレス値を修正するには、**アドレス**列のエントリーを選択してスペースバーを押して編集フィールドを開きます。値を修正するかそのままにして編集フィールドをリターンキーで閉じます。アドレスフィールドは **M** シンボルでマークされ、現在の値は修正されていることを示します。

値が修正された場合は、次のアドレス (次の固定アドレスまで) 調整されます。



値の修正を取り除きたい場合は、アドレス編集フィールドを再び開きアドレスエントリーを削除し、**Enter** キーで閉じます。アドレスおよび認識された後続アドレスは、手動修正前の値に戻ります。**M** シンボルは削除されます。

I/O マッピングの設定

変数列で適切なプロジェクト変数をデバイスの各入力および出力チャンネルに割り当てることで I/O マッピングを行います。

- チャンネルの種類は、**変数**列にシンボル (は入力、 は出力) で既に示されています。この場合、チャンネルをマップする変数の名前またはパスを入力します。既存のプロジェクト変数にマップするか新しい変数を定義することができ、それは自動的にグローバル変数と宣言されます。
- 構造変数をマップする場合、エディターは構造変数 (例 : %QB0) と特定の構造要素 (例 : この場合 %QB0.1 および QB0.2) の両方を入力できないようにします。
  これは、マッピングテーブルにビットチャンネルのサブツリーをもったエントリーがある場合 (下図参照)、メインエントリーまたはサブエレメント (ビットチャンネル) のどちらかの行だけに変数が入力できるということです。
- 既存の変数のマッピングには完全なパスを指定します。
  例 : < アプリケーション名 >.<POU パス >.< 変数名 >';
  例 : app1.plc_prg.ivar

  ... ボタンで入力アシスタントを開くと便利です。**マッピング**列には、 シンボルが表示されアドレス値に取り消し線が付けられます。このメモリーアドレスが存在しないという意味ではありません。ただし、既存変数の値は別のメモリーロケーションで管理されているためこのメモリーアドレスは直接使用されません。特に出力の場合は、値を書き込む際にあいまいにならないように他の既存の変数はこのアドレス (I/O マッピングの %Qxx) に保存されるべきではありません。

既存変数 xBool_4 への出力マッピング例を参照してください。

注記 : 既存の変数にマッピングする場合は、変数の初期化値がデフォルト値として使用されます。**デフォルト値**フィールドは、新しい変数にマッピングするとき、またはマッピングがない場合に編集できます。

- 新規変数を定義する場合は、使用する変数名を入力します。
  例 : bVar1

この場合、  シンボルが **マッピング**列に挿入され変数は内部的にグローバル変数として宣言されます。この時点からこの変数はアプリケーション内でグローバルに使用できます。マッピングダイアログはグローバル変数宣言を行うもう 1 つの場所です。
注記：また、アドレスは ST (構造化テキスト) などのプログラムコード内で読み込みおよび書き込みが可能です。

- デバイス設定の変更の可能性を考慮し、マッピングはデバイス設定ダイアログで行います。

注記：マッピングダイアログで UNION が I/O チャンネルで表されている場合、ルート要素のマッピングが可能かどうかはデバイスによって異なります。

指定されたデータ型に宣言された変数が、マップ先のデータ型よりも大きい場合、マップ先の値にはマップ先のサイズに切り捨てられた値が割り当てられます。

例えば、変数が WORD データ型で宣言され、BYTE にマップされた場合、WORD の 8 ビットのみが BYTE に割り当てられます。

これは、マッピングダイアログの値を監視する場合にアドレスのルート要素に表示された値は、(プロジェクトで現在有効な) 宣言された変数の値になることを意味します。ルートの下にあるサブ要素では特定のマップされた変数の要素の値が監視されます。ただし、宣言された値の一部のみがサブ要素に表示されます。

さらに、宣言された変数を物理的な出力にマップすることも意味します。同様に、データ型が出力データ型よりも大きい場合、出力データ型は切り捨てられた値を受信する可能性があるため、アプリケーションに意図しない影響を及ぼすかもしれません。

⚠ 警告

装置の意図しない動作

物理的 I/O にマップされている宣言されたデータ型がマシンの意図した動作と互換性があることを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

要素	詳細
Reset mapping	このボタンをクリックするとマッピング 設定をデバイスディスクリプションファイルで定義されたデフォルト値にリセットします。
常に変数を更新	バス周期タスク (*110* ページ) で I/O 変数が更新された場合の定義。デフォルト値は、デバイス説明で定義されます。 ● **Use parent device settings**: 親デバイスの設定に従って更新します。 ● **有効 1 (タスクで使用されていない場合はバス周期タスクを使用)**: I/O 変数は、他のタスクで使用されていない場合バス周期タスクで更新されます。 ● **有効 2 (バス周期タスクでは常に)**: 変数は、使用されているか、または、入力または出力チャンネルにマップされているかにかかわらず、バス周期タスクの周期ごとに更新されます。

IEC オブジェクト

タブのこの部分は、デバイスオブジェクトのファンクションブロックインスタンスが暗黙的に作成された場合にのみ使用できます。このインスタンスは (例えば、バスを再起動や情報のポーリングのために) アプリケーションからアクセスできます。そのようなインスタンスが使用可能かどうか、またどのように使用するかはコントローラーによって異なります。各コントローラーについては、*プログラミングガイド*を参照してください。

バスサイクルオプション

このオプションは、入力または出力の読み込み前後のサイクリックコールが可能なデバイスに使用できます。デバイス特有なバスサイクルタスク (*110* ページ) の設定に使用します。

デフォルトでは、親バスサイクル設定が有効になります (**親バスサイクル設定を使用します**)。これは、次の有効なバスサイクルタスクの定義のために**デバイスツリー**が検索されることを意味します。

特定のバスサイクルタスクを割り当てるには、選択リストから選択してください。リストにはアプリケーションタスク設定で定義されているタスクが表示されます。

オンラインモードの I/O マッピング

オンラインモードの I/O マッピング

構造体変数がアドレスのルート要素 (マッピングダイアログの各アドレスツリーの最上位) にマップされている場合、オンラインモードではこの行に値は表示されません。ただし、例えば、DWORD 変数がこのアドレスにマップされている場合、ルートの行の値とその下に展開されたビットチャンネルの行の値が監視されます。基本的に、値が複数のサブ要素で構成される場合はルートの行は空になります。

I/O の強制用暗黙的変数

概要

工場やマシンの試運転中に、I/O の強制が必要になることがあります。これを目的として、デバイスエディターの **I/O マッピング**タブで変数にマップされた各 I/O チャンネルのための特別な強制変数を生成することができます。

前提条件として、**PLC 設定**タブの **IO マッピング用強制変数の生成**設定は有効になっている必要があります。アプリケーションのビルドを実行するたびに、マップされた I/O チャンネルにつき 2 つの変数が下記の構文で作成されます。チャンネル名にスペースが使用されている場合はアンダースコアに置き換えられます。

< デバイス名 >_< チャンネル名 >_<IEC アドレス >_Force、BOOL タイプ、強制を有効 / 無効にするために使用します。

< デバイス名 >_< チャンネル名 >_<IEC アドレス >_Value、チャンネルのデータ型、チャンネルに強制する値を定義します。

これらの変数は入力アシスタントの**変数 → IoConfig_Globals_Force_Variables** カテゴリーで使用可能です。これらは、プログラミングシステム内のすべてのプログラミングオブジェクト、ビジュアライゼーション、シンボル設定などで使用できます。

強制変数の立上がりで、Value (値) 変数に定義された値での I/O の強制を有効にします。立下りで強制を無効にします。新しい値で強制する前に、Force (強制) 変数の設定を FALSE にして無効にする必要があります。

下記の制限に注意してください。

例

デバイスエディターの **I/O マッピング** タブの図 (*125* ページ) のようにマッピングさされている場合は、アプリケーションのビルド (F11) で次の変数が入力アシスタントで使用できます。

- Digitax_ST_Control_word_QW0_Force : BOOL;
- Digitax_ST_Control_word_QW0_Value : UINT;
- Digitax_ST_Target_position_QD1_Force : BOOL;
- Digitax_ST_Target_position_QD1_Value : DINT;
- Digitax_ST_Status_word_IW0_Force : BOOL;
- Digitax_ST_Status_word_IW0_Value : UINT;
- Digitax_ST_Position_actual_value_ID1_Force : BOOL;
- Digitax_ST_Position_actual_value ID1_Value : DINT;

制限事項

- **I/O マッピング**タブで変数にマップされたチャンネル (すなわち変数は新規でも既存でも**変数**列で定義する必要があります) のみが上記の暗黙的変数によって強制できます。
- 未使用およびアプリケーションプログラムの AT 宣言でマップされた入力 / 出力は強制できません。
- 各 I/O チャンネルは少なくとも 1 つのタスクで使用されている必要があります。
- 強制 I/O は監視 (ウォッチビュー、I/O マッピングダイアログ) で表示されません。値は I/O ドライバーでデバイスに書き込みをするためにのみ暗黙的に使用されます。
- 強制入力は、強制入力 / 出力ではなく、赤い強制シンボル (**F**) で表示されます。

第 III 部
プログラム

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
7	プログラムコンポーネント	133
8	タスク設定	191
9	アプリケーションの管理	195

第 7 章
プログラムコンポーネント

この章について

この章には次のセクションが含まれています。

7.1
プログラム構成単位 (POU)

このセクションについて

このセクションには次の項目が含まれています。

POU

概要

プログラム構成単位 (POU) という用語は、コントローラーアプリケーションの作成に使用されるすべてのプログラミングオブジェクト (プログラム、ファンクションブロック、ファンクションなど) に使用されます。

IEC 61131 規格の定義では、POU はプログラム、ファンクションブロック、またはファンクションです。本書では、*POU* は IEC コードを含むプログラミング要素である、メソッド、プロパティ、インターフェイスなどを意味します。本書で POU を IEC 61131 で定義された限定された意味合いで使用する場合は *POU オブジェクト* と表記しています。

POU 管理

アプリケーションツリーの**グローバル**ノードで管理される POU はデバイス固有ではありませんが、デバイス (アプリケーション) 上で使用するためにインスタンス化できます。そのためには、プログラム POU を対応するアプリケーションのタスクから呼び出してください。

POU は、**オブジェクトの追加**メニューにあるこれらのオブジェクトの特定のサブカテゴリーの名前でもあります。ここでは、プログラム、ファンクションブロック、およびファンクションのみで構成されています。

従って、一般的には POU オブジェクトはプログラグ単位です。これは、**アプリケーションツリー**の**グローバル**ノードでデバイスに非固有に管理されるか、または**アプリケーションツリー**のアプリケーションの下で直接管理されるオブジェクトです。エディタービューで表示および編集できます。POU オブジェクトは、プログラム、ファンクション、またはファンクションブロックです。

特定の POU オブジェクトごとに特定の**プロパティ** (ビルド条件など) を設定できます。

POU オブジェクトの作成方法については、*アプリケーションに POU オブジェクトを追加* (*136* ページ) の章を参照してください。作成した POU オブジェクトは、**ソフトウェアカタログ**の**アセット**ビューに追加されます。

2 種類の方法によって、**アセット**ビューにある POU オブジェクトをプロジェクトに追加できます。

- **アセット**ビューの POU オブジェクトを選択し、**アプリケーションツリー**の適切なノードにドラッグします。
- **アセット**ビューの POU オブジェクトを選択し、ロジックエディタービュー (*227* ページ) にドラッグします。

POU オブジェクトの他に、対象システム (**リソース**、**アプリケーション**、**タスク設定**など) でプログラムを実行するために使用されるデバイスオブジェクトがあります。それらは**アプリケーションツリー**の中で管理されます。

POU オブジェクトの追加および呼び出し

概要

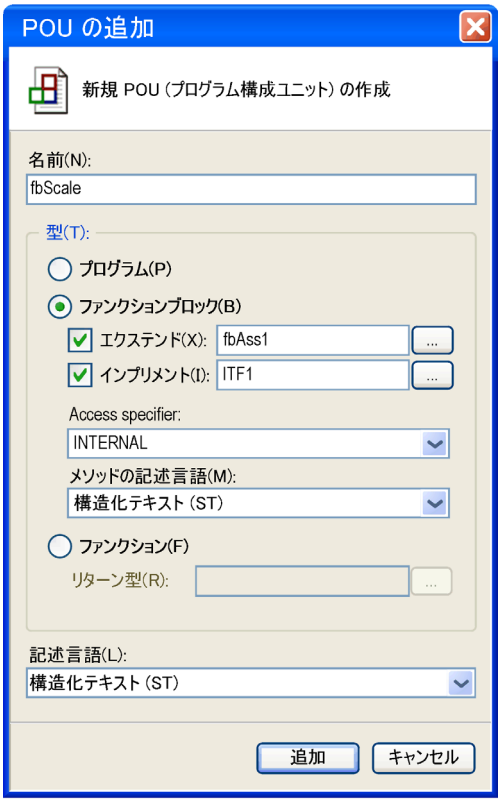
ソフトウェアカタログ → アセットまたは**アプリケーションツリー**で、プログラム構成単位 (POU) オブジェクトを追加できます。

POU オブジェクトの種類は次のとおりです。

- **プログラム** : 処理中に 1 つまたは複数の値を返します。プログラムの最後の実行から次の実行まですべての値が保持されます。別の POU オブジェクトから呼び出すことができます。
- **ファンクションブロック** : プログラムの処理中に 1 つまたは複数の値を供給します。ファンクションとは異なり、出力変数と必要な内部変数の値が、ファンクションブロックの実行から次の実行まで保持されます。従って、同じ引数 (入力パラメーター) のファンクションブロックの呼び出しにより常に同じ出力値が得られるとは限りません。
- **ファンクション**: 処理後にデータ要素 (複数のフィールドまたは構造体のような要素で構成されている場合もあり) を 1 つ生成します。テキスト言語での呼び出しは、式の演算子として実行されます。

POU オブジェクトをアプリケーションに追加

POU オブジェクトをコントローラーのアプリケーションに追加するには、以下の手順に沿って進めます。

手順	手順内容
1	**ソフトウェアカタログ → アセット → POU** セクションで、**アプリケーション**ノードを選択して緑色のプラスボタンをクリックし、**POU...** コマンドを実行します。代わりに、コントローラーの**アプリケーション**ノードを右クリックし、**オブジェクトの追加 → POU** を選択することもできます。この 2 つの方法は、**アプリケーションツリー**でも使用できます。 **結果** : **POU の追加**ダイアログボックスが開きます。 
2	**POU の追加**ダイアログボックスの**名前**のテキストフィールドに名前を入力して POU オブジェクトに名前を付けます。 **注記**：名前にスペース文字を使用しないでください。名前を入力しない場合は、初期設定の名前になります。 POU に意味のある名前を付けることで、プロジェクトの構成が容易になります。

手順	手順内容
3	POU オブジェクトの種類を選択します。 ● **プログラム** ● **ファンクションブロック**： **a.** 既存のファンクションブロックを拡張したい場合は、**Extends** を選択しブラウザーをクリックして**入力アシスタント**の中からファンクションブロックを選択します。 **b.** **OK** ボタンをクリックします。 **c.** ファンクションブロックでインターフェイスを実装したい場合は、**実装** を選択しブラウザーをクリックして**入力アシスタント**の中からインターフェイスを選択します。 **d.** **OK** ボタンをクリックします。 **e.** ファンクションブロックでインターフェイスを実装する場合は、**メソッドの記述言語**リストボックスで、実装インターフェイスで定義されたメソッドやプロパティを編集するプログラミング言語を選択します。 ● **ファンクション**： **a.** 参照ボタンをクリックして、**入力アシスタント**から**リターン型**を選択します。 **b.** **OK** ボタンをクリックします。
4	**記述言語**のリストボックスから、POU オブジェクトの編集に使用するプログラミング言語を選択します。
5	**開く**ボタンをクリックします。

すでに定義されている POU オブジェクトは、**ソフトウェアカタログ → アセット → POU** セクションにリスト表示されます。POU を**アプリケーションツリー**にドラッグして**アプリケーション**ノードにドロップすることで、アプリケーションに追加できます。POU オブジェクトをロジックエディターにドロップすることもできます。

プログラムのタスクへの割り当て

タスクには、少なくとも 1 つプログラムを割り当ててください。タスクにプログラムを追加するには、以下の手順に沿って進めます。

手順	手順内容
1	コントローラーの**タスク設定**ノードの下から、プログラムを追加するタスクをダブルクリックします。**設定**タブで、**呼び出しの追加**をクリックします。 または、**アプリケーションツリー**で、プログラムに割り当てるタスクノードを選択し、緑色のプラスボタンをクリックします。リストから **Program Call...** コマンドを実行します。**...** ボタンをクリックします。 **結果：入力アシスタント**ダイアログボックスが表示されます。
2	**入力アシスタント**ダイアログボックスの**カテゴリー**タブで、**プログラム**を選択します。
3	**構造化ビュー**チェックボックスのチェックをはずします。
4	**アイテム**パネルで、必要な POU を選択します。
5	**OK** ボタンをクリックします。

POU の呼び出し

POU から他の POU を呼び出すことができます。ただし、再帰 (自身を呼び出す POU) はできません。

アプリケーションに割り当てられた POU から別の POU を名前のみ (名前空間 (*673* ページ) の追加なし) で呼び出す場合は、以下にある POU を呼び出すためのプロジェクトの参照順序を考慮してください。

1.	現在のアプリケーション
2.	**ツールツリー**にある現在のアプリケーションの**ライブラリーマネージャー**
3.	**アプリケーションツリー**の**グローバル**ノード
4.	**ツールツリー**の**グローバル**ノードにある**ライブラリーマネージャー**

呼び出しで指定された名前の POU がアプリケーションの**ライブラリーマネージャー**のライブラリーだけでなく、**アプリケーションツリー**の**グローバル**ノードのオブジェクトにもある場合、明示的に**アプリケーションツリー**の**グローバル**ノードの POU を呼び出す構文はなく、名前のみを使用して呼び出します。この場合、対応するライブラリーをアプリケーションの**ライブラリーマネージャー**から**アプリケーションツリー**の**グローバル**ノードの**ライブラリーマネージャー**に移動させます。その結果、**アプリケーションツリー**の**グローバル**ノードから名前のみで (必要に応じて、ライブラリーからライブラリーの名前空間を前に付けて) POU を呼び出せます。

暗黙的チェック用 POU (*153* ページ) の章も参照してください。

プログラム

概略

プログラムは、処理中に 1 つまたは複数の値を返す POU オブジェクトです。プログラムの最後の実行から次の実行まですべての値が保持されます。ただし、ファンクションブロックとは異なり、プログラムの個別のインスタンスはありません。ファンクションブロックを呼び出すと、ファンクションブロックの指定されたインスタンスの値のみが変更されます。変更は、再度同じインスタンスが呼び出されたときにのみ影響します。別の POU から呼び出された場合でも、プログラム値の変更は次回呼び出されるときまで保持されます。

プログラムの追加

プログラムを既存のアプリケーションに追加するには、**アプリケーションツリー**のアプリケーションノードを選択し、緑色のプラスボタンをクリックして **POU...** コマンドを実行します。代わりに、**アプリケーション**ノードを右クリックして、コンテキストメニューから**オブジェクトの追加 → POU** コマンドを実行することもできます。アプリケーションに依存しない POU を追加するには、**アプリケーションツリーのグローバル**ノードを選択し、同じコマンドを実行します。

POU の追加ダイアログボックスで**プログラム**オプションを選択し、プログラムの名前を入力して記述言語を選択します。**開く**をクリックして確認します。新しいプログラム用のエディタービューが開き、プログラムの編集を開始できます。

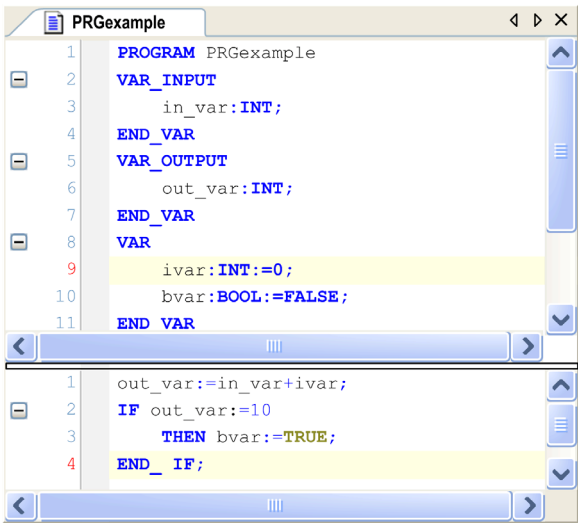
プログラムの宣言

構文 :

PROGRAM < プログラム名 >

この後に、入力 (*493* ページ) 変数、出力 (*493* ページ) 変数、およびプログラ変数の変数宣言が続きます。オプションとしてアクセス変数も使用できます。

プログラムの例



プログラムの呼び出し

プログラムは別の POU から呼び出すことができます。ただし、ファンクション (*141* ページ) 内でプログラムの呼び出しはできません。プログラムのインスタンスはありません。

POU がプログラムを呼び出したときにプログラムの値が変更された場合、プログラムが再び呼び出されるまでこれらの変更が保持されます。これは、別の POU 内から呼び出された場合にも適用されます。これはファンクションブロックの呼び出しとは異なります。ファンクションブロックを呼び出すと、ファンクションブロックの指定されたインスタンスの値のみが変更されます。変更は、再度同じインスタンスが呼び出されたときにのみ影響します。

プログラムの呼び出しの途中で入力および出力パラメーターを設定するには、テキスト言語エディター (例、ST) で値を丸括弧内のプログラム名の後のパラメーターに値を割り当てます。入力パラメーターの場合は、宣言位置の変数の初期化 (*486* ページ) のように割り当てに := を使用します。出力パラメーターの場合は、=> を使用します。次の例を参照してください。

テキスト言語エディターの実装ビューで、**Insert with arguments** オプションを使用して**入力アシスタント**からプログラムを挿入すると、この構文に従ってすべてのパラメーターが自動的に表示されます。ただし、必ずしもすべてのパラメーターを割り当てる必要はありません。

プログラム呼び出しの例

IL のプログラム :

```
CAL             PRGexample              (
      in_var:= 33                       )
LD              PRGexample.out_var
ST              erg
```

パラメーターの割り当ての例 (**Insert with arguments** オプションを使用した**入力アシスタント**):

引数付き IL のプログラム :

```
CAL             PRGexample              (
      in_var:= 33                       ,
     out_var=> erg                      )
```

ST の例

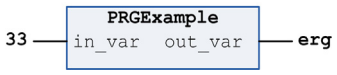
```
PRGexample(in_var:= 33);
erg := PRGexample.out_var;
```

パラメーターの割り当ての例 (前に説明されている **Insert with arguments** オプションを使用した**入力アシスタント**):

```
PRGexample (in_var:=33, out_var=>erg );
```

FBD の例

FBD のプログラム :



ファンクション

概要

ファンクションは、処理後にデータ要素 (複数のフィールドまたは構造体のような要素で構成されている場合もあり) を 1 つ生成する POU です。テキスト言語での呼び出しは、式の演算子として実行されます。

ファンクションの追加

ファンクションを既存のアプリケーションに割り当てるには、**アプリケーションツリー**のアプリケーションノードを選択し、緑色のプラスボタンをクリックして **POU...** コマンドを実行します。代わりに、**アプリケーション**ノードを右クリックして、コンテキストメニューから**オブジェクトの追加 → POU** コマンドを実行することもできます。アプリケーションに依存しない POU を追加するには、**アプリケーションツリー**の**グローバル**ノードを選択し、同じコマンドを実行します。

POU の追加 ダイアログボックスで、**ファンクション**オプションを選択します。新しいファンクションの**名前** (< ファンクション名 >) および **Return Data Type** (< データ型 >) を入力し、記述言語を選択します。戻り値のデータ型を選択するには、**...** ボタンをクリックして**入力アシスタント ...** ダイアログボックスを開きます。**開く**をクリックして確認します。新しいファンクション用のエディタービューが開き、編集を開始できます。

ファンクションの宣言

構文 :

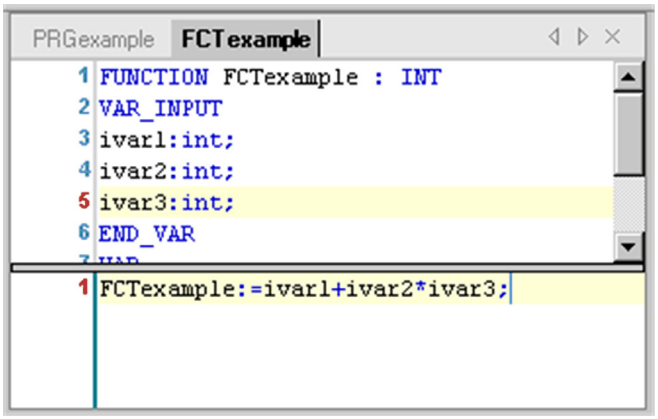
FUNCTION < ファンクション名 > : < データタイプ >

この後に、入力変数およびファンクション変数の変数宣言が続きます。

結果をファンクションに割り当てます。これは、ファンクション名が出力変数として使用されることを意味します。

ファンクション内ではローカル変数を RETAIN または PERSISTENT として宣言しないでください。効果がありません。

ST のファンクション例 : このファンクションは 3 つの入力変数をとり、最後の 2 つの変数の積を最初の変数に加算したものを返します。



ファンクションの呼び出し

ST でのファンクションの呼び出しは、式のオペランドとして表示できます。

IL では、ステップのアクションまたは遷移内にのみファンクションの呼び出しを配置できます。

ファンクションには (プログラムまたはファンクションブロックとは異なり)、内部状態情報が含まれていません。つまり、同じ引数 (入力パラメーター) をもつファンクションの呼び出しは常に同じ値 (出力) を返します。そのため、ファンクションにグローバル変数およびアドレスを含めることはできません。

IL でのファンクション呼び出しの例

IL でのファンクション呼び出し

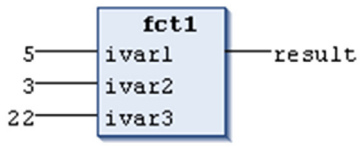
```
LD          5
Fct         3           ,
           22
ST           result
```

ST でのファンクション呼び出しの例

```
result := fct1(5,3,22);
```

FBD でのファンクション呼び出しの例

FBD でのファンクション呼び出し



例 :

```
fun(formal1 := actual1, actual2); // -> error message
fun(formal2 := actual2, formal1 := actual1); // same semantics as the following:
fun(formal1 := actual1, formal2 := actual2);
```

IEC 61131-3 規格に準じて、ファンクションに出力を追加することができます。それらはファンクションの呼び出しで割り当てられます。例えば、ST では次の構文に従います。

out1 => < 出力変数 1> | out2 => < 出力変数 2> | ... さらに出力変数

例

ファンクション fun は 2 つの入力変数 in1 および in2 と、2 つの出力変数 out1 および out2 で定義されます。fun の出力値は、ローカルで宣言された変数 (loc1 および loc2 に書き込まれます。

```
fun(in1 := 1, in2 := 2, out1 => loc1, out2 => loc2);
```

メソッド

概要

メソッドとはファンクションブロック (*154* ページ) のコンテキストで使用できるファンクションに似た言語要素です。各ファンクションブロックのインスタンスを含むファンクションと見なすことができます。ファンクションのように、メソッドには戻り値、一時変数およびパラメーター用の宣言があります。

また、オブジェクト指向プログラミングの手法として、インターフェイス (*147* ページ) を使用してプロジェクトで使用可能なメソッドを整理できます。

注記：メソッドまたはプロパティを POU からインターフェイスにコピーまたは移動すると、含まれている実装は自動的に削除されます。インターフェイスから POU にコピーまたは移動するときに、記述言語を指定してください。

メソッドの挿入

メソッドをファンクションブロックまたはインターフェイスに割り当てるには適切なファンクションブロックまたは**アプリケーションツリー**のインターフェイスノードを選択し、緑色のプラスボタンをクリックして**メソッド**コマンドを実行します。または、ファンクションブロックまたはインターフェイスノードを右クリックし、コンテキストメニューの**オブジェクトの追加 → メソッド**コマンドを実行することもできます。

メソッドの追加ダイアログボックスで、**名前**、**戻るタイプ**、**記述言語**、および**アクセス指定子** (以下参照) を入力します。戻り値のデータ型を選択するには、**...** ボタンをクリックして**入力アシスタント ...** ダイアログボックスを開きます。

アクセス指定子 : 互換性のため、アクセス指定子はオプションです。指定子 **PUBLIC** は、指定子なしの設定と同等に使用できます。

または、選択リストからオプションを 1 つ選択します。

- **PRIVATE**: メソッドへのアクセスはファンクションブロックに制限されています。
- **PROTECTED**: メソッドへのアクセスはファンクションブロックおよびその派生に制限されています。
- **INTERNAL**: メソッドへのアクセスは現在の名前空間 (ライブラリー) に制限されています。

開くをクリックして確認します。メソッドエディタービューが開きます。

継承ファンクションブロック作成時の入力アシスタント

EcoStruxure Machine Expert は、ファンクションブロック内の継承を使用することで、オブジェクト指向プログラミングを円滑に行えるようにします : 別のファンクションブロックから継承したファンクションブロックの **オブジェクトの追加** を実行すると、**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素が選択リストに表示されます :

- ベースとなるファンクションブロックで定義された**アクセス指定子 PUBLIC**、**PROTECTED**、**INTERNAL** を用いた**アクション**、**メソッド**、**プロパティ**、および**遷移**の要素が選択できます。継承したオブジェクトの定義は調整することができます。継承したオブジェクトでは、ソース要素と同じ**アクセス指定子**が割り当てられます。
- **アクセス指定 PRIVATE** を用いた**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素は、アクセスがベースのファンクションブロックのみに制限されているため、選択することはできません。

メソッドの宣言

構文 :

METHOD < アクセス指定子 > < メソッド名 > : < 返り値データタイプ >VAR_INPUT ... END_VAR

メソッドを処理するインターフェイスの宣言方法については、*インターフェイス*の章 (*147* ページ) を参照してください。

メソッドの呼び出し

メソッドの呼び出しは仮想ファンクション呼び出しとも呼ばれます。詳細については、*メソッドの呼び出し* (*165* ページ) の章を参照してください。

メソッドを呼び出す場合は、次の点に注意してください。

- メソッドのデータは一時的で、メソッドの実行中のみ有効です (スタック変数)。これは、メソッドで宣言された変数とファンクションブロックは、メソッドの呼び出しのたびに再初期化されることを意味します。
- インターフェイス (*147* ページ) で定義されたメソッドは入力変数、出力変数、および入出力変数のみが許可され、本体 (実装部) をもつことは許可されていません。
- ファンクションなどのメソッドは、追加の出力をもつことができます。*メソッドの呼び出し* (*165* ページ) 中に割り当ててください。

メソッドの実施

メソッドを実装する場合は、次の点に注意してください。

- メソッドの本体では、ファンクションブロックインスタンス変数へのアクセスが許可されています。
- 必要に応じて、常に現在のインスタンスを指す THIS ポインター (*168* ページ) を使用します。
- ファンクションブロックの VAR_TEMP 変数はメソッドではアクセスできません。

ファンクションブロックの特別なメソッド

メソッド	詳細
Init	FB_init という名前のメソッドは初期設定で暗黙的に宣言されていますが、明示的にも宣言できます。ファンクションブロックの宣言部で宣言されたようにファンクションブロックの初期化コードが含まれています。FB_init メソッド (*501* ページ) を参照してください。
Reinit	FB_reinit という名前のメソッドがファンクションブロックインスタンス用に宣言されている場合、インスタンスがコピーされた後 (**オンライン変更**中など) に呼び出され、新しいインスタンスモジュールがを再初期化します。FB_init, FB_reinit メソッド (*501* ページ) を参照してください。
Exit	FB_exit という名前の Exit メソッドが指定された場合は、明示的に宣言してください。暗黙的宣言はありません。Exit メソッドは、新しいダウンロード、リセット、またはすべての移動または削除されたインスタンスのオンライン変更前に、ファンクションブロックのインスタンスごとに呼び出されます。FB_exit メソッド (*501* ページ) を参照してください。

プロパティ (*145* ページ) およびインターフェイスのプロパティ (*148* ページ) にはそれぞれ、Set および Get アクセサーメソッドで構成されています。

アプリケーションが停止したときのメソッド呼び出し

デバイスディスクリプションファイルでは、特定のメソッドが (ライブラリーモジュールの) 特定のファンクションブロックインスタンスによって常にタスクサイクル的に呼び出されるように定義できます。このメソッドに次の入力パラメーターがある場合、有効なアプリケーションが実行されていないときにも処理されます。

例

```
VAR_INPUT
pTaskInfo : POINTER TO DWORD;
pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO;
END_VAR
```

プログラマーは、pApplicationInfo からアプリケーション状態を確認でき、何が発生するかを定義できます。

```
IF pApplicationInfo^.state = RUNNING THEN <instructions> END_IF
```

プロパティ

概要

オブジェクト指向プログラミングの手法として IEC 61131-3 の拡張プロパティを使用できます。一組のアクセサーメソッド (Get, Set) で構成されています。変数アクセスの構文を維持しながら、POU または、GVL 内部で宣言された変数への読み取りまたは書き込みアクセスをファンクションの呼び出しにカプセル化できます。

プロパティをプログラム (*139* ページ)、ファンクションブロック (*154* ページ)、GVL (*172* ページ)、またはインターフェイス (*147* ページ) ノードの下にオブジェクトとして挿入するには、**アプリケーションツリー**でノードを選択し、緑色のプラスボタンをクリックして、**プロパティ**コマンドを実行します。代わりに、ノードを右クリックして、コンテキストメニューから**オブジェクトの追加 → プロパティ** コマンドを実行することもできます。

Add Property ダイアログボックスで、**名前**、**リターン型**、**記述言語**、およびオプションで **Access Specifier** を指定します。

メソッド (*143* ページ) と同じアクセス指定子を使用できます。

- **PUBLIC**
- **PRIVATE**
- **PROTECTED**
- **INTERNAL**

注記：プロパティは、インターフェイス内で定義することもできます。

EcoStruxure Machine Expert は、ファンクションブロック内の継承を使用することで、オブジェクト指向プログラミングを円滑に行えるようにします : 別のファンクションブロックから継承したファンクションブロックの **オブジェクトの追加** を実行すると、**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素が選択リストに表示されます。

- ベースとなるファンクションブロックで定義された**アクセス指定子 PUBLIC**、**PROTECTED**、**INTERNAL** を用いた**アクション**、**メソッド**、**プロパティ**、および**遷移**の要素が選択できます。継承したオブジェクトの定義は調整することができます。継承したオブジェクトでは、ソース要素と同じ**アクセス指定子**が割り当てられます。
- **アクセス指定 PRIVATE** を用いた**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素は、アクセスがベースのファンクションブロックのみに制限されているため、選択することはできません。

プロパティの Get および Set アクセサー

アクセサーという 2 つの特別なメソッド (*143* ページ) が、プロパティオブジェクトの下の**アプリケーションツリー**に自動的に挿入されます。プロパティを書き込み専用または読み取り専用に使用する場合は、2 つのうち 1 つを削除できます。アクセサーを明示的に追加すると、プロパティ (前の項を参照) のようにアクセサーは宣言部または**オブジェクトの追加**ダイアログボックスでアクセス修飾子を割り当てられます。

- プロパティが書き込まれると Set アクセサリーが呼ばれます。
- プロパティが読み込まれると Get アクセサリーが呼ばれます。

例 :

ファンクションブロック FB1 には、seconds プロパティがあり、ローカル変数 milli を使用します。この変数は、プロパティ Get および Set によって決まります。

Get 実装例

```
seconds := milli / 1000;
```

Set 実装例

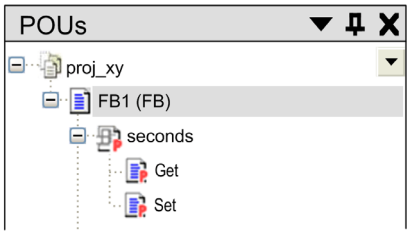
```
milli := seconds * 1000;
```

ファンクションブロックのプロパティに書き込むことができます (Set メソッド)、例えば
`fbinst.seconds := 22;`

(fbinst は FB1 のインスタンスです)。

ファンクションブロックのプロパティを読み込むことができます (Get メソッド)、例えば
`testvar := fbinst.seconds;`

次の例では、プロパティ seconds がファンクションブロック FB1 に割り当てられています。



プロパティには追加のローカル変数はありますが、追加の入力はありません。また、ファンクション (*141* ページ) またはメソッド (*143* ページ) とは異なり追加の出力もありません。

注記：メソッドまたはプロパティを POU からインターフェイスにコピーまたは移動すると、含まれている実装は自動的に削除されます。インターフェイスから POU にコピーまたは移動するときに、記述言語を指定してください。

プロパティの監視

プロパティは、インライン監視 (*324* ページ) またはウォッチリスト (*386* ページ) によってオンラインモードで監視できます。プロパティを監視するための前提条件は、定義の上に pragma {attribute 'monitoring' := 'variable'} (属性の監視 (*536* ページ) の章を参照) を追加することです。

インターフェイス

概要

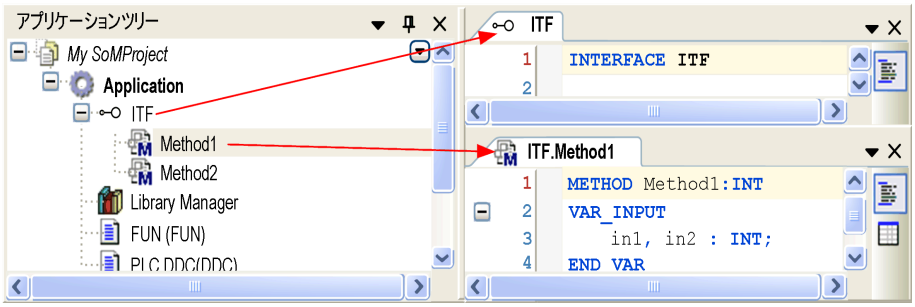
インターフェイスの使用は、オブジェクト指向プログラミングの手法です。インターフェイス POU はメソッド (*143* ページ)、およびプロパティ (*145* ページ) のセットを実装せずに定義します。インターフェイスは、ファンクションブロック (*154* ページ) の空のシェルとして記述できます。ファンクションブロックインスタンスで実現するには、ファンクションブロックの宣言で実装 (*163* ページ) してください。ファンクションブロックは、複数のインターフェイスを実装できます。

同じメソッドを、同じパラメーターと異なるファンクションブロックによる異なる実装コードで実現できます。従って、POU が関連する特定のファンクションブロックを識別する必要なしに、任意の POU でインターフェイスを使用 / 呼び出しできます。

インターフェイスの定義とファンクションブロックでの使用例

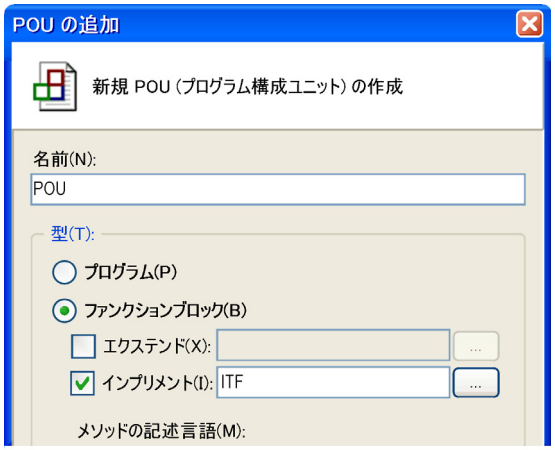
インターフェイス IFT はアプリケーションの下に挿入されます。2 つのメソッド Method1 および Method2 が含まれています。インターフェイスおよびメソッドに実装コードは含まれていません。メソッドの宣言部分に変数宣言を入れるだけです。

メソッドが 2 つのインターフェイス



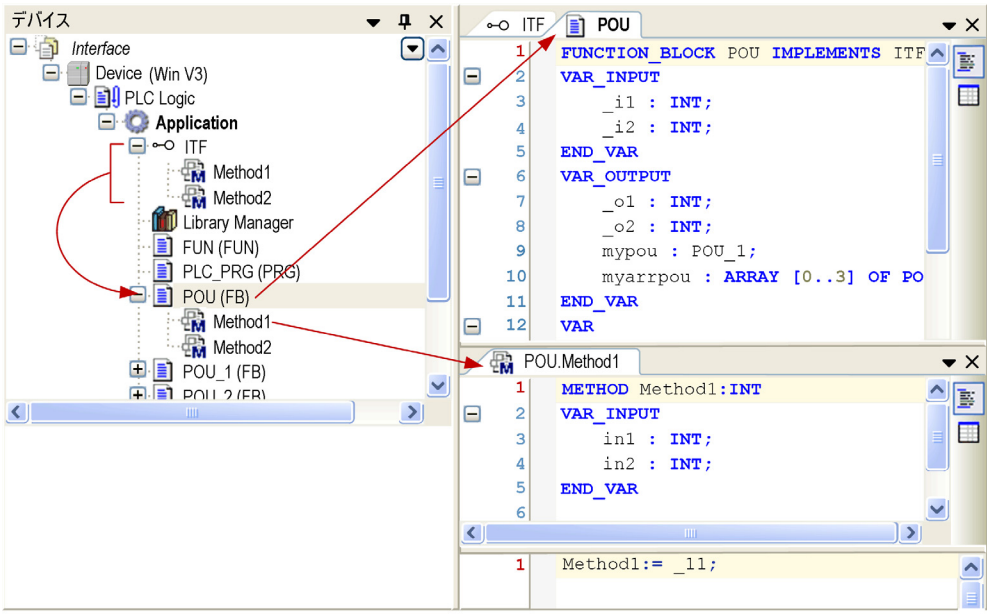
複数のファンクションブロックを挿入し、上記の定義されたインターフェイス ITF を実装できます。

インターフェイスを実装するファンクションブロックの作成



ファンクションブロック POU が**アプリケーションツリー**に追加されると、メソッド Method1 および Method2 が ITF の下に自動的に挿入されます。ここで、それらにファンクションブロック固有の実装コードを入力できます。

ファンクションブロック定義でのインターフェイスの使用



インターフェイスは、EXTENDS (次の例、*インターフェイスの拡張例* (*149* ページ) を参照してください) を使用して他のインターフェイスを拡張できます。

インターフェイスのプロパティ

インターフェイスは、アクセサーメソッド Get および Set で構成されるインターフェイスプロパティも定義できます。プロパティの詳細については、*プロパティ* (*145* ページ) の章を参照してください。メソッドを含む場合があるようなインターフェイスのプロパティは、実装コードを含まないことを意味する単なるプロトタイプです。メソッドと同様に、自動的にインターフェイスを実装するファンクションブロックに追加されます。そこに、特定のプログラミングコードを入力できます。

注意事項

以下を考慮してください。

- インターフェイス内で変数は宣言できません。インターフェイスには本体 (実装部分) およびアクションはありません。インターフェイス内でメソッドのコレクションが定義され、これらのメソッドには入力変数、出力変数、および入出力変数のみをもつことができます。
- インターフェイスタイプで宣言された変数は、参照として扱われます。
- インターフェイスを実装するファンクションブロックには、インターフェイス内の名前と同じ名前の割り当てられたメソッドおよびプロパティが必要です。それらには、同じ名前の入力、出力、および入出力を含めてください。

注記：メソッドまたはプロパティを POU からインターフェイスにコピーまたは移動すると、含まれている実装は自動的に削除されます。インターフェイスから POU にコピーまたは移動するときに、記述言語を指定してください。

インターフェイスの挿入

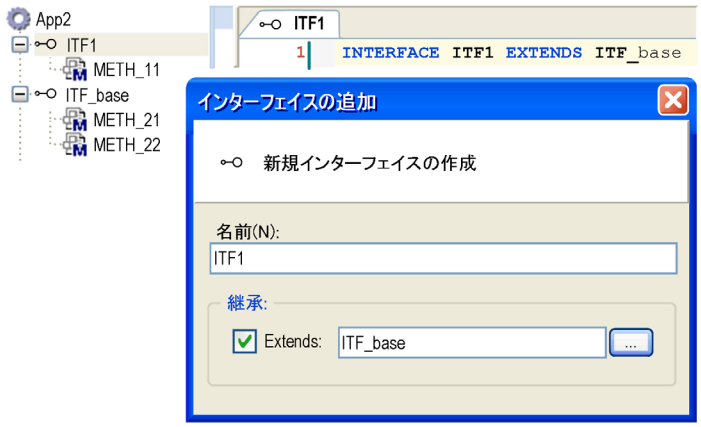
アプリケーションにインターフェイスを追加するには、**アプリケーションツリー**または**ソフトウェアカタログ → アセットのアプリケーション**ノードを選択し、緑色のプラスボタンをクリックして **Add Other Objects... → インターフェイス**を選択します。または、**オブジェクトの追加 → インターフェイス**コマンドを実行します。コマンドを実行する前に**グローバル**ノードを選択すると、新しいインターフェイスがすべてのアプリケーションで使用できます。

Add Interface ダイアログボックスで、新しいインターフェイスの名前 (< インターフェイス名 >) を入力します。オプションとして、現在のインターフェイスを別のインターフェイスの拡張 (*161* ページ) にする場合は**拡張**オプションを有効にできます。

インターフェイスの拡張例

ITF1 が ITF_base を拡張する場合、ITF_base で記述されたすべてのメソッドが自動的に ITF1 で使用可能になります。

インターフェイスの拡張



追加をクリックして、設定を確認します。新しいインターフェイスのエディタービューが開きます。

インターフェイスの宣言

構文

INTERFACE < インターフェイス名 >

別のインターフェイスを拡張するインターフェイスの場合

INTERFACE < インターフェイス名 > EXTENDS < 基になるインターフェイス名 >

例

INTERFACE interface1 EXTENDS interface_base

メソッドコレクションの追加

インターフェイスの定義を完了するには、メソッドコレクションを追加します。そのためには、**アプリケーションツリー**でインターフェイスノードを選択し、**Interface method...** コマンドを実行します。インターフェイスに含めるメソッドを定義するための **Add Interface Method** ダイアログボックスが開きます。または、**アプリケーションツリー**のインターフェイスノードを選択し、緑色のプラスボタンをクリックして**インターフェイスメソッド**を選択します。必要な数のメソッドを追加します。これらのメソッドは、入力変数、出力変数、および入出力変数のみをもつことができ、本体 (実装部) はもつことができないことに留意してください。

手順内容

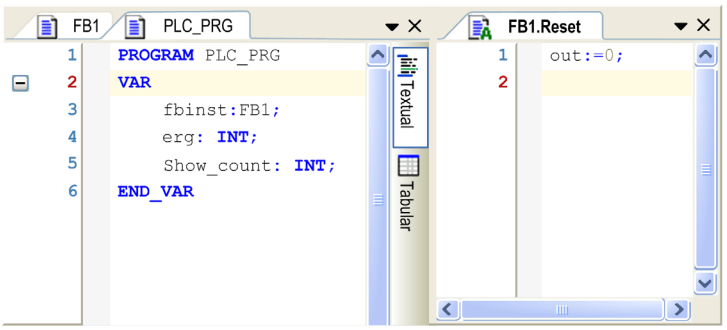
概要

アクションを定義し、それをファンクションブロック (*154* ページ) およびプログラム (*139* ページ) に割り当てることができます。アクションは追加実装です。基本的な実装とは異なる言語で作成できます。各アクションには名前が付けられます。

アクションは、それが属しているファンクションブロックまたはプログラムのデータで動作します。定義された入出力変数およびローカル変数を使用し、独自の宣言は含みません。

ファンクションブロックのアクションの例

次の図は、FB でのアクションを示しています。



この例では、ファンクションブロック FB1 を呼び出すごとに出力変数 out が、入力変数 in の値に応じて増減します。ファンクションブロックの Reset アクションの呼び出しによって、出力変数 out が 0 に設定されます。どちらの場合も同じ変数 out が書き込まれます。

アクションの挿入

アクションを追加するには、**アプリケーションツリー**または**アプリケーションツリーのグローバル**ノードから対応するプログラムまたはファンクションブロックを選択し、緑色のプラスボタンをクリックして**アクション ...** コマンドを実行します。または、プログラムまたはファンクションブロックを右クリックして**オブジェクトの追加 → アクション**コマンドを実行します。**アクションの追加**ダイアログボックスで、アクションの**名前**および**記述言語**を定義します。

EcoStruxure Machine Expert は、ファンクションブロック内の継承を使用することで、オブジェクト指向プログラミングを円滑に行えるようにします : 別のファンクションブロックから継承したファンクションブロックの **オブジェクトの追加** を実行すると、**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素が選択リストに表示されます。

- ベースとなるファンクションブロックで定義された**アクセス指定子 PUBLIC**、**PROTECTED**、**INTERNAL** を用いた**アクション**、**メソッド**、**プロパティ**、および**遷移**の要素が選択できます。継承したオブジェクトの定義は調整することができます。継承したオブジェクトでは、ソース要素と同じ**アクセス指定子**が割り当てられます。
- **アクセス指定 PRIVATE** を用いた**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素は、アクセスがベースのファンクションブロックのみに制限されているため、選択することはできません。

アクションの呼び出し

構文

<Program_name>.<Action_name>

または

<Instance_name>.<Action_name>

FBD での表記法について考慮します (次の例を参照)。

自身のブロック内、すなわちアクションが属するプログラムまたはファンクションブロック内でアクションを呼び出す必要がある場合はアクション名を使用するだけです。

例

このセクションでは、別の POU から上記のアクションを呼び出す例を示します。

すべての例の宣言 :

```
PROGRAM PLC_PRG
VAR
   Inst : Counter;
END_VAR
```

IL でプログラムされた別の POU でアクション Reset を呼び出し

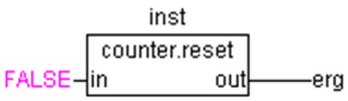
```
CAL Inst.Reset(In := FALSE)
LD Inst.out
ST ERG
```

ST でプログラムされた別の POU でアクション Reset を呼び出し

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

FBD でプログラムされた別の POU でアクション Reset を呼び出し

FBD のアクション



注記：IEC 規格では、シーケンシャルファンクションチャート (SFC) のアクション以外のアクションは認識されません。これらのアクションは、チャートの特定のステップで処理される命令を含む必須部分です。

遷移

概要

遷移オブジェクトは、SFC で実装されたプログラムまたはファンクションブロック内の遷移要素として使用できます。

詳細は、*SFC 要素 / ツールボックス* (*305* ページ) の章のステップと遷移の説明を参照してください。

EcoStruxure Machine Expert は、ファンクションブロック内の継承を使用することで、オブジェクト指向プログラミングを円滑に行えるようにします : 別のファンクションブロックから継承したファンクションブロックの **オブジェクトの追加** を実行すると、**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素が選択リストに表示されます :

- ベースとなるファンクションブロックで定義された**アクセス指定子 PUBLIC**、**PROTECTED**、**INTERNAL** を用いた**アクション**、**メソッド**、**プロパティ**、および**遷移**の要素が選択できます。継承したオブジェクトの定義は調整することができます。継承したオブジェクトでは、ソース要素と同じ**アクセス指定子**が割り当てられます。
- **アクセス指定 PRIVATE** を用いた**アクション**、**メソッド**、**プロパティ**、および **遷移**の要素は、アクセスがベースのファンクションブロックのみに制限されているため、選択することはできません。

暗黙的チェック用 POU

概要

暗黙的に利用可能なファンクションの確認を提供するには、アプリケーションの下に特別な POU を追加する必要があります。それらは配列と範囲の境界、ポインターの有効性を検証し、ランタイムでのゼロによる除算の存在を検証します。

注記：ファンクションの確認を提供するとランタイムのパフォーマンスに影響を及ぼします。

注記：ファンクションの確認が未コンパイルの参照ライブラリーのコードを検証するには、**Compiler defines** の文字列 checks_in_libs をアプリケーションの**プロパティ** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**ビルド**タブで入力します。

監視機能をアプリケーションの下に追加するには、**アプリケーションツリー**でアプリケーションノードを選択し緑色のプラスボタンをクリックするか **オブジェクトの追加 → POU for implicit checks...** コマンドを実行します。

ダイアログボックスでは、選択したアプリケーション用で次のファンクションの確認を有効にできます。

カテゴリー	追加されたファンクション
範囲の確認	CheckBounds (配列用) (*574* ページ)
除算の確認	● CheckDivDInt (*598* ページ) ● CheckDivLInt (*598* ページ) ● CheckDivReal (*598* ページ) ● CheckDivLreal (*598* ページ)
範囲の確認	● CheckRangeSigned (*581* ページ) ● CheckRangeUnsigned (*581* ページ)
L 範囲の確認	● CheckLRangeSigned (*581* ページ) ● CheckLRangeUnsigned (*581* ページ)
ポインターの確認	CheckPointer (*570* ページ)

チェック用 POU を挿入した後は、重複挿入を避けるためにダイアログボックスではこのオプションを使用できなくなります。別のタイプのチェック用 POU がアプリケーションの下に挿入されている場合、**オブジェクトの追加**ダイアログボックスに**暗黙的チェック用 POU** オプションは表示されません。

⚠ 注意

不適切な暗黙的チェック機能

機能的な整合性を維持するために、暗黙的チェックファンクションの宣言部分を変更しないでください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

注記：ただし、暗黙的なファンクションの確認の宣言部にローカル変数を追加することはできます。

注記：暗黙的チェックファンクション (CheckBounds など) をアプリケーションから削除した後は、**オンライン変更**はできず、ダウンロードのみができます。適切なメッセージが表示されます。

7.2
ファンクションブロック

このセクションについて

このセクションには次の項目が含まれています。

一般情報

概要

ファンクションブロックは、コントローラープログラムの処理中に 1 つまたは複数の値を提供する POU (*135* ページ) です。ファンクションとは異なり、出力変数と必要な内部変数の値が、ファンクションブロックの実行から次の実行まで保持されます。従って、同じ引数 (入力パラメーター) のファンクションブロックの呼び出しによって常に同じ出力値が得られるとは限りません。

IEC11631-3 規格に記述されている機能に加え、オブジェクト指向プログラミングに対応し、ファンクションブロックは他のファンクションブロックの拡張 (*161* ページ) として定義できます。メソッド呼び出し (*165* ページ) に関するインターフェイス (*163* ページ) 定義を含むことができます。そのため、ファンクションブロックを使用してプログラミングするときに継承を使用できます。

ファンクションブロックは常に、ファンクションブロックの複製 (コピー) であるインスタンス (*157* ページ) を通して呼び出されます。

ファンクションブロックの追加

既存のアプリケーションにファンクションブロックを追加するには、**ソフトウェアカタログ → アセット**または**アプリケーションツリー**の対応するノードを選択し、緑色のプラスボタンをクリックして **POU...** を選択します。または、ノードを右クリックして**オブジェクトの追加 → POU** コマンドを実行します。アプリケーションに依存しないファンクションブロックを作成するには、**アプリケーションツリー**または**アセット**の**グローバル**ノードを選択します。

オブジェクトの追加ダイアログボックスで、**ファンクションブロック**オプションを選択してファンクションブロックの**名前** (< 識別子 >) を入力し、**記述言語**を選択します。

さらに、次のオプションも設定できます。

オプション	詳細
拡張	現在のファンクションブロックの基になる、プロジェクトで使用可能な別のファンクションブロックの名前を入力します。詳細については、*ファンクションブロックの拡張* (*161* ページ) を参照してください。
実装	現在のファンクションブロックで実装する、プロジェクトで使用可能なインターフェイス (*147* ページ) の名前を入力します。カンマで区切って複数のインターフェイスを入力できます。詳細については、*インターフェイスの実装* (*163* ページ) を参照してください。
アクセス指定子	互換性のため、アクセス指定子はオプションです。指定子 **PUBLIC** は、指定子なしの設定と同等に使用できます。 または、選択リストからオプションを 1 つ選択します。 ● **INTERNAL**: ファンクションブロックへのアクセスは現在の名前空間 (ライブラリー) に制限されています。 ● **FINAL**: 派生アクセスはできません。すなわち、ファンクションブロックを別のファンクションブロックで拡張できません。最適化されたコード生成を可能にします。 **注記：** アクセス指定子は、コンパイラーバージョン 3.4.4.0 以降で有効なため、以前のバージョンでは識別子として使用できます。 詳細については、EcoStruxure Machine Expert Compatibility and Migration User Guide *(EcoStruxure Machine Expert Compatibility and Migration, User Guide 参照)* の EcoStruxure Machine Expert/CoDeSys コンパイラーバージョンマッピングテーブルを参照してください。
メソッド記述言語	ファンクションブロックに対して設定されたものとは別に、インターフェイス実装により作成されたすべてのメソッドオブジェクトとプロパティオブジェクトに対してプログラミング言語を選択します。

追加をクリックして、設定を確認します。新しいファンクションブロック用のエディタービューが開き、編集を開始できます。

ファンクションブロックの宣言

構文

FUNCTION_BLOCK < アクセスアクセス指定子 > < ファンクションブロック名 > | EXTENDS < ファンクションブロック名 > | IMPLEMENTS < カンマ区切りのインターフェイス名のリスト >

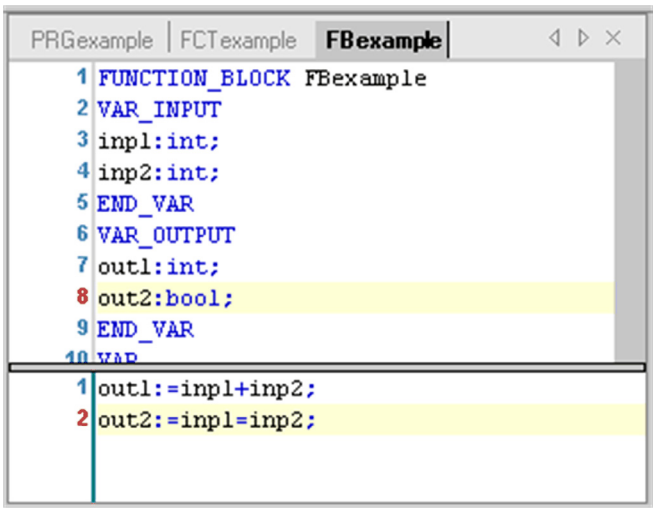
この後に変数の宣言が続きます。FBD または、LD エディターでファンクションブロックが使用されている場合は、入力と出力をグループ化してすばやくフェードアウトまたはフェードインすることができます。*属性ピングループ*の章 (*548* ページ) も参照してください。

例

次の図の FBexample には、2 つの入力変数と 2 つの出力 変数 out1 および out2 があります。

out1 は 2 つの入力の和、out2 は等式の比較結果です。

ST でのファンクションブロックの例



ファンクションブロックインスタンス

概要

ファンクションブロックは、ファンクションブロック (*155* ページ) の複製 (コピー) であるインスタンスを通して呼び出され (*158* ページ) ます。

各インスタンスは、独自の識別子 (インスタンス名) およびその入力、出力、内部変数を含むデータ構造をもっています。

変数のようにインスタンスは、ローカルまたはグローバルで宣言されます。ファンクションブロック名は、識別子のデータ型として示されます。

ファンクションブロックインスタンスの宣言用構文

< 識別子 >:< ファンクションブロック名 >;

例

ファンクションブロック FUB のインスタンス INSTANCE の宣言 (例えば、プログラムの宣言部分):

```
INSTANCE: FUB;
```

ファンクションブロックとプログラムの宣言部分にインスタンス宣言を含めることができます。

ファンクションブロックの呼び出し

概要

ファンクションブロック (*155* ページ) は、ファンクションブロックインスタンスを介して呼び出されます。そのため、ファンクションブロックインスタンスはローカルまたはグローバルにで宣言してください。宣言の方法については、*ファンクションブロックインスタンス* (*157* ページ) の章を参照してください。

その後、次の構文を使用してファンクションブロック変数にアクセスできます。

構文

< インスタンス名 >.< 変数名 >

注意事項

- 内部変数ではなくファンクションブロックインスタンスの外部から、ファンクションブロックの入力変数および出力変数にのみにアクセスできます。
- ファンクションブロックインスタンスへのアクセスは、それがグローバルに宣言されていない限り、宣言した POU (*135* ページ) に限定されます。
- インスタンスの呼び出し時に、ファンクションブロックのパラメーターに値を割り当てることができます。*呼び出し時のパラメーターの割り当て*の項を参照してください。
- ファンクションブロックの入力 / 出力変数 (VAR_IN_OUT) は、ポインターとして渡されます。
- SFC では、ファンクションブロックの呼び出しはステップでのみ実行できます。
- ファンクションブロックインスタンスのインスタンス名は、ファンクションまたは別のファンクションブロックの入力パラメーターとして使用できます。
- ファンクションブロックの値はすべて、ファンクションブロックの次の処理まで保持されます。従って、ファンクションブロックの呼び出しは、たとえ同じ引数で実行しても常に同じ出力値を返す訳ではありません。

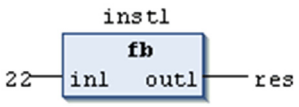
注記：少なくとも 1 つのファンクションブロック変数が残存変数である場合、すべてのインスタンスが保持データ領域に格納されます。

ファンクションブロック変数へのアクセスの例

ファンクションブロック fb には、INT タイプの入力変数 in1 があるとします。ここで、プログラム prog 内からのこの変数の呼び出しを見ます。ST での宣言と実装を参照してください。

```
PROGRAM prog
VAR
inst1:fb;
END_VAR
inst1.in1:=22;   (* fb is called and input variable in1 gets assigned value 22 *)
inst1(); (* fb is called, this is needed for the following access on the output variable *)
res:=inst1.outl; (* output variable of fb is read *)
```

FBD でのファンクションブロック呼び出しの例



呼び出し時のパラメーターの割り当て

テキスト言語 IL および ST では、ファンクションブロックを呼び出すとすぐに入力および出力パラメーターを設定できます。この値は、ファンクションブロックのインスタンス名の後にある丸括弧内のパラメーターに割り当てることができます。入力パラメーターの場合は、割り当ては宣言位置で変数の初期化 (*486* ページ) と同じようにのように := を使用して実行されます。出力パラメーターの場合は、=> が使用されます。

割り当てを伴う呼び出しの例

この例では、タイマーファンクションブロック (インスタンス CMD_TMR) がパラメーター IN および PT の割り当てと共に呼び出されます。次に、結果変数 Q が変数 A に割り当てられます。結果変数は、ファンクションブロックのインスタンス名、続いてピリオド、そして変数の名前で指定されます。

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

引数付き入力アシスタントによる挿入の例

ST または IL POU の実装ビューで **With arguments** オプションを使用して**入力アシスタント**からインスタンスを挿入すると、インスタンスはそのパラメーターと共に次の例に示されている構文に従って自動的に表示されます。ただし、必ずしもこれらのパラメーターを割り当てる必要はありません。

前述の例では、呼び出しは次のように表示されます。

```
CMD_TMR(in:=, pt:=, q=>)
-> fill in, e.g.:
CMD_TMR(in:=bvar, pt:=t#200ms, q=>bres);
```

ファンクションブロックのオンライン変更のためのメモリー予約の設定

概要

ファンクションブロックのオンライン変更のためにメモリー予約の設定ができます。ファンクションブロックの宣言を変更した後 (特に新しい変数を追加した後)、オンラインで変更を行った後は、ファンクションブロックのインスタンスを新しいメモリー領域にコピーする必要はありません。したがって、オンラインでの変更はより迅速に行われ、検出されるエラーは少なくなります。

設定したメモリー予約が使い果たされると、オンライン変更が実行される前にメッセージが表示されます。

アプリケーションをコントローラーにダウンロードする前に、ファンクションブロック用のメモリー予約を設定することを推奨します。アプリケーションがすでにコントローラー上に配置されているときにメモリー予約を設定した場合は、オンライン変更を実行する必要があります。

ファンクションブロックのオンライン変更のためのメモリー予約の設定手順

オンライン変更の実行中に、ファンクションブロックのインスタンスを他のメモリーロケーションにコピーする必要のあるファンクションブロックで変更を後でする場合は、**Online Change Memory Reserve Settings** コマンドを使用できます。

デフォルトでは、メニューでこのコマンドを使用できません。**ツール → カスタマイズ** メニューからこのコマンドを追加できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

手順	手順内容	コメント
1	**ビュー → Online Change Memory Reserve Settings** コマンドを実行します。	**結果** : **オンライン変更用メモリー予約設定** ビューオプションが開きます。
2	アプリケーションの一覧から適切なアプリケーションを選択します。	–
3	**ビルド**メニューから**ビルド**コマンドを実行します。	–
4	**Scan Application** ボタンをクリックします。	–
5	**ファンクションブロック**エリアから**すべて**のエントリーを選択します。	**結果** : アプリケーションのファンクションブロックは **Online Change Memory Reserve** ビューの中央に表示されます。
6	メモリー予約を設定するファンクションブロックを選択します。	アプリケーションがまだコントローラーにダウンロードされていない場合は、入力フィールドの **Memory reserve (in bytes)** は編集ができます。アプリケーションが既にコントローラーで使用可能な場合は、**編集**ボタンを **Enable editing** エリアでクリックします。 **注記 :** すでにコントローラーで使用可能なアプリケーションのメモリー予約を変更した場合は、影響を受けるすべてのファンクションブロックのインスタンスをコピーする必要があります。
7	**Memory reserve (in bytes)** に値を入力し、**Apply for selection** ボタンをクリックします。	**結果** : 入力した値が表の **Memory Reserve** セルに表じされます。
8	**ビルド**メニューから**ビルド**コマンドを実行します。	–
9	**Scan Application** ボタンをクリックします。	**結果** : **サイズ**、**Instance Count**,、**Additional memory for all instances**、および **Remaining size of memory reserve Memory Reserve** のセルの値が設定したファンクションブロックのリストで変更されます。

アプリケーションをコントローラーにダウンロードすると、ファンクションブロック用に予約されているメモリーサイズは、ファンクションブロックの現在のサイズに加えて **Online Change Memory Reserve** ビューで設定したメモリー予約量によって計算されます。

今後のファンクションブロックの変更は、ファンクションブロックのすべてのインスタンスを新しいメモリー領域にコピーする必要なしに、オンラインの変更でコントローラーにダウンロードできます。

ファンクションブロックの拡張

概要

オブジェクト指向プログラミングに対応することにより、ファンクションブロックから別のファンクションブロックを派生させることができます。これは、ファンクションブロックが別のファンクションブロックを拡張できるため、自身のプロパティに加えて基にしているファンクションブロックのメソッド、プロパティ、アクションおよび遷移も自動的に取得できることを意味します。

拡張は、ファンクションブロックの宣言でキーワード EXTENDS を使用して実行します。**オブジェクトの追加**ダイアログボックスでファンクションブロックをプロジェクトに追加するときに、EXTENDS オプションを選択できます。

詳細は、以下の追加方法が記述されたセクション (メソッド (*143* ページ)、プロパティ (*145* ページ)、アクション (*150* ページ)、または遷移 (*152* ページ)) を参照してください。

構文

FUNCTION_BLOCK < ファンクションブロック名 > EXTENDS < ファンクションブロック名 >

この後に変数の宣言が続きます。

例

ファンクションブロック fbA の定義

```
FUNCTION_BLOCK fbA
VAR_INPUT
   x:int;
END_VAR
...
```

ファンクションブロック fbB の定義

```
FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
   ivar: INT := 0;
END_VAR
...
```

EXTENDS による拡張

EXTENDS による拡張とは以下を意味します。

- fbB には fbA で定義されるすべてのデータおよびメソッドが含まれています。fbB のインスタンスは、fbA タイプのファンクションブロックが必要とされるあらゆるコンテキストでも使用できます。
- fbB は、fbA. で定義されたメソッド、プロパティ、アクション、または遷移をオーバーライドできます。つまり、fbB は、同じ名前および A で宣言された同じ入力と出力のメソッドを宣言できます。
- fbB は、fbA で使用されているものと同じ名前のファンクションブロック変数を定義できません。この場合、コンパイラーでエラーメッセージが生成されます。
- fbA 変数およびメソッドは、SUPER ポインター (*166* ページ) (SUPER^.<method>) を使用して fbB スコープ内から直接アクセスできます。

注記：ファンクションブロックは複数のインターフェイスを実装できますが、拡張できるのは 1 つのファンクションブロックのみです。

例

```
FUNCTION_BLOCK FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
   iCnt : INT;
END_VAR
VAR
END_VAR
THIS^.METH_DoIt();
THIS^.METH_DoAlso();

   METHOD METH_DoIt : BOOL
```

```
    VAR
    END_VAR
    iCnt := -1;
    METH_DoIt := TRUE;

    METHOD METH_DoAlso : BOOL
    VAR
    END_VAR
    METH_DoAlso := TRUE;
FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
// Calls the method defined under FB_1
THIS^.METH_DoIt();
THIS^.METH_DoAlso();
// Calls the method defined under FB_Base
SUPER^.METH_DoIt();
SUPER^.METH_DoAlso();
    METHOD METH_DoIt : BOOL
    VAR
    END_VAR
    iCnt := 1111;
    METH_DoIt := TRUE;
PROGRAM PLC_PRG
VAR
    Myfb_1: FB_1;
    iFB: INT;
    iBase: INT;
END_VAR
Myfb_1();
iBase := Myfb_1.iCnt_Base;
iFB := Myfb_1.iCnt_THIS;
```

インターフェイスの実装

概要

オブジェクト指向プログラミングに対応するため、ファンクションブロックにメソッド (*143* ページ) を使用できるインターフェイス (*147* ページ) を実装できます。

構文

FUNCTION_BLOCK < ファンクションブロック名 > IMPLEMENTS < インターフェイス _1 名前 >|,< インターフェイス _2 名前 >, ..., < インターフェイス _n 名前 >

インターフェイスを実装するファンクションブロックには、このインターフェイスで定義されたすべてのメソッドおよびプロパティ (インターフェイスプロパティ (*148* ページ)) を含めてください。そこに含まれる特定のメソッドまたはプロパティの名前、入力、および出力は完全に同じにしてください。

このような目的のために、インターフェイスを実装する新しいファンクションブロックを作成すると、このインターフェイスで定義されたすべてのメソッドおよびプロパティが**アプリケーションツリー**の新しいファンクションブロックの下に自動的に挿入されます。

注記：その後にメソッドがインターフェイス定義に追加された場合は、関連するファンクションブロックは自動的に追加されませんので、コンパイルエラーが発生します。明示的にこの更新を実行するには、Implement interfaces... *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドを実行してください。
ファンクションブロックに継承を使用する際は、メソッドまたは属性がインターフェイスの継承によって作成されたものである場合はそれらを実行するか、もしくはベースになるファンクションブロックの実行を使用する場合は削除することをご検討ください。pragma の属性は自動的に挿入されます。これらはコンパイル中に検出され、継承されたメソッドやプロパティを検証する必要があることを知らせるメッセージが生成されます。新しいファンクションブロックの実装完了後、pragma の属性を削除します。

詳細については、**インターフェイスの実装 ...** の説明を参照してください。コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

例

INTERFACE I1 に GetName メソッド含まれています。

```
METHOD GetName : STRING
```

ファンクションブロック A および B はそれぞれインターフェイス I1 を実装します。

```
FUNCTION_BLOCK A IMPLEMENTS I1
FUNCTION_BLOCK B IMPLEMENTS I1
```

そのためファンクションブロックが**アプリケーションツリー**に挿入されるときに、両方のファンクションブロックで GetName メソッドが使用可能である必要があり、それらは自動的に各ファンクションブロックの下に挿入されます。

I1 タイプの変数の宣言を考えてみます。

```
FUNCTION DeliverName : STRING
VAR_INPUT
  I_i : I1;
END_VAR
```

この入力は、インターフェイス I1 を実装するすべてのファンクションブロックを受け取れます。

ファンクション呼び出しの例 :

```
DeliverName(I_i := A_instance); // call with instance of type A
DeliverName(I_i := B_instance); // call with instance of type B
```

注記：メソッドが呼び出される前に、インターフェイスタイプの変数をファンクションブロックのインスタンスに割り当ててください。インターフェイスタイプの変数は常に、割り当てられたファンクションブロックインスタンスへの参照です。

従って、インターフェイスメソッドを呼び出すと、ファンクションブロックの実装が呼び出されます。参照が割り当てられるとすぐに、オンラインモードで対応するアドレスが監視されます。参照がまだ割り当てられていない場合、オンラインモードの監視には値 0 が表示されます。

この例では、DeliverName ファンクションの実装部分を参照してください。

```
DeliverName := I_i.GetName(); // in this case it depends on the "real" type of I_i whether A.GetName or B.GetName is called
```

注記: 宣言でキーワード EXTENDS を使用したファンクションブロックの拡張 (*161* ページ) の可能性についても参照してください。

メソッドの呼び出し

概要

ファンクションブロックを使用したオブジェクト指向プログラミングは、EXTENDS によって拡張 (*161* ページ) できることに加え、インターフェイス (*163* ページ) および継承を使用することに対応しています。これには、仮想ファンクション呼び出しとも呼ばれる、動的に解決されるメソッド呼び出しが必要です。

仮想ファンクション呼び出しは、通常のファンクション呼び出しよりも時間が掛かり、次の場合に使用します。

- ファンクションブロックへのポインターを介して呼び出しが実行されるとき (pfub^.method)
- インターフェイス変数のメソッドが呼び出されるとき (interface1.method)
- メソッドが、同じファンクションブロックの別のメソッドを呼び出すとき
- ファンクションブロックへの参照を介して呼び出しが実行されるとき
- 基本ファンクションブロックタイプの VAR_IN_OUT は、派生されたファンクションブロックタイプのインスタンスに割り当てることができます。

仮想ファンクション呼び出しにより、実行中にプログラムソースコードの同じ呼び出しで異なるメソッドを呼び出すことができます。

詳細情報およびビューの詳細については、次を参照してください。

- メソッドについての詳細は、*メソッド* (*143* ページ)。
- THIS ポインターの使用については、*THIS ポインター* (*168* ページ)。
- SUPER ポインターの使用については、*SUPER ポインター* (*166* ページ)。

メソッドの呼び出し

IEC 61131-3 規格に準じて、標準ファンクション (*141* ページ) のようなメソッドは、追加出力をもつことができます。それらはファンクションの呼び出しで割り当てることができます。

< メソッド >(in1:=< 値 > |, さらに入力割り当て , out1 => < 出力変数 1> | out2 => < 出力変数 2> | ... さらに出力変数)

これにより、メソッドの出力が呼び出し内で与えられローカルで宣言された変数に書き込まれます。

例

ファンクションブロック fub1 および fub2 EXTEND ファンクションブロック fubbase および IMPLEMENT interface1 と仮定します。メソッド method1 を含みます。

インターフェイスおよびメソッド呼び出しの使用

```
PROGRAM PLC_PRG
VAR_INPUT
  b : BOOL;
END_VAR
VAR
  pInst : POINTER TO fubbase;
  instBase : fubbase;
  inst1 : fub1;
  inst2 : fub2;
  instRef : REFERENCE to fubbase;
END_VAR
IF b THEN
  instRef REF= inst1;          (* Reference to fub1 *)
  pInst := ADR(instBase);
ELSE
  instRef REF= inst2;          (* Reference to fub2 *)
  pInst := ADR(inst1);
END_IF
pInst^.method1();              (* If b is true, fubbase.method1 is called, else fub1.method1 is called *)
instRef.method1();             (* If b is true, fub1.method1 is called, else fub2.method1 is called *)
```

上の例の fubbase には、2 つのメソッド method1 および method2 があります。fub1 は method2 をオーバーライドするが、method1 にはしません。

method1 は、上の例で示すように呼び出されます。

pInst^.method1(); (* If b is true fubbase.method1 is called, else fub1.method1 is called *)

THIS ポインター経由の呼び出しについては、*THIS ポインター* (*168* ページ) を参照してください。

SUPER ポインター

概要

基本ファンクションブロックを拡張するファンクションブロックごとに SUPER という名前のポインターが自動的に使用可能になります。基本ファンクションブロックのインスタンスをポイントします。

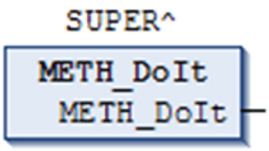
これにより、次の問題を効果的に解決します。

- SUPER により、基本ファンクションブロック実装のメソッドへアクセスできます。SUPER というキーワードを使って、ベース (親) クラスのインスタンスで有効なメソッドを呼び出すことができます。そのため、動的な名前の結合はしません。

SUPER は、メソッドおよび関連するファンクションブロックの実装でのみ使用できます。

SUPER は基本ファンクションブロックへのポインターであるため、ファンクションブロックのアドレスを取得するためにはデリファレンスする必要があります。SUPER^.METH_DoIt

異なる記述言語での SUPER の呼び出し

記述言語	例
ST	SUPER^.METH_DoIt();
FBD/CFC/LD	

注記：SUPER 機能は、まだ IL には実装されていません。

例

SUPER および THIS ポインターの使用

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
   iCnt : INT;
END_VAR
   METHOD METH_DoIt : BOOL
   iCnt := -1;

   METHOD METH_DoAlso : BOOL
   METH_DoAlso := TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_OUTPUT
   iBase: INT;
END_VAR
// Calls the method defined under FB_1
THIS^.METH_DoIt();
THIS^.METH_DoAlso();
// Calls the method defined under FB_Base
SUPER^.METH_DoIt();
SUPER^.METH_DoAlso();
iBase := SUPER^.iCnt;

   METHOD METH_DoIt : BOOL
   iCnt := 1111;
   METH_DoIt := TRUE;

PROGRAM PLC_PRG
VAR
```

```
    myBase: FB_Base;
    myFB_1: FB_1;
    iTHIS: INT;
    iBase: INT;
END_VAR
myBase();
iBase := myBase.iCnt;
myFB_1();
iTHIS := myFB_1.iCnt;
```

THIS ポインター

概要

各ファンクションブロックでは、THIS という名前のポインターが自動的に使用可能です。独自のファンクションブロックインスタンスを指します。

これにより、次の問題を効果的に解決します。

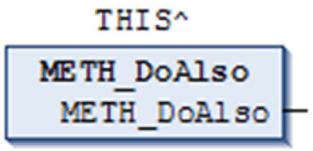
- メソッド内でローカルに宣言された変数がファンクションブロック変数を隠す場合。
- ファンクション内で使用するために、独自のファンクションブロックインスタンスへのポインターを参照する場合。

THIS は、メソッド、プロパティ、アクション、遷移、および関連するファンクションブロックの実装でのみ使用できます。

THIS は大文字で書いてください。他の綴りは許可されません。

THIS は基本ファンクションブロックへのポインターであるため、ファンクションブロックのアドレスを取得するためにはデリファレンスする必要があります。THIS^.METHDoIt

異なる記述言語での THIS の呼び出し

記述言語	例
ST	THIS^.METH_DoIt();
FBD/CFC/LD	

注記：THIS 機能は、まだ IL には実装されていません。

例 1

ローカル変数 iVarB が、ファンクションブロック変数 iVarB をシャドーイングします。

```
FUNCTION_BLOCK fbA
VAR_INPUT
   iVarA: INT;
END_VAR
iVarA := 1;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
   iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;

   METHOD DoIt : BOOL
   VAR_INPUT
   END_VAR
   VAR
      iVarB: INT;
   END_VAR
   iVarB := 22; // Here the local iVarB is set.
```

```
    THIS^.iVarB := 222; // Here the function block variable iVarB is set, although iVarB is overloaded.

PROGRAM PLC_PRG
VAR
    MyfbB: fbB;
END_VAR

MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();
```

例 2

独自のインスタンスへの参照が必要なファンクションの呼び出し。

```
FUNCTION funA
VAR_INPUT
    pFB: fbA;
END_VAR
...;

FUNCTION_BLOCK fbA
VAR_INPUT
    iVarA: INT;
END_VAR
...;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    iVarB: INT := 0;
END_VAR
iVarA := 11;
iVarB := 2;

    METHOD DoIt : BOOL
    VAR_INPUT
    END_VAR
    VAR
        iVarB: INT;
    END_VAR
    iVarB := 22;    //Here the local iVarB is set.
    funA(pFB := THIS^);    //Here funA is called with THIS^.

PROGRAM PLC_PRG
VAR
    MyfbB: fbB;
END_VAR
MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();
```

7.3
アプリケーションオブジェクト

このセクションについて

このセクションには次の項目が含まれています。

データタイプユニット (DUT)

概要

標準データタイプと併せて、独自のデータ型を定義できます。DUT エディター (*349* ページ) でデータタイプユニット (DUT) として、構造体 (*577* ページ)、列挙型 (*579* ページ)、およびリファレンス (*568* ページ) を作成できます。

特定の標準データタイプおよびユーザー定義データ型の詳細については、データ型 (*560* ページ) の説明を参照してください。

DUT オブジェクトの追加

既存のアプリケーションに DUT オブジェクトを追加するには、**ソフトウェアカタログ → アセット → タイプ**、または**アプリケーションツリー**で緑色のプラスボタンをクリックして **DUT...** を選択します。または、各ノードを右クリックし、**オブジェクトの追加 → DUT** を実行します。アプリケーションに依存しない DUT オブジェクトを作成するには、**アセット**または**アプリケーションツリー**の**グローバル**ノードを選択します。**Add DUT** ダイアログボックスに新しいデータタイプユニットの**名前**を入力し、**Structure**、**Enumeration**、**Alias**、または **Union** から必要なタイプを選択します。

列挙型の DUT オブジェクトにはテキストリストサポートを追加できます。詳細については、**テキストリストサポートコマンドの追加** の説明を参照してください。*(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*

Structure タイプの場合、継承の概念を使用できるので、オブジェクト指向プログラミングに対応します。オプションで、DUT がプロジェクトですでに定義されている別の DUT を拡張するように指定できます。これは、拡張された DUT の定義は自動的に現在のプロジェクトで有効になります。このためには、**Extends:** オプションを有効にし、他の DUT の名前を入力します。

追加をクリックして、設定を確認します。新しい DUT 用のエディタービューが開き、編集を開始できます。

DUT オブジェクトの宣言

構文

TYPE < 識別し > : <DUT コンポーネント宣言 >END_TYPE

DUT コンポーネントの宣言は、構造体 (*577* ページ) または列挙体 (*579* ページ) などの DUT のタイプにより異なります。

例

次の例には、構造体 `struct1` および `struct2` を定義する 2 つの DUT があります。`struct2` は `struct1` を拡張し、変数 `a` にアクセスする実装で `struct2.a` を使用できます。

```
TYPE struct1 :
   STRUCT
      a:INT;
      b:BOOL;
   END_STRUCT
END_TYPE
TYPE struct2 EXTENDS struct1 :
   STRUCT
  c:DWORD;
  d:STRING;
   END_STRUCT
END_TYPE
```

グローバル変数リスト - GVL

概要

グローバル変数リスト (GVL) は、グローバル変数 (*494* ページ) を宣言するために使用します。**ソフトウェアカタログ → 変数 → グローバル変数**、または**アプリケーションツリー**の**グローバル**ノードに GVL を配置すると、プロジェクト全体で変数を使用できます。GVL が特定のアプリケーションに割り当てられている場合、変数はそのアプリケーション内で有効です。

既存のアプリケーションに GVL を追加するには、**ソフトウェアカタログ → アセット → POU** または **アプリケーションツリー** のアプリケーションノードを選択して、緑色のプラスボタンをクリックして **ネットワーク変数リスト ...** を選択します。または、ノードを右クリックし、**オブジェクトの追加 → グローバル変数リストの追加 ...** コマンドを実行することもできます。これらのビューで **グローバル** ノードを選択すると、新しい GVL オブジェクトはアプリケーションに依存しません。

グローバル変数リストを編集するには、GVL エディター (*351* ページ) を使用します。

GVL に含まれる変数は、ネットワークの他のデバイスとのブロードキャストデータ交換用にネットワーク変数 (*755* ページ) として使用できるように定義できます。このためには、GVL 用に適切なネットワークプロパティ (メニューの **ビュー → プロパティ → ネットワーク変数**、または**ネットワーク変数送信機のプロパティ**) を設定します。

注記：ネットワーク変数の最大サイズは 255 バイトです。ネットワーク変数の数に制限はありません。

注記：GVL で宣言された変数は、POU のローカル変数の前に初期化されます。

ライブラリーの GVL 用に設定可能な定数 (パラメーターリスト)

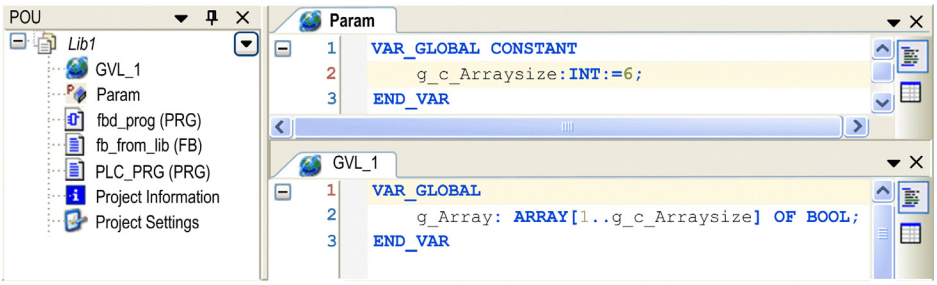
ライブラリーから提供されるグローバル定数の値は、アプリケーションによって定義された値で置き換えることができます。そのためには、定数をライブラリーのパラメーターリストで宣言してください。そうすることで、ライブラリーをアプリケーションに入れたときに、アプリケーションにある**ライブラリーマネージャー**の**パラメーターリスト**タブでその値を編集できます。詳細は、次の例を参照してください。

パラメーターリストの操作

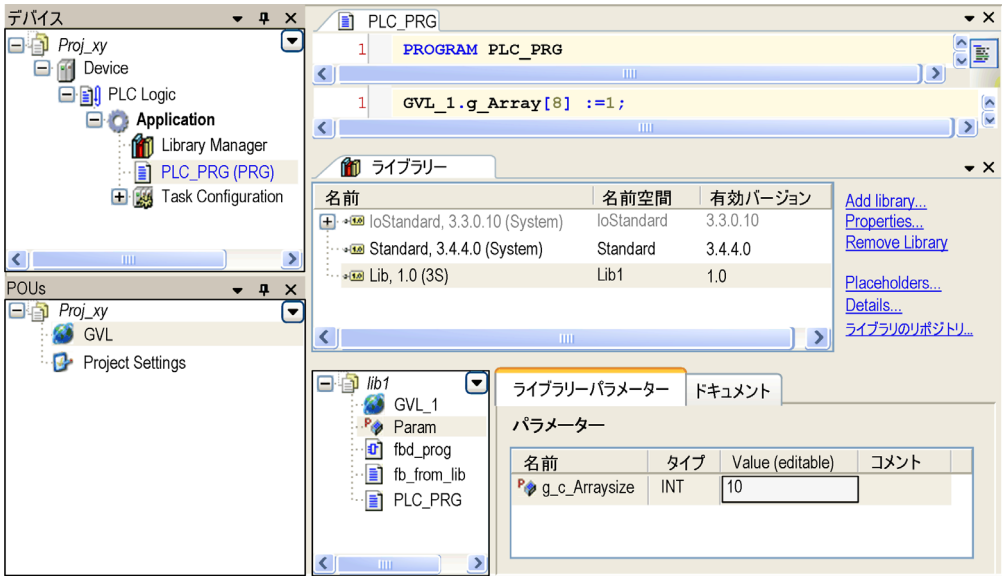
ライブラリー lib1.library は配列 g_Array を提供します。配列変数のサイズはグローバル定数 g_c_Arraysize によって定義されます。ライブラリーは複数のアプリケーションに含まれており、それぞれに異なる配列サイズが必要です。従って、ライブラリーのグローバル定数をアプリケーション固有の値で上書きしてください。

以下の手順に沿って進めます。lib1.library を作成するときに、特別なタイプのグローバル変数リスト (GVL)、いわゆるパラメーターリスト内でグローバル定数 g_c_Arraysize を定義します。そのためには、**オブジェクトの追加**コマンドを実行してパラメーターリストを追加します。現在の例では、Param という名前です。このオブジェクトのエディターで、標準の GVL と同じように変数 g_c_Arraysize の宣言を挿入します。

ライブラリー Lib1.library のパラメーターリスト Param



プロジェクトの**ライブラリーマネージャー**で、パラメーター g_c_Arraysize を編集します。



ライブラリーマネージャーの上部のライブラリーを選択して、モジュールツリーを取得します。Param を選択して、宣言を表示する**ライブラリーパラメーター**タブを開きます。**値 (編集可)** の列のセルを選択し、空白スペースを使用して編集フィールドを開きます。g_c_Arraysize の新しい値を入力します。編集フィールドを閉じると、ライブラリーの現在のローカルスコープに適用されます。

ネットワーク変数一覧 (受信側)

概要

ネットワーク変数一覧 (受信側) は、**ソフトウェアカタログ → 変数 → グローバル変数**ビューおよび**アプリケーションツリー**で使用されます。これにより、ネットワークの別のデバイスでネットワーク変数として指定する変数を定義します。

注記：ネットワーク変数の最大サイズは 255 バイトです。ネットワーク変数の数に制限はありません。

従って、他のネットワークデバイスの 1 つに特別なネットワークプロパティ (ネットワーク変数一覧) をもつネットワーク変数一覧 (送信側) が使用可能な場合、ネットワーク変数一覧 (受信側) オブジェクトをアプリケーションに追加できます。これは、同じプロジェクトで定義されているか、または異なるプロジェクトで定義されているかに無関係です。現在のネットワークに対して現在のプロジェクト内に適切なネットワーク変数一覧 (送信側) がいくつか見つかった場合は、ダイアログボックス**オブジェクトの追加 → ネットワーク変数一覧 (受信側)** でネットワーク変数一覧 (受信側) を追加するときに、**送信側**選択一覧から目的のネットワーク変数一覧 (送信側) を選択します。この章で説明するように、他のプロジェクトのネットワーク変数一覧 (送信側) はインポートする必要があります。

これは、各ネットワーク変数一覧 (受信側) が別のデバイスの 1 つのネットワーク変数一覧 (送信側) に対応していることを意味します。

ネットワーク変数リスト (受信側) の追加 ダイアログボックス

要素の説明

ネットワーク変数リスト (受信側) を追加するときは、**名前**の他に、ネットワーク変数の処理をする**タスク**も定義します。

または、別のデバイスからネットワーク変数リスト (送信側) を直接選択する代わりに、**ファイルからインポート**オプションを使用してネットワーク変数リスト (送信側) エクスポートファイル (*.GVL) を指定することもできます。このネットワーク変数一覧 (送信側) ファイルは、**ビュー → プロパティ → ファイルへのリンク** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* でネットワーク変数一覧 (送信側) から先に生成されたものです。いずれの場合も、必要なネットワーク変数一覧 (送信側) が別のプロジェクトで定義されているときに必要です。そのためには、**Sender** 選択リストの**ファイルからインポート**オプションを選択し、**ファイルからインポート**のテキストフィールドにファイルパスを入力します (または **...** ボタンをクリックして、ファイルシステムを参照するダイアログを開きます)。

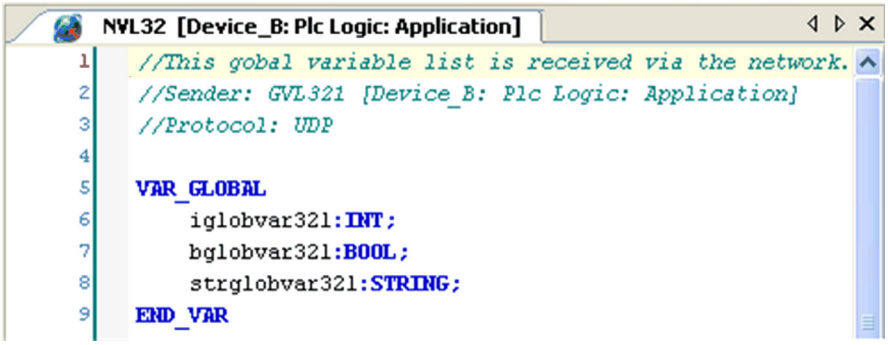
設定は後から、**表示 → プロパティ → Network Settings** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で変更できます。

ネットワーク変数一覧 (受信側) は NVL エディター (*354* ページ) に表示されますが、変更はできません。対応するネットワーク変数一覧 (送信側) の内容が表示されます。基本ネットワーク変数一覧 (送信側) を変更すると、ネットワーク変数一覧 (受信側) もそれに応じて更新されます。

ネットワーク変数一覧 (受信側) の宣言部の一番上にコメントが自動的に追加され、送信機 (デバイスパス)、ネットワーク変数一覧 (送信側) 名、およびプロトコルタイプに関する情報を提供します。

ネットワーク変数一覧の例

ネットワーク変数一覧



注記：リテラルまたは定数で範囲が定義されている配列だけが、リモートアプリケーションに転送されます。この場合、定数式は境界の定義には使用できません。
例 :
arrVar : ARRAY[0..g_iArraySize-1] OF INT ; は転送されません
arrVar : ARRAY[0..10] OF INT ; は転送されます
詳細は、*ネットワーク通信*の章 (*755* ページ) を参照してください。

単純なワーク変数交換の例

次の例では、単純なネットワーク変数交換が確立されています。送信側コントローラーでは、ネットワーク変数一覧 (送信側) が作成されます。受信側コントローラーでは、それに対応するネットワーク変数一覧 (受信側) が作成されます。

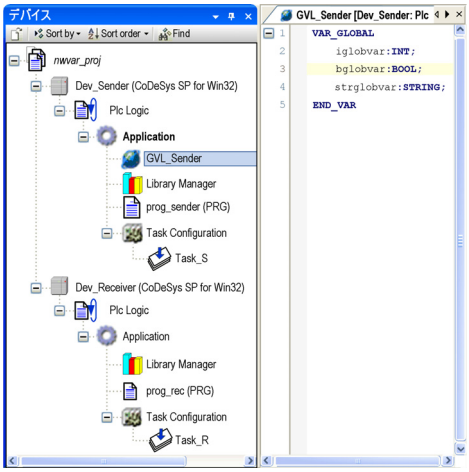
デバイスツリーに送信側コントローラー **Dev_Sender** および受信側コントローラー **Dev_Receiver** があるデフォルトプロジェクトで、以下の準備を行います。

- **Dev_Sender** の **アプリケーション**ノードの下に POU (プログラム) **prog_sender** を作成します。
- このアプリケーションの**タスク設定**ノードの下に、**prog_sender** を呼び出すタスク **Task_S** を追加します。
- **Dev_Receiver** の**アプリケーション**ノードの下に POU (プログラム) **prog_rec** を作成します。
- このアプリケーションの**タスク設定**ノードの下に、**prog_rec** を呼び出すタスク **Task_R** 追加します。
  注記：2 つのコントローラーは、Ethernet ネットワークの同じサブネットで設定してください。

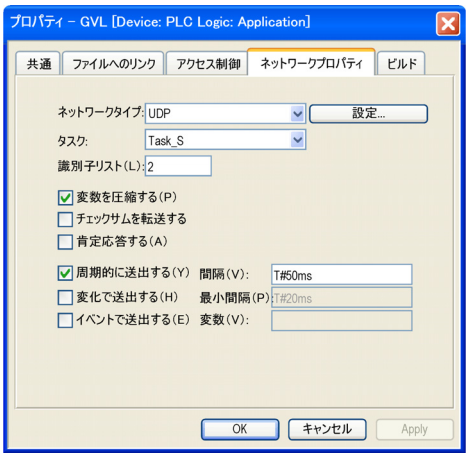
ネットワーク変数一覧 (送信側) の定義

手順 1: 送信側コントローラーでグローバル変数リストを定義します。

手順	手順内容	コメント
1	**ソフトウェアカタログ → アセット → POU** で、コントローラー **Dev_Sender** の**アプリケーション**ノードを選択して緑色のプラスボタンをクリックします。**ネットワーク変数一覧 (送信側)** コマンドを実行します。	ネットワーク変数一覧 (送信側) の**プロパティ**ダイアログボックスが表示されます。
2	**名前** GVL_Sender を入力し、**追加**をクリックして新しいグローバル変数リストを作成します。	**GVL_Sender** ノードが**アプリケーションツリー**の**アプリケーション**ノードの下に表示され、EcoStruxure Machine Expert 画面の中央にエディターが開きます。

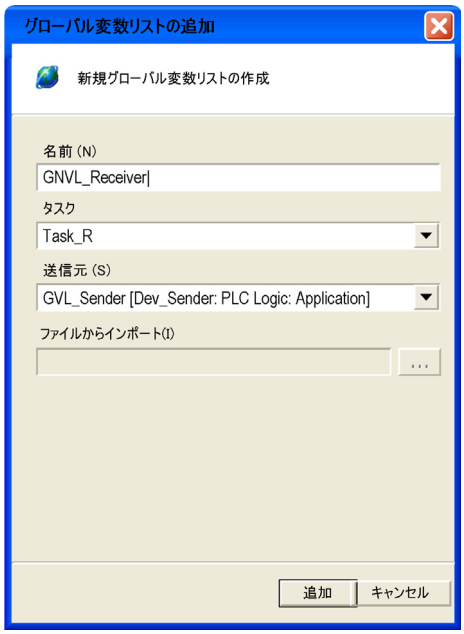
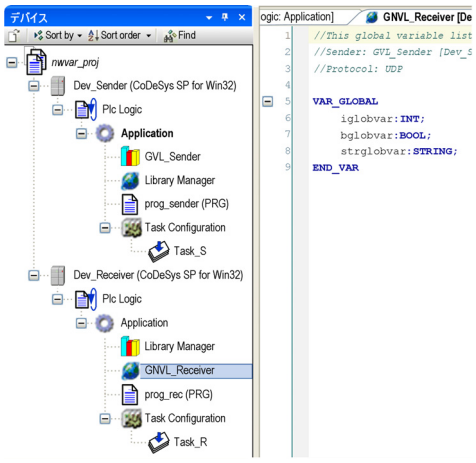
手順	手順内容	コメント
3	エディターに以下の変数定義を入力します。 VAR_GLOBAL iglobvar:INT; bglobvar:BOOL; strglobvar:STRING; END_VAR 	–

手順 2: ネットワーク変数一覧 (送信側) のネットワークプロパティを定義します。

手順	手順内容	コメント
1	**アプリケーションツリー**の **GVL_Sender** ノードを選択し、緑色のプラスボタンをクリックして、**プロパティ ...** コマンドを実行します。	**プロパティ - GVL_Sender** ダイアログボックスが表示されます。
2	**ネットワークプロパティ**タブを開き、図のようにパラメーターを設定します。 	–
3	**OK** をクリックします。	ダイアログボックスが閉じ、ネットワーク変数一覧 (送信側) のネットワークプロパティが設定されます。

ネットワーク変数一覧 (受信側) の定義

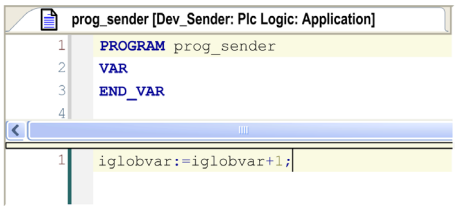
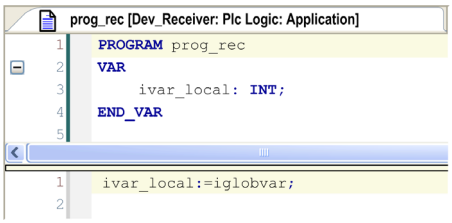
手順 1: 送信側コントローラーでグローバルネットワーク変数リストを定義します。

手順	手順内容	コメント
1	**アプリケーションツリー**で、コントローラー **Dev_Receiver** の**アプリケーション**ノードを選択し、緑色のプラスボタンをクリックして **グローバルネットワーク変数リスト ...** コマンドを実行します。	**グローバルネットワーク変数リストの追加**ダイアログボックスが表示されます。
2	図のようにパラメーターを設定します。 	このグローバルネットワーク変数リストは、送信側コントローラー用に定義されたネットワーク変数一覧 (送信側) と同じです。
3	**開く**をクリックします。	ダイアログボックスが閉じ、**Dev_Receiver** コントローラーの**アプリケーション**ノードの下に **GNVL_Receiver** が表示されます。  このネットワーク変数一覧 (受信側) には、自動的に **GVL_Sender** と同じ変数宣言が含まれます。

手順 2: ネットワーク変数一覧 (受信側) のネットワーク設定を表示、または変更します。

手順	手順内容	コメント
1	**デバイスツリー**で **GNVL_Receiver** ノードを右クリックして**プロパティ ...** コマンドを選択します。	**プロパティ - GNVL_Receiver** ダイアログボックスが表示されます。
2	**ネットワーク設定**タブが開きます。	–

手順 3: オンラインモードでネットワーク変数交換のテストをします。

手順	手順内容	コメント
1	コントローラー **Dev_Sender** の**アプリケーション**ノードの下にある POU **prog_sender** をダブルクリックします。	右側に **prog_sender** 用のエディターが開きます。
2	次のコードを変数 iglobvar に入力します。 	–
3	コントローラー **Dev_Receiver** の**アプリケーション**ノードの下にある POU **prog_rec** をダブルクリックします。	右側に **prog_rec** 用のエディターが開きます。
4	次のコードを変数 ivar_local に入力します。 	–
5	同じネットワーク内の送信側アプリケーションおよび受信側アプリケーションにログオンし、アプリケーションを起動します。	受信側の変数 ivar_local は、送信側に現在表示されている iglobvar の値を取得します。

保持変数

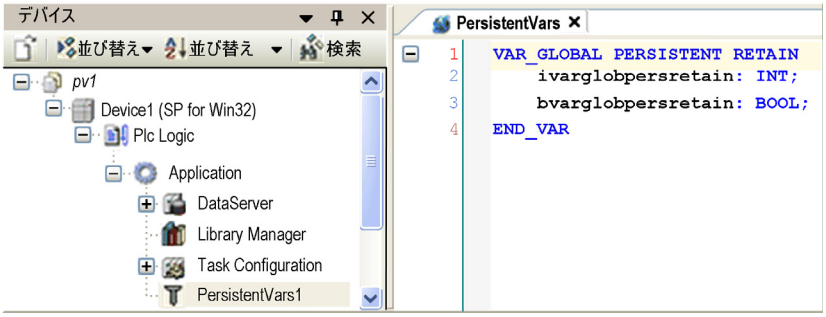
概要

このオブジェクトは、アプリケーションの保持変数のみを含むグローバル変数リスト (GVL) です。アプリケーションに割り当ててください。そのためには、各ノードを選択して緑色のプラスボタンをクリックし、**Add Other Objects → 保持変数 ...** を選択して、**アプリケーションツリー**に挿入してください。

VAR PERSISTENT で宣言され、このリストに含まれる変数のみが保持型です。**Add all Instance Paths** コマンドは *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 別の POU にあるすべての PERSISTENT 宣言をリストに追加します。

VAR PERSISTENT で宣言された変数も保持変数です。保持変数は、**ウォームリセット** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* が実行された後も値を保持することができます。違いは、保持変数は、**Reset origin** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* が実行されたとき、または新規アプリケーションがダウンロードされたとき (コントローラーからアプリケーションが削除された後) にのみ再初期化されます。名前またはデータ型の変更をした場合は例外です。

保持変数リスト



詳細については、保持変数 (*496* ページ) の説明を参照してください。

また、保持変数の処理 *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 用の特別なコマンドについての説明も参照してください。

GVL エディター (*351* ページ) に対応している保持エディターで保持変数リストの編集をします。最初の行に VAR_GLOBAL PERSISTENT RETAIN がプリセットされています。

残留変数の追加と宣言

アプリケーションに変数を追加するときに、変数の一部を残留変数として宣言できます。残留変数は、停電、再起動、リセット、およびアプリケーションプログラムのダウンロード中も値を保持します。残留変数には複数のタイプがあり、個別に残留型または保持型、またはそれを組み合わせた残留 - 保持型として宣言します。

異なるコントローラーの保持変数用のメモリーサイズに関しての情報は、各コントローラーについては、*プログラミングガイド*を参照してください。を参照してください。

保持変数と呼ばれるグローバル変数リストをアプリケーションに追加するには、以下の手順に沿って進めます。

手順	手順内容
1	**アプリケーションツリー**の各アプリケーションノードを選択して緑色のプラスボタンをクリックし、**Add Other Objects → 保持変数 ...** を選択します。 または、各アプリケーションノードを右クリックし、**オブジェクトの追加 → 保持変数 ...** コマンドを実行します。
2	**保持変数の追加** ダイアログボックスで、**名前**テキストボックスに名前を入力します。
3	**追加**をクリックします。 **結果 : アプリケーションツリー**に保持変数ノードが作成されます。例については、この章の*概要*の項を参照してください。

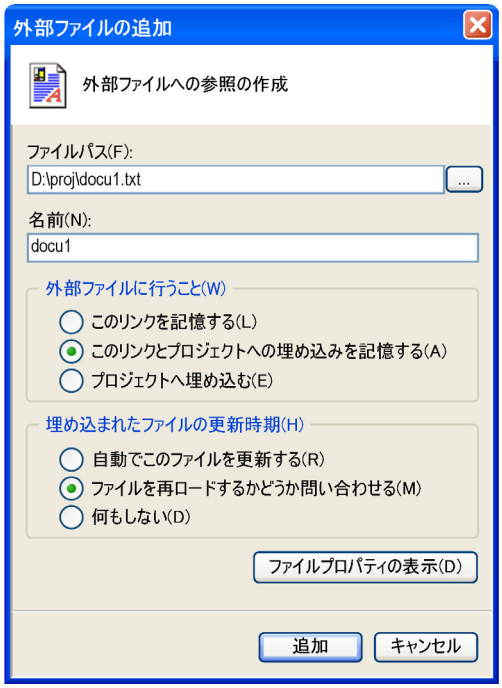
外部ファイル

概要

外部ファイルを**アプリケーションツリー**または**ツールツリー**の**グローバル**ノードに追加するには、**グローバル**ノードを選択して、緑色のプラスボタンをクリックして **Add other objects → External File...** コマンドを実行します。

... ボタンをクリックすると、ファイルを参照するためのダイアログボックスが開きます。**ファイルパス**テキストボックスにファイルのパスが入力されます。**名前**テキストボックスには、選択したファイルの名前が拡張子なしで自動的に入力されます。このフィールドを編集して、プロジェクト内で処理するファイルに別の名前を定義できます。

外部ファイルの追加ダイアログボックス



ダイアログボックスの「外部ファイルに行うこと」の説明

以下のオプションから 1 つ選択します。

オプション	詳細
このリンクを記憶する	定義されたリンクパスでこのファイルが使用できる場合にのみ、プロジェクトで使用できます。
このリンクとプロジェクトへの埋め込みを記憶する	ファイルのコピーがプロジェクトの内部に保存されますが、外部ファイルへのリンクも呼び出されます。定義されたように外部ファイルを使用できる限り、それに応じて定義された更新オプションが実装されます。そうでない場合は、プロジェクトに保存されているバージョンのファイルのみ使用できます。
プロジェクトへ埋め込む	ファイルのコピーのみがプロジェクトに保存されます。それ以上の外部ファイルへの接続はありません。

ダイアログボックスの「埋め込まれたファイルの更新時期」の説明

外部ファイルがプロジェクトにリンクされている場合は、さらにオプションを 1 つ選択できます。

オプション	詳細
自動でこのファイルを更新する	ファイルが外部で変更されるとすぐにプロジェクト内のファイルが更新されます。
ファイルを再ロードするかどうか問い合わせる	ファイルが外部で変更されるとすぐにダイアログボックスが表示されます。プロジェクト内のファイルも更新するかを決定できます。
何もしない	ファイルが外部で変更されてもプロジェクト内のファイルは変更されません。

ボタンの詳細

ボタン	詳細
ファイルプロパティの表示	このボタンでファイルのプロパティのダイアログボックスを開きます。このダイアログボックスは、**アプリケーションツリー**または**ツールツリー**のファイルオブジェクトを選択し、**プロパティ**コマンドを実行しても表示されます。このダイアログボックスの**外部ファイル**タブで、プロパティを表示および変更できます。
追加	設定の完了後に**追加**ボタンをクリックして、**アプリケーションツリー**または**ツールツリー**の**グローバル**ノードにファイルを追加します。指定されたファイル形式用にデフォルトとして定義されているツールで開かれます。

テキストリスト

概要

テキストリストは、**アプリケーションツリー**の**グローバル**ノードでグローバルに管理されるオブジェクト、または**アプリケーションツリー**のアプリケーションに割り当てられたオブジェクトです。

次の目的に役立ちます。
- 静的 (*182* ページ) および動的 (*183* ページ) テキスト、およびビジュアライゼーションおよびアラーム処理のツールチップにおける多言語対応
- 動的テキストの交換

テキストリストは、エクスポートおよび (再) インポート (*185* ページ) できます。表示器のビジュアライゼーションに XML 形式の言語ファイルが必要な場合にはエクスポートが必要ですが、翻訳 (*185* ページ) にも役立ちます。

テキストリストに使用可能なフォーマット
- テキスト
- XML

Unicode に対応 (*185* ページ) を有効にできます。

各テキストリストは、その名前空間によって固有に定義されます。これは、識別子 (ID、任意の文字シーケンスで構成される) および言語識別子によってリスト内で一意に参照されるテキスト文字列を含みます。使用するテキストリストは、ビジュアライゼーション要素のテキストを設定するときに指定します。

ビジュアライゼーションで設定した言語に応じて、対応するテキスト文字列がオンラインモードで表示されます。ビジュアライゼーションで使用する言語は、**言語の変更**入力によって変更されます。これは、指定したビジュアライゼーション要素に設定したマウス操作によって実行されます。各テキストリストには少なくともデフォルトの言語が含まれ、オプションで他の言語も定義できます。EcoStruxure Machine Expert に現在設定されている言語に一致する言語が見つからない場合、テキストリストのデフォルト言語が使用されます。各テキストには書式設定の定義 (*185* ページ) をもたせることができます。

テキストリストの基本構造

識別子 (インデックス)	**デフォルト**	**< 言語 1>**	**< 言語 2>**	**....< 言語 n>**
< 固有な文字列 >	< デフォルト言語のテキスト abc>	< 言語 1 のテキスト abc>	< 言語 2 のテキスト abc>	...
< 固有な文字列 >	< デフォルト言語のテキスト xyz>	< 言語 1 のテキスト xyz>	< 言語 2 のテキスト xyz>	...

テキストリストのタイプ

ビジュアライゼーション要素に使用できるテキストには 2 種類あり、それに対応して 2 種類のリストがあります。
- 静的テキスト用 GlobalTextList
- 動的テキスト用テキストリスト

静的テキスト用 GlobalTextList

GlobalTextList は、特定のテキスト項目の識別子が暗黙的に処理され、編集できない特別なテキストリストです。リストをエクスポートし、外部で編集してから再度インポートすることはできます。

ビジュアライゼーションの静的テキストは、動的テキストとは異なり、オンラインモードで変数による交換はされません。ビジュアライゼーション要素の言語を交換する唯一のオプションは、**言語の変更**入力によるものです。静的テキストは、**テキスト**カテゴリーの**テキスト**プロパティまたは**ツールチップ**プロパティによってビジュアライゼーション要素に割り当てられます。初めて静的テキストをプロジェクトで定義すると、**GlobalTextList** という名前のテキストリストオブジェクトが**アプリケーションツリー**の**グローバル**ノードに追加されます。そこには、**デフォルト**列に定義されたテキスト文字列と、テキスト識別子として自動的に割り当てられた整数の番号が含まれます。その後に作成される静的テキストごとに識別子番号がインクリメントされ、ビジュアライゼーション要素に割り当てられます。

ビジュアライゼーション要素に静的テキストが入力されている場合 (例えば、**テキスト**のプロパティカテゴリーの長方形に、文字列**テキスト例**が指定されている場合)、このテキストは **GlobalTextList** で検索されます。

- テキストが見つかった場合 (例えば、**ID 4711**、**テキスト例**)、**TextId** の要素値 **4711** が内部変数に割り当てられます。これにより、要素と **GlobalTextList** の対応する行の間の関係が確立されます。
- テキストが見つからない場合は、**GlobalTextList** に新しい行が挿入されます (例えば、**ID 4712**、**テキスト例**)。要素では、値 **4712** が内部変数に割り当てられます。

注記：存在しない場合は、**グローバルテキストリストの作成**コマンドでグローバルテキストリストを明示的に作成できます。

GlobalTextList をエクスポート、編集、および再インポートした場合、識別子が各ビジュアライゼーション要素の設定で使用されている識別子とまだ一致しているかを検証されます。必要に応じて、設定で使用される識別子の暗黙的な更新が実行されます。

視覚化用に定義したテキストの ID 番号を変更するには、ノードを右クリックし**削除**コマンドを実行して **GlobalTextList** を削除します。視覚化を開き **視覚化 → Create Global Text List** コマンドを実行します。**GlobalTextList** ノードがプロジェクトで使用可能な視覚化の静的テキストと一緒に**アプリケーションツリー**に作成されます。

注記：**GlobalTextList** に翻訳されさた文字列がある場合は、**Create Global Text List** コマンドが実行されたとき、それらは再生成されません。

GlobalTextList の例

グローバルテキストリストの作成

GlobalTextList

ID	Standard	Deutsch	English
5	%s	%s	%s
3	Deutsch	De Deutsch	German
4	Deutsch Tooltip	De Deutsch Tooltip	En German Tooltip
1	Englisch		
2	Englisch Tooltip		
0	Inkrement		

動的テキスト用テキストリスト

動的テキストは、オンラインモードで動的に変更できます。文字列であるテキストインデックス (**ID**) は、テキストリスト内で固有にしてください。**GlobalTextLists** とは異なり、動的テキストは定義する必要があります。また、**GlobalTextList** とは異なり、**グローバル**ノードを選択し、緑色のプラスボタンをクリックして **Add Other Objects → テキストリスト ...** コマンドを実行して動的テキスト用のテキストリストを明示的に作成します。

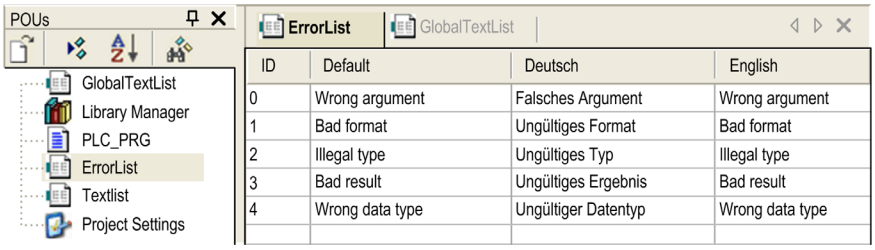
動的テキスト / **テキストリスト**プロパティによってビジュアライゼーション要素を設定するときに、使用できる動的テキストリストが提供されます。テキストインデックス (**ID**) と組み合わせたテキストリスト名を (直接入力するか、ID 文字列を定義するプロジェクト変数を入力して) 指定すると、テキストをオンラインモードで変更できます。

表示器のビジュアライゼーションの言語切り替え用の言語ファイルとして必要な場合は、動的テキストリストをエクスポートしてください。**ビジュアライゼーションオプション**でファイルパスを指定します。**GlobalTextList** などの動的テキストリストは、外部で編集するためにエクスポートしたり再インポートすることもできます。**GlobalTextList** とは異なり、動的テキストリストをインポートしたときに、識別子の自動チェックおよび更新はありません。

注記
識別子の意図しない変更 エクスポートしたリストを編集するときに識別子を変更しないでください。 **上記の指示に従わないと、物的損害を負う可能性があります。**

ErrorList という名前の動的テキストリストの例

ErrorList の例



ID	Default	Deutsch	English
0	Wrong argument	Falsches Argument	Wrong argument
1	Bad format	Ungültiges Format	Bad format
2	Illegal type	Ungültiges Typ	Illegal type
3	Bad result	Ungültiges Ergebnis	Bad result
4	Wrong data type	Ungültiger Datentyp	Wrong data type

例の詳細

この例では、整数の変数 ivar_err に割り当てられた数値 ID で識別されるエラーイベントを処理するアプリケーションで、エラーが検出された場合に対応するメッセージを表示するビジュアライゼーション要素を設定する方法を説明します。

エラー ID 0 ~ 4 のメッセージテキストが、**ドイツ語**、**英語**、および**デフォルト**言語で定義されている **ErrorList** という名前の動的テキストリストがあります。

ErrorList

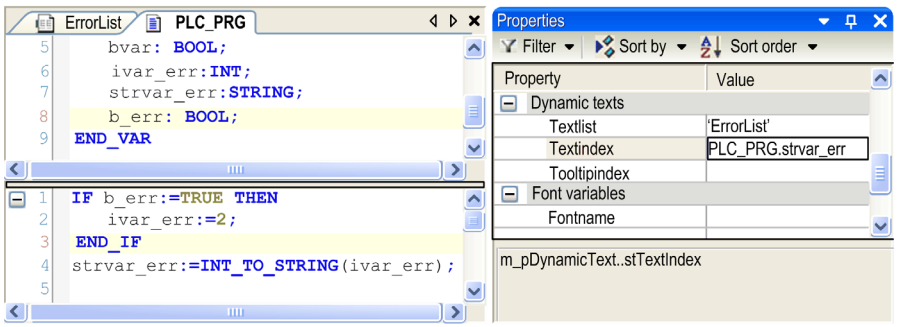
ID	デフォルト	Deutsch	English
0	This is Error 0. Do the following...	Fehler 0. Führen Sie folge...	Error 0. Do the...
1	This is Error 1. Close...	Fehler 1. Schließen Sie...	Error 1. Close the...
2	This is Error 2. Perform a...	Fehler 2. Führen Sie einen...	Error 2. Perform...
3	This is Error 3. Try to...	Fehler 3. Versuchen Sie...	
4	This is Error 4. Start...	Fehler 4. Starten Sie...	

表のセル内で改行を挿入するには、キーボードのショートカットキー **Ctrl** + **Enter** を押します。

ビジュアライゼーション設定でエラー ID を使用するには、STRING 変数、例えば、strvar_err を定義します。ivar_err の整数値を strvar_err に割り当てるには、strvar_err:=INT_TO_STRING(ivar_err); を使用します。

strvar_err は、ビジュアライゼーション要素の **Dynamic texts** プロパティの設定に **Textindex** パラメーターとして入力できます。この要素は、オンラインモードで適切なメッセージを表示します。

次の例は、プロジェクト変数およびビジュアライゼーション要素 (**プロパティ**) の設定を使用したエラー ID の処理用で、適切なメッセージを表示します。



テキストリストの作成

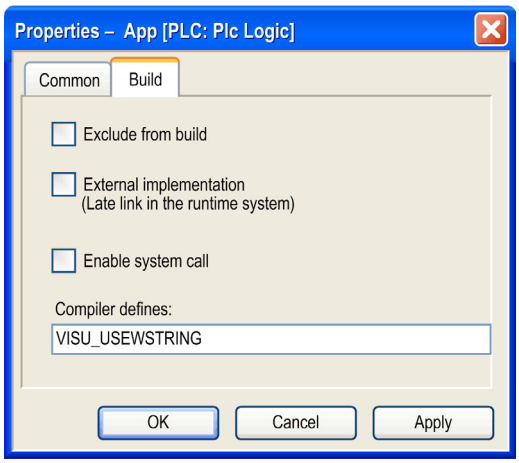
- 動的テキスト用のテキストリスト (*183* ページ) を作成するには、**アプリケーションツリー**のプロジェクトに**テキストリスト**オブジェクトを追加します。アプリケーション固有のテキストリストを作成するには、アプリケーションノードを選択します。グローバルテキストリストを作成するには、**グローバル**ノードを選択します。次に、選択したノードの緑色のプラスボタンをクリックし、**Add Other Objects → テキストリスト ...** コマンドを実行します。リスト名を指定して **Add Textlist** ダイアログボックスを確認すると、選択したノードの下に新しいリストが挿入され、テキストリストエディタービューが開きます。
- 静的テキスト (*182* ページ) (**GlobalTextList**) のテキストリストを取得するには、ビジュアライゼーションオブジェクトの**テキスト**カテゴリーにある**テキスト**プロパティのテキストを割り当てて自動的に作成されたリストを取得するか、**Create Global Text List** コマンドで明示的に生成します。

- 編集のために既存のテキストリストを開くには、**アプリケーションツリー**または**アプリケーションツリーのグローバル**ノードにあるリストオブジェクトを選択します。テキストリストノードを右クリックし、**Edit Object** コマンドを実行するか、テキストリストノードをダブルクリックします。テキストリストの構成については、*テキストリストの基本構成*の表を参照してください。
- テキストリストに新しいデフォルトのテキストを追加するには、**Insert Text** コマンドを使用するか、リストの空の行にある各フィールドを編集します。テキストリストのフィールドを編集するには、そのフィールドをクリックして選択してから再度フィールドをクリックするか、SPACE を押して編集枠を取得します。必要な文字を入力し、編集枠を RETURN で閉じます。

Unicode 形式に対応

Unicode 形式を使用するには、**ビジュアライゼーションマネージャー**で各オプションを有効にします。さらに、アプリケーションに特別なコンパイル指令を設定します。**デバイスツリー**でアプリケーションを選択し、**プロパティ**ダイアログボックスの**ビルド**タブを開きます。**Compiler defines** フィールドで、VISU_USEWSTRING を入力します。

ダイアログボックスのコンパイラー定義



テキストリストのエクスポートとインポート

静的および動的テキストリストは、CSV 形式のファイルとしてエクスポートできます。エクスポートされたファイルは、例えば、外部の翻訳者などによって、外部でテキストを追加するためにも使用できます。ただし、テキスト形式 (*.csv) のファイルのみ再インポートできます。

各テキストリストコマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。

ファイル → プロジェクト設定 → ビジュアライゼーションダイアログボックスで、エクスポートファイルを保存するフォルダーを指定します。

テキストの書式設定

テキストに、書式設定の定義 (%s, %d、...) を含めることができるため、テキストの変数の現在値も含めることができます。書式設定文字列については、EcoStruxure Machine Expert オンラインヘルプの *ビジュアライゼーション*の部分を参照してください

書式設定文字列を使ったテキストを使用する場合は、次の順序で置換します。

- 使用する実際のテキスト文字列をリスト名および ID で検索します。
- テキストに書式定義が含まれている場合、それらを各変数の値で置き換えます。

翻訳されたテキストの統合

テキストファイルの読み込みに使用するディレクトリーに *GlobalTextList.csv* を挿入することにより、後で翻訳されたテキストを統合できます。起動プロジェクトが開始するときに、ファームウェアによって追加のファイルを使用できることが検出されます。テキストは、既存のテキストリストファイルのテキストと比較されます。その後、新しいテキストと変更されたテキストがテキストリストファイルに適用されます。更新されたテキストファイルは、次の起動時に適用されます。

テキスト入力のコンポーネントリスト

ツール → オプション → ビジュアライゼーションダイアログボックスで、テキストテンプレートファイルを指定できます。このファイルの**デフォルト**列のテキストはすべて、**List Components** 機能に使用されるリストにコピーされます。以前に**エクスポート**コマンドで作成したテンプレートファイルを使用できます。

複数ユーザーによる操作

ソース制御を使用することによって、複数のユーザーが同時に同じプロジェクトで作業できます。複数のユーザーによってビジュアライゼーション要素の静的テキストが変更された場合、**GlobalTextList** (**GlobalTextList** (*182* ページ) を参照) が変更されます。この場合、Text-Id はビジュアライゼーション要素との一貫性がなくなる可能性があります。次のエラー検出および修正方法を使用してください。

- **Check Visualization Text Ids** コマンドを使用することで、ビジュアライゼーションでそのようなエラーが検出される場合があります。
- **Update Visualization Text Ids** コマンドを使用することで、このようなエラーが自動的に解決される場合があります。影響を受けるビジュアライゼーションおよび **GlobalTextList** には書き込み権限が必要です。

ビジュアライゼーションの言語変更のためのテキストリストの使用

適切なテキストリストが使用できる場合、すなわち複数の言語バージョンのテキストが定義されたテキストリストがある場合、ビジュアライゼーション要素の入力によってオンラインモードでビジュアライゼーションのテキストに使用する言語を切り替えできます。要素の **Dynamic Texts** プロパティに使用するテキストリストを指定します。また、**OnMouse..** 入力のアクション、**Change the language** には、マウス操作が実行された後に使用する言語を設定をします。

注記：言語は、各テキストリストのヘッダー列に表示されている文字列と完全に同じように指定します。

イメージプール

概要

イメージプールは、各イメージのファイルパス、プレビュー、および文字列 ID を定義するテーブルです。ID およびイメージファイル名 (一意のアクセス用) を指定することで、イメージを参照できます。例えば、ビジュアライゼーションにイメージを挿入する場合など (イメージ要素のプロパティの設定については、*イメージプールで管理されているイメージの使用* (*189* ページ) を参照してください)。

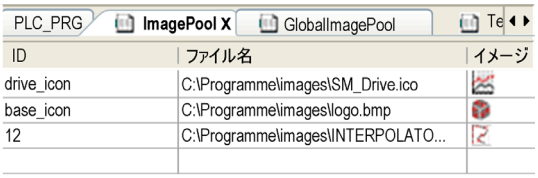
ライブラリープロジェクトでイメージプールを作成できます。その後そのライブラリーをシンボルライブラリーと宣言した場合は、プロジェクトの視覚化内に含まれているイメージを使用できます。これを実現するには、ライブラリープロジェクトの **ImagePool** ノードを右クリックし、**プロパティ**を選択します。**Symbol library settings** を**イメージプール**タブで (**Mark library as symbol library** ボタンをクリックし、**Text list for symbol translation** をオプションで選択することによって) 選択します。

ライブラリーをプロジェクトに追加すると、視覚化エディターがアクティブな場合にイメージプールが (つけた名前で) **ToolBox** に表示されます。

注記：イメージプールにイメージファイルを追加する前に、イメージファイルのサイズを出来る限り小さくします。そうでない場合、イメージを含むビジュアライゼーションアプリケーションのプロジェクトサイズおよび読み込みや格納の負荷が大きくなります。

イメージプールの構造

イメージプールの例



要素	詳細
ID	文字列 ID (例、**logo**、**y_icon**、**2**); イメージリスト名と ID の組み合わせ (例、**List1.basic_logo**) によりイメージの一意参照が可能です。
ファイル名	イメージファイルのパス (例、*C:\programs\images\logo.bmp*) EcoStruxure Machine Expert でサポートされているイメージフォーマット ● BMP ● EMF ● GIF ● ICO ● JPG ● PNG ● SVG ● TIFF ご使用のコントローラーはすべてのイメージフォーマットをサポートしていない場合もあります。詳細は 各コントローラーについては、*プログラミングガイド*を参照してください。を参照してください。 イメージファイルがイメージファイル用のディレクトリー (**ツール → オプション → 視覚化**で定義) に保存される場合、このテキストボックスにはファイル名のみを入力します。
イメージ	イメージのプレビュー
Link type	イメージファイルがプロジェクトにどのようにリンクしているかの情報。**Select image** ダイアログボックスでイメージファイルを手動で追加するときに、**Link type** を指定します。*イメージプールの作成と編集* の説明を参照してください (*188* ページ)。

注記：ターゲットシステムがベクターイメージフォーマット SVG のイメージをサポートしていない場合は、ダウンロード中に自動的に PNG フォーマットに変換されます。イメージフォーマットのサポートについての詳細は、ご使用のハードウェア製造元のデバイス説明を参照してください。

イメージプールの作成と編集

プロジェクトには複数のイメージプールを含めることができます。自動的に作成された **GlobalImagePool**、および手動で作成されたイメージプール。

GlobalImagePool

プロジェクトのイメージプールにまだ存在しないイメージを視覚化に追加します。その際、要素プロパティにそのイメージの静的 ID を入力します。これにより、それぞれのイメージファイルのエントリーを含む **GlobalImagePool** が自動的に作成されます。**Link type** は **Link to file** です。

手動で空のイメージプールを作成する

イメージプールオブジェクトは、**アプリケーションツリー**の**グローバル**ノードで、緑色のプラスボタンをクリックして **他のオブジェクトを追加 → イメージプール ...** コマンドを実行することで挿入できます。**Add Image Pool** ダイアログボックスで、プールの**名前**を定義します。

イメージファイルをイメージプールに追加する

イメージファイルをイメージプールに追加する	実行手順
Insert Image コマンドを実行する	1. イメージプールエディターにフォーカスをおきます。 2. コンテキストメニューから **Insert Image** コマンドを実行します *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。 **結果** : 編集可能な一意の ID が入力されます。 3. 新しい行の**ファイル名**フィールドをダブルクリックしてイメージファイルのパスを指定します。 4. この目的で  ボタンをクリックして **Select Image** ダイアログボックスを開くことができます。このダイアログボックスで編集するフィールド、およびオプションはこの表の下で説明します。 **注記：Select Image** ダイアログボックスを使用しない場合は、イメージファイルをパスを直接入力します。リンクタイプ設定は **このリンクを記憶する**が使用されます。
ファイル名を直接入力する	イメージプールのエディターで、最初の空の行の**ファイル名**フィールドをダブルクリックします。使用するイメージファイル - 上記の最初のオプションで説明されているように (**Insert Image** コマンドを実行) - のパスを入力します。 **結果** : ファイル名は自動的に ID として入力されます。
ファイルシステムからドラッグ & ドロップ	ローカルのファイルシステムブラウザーで、使用するイメージファイルを選択しイメージプールエディターにドラッグします。複数が選択可能です。 **結果** : ファイル名は自動的に ID として入力されます。リンクタイプ設定の**このリンクを記憶する**が自動的に使用されます。

イメージの選択ダイアログボックスの要素

要素	詳細
イメージファイル	イメージファイルのパスを入力するか、  ボタンをクリックして、ローカルファイルシステムを閲覧するスタンダードダイアログボックスを開きます。ファイルを選択します。複数が選択可能です。

要素	詳細
What do you want to do with the image file?	リンクタイプを選択します。 ● **このリンクを記憶する** : 指定したパスにファイルがある場合のみ、プロジェクトで使用できます。パスなしで指定されているファイルはプロジェクトフォルダーに保存されている必要があります。 ● **このリンクとプロジェクトへの埋め込みを記憶する** : ファイルのコピーがプロジェクト内部に保存されます。指定されたパスのリンクも保存されます。イメージファイルが保存されたパスにある場合、以下のように定義された更新動作は有効です。イメージファイルが指定パスから削除されると、プロジェクト内部に保存されたコピーのみ使用されます。 ● **プロジェクトへ埋め込む** : ファイルのコピーがプロジェクト内部のみに保存されます。指定された外部パスのリンクは保存されません。 **このリンクとプロジェクトへの埋め込みを記憶する**オプションを選択すると、以下のいずれかの更新動作を選択できます。 **イメージファイルが変更された場合** ● **自動でこのファイルを更新する** ● **ファイルを再ロードするかどうか問い合わせる** ● **何もしない**

イメージプールで管理されているイメージの使用

使用するイメージの ID が複数のイメージプールで指定されている場合。

- 検索順序 : **GlobalImagePool** で管理されているイメージを選択した場合は、プール名を指定する必要はありません。イメージの検索順序は、グローバル変数の検索順序に対応します。
  1. **GlobalImagePool**
  2. 現在有効なアプリケーションに割り当てられたイメージプール
  3. **アプリケーションツリーのグローバル**ノードにある **GlobalImagePool** 以外のイメージプール
  4. ライブラリーのイメージプール
- 一意のアクセス : ID の前にイメージプール名を追加することで、目的のイメージを直接呼び出すことができます。< プール名 >.< イメージ ID> (例えば、前の図の imagepool1.drive_icon を参照してください。)

イメージタイプのビジュアライゼーション要素でのイメージの使用

ビジュアライゼーションにイメージ要素を挿入するときに、静的イメージまたは動的イメージとして定義できます。動的イメージは、オンラインモードでプロジェクト変数の値に応じて変更できます。

静的イメージ :

要素の設定 (**Static ID** プロパティ) で、イメージ ID またはイメージプール名 + イメージ ID を入力します。前の項の検索順序および一意のアクセスに関する事項を考慮してください。

動的イメージ :

要素の設定 (**Bitmap ID variable** プロパティ) で、ID を定義する変数 (例、PLC_PRG.imagevar) を入力します。

ビジュアライゼーションの背景にイメージを使用

ビジュアライゼーションの背景定義で、ビジュアライゼーションの背景として表示するイメージを定義できます。イメージファイルは、ビジュアライゼーション要素について前述したようにイメージプールおよびイメージファイル名で指定できます。

7.4
アプリケーション

アプリケーション

概要

アプリケーションとは、特定のハードウェアデバイス (コントローラー) でコントローラープログラムの特定のインスタンスを実行するために必要なオブジェクトのセットです。そのために、**アプリケーションツリー**の**グローバル**ノードで管理される独立したオブジェクトがインスタンス化され、デバイスに割り当てられます。これは、オブジェクト指向プログラミングの概念に適合しています。ただし、純粋にアプリケーション固有の POU を使用することもできます。

アプリケーションは、**アプリケーションツリー**のアプリケーションオブジェクトによって表されます。アプリケーションの項目の下に、アプリケーションリソースセットを定義するオブジェクトを挿入します。

1 つのコントローラーに 1 つのアプリケーションを使用できます。それ以上のアプリケーションは追加できません。

各アプリケーションの一部は、プログラム (POU インスタンスまたはアプリケーション固有の POU) の実行を制御する**タスク設定**です。さらに、グローバル変数リスト、ライブラリーなどのリソースオブジェクトを割り当てることができます。これらは、**アプリケーションツリー**の **グローバル**ノードで管理されているものとは異なり、特定のアプリケーションと子によってのみ使用されます。規則については、**デバイスツリー**のオブジェクトの配置と設定 (*35* ページ) を参照してください。

注意事項

対象のデバイス (コントローラーまたはシミュレーション表示器) のアプリケーションでログインすると、どのアプリケーションが現在コントローラーにあるか、またコントローラーのアプリケーションパラメーターが EcoStruxure Machine Expert のアプリケーションのパラメーターと一致しているか、の 2 つのチェックが実行されます。不一致の場合は該当するメッセージによって示され、続行するための方法が複数提示されます。また、コントローラーのアプリケーションを削除することもできます。詳細は、*ログイン*コマンド (*199* ページ) の説明を参照してください。

第 8 章
タスク設定

この章について

この章には次の項目が含まれています。

タスク設定

概要

タスク設定で、アプリケーションプログラムの処理を制御するためのタスクを定義します。

アプリケーション (*190* ページ) のリソースオブジェクトです。アプリケーションノードの下の**アプリケーションツリー**に挿入してください。タスクは、アプリケーションの下の**アプリケーションツリー**でのみ使用できるアプリケーション固有のプログラム POU を呼び出すことができます。また、**アプリケーションツリーのグローバル**ノードで管理されるプログラムを呼び出すこともできます。後者の場合、グローバルに使用可能なプログラムはアプリケーションによってインスタンス化されます。

タスク設定 エディターでタスク設定を編集できます (*373* ページ)。

オンラインモードでは、**タスク設定** エディターにサイクル、サイクルタイム、およびタスクステータスに関する情報を表示する監視ビューがあります。

デバイスで対応している場合は、タスク設定の追加機能として、監視ビューでタスクによって制御される POU の動的分析ができます。それにより、サイクルタイム、ファンクションブロック呼び出しの数、および使用されていないコード行に関する情報が提供されます。

タスクの追加

概要

アプリケーションツリーからタスクをアプリケーションに追加できます。

手順

手順	手順内容
1	**アプリケーションツリー**の**タスク設定**ノードを選択し、緑色のプラスボタンをクリックして、**タスク ...** コマンドを実行します。 または、**タスク設定**ノードを右クリックし、コンテキストメニューから**オブジェクトの追加 → タスク ...** を選択することもできます。 **結果** : **タスクの追加**ダイアログボックスが開きます。
2	**タスクの追加**ダイアログボックスで、**名前 :** テキストボックスに名前を入力します。 **注記 :** 名前はスペースを含めず、32 文字以下にしてください。
3	**追加**をクリックします。

第 9 章
アプリケーションの管理

この章について

この章には次のセクションが含まれています。

9.1
一般情報

概要

概要

アプリケーションを実行するには、初めにパソコンをコントローラーに接続してから、コントローラーにアプリケーションをダウンロードしてください。

注記：メモリーサイズに制限があるため、コントローラーによっては実行用のビルドされたアプリケーションのみが格納され、アプリケーションソースは格納されません。これは、コントローラーからパソコンにはアプリケーションソースをアップロードできないことを意味します。

⚠ 警告

装置の意図しない動作

- アプリケーションをダウンロードするときは、**通信設定**ダイアログに正しいデバイスの名称またはデバイスアドレスが入力されていることを確認してください。
- 予期しない機械の操作による人的傷害や機器の破損を避けるため、機械のガードおよびタブが適切に配置されていることを確認してください。
- ソフトウェアおよび関連機器のすべてのユーザーマニュアル、および機器または機械操作に関するドキュメントを読み理解してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

前提条件

アプリケーションをコントローラーにダウンロードする前に、次の条件を満たしていることを確認してください。

- 正しいコントローラーの有効なパスが設定されている。
- ダウンロードするアプリケーションが有効である。
- アプリケーションにコンパイルエラーがない。

起動アプリケーション

起動アプリケーションは、コントローラーの開始時に起動されるアプリケーションです。このアプリケーションはコントローラーのメモリーに格納されています。起動アプリケーションのダウンロードを設定するには、**アプリケーションツリー**の**アプリケーション**ノードを右クリックして**プロパティ**コマンドを選択します。

新しいアプリケーションのダウンロードが正常に終了すると、起動アプリケーションを作成するかを尋ねるメッセージが表示されます。

次の方法により、手動で起動アプリケーションを作成できます。

- オフラインモード：**オンライン → 起動アプリケーションの作成**をクリックして、起動アプリケーションをファイルに保存します。
- オンラインモードで、コントローラーが STOP モードになっている場合：**オンライン → 起動アプリケーションの作成** コマンドを実行して *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*、起動アプリケーションをコントローラーにダウンロードします。

9.2
アプリケーションのビルドとダウンロード

このセクションについて

このセクションには次の項目が含まれています。

アプリケーションのビルド

概要

EcoStruxure Machine Expert では、ビルドメニュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で異なるビルド方式を提供しています。これらの方式では変更されたオブジェクトのみか、または有効なアプリケーションのすべてのオブジェクトの構文チェックを処理します。

コードをデバイスにダウンロードする前に、コンパイルエラーをチェックするオフラインコードの生成を実行できます。正常にログインするには、エラーが検出されることなくコード生成を完了している必要があります。

コード生成、コンパイル情報

以下の場合にマシンコードが生成されます。

- アプリケーション (*190* ページ) プロジェクトが対象デバイス (コントローラー、シミュレーション表示機) にダウンロードされたとき。
- **ビルド → Generate code** コマンドが実行されたとき。

各ダウンロード時に、コードおよび読み込まれたアプリケーションのリファレンス ID を含むコンパイル情報が、プロジェクトディレクトリーのファイル *< プロジェクト名 >.< デバイス名 >.< アプリケーション ID>.compileinfo* に保存されます。*compileinfo* ファイルは、**クリーン**または**すべてクリーン**コマンドを実行すると削除されます。

ビルドコマンド (初期設定では**ビルド**メニュー) でプロジェクトをコンパイルすると、コード生成は実行されません。ビルド処理では、プログラミングエラーを検出するためにプロジェクトをチェックします。検出されたプログラミングエラーは、**メッセージ**ビュー (メッセージカテゴリー**ビルド**) に表示されます。

コード生成中に、さらにエラーが検出、表示される場合があります。これらのエラーはコード生成器でのみ検出されるか、メモリー割り当てによって発生します。

コード生成時のメッセージ

コード生成ごとに、コードとデータサイズ (バイト単位)、割り当てられたメモリー領域の内容、および最大使用アドレス (バイト) に関する追加情報が**メッセージ → ビルド**ビューに表示されます。

さまざまな種類のデータとコードがどのメモリー領域に格納されるかはコントローラーによって異なります。変数がアドレスに割り当てられていない場合、アドレス %I、%M、%Q にメモリーが割り当てられます。アプリケーションの **Clean** 後には、メモリーは全て再割り当てされます。

ログイン

概要

オンライン → ログインコマンドは、アプリケーションを対象デバイス (コントローラーまたはシミュレーション表示器) に接続し、オンラインモードに変えます。

デフォルトのショートカットキーは **Alt** + **F8** です。

⚠ 警告

装置の意図しない動作

- アプリケーションをダウンロードするときは、通信設定ダイアログに正しいデバイス名称またはデバイスアドレスが入力されていることを確認してください。
- 予期しない機械の操作による人的傷害や機器の破損を避けるため、機械のガードおよびタブが適切に配置されていることを確認してください。
- ソフトウェアおよび関連機器のすべてのユーザーマニュアル、および機器または機械操作に関するドキュメントを読み理解してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

対象デバイスにオンラインユーザー管理 (*ユーザーとグループ* (*112* ページ) の章も参照) が確立されている場合、ログイン時に適切なユーザー名とパスワードを入力するよう求められます。そのために、**デバイスユーザーログオン**ダイアログボックスが開きます。

注記：オンラインモードに切り替えると、**オンライン** パースペクティブ (*41* ページ) が自動的に選択されるため、EcoStruxure Machine Expert のビューのレイアウトは対象デバイスに接続すると変わることがあります。

デバイスディターの通信設定タブのビューを選択する

ツール → オプション → デバイスエディター ダイアログボックスの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **Communication page** で選択したオプションに応じて、デバイスエディターの**通信設定**タブは 3 つのモードで表示されます。詳細は、異なるタブの説明を参照してください。

モード	Communication page 設定	デバイスエディターのダイアログ
1	**Controller selection mode** (デフォルト設定)	コントローラーの選択モードの**通信設定**タブ (*83* ページ)
2	**Simple mode**	Simple mode の**通信設定**タブ (*98* ページ)
3	**Classic mode**	Classic mode の**通信設定**タブ (*100* ページ)

ログイン手順

ログイン手順は同じです。**通信設定**タブの表示が異なるだけです。

正常にログインするには、エラーが検出されることなくコード生成を完了している必要があります (*ログイン前のビルド処理* (*201* ページ) の章を参照してくだい)。

手順	手順内容
1	**オンライン → ログイン**コマンドを実行するか、ツールバーの**ログイン**ボタン  をクリック、または ALT + F8 を押します。 **結果** : 表示器アドレスが設定されていないため、デバイスエディターの**通信設定**タブが選択したモードで開きます。有効なアドレスが定義されていないことを示すメッセージボックスが表示されます。
2	EcoStruxure Machine Expert で、コントローラーが 1 つのみ検知された場合、そのコントローラーが対象デバイスとして使用されます。 複数のコントローラーが検出された場合は、ログインするコントローラーをダブルクリックします。 **注記**：コントローラー選択モードおよびクラシックモードでは、選択したコントローラーと同じ **Target ID** をもつコントローラーだけが一覧表示されます。リストにすべてのコントローラーを表示するには、**フィルター**基準を**なし**に設定します。

手順	手順内容
3	**オンライン → ログイン**コマンドを実行するか、ツールバーの**ログイン**ボタン  をクリック、または ALT + F8 を押します。 **結果** : 潜在的な危険を知らせるメッセージボックスが表示されます。
4	**キャンセル**をクリックしてログイン操作を中止するか、ALT + F を押してメッセージを確認し選択したコントローラーにログインします。 **結果** : ALT + F を押すとコントローラーへの接続が確立され、アプリケーションをダウンロード (*202* ページ) できます。

ログインの失敗

条件	結果	コメント
コントローラーにログイン中エラーが検知された場合。	操作が中断しエラーメッセージが表示されます。	エラーメッセージで、エラーの詳細を表示することができます。
ログメッセージに **SOURCEPOSITION** というテキストを含む例外が発生した場合。	**Show source code in editor** コマンドを実行できます。	このコマンドは、エディターでファンクションを開きエラーが検出された位置にカーソルを置きます。

変更されたアプリケーションのビルド処理

ログイン前のビルド処理

ログインする前、および影響を受けた現在のアプリケーションプロジェクトが開かれてからまたは最後に変更されてからコンパイルされていない場合、コンパイルされます。つまり、プロジェクトはオフラインモードの**ビルド**実行に応じてビルドされ、コントローラーのコンパイルコードが生成されます。

コンパイル中にエラーが検出された場合は、次の文のメッセージボックスが開きます。**コンパイルエラーがあります。ダウンロードせずにログインしますか？**検出されたエラーを修正するか、そのままログインするかを選択できます。後者の場合、コントローラーのすでに使用可能なバージョンのアプリケーションにログインします。

検出されたエラーは、**メッセージ**ビュー (カテゴリー**ビルド**) にリスト表示されます。

アプリケーションのダウンロード

概要

アプリケーションを実行するには、初めにパソコンをコントローラーに接続してから、アプリケーションをコントローラーにダウンロードしてください。

プロジェクトをダウンロードすると、現在のプロジェクトが EcoStruxure Machine Expert からコントローラーメモリーにコピーされます。

注記：メモリーサイズに制限があるため、コントローラーによっては実行用のビルドされたアプリケーションのみが格納され、アプリケーションソースは格納されません。これは、コントローラーからパソコンにはアプリケーションソースをアップロードできないことを意味します。

⚠ 警告

装置の意図しない動作

- アプリケーションをダウンロードするときは、通信設定ダイアログに正しいデバイスの名称またはデバイスアドレスが入力されていることを確認してください。
- 予期しない機械の操作による人的傷害や機器の破損を避けるため、機械のガードおよびタブが適切に配置されていることを確認してください。
- ソフトウェアおよび関連機器のすべてのユーザーマニュアル、および機器または機械操作に関するドキュメントを読み理解してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

前提条件

アプリケーションをコントローラーにダウンロードする前に、次の条件を満たしていることを確認してください。

- 正しいコントローラーの有効なパスが設定されている。
- ダウンロードするアプリケーションが有効である。
- アプリケーションにコンパイルエラーがない。

起動アプリケーション

起動アプリケーションは、コントローラーの開始時に起動されるアプリケーションです。このアプリケーションはコントローラーのメモリーに格納されています。起動アプリケーションのダウンロードを設定するには、**デバイス**ビューの**アプリケーション**ノードを右クリックして**プロパティ**コマンドを選択します。

新しいアプリケーションのダウンロードが正常に終了すると、起動アプリケーションを作成するかを尋ねるメッセージが表示されます。

次の方法により、手動で起動アプリケーションを作成できます。

- オフラインモード : **オンライン → 起動アプリケーションの作成**をクリックして、起動アプリケーションをファイルに保存します。
- オンラインモードで、アプリケーションが STOP モードになっている場合 : **オンライン → 起動アプリケーションの作成** を実行して、起動アプリケーションをコントローラーにダウンロードします。

動作モード

ダウンロードの方法は、読み込まれたアプリケーションとダウンロードしたアプリケーションの関係によって異なります。3 種類のケースがあります。

- ケース 1: コントローラーのアプリケーションが、読み込むアプリケーションと同じ。この場合、ダウンロードはされません。EcoStruxure Machine Expert をコントローラーに接続するのみです。
- ケース 2: EcoStruxure Machine Expert のアプリケーションと比べて、コントローラーに読み込まれたアプリケーションは変更されている。この場合、変更されたアプリケーションのすべてまたは一部をダウンロードするように指定するか、またはアプリケーションをそのままコントローラーに維持するか指定できます。
- ケース 3: コントローラーに異なるバージョンまたは新しいバージョンのアプリケーションがあり使用可能。この場合、このアプリケーションを置き換えるかを尋ねられます。
- ケース 4: コントローラーにアプリケーションがまだない。この場合、ダウンロードの確認を求められます。

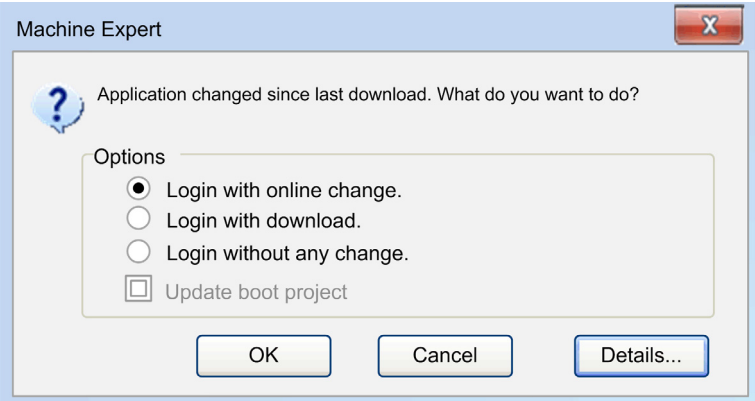
コントローラーにアプリケーションをダウンロード : ケース 1

コントローラーのアプリケーションが、読み込むアプリケーションと同じ。この場合、ダウンロードは EcoStruxure Machine Expert をコントローラーに接続するのみ。

手順	手順内容
1	コントローラーに接続するには、**オンライン → ' アプリケーション [アプリケーション名 ; Plc ロジック]' にログイン** を選択します。
2	コントローラーに接続されました。

コントローラーにアプリケーションをダウンロード : ケース 2

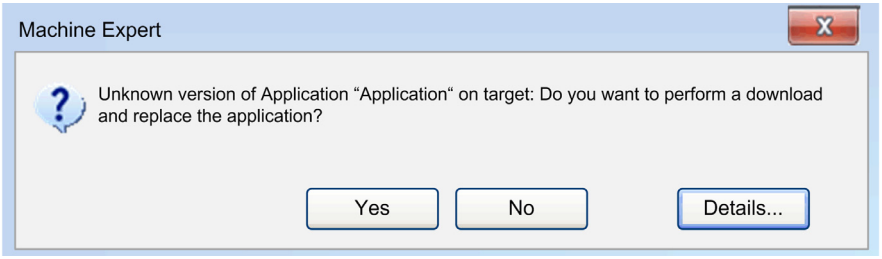
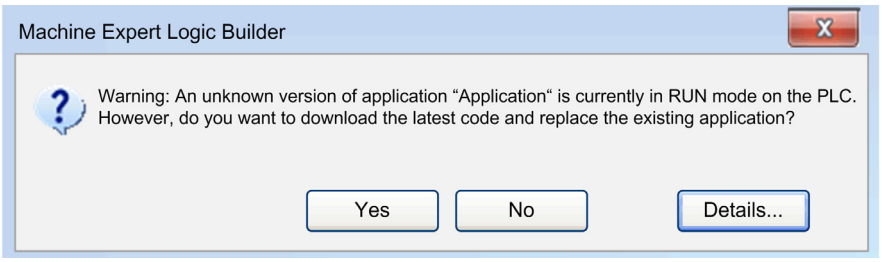
EcoStruxure Machine Expert のアプリケーションと比べて、コントローラーに読み込まれたアプリケーションは変更されている。

手順	手順内容
1	コントローラーに接続するには、**オンライン → ' アプリケーション [アプリケーション名 ; Plc ロジック]' にログイン** を選択します。
2	アプリケーションを変更してコントローラーに再度読み込む場合は、次のメッセージが表示されます。  **オンライン変更してログイン** すでに実行中のプロジェクトの変更部分のみがコントローラーに再読み込みされます。 **ダウンロードしてログイン** 変更されたアプリケーション全体がコントローラーに再度読み込まれます。 **変更せずにログイン** 変更は読み込まれません。 **注記：変更せずにログイン**オプションを選択した場合、EcoStruxure Machine Expert アプリケーションで行った変更はコントローラーにダウンロードされません。この場合、EcoStruxure Machine Expert の情報バーおよびステータスバーには動作状態として **RUNNING** が表示され、**プログラムが変更されていること (オンライン変更)** を示します。これは、情報バーおよびステータスバーに**プログラムが変更されていない**ことが示されます。**オンライン変更してログイン**オプションまたは**ダウンロードしてログイン**オプションとは異なります。 この場合変数の監視はできますが、ファンクションブロック出力の値が入力値と一致しない場合がありロジックフローが混乱する可能性があります。 **例** LD では、影響を受ける変数を基に接点の状態が監視されます。これは、接点に接続されたコイルは偽と表示されていても、青色のリンクに続く青いアニメーション表示の接点 (真を意味する) を表示させる効果があります。ST ロジックフローでは、IF 文またはループが実行されるようにみえても、条件式はプロジェクトとコントローラーで異なるため、実際には実行されません。
3	適切なオプションを選択し、**OK** をクリックします。

注記：アプリケーションのダウンロードにかかわる重要な安全情報は、各コントローラーについては、*プログラミングガイド*を参照してください。を参照してください。

コントローラーにアプリケーションをダウンロード : ケース 3

コントローラーに異なるバージョンまたは新しいバージョンのアプリケーションがあり使用可能。

手順	手順内容
1	コントローラーに接続するには、**オンライン → ' アプリケーション [アプリケーション名 ; Plc ロジック]' にログイン** を選択します。
2 a	コントローラーが RUN モードではなく、コントローラーに現在あるアプリケーションとは別のアプリケーションを読み込む場合は、次のメッセージが表示されます。  下の警告メッセージを参照し、**はい**をクリックしてコントローラーに新しいアプリケーションをダウンロードするか、**いいえ**をクリックして操作をキャンセルしてください。
2b	コントローラーが RUN モードで、コントローラーに現在あるアプリケーションとは別のアプリケーションを読み込む場合は、次のメッセージが表示されます。  下のビルドメッセージを参照し、**はい**をクリックしてコントローラーに新しいアプリケーションをダウンロードするか、**いいえ**をクリックして操作をキャンセルしてください。

⚠ 警告

装置の意図しない動作

ダウンロードを確定する前に、正しいアプリケーションであることを確認してください。

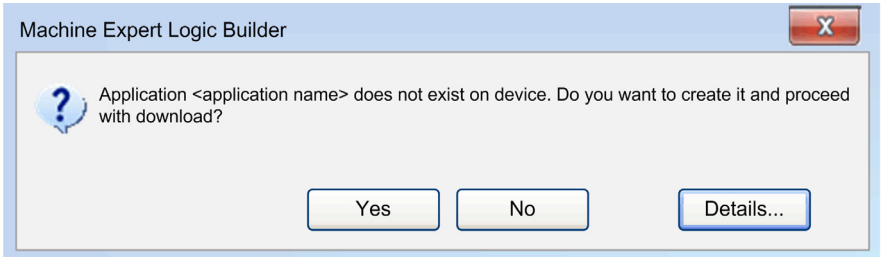
上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

はいをクリックすると、コントローラーで実行中のアプリケーションが上書きされます。

コントローラーにアプリケーションをダウンロード : ケース 4

コントローラーにアプリケーションがまだない。

手順	手順内容
1	コントローラーに接続するには、**オンライン → ' アプリケーション [アプリケーション名 ; Plc ロジック]' にログイン** を選択します。

手順	手順内容
2	コントローラーにアプリケーションがない場合、ダウンロードの確認を求められます。そのために、次の文のダイアログボックスが表示されます。  **はい**をクリックしてアプリケーションをコントローラーにダウンロードするか、**いいえ**をクリックして操作をキャンセルします。

注記：アプリケーションのダウンロードにかかわる重要な安全情報は、各コントローラーについては、*プログラミングガイド*を参照してください。を参照してください。

オンライン変更

オンライン変更コマンドにより実行中のアプリケーションプログラムが変更されますが、再起動の処理には影響しません。

- 機械が状態を維持するので、プログラムコードは初期化完了後以外であれば動作できます。
- ポインター変数は、最後のサイクルの値を保持します。オンライン変更によりサイズが変更された変数にポインターがある場合は、正しい値ではありません。ポインター変数が各サイクルで再度割り当てられることを確認します。

⚠ 警告

装置の意図しない動作

システムを動作させる前に、適切な処理を行うためのアプリケーションコードをすべてテストしてください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：特定の情報については、各コントローラーについては、*プログラミングガイド*を参照してください。***コントローラーステートコントローラーステートの詳細*** の章を参照してください。

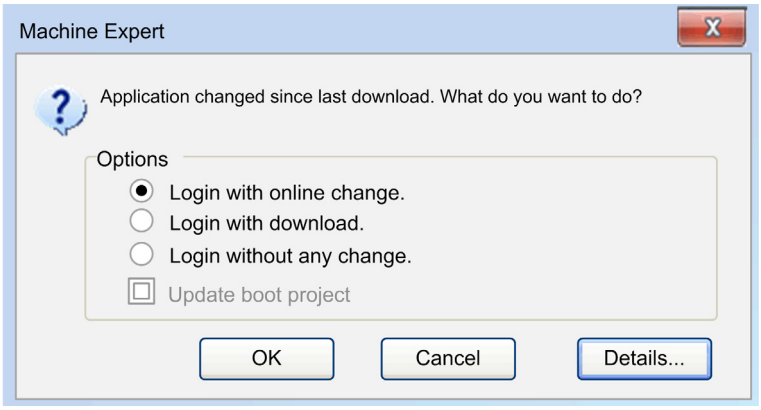
コントローラーで現在実行中のアプリケーションプロジェクトが、最後のダウンロード以降プログラミングシステムで変更された場合、プログラムは実行されたままプロジェクトの修正されたオブジェクトのみがコントローラーに読み込まれます。

Online Change Memory Reserve ビューで (*160* ページ)、ファンクションブロックのオンライン変更用のメモリー予約を設定します。ファンクションブロックの宣言を変更しオンラインで変更を行った後は、ファンクションブロックのインスタンスを新しいメモリー領域にコピーする必要はありません。

暗黙的オンライン変更

変更されたアプリケーション (最後のダウンロード時にプロジェクトフォルダーに保存された COMPILEINFO により確認) に再度ログインしようとすると、オンライン変更を実行するか、ダウンロードするか、または変更せずにログインをするかを尋ねられます。

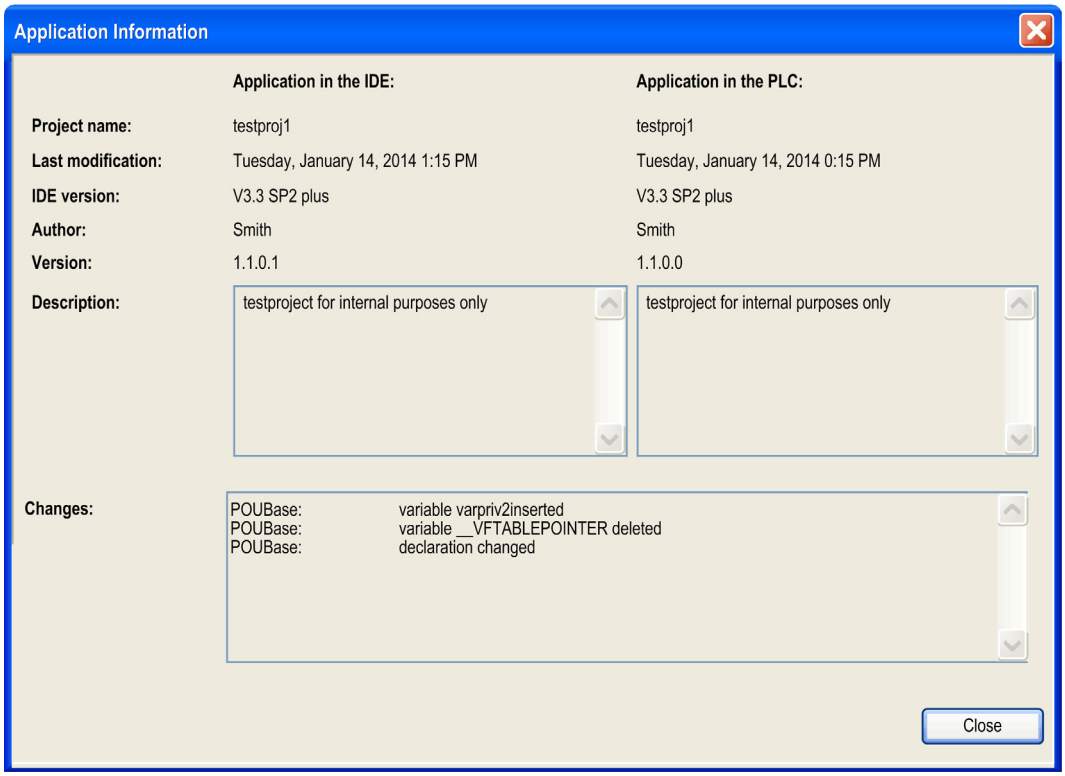
ログインダイアログボックス :



要素の説明

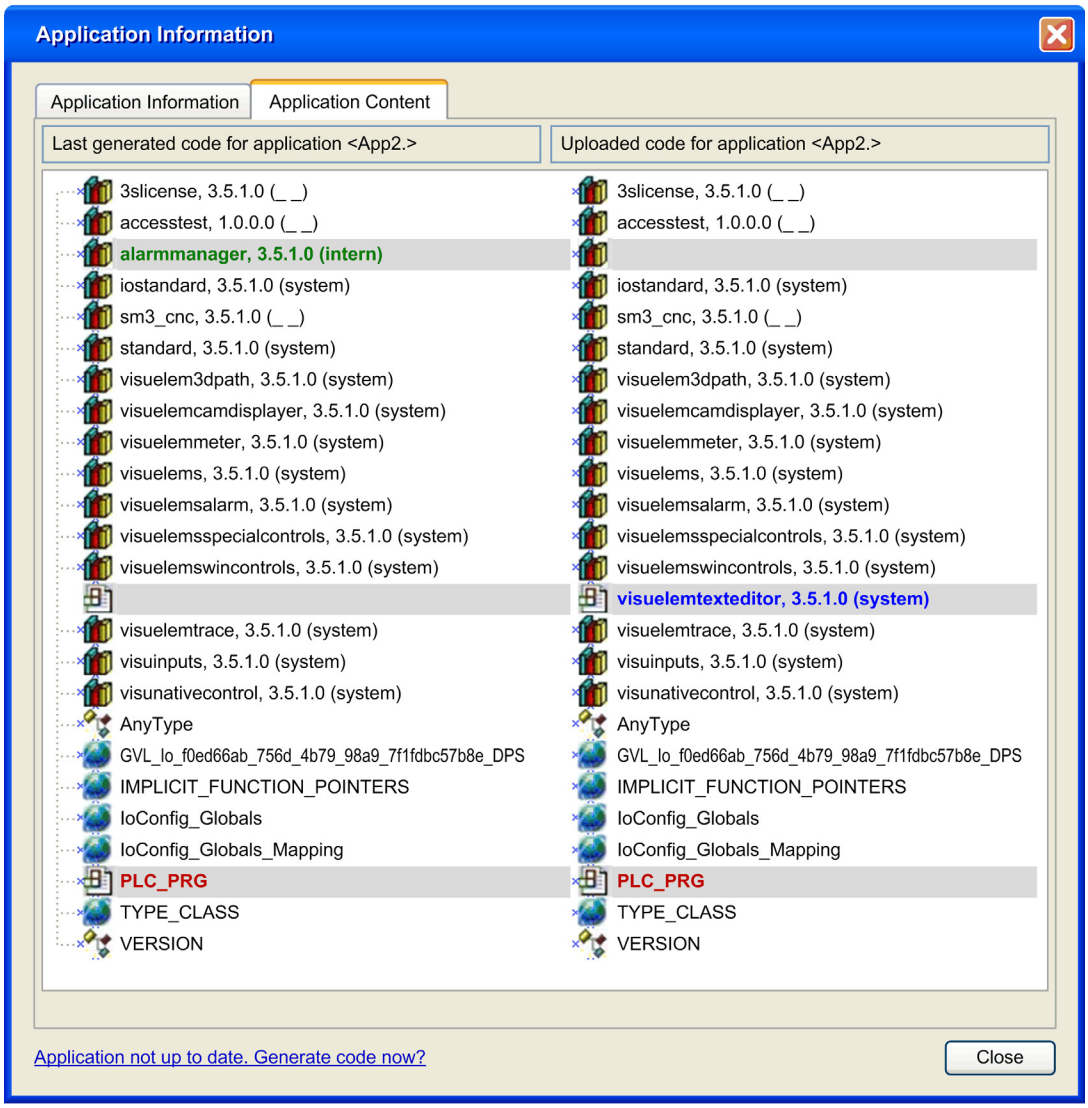
要素	詳細
オンライン変更してログイン	デフォルトでは、このオプションが選択されています。**OK** をクリックしてダイアログボックスを確認すると変更内容が読み込まれ、すぐに各オブジェクトのオンラインビュー (監視) に表示されます。
ダウンロードしてログイン	完全にアプリケーションプロジェクトを読み込んで初期化するには、このオプションを有効にします。
変更せずにログイン	コントローラーで実行中のプログラムが変更されないようにするには、このオプションを有効にします。その後、明示的にダウンロードが実行され、アプリケーションプロジェクト全体が読み込まれます。また、次のログイン時にオンラインでの変更を実行するかを再度尋ねられる場合もあります。
Update boot project	このオプションはデフォルトで選択されていません。 このオプションを選択するには、**アプリケーション**ノードの**プロパティ**ダイアログボックスの **Boot Application** タブで、**Implicit boot application on Online Change** オプションを有効にします。 その後、オンラインでの変更で起動アプリケーションが自動的に作成されます。
詳細	このボタンをクリックすると、コントローラーで現在利用可能なアプリケーションと比較するために、IDE (統合開発環境、例えば、EcoStruxure Machine Expert) の現在のアプリケーションの**アプリケーション情報**ダイアログボックス (**プロジェクト名、最後の変更、IDE バージョン、作成者、詳細**) を取得できます。次の図を参照してください。

Application Information ダイアログボックス



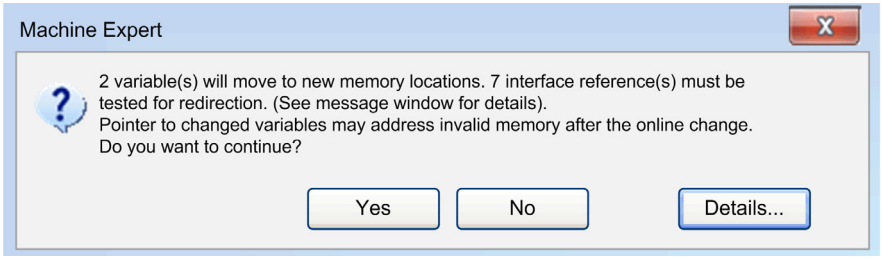
詳細は、*ログイン* の章 (*199* ページ) を参照してください。

Application Information ダイアログボックスの **Application Content** タブ



ビュー → プロパティ ダイアログボックスの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **Application Build Options** タブで**アプリケーション情報のダウンロード**オプションが有効になっている場合、このタブには以下が表示されます。コントローラーから読み込まれたアプリケーションの内容は、**Uploaded code for application <App2>** 列に表示され、プログラミングシステム内のアプリケーションの内容と比較できます。プログラミングシステムでアクティブなアプリケーションの最新バージョンを使用して、左の列の **Last generated code for application <App2>** を更新するには、**Application not up to date.** ボタンをクリックします。**Generate code now?** ボタンをクリックします。アプリケーションの内容が比較され、**プロジェクト → 比較** 機能のように、オブジェクトが色でマークされます。*(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。このより詳細な情報は、新しいアプリケーションをダウンロードする影響を評価するのに役立ちます。

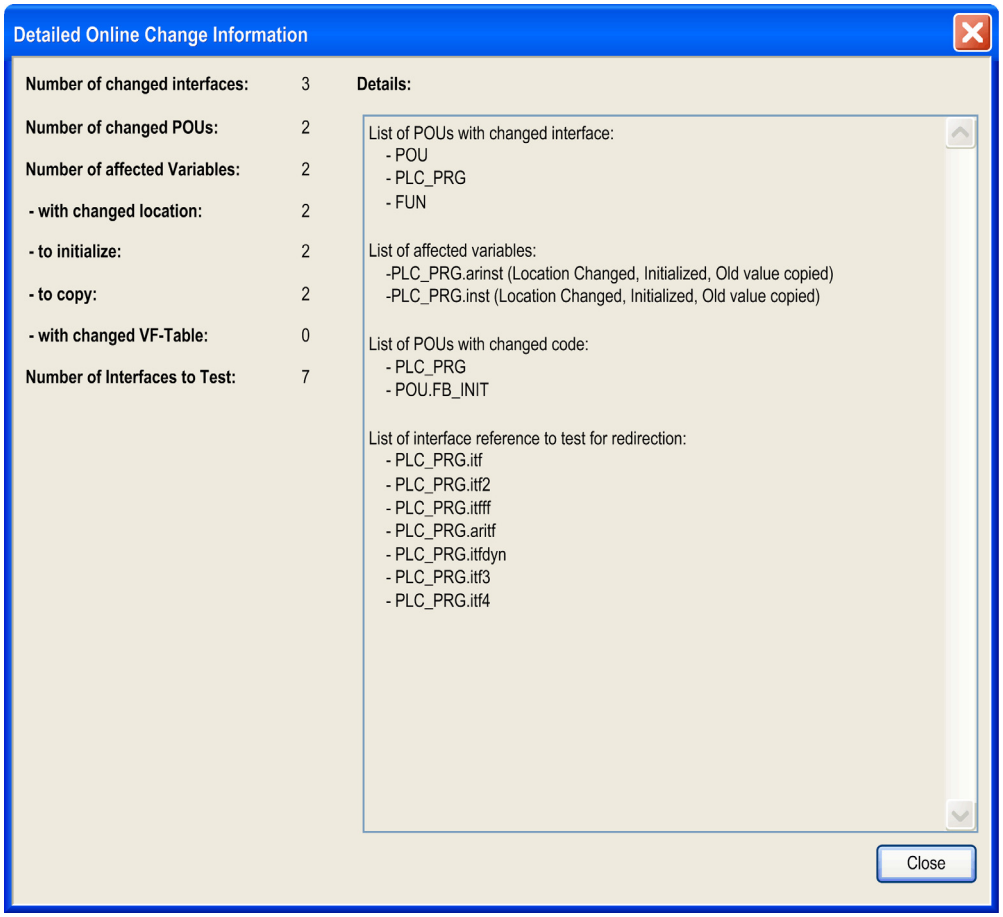
オンライン変更によってダウンロードコードが大幅に変更される場合 (例えば、ポインターアドレスの移動またはインターフェイスのリファレンス (*147* ページ) のリダイレクトが必要など) は、**オンライン変更**ダイアログボックスで **OK** の確認をした後、ダウンロードが実行される前にまた別のメッセージボックスが表示されます。検討する必要がある影響が示され、オンライン変更処理を中止するオプションも表示されます。



注記：暗黙的チェックファンクション (CheckBounds など) をアプリケーションから削除した後は、**オンライン変更**はできず、ダウンロードのみができます。該当するメッセージが表示されます。

このメッセージボックスの**詳細**ボタンをクリックすると、番号および変更されたインターフェイス、POU、影響を受けた変数のリストなどの詳細情報が表示されます。

Detailed Online Change Information ダイアログボックス



明示的オンライン変更

特定のアプリケーションで明示的にオンライン変更処理を実行するには、**オンライン変更**コマンド (初期設定で、**オンライン**メニュー内) を実行します。

クリア操作 (**ビルド → すべてクリーン**、**ビルド → クリア**) の後は、変更したプロジェクトの**オンライン変更**はできません。この場合、最後のダウンロード以降に変更したオブジェクトの情報が削除されます。従って、完全なプロジェクトのみがダウンロードできます。

注記：
オンライン変更コマンドを実行する前に、次の点を考慮してください。

- 変更されたコードに論理エラーがないことを確認します。
- ポインター変数は、最後のサイクルの値を保持します。置き換えられた変数をポイントしている場合、正しい値ではなくなります。そのため、各サイクルでポインター変数を再度割り当ててください。

ダウンロード処理に関する情報

プロジェクトが**ログイン**時に完全に読み込まれたとき、または**オンライン変更**時に部分的に読み込まれたときは、**メッセージ**ビューに生成されたコードのサイズ、グローバルデータのサイズ、コントローラーの必要なメモリー領域、およびオンライン変更の場合は影響を受けた POU の情報が表示されます。

注記：オンラインモードでは、デバイスまたはモジュールの設定は変更できません。デバイスのパラメーターを変更するには、アプリケーションからログアウトしてください。バスシステムによっては、オンラインモードで変更できる特別なパラメーターがあります。

起動アプリケーション (起動プロジェクト)

ダウンロードが正常に終了するたびに、有効なアプリケーションがコントローラーシステムフォルダーの *<application name>.app* ファイルに自動的に格納され、起動アプリケーションとして使用できます。起動アプリケーションは、コントローラーの開始時 (起動時) に自動的に起動されます。有効なアプリケーションの起動アプリケーションをダウンロードするには、**Create Boot application** (**オンライン**メニューから使用可能) コマンドを実行してください。

また、オフラインモード中に起動アプリケーション (*202* ページ) を作成することもできます。

別のパソコンのプログラミングシステムから同じコントローラーに接続する場合、またはオンライン変更やダウンロードをせずに別のパソコンから有効なアプリケーションを取得する場合は、*他のシステムへのプロジェクトの転送*の項に記述されている手順に従ってください。

他のシステムへのプロジェクトの転送

プロジェクトを別のコンピューターに転送するには、プロジェクトアーカイブ *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用します。

コントローラー *xy* 上で既に実行しているプロジェクトを、PC1 のプログラミングシステムから PC2 のプログラミングシステムに転送できます。オンライン変更またはダウンロードを必要とせずに PC2 から同じコントローラー *xy* に再接続できるようにするには、プロジェクトアーカイブを作成する前に以下のプロジェクト設定を確認します。

以下の手順を実行します。

1. 純粋なインターフェイスライブラリーを除き、最終バージョンのライブラリーのみがプロジェクトに含まれていることを確認します。(**ライブラリーマネージャー**を開き、修正バージョンではなく *(EcoStruxure Machine Expert, ファンクションおよびライブラリーユーザーガイド 参照)* アスタリスク (*) が付いた項目を確認します。)
2. **Project Settings → Compile options** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で最終バージョンのコンパイラーが設定されていることを確認します。
3. **Project Settings → ビジュアライゼーションプロファイル**ダイアログボックスで確定されたビジュアライゼーションプロファイルが設定されていることを確認します (詳細については、EcoStruxure Machine Expert オンラインヘルプの *ビジュアライゼーション*を参照してください)。
4. 現在開いているアプリケーションが、コントローラーですでに使用できるアプリケーションと同じであることを確認します。つまり、起動プロジェクト (**Online → Create boot application** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 参照) はプログラミングシステムのプロジェクトと同一にしてください。プログラミングシステムウィンドウのタイトルバーにあるプロジェクトタイトルの後にアスタリスクがある場合、そのプロジェクトは変更されているがまだ保存されていません。この場合、起動プロジェクトとは異なる可能性があります。必要に応じて、プロジェクトを別のパソコンに転送する前に (新しい) 起動プロジェクトを作成し (一部のコントローラーではダウンロード時に自動的に実行されます)、ダウンロードしてからコントローラーでプロジェクトを起動させます。
5. 次の情報を使用してプロジェクトアーカイブを作成します。**ダウンロード情報ファイル**、**ライブラリープロファイル**、**参照デバイス**、**参照ライブラリー**, **視覚化プロファイル**。
6. ログアウトします。必要に応じて、PC 2 に再接続する前にコントローラー *xy* を停止し、再起動します。
7. 手順 5 のリスト表示と同じ情報オプションを有効にした PC 2 でプロジェクトアーカイブを展開します。

注記：オンラインで変更せずにログインする場合は、必ずコントローラーでアプリケーションを作成およびダウンロードするために使用された EcoStruxure Machine Expert バージョンを使用してください。

9.3
アプリケーションの実行

アプリケーションの実行

概要

ここでは、アプリケーションの開始 / 停止方法を説明します。

EcoStruxure Machine Expert の RUN/STOP

コントローラーに接続されたパソコンで実行している EcoStruxure Machine Expert を使用して、コントローラーを開始および停止します。

アプリケーションを起動するには、**オンライン → Start 'Application [ApplicationName: Plc logic]'**、またはメニューバーの **F5** キー、または **Start 'Application [ApplicationName: Plc logic]'** をクリックします。

アプリケーションを起動するには、**オンライン → Stop 'Application [ApplicationName: Plc logic]'**、またはメニューバーの **Shift** + **F8** キー、または **Stop 'Application [ApplicationName: Plc logic]'** をクリックします。

コントローラー用 RUN/STOP 入力

この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。(例えば、運転 / 停止の章 *(Modicon M251 Logic Controller, Hardware Guide 参照)*).

9.4
アプリケーションの保守

このセクションについて

このセクションには次の項目が含まれています。

監視

概要

オンラインモードでは、さまざまな方法でコントローラーのオブジェクトの現在値を表示できます。

- オンライン時に、プログラミングエディター画面にオブジェクトの値を表示できます。詳細については、各エディターの説明を参照してください。
- 宣言エディターのオンラインビューでオブジェクト値を表示できます。詳細については、宣言エディター (*341* ページ) の説明を参照してください。
- **ウォッチ**コマンドにより、リストにオブジェクトを個別に表示できます。詳細については、ウォッチビュー / ウォッチリストエディター (*386* ページ) の説明を参照してください。ウォッチビューに変数を挿入するには、ウォッチビューを選択し、コンテキストメニューから**ウォッチリストの追加**コマンドを実行します。
- トレースサンプリングを介して値を表示できます。コントローラーからの変数値を記録および表示します。詳細については、トレースオブジェクト機能 (*416* ページ) の説明を参照してください。
- レシピに含まれているオブジェクト値を表示できます。コントローラー上でこれらの変数を書き込みおよび監視するためのユーザー定義の変数セット。レシピマネージャー (*399* ページ) の説明を参照してください。

POU またはファンクションブロックの下に挿入されているプロパティの監視については、プロパティ (*145* ページ) の章を参照してください。

ファンクション呼び出しの監視については、*属性監視* (*536* ページ) の章を参照してください。

注記：値が有効でない場合 (例えば、負の数の平方根の計算結果など)、結果は処理、オブジェクト、および特定のコントローラープラットフォームに応じて NaN (数字ではない) または INF (無限値) が表示されます。詳細情報は、各コントローラーについては、*プログラミングガイド*を参照してください。を参照してください。

デバッグ

概要

潜在的なプログラミングエラーを評価するために、デバッグ機能を使用できます。

テスト目的で、実際の対象デバイスへの必要なリンクなしで、アプリケーションをシミュレーションで実行することもできます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。シミュレーションでは物理的にハードウェアに接続する必要はないが、デバッグをオンラインで完了させる必要があるように制限されています。

実行停止を強制するためのブレークポイント

アプリケーションプログラムに設定されているブレークポイントは、プログラムの実行を停止します。変数値は現在の実行時点で確認できます。現在位置は、コールスタックを使用してより正確に決定できます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* 。

ブレークポイントに到達したデバッグタスクだけが停止されます。他のタスクは実行を継続します。ブレークポイントを置くことのできる位置は、それぞれのプログラミングエディターによって異なります。どの場合でも、POU の最後にはブレークポイントがあります。

注記：デバッグタスクによって処理される入力 / 出力は、ブレークポイントでの停止時には更新されません。これは、デバイスエディターの **PLC 設定**タブ (*110* ページ) の**停止中に IO を更新**オプションが有効である場合でも適用されます。

ブレークポイントに関するコマンドの詳細については、*ブレークポイントのコマンド*の章を参照してください。**ブレークポイント** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* にすべてのブレークポイントの概要が表示され、ブレークポイントを追加、削除、および変更できます。

条件付きブレークポイント

ブレークポイントでの停止は、以下の条件によって異なります。

- 特定のブール式がその時点で TRUE の場合。
- 実行中のサイクル番号。
- 実行中のタスク。
- 変数の値が変更されたとき (データブレークポイント)。

特定のデバッグタスクを宣言すると、複数のタスクがエラーチェックの影響を受けないようにすることができます (*複数のタスクをもつアプリケーションのブレークポイントおよびステップ*の項を参照 (*216* ページ))。

実行ポイント

オンラインモードでは、**ブレークポイント** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用して、ブレークポイントとデータブレークポイントを実行ポイントになるように変更できます。その後、プログラムはその位置で停止しませんが、コード処理は開始されます。

シンボル

シンボル	詳細
	ブレークポイントが有効です。
	ブレークポイントが無効です。
	ブレークポイントがエディターで開かれている別のファンクションブロックのインスタンスで設定されています。
	ブレークポイントで停止します。
	条件付ブレークポイントが有効です。

シンボル	詳細
	条件付ブレークポイントが無効です。
	実行ポイントが有効です。
	実行ポイントが無効です。
	条件付き実行ポイントが有効です。
	条件付き実行ポイントが無効です。
	現在のステップ位置。 各行の前に黄色の矢印が示され、関連する処理の背後が黄色の影で表示されます。
	データブレークポイントが有効です。
	データブレークポイントが無効です。
	データブレークポイントで停止します。
	データ実行ポイントが有効です。
	データ実行ポイントが無効です。
	データ実行ポイントで停止します。
	条件付きデータ実行ポイントが有効です。
	条件付データブレークポイントが有効です。

プログラムのステッピング

ステッピングを使用すると、デバッグするためにオンラインモードでアプリケーションプログラムの実行を制御できます。ステッピングコマンドを使用する前に、例えばブレークポイントを設定して、プログラムを定義したプログラムステップで停止する必要があります。基本的には、ステップイン命令によってある命令から次の命令に入り、次の命令をステップオーバーまたは命令からステップアウトします。ステップコマンドの詳細については、*ブレークポイントに関するコマンド (EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の章を参照してください。

EcoStruxure Machine Expert は次のステッピング機能を提要します。

- **Step into** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: シングルステップの実行。ステップ内で呼び出された POU にも適用されます。
- **Step over** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: シングルステップの実行。POU はステップ内で完全に実行されます。
- **Step out** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: アプリケーションプログラムに呼び出しがない場合は、実行されたアプリケーションの先頭にジャンプして戻ります。
- **Run to Cursor** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: 一時的に定義可能な位置までプログラムを実行します。
- **Set next statement** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: 次のステップで実行する命令 (ステートメント) の定義。

- **Show next statement** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: 次のステップで実行する命令 (ステートメント) にジャンプします。
- **コールスタック** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*: このビューは、完全な呼び出しパスで達しているステップ位置を示します。

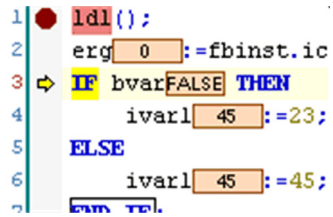
アプリケーションプログラムをステップスルーするときに可能な停止位置は、エディターの種類によって異なります。現在の位置は黄色い矢印で示されています。

Call Tree ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* には、アプリケーションのコンパイル前でも、アプリケーションプログラムの呼び出し構造内のファンクションブロックの場所が表示されます。

処理のステップインの例

ブレークポイントから開始して、ステップコマンドで 1 行づつコマンドを実行できます。

ステップインの例



データブレークポイント

データブレークポイントを使用すると、変数の値が変化したときにプログラムの実行を停止できます。

この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

定義できるデータブレークポイントの数は、使用しているコントローラーで使用可能なレジスタの数によって異なります。データブレークポイントの設定時に最大数に達すると、メッセージが表示されます。

通常のブレークポイントと同様に、データブレークポイントには次のオプションがあります。

- データブレークポイントでの停止を追加の条件にリンクさせるには、*条件付きブレークポイント* を参照してください (*214* ページ)。
- プログラムが停止せずに特定のコードを処理するデータ実行ポイントにデータブレークポイントを再定義するには、*実行ポイント* を参照してください (*214* ページ)。

複数のタスクがあるアプリケーションのブレークポイントとステップ

同時に複数のタスクに対してデバッグを実行することはできません。ブレークポイントまたはステッピングを使用してタスクに取り組んでいる間、他のタスクではブレークポイントは無視されます。

POU が複数のタスクによって使用されているためにブレークポイントが複数のタスクによってヒットされる可能性がある場合は、最初に実行されたタスクのみが停止されます。単一ステップの場合、または停止後にデバッグを続行する場合はこれを考慮してください。1 つのタスク (デバッグタスク) のみを考慮する場合は、ブレークポイント条件プロパティ (**ブレークポイント → 新しいブレークポイント**ダイアログボックスの**条件**タブ) で指定できます。

処理中に正確な値を表示するためのフロー制御

フロー制御を有効にして、アプリケーションのすでに実行された部分を追跡することができます。2 サイクル間の変数値のみを表示する標準監視とは対照的に、フロー制御はレンダリングされた瞬間に各処理ステップの値を返します。*メニューコマンドオンラインヘルプの フロー制御*の説明も参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

実行停止時の変数値

実行が停止されるたびに、EcoStruxure Machine Expert は、現在のサイクル内の変数の値を表示します (監視)。現在位置を特定するために、コールスタックを表示することができます。*メニューコマンドオンラインヘルプの コールスタック*の説明も参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

コアダンプ

例外のコアダンプ

コアダンプは、アプリケーションデータのメモリースナップショットです。ランタイムシステムでサポートされている場合、コアダンプは、例外が検出されたときにコントローラー上のアプリケーションディレクトリ内のファイル < *アプリケーション名* >.*core* に自動的に保存されます。

オンラインモードでは、アプリケーションが現在ブレークポイントで停止している場合や例外が検出された場合にも、コアダンプを明示的に生成できます。この場合、コアダンプファイルはプロジェクトディレクトリにのみ保存され、コントローラーには保存されません。

オフラインモードでは、アプリケーションのコアダンプをコントローラーから EcoStruxure Machine Expert プロジェクトに読み込めます。その後、例外が検出されてコアダンプが作成された時点のデータ、および値と共にアプリケーションがオンラインビューに表示されます。

また、プロジェクトでコアダンプが作成されたときからコントローラーのログファイルを表示できます。

注記：コアダンプビューに変数値を正しく表示するために、ファームウェアで機能がサポートされている必要があります。コアダンプをプロジェクトに読み込むことによって作成されたアプリケーションのオンラインビューでは、メニューコマンドは使用可能として表示されますが、このステータスでは使用できません。これらのコマンドのいずれかを選択すると、メッセージが表示されます。
コアダンプ機能をサポートしているかどうかを確認するには、コントローラーのプログラミングマニュアルを参照してください。

注記：ランタイムシステムでの例外処理は、呼び出しスタック内の変数値の一部を上書きする可能性があります。その結果、元の値が失われ、ファンクションやメソッドの変数を監視するときに重要な情報が表示されなくなる可能性があります。

プロジェクトアーカイブを作成するために、次の点を考慮してください。

条件	結果
コアダンプの作成コマンドを選択して明示的にコアダンプを作成した場合。	アーカイブ設定でオプションが自動的に使用可能になります。
コアダンプをコントローラからコピーした場合。	**Additional files** を選択することによってのみプロジェクトアーカイブに追加することができます。

注記：コアダンプ付きのプロジェクトアーカイブにはダウンロード情報を含める必要があります。含まれない場合は、ダンプは使用できません。

コアダンプを使用して例外を分析する

前提条件

- コントローラー上で例外が発生したアプリケーションでプロジェクトが開かれている。
- ランタイムシステムはコアダンプを作成することができる。
- プロジェクト内のアプリケーションはオフラインモードである。

手順	手順内容
1	コマンド**デバッグ → コアダンプのロード**を実行して、コントローラーからコアダンプを読み込みます。 **結果**：アプリケーションのオンラインビューが表示されます。エラーが検出された時点からの変数値とコールスタックが表示されます。これには、デバイス設定のマッピングダイアログ、およびタスク設定の I/O 変数の値も含まれます。**コアダンプがロードされました**というメッセージがステータス行に表示されます。コアダンプファイルは < *プロジェクト名* >.< *デバイス名* >.< *アプリケーション名* >.< *アプリケーションガイド* >.*core* としてローカルディレクトリーにコピーされます。
2	**デバッグ → デバイスログをコアダンプからロード**コマンドを実行して、エラーがプロジェクトに検出された時点からのデバイスログを読み込みます。 **結果**：ログビューが開き (デバイス設定のオンラインモードの場合と同様)、コアダンプが作成された時点のイベントが表示されます。
3	コアダンプの分析が完了後、**デバッグ → コアダンプを閉じる** コマンドを実行します。 **結果**：アプリケーションのコアダンプビューが閉じて、プロジェクトは通常のオフラインモードのビューに戻ります。

実行中のアプリケーションのコアダンプを手動で作成する

前提条件
- アプリケーションはオンラインモードである。
- ランタイムシステムがコアダンプ機能に対応している。

手順	手順内容
1	**デバッグ → コアダンプの作成**コマンドを実行します。 **結果** : コアダンプの作成が開始します。プログレスバーがキャンセルボタンと一緒にステータス行に表示されます。 コアダンプファイルは < *プロジェクト名* >.< *デバイス名* >.< *アプリケーション名* >.< *アプリケーションガイド* >.*core* としてローカルディレクトリーに保存されます。

例外のプログラムエラーハンドリング

IEC 61131-3 拡張として、EcoStruxure Machine Expert は、例外をキャッチするために特定の演算子 (__TRY、__CATCH、__FINALLY、__ENDTRY) をサポートします。プログラムを停止する代わりに、エラーが検出された場合に実行されるステートメントをプログラミングできます。詳細については、これらの動作の説明を参照してください (*670* ページ)。

この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

プログラミングサポート

概要

EcoStruxure Machine Expert は、プログラムコードの入力を容易にし、検出されたエラーを早い段階で示す設定、ダイアログ、および機能を提供します。

詳細については、次の機能の説明を参照してください。

- **入力アシスタント** (*392* ページ)
- **Intelli-sense** (*558* ページ)
- **自動宣言 ...** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **スマートコーディング** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **Static Analysis Light** (*223* ページ)

リファクタリング

概要

一般的にリファクタリングは、既存のソフトウェアコードをその動作を変更することなく再構築および改善するプロセスです。

EcoStruxure Machine Expert のリファクタリングでは、オブジェクト名と変数名の名前変更機能を提供します。変更されたオブジェクトと変数のすべての使用場所を示し、それらすべてを一度に、または個々の名前を変更することができます。リファクタリングについて、実行するかどうか、またどこで実行するかの確認をするように設定できます。明示的に、**リファクタリング → 名前の変更**コマンドを使用できます。

詳細は*オプション*、*リファクタリング*も参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

" の名前を変更

リファクタリング → " の名前を変更コマンドは、グローバルな、プロジェクト全体にわたるオブジェクト名および変数名の名前変更に使用します。

宣言内の変数名にカーソルを合わせた場合、またはナビゲータで次のいずれかのオブジェクトを右クリックした場合、コンテキストメニューで使用できます。

- ファンクション
- POU
- GVL
- メソッド
- プロパティ
- デバイス

このコマンドは、単位変換にエディター内で変数と単位変換を選択した場合にも使用できます。

このコマンドはオンラインモードでは使用できません。

名前の変更処理

リファクタリング → " の名前を変更コマンドを実行すると、ダイアログが表示されます。関係のある名前が使用されている場所が表示されます。ウィンドウの右側には特定の場所、左側には関係のあるオブジェクトがマークされているナビゲーションツリーが表示されます。

それぞれの場所に対して次のコマンドを個別に選択できます。

- **Reject/accept this change**: 右側のウィンドウで変更を却下 / 承認します。
- **Reject/accept this object**: 右側のウィンドウで関係するオブジェクトのすべての変更を却下 / 承認します。
- **Reject/accept whole project**: 右側のウィンドウでプロジェクト内のすべての変更を却下 / 承認します。

承認された変更は黄色の背景色で表示され、拒否された変更は灰色の背景色で表示されます。

変数を追加

リファクタリング → 変数を追加コマンドを使用すると、POU で変数を宣言でき、さまざまな使用場所で自動更新を実行します。

カーソルが宣言エディターにある場合は、コンテキストメニューまたは**編集 → リファクタリング**メニューで使用できます。**自動宣言** ダイアログボックス (*487* ページ) が開きます。

OK をクリックして**自動宣言**ダイアログボックスを確定すると、**リファクタリング**ダイアログボックスが開きます。左側のプロジェクトのツリー構造で構成されています。新しい変数が使用されている POU がツリーで強調表示されます。POU をダブルクリックして、ダイアログボックスの右側に詳細ビューを開くことができます。この宣言セクションと右側の POU の実装では、新しい宣言は黄色で強調表示されます。

どこでどの変更を承認するかを決定する前に、宣言セクションの上のリストおよび右側の POU の実装から適切なオプションを選択します。

オプション	詳細
Add inputs with placeholder text	このオプションを選択すると、新しい変数が実装コード内に現れるたびに標準のプレースホルダテキスト _REFACTOR_ が挿入されます。影響を受ける場所を特定するために、後でこの標準テキストを検索できます。

オプション	詳細
Add inputs with the following value	このオプションを選択すると、新しい変数が実装コード内に現れるたびに初期化する値が挿入されます。**Add inputs with the following value** オプションの右側にあるテキストボックスに初期化する値を入力します。

変更を承認または拒否するには、この章の*名前の変更処理*の説明に従って、変更した場所を右クリックするか、ダイアログボックスの左側または右側にあるコマンドを実行します。

変数を追加する例

例 1 :

リファクタリングすることで、fun ブロックは初期化する値 1 を持つ新しい入力変数 input3 を受け取ります。この変更は次のような作用があります。

リファクタリング前

```
fun(a + b, 3, TRUE);
fun(input1:= a + b , input2 :=3 , inputx := TRUE);
```

リファクタリング後

```
fun(a + b, 3, input3:=1, TRUE);
fun(input1:= a + b , input2 :=3 , input3:=_REFACTOR_,inputx := TRUE);
```

例 2:

リファクタリングすることで、fun ブロックはプレースホルダーテキストの _REFACTOR_ を持つ新しい入力変数 input3 を受け取ります。

リファクタリング前

```
inst(input1 := a + b, input2 := 3, inputx := TRUE);
fun(a + b, 3, TRUE);
```

リファクタリング後

```
inst(input1 := a + b, input2 := 3, input3 := _REFACTOR_, inputx := TRUE);
fun(a + b, 3, input3:=_REFACTOR_, TRUE);
```

” を削除

リファクタリング → ” を削除コマンドを使用すると、POU から入力変数または出力変数を削除できます。さまざまな使用場所で POU から自動的に削除されます。

このコマンドは、カーソルが宣言エディター内の削除される変数の識別子に置かれている場合、コンテキストメニューまたは**編集 → リファクタリング**メニューで使用できます。削除に関する情報を提供するダイアログボックスが開きます。メッセージを確認すると、**リファクタリング**ダイアログボックスが開きます。

リファクタリングダイアログボックスで提示された変更を承認すると、それらの入力変数または出力変数は、それぞれの POU からそれぞれの使用場所で自動的に削除されます。

注記 : CFC では、削除された入力または出力とブロックの間の接続のみが削除されます。入力または出力自体はチャートに残ります。

ST で変数を削除する例

POU では、リファクタリングによって input4 i 入力変数が削除されます。さまざまな使用場所で POU から自動的に削除されます。

削除前

```
inst(input1 := a + b, input2 := 3, input4 := 1, input5 := TRUE);
fun(a + b, 3, 1, TRUE);
```

削除後

```
inst(input1 := a + b, input2 := 3, input5 := TRUE);
fun(a + b, 3, TRUE);
```

参照されたピンを更新

リファクタリング → 参照されたピンを更新 コマンドは、CFC エディターおよび FBD/LD/IL エディターでのみ有効です。これは **Reset Pins** コマンド、および **Update parameters** コマンドに対応します。

リファクタリング → 参照されたピンを更新 コマンドは、現在の POU 宣言に従って、異なる使用時点で POU の入力ピンと出力ピンを更新します。

このコマンドは、カーソルが POU 名、POU 宣言の最初の行、またはデバイスエディターにある場合、コンテキストメニューまたは**編集 → リファクタリング**メニューで使用できます。

変数を並べ替え

リファクタリング → 変数を並べ替え コマンドを使用すると、VAR_INPUT, VAR_OUTPUT、または VAR_IN_OUT スコープ内の変数宣言の順序を変更できます。

例 :

```
VAR_INPUT
  invar2 : INT;
  invar1 : INT;
  in : DUT;
  bvar : BOOL;
  invar3 : INT;
```

宣言の順序を変更するには、次の手順に従います。

手順	手順内容
1	宣言部分を右クリックして、**リファクタリング → 変数を並べ替え** コマンドを実行します。 **結果** : **並べ替え**ダイアログボックスが開き、VAR_INPUT 変数のリストが表示されます。
2	invar1 : INT; エントリーを invar2. エントリーの前にドラッグします。 **結果** : invar1 宣言が一番上になります。
3	**OK** をクリックしてダイアログボックスを閉じます。 **結果** : **リファクタリング**ダイアログボックスが開きます。影響を受ける要素は黄色でマークされます。
4	**OK** をクリックして、ファンクションブロックのパラメーターの新しい順序を承認します。 パラメーターの変更された順序は、POU の呼び出し使用時にも反映されることに注意してください。

変数宣言の修正とリファクタリングの自動適用

次の例に示すように、リファクタリングは変数の名前変更 (**自動宣言**を使用) に役立ちます。

この例の前提条件として、入力 iA を持つファンクションブロック fb_A が使用可能でなければなりません。

手順	手順内容	コメント
1	**ツール → オプション ...** コマンドを実行します。	**結果** : **オプション**ダイアログボックスが開きます。
2	左側のカテゴリーの**リファクタリング**を選択します。	–
3	**自動宣言** セクションで、**変数の追加または削除時、または範囲の変更時**、および**変数の名前変更時** オプションを選択します。	–
4	ファンクションブロック fb_A をダブルクリックします。	–
5	fb_A の宣言で、例えば iA などの変数を選択します。	代わりに、カーソルを変数の前または中に置くこともできます。
6	**編集 → 自動宣言 ...** コマンドを実行します。	ショートカット **Shift** + **F2** も使用できます。 **結果** : **自動宣言**ダイアログボックスが開きます。変数 iA の設定が表示されます。
7	**名前**を編集して、iA を iCounter_A に変更します。	**結果** : **リファクタリング - Renaming from iA to iCounterA** ダイアログボックスが開きます。その変数が使用されている POU がツリー構造で強調表示されます。
8	**OK** をクリックして名前の変更を適用し、ダイアログボックスを閉じます。	–

Static Analysis Light

概要

Static Analysis 機能は、プロジェクトが対象システムにダウンロードされる前に、プロジェクトのソースコードで特定のコーディング指令からの逸脱をチェックします。これは、*lint* 分析ツールの基本的な考え方に従います。

プロジェクト設定 → Static Analysis Light ダイアログボックスで必要なルールセットを定義します *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

注記：コード生成ごとにチェックが自動的に実行されます。

ルールからの逸脱は、**メッセージ**ビューに**ビルド**カテゴリーのメッセージとして表示されます。エラー番号は SA< 番号 > として表示されます。

注記：分析は現在のプロジェクトのアプリケーションコードでのみ行われます。ライブラリーは分析されません。

注記：GVL 変数の場合、プロジェクト内に複数のアプリケーションがある場合は、現在動作中のアプリケーションの下のオブジェクトのみがチェックされます。プロジェクト内にアプリケーションが 1 つしかない場合は、POU プール内のオブジェクトもチェックされます。

Static Analysis Light の pragma と属性

Pragma 命令を使用して、コードの一部をチェックから除外することができます。

前提条件：**プロジェクト設定**でルールを有効にする必要があります。

注記：出力に対する複数の書き込みアクセスは、pragma を介して無効にすることはできません。

Pragma {analysis ...}

後続のコード行の特定のコーディング規則を無効にするには、pragma {analysis ...} を使用します。この目的のために、それを 2 回配置します。関係するコードの上の行 (無効化) と、関係するコードの下の行 (再有効化) です。無効にする場合はマイナス記号 (-)、再有効にする場合はプラス記号 (+) を付けて規則番号を指定する必要があります。規則によって、pragma はプログラミングオブジェクトの宣言部または実装部で使用できます。

構文

{analysis <sign><rule number>|,<further sign / rule number combinations separated by commas>}

例

規則 24 は 2 行無効になり (例えば nTest:=DINT#99 をここで記述する必要がない)、その後再び有効になります。

```
{analysis -24}
nTest := 99;
iVar := INT#2;
{analysis +24}
```

複数規則の指定

{analysis -10, -24, -18}

属性 {attribute 'analysis' := '...'}

宣言部で属性 {attribute 'analysis' := '<sign><rule number>'} を使用して、完全なプログラミングオブジェクトの特定の規則を有効または無効にすることができます。

構文

{attribute 'analysis' := '<sign><rule number>|,<further rule numbers separated by commas>'}

例

規則 33 と 31 は構造体全体で無効になります。

```
{attribute 'analysis' := '-33, -31'}
TYPE My_Structure :
STRUCT
   iLocal : INT;
   uiLocal : UINT;
   udiLocal : UDINT;
END_STRUCT
END_TYPE
```

規則 100 は配列で無効になります。

```
{attribute 'analysis' := '-100'}
big: ARRAY[1..10000] OF DWORD;
```

ダウンロード時にコントローラーにアーカイブを作成する

概要

コントローラーにアーカイブを作成すると、コントローラーに完全なプロジェクトを持つことなり保守作業に役立ちます。ダウンロードで自動的にプロジェクトアーカイブを作成に必要な手順を次に示します。

メニューコマンドオンラインヘルプ (EcoStruxure Machine Expert, Menu Commands, Online Help 参照) に記述されているように **ファイル → Source download...** コマンドを実行してプロジェクトのアーカイブを作成できます。

コントローラーにアーカイブを作成する

ダウンロードでコントローラーにアーカイブを作成するには、次の 3 つの主要タスクを行います。

1.) ダウンロードするソースを設定します。

アーカイブに追加する要素を定義するには、次の手順に沿って進めます。

手順	手順内容
1	**プロジェクト → プロジェクト設定**コマンドを実行します。
2	**プロジェクト設定**ダイアログボックスで **Source download → Additional Files...** を選択します。 次のオプションを選択できます。 アップロードされたプロジェクトに接続するオプション ● **Download information files** 別の EcoStruxure Machine Expert バージョンと接続する必要のあるオプション ● **ダウンロード情報ファイル** ● **ライブラリープロファイル** ● **参照先デバイス** ● **参照先ライブラリー** ● **ビジュアライゼーションプロファイル**

詳細は、**プロジェクト設定 - Source download** の説明を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

2.) プロジェクト (ソース) をコントローラーにダウンロード

プロジェクトをコントローラー上でアーカイブするには、以下の手順に沿って進めます。

手順	手順内容	コメント
1	**オンライン → ログイン** コマンドを実行して コントローラーに接続します (*199* ページ)。	メッセージが表示されます。
2	**オンライン → Source download to connected device** を選択します。	プロジェクトはアーカイブされコントローラーにダウンロードされます。

3.) オンライン → Source Upload... を実行して、コントローラーからプロジェクト (ソース) をアップロードします。*(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

第 IV 部
ロジックエディター

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
10	グラフィックエディターの共通機能	229
11	FBD/LD/IL エディター	231
12	コンティニュアスファンクションチャート (CFC) エディター	279
13	シーケンシャル ファンクションチャート (SFC) エディター	297
14	構造化テキスト (ST) エディター	321

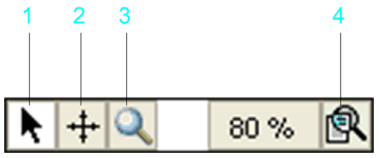
第 10 章
グラフィックエディターの共通機能

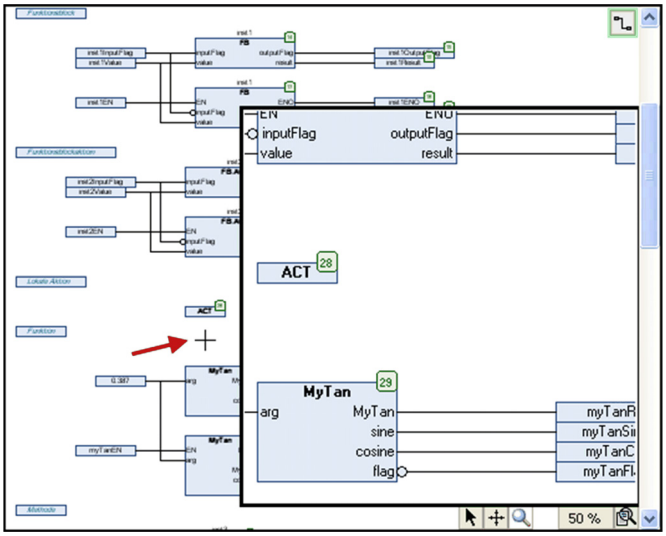
グラフィックエディターの共通機能

ズーム、パン、拡大鏡ツール

次のズーム、パン、拡大鏡ツールはすべてのグラフィックプログラミング言語のエディターで使用できます。

FBD、LD、CFC、および SFC のグラフィックエディターではエディターウィンドウの右下にツールバーが表示されます。



項目番号	名前	詳細
1	**標準編集モードに戻る**	ポインターは通常の矢印の形になり、エディターウィンドウで要素を選択して編集できる通常の編集モードに戻ります。
2	**パンツール**	ポインターが十字の矢印の形になります。エディターウィンドウの任意の場所をクリックし、マウスボタンを押したまま、ウィンドウ内でチャートの表示領域を移動させることができます。
3	**拡大鏡ツール**	この機能は、表示されているチャートを 100% 未満に縮小した場合に役立ちます。エディターウィンドウの右下にサブウィンドウが開きます。チャートの上でポインターを移動させると、このサブウィンドウにチャートのポインター部分が 100% で表示されます。 ここでウィンドウをクリックするとサブウィンドウが閉じ、サブウィンドウに表示されていたチャートが 100% のサイズで表示されます。 以前に設定したズーム倍率を維持したい場合は、**標準編集モードに戻る**ボタンを使用して通常の編集モードに戻ります。 
4	**ズームツール**	ズームボタンを押すとサブメニューが開き、チャートのズーム倍率を選択できます。他の値を指定するには、... ボタンをクリックして編集ダイアログボックスを開きます。現在のズーム率はボタンの左側に表示されます。

マウスホイールでズームするには、マウスホイールをスクロールしながら **Ctrl** キーを押します。現在のズームレベルが 10% 刻みで増減します。

第 11 章
FBD/LD/IL エディター

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
11.1	FBD/LD/IL エディターの情報	232
11.2	FBD/LD/IL 要素	258
11.3	LD 要素	275

11.1
FBD/LD/IL エディターの情報

このセクションについて

このセクションには次の項目が含まれています。

FBD/LD/IL エディター

概要

FBD (ファンクションブロックダイアグラム (*234* ページ))、LD (ラダー図 (*235* ページ))、および IL (命令リスト (*236* ページ)) の POU を編集するための複合エディターを使用できます。

ツール → オプションダイアログボックスカテゴリー **FBD、LD、および IL エディター** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の **IL** タブで EcoStruxure Machine Expert の IL 記述言語を有効にします。

共通なコマンドおよび要素のセットが使用され、3 つの言語間の自動内部変換が実行されます。オフラインモードでは、プログラマーがエディタービュー (**ビュー**) を切り替えできます。

一部変換できない特殊な要素があり、そのため一部の言語でしか表示されない要素があることに留意してください。また、IL と FBD 間では変換できない構造体があり、FBD へ戻す変換の時には正規化されます。特に、否定の式および明示的 / 暗黙的な出力の割り当てです。

カスタマイズおよび**オプション**ダイアログボックスで、FBD/LD/IL エディターの動作、外観、およびメニューを設定できます。コメントとアドレスの表示も設定できます。

エディターは二部構成のウィンドウで開きます。FBD/LD/IL でプログラミングされたオブジェクトを編集すると、上部には宣言エディター (*342* ページ)、下部にはコーディング領域が表示されます。

オブジェクトの作成時に、新しいオブジェクトのプログラミング言語を指定します。

詳細については、以下を参照してください。

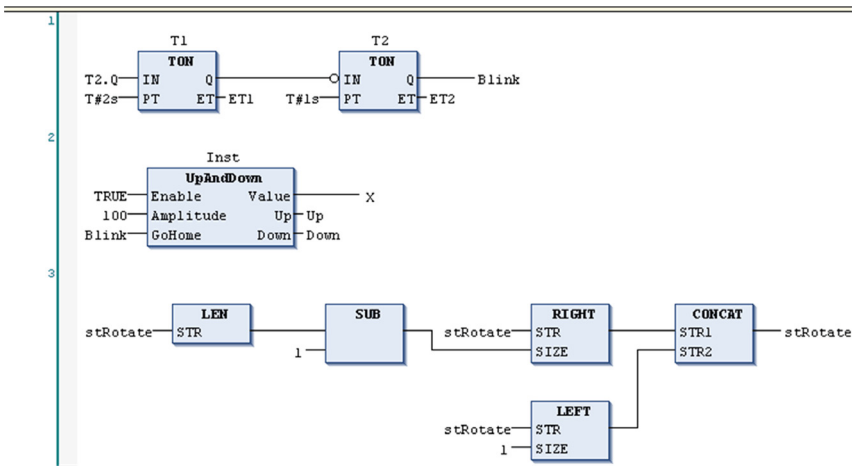
- *FBD および LD エディタービューの操作* (*241* ページ)
- *IL エディタービューの操作* (*244* ページ)

ファンクションブロックダイアグラム (FBD) 言語

概要

ファンクションブロックダイアグラムは、グラフィカルなプログラミング言語です。ネットワークの一覧で動作します。各ネットワークは、論理式、算術式、ファンクションブロックの呼び出し、ジャンプ、またはリターン命令のいずれかを表すボックスと接続線のグラフィックで構成されます。

FBD ネットワーク



ラダー図 (LD) 言語

概要

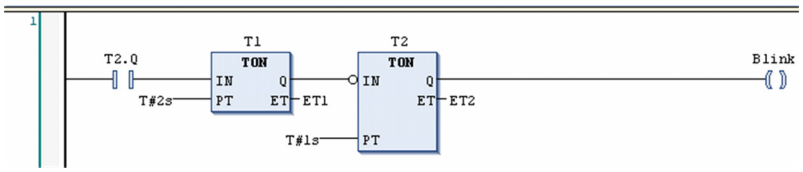
ラダー図は、電気回路の構造に似たグラフィック指向のプログラミング言語です。

ラダー図は論理スイッチの構築に適している一方、FBD のようにネットワークの作成もできます。そのため、LD は他の POU の呼び出しを制御するのに便利です。

ラダー図は一連のネットワークで構成され、左側はそれぞれが垂直方向の電流ライン (電源レール) によって制限されています。ネットワークには、接点、コイル、オプションで追加の POU (ボックス)、および接続線で構成された回路図があります。

左側には、ブール値の TRUE および FALSE に対応する条件 ON または OFF を左から右へ渡す 1 つまたは一連の接点があります。各接点に、ブール変数が割り当てられます。この変数が TRUE の場合、その条件が接続線に沿って左から右に渡されます。それ以外の場合は、OFF が渡されます。その後、ネットワークの右側に配置された 1 つまたは複数のコイルが、左側から来る ON または OFF を受け取ります。それに応じて、値 TRUE または FALSE が割り当てられたブール変数に書き込まれます。

ラダー図ネットワーク



命令リスト (IL) 言語

概要

命令リスト (IL) はアセンブラのような IEC 61131-3 に準拠したプログラミング言語です。

この言語は、アキュムレーターを基にしたプログラミングに対応しています。複数入力 / 複数出力、否定、コメント、出力のセット / リセット、および条件なし / 条件付きジャンプだけでなく、IEC 61131-3 演算子にも対応しています。

各命令は主に、LD 演算子を使用して値をアキュムレーターに読み込むことを基本にしています。その後、アキュムレーターから取り出した最初のパラメーターで処理を実行します。処理の結果はアキュムレーターにあります。そこから ST 命令で保存します。

条件付き実行またはループをプログラミングするために、IL は EQ、GT、LT、GE、LE、NE およびジャンプなどの比較演算子に対応しています。後者は、条件なし (JMP) または条件付き (JMPC / JMPCN) にできます。条件付きジャンプの場合、アキュムレーターの値は TRUE または FALSE で参照されます。

構文

命令リスト (IL) は、一連の命令で構成されています。各命令は新しい行から始まり、演算子および処理のタイプに応じてカンマで区切られた 1 つまたは複数のオペランドを含みます。修飾子で演算子を拡張できます。

命令の前の行には、後ろにコロン (:) を付けた識別マーク (ラベル) を置くことができます (ml：下記の例参照)。ラベルをジャンプ命令のジャンプ先にできます (下の例では、JMPC m1)。

行の最後の要素としてコメントを配置します。

命令の間に空白行も挿入できます。

例

```
LD          BVar1
ST          tonInst1.IN
CAL          tonInst1(
        PT:=t1,
        ET=>tOut2)
LD          toninst1.Q
JMPC          mark1
ST          tonInst2.IN
```

```
markl:
LD          iVar2
ADD          230
```

詳細については、以下を参照してください。

- *IL エディタービューの操作* (*244* ページ)
- *IL の修飾子と演算子* (*237* ページ)

IL の修飾子と演算子

修飾子

命令リスト (*236* ページ) では、次の修飾子を使用できます。

C	JMP, CAL, RET:	前の式の結果が TRUE の場合にのみ、命令が実行されます。
N	JMPC, CALC, RETC:	前の式の結果が FALSE の場合にのみ、命令が実行されます。
N	下記の*演算子*表に応じた演算子 (*修飾子*の列の *N*)	オペランドの否定 (アキュムレーター以外)。
(	下記の*演算子*表に応じた演算子 ((*修飾子*列の (左括弧))	複雑なオペランドに使用します。詳細は、複雑なオペランドの使用例一覧を参照してください (*238* ページ)。

注記：一般的に、STN、S、または R 演算子の直後に CALC (/RETC/JMPC) 文を使用することは推奨されていません。これらの命令は、アキュムレーターの値を任意に変更するためプログラミングエラーを見つけ難くします。

演算子

特定の修飾子と組み合わせて使用できる演算子を表に示します。

アキュムレーターは、前の演算の結果である現在の値を格納します。

演算子	修飾子	意味	例
LD	N	オペランドの (否定の) 値をアキュムレーターに読み込みます。	LD iVar
ST	N	アキュムレーターの (否定の) 内容をオペランド変数に格納します。	ST iErg
S	–	アキュムレーターの内容が TRUE の場合、オペランド (BOOL 型) を TRUE に設定します。	S bVar1
R	–	アキュムレーターの内容が TRUE の場合、オペランド (BOOL 型) を FALSE に設定します。	R bVar1
AND	N,(	アキュムレーターと (否定の) オペランドのビット論理積。	AND bVar2
または	N,(	アキュムレーターと (否定の) オペランドのビット論理和。	OR xVar
XOR	N,(	アキュムレーターと (否定の) オペランドのビット排他的論理和。	XOR N, (bVar1, bVar2)
NOT	–	アキュムレーターの内容のビット反転。	–
ADD	(	アキュムレーターとオペランドの加算結果をアキュムレーターにコピーします。	ADD iVar1
SUB	(	アキュムレーターとオペランドの減算結果をアキュムレーターにコピーします。	SUB iVar2
MUL	(	アキュムレーターとオペランドの積算結果をアキュムレーターにコピーします。	MUL iVar2
DIV	(	アキュムレーターとオペランドの除算結果をアキュムレーターにコピーします。	DIV 44
GT	(	アキュムレーターがオペランドより大きいことを検証し、結果 (BOOL) をアキュムレーターにコピーします。>	GT 23
GE	(	アキュムレーターがオペランドより大きい、あるいは等しいことを検証し、結果 (BOOL) をアキュムレーターにコピーします。>=	GE iVar2
EQ	(	アキュムレーターがオペランドと等しいことを検証し、結果 (BOOL) をアキュムレーターにコピーします。=	EQ iVar2
NE	(	アキュムレーターがオペランドと等しくないことを検証し、結果 (BOOL) をアキュムレーターにコピーします。<>	NE iVar1
LE	(	アキュムレーターがオペランドより小さい、あるいは等しいことを検証し、結果 (BOOL) をアキュムレーターにコピーします。<=	LE 5
LT	(	アキュムレーターがオペランドより小さいことを検証し、結果 (BOOL) をアキュムレーターにコピーします。<	LT cVar1

演算子	修飾子	意味	例
JMP	C, CN	ラベルへの無条件 (条件付き) ジャンプ。 ● JMP = 条件なしジャンプ ● JMPC = アキュムレーターが TRUE の場合の条件付きジャンプ ● JMPCN = アキュムレーターが FALSE の場合の条件付きジャンプ	JMP next
CAL	C, CN	PROGRAM または FUNCTION_BLOCK の 条件なし (条件付き) 呼び出し。 ● CAL = 条件なし呼び出し ● CALC = アキュムレーターが TRUE の場合の条件付き呼び出し ● CALCN = アキュムレーターが FALSE の場合の条件付き呼び出し	CAL prog1
RET	C, CN	条件なし (条件付き) POU のリターンおよび呼び出し元の POU へ戻るジャンプ。 ● RET = 条件なしリターン ● RETC = アキュムレーターが TRUE の場合の条件付きリターン ● RETCN = アキュムレーターが FALSE の場合の条件付きリターン	RET
)	–	遅延演算の評価	–

複数のオペランド、複雑なオペランド、ファンクション / メソッド / ファンクションブロック / プログラム / アクション呼び出し、ジャンプの使用方法および処理については、IEC オペランド (*593* ページ) および IL エディターでの操作 (*244* ページ) も参照してください。

例

複数の修飾子を使用した IL プログラムの例

```
LD    TRUE      load TRUE to accumulator
ANDN  bVar1     execute AND with negative value of bVar1
JMPC  m1        if accum. is TRUE, jump to label "m1"
LDN   bVar2     store negated value of bVar2...
ST    bRes      ... in bRes
```

```
m1:
LD    bVar2     store value of bVar2...
ST    bRes      ... in bRes
```

オペランドの使用

オペランドの使用例一覧を次の表に示します。

使用例	詳細	例
1 つの演算子に複数のオペランド	オプション ● 2 列目にオペランドをカンマで区切り、連続した行に入力します。 ● 演算子を連続した行で繰り返します。	● オプション 1: LD 2 ADD 3, 4, 6 ST iVAR ● オプション 2: LD 2 ADD 3 ADD 4 ADD 6 ST iVAR

使用例	詳細	例

複雑なオペランド	複雑なオペランドの場合、最初の行で左括弧（を入力します。
	新しい行の最初の列に右括弧を入力し、その後に続く行にオペランド項目を続けます。
		文字列はサイクルごとに 1 文字ずつ回転します。
		`LD stRotate`
		`RIGHT( stRotate`
		`LEN`
		`SUB 1`
		`)`
		`CONCAT( stRotate,`
		`LEFT 1`
		`)`
		`ST stRotate`

ファンクションブロック呼び出しとプログラム呼び出し	行 1:
	● 列 1:
	CAL または CALC 演算子
	● 列 2:
	ファンクションブロックインスタンスの名前またはプログラム名と、左括弧（。パラメーターがない場合は、右括弧）がここに入ります。
	続く行：
	● 列 1:
	パラメーター名に続き
	❍ 入力パラメーターの場合 :=
	❍ 出力パラメーターの場合 =>
	● 列 2:
	さらにパラメーターがある場合、パラメーター値の後にカンマ，が入ります。
	右括弧）を最後のパラメーターの後に入れます。
	IEC 規格によると複雑な式はここでは使用できません。そのような構成は、呼び出される前にファンクションブロックまたはプログラムに割り当ててください。
		`CAL POUToCall(`
		`iCounter:=1,`
		`iDecrement:=1000,`
		`wError:=wResult)`
		
		`LD POUToCall.bError,`
		`ST bErr`

ファンクションの呼び出し	行 1:
	● 列 1:
	LD
	● 列 2:
	入力変数
	行 2:
	● 列 1:
	機能名
	● 列 2:
	カンマで区切られた入力パラメーター
	戻り値はアキュムレーターに書き込まれます。
	行 3:
	● 列 1:
	ST
	● 列 2:
	戻り値が書き込まれる変数
		`LD X7`
		`GeomAverage 25`
		`ST Ave`

アクション呼び出し	ファンクションブロック呼び出しとプログラム呼び出しに似ています。 アクション名は、ファンクションブロックインスタンスの名前またはプログラム名に追加されます。	`CAL POU1.ResetAction`

使用例	詳細	例
ジャンプ	● 列 1: JMP または JMPC 演算子 ● 列 2: ジャンプ先ネットワークのジャンプラベル名 ● 条件なしジャンプ: 先行の命令シーケンスは、次のいずれかのコマンドで終了してください。 ❍ ST ❍ STN ❍ S ❍ R ❍ CAL ❍ RET ❍ JMP ● 条件付きジャンプ: ジャンプの実行は読み込まれた値によって異なります。	LD BVar1 JMPC Label1

FBD および LD エディターの操作

概要

ネットワークは、FBD および LD プログラミングの基本要素です。各ネットワークには、論理式または算術式、POU (ファンクション、プログラム、ファンクションブロック呼び出しなど)、ジャンプ、リターン命令を表示する構造体があります。

新しいオブジェクトを作成すると、エディターウィンドウに空のネットワークが 1 つ自動的に表示されます。

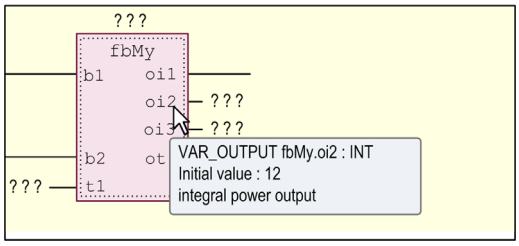
エディターの表示オプションについては、**オプション**ダイアログボックスの **FBD/LD/IL** カテゴリーの一般的なエディター設定を参照してください。

ツールチップ

ツールチップには、変数またはボックスパラメーターの情報が表示されます。

変数またはボックスパラメーターの名前にカーソルを置くと、ツールチップが表示されます。それぞれのタイプが表示されます。ファンクションブロックインスタンスの場合、スコープ、名前、データ型、初期値、およびコメントが表示されます。IEC 演算子 SEL, LIMIT および MUX の場合、入力の簡単な説明が表示されます。設定されている場合は、アドレスとシンボルのコメントだけでなく、オペランドのコメントも表示されます (2 行目に引用符付き)。

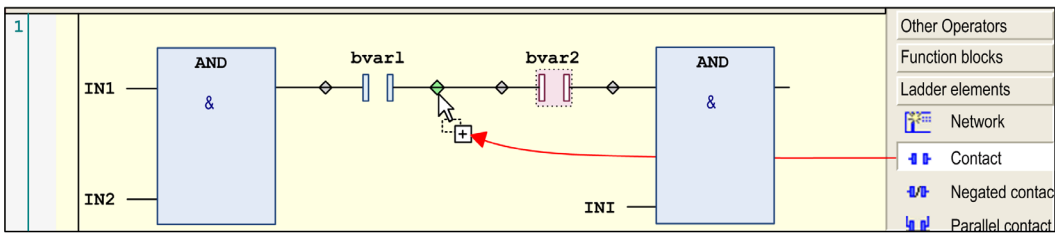
例 : POU 出力のツールチップ



要素の挿入、配置、および置換

- エディターで作業するためのコマンドは、初期設定で FBD/LD/IL メニュー (*252* ページ) にあります。頻繁に使用するコマンドは、コンテキストメニューにもあります。現在のカーソル位置または現在選択されている項目 (複数選択可、以下選択 (*242* ページ) を参照) によって、メニューコマンドから挿入できる要素は異なります。
- **ツールボックス** (*259* ページ) からエディターウィンドウへ、またはエディター内の位置から別の位置へマウスで要素をドラッグ (ドラッグ & ドロップ) することもできます。そのためには、マウスクリックでドラッグする要素を選択し、マウスボタンを押したまま要素をエディタービューの対応するネットワークにドラッグします。ネットワークに入るとすぐに、要素の各タイプを挿入できる場所すべてが灰色の位置マーカーで示されます。これらの位置にマウスカーソルを移動するとマーカーが緑色に変わり、マウスボタンを離すとその位置に要素が挿入されます。
- ファンクションブロック、演算子、またはネットワークを**ツールボックス** (*259* ページ) からエディターの左側の上矢印または下矢印にドラッグすると、既存要素の上または下に新しいネットワークが自動的に作成されます。
- 要素を別の要素と置き換えるには、その位置に他の要素を描画します。
- ボックスの入力接続または出力接続を再配置するには、ボックスで直接接続ピンを選択し、ドラッグ & ドロップでボックスの目的の位置に配置します。

LD エディターの挿入位置



- 要素を配置するために、**編集**メニューの切り取り、コピー、貼り付け、および削除コマンドを使用できます。ドラッグ & ドロップで要素をコピーすることもできます。マウスをクリックしてネットワーク内の要素を選択したら CTRL キーを押し、マウスボタンとキーを押したまま要素を目的の位置までドラッグします。位置までくると (緑色の位置マーカー) すぐに、カーソル記号にプラス記号が追加されます。次に、マウスボタンを離して要素を挿入します。
- 配置可能なカーソル位置については、*FBD、LD、および IL のカーソル位置* (*249* ページ) を参照してください。
- EN/ENO ボックスの挿入は、FBD および LD エディターでは多様に処理されます。
詳細については、**ボックスを挿入**コマンドの説明を参照してください (IL エディターは EN/ENO ボックスの挿入に対応していません)。

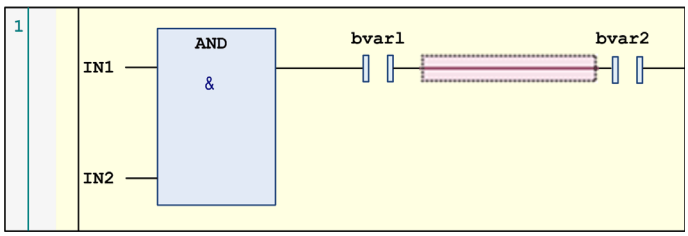
エディター内の移動

- 矢印キーを使用して、次または前のカーソル位置 (*249* ページ) にジャンプできます。これはネットワーク間でも可能です。← と → キーによる移動は、通常は左から右の信号の流れ、またはその逆の流れに従います。改行の位置では、次のカーソル位置は現在印が付いている位置に残る場合があります。↑ または ↓ キーを押すと、隣接要素が同じロジックグループ (例えば、ボックスのピン) 内の場合、現在の位置の上または下にある次の隣接要素にジャンプします。そのようなグループが存在しない場合は、上または下にある最も近い隣接要素にジャンプします。並列に接続された要素を通る移動は、第 1 の分岐に沿って実行されます。
- HOME キーを押すと、最初の要素にジャンプします。END キーを押すと、ネットワークの最初の要素にジャンプします。
- TAB キーを使用すると、ネットワーク内の次または前のカーソル位置 (*249* ページ) にジャンプします。
- CTRL + HOME キーを押すと、ドキュメントの先頭にスクロールし、最初のネットワークに印が付きます。
- CTRL + END キーを押すと、ドキュメントの最後にスクロールし、最後のネットワークに印が付きます。
- PAGE UP を押すと、1 画面上にスクロールし、最上部の長方形に印が付きます。
- PAGE DOWN を押すと、1 画面下にスクロールし、最上部の長方形に印が付きます。

選択

- マウスをクリックするか、矢印キーまたはタブキーを使用してそれぞれの位置にカーソルを移動させることで要素またはネットワークを選択できます。選択された要素は赤色の影付きで表示されます。*FBD、LD、および IL のカーソル位置* (*249* ページ) も参照してください。
- LD エディターでは、コマンドを実行するために要素の間の行も選択できます。例えば、その位置にさらに要素を挿入する場合などです。

LD エディターの行の選択

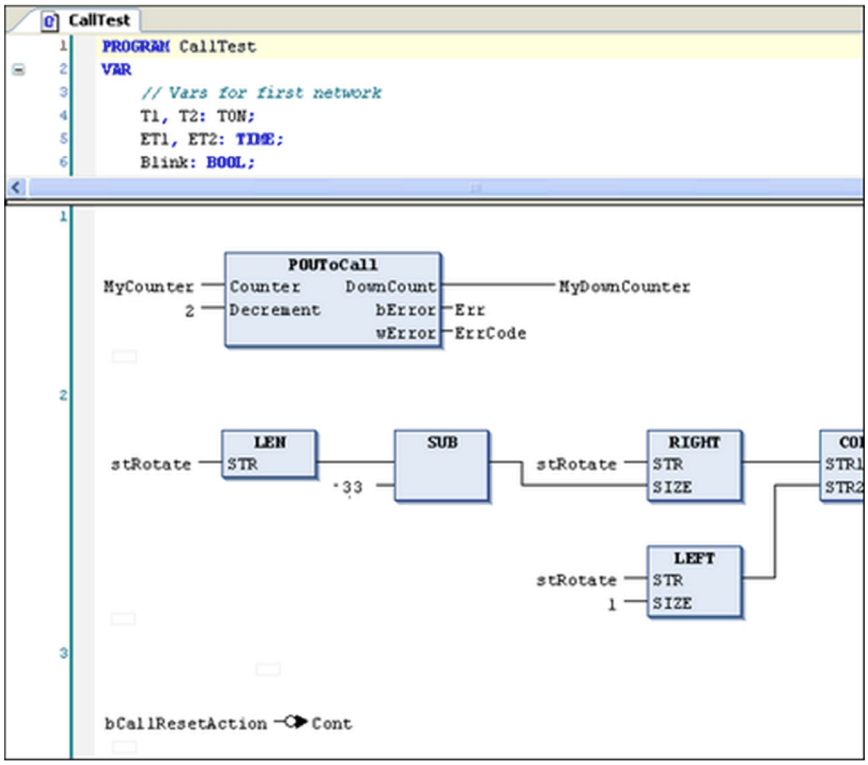


- 複数の要素またはネットワークを選択するには、CTRL キーを押しながら要素を順に選択します。

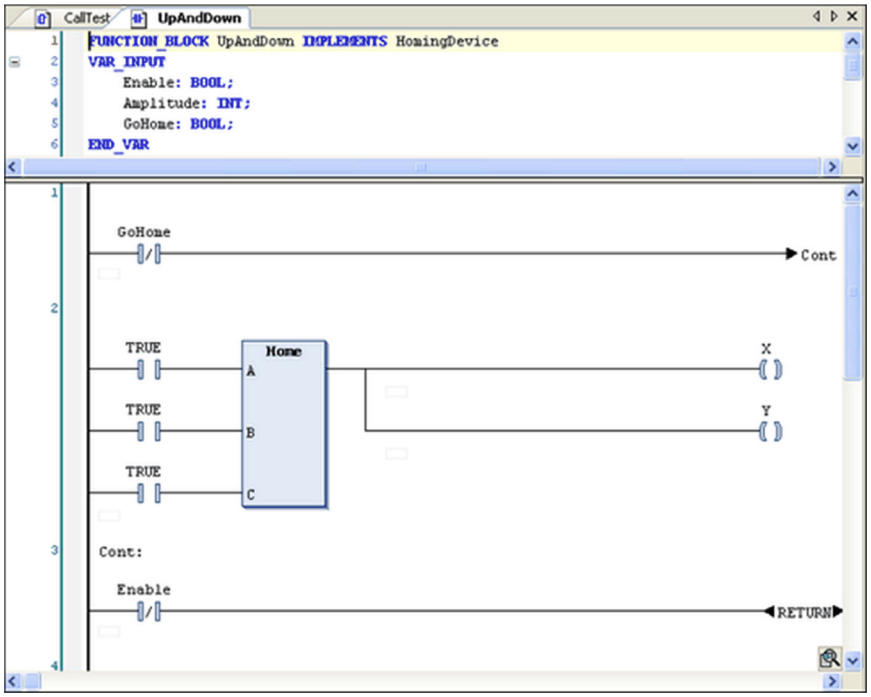
ファンクションブロックを開く

エディターにファンクションブロックが追加されている場合は、ダブルクリックでこのブロックを開くことができます。または、コンテキストメニューから**ブラウズ - 宣言へ移動**コマンドを使用することもできます。

FBD エディター



LD エディター



言語の詳細については、以下を参照してください。

- *ファンクションブロックダイアグラム - FBD* (*234* ページ)
- *ラダーダイアグラム - LD* (*235* ページ)

IL エディターの操作

概要

IL 命令リスト (*236* ページ) エディターは表形式のエディターです。FBD または LD プログラムのネットワーク構造も IL プログラムで表されます。基本的には 1 つの IL プログラムに 1 つのネットワーク (*260* ページ) で十分ですが、FBD、LD、および IL 間の切り替えを考慮して、IL プログラムを構築するネットワークも使用できます。

ツール → オプションダイアログボックス、カテゴリ **FBD、LD、および IL エディター** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*、**IL** タブで EcoStruxure Machine Expert の IL 記述言語を有効にします。

ツールチップ

ツールチップには、変数またはボックスパラメーターの情報が表示されます。

FBD および LD エディタービューの操作 (*241* ページ) を参照してください。

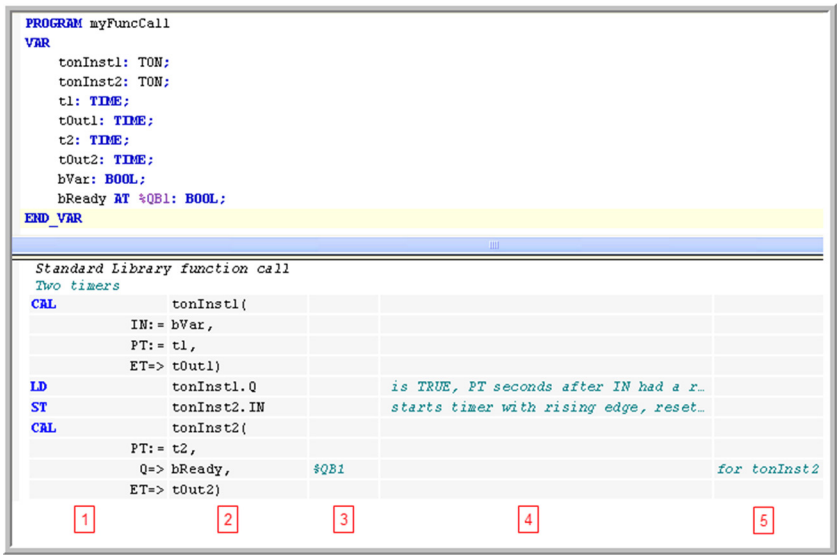
要素の挿入と配置

- エディターで作業するためのコマンドは、**FBD/LD/IL** メニューにあります。頻繁に使用するコマンドは、コンテキストメニューにもあります。
- 要素であるプログラム単位は、**FBD/LD/IL** メニューの**挿入**コマンドで現在のカーソル位置に挿入されます。
- **ネットワーク**要素を**ツールボックス** (*259* ページ) からエディター左側の上矢印または下矢印の上にドラッグすることができます。その後、既存要素の上または下に新しいネットワークが作成されます。
- 要素を配置するために、**編集**メニューの切り取り、コピー、貼り付け、および削除コマンドを使用できます。
- プログラミング言語命令リスト - IL (*236* ページ) に関する情報も参照してください。
- EN/ENO 機能をもつ演算子は、FBD および LD エディター内にのみ挿入できます。

この章では、表形式エディターの構造、エディターでの操作方法、複雑なオペランド、呼び出し、およびャンプの使用方法について説明します。

IL 表形式エディターの構造

IL 表形式エディター



表の行に各プログラム行が記述され、表の列によって内容が構造化されています。

列	内容	詳細
1	演算子	このフィールドには、IL 演算子 (LD、ST、CAL、AND、OR など) またはファンクション名が入ります。 ファンクションブロック呼び出しの場合は、ここで対応するパラメーターも指定します。プレフィックスフィールド := または => を入力します。 詳細については、*IL の修飾子と演算子* (*237* ページ) を参照してください。
2	オペランド	このフィールドには、ただ 1 つのオペランドまたはジャンプラベルが入ります。複数のオペランドが必要な場合 (複数の拡張可能な演算子 AND A、B、C、または複数のパラメーターをもつファンクション呼び出し) は、次の行に記述し演算子フィールドは空のままにします。この場合、パラメーターを区切るカンマを追加します。 ファンクションブロック、プログラム、またはアクション呼び出しの場合は、適切な開き括弧および閉じ括弧を追加します。
3	アドレス	このフィールドには、宣言部で定義したオペランドのアドレスが入ります。このフィールドは変更できません。オンまたはオフに切り替えるには、**シンボルアドレスの表示**オプションを使用します。
4	シンボルコメント	このフィールドには、宣言部でオペランドとして定義したコメントが入ります。このフィールドは変更できません。オンまたはオフに切り替えるには、**シンボルアドレスの表示**オプションを使用します。
5	オペランドコメント	このフィールドには、現在の行のコメントが入ります。編集可能で、**オペランドコメントを表示**オプションで表示 / 非表示を切り替えできます。

表内の移動

- UP および DOWN 矢印キー : 上または下のフィールドに移動します。
- TAB: 行内の右のフィールドに移動します。
- SHIFT + TAB: 行内の左のフィールドに移動します。
- SPACE: 現在選択しているフィールドを編集用に開きます。または、フィールドをさらにクリックします。入力アシスタントを利用できる場合、... ボタンから使用できます。現在開いている編集フィールドを閉じるには、ENTER キーを押して現在の入力を確定するか、ESC キーを押して入力をキャンセします。
- CTRL + ENTER: 現在の行の下に新しい行を入力します。
- DEL: 現在選択しているフィールドの行を削除します。
- **切り取り** , **コピー** , **貼り付け** : 1 行または複数行をコピーするには、1 行または複数行のフィールドを選択して、**コピー**コマンドを実行します。行を切り取るには、**切り取り**コマンドを使用します。**貼り付け**は、以前にコピー / 切り取りされた行を現在選択しているフィールドの行の前に挿入します。フィールドが選択されていない場合は、ネットワークの最後に挿入されます。
- CTRL + HOME により、ドキュメントの先頭にスクロールし、最初のネットワークに印が付きます。
- CTRL + END により、ドキュメントの最後にスクロールし、最後のネットワークに印が付きます。
- PAGE UP により、1 画面上にスクロールし、最上部の長方形に印が付きます。
- PAGE DOWN により、1 画面下にスクロールし、最上部の長方形に印が付きます。

複数オペランド (拡張演算子)

複数のオペランドで同じ演算子 (*237* ページ) を使用する場合、プログラミング方法が 2 つあります。

- 連続する行にオペランドをカンマで区切って入力します。例えば、次のようになります。

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

- 連続する行で命令を繰り返します。例えば、次のようになります。

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

複雑なオペランド

複雑なオペランドを使用する場合は、前に開き括弧を入力し、次の行に特定のオペランドの構成要素を入力します。その下の別の行に、閉じ括弧を入力します。

例 : 各サイクルごとに文字列を 1 文字づつ回転させます。

対応する ST コード :

```
stRotate := CONCAT(RIGHT(stRotate, (LEN(stRotate) - 1)), (LEFT(stRotate, 1)));
```

```
LD      stRotate
RIGHT(   stRotate
LEN
SUB     1
)
CONCAT(  stRotate
LEFT    1
)
ST      stRotate
```

ファンクションの呼び出し

演算子フィールドにファンクション名を入力します。前の LD 演算で、(最初の) 入力パラメーターをオペランドとして入力します。さらにパラメーターがある場合は、ファンクション名と同じ行に次のパラメーターを入力します。パラメーターはこの行にカンマで区切って追加するか、または次の行に追加できます。

ファンクションの戻り値はアキュムレーターに格納されます。IEC 規格に関する次の制限が適用されます。

注記 : 複数の戻り値をもつファンクションの呼び出しはできません。次に続く演算に使用できる戻り値は 1 つのみです。

例 : 3 つの入力パラメーターをもつファンクション GeomAverage を呼び出します。最初のパラメーターは、前の演算の X7 によって指定されます。2 番目パラメーター 25 は、ファンクション名で指定されます。3 番目のパラメーターは、同じ行または次の行の変数 tvar によって指定されます。戻り値は変数 Ave に代入されます。

対応する ST コード :

```
Ave := GeomAverage(X7, 25, tvar);
```

IL でのファンクション呼び出し

```
LD          X7
GeomAverage   25
            tvar
ST          Ave
```

ファンクションブロック呼び出しとプログラム呼び出し

CAL- または CALC 演算子 (*237* ページ) を使用します。オペランドフィールド (2 列目) にファンクションブロックのインスタンス名またはプログラム名を入力し、その後に開き括弧を入力します。続く行の 1 行に 1 つの入力パラメーターを入力します。

演算子フィールド : パラメーター名

プレフィックスフィールド :
- 入力パラメーターの場合 :=
- 出力パラメーターの場合 =>

オペランドフィールド : 現在のパラメーター

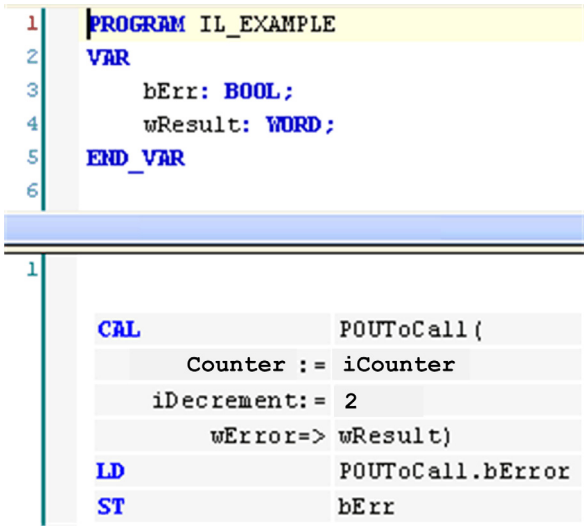
ポストフィックスフィールド :
- さらにパラメーターが続く場合 ,
  最後のパラメーターの後に)
- パラメーターのない呼び出しの場合 ()

例 : 2 つの入力パラメーターと 2 つの出力パラメーターのある呼び出し POUToCAll

対応する ST コード :

```
POUToCall(Counter := iCounter, iDecrement:=2, bError=>bErr, wError=>wResult);
```

IL の入力および出力パラメーター付きプログラムの呼び出し



ファンクションブロックまたはプログラムのすべてのパラメーターを使用する必要はありません。

注記：複雑な式は使用できません。呼び出し命令の前にファンクションブロックまたはプログラムの入力に割り当ててください。

アクションの呼び出し

ファンクションブロックまたはプログラムの呼び出しのように実行するには、アクション名をインスタンス名またはプログラム名に追加します。

例 : アクション ResetAction の呼び出し。

対応する ST コード :

```
Inst.ResetAction();
```

IL でのアクション呼び出し

```
CAL      Inst.ResetAction()
```

メソッドの呼び出し

ファンクションの呼び出しのように実行するには、追加されたメソッド名付きのインスタンス名を最初の列 (演算子) に入力します。

例 : メソッド Home の呼び出し。

対応する ST コード :

```
Z := IHome.Home(TRUE, TRUE, TRUE);
```

IL でのメソッドの呼び出し

```
LD          TRUE
IHome.Home    TRUE
            TRUE
ST          Z
```

ジャンプ

ジャンプ (*263* ページ) は、最初の列 (演算子) に JMP、2 番目の列 (オペランド) にラベル名を入れてプログラミングします。ラベルは、ジャンプ先のネットワークのラベル (*264* ページ) フィールドで定義されます。

無条件ジャンプの前の命令文リストは、次のコマンドのいずれかで終了してください。ST、STN、S、R、CAL、RET、または別の JMP。

これは、条件付きジャンプ (*263* ページ) の場合ではありません。ジャンプの実行は読み込まれた値によって異なります。

例 : 条件付きジャンプ命令 : bCallRestAction が TRUE の場合、プログラムは Cont というラベルのネットワークにジャンプします。

IL における条件付きジャンプ命令 :

```
LDN     bCallResetAction
JMPC    Cont
```

FBD、LD、および IL のカーソル位置

IL エディター

これは、表形式で構成されたテキストエディターです。表の各セルにカーソルを置くことができます。*IL エディタービューの操作* (*244* ページ) を参照してください。

FBD および LD エディター

グラフィックエディターです。下の例 (1) ~ (15) に、テキスト、入力、出力、ボックス、接触、コイル、リターン、ジャンプ、要素とネットワークの間の線を配置可能なカーソル位置を示します。

切り取り、コピー、貼り付け、削除、およびその他のエディター固有のコマンドなどのアクションは、現在のカーソル位置または選択した要素を対象にします。*FBD および LD エディタービューの操作* (*241* ページ) を参照してください。

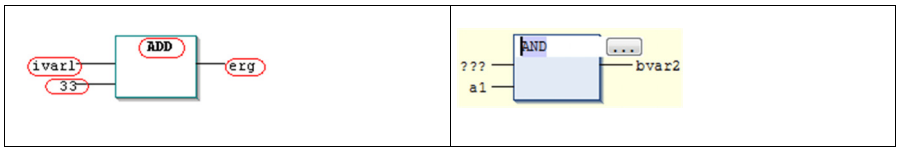
基本的に FBD では、各要素の周りの点線の四角形が現在のカーソル位置を示します。さらに、テキストおよびボックスは青または赤の影付きになります。

LD では、カーソルをコイルおよび接点の上に置くと赤色になります。

カーソル位置によって、コンテキストメニューから挿入 (*241* ページ) できる要素が決まります。

カーソルの配置可能位置

(1) すべてのテキストフィールド



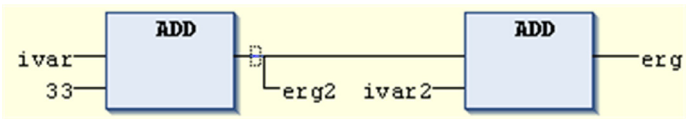
左の図では、カーソルを置くことができる位置が赤枠で記されています。右の図には、ボックスが表示されカーソルは AND フィールドに置かれています。**FBD、LD および IL エディターのオプション**ダイアログボックスで設定されている場合は、変数名の代わりにアドレスが入力される可能性があることに留意してください。

(2) すべての入力

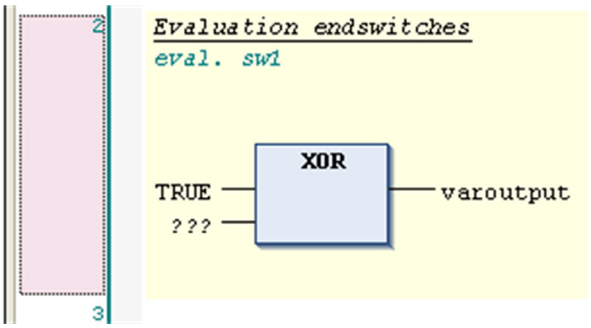
(3) すべての演算子、ファンクション、またはファンクションブロック

(4) 後から代入またはジャンプが来る場合の出力

(5) ジャンプまたはリターン命令の前の代入の上で交差する線の直前

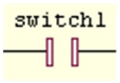


(6) ネットワーク内の最も右のカーソル位置、またはネットワーク内の他のカーソル位置以外の場所。これはネットワーク全体を選択します。



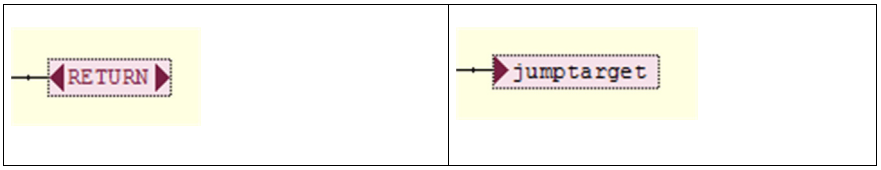
(7) 代入の直前の線の交差位置

(8) すべての接点



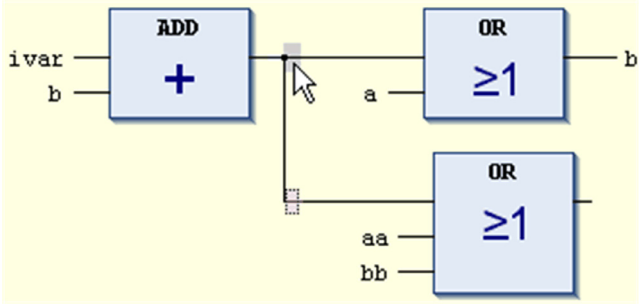
(9) すべてのコイル

(10) すべてのリターンとジャンプ

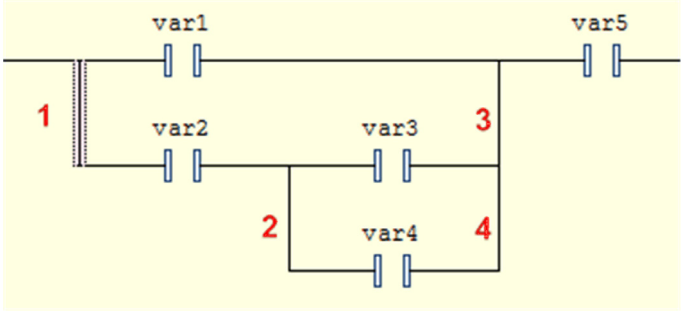


(11) 接点とコイルとの接続線

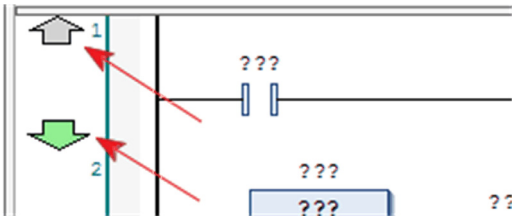
(12) ネットワーク内の分岐またはサブネットワーク



(13) 並列接点間の接続線 (位置 1 ~ 4)

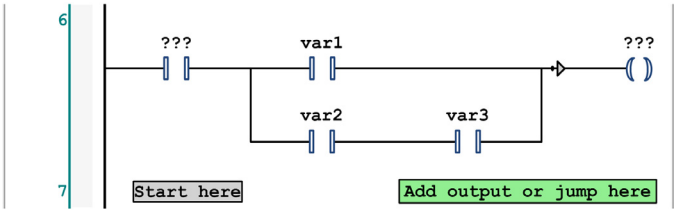


(14) ネットワークの前後



エディターの一番左側に新しいネットワークを追加できます。既存のネットワークの前に新しいネットワークを挿入できるのは、ネットワーク 1 の前のみです。

(15) ネットワークの開始または終了



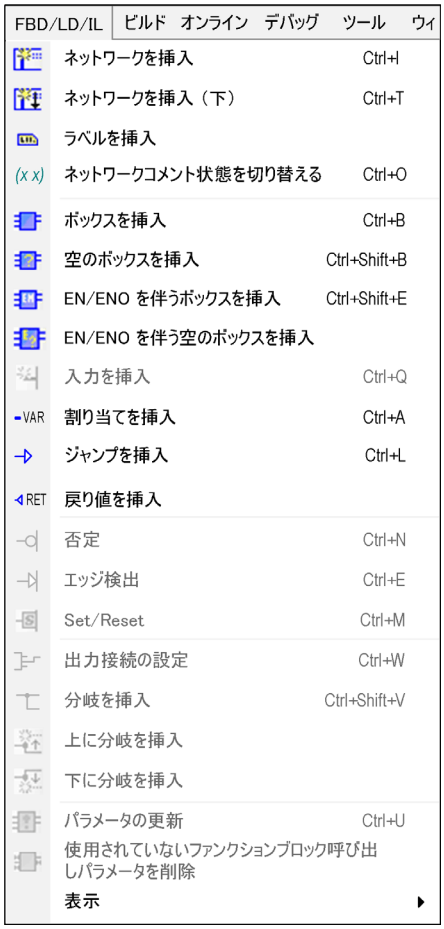
ネットワークの最初の**ここから開始**フィールドに接点およびファンクションブロックを追加できます。ネットワークの最後の**ここに出力またはジャンプを追加**フィールドに要素、リターン、ジャンプ、およびコイルを追加できます。

FBD/LD/IL メニュー

概要

カーソルを FBD/LD/IL エディター (*233* ページ) ウィンドウに置くと、メニューバーの FBD/LD/IL メニューに現在設定されているエディタービューでプログラミングするためのコマンドが表示されます。

FBD エディタービューの FBD/LD/IL メニュー



- コマンドの説明については、*FBD/LD/IL エディターコマンド*の参照してください。
- メニューの設定については、**カスタマイズメニュー**の説明を参照してください。

オンラインモードの FBD/LD/IL エディター

概要

オンラインモードでは、FBD/LD/IL エディターに監視 (*253* ページ) 用およびコントローラーの値と式の書き込みおよび強制用のビューが表示されます。

デバッグ機能 (ブレークポイント、ステップなど) を使用できます。*ブレークポイントまたは停止位置* (*256* ページ) を参照してください。

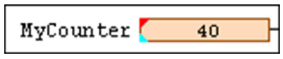
- オンラインモードでのオブジェクトの開き方については、*オンラインモードのユーザーインターフェイス* (*43* ページ) の章を参照してください。
- FBD、LD または IL オブジェクトのエディターウィンドウには、上部に**宣言エディター**が含まれています。また、*オンラインモードの宣言エディター* (*346* ページ) の章も参照してください。

監視

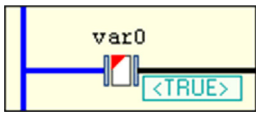
オプションダイアログボックスで明示的にインライン監視を無効にしていない場合、FBD または LD エディターには変数の後ろに小さな監視ウィンドウが表示されるか、実際の値が表示される追加の監視列 (インライン監視) が表示されます。割り当てられていないファンクションブロックの入力および出力の場合でも同様です。

変数が現在強制 (*255* ページ) されている場合は変数のインライン監視ウィンドウの左上の角に小さな赤い三角形が表示され、変数が現在書き込みまたは強制の準備ができている場合は左下の角に青い三角形が表示されます。LD では、接点およびコイルの場合、現在の設定済みの値 (TRUE または FALSE) が要素のすぐ下に表示されます。

現在強制されていて、強制解除の準備ができている変数の例



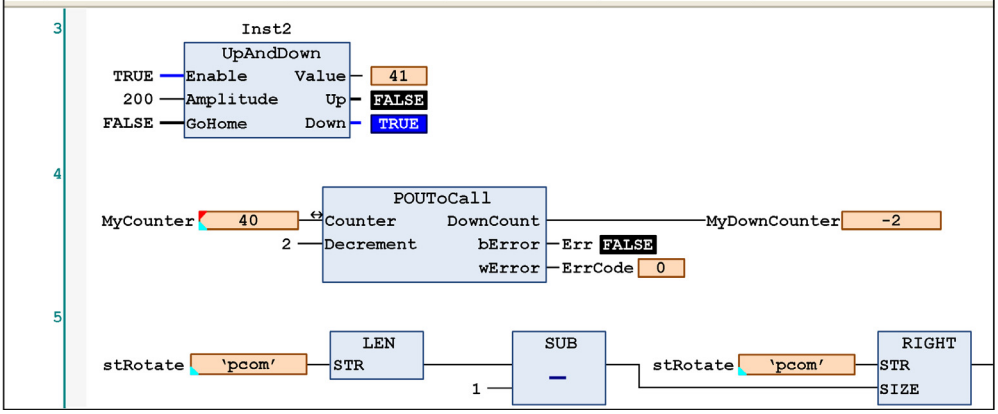
現在値 TRUE で書き込みまたは強制するように準備されている接点変数の例



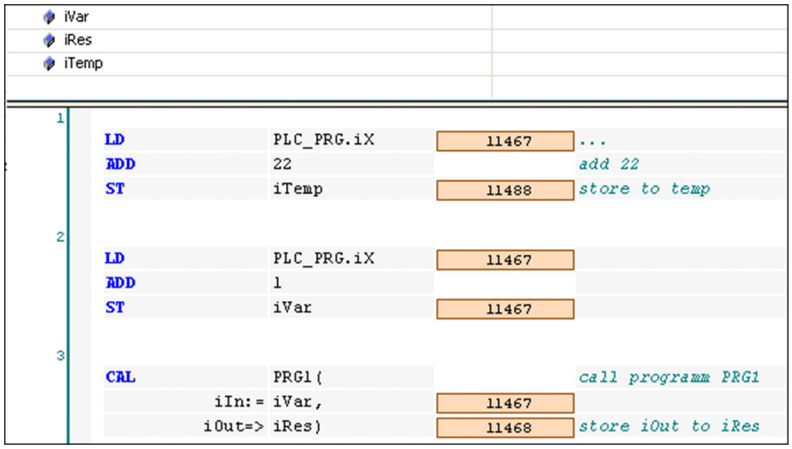
FBD プログラムのオンラインビュー

SP_for_Win32.App.CallTest

式	コメント	タイプ	値	設定済みの値
Inst2		UpAndDown		
Enable		BOOL	TRUE	
Amplitude		INT	200	
GoHome		BOOL	FALSE	
Value		INT	41	
Up		BOOL	FALSE	
Down		BOOL	TRUE	
Counter		DINT	2159	
diValue		DINT	41	
X		BOOL	FALSE	
X		INT	0	
X2		INT	0	
Up		BOOL	TRUE	
Down		BOOL	FALSE	
MyCounter	Vars for third network	DINT	F 40	<Unforce and restore>
MyDownCounter		DINT	-2	
Err		BOOL	FALSE	
ErrCode		WORD	0	
stRotate	Vars for fifth network	STRING	'pcom'	'abc'
Ave	Vars for fifth network	DINT	38	



IL プログラムのオンラインビュー

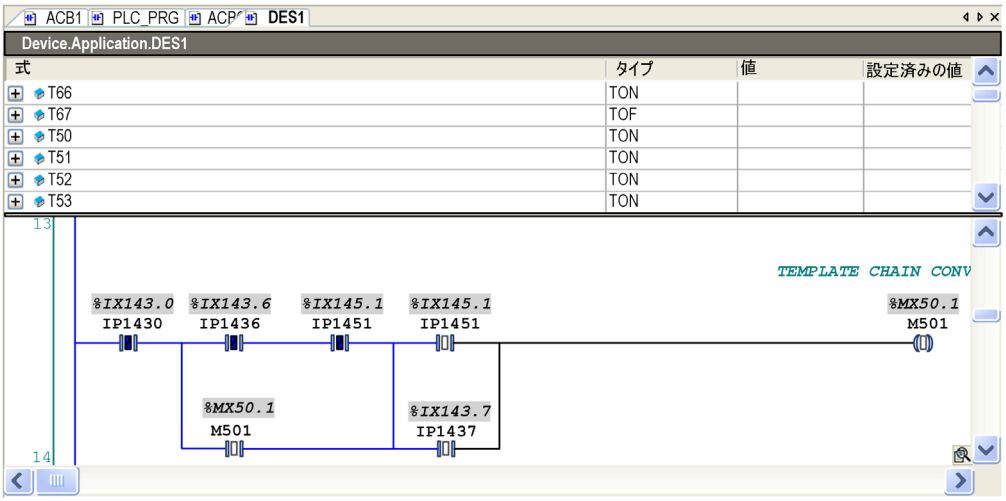


オンラインビューでは、ラダーネットワークは接続がアニメーション化されています。

- 値が TRUE の接続は青色の太字で表示されます。
- 値が FALSE の接続は黒色の太字で表示されます。
- 値が不明な接続またはアナログ値の接続は、標準の線画 (太字ではない黒い線) で表示されます。

接続の値は、監視値から計算されます。

LD プログラムのオンラインビュー

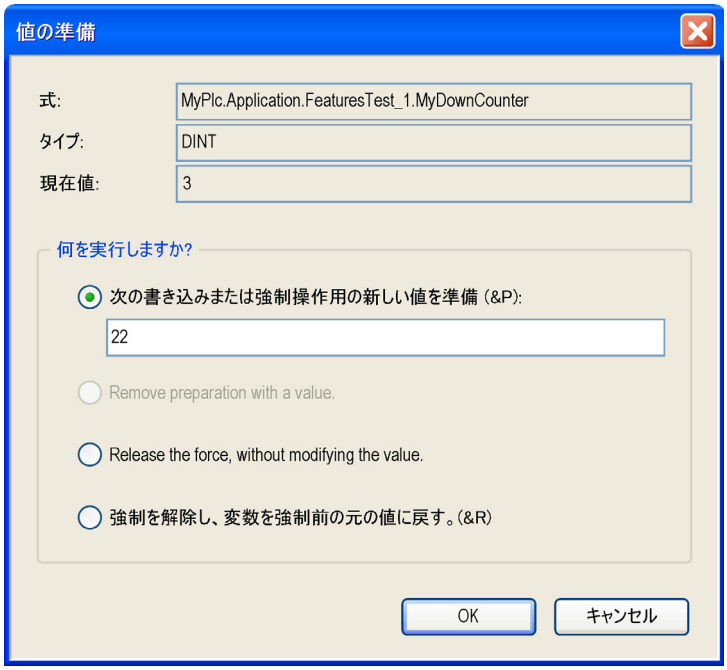


ファンクションを開くには、ダブルクリックするかコンテキストメニューから**ブラウズ - 宣言へ移動**コマンドを実行します。詳細については、*オンラインモードのユーザーインターフェイス* (*43* ページ) の説明を参照してください。

変数の強制 / 書き込み

オンラインモードでは、宣言エディター内 (*346* ページ) またはエディター内で変数の強制または書き込みするための値を準備できます。エディターで変数をダブルクリックすると、次のダイアログボックスが開きます。

値の準備ダイアログボックス



変数の名前とデバイスツリー (**式**) 内の絶対パス、タイプ、および現在値が表示されます。対応する項目を有効にすると、以下を実行できます。

- 編集フィールドに入力する新しい値を準備。
- 設定済みの値を削除。
- 強制された変数を解除。
- 強制されている変数を解除し、強制前に割り当てられた値にリセット。

選択したアクションは、メニューコマンド**値の強制** (**オンライン**メニュー) コマンドを実行するか、F7 を押すと実行されます。

変数 (強制または設定済みの値) の現在の状態が、ネットワークの対応する要素でどのように表示されるかについは、*監視* (*253* ページ) の章を参照してください。

ブレークポイントまたは停止位置

デバッグ用にブレークポイント (停止位置) を設定できる位置は、変数の値が変わる位置 (命令)、プログラムフローが分岐する位置、または別の POU が呼び出される位置です。

以下が、その位置です。

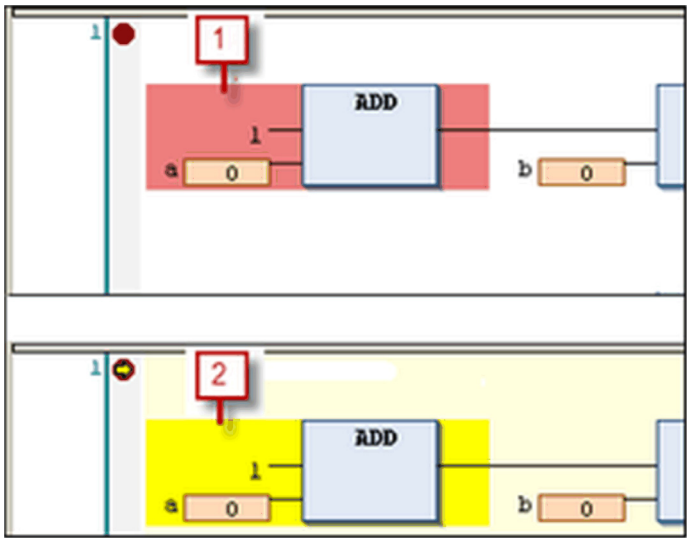
- ネットワーク全体で、ブレークポイントはネットワーク内の最初に配置可能な位置に適用されます。
- ボックス (*265* ページ) の上。ここに命令が含まれている場合。従って、ADD, DIV のような演算子ボックスの上には配置できません。下記の注記を参照してください。
- 代入の上。
- 呼び出し元に戻る時点の POU の終了位置。そのためにオンラインモードでは、空のネットワークが自動的に表示されます。ネットワーク番号の代わりに、RET で識別されます。

注記： ブレークポイントは、ネットワークの最初のボックスには直接設定できません。ただし、ネットワーク全体にブレークポイントが設定されている場合、停止位置は自動的に最初のボックスに適用されます。

現在の設置可能な位置については、**表示 → ブレークポイント**ダイアログボックスの選択リストを参照してください。

有効なブレークポイント位置を含むネットワークは、ネットワーク番号の右にブレークポイントシンボル (赤色に塗りつぶされた円) が付き、ネットワーク内の配置可能なブレークポイント位置には赤色の影付き長方形の背景が付きます。無効なブレークポイント位置には、塗りつぶしなしの赤色の円または長方形で囲まれて表示されます。

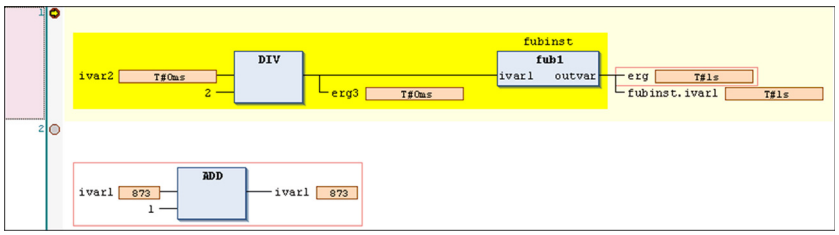
設定されたブレークポイントおよび到達したブレークポイント



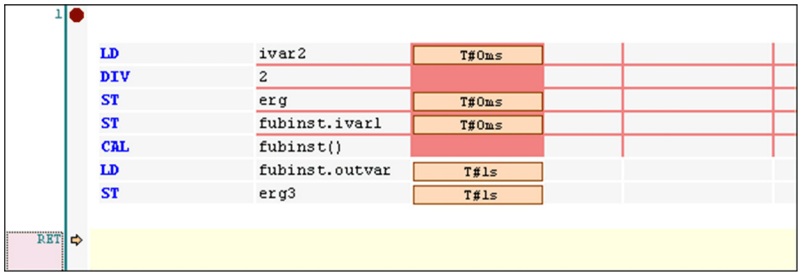
1 設定されたブレークポイント
2 到達したブレークポイント

ステップまたはプログラム処理中にブレークポイントに到達するとすぐにブレークポイントのシンボルに黄色の矢印が表示され、赤色の影付き領域が黄色に変わります。

FBD に表示された停止位置



IL に表示された停止位置



注記：ブレークポイントは、呼び出される可能性があるすべてのメソッドに自動的に設定されます。インターフェイス管理メソッドが呼び出されると、そのインターフェイスを実装するファンクションブロックのすべてのメソッドと、メソッドをサブスクライブするすべての派生ファンクションブロックにブレークポイントが設定されます。ファンクションブロックのポインターによってメソッドが呼び出される場合、ブレークポイントはファンクションブロックのメソッドとメソッドをサブスクライブしているすべての派生ファンクションブロックに設定されます。

11.2
FBD/LD/IL 要素

このセクションについて

このセクションには次の項目が含まれています。

FBD/LD/IL ツールボックス

概要

FBD/LD/IL エディター (*233* ページ) には、ドラッグ & ドロップでエディターウィンドウに挿入できるプログラミング要素が表示されるツールボックスがあります。**表示**メニューの**ツールボックス**コマンドを実行してツールボックスを開きます。

挿入できる要素は現在有効なエディタービューによって異なります (各挿入コマンドの説明を参照してください)。

要素はカテゴリー順に並んでいます。**General** (**Network**、**Assignment** などのような一般的な要素)、**Boolean operator**、**Math operators**、**Other operators** (例えば、SEL, MUX, LIMIT および MOVE)、**Function blocks** (例えば、R_TRIG, F_TRIG, RS, SR, TON, TOF, CTD, CTU)、**Ladder elements**、および **POUs** (ユーザー定義)。

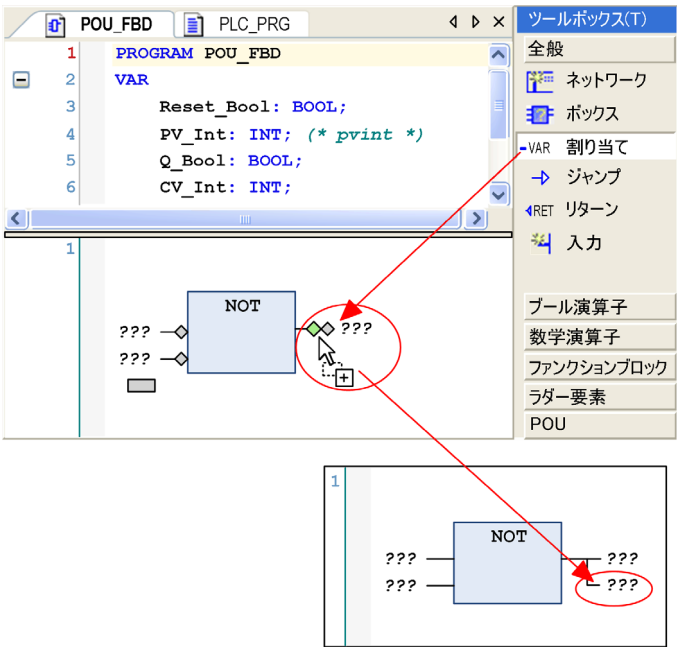
POU カテゴリーには、エディターで開いている FBD/LD/IL オブジェクトと同じアプリケーションの下に定義されているすべての POU がリスト表示されます。POU がプロパティのビットマップに割り当てられている場合は、POU 名の前に表示されます。それ以外の場合は、POU のタイプを示す標準アイコンが使用されます。POU がアプリケーションに追加または削除されると、リストは自動的に更新されます。

Other operators カテゴリーには、SEL, MUX, LIMIT、および MOVE 演算子間の変換処理プレースホルダー要素が含まれています。この要素はネットワークの適切な位置にドラッグ & ドロップできます。変換タイプは、挿入位置で必要なタイプに応じて自動的に設定されます。ただし、状況によっては必要な変換タイプが自動的に決定されません。その場合は、要素を手動で変更します。

カテゴリーフォルダーを開くには、各カテゴリー名が表示されているボタンをクリックします。次の図を参照してください。**一般**カテゴリーが展開され、その他は折りたたまれます。図は、ツールボックスから **Assignment** 要素をドラッグ & ドロップで挿入する例を示しています。

ツールボックスの **General** セクションのみが開いています。

ツールボックスから挿入



エディターに要素を挿入するには、マウスでツールボックスの要素をクリックし、ドラッグ & ドロップでエディターウィンドウに移動させます。挿入可能な位置は、要素を描画している間 (エディターウィンドウ上でマウスボタンを押したままの状態) に表示される位置マーカーで示されます。最も近い挿入可能位置が緑色に点灯します。マウスボタンを離すと、緑色の位置に要素が挿入されます。

ボックス要素を既存のボックス要素にドラッグすると、古い要素が新しい要素に置き換わります。入力および出力がすでに割り当てられている場合、それらは定義されたまま残りますが、新しい要素ボックスには接続されません。

FBD/LD/IL のネットワーク

概要

ネットワークは、FBD (*234* ページ) または LD (*235* ページ) プログラムの基本要素です。FBD/LD エディターでは、ネットワークは垂直なリストに配置されます。各ネットワークは、左側にシリアルネットワーク番号で指定され、論理式または算術式、プログラム、ファンクションまたはファンクションブロックの呼び出し、場合によりジャンプまたはリターン命令で構成されます。

IL エディター (*236* ページ) も、FBD および LD エディターと共通なエディターを基にしているため、ネットワーク要素を使用します。オブジェクトが最初に FBD または LD でプログラミングされてから IL に変換された場合、ネットワークは IL プログラムにもまだ存在します。逆に、IL でオブジェクトのプログラミングを始める場合は、少なくとも 1 つはネットワーク要素が必要であり、そこにすべての命令が含まれる場合があります。ただし、プログラムを構築するためにネットワークを追加することもできます。

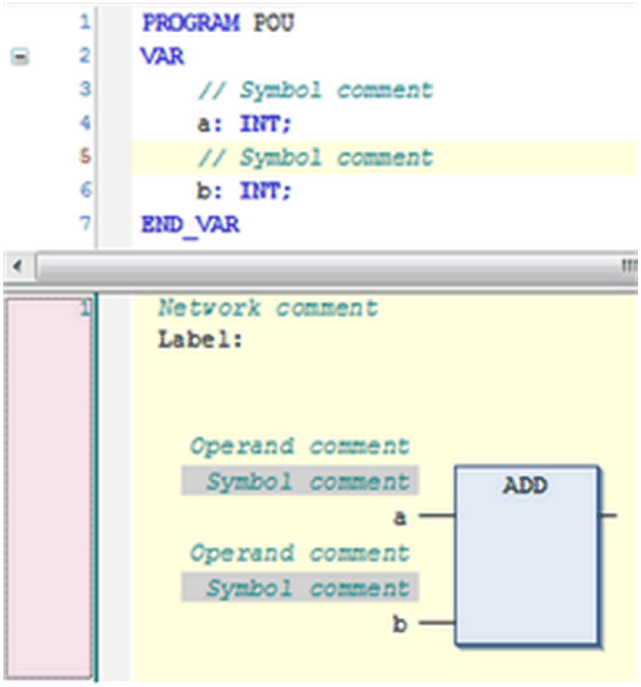
オプションとして、ネットワークにタイトル、コメント、およびラベル (*264* ページ) を割り当てることができます。

FBD、LD、および IL エディターのオプションダイアログボックスで、タイトル、コメントフィールド、およびネットワークセパレーターの表示を切り替えできます。このオプションが有効な場合は、上部境界線のすぐ下のネットワークをクリックすると、タイトルの編集フィールドが開きます。コメントを入力するには、それに応じてタイトルフィールドのすぐ下の編集フィールドを開きます。コメントは複数行にできます。改行するには ENTER を押します。コメントテキストの入力を終了するには CTRL + ENTER を押します。

エディターにネットワークコメントを表示するか、またどのように表示するかは、FBD、LD、および IL エディターのオプションダイアログボックスで設定します。

ジャンプ (*263* ページ) によって処理されるラベル (*264* ページ) を追加するには、**ラベルを挿入**コマンドを使用します。ラベルが定義されている場合はタイトルとコメントフィールドの下に表示されます。それらがない場合は、ネットワークの上部境界線のすぐ下に表示されます。

ネットワークのコメントとラベル



ネットワークをコメント状態として設定できます。ネットワークは処理されずにコメントとして表示および処理されます。これを実現するには、**ネットワークコメント状態を切り替える**コマンドを使用します。

現在選択しているネットワーク (カーソル位置 6 (*249* ページ)) で、コピー、切り取り、挿入、および削除のデフォルトコマンドを使用できます。

注記：タイトル、コメント、またはラベルを右クリック (カーソル位置 6 (*249* ページ)) すると、ネットワーク全体ではなく項目だけが選択されます。デフォルトコマンドの実行はネットワークには影響しません。

ネットワークを挿入するには、**ネットワークを挿入**コマンドを使用するか、ツールボックス (*259* ページ) からドラッグします。エディター内でドラッグ & ドロップすることで、ネットワークに含まれるすべての要素と一緒にネットワークをコピーまたは移動 (*241* ページ) できます。

分岐を挿入してサブネットワーク (*267* ページ) を作成することもできます。

RET ネットワーク

オンラインモードでは、既存のネットワークの下に追加で空のネットワークが表示されます。ネットワーク番号の代わりに RET で識別されます。

これは、実行が POU の呼び出し元に戻る位置を表し、可能な停止位置 (*253* ページ) が表示されます。

FBD/LD/IL の代入

概要

FBD または LD で選択したカーソル位置 (*249* ページ) に応じて、選択した入力 (カーソル位置 2 (*249* ページ)) の前に直接、または選択した出力 (カーソル位置 4 (*249* ページ)) の後に直接、またはネットワークの終わり (カーソル位置 6 (*249* ページ)) に代入を挿入できます。LD ネットワークでは、代入はコイル (*277* ページ) として表示されます。または、ツールボックス (*259* ページ) から代入要素をドラッグするか、エディタービュー内の別の位置からドラッグ & ドロップでコピーまたは移動 (*241* ページ) させます。

挿入後、テキスト文字列 ??? を割り当てる変数の名前で置き換えることができます。そのためには、**...** ボタンを使用して**入力アシスタント**を開きます。

IL (*236* ページ) では、代入は LD および ST 命令でプログラミングします。*IL の修飾子と演算子* (*237* ページ) を参照してください。

FBD/LD/IL のジャンプ

概要

FBD (*234* ページ) または LD (*235* ページ) で選択したカーソル位置 (*249* ページ) に応じて、選択した入力 (カーソル位置 2) の前に直接、または選択した出力 (カーソル位置 4) の後に直接、またはネットワークの終わり (カーソル位置 6) にジャンプを挿入できます。または、ツールボックス (*259* ページ) からジャンプ要素をドラッグするか、エディター内の別の位置からドラッグ & ドロップでコピーまたは移動 (*241* ページ) させます。

挿入後、自動的に入力された ??? をジャンプに割り当てるラベルに置き換えます。

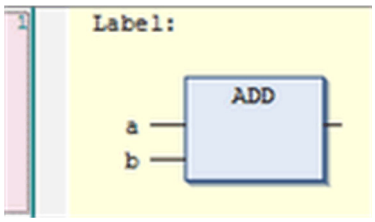
IL (*236* ページ) では、ジャンプは JMP 命令で挿入します。本書の IL の演算子と修飾子 (*237* ページ) の説明を参照してください。

FBD/LD/IL のラベル

概要

ネットワークコメントフィールドの下の各 FBD (*234* ページ)、LD (*235* ページ)、または IL ネットワークには、ラベルを定義するためのテキスト入力フィールドがあります。ラベルはネットワークのオプションの識別子で、ジャンプ (*263* ページ) を設定するときに指定できます。任意の文字列で設定できます。

ネットワークのラベルの位置



コメントおよびタイトルの表示を定義するには、**ツール → オプション → FBD、LD、および IL エディター**ダイアログボックスを参照してください。

FBD/LD/IL のボックス

概要

FBD (*234* ページ)、LD (*235* ページ)、または IL (*236* ページ) ネットワークに挿入できるボックスは複雑な要素であり、タイマー、カウンター、算術演算のような追加機能、またはプログラム、IEC ファンクションおよび IEC ファンクションブロックを表します。

ボックスは 1 点または複数の入力および出力をもつことができ、ライブラリーから提供されるか、またはユーザーがプログラミングすることもできます。

attribute pingroup を使用して、宣言内のファンクションブロックの入力と出力をグループ化できます。FBD または LD エディターでのファンクションブロックの表示で、グループを徐々に非表示にしたり、徐々に表示させたりできます。詳細については、*属性のピングループ*の章を参照してください (*548* ページ)。

ボックスに対応するモジュールがあり、**Show box icon** オプションが有効になっている場合は、ボックス内にアイコンが表示されます。

FBD、LD での使用

ボックスを挿入、**空のボックスを挿入**コマンドを使用して、LD ネットワークまたは FBD ネットワークにボックスを配置できます。または、ツールボックス (*259* ページ) から挿入するか、ドラッグ & ドロップでエディター内にコピーまたは移動できます。詳細については、**ボックスを挿入**コマンドの説明を参照してください。

IL での使用

IL (*236* ページ) プログラムでは、ボックス要素を表すためにパラメーター付き CAL (*237* ページ) 命令が挿入されます。

ボックスのパラメーター (入力、出力) を更新するには (ボックスのインターフェイスを変更した場合)、**Update parameters** コマンドを実行することで、現在の実装のままボックスを再挿入することなく更新できます。

FBD/LD/IL の RETURN 命令

概要

RETURN 命令で、FBD (*234* ページ)、LD (*235* ページ)、または IL (*236* ページ) POU を終了できます。

FBD または LD ネットワークでは、並列または前の要素の最後に配置できます。RETURN の入力が TRUE の場合、POU の処理はすぐに終了します。

戻り値を挿入コマンドを実行して RETURN 命令を挿入します。または、ツールボックス (*259* ページ) から要素をドラッグするか、エディター内の別の位置からコピーまたは移動 (*241* ページ) させます。

RETURN 要素

IL では、同じ用途で RET (*237* ページ) 命令を使用します。

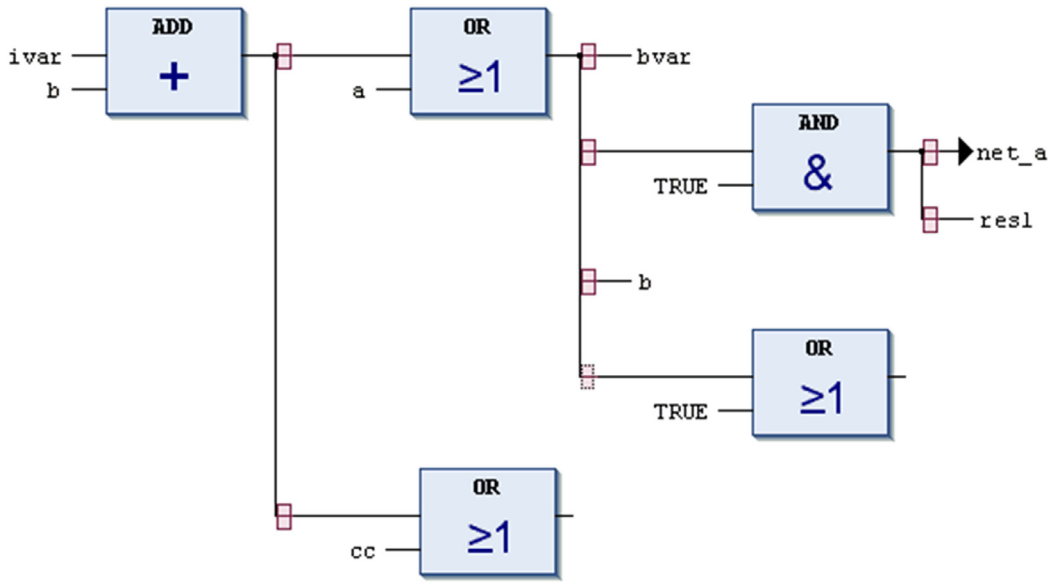
FBD/LD/IL の分岐 / 吊りコイル

概要

ファンクションブロックダイアグラム (*234* ページ) またはラダーダイアグラム (*235* ページ) ネットワークでは、分岐または吊りコイルによって現在のカーソル位置から処理線を分割します。処理線は 2 つのサブネットワークに継続され、上から下に 1 つずつ実行されます。各サブネットワークはさらに分岐できるので、ネットワーク内で複数の分岐ができます。

各サブネットワークには、独自のマーク (縦の長方形の記号) が付きます。この分岐でアクションを実行するには、そのマーク (カーソル位置 11 (*249* ページ)) を選択します。

分岐マーク

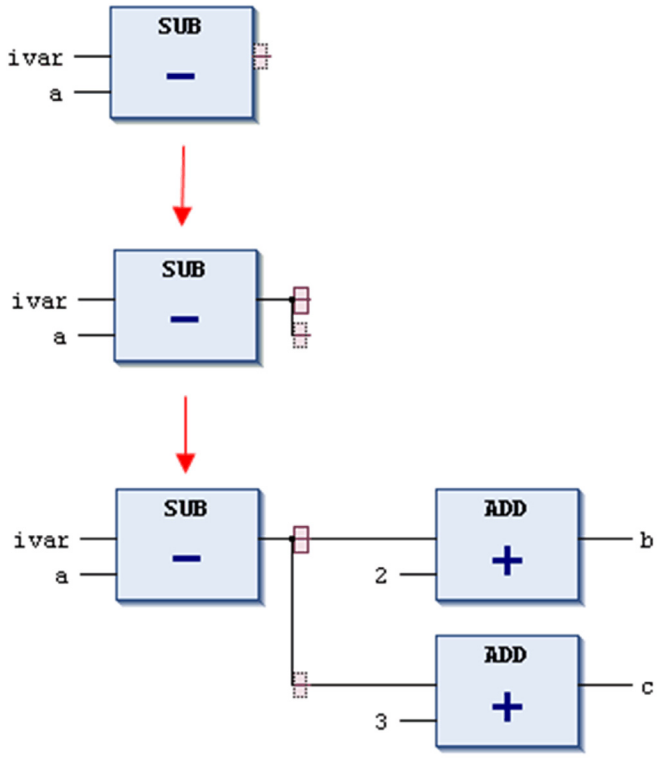


FBD では、**分岐の挿入**コマンドによって分岐を挿入します。または、ツールボックス (*259* ページ) から要素をドラッグします。挿入可能な位置については、**分岐の挿入** コマンドの説明を参照してください。

注記：サブネットワークでは、切り取りおよび貼り付け機能は実装されていません。

下の例では、SUB ボックス出力に分岐が挿入されています。これにより 2 つのサブネットワークが作成され、サブネットマーカーで選択できます。その後、各サブネットワークに ADD ボックスが追加されます。

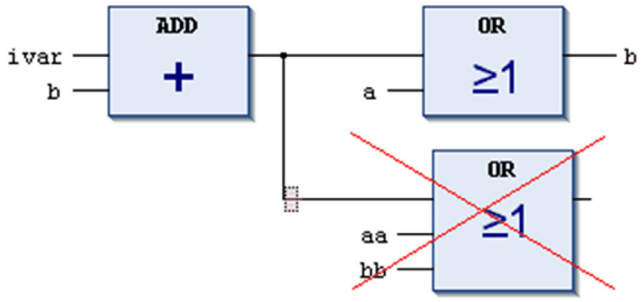
FBD でのネットワークと挿入された分岐



サブネットワークを削除するには、最初にサブネットワークのすべての要素、つまりサブネットワークのマークの右側に配置されたすべての要素を削除します。その後マークを選択し、標準の**削除**コマンドを実行するか DEL キーを押します。

次の図では、下位サブネットワークのマークを選択して削除する前に、入力 3 点の OR 要素を削除する必要があります。

分岐またはサブネットワークの削除



オンラインモードでの実行

特定の分岐は、左から右、その後に上から下に実行されます。

IL (命令リスト)

IL (*236* ページ) は、分岐のあるネットワークに対応していません。元の表示のままになります。

並列分岐

ラダーネットワークでは、並列分岐を使用して並列分岐 (*269* ページ) 評価を設定できます。

開いている分岐 (接合点なし) とは対照的に、並列分岐は閉じています。共通の分岐点と接合点があります。

並列分岐

概要

並列分岐により、論理要素の並列評価を実装できます。これは、短絡評価 (SCE) で記述される方法で実行されます。特定の並列条件が TRUE と評価された場合、ブール値出力をもつファンクションブロックの実行は回避されます。条件は、LD エディターでファンクションブロック分岐への並列分岐で表すことができます。SCE 条件はこの並列または連続に接続された分岐内の 1 つまたは複数の接点によって定義されます。

並列に実行される短絡評価分岐の垂直接続は、単一の線で表される OR 構造と区別するために、二重線で表されます (*ラダーネットワーク における SCE の並列分岐* を参照してください)。

注記：分岐という用語は、信号フローを分割する要素にも使用されます。並列分岐とは対照的に、この分岐 (*267* ページ) には接合点がありません。

並列分岐は、次のように動作します。まず、ファンクションブロックを含まない分岐が解析されます。そのような分岐の 1 つが TRUE と評価された場合、並列分岐のファンクションブロックは呼び出されず、ファンクションブロック分岐の入力の値が出力に渡されます。SCE 条件が FALSE と評価された場合、ファンクションブロックが呼び出され、ファンクションブロック実行呼び出しのブール型の結果が渡されます。

すべての分岐にファンクションブロックが含まれている場合は上から下へ順番に評価され、それらの出力が論理和演算で組み合わされます。ファンクションブロック呼び出しを含む分岐がない場合、通常の OR 演算が実行されます。

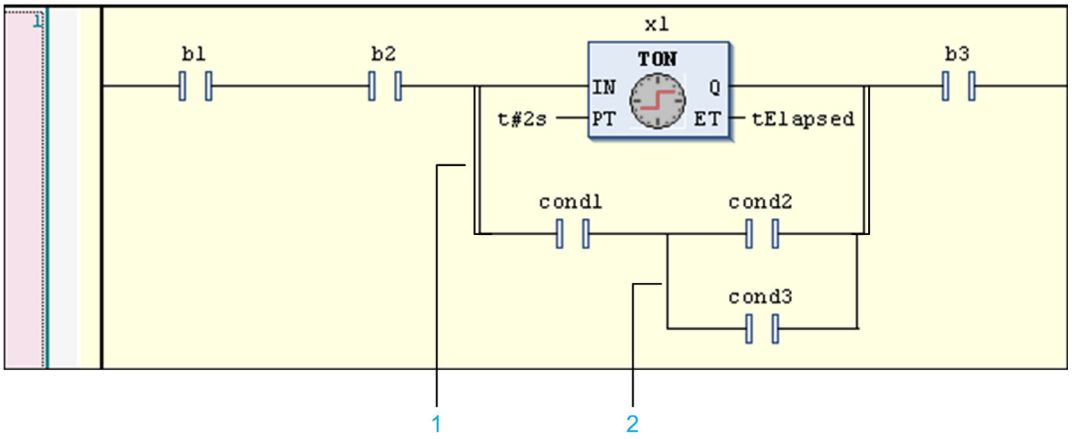
SCE ファンクションの並列分岐を挿入するには、ファンクションブロックボックスを選択して、**Insert Contact Parallel above** または **Insert Contact Parallel below** コマンドを実行します。これは、ファンクションブロックの最初の入力およびメインの出力が BOOL タイプの場合にのみ可能です。

以下は、ネットワーク用に生成された言語モデルの例です。

SCE の例

ファンクションブロックインスタンス x1（TON）に、ブール型入力とブール型出力があります。並列分岐内の条件が TRUE と評価された場合、その実行はスキップされます。この条件値は、接点 cond1、cond2、および cond3 を接続する OR および AND 演算の結果です。

ラダーネットワーク における SCE の並列分岐



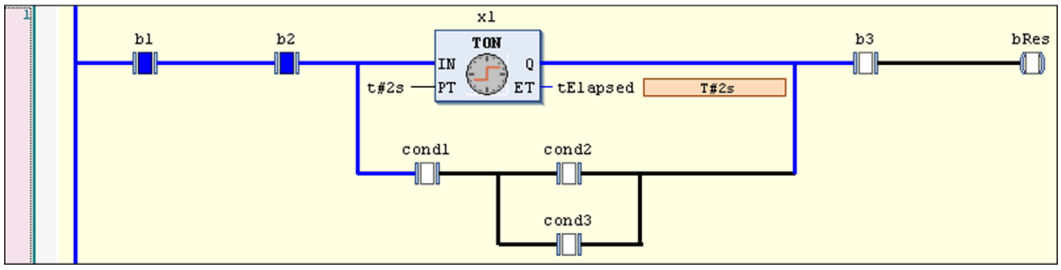
1 二重垂直接続線は、SCE の影響を受ける構成を示します。
2 単一の垂直接続線は OR 構造を示します。

処理は以下のようになります。P_IN および P_OUT はそれぞれ並列分岐の入力 (分岐点) におけるブール値と出力 (接合点) おけるブール値を表します。

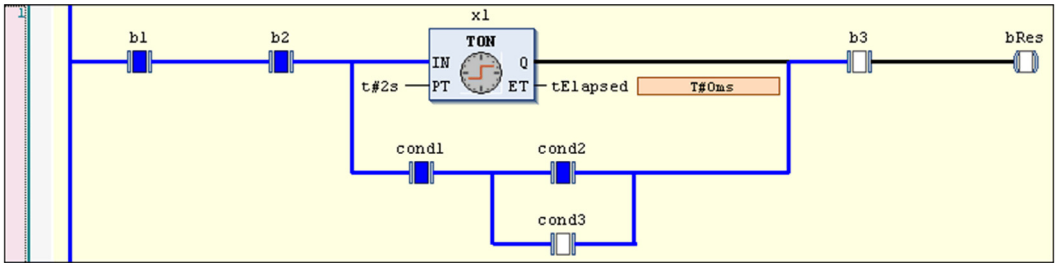
```
P_IN := b1 AND b2;
IF ((P_IN AND cond1) AND (cond2 OR cond3)) THEN
  P_OUT := P_IN;
ELSE
  x1(IN := P_IN, PT := {p 10}t#2s);
  tElapsed := x1.ET;
  P_OUT := x1.Q;
END_IF
bRes := P_OUT AND b3;
```

次の図に、ファンクションブロックが実行される場合 (cond1、cond2、および cond3 の結果の条件が FALSE)、または回避される場合 (条件が TRUE) のデータフロー (青色) を示します。

条件 =FALSE の場合、ファンクションブロックが実行されます。



条件 =TRUE の場合、ファンクションブロックは回避されます。



分岐の開始 / 終了

概要

分岐の開始 / 終了要素は LD 要素です。

分岐の開始点 / 終点を定義することができます。開始点が定義されていない場合は、要素を使用して定義できます。開始点がすでに定義されている場合、要素がによって終点が決定されます。

プロパティの詳細については、*並列分岐* (*269* ページ) の章を参照してください。

FBD/LD/IL のセット / リセット

FBD および LD

FBD (*234* ページ) のブール型出力またはそれに対応する LD (*235* ページ) コイルをセットまたはリセットできます。セット / リセットを変更するには、出力が選択された状態でコンテキストメニューの**セット / リセット**コマンドを使用します。出力またはコイルに、S または R の印が付きます。

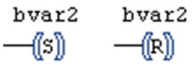
セット	値 TRUE がセット出力またはセットコイルに到達すると、この出力 / コイルは TRUE になり TRUE のまま維持されます。アプリケーションの実行中、この位置ではこの値を上書きできません。
リセット	値 TRUE がリセット出力またはリセットコイルに到達すると、この出力 / コイルは FALSE になり FALSE のまま維持されます。アプリケーションの実行中、この位置ではこの値を上書きできません。

FBD のセット出力

LD エディターでは、ドラッグ & ドロップでセットコイルおよびリセットコイルを挿入できます。この操作を実行するには、**ツールボックス**、カテゴリー **ラダー要素**、またはツールバーの S 要素および R 要素を使用します。

例 :

セットコイル、リセットコイル



詳細については、コイルのセット / リセット (*273* ページ) を参照してください。

IL

命令リストでは、S および R (*237* ページ) 演算子を使用してオペランドをセットまたはリセットします。

セット / リセットコイル

概要

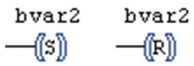
コイル (*277* ページ) は、セットコイルまたはリセットコイルとしても定義できます。

セットコイルは、コイル記号の S で識別できます : (S)。セットコイルは、適切なブール変数の値 TRUE を上書きしません。つまり、TRUE に設定された変数は TRUE のままです。

リセットコイルは、コイル記号の R で識別できます : (R)。リセットコイルは、適切なブール変数の値 FALSE を上書きしません。つまり、FALSE に設定された変数は FALSE のままです。

LD エディターでは、**ツールボックス**、**ラダー要素**カテゴリーからドラッグ & ドロップでセットコイルおよびリセットコイルを直接挿入できます。そうすることで、すでに挿入されているコイル要素を他の要素に置き換えることもできます。

セットコイル、リセットコイル



実行

概要

Execute 要素は、FBD または LD ネットワークに挿入でき、ST　コードで入力することができる EN/ENO をもつブロックです。ST コードで、ブロックが EN 入力の TRUE 信号で処理するために有効になったときに実行されます。

Execute 要素をツールボックス (*259* ページ) からネットワークにドラッグするか、または **FBD/LD/IL → Execute** コマンドを実行して要素を挿入します *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

ST コードの入力

手順	手順内容	コメント
1	ボックスで、Enter ST-Code here... というテキストの入力フィールドをクリックします。	ST エディター (*323* ページ) が開き通常の機能が表示されます。
2	ST コードを入力します。	必要な変数宣言は、FBD/LD/IL ファンクションブロックの宣言セクションに挿入されています。
3	ブロック外をクリックするか、**Ctrl** + **Enter** キーを押してコードの入力を完了します。	–

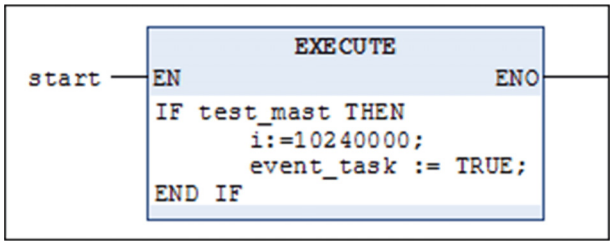
オンラインモード

オンラインモードでは、ファンクションブロックの EN 入力の下にあるプラス記号をクリックして ST エディターを開くことができます。通常のオンライン機能 (監視、デバッグ) をエディターで利用できます。

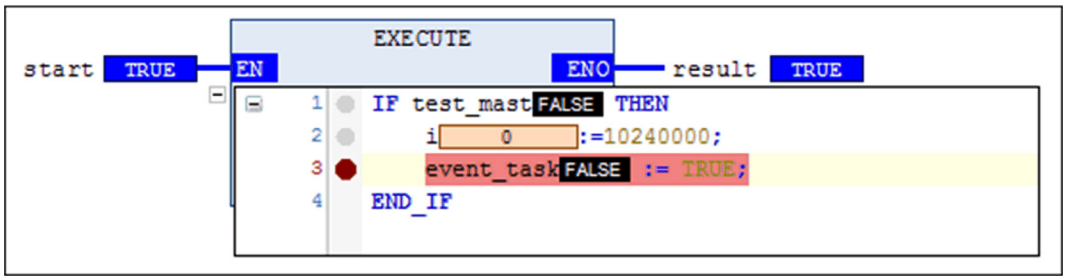
例

FBD ネットワークの Execute ブロックの例

オフラインモード、ST プログラムの挿入。



オンラインモード、ST エディターが開いています。



11.3
LD 要素

このセクションについて

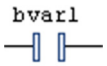
このセクションには次の項目が含まれています。

接点

概要

これは LD 要素です。

左側の LD (*235* ページ) には、各ネットワークに 1 つまたは複数の接点があります。接点は、2 本の垂直に平行な線で表されています。

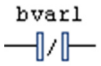


接点は、状態 ON (TRUE) または OFF (FALSE) を左から右へ渡します。各接点にブール変数が割り当てられます。この変数が TRUE の場合、状態が左から右に渡され、最後にネットワークの右側のコイルに渡されます。そうでない場合は、右側の接続は値 FALSE を受け取ります。

複数の接点を、直列および並列に接続できます。並列の接点とは、値 TRUE を並列分岐に送信させるためにはそれらのうちの 1 つが TRUE である必要がある論理 OR 条件を表します。逆に、直列の接点とは、最終接点に TRUE を送信させるためにはすべての接点が TRUE である必要がある論理 AND 条件を表します。

従って、接点の配置は電気的に並列または直列回路のいずれかに対応します。

接点は反転させることもできます。それは、接点の記号にスラッシュで示されます。



B 接点は、割り当てられたブール変数が FALSE の場合にのみ、受信した状態 (TRUE または FALSE) を送信します。

LD メニューの **Insert Contact**、**Insert Contact (right)**、**Insert Contact Parallel (above)**、**Insert Contact Parallel (below)**、**立上がり接点の挿入**、または**立下り接点の挿入**コマンドの内の 1 つで接点を LD ネットワークに挿入できます。または、**ツールボックス** (*259* ページ) からドラッグ & ドロップで、またはエディター内で別の位置から (ドラッグ & ドロップで) 要素を挿入することもできます。

すでに挿入されている A 接点を新しい A 接点または B 接点に置き換えることができます。これをするには、ツールボックス (*259* ページ) から既存の A 接点に A 接点または B 接点をドラッグ & ドロップします。

FBD および IL

FBD (*234* ページ) または IL (*236* ページ) ビューで作業中の場合、コマンドは使用できません。ただし、LD ネットワークに挿入された接点とコイルは対応する FBD 要素または IL 命令で表示されます。

コイル

概要

これは LD 要素です。

LD ネットワークの右側には、任意の数の括弧で表されるコイルを置くことができます。

並列にのみ配置できます。コイルは接続の値を左から右に送信し、適切なブール変数にコピーします。入力行では、値 ON (TRUE) または 値 OFF (FALSE) がある場合があります。

コイルを反転させることもできます。これはコイル記号にスラッシュで表示されます。

この場合、入力信号の反転した値が適切なブール変数にコピーされます。

LD メニューの**コイルの挿入**、**セットコイルの挿入**、**リセットコイルの挿入**、または**反転コイルの挿入**コマンドの 1 つを使用してネットワークにコイルを挿入できます。または、**ツールボックス** (**ラダー要素**) から要素をドラッグ & ドロップするか、エディター内の他の場所からドラッグ & ドロップして要素を挿入することもできます。*セットコイルとリセットコイル* (*273* ページ) も参照してください。

FBD および IL

FBD (*234* ページ) または IL (*236* ページ) ビューで作業中の場合、コマンドは使用できません。ただし、LD ネットワークに挿入された接点とコイルは対応する FBD 要素または IL 命令で表示されます。

第 12 章
コンティニュアスファンクションチャート (CFC) エディター

この章について

この章には次の項目が含まれています。

コンティニュアスファンクションチャート (CFC) 言語

概要

コンティニュアスファンクションチャートは IEC 61131-3 規格の拡張であり、ファンクションブロックダイアグラム (FBD) 言語 (*234* ページ) に基づくグラフィックなプログラミング言語です。ただし、FBD 言語とは異なりネットワークはありません。CFC では、グラフィック要素を自由に配置でき、フィードバックループが可能です。

EcoStruxure Machine Expert で CFC プログラミングオブジェクトを作成するには、CFC エディター (*281* ページ) の説明を参照してください。

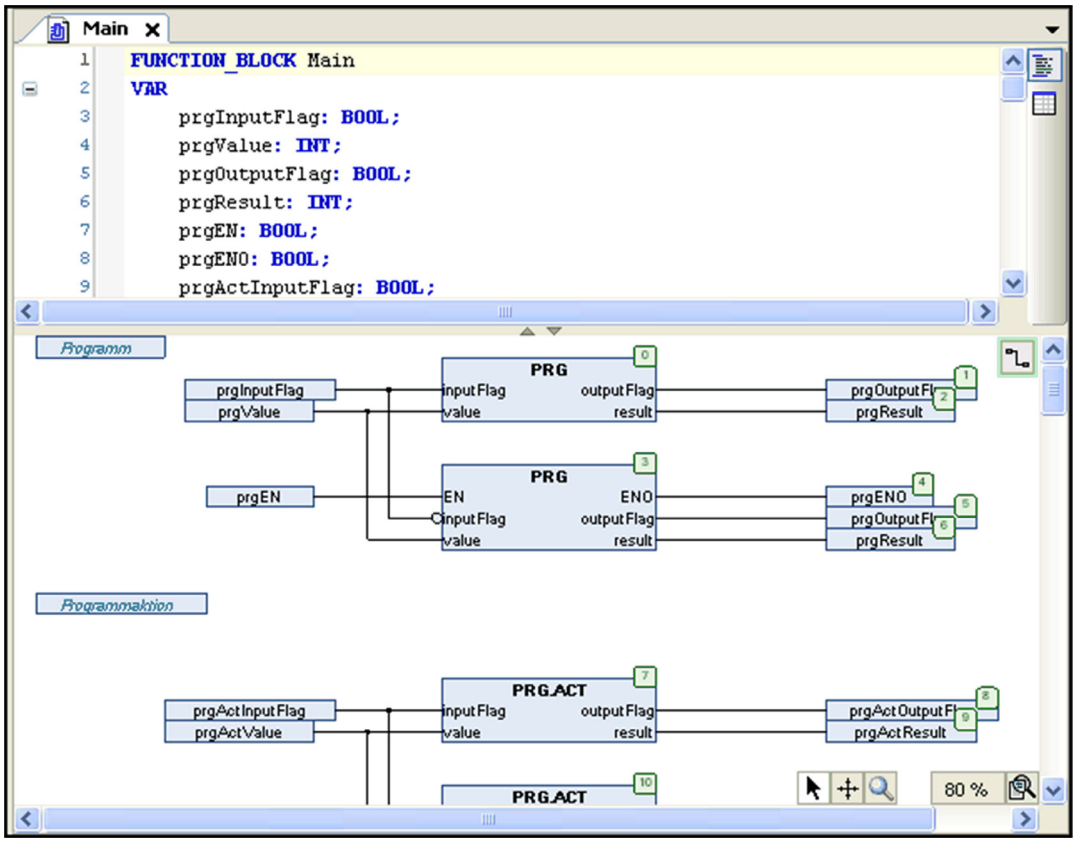
CFC エディター

概要

CFC エディターは、IEC 61131-3 プログラミング言語の拡張であるコンティニュアスファンクションチャート (CFC) プログラミング言語 (*280* ページ) でオブジェクトをプログラミングするためのグラフィックエディターです。新しいプログラム構成単位 (POU) オブジェクトをプロジェクトに追加するときに言語を選択します。大規模なプロジェクトの場合は、ページ指向バージョン (*294* ページ) の使用を検討してください。

この SFC エディターは、CFC POU オブジェクトを開くときに開くウィンドウの下部にあります。ウィンドウの上部には宣言エディター (*342* ページ) があります。

CFC エディター



FBD / LD エディターとは異なり CFC エディターでは自由に要素を配置 (*290* ページ) でき、フィードバックパスの直接挿入ができます。処理のシーケンスは、現在挿入されているすべての要素を含む変更可能なリストによって決定します。

ツールボックス (*285* ページ) には次の要素があり、ドラッグ & ドロップで挿入できます。

- ボックス (演算子、ファンクション、ファンクションブロック、およびプログラム)
- 入力
- 出力
- ジャンプ
- ラベル
- リターン
- コンポーザー
- セレクター
- 接続マーク
- コメント

マウスで線を描くことによって、要素の入力ピンと出力ピンを接続できます。接続線の経路は自動的に可能な限り最短の経路で作成されます。接続線は、要素を移動するとすぐに自動的に調整されます。詳細については、要素の挿入と配置 (*290* ページ) の説明を参照してください。複雑なチャートの場合、接続線の代わりに接続マーク (*285* ページ) を使用できます。また、経路の変更も検討してください。

要素が既に配置された接続線に重ねて配置される場合があります。これらの重複は、赤色の接続線で示されます。チャートに重複がある場合、エディタービューの右上の角にあるボタンの枠が赤色になります  。1 つずつ重複を編集するには、このボタンをクリックして **Show next collision** コマンドを実行します。次の問題のある接続が選択されます。

複雑なチャートの場合、接続線の代わりに接続マーク (*285* ページ) を使用できます。ページ指向バージョンの CFC エディターを使用することもできます。

ズーム機能を使用すると、エディターウィンドウのサイズを変更できます。ウィンドウの右下隅にある  ボタンを使用して、表示されているズーム倍率の中から選択します。または、**...** を選択して任意のズーム率を入力できるダイアログボックスを開くこともできます。

コンテキストメニューまたは、CFC エディターが有効になるとすぐに使用できる **CFC** メニューから、CFC エディターで作業するためのコマンドを呼び出せます。

CFC のカーソル位置

概要

CFC プログラムのカーソル位置は、カーソルをプログラミング要素上に乗せたときに灰色の背景で表示されます。

この影付き領域をクリックすると、マウスボタンを離す前に背景色が赤に変わります。マウスボタンを離すとそこが現在のカーソル位置になり、それぞれの要素またはテキストが選択されて赤色で表示されます。

カーソルの位置には 3 種類あります。次の項の図に灰色の影付き領域で表示されているカーソル位置を参照してください。

テキスト上のカーソル

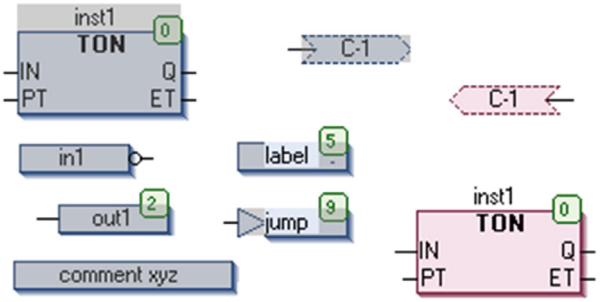
テキスト上にカーソルがあるときにマウスボタンをクリックすると青い影付きで表示され、編集ができます。... ボタンを使用して入力アシスタントを開くことができます。要素の挿入後、初めは要素の名前が文字 ??? で表示されます。この文字を有効な識別子に置き換えます。その後、変数またはボックスパラメーターの名前にカーソルを置くとツールチップが表示されます。ツールチップには変数またはパラメーターのタイプが表示され、関連するコメントがあればそれが 2 行目に表示されます。

カーソル位置と選択されたテキストの例

要素の本体上のカーソル

カーソルが要素の本体 (ボックス、入力、出力、ジャンプ、ラベル、リターン、コメント、接続マーク) にある場合は赤色で表示され、マウスを動かすことで移動させることができます。

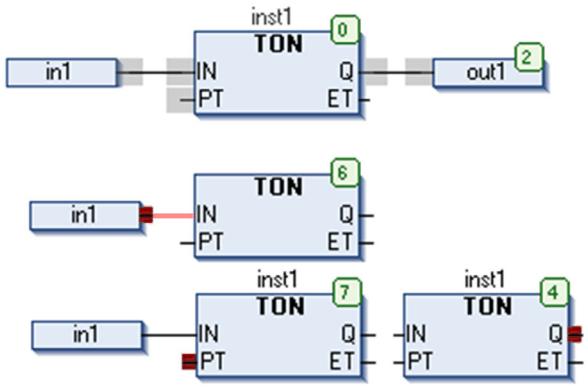
カーソル位置と選択された本体の例



要素の入力接続または出力接続の本体上のカーソル

カーソルが要素の入力接続または出力接続上にある場合は、赤色で塗りつぶされた四角がその位置 (接続点) を示します。これは反転、またはセット／リセットできます。

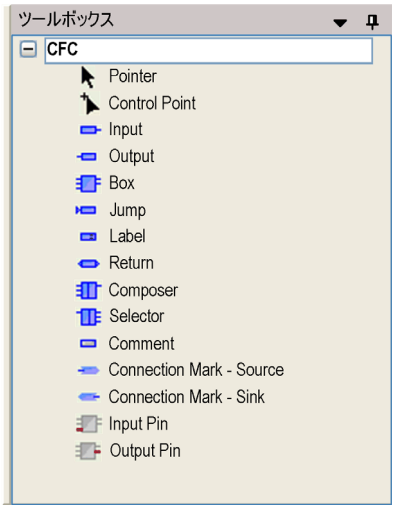
カーソル位置 (灰色の影) および選択された出力および入力位置の例 (赤い四角)



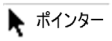
CFC 要素 / ツールボックス

概要

CFC エディター (*281* ページ) ウィンドウでプログラミング用のグラフィック要素をツールボックスから使用できます。**表示**メニューの**ツールボックス**コマンドを実行して表示ウィンドウのツールボックスを開きます。

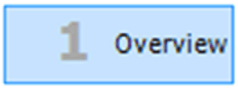


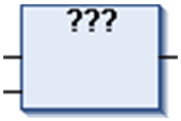
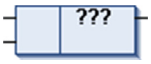
ツールボックスから必要な要素を選択し、ドラッグ & ドロップでエディターウィンドウに挿入 (*290* ページ) します。

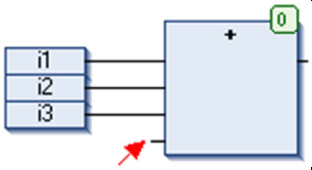
プログラミング要素の他に、ツールボックスリストの一番上には 

があります。このポインターが選択されているとカーソルが矢印の形になり、エディターウィンドウから配置および編集する要素を選択できます。

CFC 要素

名前	シンボル	詳細
ページ		位置に応じて自動的にページ番号が付きます。ページ上部のオレンジ色のフィールドに名前 (この例では Overview) を入力できます。
制御ポイント		手動で変更された接続線の経路を修正するためには制御ポイントが必要です。これにより、**Route All Connections** コマンドによって修正が元に戻ることを防ぎます。2 つの制御ポイントにより、経路を修正する線の正確なセグメントをマークできます。
入力		テキスト ??? を選択し、それを変数または定数に置き換えることができます。入力アシスタントから有効な識別子を選択できます。
出力		

名前	シンボル	詳細
ボックス		ボックスを使用して、演算子、ファンクション、ファンクションブロック、およびプログラムを表すことができます。テキスト ??? を選択し、それを演算子、ファンクション、ファンクションブロック名に置き換えることができます。入力アシスタントから使用可能なオブジェクトを選択できます。 ファンクションブロックを挿入すると、ボックスの上に別の ??? が表示されます。疑問符をファンクションブロックインスタンスの名前に置き換えます。定数入力パラメーターを持つファンクションブロックがインスタンス化されると、ボックス要素は左下にフィールド **Parameters...** が表示されます。このボタンをクリックして入力パラメーターの編集をするダイアログボックスを開きます。*パラメーターを編集しています ...* の章を参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。 既存のボックスを別のボックスに (入力した名前を変更して) 置き換えたとき、ボックスの入力または出力ピンの最小数または最大数が異なる場合は対応するようにピンが調整されます。ピンが削除される場合は、一番低いピンが最初に削除されます。
ジャンプ		ジャンプ要素を使用して、プログラムの実行を継続させる位置を指定します。位置はラベルで定義します (下記参照)。そのため、??? というテキストはラベル名に置き換えてください。
ラベル		ラベルでプログラムのジャンプ先の位置を指定できます (要素のジャンプを参照してください)。 オンラインモードでは、POU の終了を指定するためのリターンラベルが自動的に挿入されます。
戻る		オンラインモードでは、エディターの最初の列と最後の要素の後にリターン要素が自動的に挿入されます。ステップでは、実行が POU から抜ける前に自動的にジャンプします。
コンポーザー		コンポーザーを使用して、構造体型のボックスの入力を処理します。コンポーザーは構造体のコンポーネントを表示し、プログラマーが CFC でそれらにアクセスできるようにします。そのためには、コンポーザーを関連する構造体のような名前にし (??? を名前に置き変えて)、入力要素を使用せずにボックスに接続します。
セレクター		コンポーザーとは異なり、セレクターは構造体型のボックスの出力を処理するために使用します。セレクターは構造体のコンポーネントを表示し、プログラマーが CFC でそれらにアクセスできるようにします。そのためには、セレクターを関連する構造体のような名前にし (??? を名前に置き変えて)、出力要素を使用せずにボックスに接続します。
コメント		この要素を使用して、チャートにコメントを追加します。プレースホルダーテキストを選択して任意のテキストに置き換えます。コメントに新しい行を入れるには CTRL + ENTER を押します。

名前	シンボル	詳細
接続マーク - ソース 接続マーク - シンク		要素間の接続線 (*290* ページ) の代わりに接続マークを使用できます。これにより複雑なチャートがわかり易くなります。 有効な接続にするには、要素の出力に接続マーク - ソース要素を割り当て、別の要素の入力に接続マーク - シンク (下記参照) を割り当てます。両方のマークに同じ名前を割り当てます (大文字、小文字を区別しません)。 命名 : CFC に挿入された最初の接続マーク - ソース要素の名前の初期設定は C-1 です。これは手動で変更できます。それに対応する接続マーク - シンクは、??? をソースマークで使用した名前と同じ文字列に置き換えます。エディターによってマークの名前が固有であることが確認されます。ソースマークの名前を変更すると、接続したシンクマークの名前も自動的に変更されます。ただし、シンクマークの名前を変更した場合、ソースマークは古い名前のままです。これにより、接続を再設定できます。同様に、接続マークを削除しても、それに対応する接続マークは削除されません。 チャートで接続マークを使用するには、接続マークをツールボックスからエディターウィンドウにドラッグし、そのピンを対応する要素の出力または入力ピンに接続します。または、**Connection Mark** コマンドを使用して既存の通常接続線を変換することもできます。このコマンドによって、接続マークを通常の接続線に戻すこともできます。 接続マークの例は、*接続マーク*の章を参照してください。
入力ピン		ボックスのタイプによっては入力を追加できます。追加するには、CFC ネットワークでボックス要素を選択して、ボックスに入力ピン要素を描画します。 **Ctrl** キーを押しながら、入力または出力接続をボックスの別の位置にドラッグできます。
出力ピン	–	ボックスのタイプによっては出力を追加できます。追加するには、CFC ネットワークでボックス要素を選択して、ボックスに出力ピン要素を描画します。 **Ctrl** キーを押しながら、入力または出力接続をボックスの別の位置にドラッグできます。

コンポーザーの例

CFC プログラム cfc_prog は、構造体型の入力変数 struvar をもつファンクションブロック fublo1 のインスタンスを処理します。コンポーザー要素を使用して構造体コンポーネントにアクセスします。

構造体定義 stru1 :

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

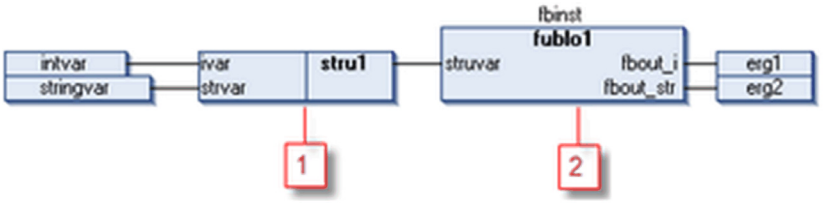
ファンクションブロック fublo1 の宣言と実装 :

```
FUNCTION_BLOCK fublo1
VAR_INPUT
  struvar:STRU1;
END_VAR
VAR_OUTPUT
  fbout_i:INT;
  fbout_str:STRING;
END_VAR
VAR
  fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

プログラム cfc_prog の宣言と実装 :

```
PROGRAM cfc_prog
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
END_VAR
```

コンポーザー要素



1 コンポーザー
2 構造体型 stru1 の入力変数 struvar をもつファンクションブロック

セレクターの例

CFC プログラム cfc_prog は、構造体型 stru1 の出力変数 fbout をもつファンクションブロック fublo2 のインスタンスを処理します。セレクター要素を使用して構造体コンポーネントにアクセスします。

構造体定義 stru1 :

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

ファンクションブロック fublo1 の宣言と実装 :

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
  fbin1:INT;
  fbin2:DWORD:=24354333;
  fbin3:STRING:='hallo';
END_VAR
VAR_INPUT
  fbin : INT;
END_VAR
VAR_OUTPUT
  fbout : stru1;
  fbout2:DWORD;
END_VAR
VAR
  fbvar:INT;
  fbvar2:STRING;
END_VAR
```

プログラム cfc_prog の宣言と実装 :

```
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
  fbinst2: fublo2;
END_VAR
```

Remove Unused Pins コマンドを実行して未使用ピンを削除したセレクター要素を下の図に示します。

1 構造体型 stru1 の出力変数 fbout をもつファンクションブロック
2 セレクター

CFC エディターの操作

概要

CFC エディターのプログラミング用要素は、CFC エディターを開いたときに初期設定でウィンドウにある**ツールボックス** (*285* ページ) にあります。

ツール → オプション → CFC エディターで、エディターで作業するための一般設定を定義します。

挿入

要素を挿入するには、マウスをクリックして**ツールボックス** (*285* ページ) の要素を選択し、マウスボタンを押したまま要素をエディターウィンドウの目的の位置にドラッグします。ドラッグ中はカーソルが矢印になり、長方形とプラス記号が表示されます。マウスボタンを離すと要素が挿入されます。

選択

挿入した要素を編集または再配置するために選択するには、要素の本体をクリックして要素を選択します。初期設定では、赤色の影付きで表示されます。SHIFT キーを押しながらさらに要素をクリックして選択できます。また、左のマウスボタンを押して、選択するすべての要素の周りに点線の長方形を描くこともできます。ボタンを離すと、選択された要素が表示されます。**すべて選択**コマンドでは、一度にすべての要素が選択されます。

矢印キーを使用して、選択マークを次のカーソル位置に移動できます。シーケンスは、実行順序または要素番号 (*291* ページ) で表示される要素によって異なります。

入力ピンを選択して CTRL + 左矢印を押すと、対応する出力が選択されます。出力ピンを選択して CTRL + 左矢印を押すと、対応する出力が選択されます。

変数のドラッグ & ドロップ

ファンクションブロックの入力要素または出力要素として、GVL (*172* ページ)、または POU (*135* ページ) から直接変数を挿入できます。これを実現するには、GVL または POU で変数を選択し、それをファンクションブロックの入力または出力ピンにドラッグします。この変数の入力要素または出力要素は自動的に作成され、ファンクションブロックのピンに接続されます。

ボックスの置き換え

既存のボックス要素を置き換えるには、現在挿入されている識別子を新しい要素の識別子に置き換えます。POU の定義により必要に応じて入力ピンおよび出力ピンの数が調整されるため、既存の割り当てが削除される場合があります。

移動

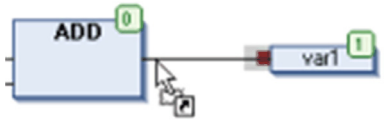
要素を移動するには、要素の本体をクリックして (配置可能なカーソル位置 (*283* ページ) を参照) 要素を選択し、マウスボタンを押したままドラッグして目的の位置に移動させます。次に、マウスボタンを離して要素を配置します。この操作に、**切り取り**と**貼り付け**コマンドを使用することもできます。

接続

接続線または接続マークで、2 つの要素の入力ピンと出力ピンを接続できます。

接続線 : 要素の入力ピンまたは出力ピンである有効な接続点 (*CFC のカーソル位置* (*283* ページ) を参照) を選択し、マウスを使用して別の接続点へ線を引きます。または、2 つの接続点を選択して **Select connected pins** コマンドを実行します。選択した接続点は、赤色で塗りつぶされた四角で表示されます。そのような点から目的の要素に線を引くと、接続可能な接続点を特定できます。接続可能な接続点にカーソルを合わせると、その点の上に移動させたときにカーソルに矢印記号が追加され、接続が可能であることが示されます。

例を次の図で示します。var1 要素の入力ピンをマウスでクリックすると赤い四角が表示され、接続点が選択されていることが示されます。マウスボタンを押したまま、カーソル記号が図のように表示されるまでカーソルを ADD ボックスの出力ピンへ移動させます。マウスボタンを離して接続線を構築します。



他の要素および接続を考慮した可能な限り最短の接続線が作成されます。

Ctrl キーを押しながら、入力または出力接続をボックスの別の位置にドラッグできます。

接続マーク : 複雑なチャートを簡素化するために、接続線の代わりに接続マークを使用することもできます。接続マーク (*285* ページ) の説明を参照してください。

コピー

要素をコピーするには、要素を選択して**コピー**と**貼り付け**コマンドを使用します。

編集

要素を挿入すると、初期設定ではテキスト部分に ??? が表示されます。これを目的のテキスト (POU 名、ラベル名、インスタンス名、コメントなど) に置き換えるには、テキストをクリックして編集フィールドを表示させます。また、**入力アシスタント**を開くための **...** ボタンも使用できます。

削除

選択した要素または接続線を削除するには、コンテキストメニューから**削除**コマンドを実行するか、DEL キーを押します。

ファンクションブロックを開く

エディターにファンクションブロックが追加されている場合は、ダブルクリックでこのブロックを開くことができます。または、コンテキストメニューから**ブラウズ → 宣言へ移動** コマンドを使用することもできます。

実行順序、要素番号

CFC ネットワークの要素がオンラインモードで実行される順序は、ボックス、出力、ジャンプ、リターン、およびラベル要素の右上の角の数字で示されます。最も小さい番号 (0) の要素から処理が開始します。

実行順序は、**CFC** メニューのサブメニュー**実行順序**にあるコマンドで変更できます。

要素を追加すると、番号が自動的に位相順序 (左から右、上から下) に従って付けられます。既に順序が変更されている場合、新しい要素には位相的に次に続く番号が付けられ、それより大きい番号はすべて 1 つ増えます。

要素を移動しても要素の番号は同じままであると考えてください。

場合によっては順序が結果に影響するため、順序を変更する必要があることを考慮してください。

作業シートのサイズの変更

エディターウィンドウで既存の CFC チャートの周りのスペースを増やすには、作業領域 (作業シート) のサイズを変更します。そのためには、マウスですべての要素を選択してドラッグするか、切り取りと貼り付けコマンド (*移動* (*290* ページ) を参照) を使用します。

または、特別な寸法設定ダイアログボックスを使用することもできます。これにより、大きなチャートの場合に時間を節約できます。**作業シートの編集** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。ページ指向 CFC の場合は、**Edit Page Size** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用できます。

オンラインモードの CFC エディター

概要

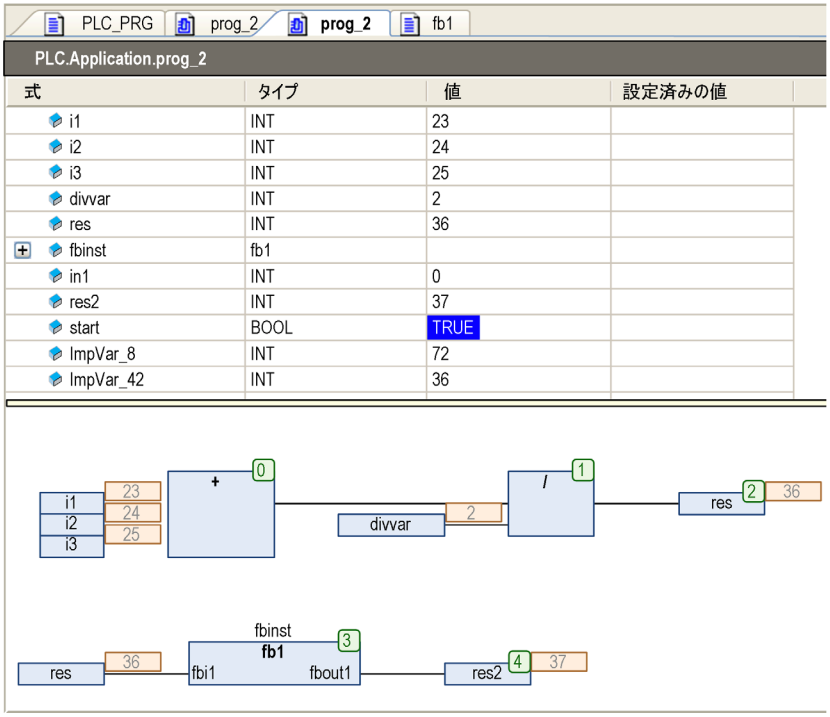
オンラインモードの CFC エディターでは、監視用、およびコントローラーの変数と式の書き込みと強制用のビューが表示されます。下の説明にあるように、デバッグ機能 (ブレークポイント、ステップなど) を使用できます。

- オンラインモードでのオブジェクトの開き方については、オンラインモードのユーザーインターフェイス (*43* ページ) の説明を参照してください。
- CFC オブジェクトのエディターウィンドウの上部には宣言エディターがあります。オンラインモードの宣言エディター (*346* ページ) の説明を参照してください。
- CFC エディターのパラメーター編集についての情報は、**パラメーターの編集** の説明を参照してください。コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

監視

実際の値が各変数の後ろの小さな監視ウィンドウに表示されます (インライン監視)。

プログラミングオブジェクト PLC_PRG のオンラインビュー



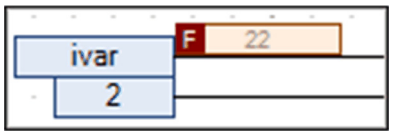
ファンクションブロック POU のオンラインビューにおいて、監視はインスタンスビューでのみ可能です。ファンクションブロック POU の基本実装では、値は表示されません。**値**列には式のテキスト **Value of the expression** が表示され、実装部分のインライン監視フィールドにはそれぞれ 3 つの疑問符が表示されます。

ブール接続は、TRUE = 青、および FALSE = 黒で監視されます。

変数の強制 / 書き込み

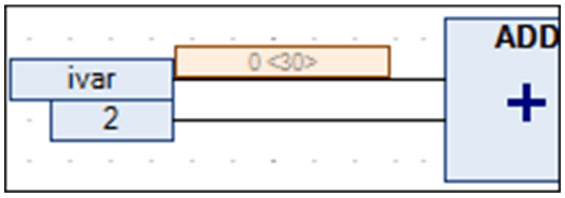
オンラインモードでは、宣言エディター、(または、**実装パートに値の準備**オプションが有効化されている場合は実装部分でも)、監視対象の変数の強制または書き込み用の値を準備できます。宣言エディターの作業については、*オンラインモードでの宣言エディターの章を参照してください* (*346* ページ)。実装部分で、それぞれの要素の横にある監視ボックスをクリックするか、または直接要素をクリックして**値の準備** ダイアログボックスを開きます (*324* ページ)。ブール変数の場合ダイアログボックスは開きませんが、変数の横に現在表示されている値をクリックすると、強制または書き込み可能な値を直接切り替えることができます。現在強制されている変数の監視ボックスに、赤い **F** が表示されます。

実装部分での強制値



CFC オプションの **実装パートに値の準備** が有効になっている場合、現在書き込みまたは強制のために準備されている値が、変数の監視フィールド内の山括弧 <> 内の現在値の後ろに表示されます。

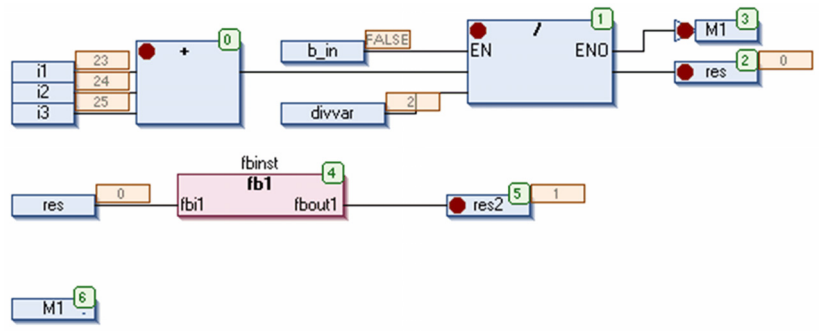
実装部分での設定済みの値



CFC エディターのブレークポイント位置

ブレークポイントを配置できる基本的な位置は、POU 内で変数の値が変更できる位置、またはプログラムフローが分岐される位置または他の POU が呼び出される位置です。次の図の配置できる位置を参照してください。

CFC エディターのブレークポイント位置



注記：ブレークポイントは、呼び出される可能性があるすべてのメソッドに自動的に設定されます。インターフェイス管理メソッドが呼び出されると、そのインターフェイスを実装するファンクションブロックのすべてのメソッドと、メソッドをサブスクライブするすべての派生ファンクションブロックにブレークポイントが設定されます。ファンクションブロックのポインターによってメソッドが呼び出される場合、ブレークポイントはファンクションブロックのメソッドとメソッドをサブスクライブしているすべての派生ファンクションブロックに設定されます。

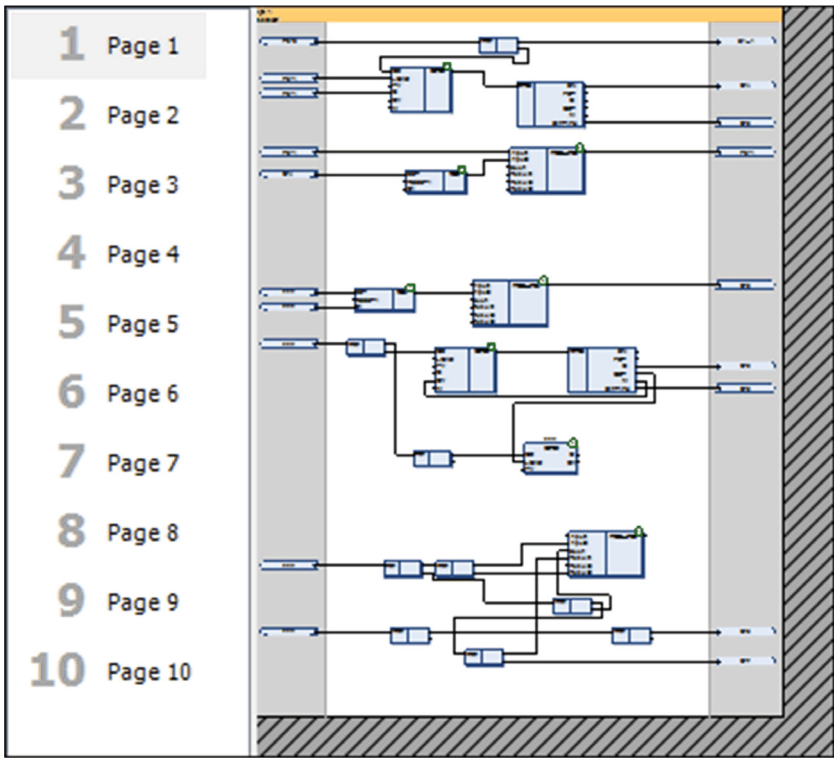
ページ指向 CFC エディター

概要

CFC の標準エディターだけでなく、EcoStruxure Machine Expert CFC エディターでもページ付けができます。標準の CFC エディターのツール (*285* ページ) およびコマンドに加えて、このエディターでは任意の数の異なるページに要素を配置できます。

注記：ページ指向 CFC 言語で作成した POU を標準の CFC には変換できません。また、その逆もできません。コピーおよび貼り付け (クリップボードから) またはドラッグ & ドロップ機能でこれら 2 つのエディター間の要素をコピーできます。

CFC ページ付け



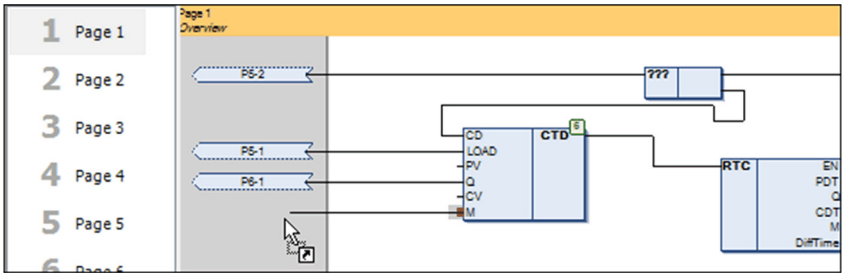
ページのサイズを変更するには、**ページサイズの編集**コマンドを実行します。

2 ページ間の接続

2 ページ間の接続は、要素である**接続マーク - ソース**、および**接続マーク - シンク**で実現します。ドラッグ & ドロップで**接続マーク - ソース** を右の余白に、**接続マーク - シンク**を左の余白に配置します。要素の入力または出力から余白に接続ラインを描くと、自動的に接続マークが配置されます。**...** ボタンをクリックして、**入力アシスタント**から接続マークのリストを開きます。

矢印キーを使用して、図内の要素から次の要素に移動します。

接続マークの挿入



実行順序

ページは、上から下の順序で実行されます。ページ内の順序は、標準 CFC エディターの規則に従います (詳細は、実行順序 (*291* ページ) を参照してください)。関連するページ内でのみ要素の実行順序を変更できます。異なるページの要素の実行順序は変更できません。

第 13 章
シーケンシャル ファンクションチャート (SFC) エディター

この章について

この章には次の項目が含まれています。

SFC エディター

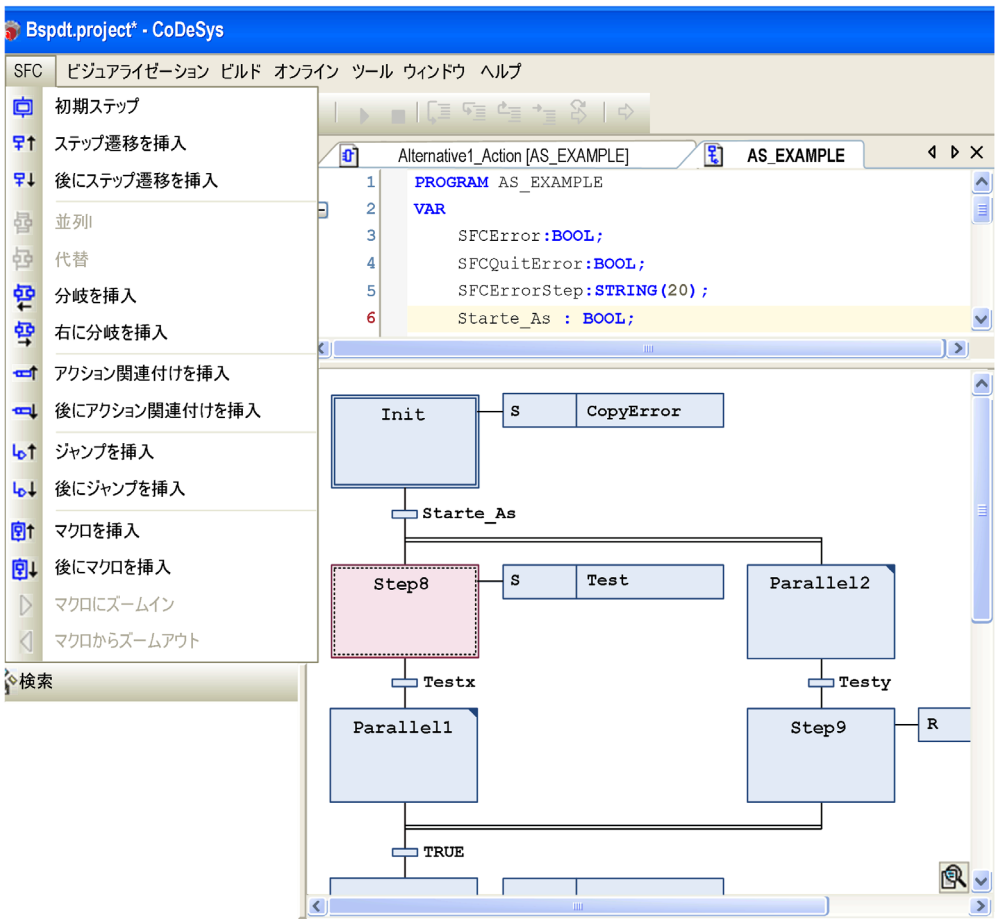
概要

SFC エディターは、IEC 61131-3 プログラミング言語 SFC - シーケンシャルファンクションチャート (*299* ページ) でオブジェクトをプログラミングするために使用します。プロジェクトに新しい POU オブジェクトを追加するときに言語を選択します。

SFC エディターはグラフィックエディターです。**オプション → SFC エディター**ダイアログボックスで、動作と表示に関する一般設定を行います。

SFC エディターは、SFC POU オブジェクトを編集するときに開くウィンドウの下部にあります。ウィンドウの上部には**宣言エディター** (*342* ページ) があります。

SFC エディター



SFC エディターの操作

SFC 図で使用する 要素 (*305* ページ) は **SFC** メニューにあります。このメニューは、SFC エディターが有効になるとすぐに使用できます。それらを、遷移で接続されたシーケンスまたは平行な一連のステップに配置します。詳細については、*SFC エディターの操作* (*301* ページ) を参照してください。

別のプロパティ (*303* ページ) ウィンドウでステップのプロパティを編集できます。特に、各ステップの動作の最小時間と最大時間を定義できます。

SFC の処理を制御するために暗黙の変数 (*314* ページ) にアクセスできます (例えば、ステップの状態、タイムアウトの分析、リセットなど)。

SFC - シーケンシャルファンクションチャート言語

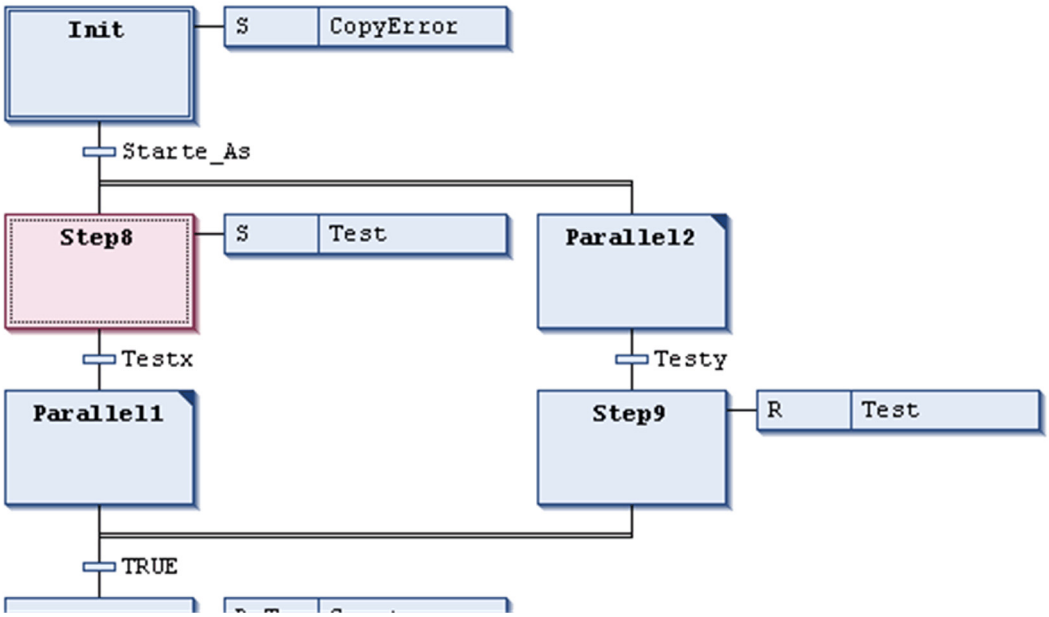
概要

シーケンシャルファンクションチャート (SFC) は、プログラム内に特定のアクションを時系列に記述するグラフィック指向の言語です。アクションは、使用可能なプログラミング言語で書かれた独立したプログラミングオブジェクトとして利用できます。SFC では、それらはステップ要素に割り当てられ、処理の順序は遷移要素で制御されます。オンラインモードでのステップの処理方法についての詳細は、*SFC の処理順序* (*318* ページ) を参照してください。

EcoStruxure Machine Expert の SFC エディターの使用方法については、*SFC エディター* (*298* ページ) の説明を参照してください。

例

SFC モジュールの一連のステップの例



SFC のカーソル位置

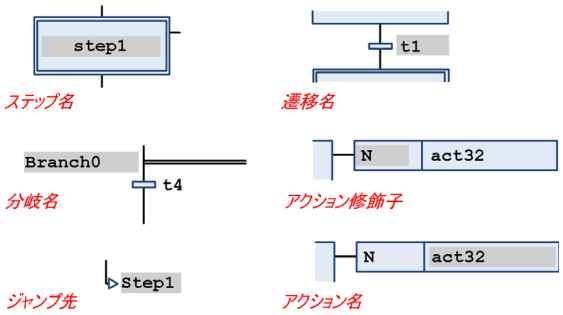
概要

SFC エディター (*298* ページ) の SFC 図のカーソル位置は、カーソルを要素の上に移動したときに灰色の影付きで表示されます。

テキストのカーソル位置

カーソル位置には、テキストと要素の 2 種類があります。次の図の灰色の影付き領域で示されているカーソルの配置可能位置を参照してください。

テキストのカーソル位置

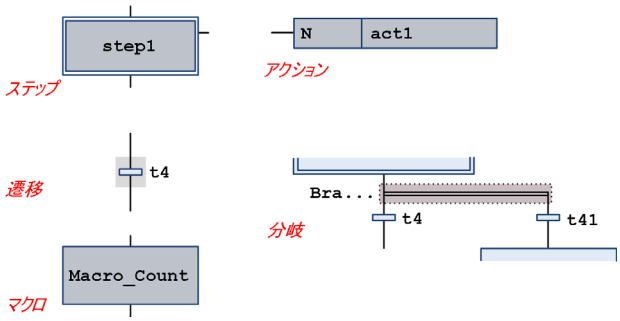


テキストのカーソル位置をクリックすると、文字列が編集できます。

編集するアクション名の選択

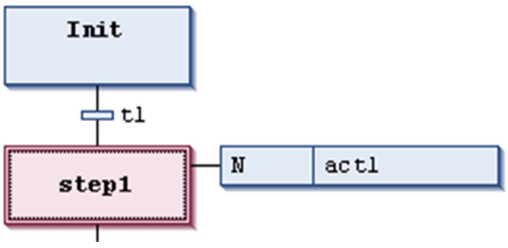
要素のカーソル位置

要素のカーソル位置



影付き領域をクリックすると、要素が選択されます。点線枠になり、赤い影付きで表示されます (複数の選択については、*SFC エディターでの作業* (*301* ページ) を参照してください)。

選択したステップ要素



SFC エディターの操作

概要

初期設定では、新しい SFC POU には初期ステップとその後の遷移が含まれています。この章では、要素の追加方法と要素の配置および編集方法を説明します。

カーソルの配置可能位置

詳細については、*SFC のカーソル位置* (*300* ページ) の章を参照してください。

移動

チャートの次の要素または前の要素にジャンプするには、矢印キーを使用します。

要素の挿入

特定の SFC 要素 (*305* ページ) を挿入するには、**SFC** メニューから各コマンドを実行します。詳細については、SFC エディターのコマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。すでに挿入されているがまだプログラミングオブジェクトを参照していないステップ、遷移、またはアクション要素をダブルクリックして、それを割り当てるためのダイアログボックスを開きます。

要素の選択

カーソルの配置可能な位置をクリックして要素またはテキストフィールドを選択します。また、矢印キーを使用して隣接する要素に選択範囲を広げることもできます。要素の色が赤に変わります。詳細については、*SFC のカーソル位置* (*300* ページ) の章を参照してください。

注記：以前のバージョンの EcoStruxure Machine Expert とは異なり、ステップおよび遷移を個別に選択して移動 (切り取り、コピー、貼り付け) または削除できます。

複数選択する場合は、次の操作を行います。

- SHIFT キーを押したまま、選択する要素をクリックします。
- マウスの左キーを押して、選択する要素の周りに長方形 (点線) を描きます。
- 初期設定では**編集**メニューから**すべて選択**コマンドを実行します。

テキストの編集

テキストのカーソル位置をクリックして、テキストの編集ができる編集フィールドを開きます。テキスト領域が矢印キーで選択されている場合、スペースバーを使用して明示的に編集フィールドを開きます。

関連付けられたアクションの編集

ステップ (入力、アクティブ、または終了) または遷移アクションの関連付けをダブルクリックし、対応するエディターで関連付けられたアクションを開きます。例えば、遷移要素またはステップ要素の終了アクションを示す三角をダブルクリックできます。

要素の切り取り、コピー、貼り付け

要素を選択し、**切り取り**、**コピー**、または**貼り付け**コマンド (**編集**メニューから) を実行するか、対応するキーを使用します。

注記：

- 切り取りまたはコピーされた 1 つまたは複数の要素を貼り付けると、クリップボードの内容が現在選択している位置の前に挿入されます。何も選択されていない場合、要素は現在読み込まれているチャートの最後に追加されます。
- 現在選択している要素が分岐の場合に分岐を貼り付けると、貼り付けられた分岐は既存の要素の左側に挿入されます。
- 現在選択しているステップにアクション (リスト) を貼り付けると、ステップのアクションリストの先頭にアクションが追加されるか、またはステップのアクションリストが作成されます。
- 切り取り / コピー時の互換性のない要素 :
  関連するアクション (リスト) を選択し、さらにアクション (リスト) に属しているステップではない要素を選択すると、次のメッセージボックスが表示されます。**現在の選択に互換性のない要素が含まれています。クリップボードにデータは保存されません。**選択内容は保存されず、別の場所への貼り付けまたはコピーもできません。

- 貼り付け時の互換性のない要素 :
  現在選択している要素がステップではない、または他に関連がない場合にアクション (リスト) を貼り付けようとすると、次のメッセージボックスが表示されます。**現在のクリップボードの内容を現在の選択場所に貼り付けできません。**現在関連付けられているアクション (リスト) を選択しているときに、ステップ、分岐、または遷移のような要素を貼り付けようとすると、同じメッセージボックスが表示されます。
- アクションオブジェクトまたは遷移オブジェクトを呼び出すステップおよび遷移要素をコピーするために、2 つの複製モードが利用可能です。リファレンスのみがコピーされるか、または参照されたオブジェクトがコピー時に埋め込まれて複製されます。**Duplicate on copy** オプションを使用して**要素プロパティ**ダイアログボックスでモードを選択します (*303* ページ)。

要素の削除

要素を選択し、**削除**コマンドを実行するか、DEL キーを押します。

以下を考慮してください。
- ステップを削除すると、関連付けられたアクションリストも削除されます。
- 初期ステップを削除すると、自動的に次のステップが初期ステップに設定されます。この**初期ステップ**オプションは、このステップのプロパティで有効になります。
- 分岐した領域の前にある水平ラインを削除すると、すべての分岐が削除されます。
- 分岐の要素をすべて削除すると、その分岐は削除されます。

SFC 要素のプロパティ

概要

要素のプロパティダイアログボックスで、SFC 要素のプロパティを表示および編集できます。**表示**メニューの**要素のプロパティ**コマンドからこのダイアログボックスを開きます。

現在選択しているプロパティによって、表示されるプロパティは異なります。

SFC エディターオプションの**ビュー**タブで、SFC チャートの要素の隣に特定のタイプのプロパティを表示するかを設定できます。

共通プロパティ

プロパティ	詳細
名前	要素名、デフォルトでは < 要素 >< 実行番号 >。例 : ステップ名 Step0、Step1、分岐名 branch0 など。
コメント	コメント要素のコメント、テキスト。例 : カウンタをリセットします。 改行するには CTRL + ENTER を押します。
シンボル	各 SFC 要素に対して、暗黙的に要素と同じ名前のフラグが作成されます。 ここで、このフラグ変数をシンボル設定にエクスポートするか、またそのシンボルをコントローラーでどのようにアクセスさせるかを指定できます。 **なし** : シンボルはシンボル設定にエクスポートされるが、コントローラーではアクセスできません。 **読込み** : シンボルはシンボル設定にエクスポートされ、コントローラーで読み込みできます。 **書き込み** : シンボルはシンボル設定にエクスポートされ、コントローラーで書き込みできます。 **読込み / 書き込み** : 読み取りと書き込みの組み合わせ。 デフォルトで、このフィールドは空のままです。つまり、シンボルはシンボル設定にエクスポートされません。

特定のプロパティ

特定のプロパティ	詳細
初期ステップ	このオプションは、現在の初期ステップ (init step) (*305* ページ) のプロパティで有効になります。初期設定では、SFC の最初のステップで有効になり、他のステップでは無効になります。別のステップでこのオプションを有効にする場合は、前の初期ステップで無効にしてください。しない場合は、コンパイラエラーが生成されます。
Duplicate on copy	このオプションは、ステップアクション (エントリーアクション、メインアクション、または終了アクション) を含むステップ、および遷移オブジェクトにリンクされている遷移に使用できます。 **Duplicate on copy** オプションが選択されている場合、ステップまたは遷移がコピーされると、呼び出された各アクション、ステップ、およびプロパティに対して新しいオブジェクトが作成されます。新しいオブジェクトには、ソースオブジェクトの実装コードのコピーが含まれています。 **Duplicate on copy** オプションが選択されていない場合、ステップまたは遷移がコピーされても、呼び出されたオブジェクトへのリンクは、関連付けられているアクション、ステップ、およびプロパティに対して保持されます。新しいオブジェクトは作成されません。ステップまたは遷移のソースとコピーは、同じアクション、ステップ、またはプロパティを呼び出します。

特定のプロパティ		詳細
Times		ステップの最小処理時間と最大処理時間を定義します。
	最小時間	次の遷移が TRUE であっても、このステップの処理にかかる最小時間。許可されている値 : IEC 構文 (例えば、t#8s) または TIME 変数に準拠した時間 ; 初期値 : t#0s。
	最大時間	このステップの処理に必要な最大時間。ステップのタイムアウトは、暗黙の変数 (*314* ページ) SFCError フラグによって示されます。許可されている値 : IEC 構文 (例えば、t#8s) または TIME 変数に準拠した時間 ; 初期値 : t#0s。
アクション		ステップが有効なときに実行するアクション (*308* ページ) を定義します。詳細については、SFC での処理のシーケンス (*318* ページ) の説明を参照してください。
	ステップの開始	ステップが有効になった後でこのアクションが実行されます。
	ステップの有効	このアクションは、ステップが有効で開始アクションがすでに処理されているときに実行されます。
	ステップの終了	このアクションは、ステップが無効になった後の次のサイクルに実行されます (終了アクション)。

注記：SFC フラグ (*314* ページ) によってアクションの状態およびタイムアウト判断するために、適切な暗黙の変数を使用してください。

SFC 要素 / ツールボックス

概要

SFC メニューのコマンドを実行して、SFC エディターウィンドウでプログラミングに使用できるグラフィック要素を挿入できます。

エディターでの作業の詳細については、*SFC エディターでの作業* (*301* ページ) の章の説明を参照してください。

次の要素が使用できます。この章で説明しています。

- 手順 (*305* ページ)
- 遷移 (*305* ページ)
- アクション (*308* ページ)
- 分岐 (代替) (*310* ページ)
- 分岐 (並列) (*310* ページ)
- ジャンプ (*311* ページ)
- マクロ (*312* ページ)

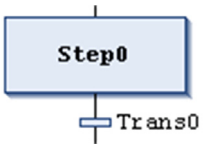
ステップ / 遷移

ステップまたは遷移を 1 つ挿入するには、**ツールボックス**の**ステップ**または**遷移**コマンドを実行します。ツールバーの**ステップ遷移を挿入** ( ) または **後にステップ遷移を挿入** ( ) コマンドで、ステップと遷移を組み合わせて挿入することもできます。

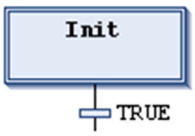
ステップは、最初に自動的に生成されたステップ名の付いたボックスで表されます。前の遷移とそれに続く遷移が線で接続されています。SFC 内の最初のステップである初期ステップのボックスの枠は二重線です。

遷移は小さな長方形で表されます。挿入後、デフォルトの名前 Trans<n> が付けられます。n は実行番号です。

ステップとそれに続く遷移の例



初期ステップとそれに続く遷移の例



ステップ名と遷移名は行内で編集できます。

ステップ名は、親 POU のスコープ内で固有にしてください。これは特に SFC でプログラミングされたアクションを使用する場合に考慮してください。そうでない場合、ビルド処理中にエラーが検出されます。

コマンド **Init step** を実行するか、対応するステップのプロパティを有効にすることで、各ステップを初期ステップに変換できます。初期ステップは、IL - POU が呼び出されたときに最初に実行されます。

各ステップは、ステップのプロパティ (*303* ページ) で定義します。

ステップを挿入した後、ステップが有効 (処理済み) なときに実行されるアクションを関連付けます。アクション (*308* ページ) についての詳細は、以下を参照してください。

遷移に関する推奨事項

遷移は、条件値が TRUE になるとすぐに次のステップが有効になる条件が必要です。従って、遷移条件の値は TRUE または FALSE にしてください。

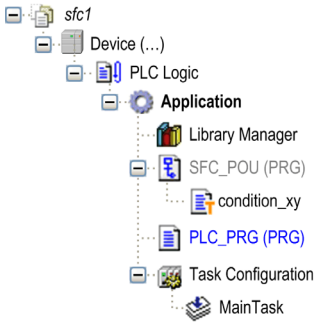
遷移条件は、以下 の 2 つの方法で定義できます。

定義のタイプ	条件のタイプ	詳細
直接	インライン	デフォルトの遷移名を次のいずれかの名前に置き換えます。 ● ブール型変数 ● ブール型アドレス ● ブール型定数 ● ブール値の結果を持つ命令 (例 : (i<100) AND b). ここでは、プログラム、ファンクションブロック、または割り当ては指定できません。
別の遷移またはプロパティオブジェクトを使用	複数使用	デフォルトの遷移名プロジェクトにある遷移 () またはプロパティオブジェクト () の名前に置き換えます。(これにより、遷移を複数使用できます。例として、下の図の condition_xy を参照してください。) インライン遷移のようなオブジェクトには、次の要素を含めることができます。 ● ブール型変数 ● アドレス ● 定数 ● 命令 ● 任意のコードの複数の命令文

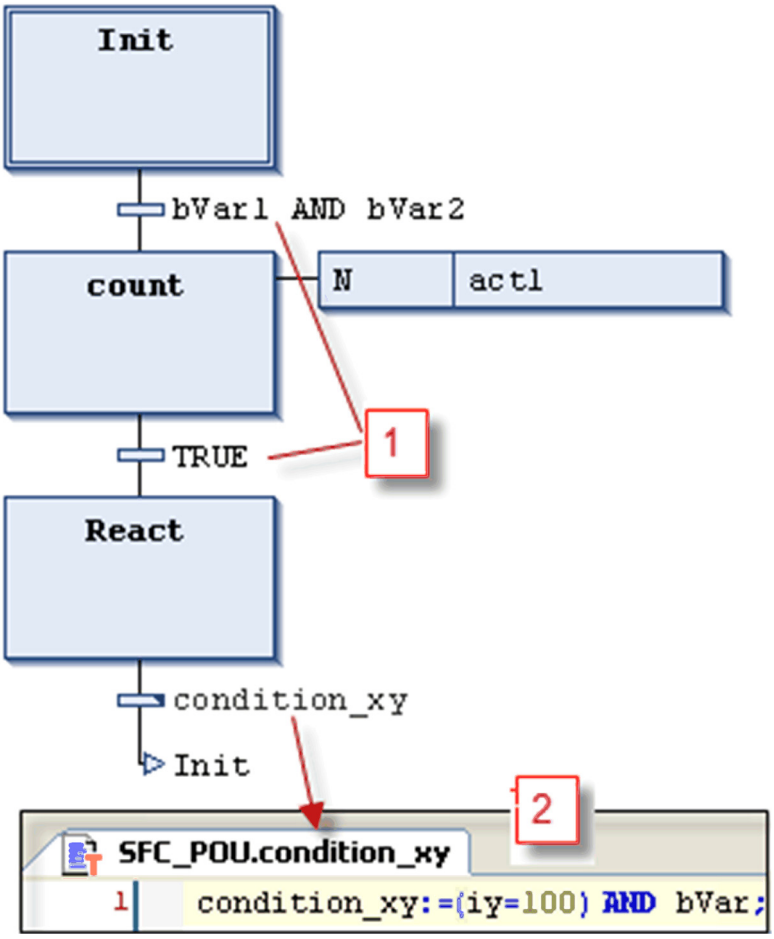
注記：遷移で複数の命令文が生成される場合は、式を遷移変数に割り当てます。

注記：遷移またはプロパティオブジェクトで構成される遷移は、長方形の右上の角に小さな三角形で示されます。

遷移オブジェクト (複数使用遷移)

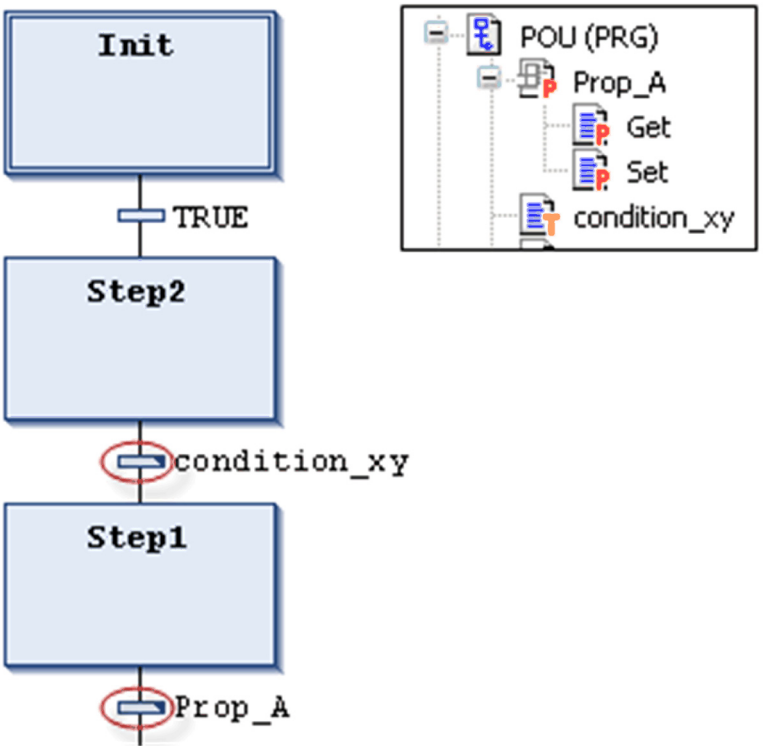


遷移の例



1 直接入力された遷移条件
2 ST でプログラミングされた遷移 condition_xy

複数使用の条件 (遷移またはプロパティ) は、三角形で表されます。



EcoStruxure Machine Expert の以前のバージョンとは異なり、遷移呼び出しはメソッド呼び出しと同様に処理されます。次の構文に従って入力します。

< 遷移名 >:=< 遷移条件 >;

例 : trans1:= (a=100)

または、単に

< 遷移条件 >;

例 : a=100

遷移の例 (condition_xy) も参照してください。

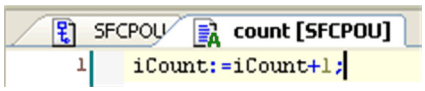
アクション

アクションには、有効なプログラミング言語のうちの 1 つで書かれた一連の命令を含めることができます。アクションはステップに割り当てられ、オンラインモードでは定義された処理順序 (*318* ページ) に従って処理されます。

SFC ステップで使用する各アクションは、SFC POU またはプロジェクト内で有効な POU として利用可能である必要があります。( )。

ステップ名は、親 POU のスコープ内で固有にしてください。アクションに割り当てられているステップと同じ名前のステップが含まれていない場合があります。そうでない場合、ビルド処理中にエラーが検出されます。

ST で書かれたアクションの例



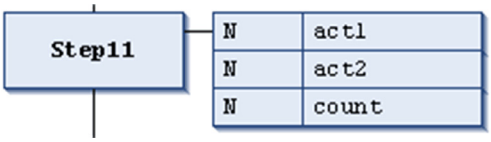
IEC 準拠および IEC 拡張のステップアクションを次の項で説明します。

IEC に準拠したステップアクション (IEC アクション)

これは、IEC61131-3 規格に準じたアクションで、ステップが有効になったとき、およびその後無効になったとき (2 回目) に修飾子 (*313* ページ) に従って処理されます。ステップに複数のアクションを割り当てた場合、アクションリストが上から下に実行されます。

- 通常のステップアクションとは違い、IEC ステップアクションには異なる修飾子を使用できます。
- 通常のステップアクションとさらに異なる点は、IEC ステップアクションには制御フラグがあることです。これにより、アクションが別のステップからも呼び出されたとしても、アクションは常に一度に 1 回のみ実行されます。通常のステップアクションの場合とは異なります。
- IEC ステップアクションは、接続線によってステップの右側に接続された二部構成のボックスによって表示されます。左側にはアクション修飾子、右側にはアクション名が表示されます。どちらもインラインで編集できます。
- IEC ステップアクションは、**アクション関連付けを挿入**コマンドを使用してステップに関連付けられます。1 つのステップに 1 つまたは複数のアクションを関連付けできます。新しいアクションの位置は、現在のカーソルの位置およびコマンドによって異なります。アクションはプロジェクトで使用できる状態にし、固有なアクション名で挿入してください (例えば、plc_prg.a1)。

ステップに関連付けられた IEC 準拠ステップアクションリスト



各アクションボックスの最初の列には修飾子が、2 番目の列にはアクション名が表示されています。

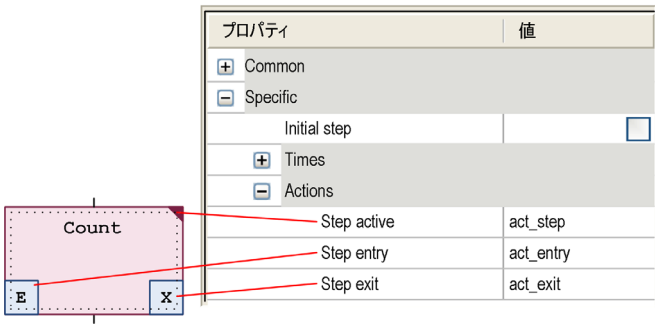
IEC 拡張ステップアクション

IEC 規格を拡張したアクションです。SFC オブジェクトの下にオブジェクトとして使用可能にしてください。固有なアクション名を選択します。これは、ステップのプロパティで定義します。

表は、IEC に拡張されたステップアクションです。

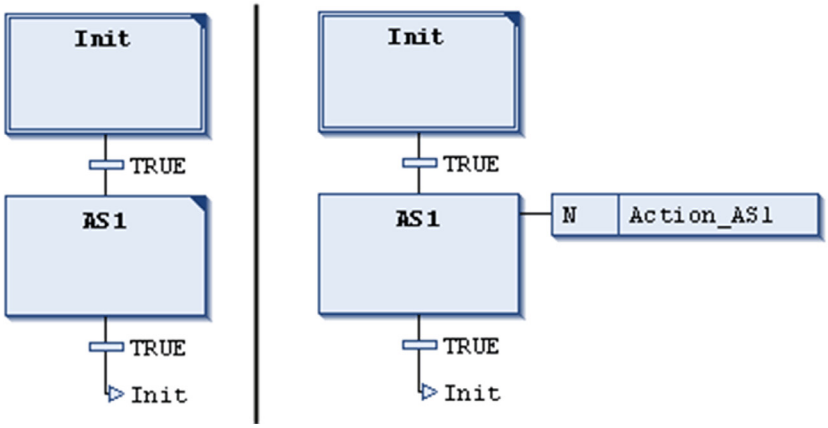
アクションタイプ	処理	関連付け	表示
ステップの開始アクション (ステップの有効化)	このタイプのステップアクションは、ステップが有効になるとすぐに、ステップの有効アクションの前に処理されます。	このアクションは、step properties の (*303* ページ) **Step entry** フィールドの入力によりステップに関連付けられます。	対応するステップボックスの左下の角に E と表示されます。
ステップの有効アクション (ステップの動作)	このタイプのステップアクションは、ステップが有効になり、このステップの有効な開始アクションが処理された後に処理されます。ただし、IEC のステップアクション (上記参照) とは異なり、無効になったときには再度実行されず、修飾子も割り当てられません。	このアクションは、step properties の (*303* ページ) **Step active** フィールドの入力によりステップに関連付けられます。	対応するステップボックスの左上の角に小さな三角形が表示されます。
ステップの終了アクション (ステップの無効化)	ステップが無効になると終了アクションが実行されます。ただし、この実行は同じサイクルではなく、次のサイクルの開始時に実行されます。	このアクションは、step properties の (*303* ページ) **Step exit** フィールドの入力によりステップに関連付けられます。	対応するステップボックスの右下の角に X と表示されます。

IEC 拡張ステップアクション



例 : IEC 適合ステップアクションと拡張ステップアクションの違い

ステップアクションと修飾子 N が付いた IEC アクションの主な違いは、IEC アクションはステップが有効になったときと、ステップが無効になったときの少なくとも 2 回実行されることです。次の例を参照してください。



アクション Action_AS1 はステップアクションとしてステップ AS1 に関連付けられている (左側)、または修飾子 N 付きの IEC アクションとしてステップ AS1 に関連付けられている (右側) とします。どちらの場合も、2 つの遷移が使用されているため、初期ステップに再度到達するまでに 2 サイクルかかり、変数 iCounter は Action_AS1 でインクリメントされると仮定します。ステップ Init が再度有効になった後、左側の例の iCounter は値 1 になります。ただし、AS1 が無効になり IEC アクションが 2 回実行されたため、右側の値は 2 です。

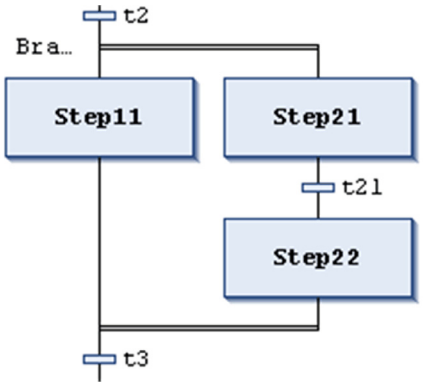
修飾子の詳細については、使用可能な修飾子リスト (*313* ページ) を参照してください。

IEC アクションとは対照的に、ステップアクションは対応するステップからしか呼び出せないように埋め込むことができます。このステップをコピーすると、新しいアクションオブジェクトが自動的に作成され、実装コードが複製されます。ステップに最初のアクションを挿入したときに表示されるダイアログボックスで、ステップ参照アクションの複製モードを**参照をコピー**または **Copy implementation** として選択します。後で、**要素プロパティ**ダイアログボックスの **Duplicate on copy** オプション (*303* ページ) を使用して複製モードを選択できます。**ツール → オプション → SFC エディター** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で、一般的なオプションとして **Default insertion method** を設定することもできます。

分岐

シーケンシャルファンクションチャート (SFC) は分岐できます。つまり、処理線を 2 つまたそれ以上の複数の線に分岐できます。並列分岐 (*310* ページ) は、並列 (同時) に処理されます。代替分岐 (*310* ページ) の場合、前の遷移条件により 1 つのみ処理されます。チャート内の各分岐は、水平の二重線 (並列) または単線 (代替) で始まり、同じ線またはジャンプ (*311* ページ) で終了します。

並列分岐



並列分岐は、ステップで開始および終了してください。並列分岐には、入れ子になった代替分岐、またはその他の並列分岐を入れることができます。

分岐した領域の前後の水平ラインは二重線です。

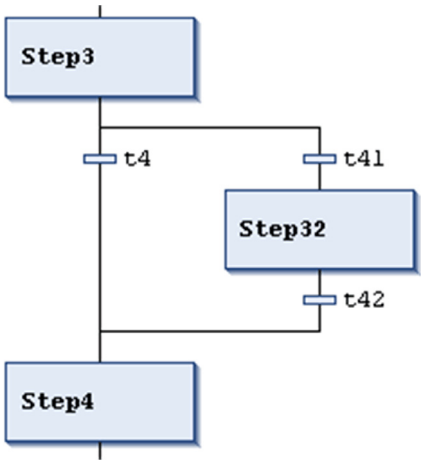
オンラインモードでの処理 : 前の遷移 (左の例では t2) が TRUE の場合、すべての並列分岐の初期ステップが有効になります (Step11 および Step21)。特定の分岐は、次の遷移 (t3) が認識される前にお互い並列に処理されます。

並列分岐を挿入するには、ステップを選択して **右に分岐を挿入**コマンドを実行します。

並列および代替分岐は、**並列**または**代替**コマンドを実行することによってお互いに変換できます。

分岐の前の水平線に Branch<n> という名前の分岐ラベルが自動的に追加されます。n は 0 から始まる実行番号です。ジャンプ先 (*311* ページ) を定義するときにこのラベルを指定できます。

代替分岐



分岐した領域の前後の水平ラインは単線です。

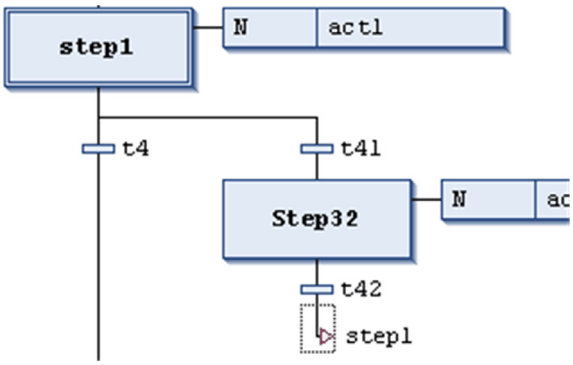
代替分岐は、遷移で開始および終了してください。代替分岐は、並列分岐および他の代替分岐を入れることができます。

代替分岐の開始線の前のステップが有効である場合、各代替分岐の最初の遷移は左から右に評価されます。遷移条件が TRUE である一番左の遷移が開かれ、以下のステップが有効になります。

並列分岐を挿入するには、遷移を選択して **右に分岐を挿入**コマンドを実行します。

並列および代替分岐は、**並列**または**代替**コマンドを実行することによってお互いに変換できます。

ジャンプ



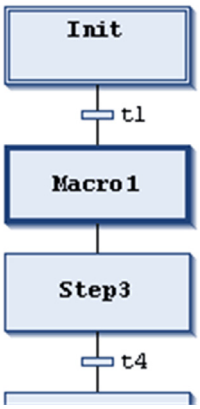
ジャンプは、垂直な接続線と水平な矢印、およびジャンプ先の名前で表示されます。前の遷移が TRUE になるとすぐに処理される次のステップを定義します。ジャンプを使用して、処理線が交差または上方向に進むことを避けることができます。

チャートの最後のデフォルトのジャンプ以外では、ジャンプは分岐の最後でのみ使用できます。ジャンプを挿入するには、分岐の最後の遷移を選択して**ジャンプを挿入**コマンドを実行します。

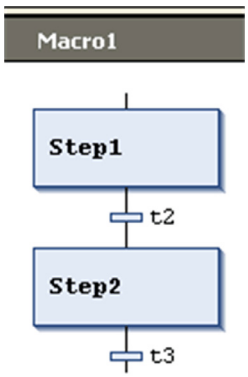
ジャンプ先は、関連付けられた編集可能なテキスト文字列で指定します。ステップ名または並列分岐のラベルを指定できます。

マクロ

メイン SFC エディタービュー



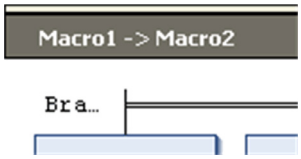
Macro1 のマクロエディタービュー



マクロは、マクロ名を含む太枠のボックスで表示されます。これには SFC チャートの一部が含まれているため、メインエディタービューには直接表示されません。マクロを使用することによる処理フローへの影響はなく、例えば、表示を簡素化するためにプログラムの一部を非表示にするなどの手段にすぎません。マクロボックスを挿入するには、**後にマクロを挿入**コマンドを実行します。マクロ名は編集できます。

マクロエディターを開くには、マクロボックスをダブルクリックするか、**マクロにズームイン**コマンドを実行します。ここで、メインエディタービューと同様に編集し、SFC チャートの必要なセクションを入力できます。そこから出るには、**マクロからズームアウト**コマンドを実行します。

マクロエディターのタイトル行には、現在の SFC のマクロのパスが表示されます。例 :



SFC におけるアクションの修飾子

概要

アクション (*308* ページ) をどのように IEC ステップに関連付けるかを設定するための修飾子を使用できます。修飾子はアクション要素の修飾子フィールドに挿入されます。

使用可能な修飾子

修飾子	詳細	説明
N	non-stored (未保存)	ステップが有効である限り、アクションは有効です。
R0	overriding reset (オーバーライドリセット)	アクションは無効になります。
S0	set (stored) (設定 (保存))	ステップが有効になるとアクションが開始され、ステップが無効になった後もアクションがリセットされるまでアクションが続行します。
L	time limited (時間制限)	ステップが有効になるとアクションが開始します。ステップが無効になるか、設定時間が経過するまで続行します。
D	time delayed (時間遅延)	ステップが有効になると遅延タイマーが開始されます。時間遅延後にステップがまだ有効である場合、アクションが開始され、無効になるまで続行します。 **注記：**2 つの連続するステップに同じブール変数を設定する D (時間遅延) アクションがある場合、この変数はステップからもう 1 つのステップへの遷移中この変数はリセットされません。変数をリセットするには、2 つのステップ間に中間ステップを挿入します。
P	pulse (パルス)	ステップが有効 / 無効になるとアクションが開始され、1 回のみ実行します。
SD	stored and time delayed (保存および時間遅延)	設定した時間遅延が経過した後でアクションが開始され、リセットされるまで続行します。
DS	delayed and stored (遅延および保存)	指定した時間遅延後にステップがまだ有効である場合、アクションが開始され、リセットされるまで続行します。
SL	stored and time limited (保存および時間制限)	ステップが有効になるとアクションが開始され、指定した時間中、またはリセットされるまで続行します。

修飾子 L、D、SD、DS、および SL は、TIME 定数フォーマットの時間の値が必要です。修飾子の直後に空白で区切って入力します (例 L T#10s)。

注記：IEC アクションが無効な場合は、もう 1 度実行されます。これは、各アクションが少なくとも 2 回実行されることを意味します。

暗黙の変数 - SFC フラグ

概要

各 SFC ステップおよび IEC アクションには、実行時にステップおよび IEC アクションのステータス (*314* ページ) を監視するために生成された暗黙の変数があります。また、SFC の実行 (タイムアウト、リセット、チップモード) を監視および制御するための変数も定義できます。これらの変数は、SFC オブジェクトによって暗黙的に生成することもできます。

基本的に、各ステップおよび各 IEC アクションごとに暗黙の変数が生成されます。要素用に名前が付けられた構造体インスタンス、例えば、ステップ名 **step1** のステップの場合は step1 です。要素のプロパティ (*303* ページ) で、このフラグのシンボル定義をシンボル設定にエクスポートするか、またこのシンボルをコントローラーでどのようにアクセスするかを定義できます。

これらの暗黙の変数のデータ型は、ライブラリー IecSFC.library で定義されています。SFC オブジェクトが追加されるとすぐに、このライブラリーがプロジェクトに自動的に含まれます。

ステップとアクションのステータスおよびステップ時間

基本的に、各ステップおよび各 IEC アクションごとに SFCStepType または SFCActionType 型の暗黙の構造体変数が生成されます。構造体のコンポーネント (フラグ) は、ステップまたはアクションのステータス、または有効なステップの現在の処理時間を表します。

暗黙的に実行される変数宣言の構文は次のとおりです。

< ステップ名 >: SFCStepType;

または

_< アクション名 >:SFCActionType;

注記：IEC アクションの暗黙の変数の前にはアンダースコアが付きます。

ステップまたはアクションステータスのブール型フラグは次のとおりです。

ステップのブール型フラグ

ブール型フラグ	詳細
< ステップ名 >.x	現在の有効化ステータスを示します。
< ステップ名 >._x	次のサイクルの有効化ステータスを示します。

< ステップ名 >.x = TRUE の場合、現在のサイクルでステップが実行されます。

< ステップ名 >._x = TRUE かつ < ステップ名 >.x = FALSE の場合、次のサイクルでステップが実行されます。これは、サイクルの開始時に < ステップ名 >._x が < ステップ名 >.x にコピーされることを意味します。

アクションのブール型フラグ

ブール型フラグ	詳細
_< アクション名 >.x	アクションが実行されると TRUE になります。
_< アクション名 >._x	アクションが有効になると TRUE になります。

シンボルの生成

ステップまたはアクションの要素プロパティ (*303* ページ) で、ステップまたはアクション名フラグ用に作成およびダウンロードされるシンボル設定にシンボル定義を追加するかを定義できます。そのためには、要素プロパティビューの**シンボル**列に目的のアクセス権を入力します。

注記：ブール型フラグ < ステップ名 >.x を使用してステップに特定のステータス値 (ステップを有効に設定) を強制する場合は、これによって SFC 内の制御されていないステータスに影響することに注意してください。

⚠ 警告

装置の意図しない動作

ステップをアクティブに設定するためにステータス値を強制する、そのためにブールフラグ < ステップ名 >.x を使用しないでください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

TIME 変数を介した時間:

フラグ t は、ステップが有効になってから経過した現在の時間を示します。これはステップのみに適用され、ステップ属性 (*303* ページ) に最小時間が設定されているかに関わりません (以下参照 : SFCError)。

ステップの場合

< ステップ名 >.t (< ステップ名 >._t は外部目的には使用できません)

アクションの場合

暗黙の時間変数は使用されません。

SFC 実行の制御 (タイムアウト、リセット、ヒントモード)

SFC の動作を制御するために、SFC フラグとも呼ばれる 暗黙的に使用可能な変数 (以下の表を参照) を使用できます。例えば、時間のオーバーフローの表示、または遷移の切り替え用のチップモードを有効にするために使用できます。

これらのフラグにアクセスできるようにするには、それらを宣言し、有効にしてください。**SFC 設定**ダイアログボックスで行います。それは、オブジェクトの**プロパティ**ダイアログボックスのサブダイアログボックスです。

SoMachine / SoMachine Motion V3.1 で必要であった手動宣言は、別の POU からの書き込みアクセスを有効にする場合のみ必要です (*フラグへのアクセス* の項を参照してください)。

その場合、次の点を考慮してください。

フラグをグローバルに宣言する場合は、**SFC 設定**ダイアログボックスの**宣言**オプションを有効にします。そうでない場合、暗黙的に宣言されたローカルフラグになり、グローバルなフラグの代わりに使用されます。SFC POU の SFC 設定は、初め**オプション → SFC** ダイアログボックスで設定された定義によって決まることに留意してください。

SFC 設定ダイアログボックスでのみフラグ変数の宣言を行った場合は、SFC POU のオンラインビューでのみ表示されることを考慮してください。

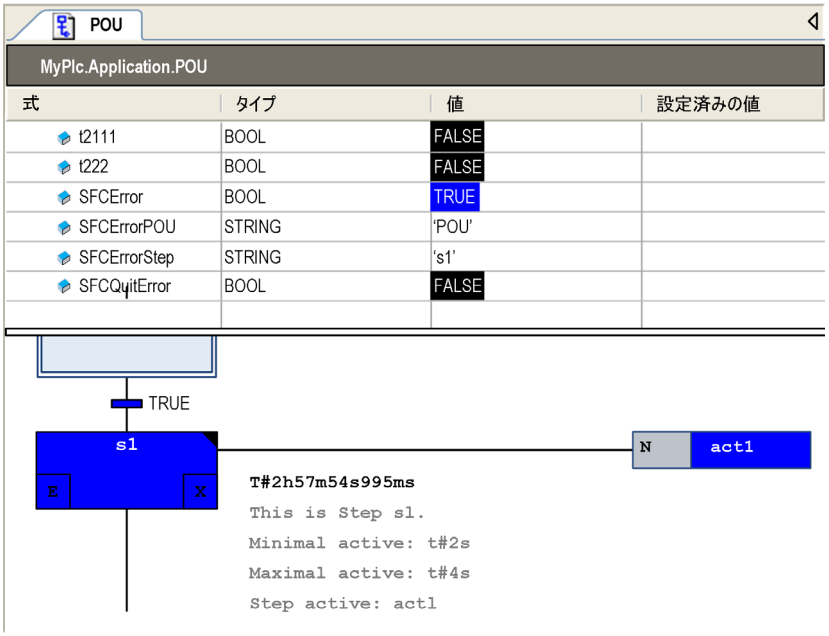
次の暗黙の変数 (フラグ) を使用できます。そのためには、**SFC 設定**ダイアログボックスで宣言し、有効にしてください。

変数	タイプ	詳細
SFCInit	BOOL	この変数が TRUE になると、シーケンシャルファンクションチャートは初期ステップ (*305* ページ) に戻ります。すべてのステップとアクション、およびその他の SFC フラグがリセット (初期化) されます。初期ステップは有効なままですが、変数が TRUE である限り実行されません。通常の処理に戻すには、SFCInit を FALSE に戻します。
SFCReset	BOOL	この変数は、SFCInit と似た動作をします。ただし後者とは異なり、初期ステップの初期化後にさらに処理が行われます。そのため、この場合は初期ステップで SFCReset フラグを FALSE にリセットできます。
SFCError	BOOL	SFC のステップの 1 つでタイムアウトが発生すると、この変数は TRUE になります。前提条件 : SFCEnableLimit は TRUE である必要があります。 SFCError をリセットする前にそれ以上のタイムアウトは登録できないことに考慮してください。他の時間制御フラグ (SFCErrorStep、SFCErrorPOU、SFCQuitError) を使用する場合は、SFCError を定義してください。
SFCEnableLimit	BOOL	この変数を使用して、SFCError によるステップでの時間制御を暗黙的に有効 (TRUE) または無効 (FALSE) にできます。これは、この変数が宣言されて有効 (**SFC 設定**) になっている場合、SFCError を動作させるために TRUE に設定する必要があることを意味します。そうでない場合、ステップのタイムアウトは登録されません。起動時または手動操作時に使用することが適当です。変数が定義されていない場合、SFCError は自動的に動作します。 前提条件 : SFCError が定義されている必要があります。
SFCErrorStep	STRING	この変数は、SFCError.timeout で登録されたタイムアウトでステップの名前を格納します。 前提条件 : SFCError が定義されている必要があります。

変数	タイプ	詳細
SFCErrorPOU	STRING	この変数は、タイムアウトが発生した SFC POU の名前を格納します。 前提条件 : SFCError が定義されている必要があります。
SFCQuitError	BOOL	この変数が TRUE である限り、SFC 図の実行は停止し、変数 SFCError はリセットされます。変数が FALSE にリセットされるとすぐに、有効なステップの現在の時間状態がすべてリセットされます。 前提条件 : SFCError が定義されている必要があります。
SFCPause	BOOL	この変数が TRUE である限り、SFC 図の実行は停止します。
SFCTrans	BOOL	遷移が動作するとすぐにこの変数は TRUE になります。
SFCCurrentStep	STRING	この変数は、時間監視とは関係なく現在有効なステップの名前を格納します。同時シーケンスの場合、すぐ外側のステップの名前が登録されます。
SFCTip SFCTipMode	BOOL	これらの変数によって、現在のチャート内でインチングモードを使用できます。SFCTipMode = TRUE によってこのモードのスイッチが入っている場合、SFCTip = TRUE (立上がり) に設定することによってのみ次のステップにスキップできます。SFCTipMode が FALSE に設定されている限り、遷移によってスキップできます。

次の図は、エディターのオンラインモードで SFC が検出したエラーフラグの例です。

SFCError フラグによって SFC オブジェクト **POU** のステップ s1 でタイムアウトが検出されました。



フラグへのアクセス

SFC 実行 (タイムアウト、リセット、ヒントモード) の制御のためのフラグにアクセスするには、上記 (SFC 実行の制御 (*315* ページ)) のようにフラグ変数を宣言して有効にします。

SFC POU 内でアクションまたは遷移からアクセスする場合の構文

< ステップ名 >.< フラグ >

または

_< アクション名 >.< フラグ >

例 :

```
status:=step1._x;
checkerror:=SFCerror;
```

別の POU からアクセスする場合の構文

<SFC POU>.< ステップ名 >.< フラグ >

または

<SFC POU>._< アクション名 >.< フラグ >

例 :

```
status:=SFC_prog.step1._x;
checkerror:=SFC_prog.SFCerror;
```

別の POU からの書き込みアクセスの場合は、次の点を考慮してください。

- 暗黙の変数は、さらに SFC POU の VAR_INPUT 変数として明示的に宣言してください。
- または、GVL (グローバル変数リスト) でグローバルに宣言してください。

例 : 位置の宣言

```
PROGRAM SFC_prog
VAR_INPUT
  SFCinit:BOOL;
END_VAR
```

例 : GVL でのグローバル宣言

```
VAR_GLOBAL
  SFCinit:BOOL;
END_VAR
```

PLC_PRG のフラグへのアクセス

```
PROGRAM PLC_PRG
VAR
  setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;  //Write access to SFCinit in SFC_prog
```

SFC での処理のシーケンス

概要

オンラインモードでは、定義されたシーケンスに従って特定のアクションタイプが処理されます。次の表を参照してください。

用語の定義

次の用語を使用します。

用語	詳細
有効なステップ	ステップアクションが実行されているステップ。 オンラインモードでは、有効なステップは青色で塗りつぶされます。
初期ステップ	SFC POU が呼び出された後の最初のサイクルでは、初期ステップが自動的に有効になり、関連付けされたステップアクション (*308* ページ) が実行されます。
IEC アクション	IEC アクションは少なくとも 2 回実行されます。 ● 1 回目は、初めて有効になったとき。 ● 2 回目は、次のサイクルで無効になったとき。
代替分岐	代替分岐の水平な開始ラインの前のステップが有効である場合、各分岐の最初の遷移は左から右に評価されます。遷移条件が TRUE である一番左の遷移が検索され、各分岐が実行されます。つまりこの分岐内の次のステップが有効になります。
並列分岐	並列分岐の開始ラインの二重線が有効であり、前にある遷移条件の値が TRUE の場合、すべての並列分岐の第 1 ステップが有効になります。分岐は、相互に並行して処理されます。分岐の終了の二重線に続くステップは、前のステップがすべて有効であり、二重線の後の遷移条件が TRUE のときに有効になります。

処理の順序

シーケンスの要素の処理順序

手順	詳細
1. IEC アクションのリセット	IEC アクション (*308* ページ) のすべてのアクション制御フラグをリセットします (ただし、アクション内で呼び出される IEC アクションのフラグ以外)。
2. ステップの終了アクション (ステップの無効化)	すべてのステップをシーケンスチャートで想定されている順序 (上から下、左から右) でチェックし、ステップの終了アクションの要件が満たされているかを判別します。満たされている場合、終了アクションが実行されます。ステップが無効 (*305* ページ) になる場合、終了アクションが実行されます。これは、開始およびステップアクション (が存在する場合) が前回のサイクルで実行され、次のステップの遷移が TRUE である場合を意味します。
3. ステップの開始アクション (ステップの有効化)	すべてのステップをシーケンスの順序でテストし、ステップ開始アクションの実行要件が満たされているかを判別します。満たされている場合、開始アクションが実行されます。ステップ前の遷移条件が TRUE でありステップが有効な場合は開始アクションが実行されます。
4. タイムアウトチェック、ステップ有効アクション	また、IEC ではないステップの場合、シーケンスでの位置の順序 (上から下、左から右) で対応するステップ開始アクションが実行されます。
5. IEC アクション	シーケンスで使用される IEC アクション (*308* ページ) はアルファベット順に実行されます。これは、アクションリストを 2 回通過して行われます。最初の通過で、現在のサイクルで無効な IEC アクションがすべて実行されます。2 回目の通過で、現在のサイクルで有効な IEC アクションがすべて実行されます。
6. 遷移チェック、次のステップの有効化	遷移 (*305* ページ) が評価されます。現在のサイクルのステップが有効であり、次に続く遷移が TRUE を返した場合 (および最小有効化時間が適用されている場合は、最小有効化時間が既に経過している場合)、次のステップが有効になります。

注記：複数のステップが有効な場合、アクションが他の複数の IEC アクションから呼び出されるため、1 つのサイクルでアクションが複数回実行される場合があります。つまり、同じ IEC アクションが SFC の異なるレベルで同時に使用され、これにより意図しない影響を引き起こす可能性があります。
例 : SFC に 2 つの IEC アクション A と B があり、両方が SFC に実装され、IEC アクション C を呼び出します。IEC アクション A と B は両方同じサイクルで有効になり、さらに両方のアクションで IEC アクション C が有効になります。そのため C は 2 回呼び出されます。

⚠ 警告
装置の意図しない動作 同じサイクルで IEC を複数の他の IEC アクションから呼び出さないでください。 **上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。**

注記：ステップおよびアクションの状態、またはチャートの実行を決定するには暗黙の変数 (*314* ページ) を使用してください。

オンラインモードの SFC エディター

概要

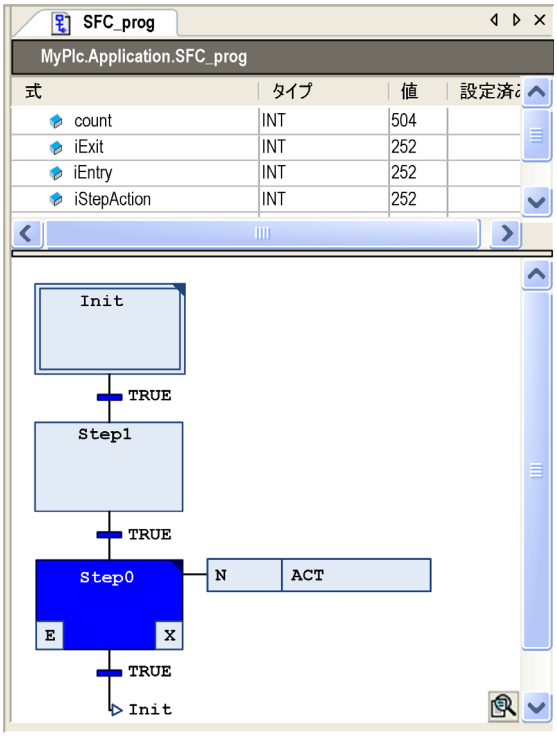
オンラインモードの SFC エディターでは、監視用 (以下を参照) およびコントローラーの変数と式の書き込みと強制用のビューが表示されます。SFC には、他の IEC 言語のようなデバッグ機能 (ブレークポイント、ステップなど) はありません。ただし、SFC のデバッグには次を考慮してください。

- オンラインモードでのオブジェクトの開き方については、オンラインモードのユーザーインターフェイス (*43* ページ) の説明を参照してください。
- SFC オブジェクトのエディターウィンドウには、上部に宣言エディターが含まれています。一般情報については、*オンラインモードの宣言エディター* (*346* ページ) の章を参照してください。SFC **設定**ダイアログボックスで暗黙の変数 (SFC フラグ) (*314* ページ) を宣言した場合はここに追加されますが、オフラインモードの宣言エディターには表示されません。
- シーケンシャルファンクションチャートの要素の処理順序 (*318* ページ) を考慮してください。
- オブジェクトプロパティまたは SFC エディターオプション、およびコンパイルに関する設定の SFC デフォルトまたは SFC 要素とその属性のオンライン表示を参照してください。
- SFC の処理を監視および制御するためのフラグ (*314* ページ) の使用を検討してください。

監視

有効なステップは、青色で塗りつぶされて表示されます。ステップ属性の表示は、設定された SFC エディターのオプションによって異なります。

プログラミングオブジェクト SFC_prog のオンラインビュー



第 14 章
構造化テキスト (ST) エディター

この章について

この章には次のセクションが含まれています。

14.1
ST エディターの情報

このセクションについて

このセクションには次の項目が含まれています。

ST エディター

概要

ST エディターは、IEC プログラミング言語の構造化テキスト (ST) または IEC 61131-3 規格への拡張である拡張構造化テキストプログラミングオブジェクトを作成するために使用します。

ST エディターはテキストエディターです。従って、**オプション**および**カスタマイズ**ダイアログボックスの対応するテキストエディター設定を使用して、動作、外観、およびメニューを設定します。そこで、強調表示の色、行番号、タブ、インデント、その他多数のオプションの初期設定を定義できます。

詳細

マウスでテキスト領域を選択しながら ALT キーを押すことによってブロックを選択できます。

エディターはウィンドウの下部にあり、上部には宣言エディター (*342* ページ) があります。

編集中に構文エラーが検出されると、**プリコンパイルメッセージ**ウィンドウに対応するメッセージが表示されます。このウィンドウは、エディターウィンドウのフォーカスがリセットされるたびに更新されます (例えば、別のウィンドウにカーソルを置いた後、エディターウィンドウに戻るなど)。

オンラインモードの ST エディター

概要

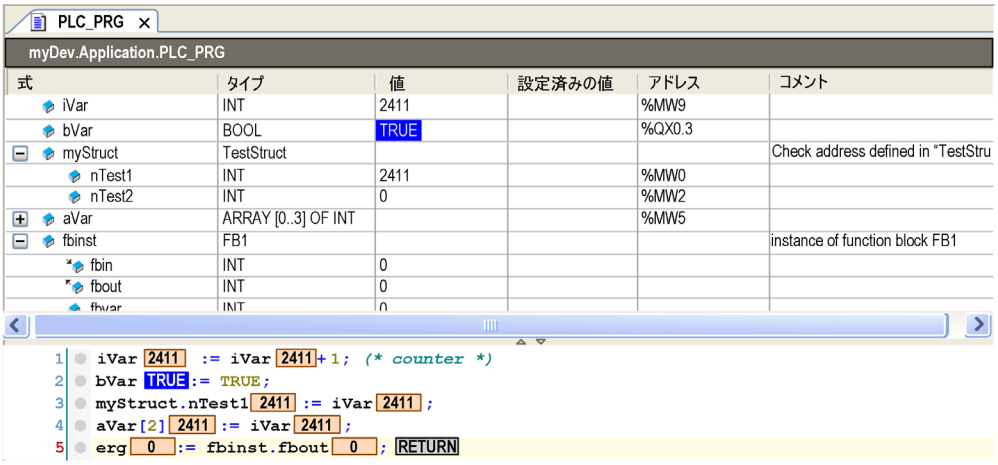
オンラインモードの構造化テキストエディター (ST エディター) では、監視 (*324* ページ) 用およびコントローラーの変数と式の書き込みと強制用のビューが表示されます。デバッグ機能 (ブレークポイント、ステップなど) を使用できます。ST エディターのブレークポイント位置 (*325* ページ) を参照してください。

- オンラインモードでのオブジェクトの開き方については、オンラインモードのユーザーインターフェイス (*43* ページ) の説明を参照してください。
- オンラインモードで変数の設定済みの値を入力する方法については、*変数の強制* (*324* ページ) を参照してください。
- ST オブジェクトのエディターウィンドウには、上部の宣言エディターも含まれています。オンラインモードの宣言エディターについては、オンラインモードの宣言エディター (*346* ページ) を参照してください。

監視

監視が**オプション**ダイアログボックスで明示的に無効になっていない場合、各変数の後ろの小さな監視ボックスに実際の値が表示されます。

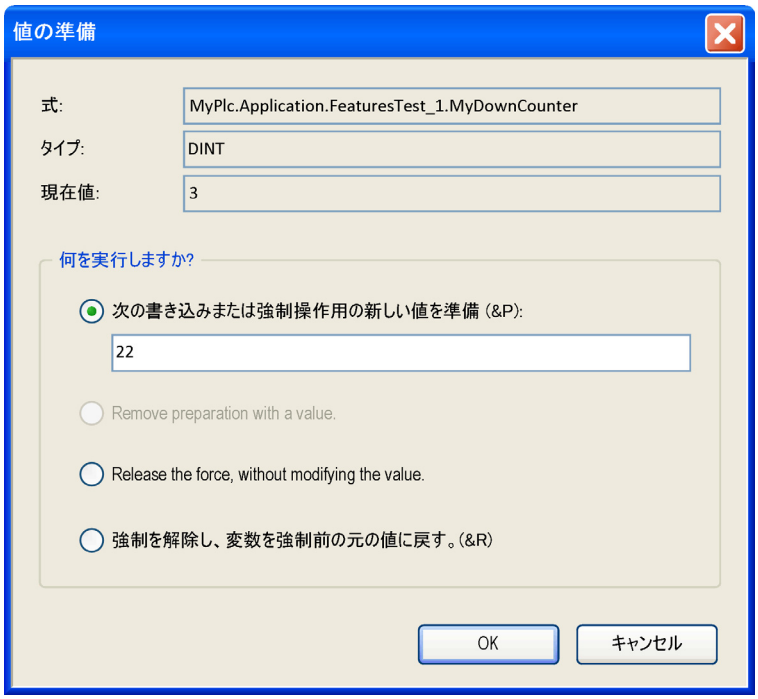
監視付きプログラミングオブジェクト PLC_PRG のオンラインビュー



変数の強制

どのエディターでも宣言内に変数の設定済みの値を入力できることに加え、ST エディターでは実装部内の変数の監視ボックスをダブルクリックすることもできます (オンラインモード)。表示されたダイアログボックスに設定済みの値を入力します。

値の準備ダイアログボックス



変数の名前と**デバイスツリー** (**式**) 内の絶対パス、タイプ、および現在値が表示されます。

対応する項目を有効にすると、次のオプションを選択できます。

- 編集フィールドに入力する新しい値を準備
- 設定済みの値を削除
- 強制されている変数を解除
- 強制されている変数を解除し、強制前に割り当てられた値にリセット

選択したアクションを実行するには、**デバッグ → 値の強制** (**オンライン**項目) コマンドを実行するか、F7 キーを押します。

ST エディターのブレークポイント位置

ブレークポイントは、基本的に POU 内の変数の値が変更する位置、またはプログラムフローが分岐する位置、または他の POU が呼び出される位置に配置できます。以下の説明では、{BP} がブレークポイントを配置できる位置を示します。

代入 :

行の開始位置。式として代入 (*329* ページ) は、行内にそれ以上ブレークポイントの位置を定義しないことに留意してください。

FOR- ループ :

1. カウンターの初期化前
2. カウンターのテスト前
3. 命令文の前

```
{BP} FOR i := 12 TO {BP} x {BP} BY 1 DO
{BP} [statement1]
...
{BP} [statementn-2]
END_FOR
```

WHILE- ループ :

1. 条件確認の前
2. 命令の前

```
{BP} WHILE i < 12 DO
{BP} [statement1]
...
{BP} [statementn-1]
END_WHILE
```

REPEAT - ループ :

- 条件確認の前

```
REPEAT
{BP} [statement1]
...
{BP} [statementn-1]
{BP} UNTIL i >= 12
END_REPEAT
```

プログラムまたはファンクションブロックの呼び出し :

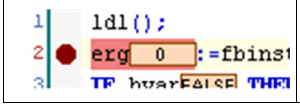
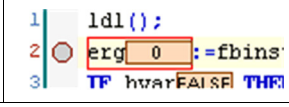
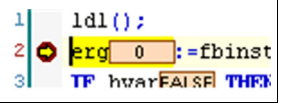
行の開始位置。

```
{{BP} POU( );
```

POU の最後 :

ステップが通ると、RETURN 命令の後にこの位置にも到達します。

ST のブレークポイントの表示

オンラインモードのブレークポイント	無効なブレークポイント	ブレークポイントでのプログラム停止
		

注記：ブレークポイントは、呼び出される可能性があるすべてのメソッドに自動的に設定されます。インターフェイス管理メソッドが呼び出されると、そのインターフェイスを実装するファンクションブロックのすべてのメソッドと、メソッドをサブスクライブするすべての派生ファンクションブロックにブレークポイントが設定されます。ファンクションブロックのポインターによってメソッドが呼び出される場合、ブレークポイントはファンクションブロックのメソッドとメソッドをサブスクライブしているすべての派生ファンクションブロックに設定されます。

14.2
構造化テキスト (ST)

このセクションについて

このセクションには次の項目が含まれています。

構造化テキスト (ST)

概要

構造化テキストは、PASCAL または C に類似したテキスト形式の高水準言語です。プログラムコードは式 (*329* ページ) と命令 (*331* ページ) で構成されています。IL (命令リスト) とは異なり、プログラミングループに多数の構造を使用でき、複雑なアルゴリズムの開発ができます。

例

```
IF value < 7 THEN
 WHILE value < 8 DO
  value:=value+1;
 END_WHILE;
END_IF;
```

式

概要

式は、評価の後に値を返す構造です。この値は命令で使用されます。

式は、演算子 (*593* ページ)、オペランド (*677* ページ)、および代入で構成されています。オペランドは、定数、変数、ファンクションの呼び出し、または別の式です。

例

33	(* 定数 *)
ivar	(* 変数 *)
fct(a,b,c)	(* ファンクションの呼び出し *)
a AND b	(* 式 *)
(x*y) / z	(* 式 *)
real_var2 := int_var;	(* 代入、以下を参照 *)

処理の順序

式の評価は、ある規則に従って演算子を処理するこで実行されます。処理の順序が一番上の演算子が最初に処理され、その後に次の処理レベルの演算子が処理されていくように、すべての演算子が処理されるまで続きます。

以下は、ST 演算子の処理レベル順の表です。

処理	シンボル	処理レベル
括弧内に配置	(式)	最高順位
ファンクションの呼び出し	ファンクション名 (パラメーターリスト)	.....
累乗	EXPT	.....
否定	–	.....
補数の作成	NOT	.....
乗算	*	.....
除算	/	.....
剰余	MOD	.....
加算	+	.....
減算	–	.....
V の比較	<,>,<=,>=	.....
等号	=	....
等号否定	<>	...
ブール型 AND	AND	..
ブール型 XOR	XOR	.
ブール型 OR	または	最低順位

式としての代入

IEC 61131-3 規格 (ExST) の拡張では、代入を式として使用できます。

例 :

int_var1 := int_var2 := int_var3 + 9;	(* int_var1 および int_var2 は両方 int_var3 + 9 の値と等しい *)
real_var1 := real_var2 := int_var;	(* 補正代入、real_var1 および real_var2 は int_var の値を取得 *)
int_var := real_var1 := int_var;	(* タイプ不一致 real-int のため メッセージが表示されます *)

IF b := (i = 1) THEN
i := i + 1;
END_IF
	(* 式が IF 条件文内で使用： i が 1 であるか否かによって最初の b が TRUE または FALSE に割り当てられ、その後 b の結果値が評価されます *)

使い方

概略

命令には、与えられた式 (*329* ページ) で何をするかが記述されています。

ST では以下の命令を使用できます。

命令	例
代入 (*331* ページ)	`A:=B; CV := CV + 1; C:=SIN(X);`
ファンクションブロック (*332* ページ) の呼び出しおよびファンクションブロックの出力の使用	`CMD_TMR(IN := %IX5, PT := 300);` `A:=CMD_TMR.Q`
RETURN（*332* ページ）	`RETURN;`

IF（*333* ページ）	`D:=B*B;`
	`IF D<0.0 THEN`
	`C:=A;`
	`ELSIF D=0.0 THEN`
	`C:=B;`
	`ELSE`
	`C:=D;`
	`END_IF;`

CASE（*333* ページ）	`CASE INT1 OF`
	`1: BOOL1 := TRUE;`
	`2: BOOL2 := TRUE;`
	`ELSE`
	`BOOL1 := FALSE;`
	`BOOL2 := FALSE;`
	`END_CASE;`

FOR（*334* ページ）	`J:=101;`
	`FOR I:=1 TO 100 BY 2 DO`
	`IF ARR[I] = 70 THEN`
	`J:=I;`
	`'EXIT;`
	`END_IF;`
	`END_FOR;`

WHILE（*335* ページ）	`J:=1;`
	`WHILE J<= 100 AND ARR[J] <> 70 DO`
	`J:=J+2;`
	`END_WHILE;`

REPEAT（*335* ページ）	`J:=-1;`
	`REPEAT`
	`J:=J+2;`
	`UNTIL J= 101 OR ARR[J] = 70`
	`END_REPEAT;`

EXIT（*336* ページ）	`EXIT;`
CONTINUE（*336* ページ）	`CONTINUE;`

JMP（*336* ページ）	`label1: i:=i+1;`
	`IF i=10 THEN`
	`JMP label2;`
	`END_IF`
	`JMP label1;`
	`label2:`

empty 命令	`;`

代入演算子

デフォルト割り当て

代入の左側には、代入演算子 := によって右側の式の値が代入されるオペランド (変数、アドレス) があります。

また、同じ機能の MOVE 演算子 (*601* ページ) の説明も参照してください。

割り当ての例については、*ファンクションブロックの呼び出し* の章の *割り当てを伴う呼び出しの例* の項を参照してください (*159* ページ)。

例

a:= b;

a は b の値を取得します。

セット演算子 S=

値がセットされます。一度 TRUE に設定されると、TRUE のまま維持されます。

例

a S= b;

割り当て処理中 b が TRUE の場合、a は TRUE に設定されます。

割り当て処理中 b が FALSE の場合、a の値は変更されません。

リセット演算子 R=

値がリセットされます。一度 FALSE に設定されると、FALSE のまま維持されます。

例

a R= b;

b = TRUE になるとすぐに、a が FALSE に設定されます。

注記：複数代入の場合、セットおよびリセットの代入は代入の最後のメンバーを参照します。

例

a S= b R= fun1(par1,par2)

この場合、b は fun1 のリセット出力値になります。ただし、a は b のセット値は取得せず、fun1 のセット出力値を取得します。

代入は式 (*329* ページ) として使用できると考えてください。これは IEC 61131-3 規格への拡張です。

Assignment operator REF

この演算子は値へのリファレンスを生成します。

構文

REF= <variable>

例

```
A : REFERENCE TO DUT;
B : DUT;
C : DUT;
A REF= B; // corresponds to A := ADR(B);
A := C; // corresponds to A^ := C;
```

ST でのファンクションブロック呼び出し

ファンクションブロック (略称 FB) は、構造化テキストで次の構文に従って呼び出されます。

<FB インスタンス名 >(FB 入力変数 :=< 値またはアドレス >|, < さらに FB 入力変数 :=< 値またはアドレス >|... さらに FB 入力変数);

例

次の例では、タイマーファンクションブロック (TON) がパラメーター IN および PT の代入と共に呼び出されます。次に、結果変数 Q が変数 A に代入されます。タイマーファンクションブロックは、TMR:TON; でインスタンス化されます。IL のように、結果変数は構文 <FB インスタンス名 >.<FB 変数 > に従って指定されます。

```
TMR(IN := %IX5, PT := 300);
A:=TMR.Q;
```

出力には別の構文もあります。

```
fb(in1:=myvar, out1=>myvar2);
```

RETURN 使い方

POU から抜けるには、RETURN 命令を使用できます。

構文

RETURN;

例

```
IF b=TRUE THEN
RETURN;
END_IF;
a:=a+1;
```

b が TRUE の場合、命令 a:=a+1; は実行されません。その結果、POU からすぐに抜けます。

IF 使い方

IF 命令で条件をテストし、この条件に応じて命令を実行します。

構文

IF < ブール型 _ 式 1> THEN

<IF_ 命令 >

{ELSIF < ブール型 _ 式 2> THEN

<ELSIF_ 命令 1>

..

..

ELSIF < ブール型 _ 式 n> THEN

<ELSIF_ 命令 -1>

ELSE

<ELSE_ 命令 >}

END_IF;

括弧 {} の部分はオプションです。

< ブール型 _ 式 1> が TRUE を返した場合 <IF_ 式 > のみが実行され、その結果、他の命令は実行されません。そうでない場合、< ブール型 _ 式 2> からブール式の 1 つが TRUE を返すまで順番に評価されます。次に、このブール式の後、および次の ELSE または ELSIF の前の命令だけが評価されます。ブール式のどれも TRUE でない場合は、<ELSE_ 命令 > のみが評価されます。

例

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

この例では、温度が 17 度未満になると加熱がオンになります。それ以外の場合は、オフのままです。

CASE 使い方

CASE 命令を使用して、1 つの構成の中で複数の条件付命令と同じ条件変数を組み合わせできます。

構文

CASE <Var1> OF

< 値 1>: < 命令 1>

< 値 2>: < 命令 2>

< 値 3、値 4、値 5>: < 命令 3>

< 値 6、値 10>< 命令 4>

..

..

< 値 n>: < 命令 n>

ELSE <ELSE 命令 >

END_CASE;

CASE 命令は、以下のモデルに従って処理されます。

- 〈Var1〉の変数の値が < 値 1> の場合、その命令 < 命令 1> が実行されます。
- 〈Var 1〉の値が示されている値のどれでもない場合、<ELSE 命令 > が実行されます。
- 変数の複数の値に対して同じ命令を実行する場合、値をコンマで区切って順番に記述することで、共通実行を条件付けできます。
- 変数の値の範囲に対して同じ命令を実行する場合は、初期値と終了値を 2 つのドットで区切って記述します。その結果、共通な条件を条件付けできます。

例

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
    BOOL3 := FALSE;
2: BOOL2 := FALSE;
    BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
    BOOL3:= TRUE;
ELSE
   BOOL1 := NOT BOOL1;
   BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR ループ

FOR ループを使用して、繰り返す処理をプログラムできます。

構文

```
INT_Var:INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY < ステップサイズ >} DO

< 命令 >

END_FOR;
```

括弧 {} の部分はオプションです。

カウンター〈INT_Var〉が <END_VALUE> 以下である限り < 命令 > が実行されます。これは、< 命令 > を実行する前にチェックされるため、<INIT_VALUE> が <END_VALUE> より大きい場合 < 命令 > は実行されません。

< 命令 > が実行されると、〈INT_Var〉が < ステップサイズ > 分増えます。ステップサイズは任意の整数値にできます。値がない場合は、1 に設定されます。〈INT_Var〉が <END_VALUE> より大きいときにループが終了します。

例

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Var1 のデフォルトの設定が 1 とします。その後、FOR ループの後に値が 32 になります。

注記：<END_VALUE> が <INT_Var> (上記の例ではカウンター) のデータ型の制限値に等しい場合、無限ループが生成されます。カウンターが SINT タイプで、例えば、<END_VALUE> が 127 (SINT タイプ変数の正の最大値) である場合、この最大値に 1 を加えると変数が負になり、FOR 命令による制限を超えることがないためループが終了できません。

⚠ 警告

無限ループによる装置の意図しない動作

FOR 命令で使用する変数型に <END_VALUE> + 1 となる十分な容量 (十分な上限) があることを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

FOR ループ内で CONTINUE 命令を使用できます。これは IEC 61131-3 規格への拡張です。

WHILE ループ

FOR ループの代替は、ブール条件が TRUE である場合、また TRUE である限りループを実行する WHILE ループです。最初に条件が TRUE でない場合、ループは実行されません。最初に TRUE であった条件が FALSE になると、ループは終了します。

構文

WHILE < ブール式 > DO

< 命令 >

END_WHILE;

明らかに、初期および進行中のブール式は、ループ命令内のどこかで FALSE の値になることを前提としてください。それ以外の場合、ループは終了せず、無限ループ状態を引き起こします。

⚠ 警告

無限ループによる装置の意図しない動作

ブール式の FALSE 条件を作成し、ループの命令内で WHILE ループが終了することを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

次の例は、ループ内でループを終了させる命令の例です

```
WHILE Counter>0 DO
 Var1 := Var1*2;
 Counter := Counter-1;
END_WHILE
```

REPEAT 命令をまだ紹介していないので、下の項 (変更あり) に移動します。

WHILE ループ内で CONTINUE 命令を使用できます。

REPEAT ループ

REPEAT ループは、WHILE ループの場合と同様に、FOR ループのもう 1 つの代替です。REPEAT ループは、少なくとも 1 回はループが実行され、そのループの終了時に終了条件が評価されるという点で WHILE ループと異なります。

構文

REPEAT

< 命令 >

UNTIL < ブール式 >

END_REPEAT;

< 命令 > は、< ブール式 > が TRUE を返す限り繰り返し実行されます。< ブール式 > が最初の UNTIL の評価時にすでに生成されている場合、< 命令 > は一度のみ実行されます。< ブール式 > は、ループ命令内のどこかで TRUE の値になることを前提としてください。それ以外の場合、ループは終了せず、無限ループ状態を引き起こします。

⚠ 警告

無限ループによる装置の意図しない動作

ブール式の TRUE 条件を作成し、ループの命令内で REPEAT ループが終了することを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

次の例は、ループ内でループを終了させる命令の例です

```
REPEAT
 Var1 := Var1*2;
 Counter := Counter-1;
UNTIL
 Counter=0
END_REPEAT;
```

REPEAT ループ内で CONTINUE 命令を使用できます。これは IEC 61131-3 規格への拡張です。

WHILE および REPEAT ループはループを実行する前にサイクル数を知る必要がないため、ある意味 FOR ループより強力です。そのため場合によっては、これら 2 つのループタイプのみで作業できます。ただし、ループのサイクル数が明確な場合は、殆どの場合で本質的に無限ループを排除する (FOR ループ項 (*334* ページ) の警告メッセージを参照) ため FOR ループが推奨されます。

CONTINUE 使い方

IEC 61131-3 規格への拡張として、FOR、WHILE、および REPEAT ループで CONTINUE 命令が対応されています。CONTINUE により次のループサイクルでの実行が処理されます。

例

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
IF INT1=0 THEN
CONTINUE; (* to avoid division by zero *)
END_IF
Var1:=Var1/INT1; (* only executed, if INT1 is not "0" *)
END_FOR;
Erg:=Var1;
```

EXIT 使い方

EXIT 命令は、条件に関係なく命令の入っている FOR、WHILE、または REPEAT ループを終了させます。

JMP 使い方

JMP 命令を使用して、ジャンプラベルで印付けされたコード行に無条件でジャンプできます。

構文

JMP < ラベル >;

< ラベル > はプログラム行の先頭に置かれる任意であるが固有の識別子です。JMP 命令の後に、あらかじめ定義されたラベルと等しいジャンプ先の指示が続きます。

⚠ 警告

無限ループによる装置の意図しない動作

JMP 命令の使用は、無限ループを引き起こさなという条件付きであることを確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

以下は、ジャンプとそのジャンプ先の間の無限ループを回避する論理条件を作成する命令の例です。

```
aaa:=0;
_label1: aaa:=aaa+1;
(*instructions*)
IF (aaa < 10) THEN
JMP _label1;
END_IF;
```

0 で初期化された変数 i が 10 より小さい値である限り、上記の例のジャンプ命令は、ラベル _label1 で定義されたプログラム行への繰り返しフライバックに影響します。従って、ラベルと JMP 命令間に構成された命令の繰り返し処理に影響します。これらの命令には、変数 i のインクリメントも含まれているため、ジャンプ条件が (9 番目のチェックで) 侵害され、プログラムフローが進行します。

例に WHILE または REPEAT ループを使用することでもこの機能を実現できます。一般的に、ジャンプ命令を使用すると、コードの読みやすさが低下します。

ST のコメント

構造化テキストオブジェクトにコメントを書く方法は 2 つあります。

- (* でコメントを開始し、*) でコメントを閉じます。これにより、複数の行に渡るコメントを挿入できます。例 : (* これはコメントです。*)
- IEC 61131-3 規格への拡張としての単一行コメント : // は行の終わりで終わるコメントの始まりを示します。例 : // これはコメントです。

コメントは、ST エディターの宣言部または実装部のどこにでも配置できます。

コメントのネスト : コメントを他のコメント内に配置できます。

例

```
(*
a:=inst.out; (* to be checked *)
b:=b+1;
*)
```

この例では、最初の括弧で始まるコメントは checked に続く括弧ではなく、最後の括弧で閉じられます。

第 V 部
オブジェクトエディター

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
15	宣言エディター	341
16	デバイスタイプマネージャー (DTM) エディター	347
17	データユニットタイプ (DUT) エディター	349
18	グローバル変数リスト (GVL) エディター	351
19	ネットワーク変数リスト (NVL) エディター	353
20	タスクエディター	373
21	ウォッチリストエディター	385
22	ロジックエディターのツール	389

第 15 章
宣言エディター

概要

テキスト宣言エディターで POU オブジェクトの宣言部を作成します。表形式ビューで補足することもできます。1 つのビューで行われた変更は、直ぐにもう 1 つのビューにも適用されます。

宣言エディターオプションの現在の設定に応じて、テキストビューまたは表形式ビューのどちらか一方のみが使用できます。エディターウィンドウの右端にあるボタン (**テキスト** / **表**) で切り替えができます。

宣言エディターは、プログラミング言語エディターと組み合わせて使用します。つまり、オフラインまたはオンラインモードでオブジェクトを編集または表示 (監視) するときに開くウィンドウの上部に配置されます。宣言ヘッダーには POU タイプが記述されます (例 : PROGRAM, FUNCTION_BLOCK, FUNCTION)。グローバルな POU pragma 属性で拡張できます。

宣言エディターのオンラインモード (*346* ページ) は、**ウォッチ**ビューのように構成されています。

変数宣言は、別のエディターで作成される**グローバル変数リスト**および**データユニットタイプ**でも実行されます。

*変数宣言*の章 (*481* ページ) も参照してください。

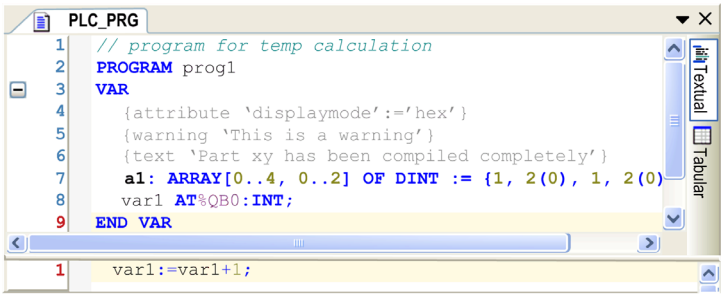
この章について

この章には次の項目が含まれています。

テキスト宣言エディター

概要

テキストエディタービュー

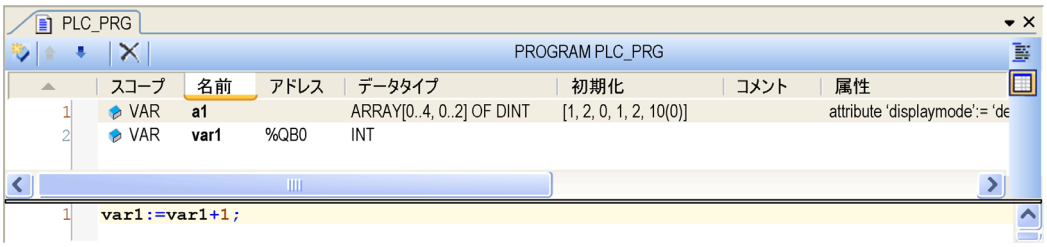


動作と外観は、**オプション**ダイアログボックスおよび**カスタマイズ**ダイアログボックスの現在のテキストエディターの設定によって決まります。そこで、強調表示の色、行番号、タブ、インデント、その他多数のオプションの初期設定を定義できます。コピーや貼り付けなどの通常の編集機能が使用できます。マウスでテキスト領域を選択しながら ALT キーを押すことによって、ブロック選択できます。

表形式宣言エディター

概要

表形式エディタービュー



エディターの表形式ビューには、変数宣言 (*481* ページ) の通常の定義の列 : **スコープ**、**名前**、**アドレス**、**データタイプ**、**初期化**、**コメント**、および (pragma) **属性**があります。特定の宣言は、番号付きの行として挿入されます。

既存の行の上に新しい行の宣言を追加するには、初めに行を選択し、ツールバーまたはコンテキストメニューから **挿入**コマンドを実行します。

表の最後に新しい宣言を追加するには、既存の宣言の最後の行の次をクリックし、**挿入**コマンドを使用します。

新しく挿入された宣言は、デフォルトとしてスコープ **VAR** と最後に入力したデータ型が使用されます。必須入力項目の変数**名**が自動的に開きます。有効な識別子を入力して **Enter** キーを押すか、ビューの別の場所をクリックしてフィールドを閉じます。

表のセルをダブルクリックすると、値が入力できます。

スコープをダブルクリックすると、スコープおよびスコープ属性のキーワード (フラグ) を選択できるリストが開きます。

データ型に直接入力するか、**>** ボタンをクリックして**入力アシスタント**または**配列ウィザード**を使用します。

初期値を直接入力するか、**...** ボタンをクリックして**初期値** ダイアログボックスを開きます (*344* ページ)。これは特に構造化変数の場合に便利です。

各変数は、行番号が付けられた個別の行で宣言します。

行の順序 (行番号) は、行を選択してツールバーまたはコンテキストメニューの **上に移動**または **下に移動**コマンドで 1 行上または下に移動させることで変更できます。

それぞれの列のヘッダーをクリックすることで、各列に従って宣言リストを並べ替えできます。

1 つまたは複数の宣言を削除するには、それぞれの行を選択して **Del** キーを押すか、コンテキストメニューから**削除**コマンドを実行するか、またはツールバーの ボタンをクリックします。

配列の宣言

配列変数を宣言するには、**タイプ**フィールドの右側にある矢印ボタン **>** を使用して、**Array Wizard** を選択します。**配列**ダイアログボックスが開きます。

赤い感嘆符アイコンでマークされたフィールドは必ず記入してください。下限と上限、および変数の**ベースタイプ**を入力して、**Dimension** を定義します。矢印ボタンをクリックして、**入力アシスタント** ダイアログボックスまたは基本タイプを宣言するための別の **Array Wizard** を開くことができます。

[*, *, *] を使用して配列変数の長さを定義できます。可変長の配列は、ファンクションブロック、メソッド、およびファンクションの VAR_IN_OUT 宣言でのみ使用できます。可変長の配列を宣言するには、各次元にアスタリスク * を入力します。これは ARRAY [*..*] OF INT になります。**OK** で確定したら、次元の文字列を [*] (1 つのアスタリスクのみ) に合わせます。

可変長の 2 次元配列の例

ARRAY [*,*]

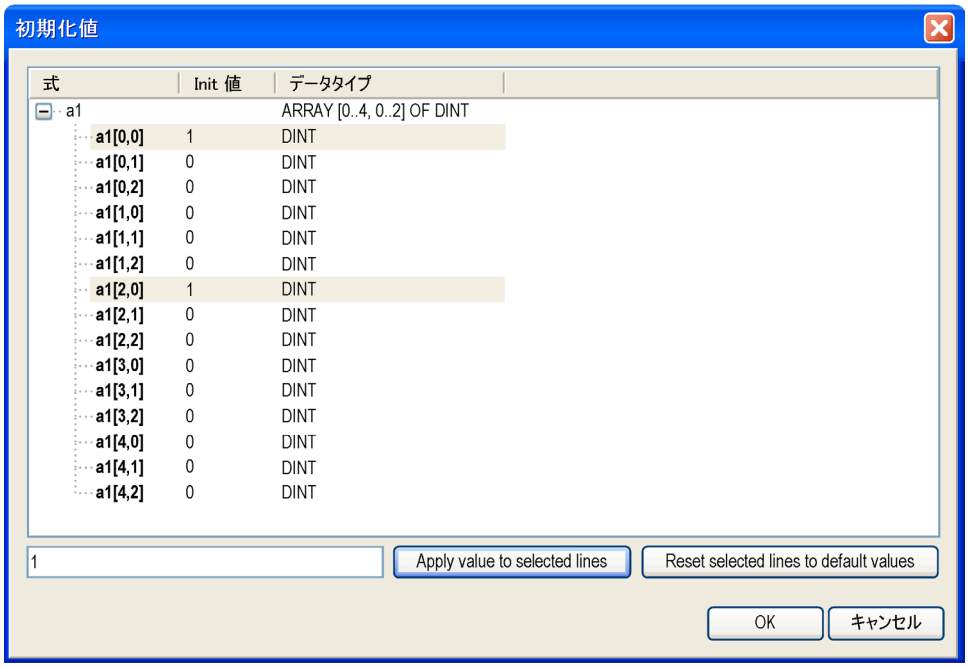
ダイアログボックスの**結果**領域に、構成された配列宣言のプレビューが表示されます。

詳細については、*配列*の詳細 (*574* ページ) を参照してください。

OK をクリックして宣言ダイアログボックスを閉じます。変数宣言は、IEC 構文に従って宣言エディターに表示されます。

初期化値

初期値ダイアログボックス



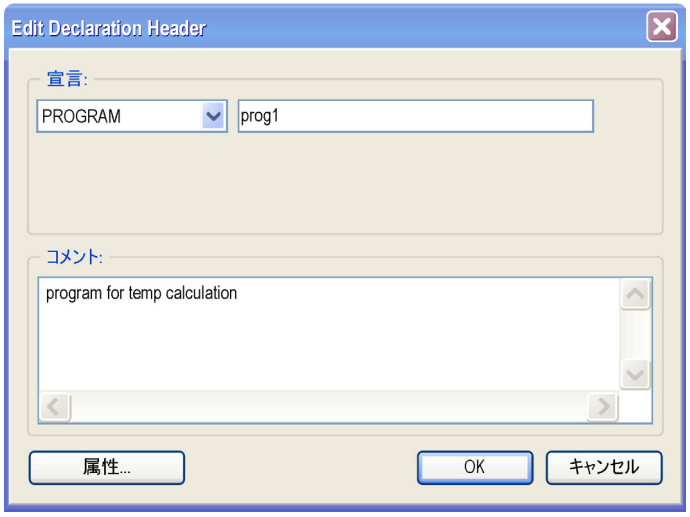
変数の**式**と現在の初期値が表示されます。変数を選択し、リストの下のフィールドで初期値を編集します。次に、**Apply value to selected lines** ボタンをクリックします。デフォルトの初期値を復元するには、**Reset selected lines to default values** ボタンをクリックします。

Ctrl + **Enter** を押すと、**コメント**入力に改行が挿入されます。

Edit Declaration Header

Edit Declaration Header ダイアログボックスで宣言ヘッダーを編集できます。エディターのヘッダーバー (前のページの図の PROGRAM PLC_PRG) をクリックするか、**Edit Declaration Header** コマンドで開きます。

Edit Declaration Header ダイアログボックス



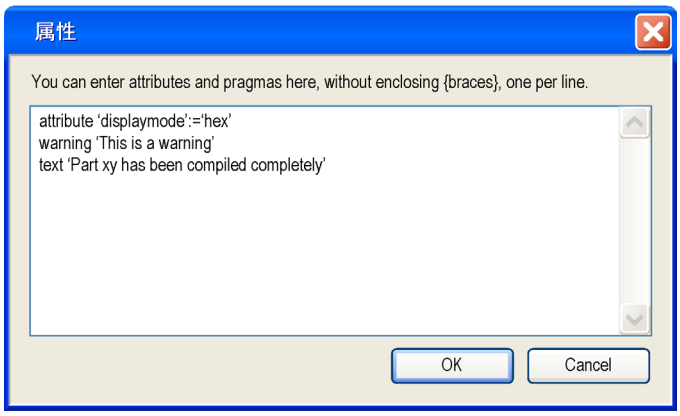
Edit Declaration Header ダイアログボックスには次の要素が表示されます。

要素	詳細
宣言	POU オブジェクトのタイプ (リストから選択) および名前を挿入します。
コメント	コメントを挿入します。**Ctrl** + **Enter** を押すと、改行が挿入されます。
属性	Pragma および属性を挿入するための**属性**ダイアログボックス (この章の下を参照) が開きます。

属性

Edit Declaration Header ダイアログボックスの **属性 ...** ボタンをクリックして、**属性** ダイアログボックスを開きます。テキスト形式で複数の属性と pragma を入力できます。括弧 {} で囲まずに挿入し、それぞれ別の行を使用します。次の図の例については、上の対応するテキストエディタービュー (*342* ページ) の図を参照してください。

属性 ダイアログボックス



オンラインモードの宣言エディター

概要

宣言エディターのオンラインビューには、ウォッチビュー (*388* ページ) で使用される表と同じものが表示されます。ヘッダー行には実際のオブジェクトパス < デバイス名 >.< アプリケーション名 >.< オブジェクト名 > が表示されます。各ウォッチ式の表には、タイプと現在値が表示されます。強制または書き込み用の値が設定されている場合は、設定済みの値も表示されます。直接割り当てられた IEC **アドレス**および**コメント**が設定されている場合、追加された列に表示されます。

変数の設定済みの値を設定するには、**値の準備**ダイアログボックスを使用するか、**設定済みの値**のフィールドをクリックして値を直接入力します。列挙型の場合は、表示される列挙値のリストから値を選択します。ブール変数の場合はさらに簡単です。

ブール型の設定済みの値は、**Return** キーまたは **Space** キーを使用して以下の順序に従って切り替えできます。

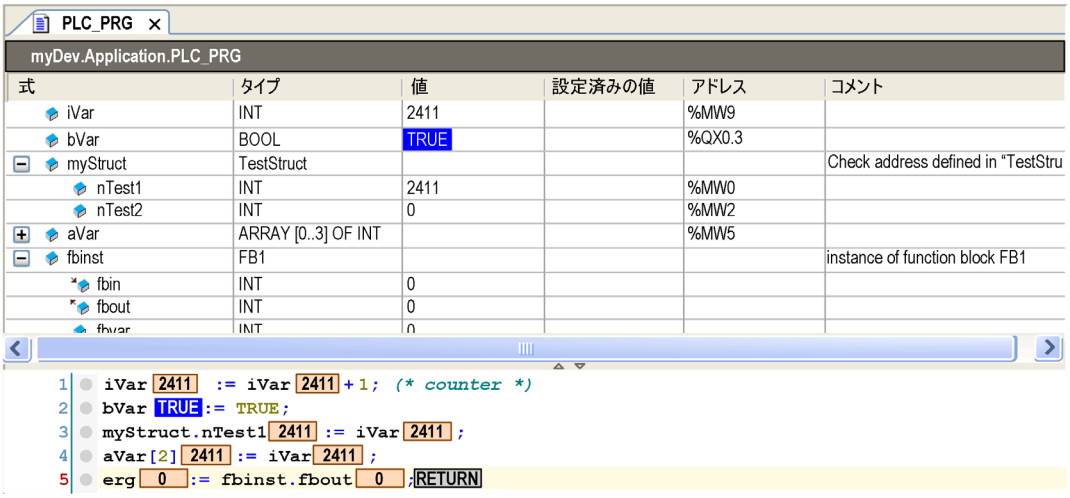
- 値が TRUE の場合、準備ステップは FALSE -> TRUE -> なし です。
- 値が FALSE の場合、準備ステップは TRUE -> FALSE -> なし です。

ウォッチ式 (変数) が構造化型、例えばファンクションブロックのインスタンスや配列変数の場合、式の前にプラスまたはマイナスの符号が付きます。この記号をマウスでクリックすると、インスタンス化されたオブジェクトの特定の要素を追加表示 (次の図の fbinst を参照) または非表示 (aVar を参照) にすることができます。アイコンは、変数が入力  、出力  、または通常の変数  のどれであるかを示します。

式が配列の場合は、監視する配列インデックスの範囲を定義できます。これを行うには、**タイプ**列をダブルクリックして**監視範囲**ダイアログボックスを開きます。範囲を定義するために、**Start Index**、および **End Index** を入力します。

実装部分の変数にカーソルを合わせると、ツールチップに変数の宣言とコメントが表示されます。次の図に示すオンラインビューでのプログラミングオブジェクト PLC_PRG の上部にある宣言エディターを参照してください。

宣言エディターのオンラインビュー



第 16 章
デバイスタイプマネージャー (DTM) エディター

DTM エディター

概要

DTM エディタービューはデバイスタイプマネージャーによって異なります。

詳細は、*デバイスタイプマネージャー (DTM) ユーザーガイド*を参照してください *(EcoStruxure Machine Expert, Device Type Manager (DTM), User Guide 参照)*。

現在 EcoStruxure Machine Expert で対応している DTM のバージョンのリストについては、ご使用の EcoStruxure Machine Expert のリリースノートを参照してください。

第 17 章
データユニットタイプ (DUT) エディター

データユニットタイプエディター

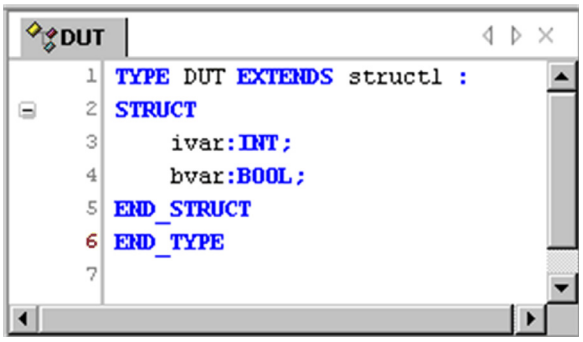
概要

データユニットタイプエディター (DUT エディター) でユーザー定義データ型 (*560* ページ) を作成できます。このエディターは現在設定されているテキストエディターのオプションに準じて動作するテキストエディターです。

オブジェクトの追加ダイアログボックスで DUT オブジェクトを追加すると、ウィンドウに DUT エディターが自動的に開きます。この場合、初期設定で拡張構造宣言の構文が表示されます。必要に応じて、簡単な構造宣言の入力または列挙型などの別のデータタイプユニットの宣言の入力に使用できます。

POU ビューで現在選択している既存の DUT オブジェクトを開いても、エディターが開きます。

DUT エディターウィンドウ



第 18 章
グローバル変数リスト (GVL) エディター

GVL エディター

概要

GVL エディターは、**グローバル変数リスト**を編集するための**宣言エディター**です。GVL エディターは宣言エディターと同じように動作します。オフラインおよびオンラインの両方のオプションに対応し、テキストエディターに設定されています。宣言は VAR_GLOBAL から始まり、END_VAR で終わります。これらのキーワードは自動的に付けられます。その間にグローバル変数の有効な宣言を入力します。

GVL エディター

保持変数エディターとして使用される GVL エディター

GVL エディターを使用して**保持変数**オブジェクトを管理することもできます。この場合、エディターには VAR_GLOBAL PERSISTENT 変数の宣言が含まれます。プロパティの詳細については、*並列分岐* (*179* ページ) の章を参照してください。

第 19 章
ネットワーク変数リスト (NVL) エディター

この章について

この章には次のセクションが含まれています。

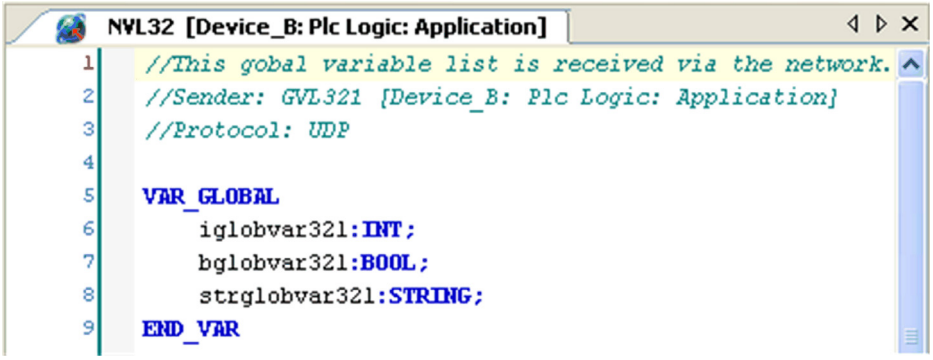
19.1
NVL エディターの情報

ネットワーク変数リストエディター

概要

NVL エディターは、**ネットワーク変数リスト**を編集するための**宣言エディター**です。NVL エディターは宣言エディターと同じように動作します。オフラインおよびオンラインの両方のオプションに対応し、テキストエディターに設定されています。宣言は VAR_GLOBAL で始まり、END_VAR で終わります。これらのキーワードは自動的に付けられます。その間にグローバル変数の有効な変数宣言 (*479* ページ) を入力します。

NVL エディター



19.2
ネットワーク変数の一般情報

このセクションについて

このセクションには次の項目が含まれています。

ネットワーク変数リスト (NVL)

概要

ネットワーク変数リスト (NVL) 機能は、通信ネットワークを介して送受信できる変数の固定リストで構成されています。コントローラー (対象システム) で対応されている場合、ネットワーク変数を介してネットワーク内のデータ交換ができます。

送信側コントローラーおよび受信側コントローラーでリストを定義してください (単一または複数のプロジェクトで処理できます)。変数の値はユーザーデータグラムプロトコル (UDP) データグラムを使って、ブロードキャストで送信されます。UDP は、IETF RFC 768 で定義されているコネクションレスインターネット通信プロトコルです。このプロトコルにより、インターネットプロトコル (IP) ネットワーク上でのデータグラムの直接送信が容易になります。UDP/IP メッセージは応答を待つことがないため、損失したパケットの再送信の必要がないアプリケーション (リアルタイムのパフォーマンスが必要なストリーミングビデオやネットワークなど) に最適です。

NVL 機能は、EcoStruxure Machine Expert の強力な機能です。コントローラーおよびアプリケーションでデータの共有や監視ができます。コントローラーのデータ交換に関して制限はありません。そのため、機械の試運転、インターロック処理、およびコントローラー状態の変更の目的にも使用できます。

注記：ネットワーク変数のタイプは異なるコントローラー間では共有されません。使用するタイプがすべてのデバイスで同じ定義を持つようにする必要があります。それ以外の場合、NVL 通信はできません。例えば、SEC.ETH_R_STRUCT や SEC.PLC_R_STRUCT などのタイプが適用されます。それらはデフォルトで、サイズやフィールドが異なるさまざまなコントローラーで利用できます。

アプリケーションの設計者またはプログラマーのみが、機械の動作中または処理中のすべての状態および要因を認識することができ、この機能を使用したコントローラー間のデータ交換の目的に必要な通信方式、インターロックおよび関連する安全性を判断できます。このタイプの通信機能を監視し、機械および処理の設計によって人員または機械に安全上のリスクを与えないように厳重に注意してください。

⚠ 警告

制御不能

- 制御手法の設計者は制御パスの障害モードが発生するおそれを考慮する必要があり、特定の重要制御機能については、パス障害の最中および終了後に安全な状態を実現するための方策を準備しておく必要があります。重要制御機能の例としては、緊急停止、オーバートラベル停止、停電、および再起動があります。
- 重要な制御機能に対しては、別のまたは冗長性のある制御パスを用意してください。
- システム制御パスには、データ通信が含まれることがあります。予期しないデータの転送遅れや障害について考慮する必要があります。
- あらゆる事故防止規制および地域の安全性ガイドライン [1] を遵守してください。
- 運用を開始する前に、各実装について、正しく動作するかどうかを個別に十分にテストする必要があります。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

[1] 詳細は、NEMA ICS 1.1 (最新版)、"Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control"、および NEMA ICS 7.1 (最新版)、"Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems"、または該当地域での同等のガイドラインを参照してください。

この機能と共に、診断 *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)*、エラー管理 *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)*、およびネットワークプロパティパラメーターを使用して、通信のヘルス、状態、および完全性を監視できます。この機能は、データの共有と監視のために設計されており、重要な制御機能には使用できません。

ネットワーク変数リスト (NVL)

交換されるネットワーク変数は、次の 2 種類のリストで定義されます。

- 送信コントローラーのネットワーク変数リスト (NVL 送信機)
- 受信コントローラーのネットワーク変数リスト (NVL 受信機)

通信している NVL (送信機) と NVL (受信機) には同じ変数宣言が含まれています。その内容は、**デバイスツリー**の (NVL 送信機) または (NVL 受信機) ノードをダブルクリックすると、それぞれのエディターに表示されます。

NVL (送信機) には、送信機のネットワーク変数が含まれています。送信機の**ネットワークプロパティ**には、プロトコルおよび送信パラメーターが定義されています。これらの設定に従って、変数値がネットワーク内で送信されます。通信している NVL (受信機) をもつコントローラーで受信できます。

注記：ネットワーク変数を交換するには、それぞれのネットワークライブラリーをインストールしてください。これは、NVL (送信機) のネットワークプロパティが設定されるとすぐにネットワークタイプの UDP に対して自動的に行われます。

ネットワーク変数は、NVL (送信機) から 1 台または複数の NVL (受信機) に送信されます。コントローラーごとに NVL (送信機) および NVL (受信機) を定義できます。そのため、各コントローラーは、受信機としてだけでなく送信機としても機能できます。

同じプロジェクトまたは別のプロジェクトから送信機 NVL (送信機) を提供できます。NVL (受信機) を作成するときに、NVL (送信機) をネットワーク内で使用可能なすべての NVL (送信機) の選択リストから選択するか、以前に NVL (送信機) から生成したエクスポートファイル (例えば、**ファイルへのリンク**ダイアログボックスを使用して生成) から読み込むことができます。

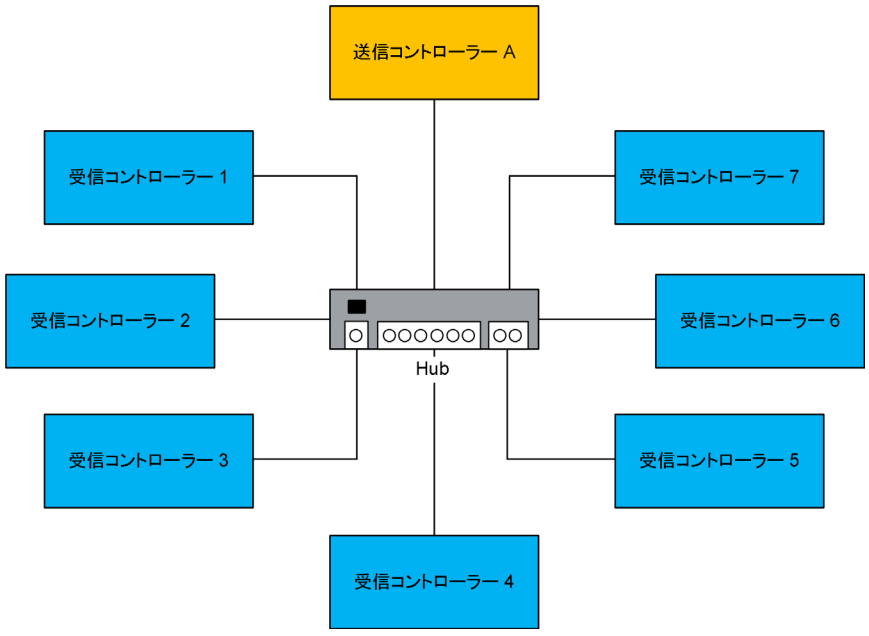
注記：使用する NVL (送信機) が別のプロジェクトで定義されている場合は、エクスポートファイルが必要です。

NVL に関する注意事項

次の表は、ネットワーク変数リスト (NVL) 機能に対応しているコントローラーのリストです。

ファンクション名	M241	M251	M258 LMC058	LMC Eco LMC Pro LMC Pro2
ネットワーク変数リスト	あり	あり	あり	あり

図は、送信機 1 台と最大 7 台の受信機で構成されるネットワークを示しています。



コントローラー送信 A: NVL (送信機) を持つ送信側とネットワーク変数リスト (NVL (受信機)) を持つ受信側コントローラー

コントローラー受信 1~7: A からの NVL (受信機) の受信および A 専用の送信コントローラー (NVL (送信機))

ネットワーク変数交換の設定

概要

送信機と受信機の間でネットワーク変数を交換するには、EcoStruxure Machine Expert **デバイスツリー**で送信コントローラー 1 台、受信コントローラー 1 台を使用可能にしてください。これらのコントローラーには、以下で説明するネットワークプロパティが割り当てられています。

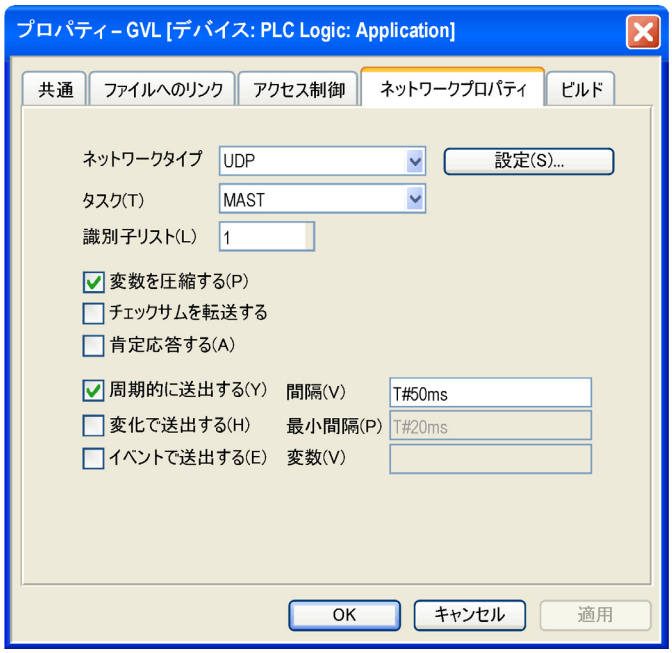
ネットワーク変数リストを設定するには、以下の手順に沿って進めます。

手順	手順内容
1	**デバイスツリー**に送信機および受信機のコントローラーを作成します。
2	送信側および受信側のコントローラーのプログラム (POU) を作成します。
3	送信側および受信側のコントローラーのタスクを追加します。 **注記**：性能における透過性を維持するために、専用 NVL タスクのタスク優先度を 25 より高く設定し、通信を規制してネットワークが不必要に飽和することを避けてください。
4	送信機の NVL （送信機）を定義します。
5	受信機の NVL （受信機）を定義します。

詳細情報を含む例は、付録 *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* に記載されています。

グローバル変数リスト

NVL (送信機) を作成するには、**GVL → プロパティ → ネットワークプロパティ**のダイアログボックスで、次のネットワークプロパティを定義します。



パラメーターの説明

パラメーター	初期値	詳細
ネットワークタイプ	**UDP**	ネットワークタイプ **UDP** のみ使用可能。 **Broadcast Address** および **Port** を変更するには、**設定 ...** ボタンをクリックします。
タスク	**MAST**	NVL コードを実行するために、**タスク設定**で設定したタスクを選択します。 性能の透過性を維持するために、このタスクのサイクルタイムの**間隔**を 50ms 以上に設定することを推奨します。 **注記**：性能における透過性を維持するために、専用 NVL タスクのタスク優先度を 25 より高く設定し、通信を規制してネットワークが不必要に飽和することを避けてください。

パラメーター	初期値	詳細
識別子リスト	**1**	ネットワークの各 NVL (送信機) に固有な番号を入力します。これは、受信機で変数リスト *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* を識別するために使用します。
変数を圧縮する	有効	このオプションを有効にすると、変数は送信用にパケット (データグラム) にまとめられます。 このオプションを無効にすると、変数ごとに 1 つのパケットが送信されます。
チェックサムを転送する	無効	このオプションを有効にすると、送信中に変数の各パケットにチェックサムが追加されます。 受信機は、受信した各パケットのチェックサムを確認し、チェックサムが一致しないものは拒否します。通知は、NetVarError_CHECKSUM パラメーター *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* で発行されます。
肯定応答する	無効	このオプションを有効にすると、受信したデータパケットごとに受信機から確認メッセージを送信するように指示します。 送信機が次のデータパケットを送信する前に受信機から確認メッセージを受信しなかった場合は、NetVarError_ACKNOWLEDGE パラメーター *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* で通知が発行されます。
周期的に送出する ● **間隔**	有効	定義された**間隔**で周期的にデータを送信する場合は、このオプションを選択します。 この**間隔**は、ネットワーク変数の送信時間を正確にするために NVL コードの実行用タスクで定義したサイクルタイムの倍数にしてください。
変化で送出する ● **最小間隔**	無効 ● **T#20ms**	このオプションを有効にすると、変数の値が変更されたときに変数を送信します。 **注記 :** 最初のダウンロード後、またはオンラインモードで**リセットコールド**または**リセットウォーム**コマンドを使用した後は、受信側コントローラーは更新されず最後の値が保持されます。ただし、送信側コントローラーの値は 0 (ゼロ) になります。 **最小ギャップ**パラメーターは、データ送信に最小限必要な間隔時間を定義します。
イベントで送出する ● **変数**	無効 ● –	このオプションを有効にすると、指定した**変数**が TRUE に等しい場合に変数を送信します。変数は、NVL コードを実行するタスクの各サイクルで確認されます。

設定 ... ボタンの説明

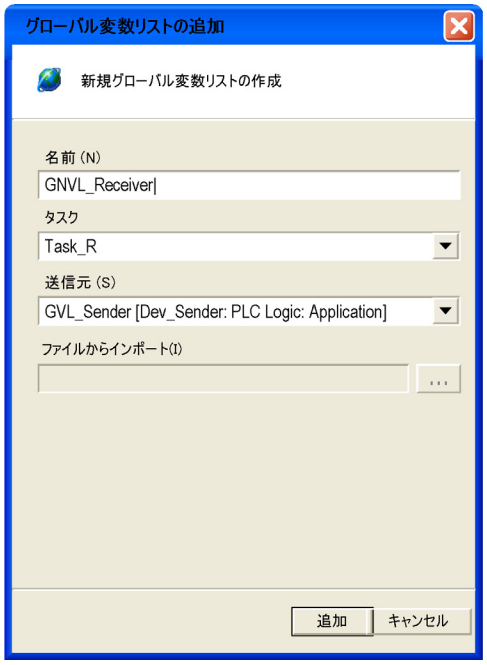
パラメーター	初期値	詳細
ポート	1202	各 NVL (送信機) に固有のポート番号 (≥ 1202) を入力します。
ブロードキャストアドレス	255.255.255.255	アプリケーションの特定のブロードキャスト IP アドレスを入力します。

ネットワーク変数リスト (受信機)

グローバルネットワーク変数リストは、**デバイスツリー**にのみ追加できます。これにより、ネットワーク内の別のコントローラーでネットワーク変数として指定される変数を定義します。

従って、ネットワークプロパティ (ネットワーク変数リスト) をもつ NVL (送信機) が、既に他のネットワークのコントローラーで作成されている場合は、NVL (受信機) オブジェクトはアプリケーションにのみ追加できます。これらのコントローラーは、同じプロジェクト内の場合もあり、異なるプロジェクト内の場合もあります。

NVL (受信機) を作成するには、**オブジェクトの追加 → グローバルネットワーク変数リスト**ダイアログボックスで次のパラメーターを定義します。



パラメーターの説明

パラメーター	初期値	詳細
名前	**NVL**	NVL (受信機) の名前を入力します。
タスク	このアプリケーションの**タスク設定**ノードで定義されているタスク	受信側コントローラーの**タスク設定**ノードで使用可能な送信機からのフレームを受信するタスクをタスクリストから選択します。
送信機	プロジェクトで現在使用できる NVL (送信機) のうちの 1 つ	プロジェクトで使用可能なネットワークプロパティのある NVL (送信機) のリストから NVL (送信機) を選択します。 他のプロジェクトの NVL (送信機) を使用するには、リストの**ファイルからインポート**を選択します。これにより、以下の**ファイルからインポート :** パラメーターが有効になります。
ファイルからインポート :	–	このパラメーターは、**送信機**パラメーターの**ファイルからインポート**オプションを選択した後にのみ使用できます。 **...** ボタンから、別のプロジェクトの NVL (送信機) から作成したエクスポートファイル **.gvl* を参照できる Windows Explorer ウィンドウが開きます。 詳細については、下記の*異なるプロジェクトから NVL (受信機) を追加する方法* の項を参照してください。

同じプロジェクトで NVL (受信機) を追加する方法

オブジェクトの追加ダイアログボックスで NVL (受信機) を追加すると、ネットワークの現在のプロジェクト内にある適切な NVL (送信機) が、**Sender** のリストボックスに表示されます。詳細については、下記の*異なるプロジェクトから NVL (受信機) を追加する方法* の項を参照してください。

この選択により、現在のコントローラー (送信機) の各 NVL (受信機) が別のコントローラー (受信機) の 1 つの NVL (送信機) にリンクされます。

さらに NVL 受信機) を追加するときは、名前およびネットワーク変数の処理をするタスクを定義してください。

異なるプロジェクトで NVL (受信機) を追加する方法

別のコントローラーから NVL (送信機) を直接選択する代わりに、**ファイルへのリンク**プロパティを使用して、以前に NVL (送信機) から生成した NVL (送信機) エクスポートファイルを指定することもできます。これにより、別のプロジェクトで定義された NVL (送信機) を使用できます。

これを実行するには、**Sender:** パラメーターの **Import from file** オプションを選択し、**Import from file:** パラメーターでパスを指定します。

設定は後から、**プロパティ - GVL** ダイアログボックスで変更できます。

NVL (受信機) のプロパティ

デバイスツリーの NVL (受信機) 項目をダブルクリックすると、エディターの右側に内容が表示されます。ただし、通信する NVL (送信機) の内容への参照のみであるため、NVL (受信機) の内容は編集できません。通信する NVL (送信機) を含む送信機の正確な名前とパスは、使用するネットワークプロトコルのタイプと共にエディターペインの上部に表示されます。通信する NVL (送信機) が変更されると、それに応じて NVL (受信機) の内容も更新されます。

ネットワーク変数リスト (NVL) の規則

データ量に関する規則

性能上の制限があるため、次の規則に従ってください。

番号	規則
1	1 つの NVL (送信機) から 1 台の NVL (受信機) へのデータ送信は 200 バイト以内にしてください。
2	1 つのコントローラーの複数の NVL (送信機) とそれに関連する NVL (受信機) のデータ交換は変数の合計サイズが 1000 バイト以内にしてください。

データグラムの数に関する規則

NVL タスクの最大サイクル数を制限するために、次の規則に従ってください。

番号	規則	詳細
1	1 サイクルあたりの受信データグラムの数は 20 に制限してください。	制限を超えた場合、残りのデータグラムは次のサイクルで処理されます。制限に達したときに、診断データ *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* に**受信オーバーフロー**の通知が生成されます。 1 つのデータグラムは最大 256 バイトです。つまり、1 台の受信機に送信するデータは 5120 バイトの制限を超えないでください。
2	1 サイクルあたりの送信データグラムの数は 20 に制限してください。	制限を超えた場合、残りのデータグラムは次のサイクルで処理されます。制限に達したときに、診断データ *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* に**送信オーバーフロー**の通知が生成されます。 1 つのデータグラムは最大 256 バイトです。つまり、1 台の送信側コントローラーで送信するデータは 5120 バイトの制限を超えないでください。

1 サイクルあたりの受信 / 送信データグラムの数の制限を複数回超えると、以下が発生する場合があります。

- UDP (ユーザーデータグラムプロトコル) データグラムの損失
- 非干渉または一貫性のない変数の交換

必要に応じて次のパラメーターを調整します。

- 送信側コントローラーのサイクルタイム
- 受信側コントローラーのサイクルタイム
- ネットワークの送信機の数

注記

データの損失

システムを稼働させる前に、UDP データグラムが適切に送受信されるかアプリケーションを十分にテストしてください。

上記の指示に従わないと、物的損害を負う可能性があります。

NVL (送信機) の最大数

性能の透過性を維持するために、コントローラー (送信機) あたりの NVL (送信機) の最大数は 7 に定義してください。

NVL (送信機) と NVL (受信機) のタスクサイクルタイムに関する規則

受信オーバーフローを回避するには、NVL (送信機) 送信を管理するタスクのサイクルタイムを NVL (受信機) 受信を管理するタスクのサイクルタイムの少なくとも 2 倍以上に定義してください。

識別子リスト保護に関する規則

NVL 機能には、識別子リストのチェックが含まれています。

識別子リストは、同じ識別子リスト (ダイアログボックス **GVL → プロパティ → 識別子リスト**：ダイアログボックス参照) をもつ 2 台の別のコントローラーからの NVL (送信機) が、同じ NVL (受信機) の任意のコントローラーにデータグラムを送信することを防ぎます。**識別子リスト**が固有でない場合、変数の交換が中断される場合があります。

注記
通信の損失 ネットワークの識別子リストが 1 つの IP アドレスでのみ使用されていることを確認してください。 **上記の指示に従わないと、物的損害を負う可能性があります。**

識別子リストのチェック機能は、受信側コントローラーに実装されます。

NVL (受信機) で、2 つの異なる IP アドレスが同じ識別子リストを使用していることが検出されると、受信機はすぐにデータグラムの受信を停止します。

さらに、NETVARGETDIAGINFO ファンクションブロックで通知が発行されます。2 つの送信機の IP アドレスは、このファンクションブロックの *(EcoStruxure Machine Expert, Network Variable Configuration, SE_NetVarUdp Library Guide 参照)* 出力パラメーター dwDuplicateListIdIp1 および dwDuplicateListIdIp2 で提供されます。

ファンクションブロック NETVARRESETERROR を使用して、検出された NVL エラーをリセットし、通信を再開します。

ネットワーク変数のタイプの一貫性

注記：ネットワーク変数のタイプは異なるコントローラー間では共有されません。使用するタイプがすべてのデバイスで同じ定義を持つようにする必要があります。それ以外の場合、NVL 通信はできません。

例えば、SEC.ETH_R_STRUCT や SEC.PLC_R_STRUCT などのタイプ適用されます。それらはデフォルトで、サイズやフィールドが異なるさまざまなコントローラーで利用できます。

送信機および受信機の動作状態

動作状態

動作状態		ネットワーク変数動作
送信機	受信機	
RUN	RUN	ネットワーク変数が送信機と受信機の間で交換されます。
STOP	RUN	送信機は変数を受信機に送信しません。ネットワーク変数は送信機と受信機の間で交換されません。
RUN	STOP	受信機は送信機からのネットワーク変数を処理していません。 受信機が RUN 状態に戻ると、受信機はネットワーク変数の処理を再開します。
STOP	STOP	変数は交換されません。

注記：送信機の動作状態を STOP から RUN に変更すると、複数の通信初期化エラー (NetVarError_INITCOMM) が検出されます。

NVL を管理するタスクのイベント

NVL を管理するタスクで次のイベントが発生した場合、NVL の動作はコントローラーが上記の配列の STOP 状態にある場合と同じです。

- タスクを中断させるアプリケーションで例外が発生
- ブレークポイントにヒットする、またはタスクで 1 つのサイクルが処理される

例

概要

次の例では、簡単なネットワーク変数交換が確立されています。送信側コントローラーでは、NVL (送信機) が作成されます。受信側コントローラーでは、それに対応する NVL (受信機) が作成されます。

デバイスツリーに送信側コントローラー **Dev_Sender** および受信側コントローラー **Dev_Receiver** がある標準プロジェクトで、以下の準備を行います。

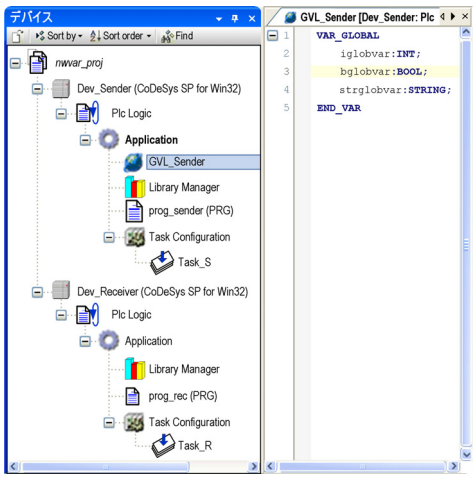
- **Dev_Sender** の **アプリケーション**ノードの下に POU (プログラム) **prog_sender** を作成します。
- このアプリケーションの**タスク設定**ノードの下に、**prog_sender** を呼び出すタスク **Task_S** を追加します。
- **Dev_Receiver** の**アプリケーション**ノードの下に POU (プログラム) **prog_rec** を作成します。
- このアプリケーションの**タスク設定**ノードの下に、**prog_rec** を呼び出すタスク **Task_R** 追加します。

注記：2 つのコントローラーは、Ethernet ネットワークの同じサブネットで設定してください。

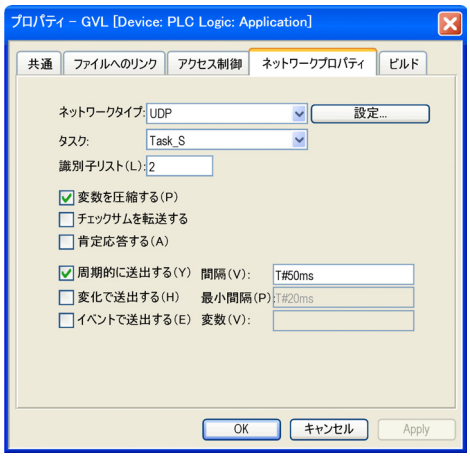
NVL (送信機) の定義

手順 1: 送信側コントローラーでグローバル変数リストを定義します。

手順	手順内容	コメント
1	**デバイスツリー**で、コントローラー **Dev_Sender** の**アプリケーション**ノードを右クリックし、**オブジェクトの追加 → グローバル変数リスト ...** コマンドを実行します。	**グローバル変数リストの追加**ダイアログボックスが表示されます。
2	**名前** NVL_Sender を入力し、**開く**をクリックして新しいグローバル変数リストを作成します。	**デバイスツリー**の**アプリケーション**ノードの下に **NVL_Sender** ノードが表示され、右側にエディターが開きます。

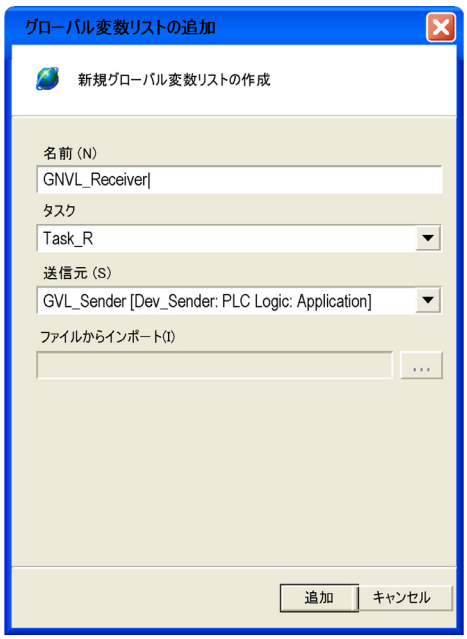
3	右側のエディターに以下の変数定義を入力します。
	`VAR_GLOBAL`
	`iglobvar:INT;`
	`bglobvar:BOOL;`
	`strglobvar:STRING;`
	`END_VAR`
	 
		–

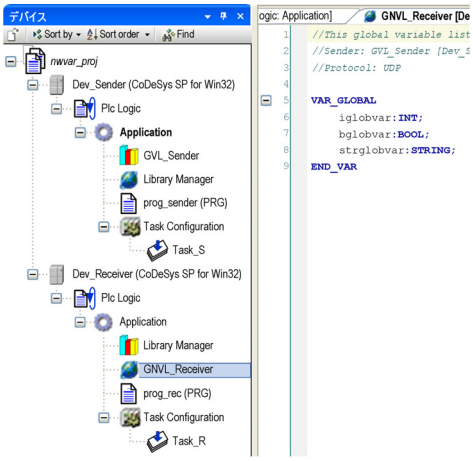
手順 2: NVL (送信機) のネットワークプロパティを定義します。

手順	手順内容	コメント
1	**デバイスツリー**で **NVL_Sender** ノードを右クリックして**プロパティ ...** コマンドを実行します。	**プロパティ - NVL_Sender** ダイアログボックスが表示されます。
2	**ネットワークプロパティ**タブを開き、図のようにパラメーターを設定します。 	–
3	**OK** をクリックします。	ダイアログボックスが閉じ、NVL (送信機) のネットワークプロパティが設定されます。

NVL (受信機) の定義

手順 1: 送信側コントローラーでグローバルネットワーク変数リストを定義します。

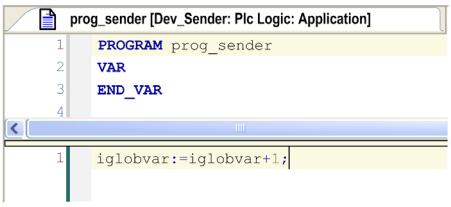
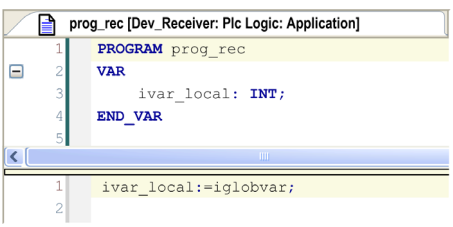
手順	手順内容	コメント
1	**デバイスツリー**で、コントローラー **Dev_Receiver** の**アプリケーション**ノードを右クリックし、**オブジェクトの追加 → グローバルネットワーク変数リスト ...** コマンドを実行します。	**グローバルネットワーク変数リストの追加**ダイアログボックスが表示されます。
2	図のようにパラメーターを設定します。 	このグローバルネットワーク変数リストは、送信側コントローラー用に定義された NVL (送信機) と同じです。

手順	手順内容	コメント
3	**開く**をクリックします。	ダイアログボックスが閉じ、**Dev_Receiver** コントローラーの**アプリケーション**ノードの下に **GNVL_Receiver** が表示さます。  この NVL (受信機) には、自動的に **GVL_Sender** と同じ変数宣言が含まれます。

手順 2: NVL (受信機) のネットワーク設定を表示、または変更します。

手順	手順内容	コメント
1	**デバイスツリー**で **GNVL_Receiver** ノードを右クリックして**プロパティ ...** コマンドを実行します。	**プロパティ - GNVL_Receiver** ダイアログボックスが表示されます。
2	**ネットワーク設定**タブが開きます。	–

手順 3: オンラインモードでネットワーク変数交換のテストをします。

手順	手順内容	コメント
1	コントローラー **Dev_Sender** の**アプリケーション**ノードの下にある POU **prog_sender** をダブルクリックします。	右側に **prog_sender** 用のエディターが開きます。
2	次のコードを変数 iglobvar に入力します。 	–
3	コントローラー **Dev_Receiver** の**アプリケーション**ノードの下にある POU **prog_rec** をダブルクリックします。	右側に **prog_rec** 用のエディターが開きます。
4	次のコードを変数 ivar_local に入力します。 	–

手順	手順内容	コメント
5	同じネットワーク内の送信側アプリケーションおよび受信側アプリケーションにログオンし、アプリケーションを起動します。	受信側の変数 ivar_local は、送信側に現在表示されている iglobvar の値を取得します。

互換性

概要

バージョンが異なるプログラミングシステム (例えば、V2.3 および V3.x) のアプリケーションでコントローラーが動作していても、ネットワーク変数による通信は可能です。

ただし、バージョン間でファイル形式が異なるエクスポートファイル (*.exp と *.gvl) は、プロジェクト間で簡単にインポートおよびエクスポートはできません。

最新バージョン (例えば、V3.x) で NVL (受信機) が設定されている場合、必要なネットワークパラメーターの設定を最新バージョン (例えば、V3.x) の送信機から受け取ってください。以前のバージョン (例えば、V2.3) の送信機で作成されたエクスポートファイル *.exp には、この情報が含まれていません。

バージョンが異なるプログラミングシステムのアプリケーション間でのネットワーク変数の交換に対する解決策を次の項で説明します。

グローバルネットワーク変数リストの更新

バージョンが異なるプログラミングシステム (例えば、V2.3 および V3.x) のアプリケーション間でネットワーク変数を交換するには、次の手順を実行してグローバルネットワーク変数リストを更新します。

手順	手順内容	コメント
1	すでに以前のバージョン (V2.3) にあるネットワーク変数リスト (NVL) を最新バージョン (V3.x) で再作成します。	これを実行するには、以前のバージョン (V2.3) の NVL と同じ変数宣言を含むネットワークプロパティをもつ NVL (送信機) を追加します。
2	**ファイルへのリンク**タブを使用して、新しい NVL (送信機) を *.exp ファイルにエクスポートします。	**注記：** プリコンパイル時に不明瞭な名前に変更されることなくプロジェクトの NVL (送信機) を維持するために、**ビルド**タブの**ビルドから除く**オプションを有効にして、プロジェクト内の NVL (送信機) がプリコンパイルされたり不明瞭な名前に変更されるのを防ぎます。NVL (送信機) に変更が必要な場合に備えて、*.exp ファイルを再作成するオプションを無効にします。
3	リストを再度インポートします。	これを実行するには、適切に設定された受信リストを作成するために、以前に生成された *.exp ファイルを使用して新しい NVL (受信機) を作成します。

この手順を次の例で示します。

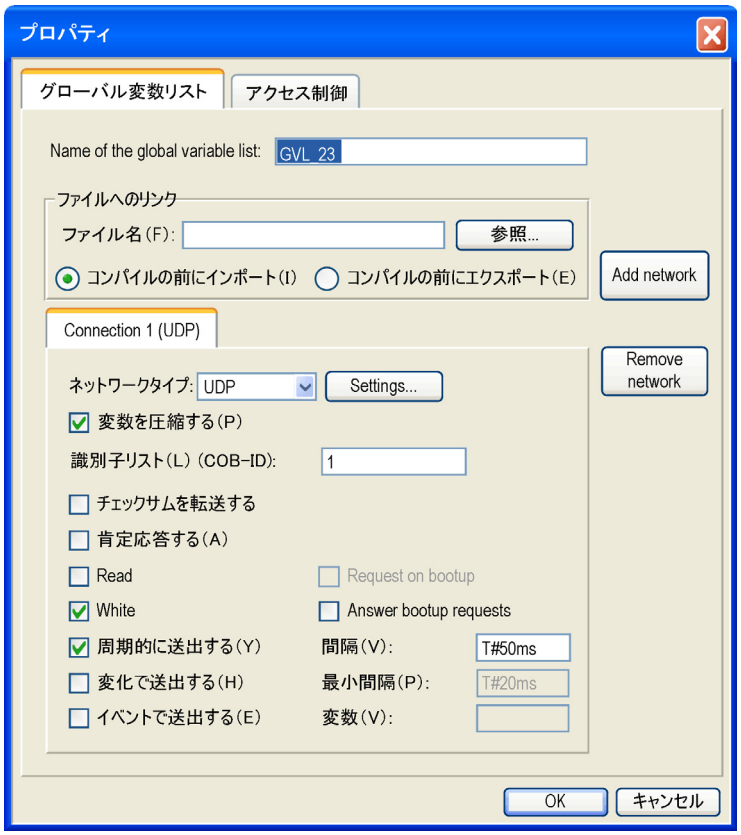
例

この例では、V2.3 アプリケーションで定義された変数 trans23 がそれ以降のバージョン (V3.x) で使用可能になります。

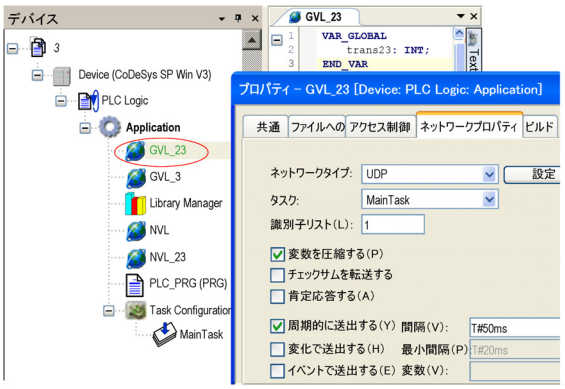
次の条件が定義されています。

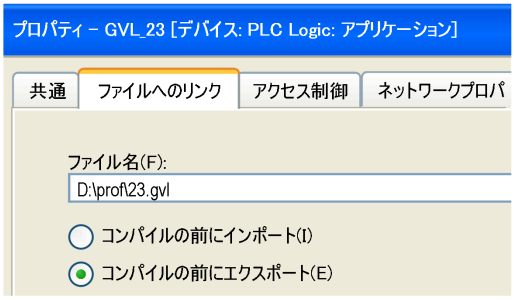
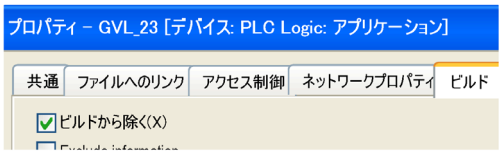
条件	詳細
1	以前のバージョンのプログラミングシステム (V2.3) では、プロジェクト *23.pro* に次の宣言と共にグローバル変数リスト **GVL_23** が含まれています。 VAR_GLOBAL trans23:INT; END_VAR

条件	詳細

2	**GVL_23** のネットワークプロパティは、次のように設定されています。
	 
	
	注記：この **GVL_23** をエクスポートすると、次の変数宣言のみを含む **.exp* ファイルが作成されます。
	`VAR_GLOBAL`
	`trans23:INT;`
	`END_VAR`
	**.exp* ファイルに構成の設定は含まれていません。

次の表に、最新バージョン (V3.x) で **GVL_23** を再作成するために実行する次の手順を示します。

手順	手順内容	コメント
1	**GVL_23** という名前の NVL (送信機) オブジェクトをアプリケーションに追加します。	–
2	ネットワークプロパティを *23.pro* プロジェクト内で定義されているように設定します。	

手順	手順内容	コメント
3	**ファイルへのリンク**タブで対象のエクスポートファイル *23.gvl* を設定します。	
4	**ビルド**タブで、**ビルドから除く**オプションを設定します。	この設定により、ファイルを後で変更できるようにディスクに保存できます。 
5	プロジェクトをコンパイルします。	*23.gvl* ファイルが生成され、変数と構成の設定が含まれます。 
6	*23.gvl* エクスポートファイルから**ファイルからインポート**コマンドを使用して V3.x プロジェクトに NVL (受信機) オブジェクトを追加します。	これにより、以前のプログラミングシステム (V.2.3) のコントローラーから変数 trans23 が読み込まれます。 ネットワークで以前のバージョン (V2.3) のプロジェクトと最新バージョン (V3.x) のアプリケーションの両方が実行されている場合は、最新バージョン (V3.x) のアプリケーションでプロジェクト *23.pro* から変数 trans23 を読み取ることができます。

第 20 章
タスクエディター

この章について

この章には次の項目が含まれています。

タスク設定についての情報

概要

タスク設定で、アプリケーションプログラムの処理を制御するためのタスクを定義します。そのため、タスク設定はアプリケーションに必須のオブジェクトであり、**アプリケーションツリー**で使用できる必要があります。

タスク設定ツリーの説明

タスク設定ツリーの一番上に**タスク設定**  があります。その下には、定義されたタスクのタスク名が表示されます。特定のタスクの POU 呼び出しはタスク設定ツリーに表示されます。

アプリケーションツリーで使用できるコマンドを使用して、タスクツリーを編集 (タスクの追加、コピー、貼り付け、または削除) できます。例えば、新しいタスクを追加するには、**タスク設定**ノードを選択して緑色のプラスボタンをクリックし、**タスク ...** コマンドを実行します。または、**タスク設定**ノードを右クリックし、**オブジェクトの追加 → タスク ...** コマンドを実行します。

さらにオンラインモードでの監視ビューがあるタスクエディター (*380* ページ) で特定のタスクを設定します。タスク設定に使用できるオプションは、コントローラーのプラットフォームによって異なります。

アプリケーションツリーのタスク設定

タスク

タスク (*380* ページ) は、IEC プログラムの処理を制御するために使用します。これは、名前、優先度、およびタスク開始のトリガー条件を決定するタイプで定義します。時間 (サイクリック、フリーホイール) またはタスクをトリガーする内部または外部イベント (例えば、グローバルプロジェクト変数の立上がりまたはコントローラーの割り込みイベント) でこの条件を定義できます。

各タスクごとに、タスクによって開始される一連のプログラム POU を指定できます。現在のサイクルでタスクが実行される場合、これらのプログラムが 1 サイクル間に処理されます。

優先度と条件の組み合わせによって、実行されるタスクの時系列 (*383* ページ) が決定します。

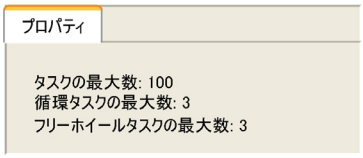
各タスクに時間制御 (ウォッチドッグ) を設定できます。特定のコントローラープラットフォームによって可能な設定は異なります。

プロパティタブ

概要

タスク設定 (*374* ページ) ノードを選択すると、**プロパティ**タブが**タスク設定**エディターで開きます。

タスク設定、**プロパティ**タブの例



コントローラーの現在のタスク設定の情報、例えば、タスクタイプごとのタスクの最大許容数などが表示されます。

システムイベントタブ

概要

タスクではなくシステムイベントでプロジェクトのファンクションを呼び出すには、**タスク設定**エディターの**システムイベント**タブを使用します。イベントごとに呼び出されるファンクションを定義でき、各システムイベントを個別に有効または無効にすることができます。

この機能はすべてのコントローラーで対応している訳ではありません。なお、各コントローラーについては、*プログラミングガイド*を参照してください。

ボタンの詳細

ボタン	詳細
Add Event Handler...	イベントとイベントに呼び出されるファンクションの割り当てを定義できる **Add Event Handler** ダイアログボックスが開きます。
Remove Event Handler	選択した割り当てを削除します。
Event Info	対応するイベントライブラリーの情報を表示します。
Open Event Function	選択した割り当ての新規ファンクション用のエディターが開きます。 新規ファンクションの記述言語は、**Add Event Handler** ダイアログボックスで選択します。

Add Event Handler ダイアログボックス

要素	詳細
イベント	システムイベントの一覧を表示します。 使用できるイベントは対象コントローラーによって異なります。使用できないイベントは、赤いシンボルでマークされます。
Function to call	ファンクション名を入力します。 **Add Event Handler** ダイアログボックスで **OK** をクリックすると、新しい **POU**、**ファンクション**タイプがこの名前で**アプリケーションツリー**に追加されます。
スコープ	• **アプリケーション**オプションを選択してアプリケーションで使用できる新しいファンクションを作成します。 • **POU** オプションを選択してプロジェクト全体で使用できる新しいファンクションを作成します。
記述言語	新しいファンクションのプログラミング言語を選択します。
詳細	選択したイベントに関する簡単な説明が表示されます。

システムイベント一覧

Add Event Handler ダイアログボックスで定義したシステムイベントは、**システムイベント**タブに表示されます。

列	詳細
名前	**Add Event Handler** ダイアログボックスで定義した**イベント**の名前が表示されます。
詳細	選択したイベントに関する簡単な説明が表示されます。
Function to call	**Add Event Handler** ダイアログボックスの **Function to call** パラメーターで定義したファンクション名が表示されます。
有効	このオプションはデフォルトで選択され、ファンクションは有効です。 無効にするには、**有効**オプションの選択を解除します。

オンラインモードでは、**システムイベント**タブに追加情報が表示されます。対応しているシステムイベントに関するコントローラー固有の情報については、各コントローラーについては、*プログラミングガイド*を参照してください。

欄	詳細
イベントステータス	**0** の場合、エラーが検出されなかったことを示します。 値が 0 ではない場合、エラーが検出されたことを示します。
呼び出し回数	イベントの発生とそれに伴うファンクションの呼び出し回数が表示されます。

ボタン	詳細
オンラインリセット	**オンラインリセット**ボタンをクリックすると、イベントリストが再初期化され、イベントおよびファンクションの呼び出しカウンターがリセットされます。正常に初期化されなかったイベントは、赤い**イベントステータス**のセルで表示されます。

システムイベント

表示可能なシステムイベントを次の表に示します。

イベント	詳細	タスク	デバッグ
AfterWritingOutputs	出力を書き込んだ後の呼び出し。	IEC タスク	あり
BeforeReadingInputs	入力を読み込む前の呼び出し。	IEC タスク	あり
CodeInitDone	オンライン変更中に新しいアプリケーションコードが初期化された後にイベントが送信されます。	通信タスク	なし
例外	アプリケーションのコンテキストで例外が検出された場合は、イベントが送信されます。	例外 IEC の間、システム例外タスク、またはその例外が発生したタスク自身からイベントが呼び出されます。	タスクにより異なります。
ログイン	このアプリケーションでのクライアントのログイン	通信タスク	なし
ログアウト	このアプリケーションからのクライアントのログアウト	通信タスク	なし
OnlineChangeDone	アプリケーションのオンライン変更後の呼び出し。	通信タスク	なし
PrepareOnlineChange	アプリケーションのオンライン変更前の呼び出し。	通信タスク	なし
sercos_BeforeChangeToPhase0	フェーズ 0 への変更前の呼び出し。	TskSercos3App	なし
sercos_BeforeConnectSlaves	**デバイスツリー**で実際のスレーブが理論スレーブにマップされる前の呼び出し。	TskSercos3App	なし
sercos_TasksPhase0	フェーズ 0 への変更中の呼び出し。	TskSercos3App	なし
sercos_TasksPhase1	フェーズ 1 への変更中の呼び出し。	TskSercos3App	なし
sercos_TasksPhase2PostFW	フェーズ 2 への変更後の呼び出し。	TskSercos3App	なし
sercos_TasksPhase2PreFW	フェーズ 2 への変更前の呼び出し。	TskSercos3App	なし
sercos_TasksPhase3PostFW	フェーズ 3 への変更後の呼び出し。	TskSercos3App	なし
sercos_TasksPhase3PreFW	フェーズ 3 への変更前の呼び出し。	TskSercos3App	なし
sercos_TasksPhase4	フェーズ 4 への変更中の呼び出し。	TskSercos3App	なし

監視タブ

概要

対象のシステムが監視に対応している場合は、監視機能を使用できます。これは、タスクによって制御される実行時間、および呼び出し回数を動的に分析します。オンラインモードでは、タスクの処理を監視できます。

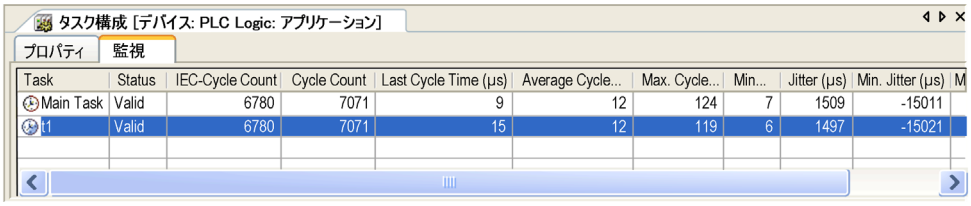
タスク設定エディターのオンラインビュー

タスク設定ツリーの上位ノードを選択すると、**プロパティ** タブ (*375* ページ) の他に**監視**タブが表示されます。オンラインモードでは、サイクルとサイクルタイムの状態および現在の統計が表形式で表示されます。値の更新間隔は、コントローラーの値の監視に使用されている間隔と同じです。

要素の説明

タスク構成ツリーの上位ノードを選択すると、**プロパティ** ダイアログ (*375* ページ) に加えて追加タブに**監視**ダイアログが表示されます。オンラインモードでは、サイクルとサイクルタイムの状態および現在の統計が表示されます。値の更新間隔は、コントローラーの値の監視に使用されている間隔と同じです。

タスク構成、監視



各タスクの行に以下の情報が表示されます。

タスク	**タスク構成**で設定したタスク名。
Status	状態表示 : ● **Not created**: 前回の更新以降に開始されていません (特にイベントタスクに使用されます)。 ● **作成済み**: タスクはランタイムシステムで認識されていますが、まだ動作するための設定がされていません。 ● **有効** : タスクは正常に動作しています。 ● **例外** : タスクに例外が発生しました。
IEC-Cycle Count	アプリケーションが起動してからの実行サイクルの数です。対象のシステムがこの機能に対応していない場合は 0 になります。
Cycle Count	すでに実行されたサイクルの数 (対象のシステムによって異なります。これは IEC サイクルカウントと同じか、またはアプリケーションが実行していないときにもサイクル数をカウントしている場合はそれ以上です。)
Last Cycle Time (µs)	最後に測定された実行時間 (µs)
Average Cycle Time (µs)	すべてのサイクルの平均実行時間 (µs)
Max. Cycle Time (µs)	すべてのサイクルの測定された最長実行時間 (µs)
Min. Cycle Time (µs)	すべてのサイクルの測定された最短実行時間 (µs)
Jitter (µs)	最後に測定されたジッター (µs)
Min. Jitter (µs)	測定されたジッターの最小 (µs)
Max. Jitter (µs)	測定されたジッターの最大 (µs)
* ジッター : タスクが開始されてからオペレーティングシステムが実行中を示すまでの経過時間。	

タスクの値を 0 にリセットするには、タスク名のフィールドにカーソルを置き、コンテキストメニューにある**リセット**コマンドを実行します。

Variable Usage タブ

概要

タスク設定の **Variable Usage** タブには変数の概要とタスクでの使用方法が表示されます。

変数の使用一覧

Variable Usage タブの一覧には、次の情報が表示されます。

列	詳細
Variables	変数名が表示されます。
Type	変数のデータ型が表示されます。
Count	その変数にアクセスするタスクの数が表示されます。
< タスク名 >	変数のアクセスが表示されます。 ● **r** = read (読み込み) ● **w** = write (書き込み) ● **rw** = read/write (読み込み / 書き込み)

コンテキストメニューのコマンド

一覧の項目を右クリックすると、以下を実行するためのコンテキストメニューが表示されます。

- タスクの選択を解除して、リストからタスクを非表示にします。
- **Browse Cross References** コマンドを実行して変数のクロスリファレンス一覧を表示します。

特定のタスクの設定

概要

アプリケーションツリーの**タスク設定**ノードにタスクを挿入すると、**設定**タブにタスクを設定するためのタスクエディタービューが開きます。

また、タスクの設定を変更するために使用可能なタスク (例えば、**MAST**) をダブルクリックしても開きます。

注記：アプリケーションツリーで該当する項目を編集することによってタスク名を変更できます。

属性を挿入します。

優先度	
優先度 (0...31)	0 から 31 までの数字。0 が最も優先度が高く、31 が最も優先度が低いです。新しいタスクのデフォルト値はコントローラーによって定義されています。 **注記：**アプリケーションタスクを適した設定にするために、コントローラー固有のタスク設定を考慮してください。通信専用のタスクに優先順位を割り当てるときや、サイバーセキュリティーなどの関連において、重要になることがあります。アプリケーションタスクを通信タスクよりも高い優先順位に設定することで、システムの頑健性を高めることができます。

タイプ	
対象デバイスによって、対応するタスクタイプが定義されます。すべてのデバイスで、すべてのタイプが使用できる訳ではありません。なお、各コントローラーについては、*プログラミングガイド*を参照してください。	
サイクリック	タスクは**間隔**フィールド (以下を参照) に指定された時間定義 (タスクサイクルタイム) で周期的に処理されます。
イベント	**イベント**フィールドで定義された変数の立上がりですぐにタスクが開始します。
フリーホイール	タスクはプログラムが開始するとすぐに処理され、1 回の実行の終了時に自動的に連続ループで再開します。サイクルタイムは定義されていません。
外部	**外部イベント**フィールドで定義されたシステムイベントが発生するとすぐにタスクが開始されます。これは、対象システムで対応しているイベントおよび選択リストにあるイベントによって異なります。(システムイベントと混同しないでください。)
ステータス	**イベント**フィールドで定義された変数が TRUE の場合、タスクが開始されます。

タスクタイプに応じた必須項目

項目	詳細
間隔 (例えば、**t#200ms**)	**サイクリック**タスクタイプで必須。 タスクを再開するまでの時間 (ミリ秒 [ms])。 タスク周期時間を設定する場合、アプリケーションで使用されているバスシステムを考慮してください。例えば、CAN バスシステムでは **CANopen I/O マッピング** タブで**バスサイクルタスク**を設定できます。タスク周期時間は、転送速度とバス上で使用するフレーム数に対応させてください。ハートビート、ノード保護、および同期の設定時間は常にタスク周期時間の倍数にしてください。そうでない場合、CAN フレームが認識されない可能性があります。詳細については、EcoStruxure Machine Expert オンラインヘルプの *デバイスエディター*を参照してください。
イベント	**イベント**タイプで必須、または**外部イベント**でトリガーされます。 立上がりが検出されるとすぐにタスクの開始をトリガーするグローバルブール型変数です。**...** ボタンまたは**入力アシスタント**を使って、使用できるすべてのグローバルイベント変数のリストを表示できます。 **注記：**タスクを実行するイベントが項目に起因する場合、少なくとも 1 つはイベントによって実行されないタスクが必要です。そうでない場合、I/O が更新されず、タスクが開始されません。 **注記：**内部 IEC 変数のみが使用できます。イベントタスクでプロパティ (システムパラメーターを含む) を参照すると、ダウンロード中にウォッチドッグ例外エラー (*381* ページ) が検出されます。

ステータスとイベントの違い

指定したイベントが TRUE であることはステータス実行型タスクの開始条件を満たしますが、イベント実行型タスクではイベントが FALSE から TRUE へ変わる必要があります。イベントが TRUE から FALSE に変わり、また TRUE に戻るのが早過ぎる場合、このイベントが検出されずに**イベント**タスクが開始しない場合があります。

次の例は、イベント (緑色の線) の反応に対するタスクの動作を示しています。

サンプルポイント 1 ~ 4 における異なるタイプのタスクの異なる反応

ポイントでの動作	**1**	**2**	**3**	**4**
ステータス	開始しない	開始	開始	開始
イベント	開始しない	開始	開始しない。イベントが TRUE から FALSE に変わり、また TRUE 戻るのが速過ぎたため。	開始しない

ウォッチドッグ設定

各タスクに時間制御 (ウォッチドッグ) を設定できます。

ウォッチドッグのデフォルト設定はコントローラーによって異なります。

有効オプションが有効になっている (チェックマークが入っている) と、ウォッチドッグが有効になります。ウォッチドッグが有効な場合、タスクの実行時間が定義されたタスク時間制限 (**時間**) を超えると、**例外**エラーが検出されます。エラーが発生したタスクとそれに対応する子アプリケーションを含むアプリケーションが停止します。これによってアプリケーションのタスクも停止するという影響があります。指定した **感度**が考慮されます。コントローラー設定ダイアログボックスの**停止中に IO を更新**オプションが有効な場合、出力は特定のコントローラープラットフォームに応じてあらかじめ定義されたデフォルト値に設定されます。この機能はすべてのコントローラーで対応している訳ではありません。各コントローラーについては、*プログラミングガイド*を参照してください。

以下のような場合が起こる可能性があります。

- 複数の連続したタイムアウト
  - **感度** : 0、1 - サイクル 1 での例外
  - **感度** : 2 - サイクル 2 での例外
  - **感度** : n - サイクル n での例外
    注記：感度 (n) の上限はコントローラーによって異なります。感度、タスク時間、その他の関連パラメーターに関する詳細は、ご使用のコントローラーのプログラミングガイドの *システムとタスクウォッチドッグの章を参照してください*。
- シングルタイムアウト : 現在のサイクルタイムが (タスク時間制限 x 感度) より長い場合は例外です。
  例 :
  時間 =t#10 ms、**感度** =5 (タスクが 50 ミリ秒を超えて実行されるとすぐに例外が示されます。)

時間 (例えば、t#200ms)	タスクの許容最大実行時間を定義します。タスクがこの時間より長く掛かると、コントローラーはタスクウォッチドッグの例外を報告します。
感度単位	コントローラーがアプリケーションエラーを検出する前に発生する必要があるタスクウォッチドッグの例外の数を定義します。

注記：ウォッチドッグ機能は、シミュレーションモードでは使用できません。

各コントローラーについては、*プログラミングガイド*を参照してください。タスク時間、感度、およびその他の使用可能なウォッチドッグパラメーターについては、*システムとタスクウォッチドッグ*の章を参照してください。

POU

タスクによって制御される POU は、POU 名がオプションの**コメント**と共にリスト表示されます。表の上には編集用のコマンドがあります。

- 新しい POU を定義するには、**Add Call** コマンドを使用して**入力アシスタント**ダイアログボックスを開きます。プロジェクトで使用可能なプログラムを 1 つ選択します。また、**アプリケーションツリー**からプログラムタイプの POU をドラッグ & ドロップでリストに追加することもできます。
- プログラム呼び出しを別のプログラムと置き換えるには、表の項目を選択し、**Change Call ..** コマンドで**入力アシスタント**を開いて別のプログラムを選択します。
- 呼び出しを削除するには、表から選択して **Remove Call** コマンドを使用します。
- **Open POU** コマンドは、選択しているプログラムに対応するエディターが開きます。

上から下にリスト表示された POU 呼び出しのシーケンスにより、オンラインモードでの実行順序が決定します。**上に移動**および**下に移動**コマンドを使って、選択した項目をリスト内で移動できます。

オンラインモードでのタスク処理

どのタスクが処理されていますか？

タスク設定で定義されたタスクの実行には、次の規則が適用されます。

- 条件が満たされたタスクが実行されます。これは、指定された時間が経過した場合または条件 (イベント) 変数が立上がりを示した後であることを意味します。
- 複数のタスクに有効な要件がある場合、最も**優先度**の高いタスクが実行されます。
- 複数のタスクに有効な条件があり優先度が同等である場合、待機時間が最も長いタスクが初めに実行されます。
- POU (プログラムタイプ) の呼び出し処理は、**タスクエディター**の順序 (上から下) に従って実行されます。同じ名前の POU がアプリケーションに割り当てられた**アプリケーションツリー**だけでなく、**アプリケーションツリー**の**グローバル**ノードのライブラリーまたはグローバルプロジェクトにもある場合、その POU が呼び出されると**アプリケーションツリー**の下に直接宣言されている方が実行されます。

第 21 章
ウォッチリストエディター

この章について

この章には次の項目が含まれています。

ウォッチビュー / ウォッチリストエディター

概要

ウォッチリストは、ユーザー定義プロジェクト変数のセットです。表の値を監視 (*388* ページ) するためにウォッチビューに表示されます。また、ウォッチビュー内で変数の書き込みおよび強制ができます。

ウォッチコマンドサブメニュー (デフォルトでは**表示**メニュー内) からウォッチビューを開きます。ウォッチリストの作成 (*387* ページ) エディターが表示されます。

初期設定では、**ウォッチ 1**、**ウォッチ 2**、**ウォッチ 3**、および**ウォッチ 4** のウォッチビューに 4 つの個別のウォッチリストを設定できます。オンラインモードの**すべての強制をウォッチ**ビューは、有効なアプリケーションで現在強制されているすべての値が自動的に表示されます (*388* ページ)。

ウォッチリストの作成

手順

次の手順に従ってウォッチリストを作成します。

手順	手順内容	コメント
1	**表示 → ウォッチ → ウォッチ <n>** コマンドを実行し**ウォッチ**ビューを開きます。	–
2	**式**列のフィールドをダブルクリックし、監視する変数を手動、または**入力アシスタント**を使用して入力します。 構文： <device name>.<application name>.<object name>.<variable name>	**注記：**構造変数の名前を入力すると、オンラインモードでは特定の構造コンポーネントが自動的にそれ以降の行に入力されます。
3	その後、監視したいすべての変数を現在のウォッチリストに指定します。ドラッグ & ドロップで項目の順序を変更できます。	**Execution point**、**タイプ**、**アドレス**、**コメント**の各フィールドは、変数の宣言にあわせて自動的に入力されます。式の前の記号は、それが入力変数 、出力変数  、または通常の変数  のどれであるかを示します。

オンラインモード

オンラインモードでは、現在の変数値が**値**の列に表示されます。他の監視位置と同様に、**設定済みの値**を定義してそれらを強制または書き込むことができます。詳細は、*値の強制*、*値の準備ダイアログボックス (EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*、および*監視*の章を参照してください (*213* ページ)。

オンラインモードでは、**ウォッチを追加**コマンドを実行して、現在フォーカスされている式をウォッチリストに追加できます。

オンラインモードのウォッチリスト

監視

オンラインモードのウォッチリスト (*386* ページ) (**ウォッチ <n>**) には、**値**の列に変数の現在値が表示されます。これは、2 つのタスクサイクル間の変数の値です。

また、割り当てられた直接 IEC アドレスおよびコメントが表示される場合もあります。ビューの内容は、宣言エディター (*346* ページ) のオンラインビューの内容に対応しています。

ウォッチリストの設定方法および構造化変数の折りたたみ方法については *ウォッチリストの作成* (*387* ページ) の章を参照してください。

オンラインモードの**ウォッチ**ビュー

注記：オンラインモードで、**ウォッチを追加**コマンドを使用してウォッチリストに式を追加できます。

値の書き込みおよび強制

設定済みの値の列には、**値の書き込み** または**値の強制**コマンドによってコントローラーのそれぞれの式に書き込み、または強制する値を入力できます。他の監視ビュー (例えば、宣言エディター) でも使用できる**書き込み**および**強制**コマンドの説明を参照してください。

すべての強制をウォッチ

これは特別なウォッチリストビューです。オンラインモードでは、有効なアプリケーションで現在強制されているすべての値が自動的に表示されます。**ウォッチ <n>** リストのオンラインビューにそれぞれの**式**、**タイプ**、**値**、および**設定済みの値**が表示されます。

強制解除 ... ボタンから次のコマンドのいずれかで値の強制を解除できます。

- **すべての選択されている値を強制しないようにして維持します**
- **すべての選択されている値を強制しないようにして復元します。**

すべての強制をウォッチダイアログボックス

第 22 章
ロジックエディターのツール

この章について

この章には次の項目が含まれています。

ファンクションおよびファンクションブロック検索

概要

EcoStruxure Machine Expert には、正確な名前がわからなくても特定のファンクションまたはファンクションブロックを見つけるための FFB (ファンクションおよびファンクションブロック) 検索があります。

ファンクションおよびファンクションブロック検索は、ファンクションブロックを挿入できる次のプログラミング言語で使用できます。

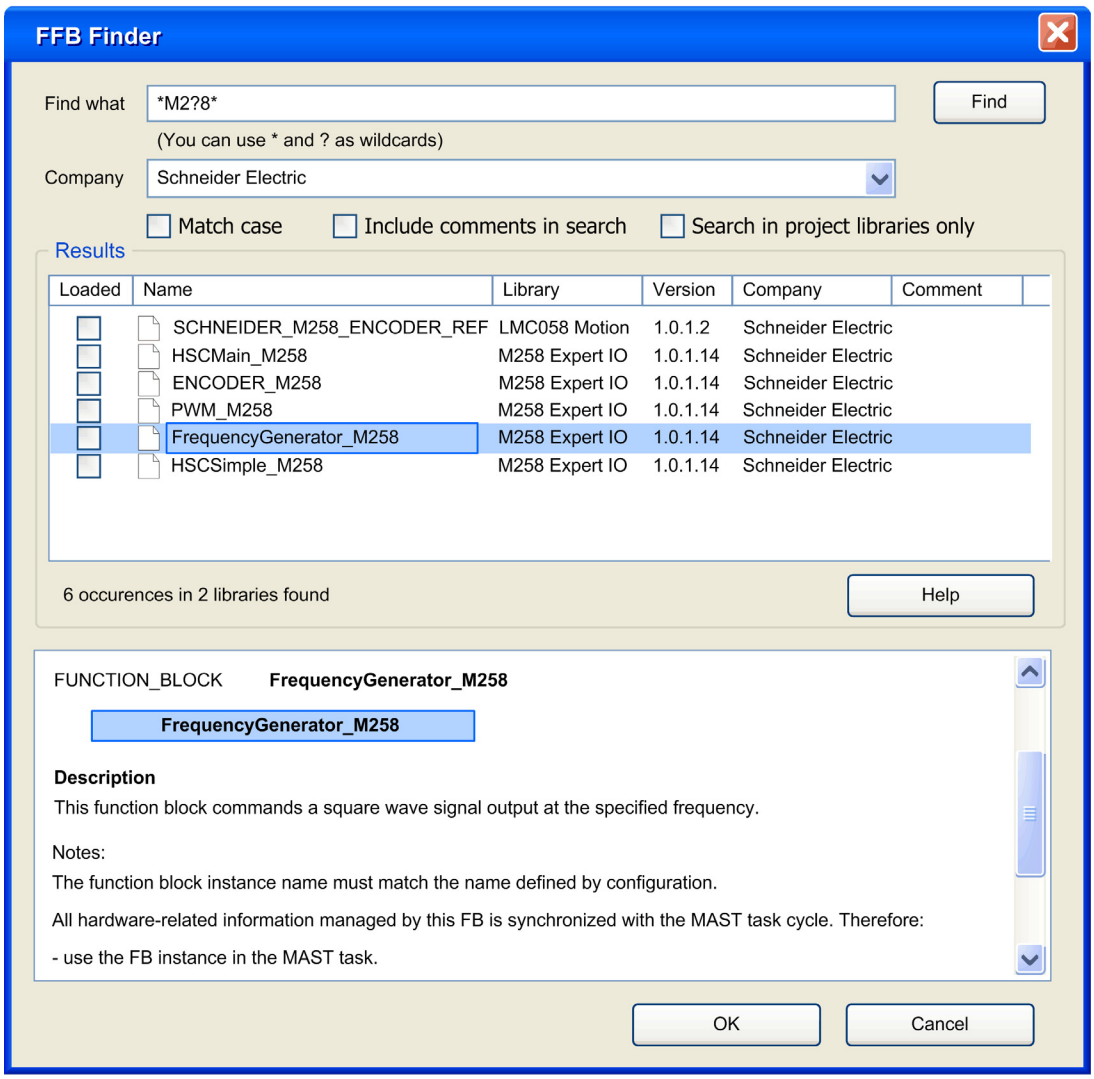
- CFC
- LD
- IL
- FBD
- ST

FFB 検索を使用したファンクションまたはファンクションブロックの検索方法

EcoStruxure Machine Expert Logic Builder でプログラミングコードを作成するときに、ファンクションブロックを挿入する位置に移動し、次のように FFB 検索を開きます。

- **編集 → FFB 検索** メニューを選択します。
  または
- エディターの各場所を右クリックし、コンテキストメニューから **FFB 検索 ...** コマンドを選択します。

FFB 検索ダイアログボックスが開きます。



FFB 検索ダイアログボックスには、ファンクションまたはファンクションブロックを検索するための次の要素が表示されます。

要素	詳細
検索	**検索**テキストボックスには、プログラミングコードに挿入するファンクションまたはファンクションブロックの名前を入力します。 ワイルドカードとして、1 文字のみを置き換える疑問符 (?)、または複数の文字 (または 0 文字) を置き換えるアスタリスク (*) を使用できます。
会社	検索するファンクションブロックを含むライブラリーを作成した会社名がわかっている場合は、**会社**リストから会社名を選択できます。 このパラメーターは、初期設定では**すべての会社**に設定されています。
大文字小文字を区別する	大文字と小文字を区別して検索するには、**大文字と小文字を区別する**チェックボックスをチェックします。 初期設定では、このチェックボックスは選択されていません。
検索コメントを含む	**検索コメントを含む**チェックボックスをチェックすると、入力した文字列をファンクションおよびファンクションブロックの名前だけでなく、それらに保存されているコメント内も検索します。 初期設定では、このチェックボックスは選択されていません。
プロジェクトライブラリのみを検索	**プロジェクトライブラリのみを検索**チェックボックスをチェックすると、検索を現在のアプリケーションで使用されているライブラリーのみに限定します。 初期設定ではこのチェックボックスは選択されておらず、EcoStruxure Machine Expert パソコンにインストールされているライブラリーをすべて検索します。
検索	ファンクションまたはファンクションブロックの検索を開始するには、**検索**ボタンをクリックするか、ENTER キーを押します。

FFB 検索で返される結果

入力した検索条件と一致するファンクションまたはファンクションブロックは、次の情報と共に**結果**リストに表示されます。

- ファンクションまたはファンクションブロックの**名前**
- ファンクションまたはファンクションブロックが保存されている**ライブラリー**
- ライブラリーの**バージョン**
- ライブラリーを作成した**会社**
- コメントがあれば、右側の列に表示されます。
- 左側の**読み込み済み**の列は、ファンクションまたはファンクションブロックが保存されているライブラリーが現在のプロジェクトですでに使用されているかを示します。

ファンクションまたはファンクションブロックに関する詳細情報を表示するには、リストから選択します。下部のフィールドには、入力および出力付きのファンクション / ファンクションブロックの図が表示されます。説明や詳細情報もあれば表示されます。

ファンクション / ファンクションブロックのプログラミングコードへの統合

FFB 検索で見つかったファンクション / ファンクションブロックをプログラミングコードに統合するには、**結果**リストから選択し、次の操作のいずれかを行います。

- **結果**リストで選択したファンクション / ファンクションブロックをダブルクリック。
- **OK** ボタンをクリック。

選択したファンクション / ファンクションブロックは、プログラミングコードのカーソルのある位置に挿入され、対応するライブラリーが自動的に読み込まれます。

特定のファンクション / ファンクションブロックを探すときは、この操作を繰り返します。

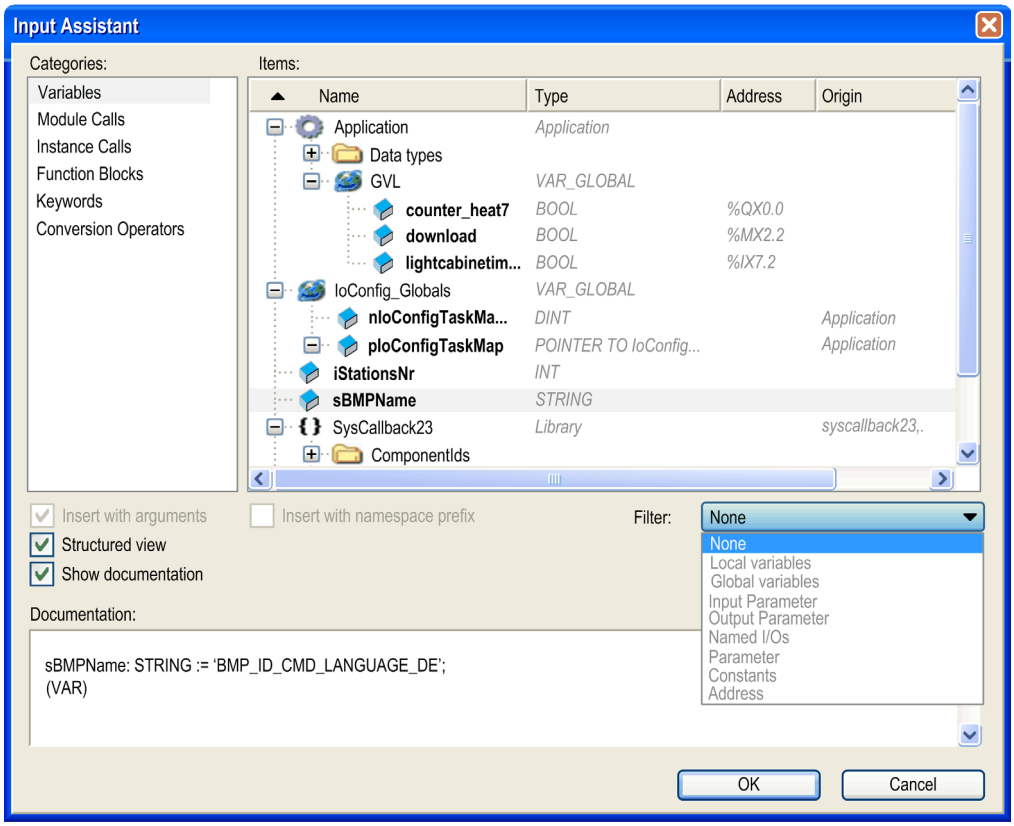
入力アシスタント

概要

入力アシスタントダイアログボックスおよび対応する**入力アシスタント**コマンド (初期設定では、**編集 → スマートコーディング**メニュー) は、カーソルがテキストエディターウィンドウにある場合のみ使用できます。ダイアログボックスには、現在のカーソル位置に挿入できるプロジェクト項目が表示されます。

デフォルトショートカット : **F2**

入力アシスタントダイアログボックス



要素の説明

入力アシスタントダイアログボックスには次の要素が表示されます。

要素	詳細
カテゴリー	この領域では、プロジェクトの項目は**カテゴリー**別に並べ替えられます。
フィルター	**変数**カテゴリーの**フィルター**を設定できます。特定のタイプの変数を表示するには、リストから**ローカル変数**、**グローバル変数**、**定数**などの項目を選択します。
項目	
名前、**型**、**アドレス**、**オリジン**	**項目**には、使用できる項目とカテゴリーで選択した**カテゴリー**に応じたデータ**型**、**アドレス**、**オリジン**が表示されます。 **オリジン**は、I/O 変数 (**デバイスツリー**内のパス) およびライブラリー定義変数 (ライブラリー名とカテゴリー) 用に表示されます。 項目は、**名前**、**型**、**アドレス**、**オリジン**でアルファベット順に昇順または降順に並べ替えできます。これを実行するには、各列のヘッダー (上矢印または下矢印) をクリックします。 **型**、**アドレス**または**オリジン**の列を非表示または表示するには、各列のヘッダーを右クリックします。

要素	詳細
構造化表示	**構造化表示**オプションを選択すると、プロジェクトの項目がアイコンが追加された構造ツリーで表示されます。 このオプションを選択していない場合、プロジェクトの項目は均一に配置されます。各プロジェクト項目は、それが属する POU と共に表示されます (例 : **GVL1.gvar1**)。

注記：同じ名前のオブジェクトが**アプリケーションツリー**の**グローバル**ノードだけでなくアプリケーション (**アプリケーションツリー**) の下にもある場合、オブジェクトの使用は通常の呼び出し優先度 (最初にアプリケーションに割り当てられたオブジェクト、その後にグローバルなオブジェクト) によって決まるため、**入力アシスタント**には 1 つのみ表示されます。

要素	詳細
Show documentation	**ドキュメントの表示**オプションが選択されている場合、**入力アシスタント**ダイアログボックスが拡張され**ドキュメント**フィールドが表示されます。 選択した要素が変数であり、この変数にアドレスが割り当てられている場合、またはその宣言にコメントが追加されている場合はここに表示されます。
引数を挿入する	このオプションを選択すると、例えば、ファンクションなどの引数を含む項目はその引数とともに挿入されます。 例 : 入力変数 fb1_in および出力変数 fb1_out を含むファンクションブロック FB1 に引数を挿入すると、エディターに次のように記述されます。 `fb1(fb1_in:= , fb1_out=> )`
ネームスペースの接頭語を挿入する	このオプションを選択すると、接頭辞付きの名前空間が挿入されます。 **プロパティ**で名前空間の接頭辞の使用が必須であると定義されているライブラリーにオブジェクトを挿入するときは、このオプションを選択する必要があります (**Only allow qualified access to all identifiers** *(EcoStruxure Machine Expert, ファンクションおよびライブラリーユーザーガイド 参照)* オプションを選択)。

第 VI 部
ツール

このパートについて

このパートには次の章が含まれています。

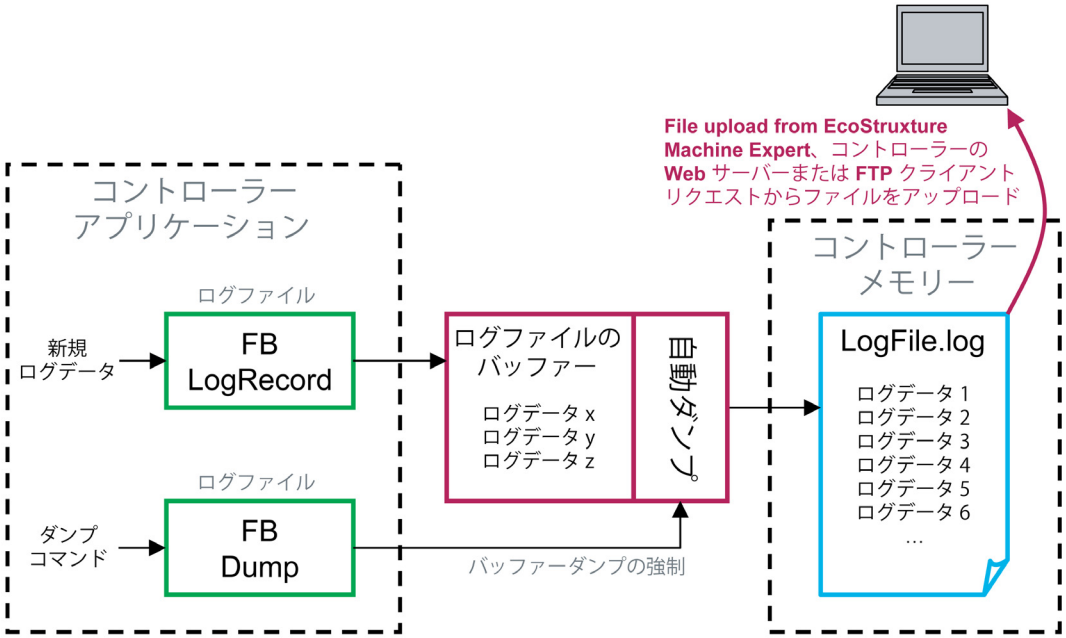
章	章タイトル	参照ページ
23	データロギング	397
24	レシピマネージャー	399
25	トレースエディター	415
26	トレンドの記録	437
27	単位の変換	451
30	Cam モーションエディター	457

第 23 章
データロギング

データロギングの概要

概要

データログファイル (.log) を調べることで、アプリケーションデータを監視および解析できます。



この図は、2 つのファンクションブロック LogRecord と Dump を含むアプリケーションを示しています。LogRecord ファンクションブロックによってデータがバッファーに書き込まれ、コントローラーメモリーにあるログファイル (.log) が空になります。バッファーのダンプはメモリーが 80% 以上になったときに自動的に行われるか、Dump 機能によって強制的に行わせることができます。コントローラーが FTP サーバーとして機能する場合、標準の FTP クライアントとしてパソコンからこのデータログファイルにアクセスできます。EcoStruxure Machine Expert またはコントローラー Web サーバーでファイルをアップロードすることもできます。

注記：ファイル管理機能を備えたコントローラーでのみ、データロギングをサポートしています。コントローラーのプログラミングマニュアルを参照して、ファイル管理がサポートされているかを確認してください。

サンプルデータログファイル (.log)

Entries in File: 8; Last Entry: 8;

18/06/2009;14:12:33;cycle: 1182;

18/06/2009;14:12:35;cycle: 1292;

18/06/2009;14:12:38;cycle: 1450;

18/06/2009;14:12:40;cycle: 1514;

18/06/2009;14:12:41;cycle: 1585;

18/06/2009;14:12:43;cycle: 1656;

18/06/2009;14:14:20;cycle: 6346;

18/06/2009;14:14:26;cycle: 6636;

実装手順

プログラムの作成を開始する前に、アプリケーションでデータログファイルを宣言して設定します。

第 24 章
レシピマネージャー

この章について

この章には次の項目が含まれています。

レシピマネージャー

概要

レシピマネージャーによって、レシピ定義と呼ばれるプロジェクト変数のユーザー定義リスト、およびレシピと呼ばれるレシピ定義内のプロジェクト変数用の値を処理します。

レシピを使用して、コントローラーの特定の変数セット (レシピ定義) のレシピ値を変更または読み取ることができます。また、ファイルから読み込み、保存もできます。これらの操作は、適切に設定されたビジュアライゼーション要素 (入力設定実行コマンド) を使用することで実行できます。アプリケーションのレシピコマンド (*407* ページ) を使用することもできます。

レシピを選択する時は、レシピが制御する処理に適していることを確認してください。

⚠ 警告

装置の意図しない動作

- インストールされたアプリケーションおよび機器の安全解析を実施してください。
- レシピが、設置処理、設備または機能に適していることを確認してください。
- 特に制限やその他の安全関連要素に適切なパラメーターを指定してください。
- すべてのセンサーとアクチュエーターが、選択されたレシピと互換性があることを確認してください。
- 検証と試運転中にすべての機能を徹底的にテストしてください。
- 安全解析および適用されるコードおよび規制に従って、重要な制御機能 (緊急停止、過限条件など) のための独立したパスを提供してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

デフォルトでは、レシピマネージャーはダウンロード中にコントローラーに読み込まれます。アプリケーションがコントローラー上で実行中に、レシピの書き込みと読み取りを処理します。ただし、システム起動時 (EcoStruxure Machine Expert がまだコントローラーに接続されている時) のパラメーター送受信のためだけにレシピを使用する場合は、レシピマネージャーをコントローラーに読み込ませる必要はありません。その場合は **PLC 内のレシピ管理**オプションを使用して、ダウンロードを無効にすることができます。レシピ値の書き込みと読み取りは、標準のオンラインコマンドとサービスによって処理されます。実行時のアプリケーションプログラムで、レシピ管理がコントローラーで実行される必要がある場合、RecipeCommands ファンクションブロックによってレシピコマンドを処理します。

それぞれのオンラインモードにおけるレシピの動作については、*レシピ定義* (*403* ページ) の章を参照してください。

レシピマネージャーがレシピの影響を受けるアプリケーション以外のコントローラーにある場合、レシピに含まれる変数の読み書きにはデータサーバーが使用されます。変数の読み書きは同期して行われます。読み書きの後に g_RecipeManager.LastError を呼び出すことにより、送信が成功したかどうかを確認することができます (この場合は、g_RecipeManager.LastError=0)。

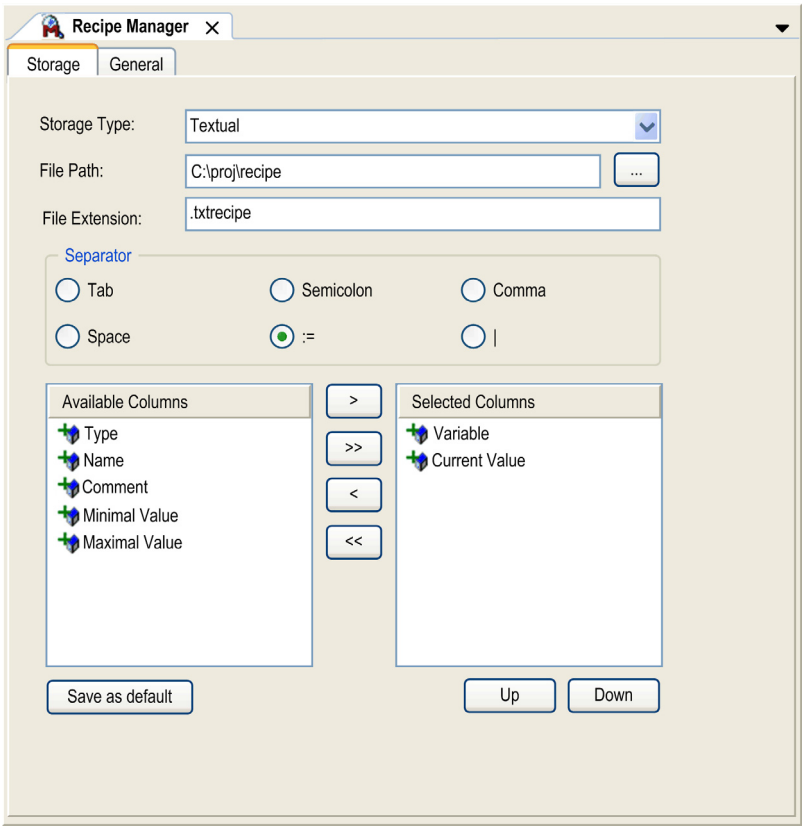
ツールツリーのレシピ管理オブジェクト

レシピマネージャーオブジェクトを**ツールツリー**に追加するには、**アプリケーション**ノードを選択し、緑のプラスボタンをクリックして **Add Other Objects... → レシピマネージャー ...** コマンドを実行します。**追加**をクリックして**レシピマネージャーの追加**ダイアログボックスで**レシピマネージャー**ノードが**アプリケーション**ノードの下に追加されているのを確認します。

1 つまたは複数の**レシピ定義**オブジェクトを**レシピマネージャー**ノードに追加することができます。これを実行するには、**レシピマネージャー**ノードの緑色のプラスボタンをクリックし、**レシピ定義 ...** コマンドを実行します。**レシピ定義の追加**ダイアログボックスの**名前**を入力し、**追加**をクリックします。ノードをダブルクリックして、別のエディターウィンドウの特定のレシピを含むレシピ定義を表示および編集します。それぞれのオンラインモードにおけるレシピの動作については、*レシピ定義* (*403* ページ) の章を参照してください。

レシピマネージャーエディター、保存タブ

デフォルトでは、レシピは**レシピマネージャー**エディターの**保存**タブの設定に従ってファイルに自動的に保存されます。



パラメーター	詳細
保存タイプ	**テキスト**または**バイナリー**の保存タイプを選択します。
ファイルパス	レシピをコントローラーまたはローカルファイルシステムに保存する場所を指定します。コントローラーでは、パスは MyRecipes¥ のように円記号 (¥) で終わる必要があります。
ファイル拡張子	レシピファイルのファイル拡張子を指定します。

注記：保存ファイルは、ビジュアライゼーション要素の入力によって定義することもできます (入力構成 - 実行コマンド - ファイルからレシピを保存 / ロードする)。ただし、ビジュアライゼーション設定でそのようなファイルの名前を定義するときは、ここで定義したレシピマネージャーの *.txtrecipe ファイルを上書きしないでください。

パラメーター	詳細
セパレーター	テキストで保存する場合、保存用に選択された列はセパレーターで区切られます。6 つのオプションから 1 つを選択します。
利用可能な列	それぞれのヘッダーで表されたレシピ定義のすべての列。
選択された列	レシピ定義の選択された列、つまり格納される列。 少なくとも**現在値**を含む列はこの部分に含まれます。選択を解除することはできません。
矢印ボタン	それぞれの項目を選択して矢印ボタンをクリックすると、列を右または左に移動できます。また、二重矢印ボタンを使用して一度にすべての項目を一方から他方に移動することもできます。
上へと**下へ**ボタン	これらのボタンをクリックして、選択した列の順序を調整します。これは、保存ファイル内の列の順序を表します。 レシピごとに、< レシピ名 > . < レシピ定義 > . < ファイル拡張子 > という名のファイルが、指定されたフォルダーに作成されます。このファイルは、アプリケーションを再起動するたびにレシピマネージャーに再読み込みされます。レシピ保存ファイルの更新設定については、**全般** タブ (*402* ページ) の説明を参照してください。

パラメーター	詳細
デフォルトとして保存	**デフォルトとして保存**ボタンをクリックすると、このダイアログボックス内の設定が、レシピマネージャーを挿入するたびに初期設定として使用されます。

注記：浮動小数点値 (REAL/LREAL 型) は、テキスト形式のレシピファイルに 16 進形式と同様に 10 進形式で格納されます。(16 進値は正確な値を表し、10 進 REAL 値は小数点第 7 位までの値を表すためです。)

例 : PLC_PRG.realVar:=22.0F16#1600000H-5

レシピファイル内の値を手動で変更するには、10 進値を編集して、それに続く 16 進値を削除します。(両方の値が使用可能な場合は、16 進値が読み込まれます。)

レシピマネージャーエディター、全般タブ

パラメーター	詳細
PLC 内のレシピ管理	アプリケーションの実行時に処理するレシピがないなど、コントローラーのレシピマネージャーが不要な場合は、このオプションを無効にするとマネージャーはダウンロードされません。 レシピファイルの自動更新は、ダウンロードが実行された後にのみ可能です。コントローラーにレシピ管理をダウンロードするには、このオプションを選択します。
レシピを保存	
レシピの変更をレシピファイルに自動的に保存します	このオプションは、**PLC 内のレシピ管理**が選択されている場合にのみ使用できます。 **レシピの変更をレシピファイルに自動的に保存します**オプションを選択すると、レシピが変更されるたびにレシピファイルがランタイムモードで自動的に更新されます。
レシピをロード	
変数リストの完全一致のみをロードします	**変数リストの完全一致のみをロードします**オプションを選択すると、アプリケーションのレシピ定義の変数リストにあるすべての変数がファイルに含まれている場合にのみレシピファイルを読み込みます。ファイル内の変数は、変数リストと同じ順序でなければなりません。末尾の追加エントリーは無視されます。そうではない場合、レシピをロードできず、戻り値 (*408* ページ) ERR_RECIPE_MISMATCH が設定されます (RecipeManCommands.GetLastError)。
変数名で一致する変数をロードします	**変数名で一致する変数をロードします**オプションを選択するとレシピファイルから変数名が一致する変数のみを読み込みます。ファイル内の変数リストがアプリケーションのレシピ定義内のリストと異なる場合、エラーステータスは設定されません。したがって、ファイル内またはレシピ定義内の変数が削除されている場合は、レシピファイルが読み込まれます。
レシピを書き込み	
Limit the variable to min/max when recipe value is out of the range	**Limit the variable to min/max when recipe value is out of the range** オプションを選択すると、レシピに値の範囲を超える値が含まれている場合に定義された最小値または最大値がコントローラーに書き込まれます。
Do not write to a variable when the recipe value is out of the min/max range	**Do not write to a variable when the recipe value is out of the min/max range** オプションを選択すると、レシピに値の範囲を超える値が含まれている場合、値はコントローラーに書き込まれません。コントローラーに存在する値は保持されます。

レシピの書き込み

アプリケーションがオンラインモードのときに**レシピを書き込み**ボタンをクリックすると、選択したレシピの値がコントローラーの変数に書き込まれます。

注記：コントローラーの現在値はレシピの値で上書きされます。

レシピ定義

概要

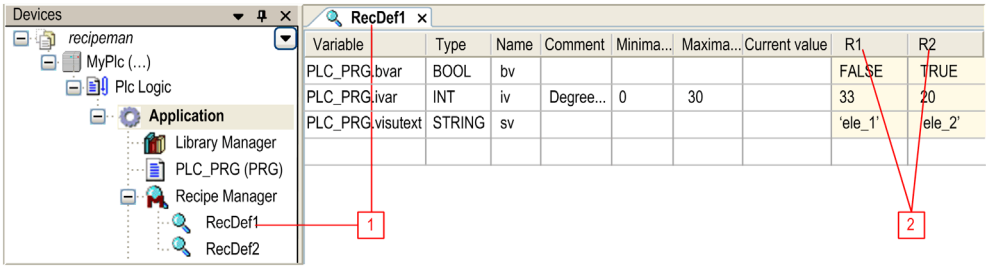
レシピマネージャー (*400* ページ) では、1 つまたは複数のレシピ定義が処理されます。レシピ定義には、変数のリストとおよびこれらの変数の 1 つまたは複数のレシピ (値のセット) が含まれています。レシピをファイルに保存したり、レシピファイルをコントローラーに書き込むことができます。異なるレシピを使用することで、コントローラー上の一連の変数に別の値のセットを 1 回の操作で割り当てることができます。レシピごとのレシピ定義、レシピ、および変数の数に制限はありません。

レシピ定義

1 つまたは複数の**レシピ定義**オブジェクトを**ツールツリー**の**レシピマネージャー**ノードに追加することができます。これを実行するには、**レシピマネージャー**ノードの緑色のプラスボタンをクリックし、**レシピ定義 ...** コマンドを実行します。

ノードをダブルクリックすると、別のエディタービューで特定のレシピを含むレシピ定義が表示され、編集できます。

レシピ定義エディタービュー



1 レシピ定義名
2 レシピ名

エディターウィンドウには、レシピ定義の名前が付いています。

パラメーター	詳細
変数	表には、1 つまたは複数のレシピを定義する複数のプロジェクト変数を入力できます。カーソルが任意の行の任意のフィールドにあるときに、**変数の挿入**コマンドを使用することができます。代わりに**変数**フィールドをダブルクリックするか、またはそれを選択してからスペースバーを押してエディターモードにすることもできます。有効なプロジェクト変数名を入力します。例えば、plc_prg.ivar。**...** ボタンをクリックすると、**入力アシスタント**が開きます。 例えば上の図に示した PLC_PRG のようなプログラム、POU を指定することもできます。この場合、POU 内で定義されているすべての変数は、入力フィールドを閉じると自動的にレシピ定義に追加されます。データ型やファンクションブロックについても同様です。 右側のボタンを使用して、標準ビューと構造ビューを切り替えることができます。 構造データ型または POU の宣言を変更した後は、関連する変数によってレシピ定義を自動的に縮小または拡張できます。詳細については、**構造変数の変更** コマンドを参照してください *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
タイプ	**タイプ** フィールドは自動的に入力されます。オプションで、シンボルの**名前**を定義することができます。
名前	シンボルの**名前**を定義することができます。
コメント	変数に記録されている値の単位などの追加情報を入力します。
最小値と最大値	オプションで、この変数に書き込むための許容値を指定することができます。
現在値	この値は、オンラインモードで監視されます。
レシピの変更をレシピファイルに自動的に保存します	レシピ管理エディターの通常の動作に影響するため、**レシピマネージャー**エディターの**全般**タブでこのオプションを有効にすることを推奨します。実行時にレシピが変更されるとすぐに保存ファイルが更新されます。このオプションは、レシピマネージャーがコントローラー上で使用可能な場合に限り有効であることを考慮してください。

セルの 1 つが選択されているときに DEL キーを押すと、表から変数 (行) を削除できます。複数の行を選択するには、Ctrl キーを押したままセルを選択します。コピーと貼り付けで、選択した行をコピーすることができます。貼り付けコマンドは、選択されている行の上にコピーした行を挿入します。そうすることでレシピ値が一致するレシピ列に挿入されます (使用可能な場合)。

レシピをレシピ定義に追加するには、エディタービューにカーソルがあるときに**新しいレシピの追加** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を実行します。レシピごとに、レシピ名の付いた独自の列が作成されます（例 : 上の表の **R1**、および **R2**）。

オンラインモードでは、適切に設定されたビジュアライゼーション要素 (入力設定実行コマンド) 、または RecipeManCommands ファンクションブロックの適切なメソッドのいずれかを使用して、レシピを変更することができます。

レシピ定義エディタービューのレシピ列のコンテキストメニューでは、次の方法を使用できます。

- ReadRecipe: 現在の変数値がレシピに取り込まれます。
- WriteRecipe: レシピが変数に書き込まれます。
- SaveRecipe: レシピは標準レシピファイルに保存されます。
- LoadRecipe: レシピは標準レシピファイルから読み込まれます。
- CreateRecipe: レシピ定義に新しいレシピが作成されます。
- DeleteRecipe: レシピ定義から既存のレシピが削除されます。

次の項で、特定のオンライン状態でのレシピの動作を示します。レシピ管理を一般的な動作にするために、**レシピの変更をレシピファイルに自動的に保存します**オプションを設定することを推奨します。

レシピ

レシピをオフラインまたはオンラインで追加または削除することができます。オフラインモードでは、レシピマネージャーエディターで**新しいレシピの追加** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドと**レシピの削除** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドを使用します。オンラインモードでは、適切に設定されたビジュアライゼーション要素の入力を設定するか、またはレシピマネージメントライブラリーの RecipeManCommands ファンクションブロックの適切なメソッドを使用します。

レシピを追加するとき、レシピの名前が付いた右端の列の後ろに列が追加されます (レシピ定義エディタービューの図を参照)。レシピ列のフィールドには、適切な値を入力することができます。従って、同じセットの変数に対して、異なるセットの値を特定のレシピで準備することができます。

オンラインモードでのレシピの使用

レシピは、アプリケーションコードのライブラリー (Recipe_Management.Library) またはビジュアライゼーション要素の入力によって提供される RecipeManCommands ファンクションブロックのメソッドを使用することにより処理 (作成、読み込み、書き込み、保存、削除) することができます。

レシピの変更をレシピファイルに自動的に保存しますが有効なときのオンラインモードのレシピ処理。

アクション	プロジェクトで定義されたレシピ	実行中に作成されたレシピ
オンラインウォームリセット **オンラインコールドリセット** **ダウンロード**	すべてのレシピ定義のレシピは、開いているプロジェクトの値で設定されます。	動的に作成されたレシピは、変更されず残ります。
オンライン元にリセット	アプリケーションが、コントローラーから削除されます。後に新しいダウンロードが行われると、**オンラインウォームリセット**と同様にレシピが復元されます。	
コントローラーがシャットダウンし再起動します。	再起動後、レシピは自動的に作成されたファイルから再読み込みされます。従って、シャットダウン前の状態が復元されます。	
オンライン変更	レシピの値は変更されません。実行中、レシピは RecipeManCommands ファンクションブロックのコマンドでのみ変更できます。	
停止	コントローラーの停止 / 開始時に、レシピは変更されません。	

注記 : 浮動小数点値 (REAL/LREAL 型) は、テキスト形式のレシピファイルに 16 進形式と同様に 10 進形式で格納されます。(16 進値は正確な値を表し、10 進 REAL 値は小数点第 7 位までの値を表すためです。)

例 : PLC_PRG.realVar:=22.0F16#1600000H-5

レシピファイル内の値を手動で変更するには、10 進値を編集して、それに続く 16 進値を削除します。(両方の値が使用可能な場合は、16 進値が読み込まれます。)

レシピの変更をレシピファイルに自動的に保存しますが無効なときのオンラインモードのレシピ処理。

アクション	プロジェクトで定義されたレシピ	実行中に作成されたレシピ
オンラインウォームリセット **オンラインコールドリセット** **ダウンロード**	すべてのレシピ定義のレシピは、開いているプロジェクトの値で設定されます。ただし、これらはメモリー内でのみ設定されます。レシピをファイルに保存するには、保存コマンドを明示的に使用する必要があります。	動的に作成されたレシピは削除されます。
オンライン元にリセット	アプリケーションが、コントローラーから削除されます。後に新しいダウンロードが行われると、レシピが復元されます。	動的に作成されたレシピは削除されます。
コントローラーがシャットダウンし再起動します。	再起動後、レシピはプロジェクトで設定された値からダウンロード時に作成された初期値で再読み込みされます。従って、シャットダウン前の状態は復元されません。	
オンライン変更	レシピの値は変更されません。実行中、レシピは RecipeManCommands ファンクションブロックのコマンドでのみ変更できます。	
停止	コントローラーの停止 / 開始時に、レシピは変更されません。	

詳細情報 :

- アプリケーションの再起動時に再読み込みされるファイル内のレシピの保存については、*レシピマネージャーエディター、保存タブ* (*401* ページ) の説明を参照してください。
- 特定の RecipeManCommands メソッド (*407* ページ) の説明については、ライブラリー内のドキュメントを参照してください。
- ビジュアライゼーション要素の適切な入力設定については、そのヘルプページ (カテゴリー**入力 → 実行コマンド**) を参照してください。

レシピでは次のアクションが可能です。

アクション	詳細
レシピの作成 (= **新しいレシピの追加**)	新しいレシピが、指定されたレシピ定義で作成されます。
レシピの取り込み	指定されたレシピ定義の変数値をコントローラーから読み込み、指定されたレシピに書き込みます。これは、値が暗黙的に (コントローラー上のファイルに) 格納されることを意味します。さらに、**レシピマネージャー**のレシピ定義テーブルで直ちに監視されます。つまり、**レシピマネージャー**で管理されているレシピは、コントローラーの実際の値で更新されます。
レシピを書き込み	レシピマネージャーに表示されている、指定されたレシピの値がコントローラー上の変数に書き込まれます。
レシピを保存	指定したレシピの値は、**.txtrecipe* の拡張子をつけてファイルに書き込まれます。ローカルファイルシステムにファイルを保存するためのダイアログボックスが開きます。 **注記 :** レシピ値の読み取りと書き込みのバッファーとして必要な暗黙的に使用されるレシピファイルは、上書きされないことがあります。これは、新しいレシピファイルの名前は、< レシピ名 > 、< レシピ定義名 > .txtrecipe とは異なる事を意味します。
レシピをロード	ファイルに保存されているレシピ (**レシピの保存**の説明を参照) は、このファイルから再読み込みすることができます。ファイルを参照するためのダイアログボックスが開きます。フィルターは、拡張子 **.txtrecipe* に自動的にセットされます。それに応じて、ファイルが再読み込みされた後、レシピ値がレシピマネージャーで更新されます。
レシピの削除 (= **レシピの削除**)	指定されたレシピが、レシピ定義から削除されます。
レシピの変更	プロジェクト変数の値を変更することができます。次のレシピ書き込みアクションで、適切なプロジェクト変数に新しい値が書き込まれます。

Modicon M241 および M251 ロジックコントローラー用の特定タスクの作成

レシピファイルへのアクセス（作成、読み込み、書き込み、削除）が必要な Modicon M241 および M251 ロジックコントローラーの場合は、優先度が低く、**ウォッチドッグ**機能が無効の特定のタスク (*380* ページ) を作成します。そうでない場合、アプリケーションのサイクルタイムに影響を与える可能性があります。

RecipeMan コマンド

概要

レシピコマンドを呼び出すと、内部データアクセスが実行されます。デバイスのタイプによっては、数ミリ秒かかります。これらの呼び出しが、MAST タスク、ウォッチドッグが設定されたタスク、またはリアルタイムタスクによって実行されていないことを確認します。これによりアプリケーションエラーが発生し、コントローラーが HALT 状態になる可能性があります。

レシピの変更をレシピファイルに自動的に保存しますオプションもまた、レシピの変更ごとにファイルアクセスを実行することを考慮してください。レシピの保存がアプリケーションによってトリガーされている場合は、このオプションを無効にします。

戻り値

レシピコマンドの戻り値は以下のとおりです。

戻り値	詳細
ERR_NO_RECIPE_MANAGER_SET	コントローラーにレシピマネージャーがありません。
ERR_RECIPE_DEFINITION_NOT_FOUND	レシピの定義が存在しません。
ERR_RECIPE_ALREADY_EXIST	レシピはすでにレシピ定義に存在します。
ERR_RECIPE_NOT_FOUND	レシピはレシピ定義に存在しません。
ERR_RECIPE_FILE_NOT_FOUND	レシピファイルが存在しません。
ERR_RECIPE_MISMATCH	レシピファイルの内容が現在のレシピと一致しません。 **注記：**この戻り値は、格納タイプがテキストで、ファイル内の変数名がレシピ定義の変数名と一致しない場合にのみ生成されます。レシピファイルが読み込まれていません。
ERR_RECIPE_SAVE_ERR	レシピファイルへの書き込みアクセスができません。
ERR_FAILED	操作に失敗しました。
ERR_OK	操作に成功しました。

CreateRecipe

このメソッドは、指定したレシピ定義に新しいレシピを作成し、その後にコントローラーの値を新しいレシピに読み込みます。最後に、新しいレシピがデフォルトファイルに保存されます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ALREADY_EXIST, ERR_FAILED, ERR_OK

CreateRecipeNoSave

このメソッドは、指定したレシピ定義に新しいレシピを作成し、その後にコントローラーの値を新しいレシピに読み込みます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

DeleteRecipe

このメソッドは、レシピ定義からレシピを削除します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

DeleteRecipeFile

このメソッドは、デフォルトレシピファイルをレシピから削除します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_OK

LoadAndWriteRecipe

このメソッドは、デフォルトレシピファイルからレシピを読み込んだ後、レシピをコントローラー変数に書き込みます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadFromAndWriteRecipe

このメソッドは、指定されたレシピファイルからレシピを読み込んだ後、レシピをコントローラー変数に書き込みます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前
FileName:	ファイルの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadRecipe

このメソッドは、デフォルトレシピファイルからレシピをロードします。デフォルトのレシピファイル名は、< レシピ >.< レシピ定義 >.< レシピ拡張子 > です。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

ReadAndSaveRecipe

このメソッドは、コントローラー値をレシピに読み込んだ後、デフォルトのレシピファイルにレシピを保存します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadAndSaveRecipeAs

このメソッドは、コントローラー値をレシピに読み込んだ後、指定されたレシピファイルにレシピを保存します。既存のファイルの内容は上書きされます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前
FileName:	ファイルの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

SaveRecipe

このメソッドはレシピをデフォルトレシピファイルに保存します。既存のファイルの内容は上書きされます。デフォルトのレシピファイル名は、< レシピ >.< レシピ定義 >.< レシピ拡張子 > です。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadRecipe

このメソッドは、コントローラー値をレシピに読み込みます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

WriteRecipe

このメソッドは、レシピをコントローラー変数に書き込みます。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND,
このメソッドは、レシピのリストをファイルシステムから再読み込みします。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK

GetRecipeCount

このメソッドは、レシピ定義と一致するレシピの数を返します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前

戻り値 : -1 : レシピ定義が見つからなかった場合。

GetRecipeNames

このメソッドは、レシピ定義と一致するレシピ名を返します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
pStrings :	レシピの値を格納する文字列
iSize :	文字列の配列のサイズ
iStartIndex :	開始インデックス。スクロール機能に使用できます。

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK

例 :

50 個のレシピがあります。一度に 10 個のレシピ名を表示する表を作成するには、文字列の配列を定義します。

strArr: ARRAY[0..9] OF STRING;

iStartIndex に対応して、レシピ名を特定の領域から読み込むこともできます。

iStartIndex := 0;

0 ~ 9 の名前が返されます。

iStartIndex := 20;

20 ~ 29 の名前が返されます。この例では :

iSize := 10;

GetRecipeValues

このメソッドは、レシピ定義と一致するレシピ変数値を返します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前
pStrings:	レシピの値を格納する文字列
iSize:	文字列の配列のサイズ
iStartIndex:	開始インデックス。スクロール機能に使用できます。
iStringLength:	配列の文字列の長さ

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

例 :

50 個のレシピがあります。一度に 10 個のレシピ名を表示する表を作成するには、文字列の配列を定義します。

strArr: ARRAY[0..9] OF STRING;

iStartIndex に対応して、レシピ名を特定の領域から読み込むこともできます。

iStartIndex := 0;

0 ~ 9 の値が返されます。

iStartIndex := 20;

20 ~ 29 の値が返されます。この例では :

iStringLength := 80;
iSize := 10;

GetRecipeVariableNames

このメソッドは、対応するレシピの変数名を返します。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前
pStrings:	レシピの値を格納する文字列
iSize:	文字列の配列のサイズ
iStartIndex:	開始インデックス。スクロール機能に使用できます。

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

例 :

50 個のレシピがあります。一度に 10 個のレシピ名を表示する表を作成するには、文字列の配列を定義します。

strArr: ARRAY[0..9] OF STRING;

iStartIndex に対応して、レシピ名を特定の領域から読み込むこともできます。

iStartIndex := 0;

0 ~ 9 の名前が返されます。

iStartIndex := 20;

20 ~ 29 の名前が返されます。この例では :

iSize := 10;

SetRecipeValues

このメソッドは、対応するレシピにレシピの値をセットします。

パラメーター	詳細
RecipeDefinitionName:	レシピ定義の名前
RecipeName:	レシピの名前
pStrings:	レシピの値を格納する文字列
iSize:	文字列の配列のサイズ
iStartIndex:	開始インデックス。スクロール機能に使用できます。

戻り値 (*407* ページ):
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_NOT_FOUND, ERR_FAILED, ERR_OK

例 :

50 個のレシピがあります。一度に 10 個のレシピ名を表示する表を作成するには、文字列の配列を定義します。

strArr: ARRAY[0..9] OF STRING;

iStartIndex に対応して、レシピ名を特定の領域から読み込むこともできます。

iStartIndex := 0;

0 ~ 9 の値が設定されます。

iStartIndex := 20;

20 ~ 29 の値が設定されます。この例では :

iStringLength := 80;
iSize := 10;

GetLastError

このメソッドは、操作で最後に検出されたエラーを返します。

戻り値 (*407* ページ): ERR_NO_RECIPE_MANAGER_SET, ERR_OK

ResetLastError

このメソッドは、最後に検出されたエラーをリセットします。

戻り値 (*407* ページ): ERR_NO_RECIPE_MANAGER_SET, ERR_OK

コントローラーからレシピ値を読み込む

概要

プロジェクトでレシピの定義が変更されても、コントローラーのレシピ値をプロジェクトのレシピ定義に適用できます。

前提条件として、**PLC 内のレシピ管理**を**レシピマネージャー**エディターの**全般**タブで有効にし、**ファイルパス**を'Recipes¥'に設定する必要があります。

コントローラーからレシピ値を読み込む

コントローラーからレシピ値を読込むには、以下の手順に沿って進めます。

手順	手順内容	コメント
1	プロジェクトで、変数 PLC_PRG.ivar および PLC_PRG.bvar を含むレシピの定義 **RecDef1** を作成します。	–
2	レシピ R1 を挿入します。 PLC_PRG.ivar の値 : 33 PLC_PRG.bvar の値 : TRUE	–
3	コントローラーにログインし、アプリケーションをダウンロードします。	**結果** : レシピファイル R1.RecDef1.txtrecipe がコントローラーの **Recipes** ディレクトリーに保存されます。
4	コントローラーからログアウトします。	–
5	変数 PLC_PRG.dwvar をプロジェクトのレシピ定義に追加します。	–
6	PLC_PRG:ivar の値を 33 から 34 に変更して、コントローラー上のレシピ定義ファイル R1.RecDef1.txtrecipe を 編集します。	–
7	R1.RecDef1.txtrecipe ファイルをコピーし、名前を R2.RecDef1.txtrecipe に変更してもうひとつレシピをコントローラーに追加します。	–
8	R2.RecDef1.txtrecipe ファイルを編集してレシピ値を変更します。 PLC_PRG:ivar:1 PLC_PRG.bvar:FALSE	**結果** : コントローラー上に 2 つのレシピ R1 と R2 が作成されます。 プロジェクトでは、R1 のみが使用可能です。レシピ R1 はコントローラー上の R1 とは異なる値をもちます。
9	コントローラーにログインします。	**結果** : ダイアログボックスが開きます。
10	ダイアログボックスで、**オンライン変更してログイン**または、**ダウンロードしてログイン**オプションを選択し **OK** をクリックします。	–
11	レシピ定義エディタービューが開いているあいだに、**Recipe definition editor view** のコンテキストメニューから **Upload recipes from device** コマンドを実行します。	**結果** : ダイアログボックスに、コントローラーから読込まれたレシピ PLC_PRG.dwvar は値をコントローラーから提供できないことが示されます。
12	ダイアログボックスで続行することを確定します。	**結果** : プロジェクトのレシピ定義のレシピ R1 にある PLC:_PRG.ivar の値が 34 に変更されます。 PLC_PRG.dwvar はレシピ定義に残ります。

レシピのメモリー使用量

メモリー使用量

レシピマネージャーエディターの**全般**タブで **PLC 内のレシピ管理** オプションが有効な場合、レシピマネージャーとレシピ定義のコードが生成されます。このコードはコントローラーに保存されます。メモリー使用量は、変数のデータ型およびレシピとその変数の数によって異なります。レシピ定義のフィールドが入力された場合、それもメモリー容量を必要とします。

メモリー使用量を判断するためのガイドラインとして、次の表の値を使用できます。

–	コードサイズ (バイト)	データサイズ (バイト)	合計 (バイト)
INT 変数 100 個のレシピ定義	194406	79400	267352
INT 変数 200 個のレシピ定義	238318	121284	459344
INT 変数 300 個のレシピ定義	282230	163084	543856
BOOL 変数 100 個のレシピ定義	192742	69884	343168
BOOL 変数 200 個のレシピ定義	235446	101568	436872
BOOL 変数 300 個のレシピ定義	278146	133284	510072
STRING 変数 100 個のレシピ定義	203278	870084	1154000
STRING 変数 200 個のレシピ定義	255570	1709784	2973296
STRING 変数 300 個のレシピ定義	307886	2549484	2964112

第 25 章
トレースエディター

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
25.1	トレースオブジェクト	416
25.2	トレース設定	421
25.3	オンラインモードのトレースエディター	434
25.4	トレースダイアグラムのキーボード操作	435

25.1
トレースオブジェクト

このセクションについて

このセクションには次の項目が含まれています。

トレースの基本

トレースの機能

トレース機能を使用すると、デジタルサンプリングオシロスコープと同様に、特定の時間のコントローラーの変数値の変化を取得することができます。さらに、トリガーを設定して入力 (トリガー) 信号でデータの取得を制御することもできます。トレース変数の値は、指定されたサイズの EcoStruxure Machine Expert バッファーに徐々に書き込まれます。それらは、時間の関数としてプロットされた 2 次元グラフの形で表示することができます。

注記：コントローラーからログオフしても、データのトレースは継続されます。

トレースの実行中に、EcoStruxure Machine Expert または EcoStruxure Machine Expert を実行しているパソコンでプロセッサーを消費するタスクが実行された場合、トレースによって変数値が取得されない場合があります。

注記
データの損失 トレースの実行中は、プロセッサーに高い負荷のかかるアクションやパソコンのアプリケーションの実行は避けてください。 **上記の指示に従わないと、物的損害を負う可能性があります。**

データトレースの方法

制御上のデータのトレースは、2 つの異なる方法で実行されます。

- トレースオブジェクトによって生成された IEC コードおよび、トレースの子アプリケーションによってコントローラーにダウンロードされるコード。
- CmpTraceMgr コンポーネント内 (**トレースマネージャー**とも呼ばれる)。

注記：データのトレースは、IEC タスクのサイクルタイムを大幅に増やす可能性があります。

取得されるデータは、対象の設定項目によって決まります。(**トレース → トレースマネージャー**)。

トレースマネージャーの高度な機能により、以下を実行できます。

- プロセッサーまたはバッテリーの温度曲線など、制御システムのパラメーターの設定およびトレースをします。詳細については、変数設定 (*424* ページ) および記録 (トリガー) 設定 (*426* ページ) を参照してください。
- ドライブの電流のトレースなどのデバイストレースを読み出します。詳細については、**トレースアップロード**コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の説明を参照してください。
- 他のシステムコンポーネントのシステム変数をトレースします。

さらに、追加コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **オンラインリスト** も利用できます。

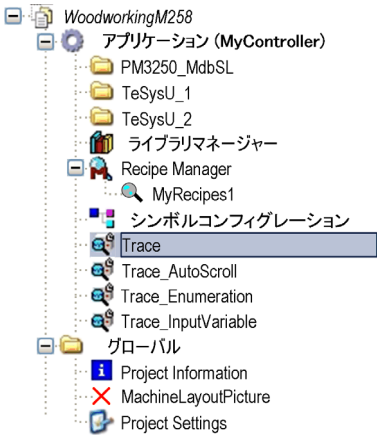
注記：

- ビジュアライゼーションでトレースを使用する場合、デバイスパラメーターをトレースをしたり、トリガーに使用したりすることはできません。
- トリガーレベルはリテラルと定数のみサポートされており、IEC 表記にセットすることはできません。
- レコード条件は変数のみサポートされており、BOOL タイプの IEC 表記にセットすることはできません。
- プロパティをトレースしたり、トリガーに使用したりする場合、IEC 宣言中の attribute monitoring (*536* ページ) で注釈を付ける必要があります。

設定

設定でトレースデータとトレースデータの表示設定の設定します。設定ダイアログボックスにアクセスするためのコマンドが表示されます。マルチチャンネルモードなどの異なるビューで、複数の変数を同時にトレースして表示することができます。独自のトレースオブジェクトの異なるトリガー設定で変数のトレースを記録します。トレースオブジェクトはいくつでも作成できます。

複数のトレースオブジェクトを含む**ツールツリー**



表示の設定を変更するコマンドについては、*機能*の項 (*419* ページ) に記述されています。トレースを実行するためのコマンドと同様にズーム機能とカーソルが使用できるので、グラフを圧縮または伸張することができます。

ビジュアライゼーション内のトレースの読み出しを統合するには、ビジュアライゼーション要素**トレース**を使用します。

データ記録のトレース設定についての詳細は、*レコード設定*を参照してください (*426* ページ)。

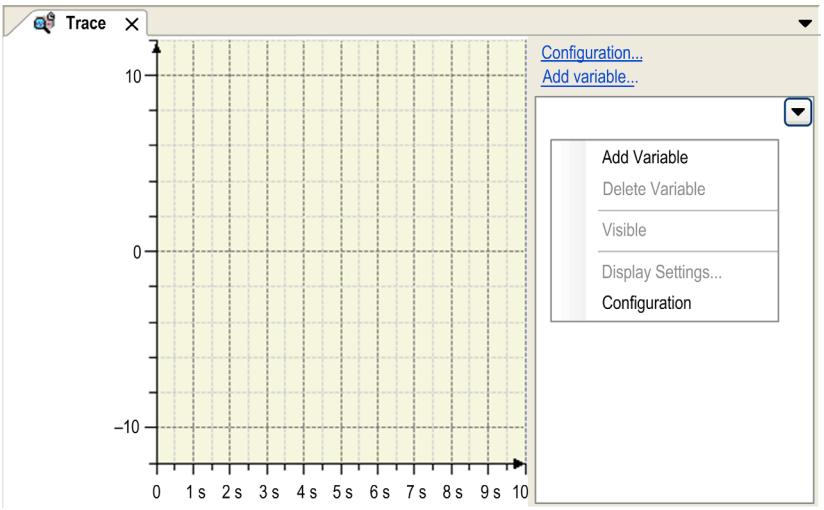
トレースオブジェクトの作成

概要

ツールツリーにトレースオブジェクトを挿入するには、**アプリケーション**ノードを選択し、緑色のプラスボタンをクリックして、**トレース ...** コマンドを実行します。**ツールツリーのトレース**ノードをダブルクリックしてトレースエディターを開きます。

設定

新しく作成されたコンテキストメニュー付きのトレース



トレースには、少なくとも 1 つのサンプリングされた変数が含まれます。

ウィンドウの右側のトレースツリー領域には設定されたトレース変数が表示されます。デフォルトではトレース変数はインスタンスのフルパスと共に表示されます。

インスタンスパスを非表示にするには、**インスタンスパスを非表示にする**チェックボックスを選択します。このチェックボックスを表示するには、トレースツリー領域の右上隅にある矢印ボタンをクリックします。

トレース設定を設定または変更するには、トレースツリー領域のコンテキストメニューのコマンドを使用します。

- **変数を追加 ...**: **変数設定** (*424* ページ) 付きの**トレース設定**ダイアログボックスが開きます。
- **削除** : 選択された変数を削除します。少なくとも 1 つのトレース変数が存在する場合にのみ使用できます。
- **表示** : このコマンドで選択した変数を表示します。少なくとも 1 つのトレース変数が存在する場合にのみ使用できます。
- **設定 ...**: **レコード設定** (*426* ページ) 付きの**トレース設定**ダイアログボックスを開きます。
- **表示設定 ...**: **表示設定** ダイアログボックスを開きます (*431* ページ)。グラフの外観と座標システムを設定することができます。このコマンドは、設定が読み込まれるまで使用できません。

機能

トレースを実行するには、次のコマンドを使用します。

- **変数を追加** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **トレースをダウンロード** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **トレースを開始 / 停止** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **トリガーをリセット** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*

グラフの表示をカスタマイズするには、次のコマンドを使用します。

- **カーソル** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **マウスでのズーム** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **ビューをリセット** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **Auto Fit** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **圧縮** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **拡張** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **Convert to Multi Channel** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*

- **Convert to Single Channel** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- 詳細については、*トレースダイアグラムのキーボード操作*の章 (*435* ページ) を参照してください。

ランタイムシステムに格納されたトレースにアクセスするには、次のコマンドを使用します。
- **オンラインリスト ...** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **トレースをアップロード** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*

ディスクに保存されているトレースにアクセスするには、次のコマンドを使用します。
- **トレースを保存 ...** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **トレースをロード ...** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*
- **シンボリックトレースコンフィグをエクスポート** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*

はじめに

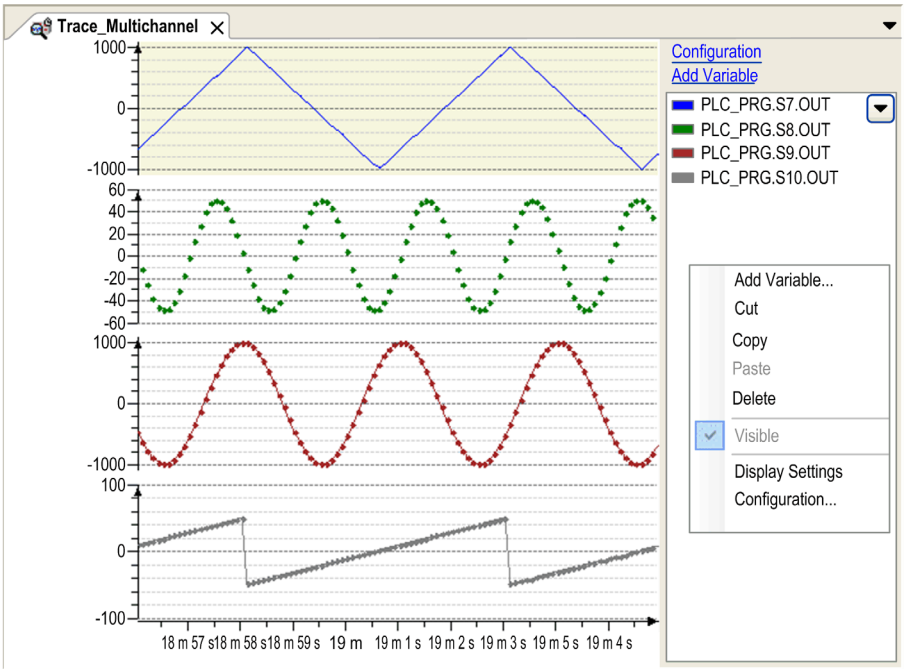
オンラインモードでトレースを開始するには、**トレースダウンロード**コマンドを実行してトレース設定をコントローラーにダウンロードします。トレース変数のグラフがトレースエディターウィンドウに表示され、トレースファイルを外部ファイルに保存できます。このファイルはエディターに再読み込みできます。*オンラインモードのトレースエディター* (*434* ページ) の章を参照してください。

手順	手順内容
1	関連するアプリケーションにログインして実行します。 **結果 :** アプリケーションがコントローラーで実行されます。
2	トレースをダウンロードします。 **結果 :** トレースグラフが、トレース設定に従って表示されます。
3	トレースグラフを配置してトレースデータを格納し、トレースを停止 / 開始します。

例

図のトレースエディターは、オンラインモードでのトレースの例を示しています。4 つの変数が、ダイアログの右側の変数ツリーに表示されるように選択されています。

オンラインモードでのトレース



25.2
トレース設定

このセクションについて

このセクションには次の項目が含まれています。

トレース設定 - ツリービュー

概要

トレース設定ダイアログボックスには、左側に 2 つの異なるツリービューがあります。
- **Trace Record** ツリービュー
- **Presentation (diagrams)** ツリービュー

Trace Record ツリービュー

Trace Record ツリービューには、トレースされた変数が表示され、変数設定にアクセスできます。

トレース設定ダイアログボックスの右側に表示されるビューはツリービューで選択された要素によって異なります。

条件	結果
トレース名が選択されている場合	**レコード設定**ビューが右側に表示されます。
トレース変数が選択されている場合	**変数設定**ビューが右側に表示されます。

Trace Record ツリービューのコンテキストメニューのコマンド

Trace Record ツリービューのコンテキストメニューには次のコマンドがあります。

コマンド	詳細
変数を追加	**変数設定** ビュー (*424* ページ) が右側に表示されます。 新規変数を追加するには、参照ボタン (**...**) をクリックし、トレース用の変数を選択します。
Assign to diagram	変数を右クリックし、**Assign to diagram** コマンドを実行して選択した変数を含まない図のリストを表示します。この図で選択された変数を表示する図を選択します。 選択した変数が既にすべての図に割り当てられている場合は、このコマンドは無効になります。
Enabled	変数はデフォルトで Enabled です。 無効にされた変数は無効として表示されます。それらは表示も記録もされません。

Presentation (diagrams) ツリービュー

Presentation (diagrams) ツリービューには、トレースエディターで表示された図が表示され、表示モードにアクセスできます。

トレース設定ダイアログボックスの右側に表示されるビューはツリービューで選択された要素によって異なります。

条件	結果
Time axis ノードが選択されている場合	**表示モード** ビュー (*428* ページ) が右側に表示されます。時間軸表示の指定ができます。
Diagram 名が選択されている場合	図の座標システムとプレビューの設定が表示されます。
Y 軸ノードが選択されている場合	**表示モード** ビュー (*428* ページ) が右側に表示されます。軸表示の指定ができます。
Shown variables ノードの下でトレース変数が選択されている場合	トレース変数の設定ができる**変数設定** ビュー (*424* ページ) が右側に表示されます。

Presentation (diagrams) ツリービューのコンテキストメニューのコマンド

Presentation (diagrams) ツリービューのコンテキストメニューには次のコマンドがあります。

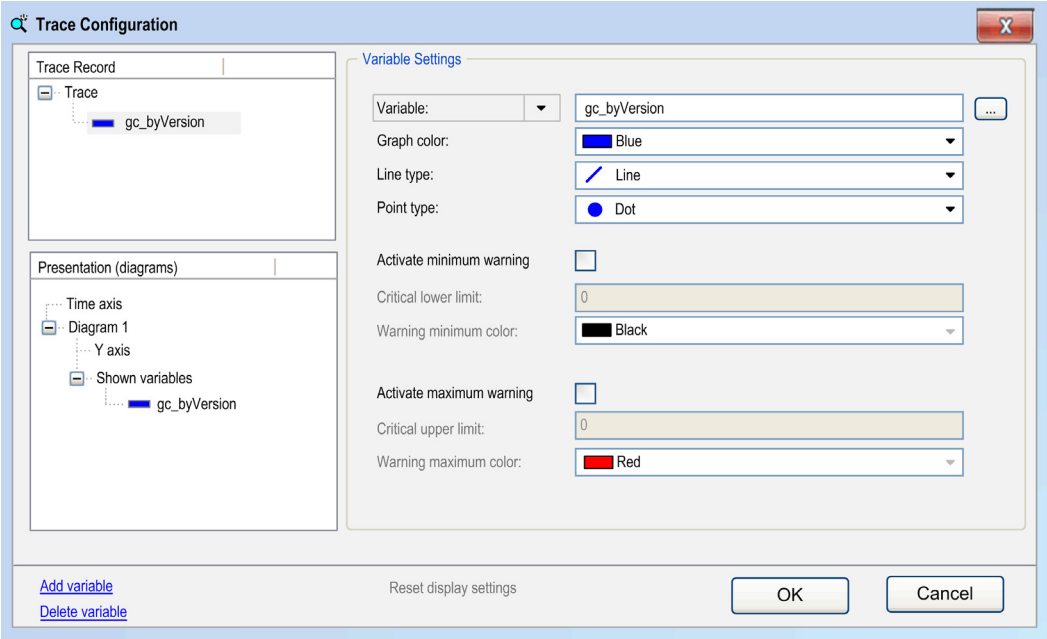
コマンド	詳細
Add diagram	新しい図を追加し、**Presentation (diagrams)** ツリービューに表示します。
新規変数の追加	新しいトレース変数を追加し、**変数設定** ビュー (*424* ページ) を右側に表示します。 値のグラフをトレースするために、入力フィールドで変数を選択します。表示を指定します。選択された図に変数が割り当てられます。
Add existing variable	図を右クリックし、**Add existing variable** コマンドを実行し選択した図が表示されていないところにトレース変数を表示します。選択した図に表示する変数を選択します。 選択された図にすべてのトレース変数がすでに表示されている場合は、このコマンドは利用できません。

変数の設定

概要

トレースツリーでトレース変数を選択すると、**変数設定**で**トレース設定**ダイアログボックスが開きます。トレースする変数とその表示方法を設定することができます。

変数の設定の**トレース設定**ダイアログボックス



トレース変数は、ウィンドウの左側にツリー構造で表示されます。先頭ノードにはトレース名が付いています。

トレース変数の追加と削除

トレースツリーに変数を追加または削除するには、トレースツリーの下にあるコマンドを使用します。

コマンド	詳細
変数を追加	トレースツリーに無名の変数を作成します。 ダイアログボックスの右側で新しい変数の設定が可能です。
Delete variable	選択した変数を関連した設定と共に削除します。

変数の設定と変更

変数の設定を選択するには、トレースツリーで目的の変数を選択します。現在の設定がトレース設定ウィンドウの右側に表示されます。後で変数設定を変更するには、トレースツリーで変数を選択し、**変数設定**ダイアログボックスを再度使用します。

パラメーター		詳細
変数		トレースする信号を指定する信号の名前 (パス) を入力します。有効なトリガー信号は、IEC 変数、プロパティ、リファレンス、ポインターの内容、またはアプリケーションの配列要素です。許可される型は、STRING、WSTRING、または ARRAY を除くすべての IEC 基本型です。基本型が STRING、WSTRING、または ARRAY ではない列挙型も使用できます。... ボタンをクリックすると、有効な入力を取得できる入力アシスタントが開きます。パラメータートレースをサポートするコントローラーでは、**変数**パラメーターをクリックするとリストが表示されます。デバイスパラメーターをトレースする場合は、このリストから**パラメーター**を選択します。その後、入力アシスタントでデバイスパラメーターを探します。変数設定を編集または確認します。デバイスパラメーターは、CmpTraceMgr コンポーネントが使用されている場合にのみサポートされます。デバイスパラメーターがトレース (またはトリガー) 変数に使用されている場合、**ビジュアライゼーションのためにトレース POU を生成**するオプションを有効にはできません。 **注記：** トレースに CmpTraceMgr が使用されている場合、トレース (またはトリガー) 変数として使用されるプロパティ (*145* ページ) には、コンパイラー属性の 属性監視 (*536* ページ) が必要です。
グラフの色		表示させる変数のトレース曲線の色を色選択リストから選択します。
線の種類		グラフのサンプルの繋ぎ方を指定します。大量のデータには**線**を使用します。これはデフォルト値です。
	ライン	サンプルは線で繋がれます (デフォルト)。
	ステップ	サンプルは階段状に繋がれます。従って、次のサンプルのタイムスタンプに対する水平ラインの後に、次のサンプルの値に対する垂直ラインが続きます。
	なし	サンプルは繋がりません。
点の種類		グラフに値を表示する方法を指定します。
	ドット	サンプルは点で表示されます (デフォルト)。
	クロス	サンプルは十字で表示されます。
	なし	サンプルは表示されません。
最小警告をアクティブ化		このオプションを有効にすると、トレースグラフが**クリティカル下限値**で定義された値より小さくなると**最小警告色**で定義された色で表示されます。
クリティカル下限値		ここに入力された変数の値を下回り**最小警告をアクティブ化**が有効な場合、曲線の値は次に指定された色に変わります。
最小警告色		下限が有効である場合の色の値。
最大警告をアクティブ化		このオプションを有効にすると、トレースグラフが**クリティカル上限値**で定義された値より大きくなるとすぐに**最大警告色**で定義された色で表示されます。
クリティカル上限値		ここに入力された変数の値を上回り**最大警告をアクティブ化**が有効な場合、曲線の値は次に指定された色に変わります。
最大警告色		上限が有効である場合の色の値。

変数の複数選択

キーボードのショートカットキー **Shift** + マウスクリックまたは **Ctrl** + マウスクリックを使用すると、編集する変数を複数選択できます。**変数の設定**ダイアログボックスでの変更は、選択したすべての変数に適用されます。**Shift** + **上矢印** / **下矢印**、または **Ctrl** + **上矢印** / **下矢印**で同じことができます。

レコード設定

概要

設定コマンドを実行するか、トレースツリー内のトレース変数をダブルクリックすると、**レコード設定**のある**トレース設定**ダイアログボックスが開きます。設定コマンドは、メイントレースエディターウィンドウの右側にあるトレースツリーのコンテキストメニューでも使用できます。

注記：レコード設定の**トレース設定**ダイアログボックスで完了した設定は、トレースグラフのすべての変数に対して有効です。

トリガーの基本

ほとんどの場合において、入力信号のトレースと表示は前回の測定の直後やユーザーがスタートボタンを押した時などの不規則な瞬間から開始することは推奨されていません。設定されたレコード数 (Post Trigger) に対するトリガーが起動した時にトレースが完了することが推奨されます。これはトリガーと呼ばれ、ここで定義する必要があります。

入力信号をトリガーするために以下の方法を使用します。

- トリガー変数の設定
- レコード条件の設定
- 両方

レコード (トリガー) の設定と変更

パラメーター		詳細
Enable Trigger		トリガーシステムを有効にするには、チェックボックスを選択します。以降の設定とは関係なくオン / オフを切り替えることができます。トリガーシステムが無効の場合、トレースは常時実行されます。
トリガー変数		変数を割り当てます。信号の名前 (およびパス) を入力して、トリガーとして使用する信号を指定します。 有効なトリガー信号は、IEC 変数、プロパティ、参照、ポインター、アプリケーションの配列要素、または式です。許可される型は、STRING 、WSTRING、または ARRAY を除くすべての IEC 基本型です。基本型が STRING、WSTRING、または ARRAY ではない列挙型も使用できます。ポインターのコンテンツは、無効な信号です。... ボタンをクリックすると、有効な入力を取得できる入力アシスタントが開きます。 **トリガー変数 :** プロパティをクリックすると、デバイスパラメーターをトリガーとして使用できるコントローラーのリストが表示されます。デバイスパラメーターをトリガーとして使用する場合は、このリストから**トリガーパラメーター**を選択します。... ボタンで**入力アシスタント**を開き、トレース可能なパラメーターを選択します。**要素**には、システムで使用可能なパラメーターがリスト表示されます。また、パラメーター名を直接入力することも、テキストフィールドに (別の設定から) コピー＆ペーストすることもできます。デバイスパラメーターは、トレースマネージャーが使用されている場合にのみサポートされます。 **注記 :** トレースに CmpTraceMgr が使用されている場合、トレース (またはトリガー) 変数として使用されるプロパティ (*145* ページ) には、コンパイラー属性の 属性監視 (*536* ページ) が必要です。
トリガーエッジ		–
	正	BOOL 型トリガー変数の立上がりでイベントをトリガーします。または、値の上昇がアナログトリガー変数の**トリガーレベル**で定義された値に達した瞬間にイベントをトリガーします。
	負	BOOL 型トリガー変数の立下がりでイベントをトリガーします。または、値の下降がアナログトリガー変数の**トリガーレベル**で定義された値に達した瞬間にイベントをトリガーします。
	両方	上記、**正**と**負**で説明されている条件でイベントをトリガーします。
Post Trigger		トリガー発生後に記録される、トレース信号あたりの記録数を入力します。 初期値 : 50 範囲 : 0...4,294,967,295

パラメーター	詳細
トリガーレベル	トリガーを発生させるポイントの値を入力します。 **トリガーエッジ**で、トリガー変数の立上がりか、立下りのどちらでトリガーを発生させるかを指定します。アナログ変数 (LREAL または INT のような数値タイプの変数) がトリガー変数として使用された時にのみ、設定する必要があります。 直接値を入力します。トリガー変数に変換可能な型であれば、GVL 定数、または ENUM 値を使用できます。 IEC コードを使用する場合は、トリガー変数に変換可能な任意の IEC 表記も入力可能です。 デフォルト値 : – (空白)
タスク	有効なタスクのリストから、入力信号の取得を行うタスクを選択します。
レコード条件	条件によって記録を開始させる時は、ここに変数を入力します。すでにトレースが開始されている場合、例えば、スタートボタンが押され、ここに割り当てられた変数が TRUE の時、データの取得が開始され、トレースグラフが表示されます。 CmpTraceMgr を使用する場合のレコード条件は、BOOL 型の変数、またはビットアクセスの変数にします。ポインターのコンテンツは無効です。プロパティもサポートされています。 IEC コードを使用する場合は、BOOL 型の任意の IEC 表記も入力可能です。
コメント	現在の記録に関連したコメントを入力します。
精度	トレースタイムスタンプの分解能を ms または us で入力します。それぞれの取得した信号で値とタイムスタンプのペアが、プログラミングシステムに送信され格納されます。送信されたタイムスタンプは相対値で、トレースの開始を参照します。 トレースタスクのサイクルタイムが 1 ms 以下の場合、タイムスタンプは µs の分解能を使用します。このオプションは、コントローラーでトレースマネージャーが有効な時のみ可能です。
Automatic Restart	トレース設定、および表示器の機器上で RTS トレースバッファーの最後のコンテンツが永続的に格納される場合、このオプションをセットします。 このオプションは、コントローラーでトレースマネージャーがトレースをしている時のみ可能です。
高度な設定 ...	このボタンをクリックすると、**高度なトレース設定** ダイアログボックス (*430* ページ) が開きます。トレーストリガーに追加設定をセットすることができます。

注記：タイムベースの異なるトレース信号を取得して表示する場合は、別のトレースオブジェクトで記録設定を行う必要があります。

表示モード

概要

Presentation (diagrams) ツリービューで軸を選択すると、**表示モード**ビューが開きます。

次のパラメーターがあります。

パラメーター	詳細
表示モード	次のスケーリングオプションを選択します。 ● **Auto** 時間軸は自動的に調整されます。 ● **固定長** このオプションを選択して、時間軸セグメントに対して一定の長さを定義します。 ● **固定** このオプションを選択し、**最小**パラメーターと**最大**パラメーターを使用して時間軸セグメントの範囲を定義します。
最小	**表示モード → 固定** が選択された場合、時間軸セグメントには開始値を入力します。
最大	**表示モード → 固定** が選択された場合、時間軸セグメントには終了値を入力します。
長さ	セグメントの固定長を定義するための値を入力します。開始値はそれに応じて調整されます。
グリッド	**グリッド**オプションを選択して X 方向にグリッド線がある図を作成します。 **グリッド**パラメーターの横にあるリストからグリッド線の色を選択します。
説明	**説明**オプションを選択し、ボックスに説明するテキストを入力します。
目盛セクション	
固定スペース	**固定スペース**オプションを選択して目盛りを表示します。
距離	目盛りの間隔を定義する値を入力します。
細分化	2 つの目盛りの間の補助目盛り数を定義する値を入力します。

次の要素があります。

要素	詳細
フォント	**フォント**ボタンをクリックし、**フォント**ダイアログボックスを開き時間軸のフォントを設定します。
プレビューリンク	**プレビュー**リンクをクリックして**表示モード**ビューで定義した図のプレビューを表示します。
変数の追加リンク	**Trace Record** ツリービューでトレースが選択されている場合は、**変数の追加**リンクをクリックして新規トレース変数を **Trace Record** ツリービューに追加します。
変数の削除リンク	**Trace Record** ツリービューでトレースが選択されている場合は、**変数の削除**リンクをクリックして選択した変数を **Trace Record** ツリービューから削除します。

プレビュー

表示モードビューで**プレビュー**リンクをクリックすると、**表示モード**ビューで定義した図のプレビューが表示されます。

図のプレビューの下に次の要素があります。

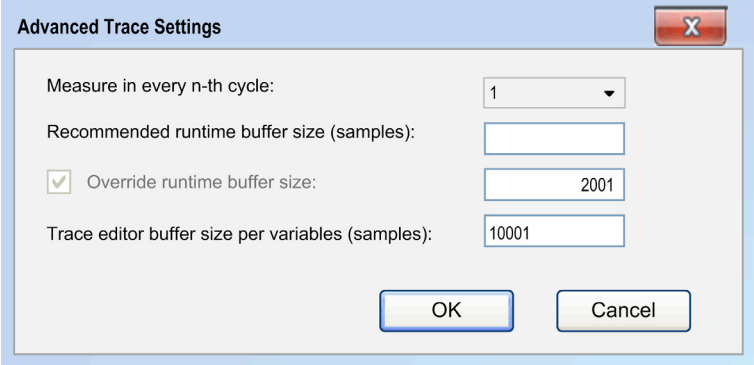
要素	詳細
Backcolor パラメーター	**Backcolor** パラメーターの横にあるリストから図の背景色を選択します。
Backcolor on Selection パラメーター	**Backcolor on Selection** パラメーターの横にあるリストから選択された図の背景色を選択します。
Add diagram リンク	図が選択されている場合、**Add diagram** リンクをクリックして**表示モード**ビューに新規の図を追加します。
Delete diagram リンク	図が選択されている場合、**Delete diagram** リンクをクリックして**表示モード**ビューから図を削除します。
Reset display settings リンク	**Reset display settings** リンクをクリックし、選択された図の表示設定、または選択された Y 軸をデフォルト値にリセットします。
OK ボタン	**OK** ボタンをクリックし、設定を確認しトレース設定を保存します。

高度なトレース設定

概要

レコード設定で**トレース設定** ダイアログボックス (*426* ページ) の中の **高度な設定 ...** ボタンをクリックすると、**高度なトレース設定**ダイアログボックスが開きます。トレースエディターが読み取るランタイムまたは開発システムのバッファーサイズなど、データ記録の設定が含まれます。アプリケーションでデータが記録する間隔は、タスク構成から算出されます。

高度なトレース設定ダイアログボックス



サンプル数を定義してバッファーサイズを設定します。EcoStruxure Machine Expert は、タスク構成の設定を使用して、サンプル数に従って時間の間隔を計算します。タスク周期時間が決定可能な場合にのみ計算ができます。結果は、標準のスタイルでテーブルの外側の右側に表示されます。例えば、1h1m1s1ms。

パラメーターの詳細

パラメーター	詳細	コメント
測定頻度 (サイクル数)	データレコードが取得される頻度を定義します (各サイクル、2 サイクルごとなど)。これがスキャンレートです。デフォルト : 1。これは、各周期ごとに 1 つのデータセットが記録されることを意味します。	例えば、データ記録のスキャン間隔は 10 ms です。
Recommended runtime buffer size (samples)	EcoStruxure Machine Expert が計算し、ランタイムシステムがトレース変数ごとに格納できるサンプルの最大数。EcoStruxure Machine Expert は、**Refresh interval** の値と、**測定頻度 (サイクル数)** の値からタスク周期時間の数を計算します。**Recommended runtime buffer size (samples)** は **ランタイムバッファーサイズのオーバーライド** オプションが無効な場合にのみ使用されます。	ランタイムシステムがデータを収集する時間間隔の最大長。例えば 2 秒 (**Refresh interval = 500 ms** で、**測定頻度 (サイクル数) = 1** の場合)。
ランタイムバッファーサイズのオーバーライド	このオプションが選択されていると、アプリケーションは **Recommended runtime buffer size (samples)** に設定された値を使用しません。アプリケーションが記録するトレース変数ごとのサンプル数で、これはランタイムのバッファーサイズです。範囲 : 10 以上、ただしトレースエディターバッファーに設定された値以下。初期値 : 100	ランタイムシステムがデータを収集する時間間隔の最大長。例えば 6 秒。
Trace editor buffer size per variables (samples)	内部使用のために EcoStruxure Machine Expert によって提供されるトレースバッファーを制限します。	–

表示設定

概要

表示設定ダイアログボックスには、トレースダイアグラムの表示設定 (x 軸と y 軸の両方) が含まれており、トレースダイアグラムにプレビューが表示されます。

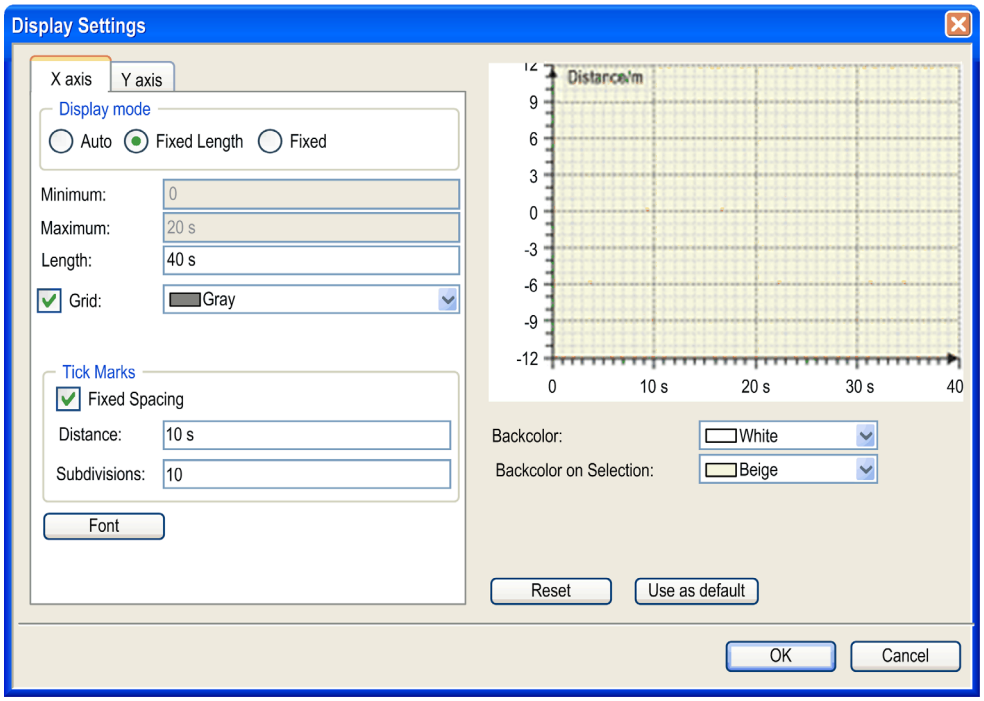
トレース設定ダイアログボックスの**表示設定**ボタンをクリックして、**表示設定**ダイアログボックスを開きます。

設定は、次のボタンで管理できます。

ボタン	詳細
リセット	このコマンドで、EcoStruxure Machine Expert の設定がデフォルト値にリセットされます。
デフォルトとして使用	このコマンドで、EcoStruxure Machine Expert の設定がデフォルト値として保存されます。

X 軸タブ

表示設定ダイアログボックスの **X 軸**タブ

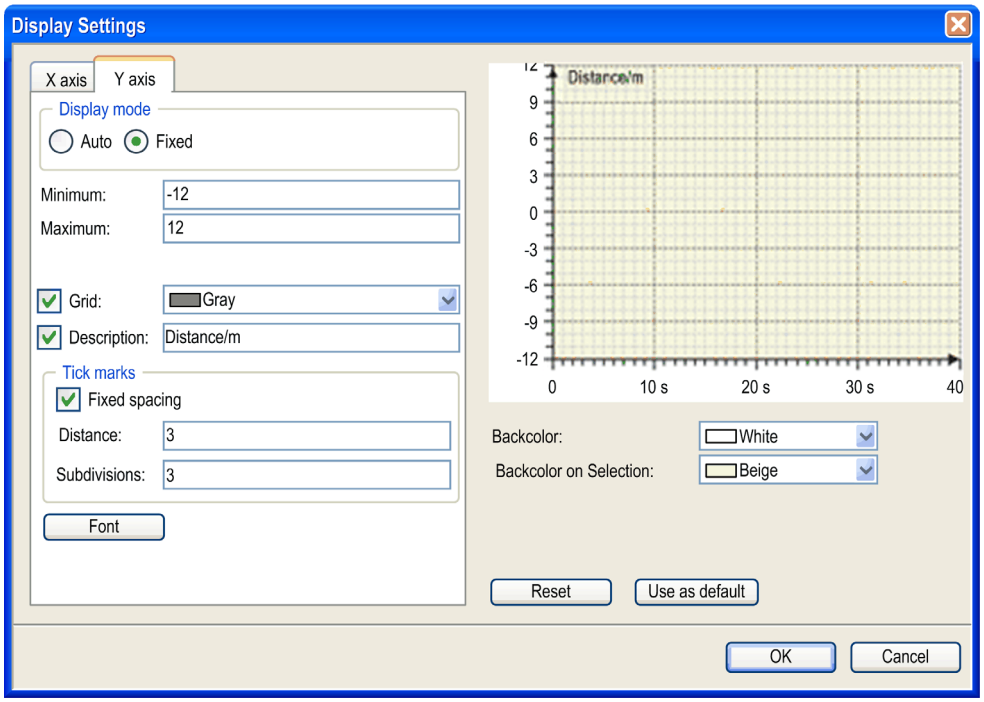


パラメーター		詳細
表示モード		スケーリング
	Auto	このオプションを有効にすると、EcoStruxure Machine Expert によって自動的に拡大縮小されます。
	固定長	このオプションを有効にすると、EcoStruxure Machine Expert によって固定長でセグメントが表示されます。
	固定	このオプションを有効にすると、EcoStruxure Machine Expert によって**最小**から**最大**の間でセグメントが表示されます。
最小		セグメントの初期値 必要条件 : **表示モード**が**固定**に設定されている。
最大		セグメントの終値 必要条件 : **表示モード**が**固定**に設定されている。
長さ		セグメントの固定長。EcoStruxure Machine Expert は初期値を採用します。
グリッド		このオプションを有効にすると、縦方向にグリッド線がある図が表示されます。リストからグリッド線の色を選択します。
目盛		–

パラメーター		詳細
	固定スペース	このオプションを有効にすると、EcoStruxure Machine Expert によって、**距離**と**細分化**を使用した目盛りが描画されます。
	距離	目盛りの間隔
	細分化	目盛り間の分割数
フォント		X 軸に使用されるフォント

Y 軸タブ

表示設定ダイアログボックスの **Y 軸**タブ



パラメーター		詳細
表示モード		スケーリング
	Auto	このオプションを有効にすると、EcoStruxure Machine Expert によって自動的に拡大縮小されます。
	固定	このオプションを有効にすると、EcoStruxure Machine Expert によって**最小**から**最大**の間でセグメントが表示されます。
最小		表示されたセグメントの初期値 必要条件 : **表示モード**が**固定**に設定されている。
最大		表示されたセグメントの終値 必要条件 : **表示モード**が**固定**に設定されている。
グリッド		このオプションを有効にすると、グリッド線がある図が表示されます。リストからグリッド線の色を選択します。
説明		このオプションを有効にすると、軸に**説明**というラベルが付きます。
目盛		–
	固定スペース	このオプションを有効にすると、EcoStruxure Machine Expert によって、**距離**と**細分化**を使用した目盛りが描画されます。
	距離	目盛りの間隔
	細分化	目盛り間の分割数
フォント		Y 軸に使用されるフォント

座標システムのプロパティ

パラメーター	詳細
Backcolor	リストから座標システムの背景色を選択します。
Backcolor on Selection	リストから座標システムの背景色を選択します。ダイアグラムがトレースウィンドウで選択されている際に使用されます。

25.3
オンラインモードのトレースエディター

オンラインモードのトレースエディター

概要

デバイス上でトレースが実行されている場合、トレースダイアログボックスの**オンラインリスト** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* に実行中であることが表示されます。

トレースのダウンロード

オンラインモードでトレースを開始するには、アプリケーションのログイン中に**トレース → トレースダウンロード** メニューコマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* で明示的にトレースをコントローラーにダウンロードします。トレースシグナルのグラフがトレースエディターウィンドウに表示されます。

アプリケーションを変更せずにログインとログアウトを行っている間は、ダウンロードを新たに行うことなくトレースが実行されます。

アプリケーションコードが変更された場合のトレースの動作はログインモードにより異なります。

- **オンライン変更してログイン** または **変更せずにログイン**：トレースはまだ実行中です。
- **ダウンロードしてログイン**：コントローラー内のトレースは削除され、新たにトレースのダウンロードが必要です。

トレースグラフ設定のオンライン変更

オンラインモードでは**レコード設定**の**トレース設定**ダイアログボックスと、**変数設定**の**トレース設定**ダイアログボックスが利用でき、トレースの実行中に多くのトレース設定を変更できます。これができない場合は、トレースシグナルの名前が変更された時です。例えば、トレースが停止し、新しいダウンロードが必要な場合です。

トレースグラフのオンラインナビゲーション

取得したトレース変数値の表示範囲は、トレース設定によって異なるだけではありません。ツールバーまたはショートカットキーを使用して**トレース**メニューのスクロールやズーム機能で再編成が可能です。トレースダイアグラムのナビゲーションの方法については、*ショートカットキー* の章 (*435* ページ) を参照してください。

25.4
トレースダイアグラムのキーボード操作

ショートカットキー

概要

次の表は、キーボードとマウスのアクションです。

アクション	キーボード操作	マウス操作
トレースグラフを時間軸に沿って水平方向にスクロールする。	トレースカーソルがない場合 : ● 左 / 右矢印 ● 大きい間隔の場合: CTRL + 左 / 右矢印 トレースカーソルが 1、または 2 の場合 : ● ALT + 左 / 右矢印 ● 間隔が広い場合 : CTRL + ALT + 左 / 右矢印	ドラッグ & ドロップでグラフをスクロールします。これは、マウスカーソルの別のビューによって示されます。
トレースグラフを Y 軸に沿って垂直にスクロールする。	上 / 下矢印 大きい間隔の場合 : CTRL + 上 / 下矢印	CTRL + ドラッグ & ドロップを使用します。
マウスで選択した四角形 (ウィンドウ) にズームする。	–	**マウス拡大** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドを使用します。
黒いトレースカーソルを移動する。	左 / 右矢印 大きい間隔の場合 : CTRL + 左 / 右矢印	トレースカーソルの黒い三角形をクリックし、ドロップするまで X 軸に沿ってドラッグします。
グレーのトレースカーソルを移動する。	SHIFT + 左 / 右矢印 大きい間隔の場合 : CTRL + SHIFT + 左 / 右矢印	トレースカーソルのグレーの三角形をクリックし、ドロップするまで X 軸に沿ってドラッグします。
時間軸を縮める。 マルチチャンネルモードでは、すべてのダイアグラムの時間軸が圧縮されます。	–	マウスホイールを使用します。 または、**圧縮** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドを使用します。
時間軸を伸ばす。 マルチチャンネルモードでは、すべてのダイアグラムの時間軸が引き伸ばされます。	+	マウスホイールを使用します。 または、**ストレッチ** *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* コマンドを使用します。
Y 軸を圧縮する。 マルチチャンネルモードでは、選択したダイアグラムの Y 軸が圧縮されます。	CTRL + –	CTRL + マウスホイール
Y 軸を伸ばす。 マルチチャンネルモードでは、選択されたダイアグラムの Y 軸が引き伸ばされます。	CTRL + +	CTRL + マウスホイール
マルチチャンネルモードで、下にある次のダイアグラムを選択する。	タブ	選択されていない図をクリックして選択します。

第 26 章
トレンドの記録

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
26.1	トレンド記録オブジェクト	438
26.2	トレンド記録設定	441
26.3	トレンドデータの記録	449

26.1
トレンド記録オブジェクト

このセクションについて

このセクションには次の項目が含まれています。

トレンド記録の概要

トレンド記録機能

トレンド記録によって、長期間にわたるデータの取得ができます。トレンド記録を使用して IEC 変数の値を取得し、データベースに永続的に保存できます。履歴データが保持されるため、トレンドからデータを読み取ることができます。データベースはコントローラー上にあり、ランタイムでデータが入力されます。トレンドデータは、データベースを読み取りレコードを視覚化するビジュアライゼーションによって表示されます。

記録されるデータ量とコントローラーの性能によっては、トレンド記録操作中にタイムアウトになることがあります。これは例外エラーとして検出されます。

コントローラーを停止したりダウンロードを実行する (**オンライン変更** コマンドを実行 *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*) 前に、トレンド記録処理を停止してトレンドデータをデータベースに保存します。

注記
データの損失 コントローラーの状態を実行状態から変更する前に、トレンドファンクションを停止しトレンドデータを保存します。 **上記の指示に従わないと、物的損害を負う可能性があります。**

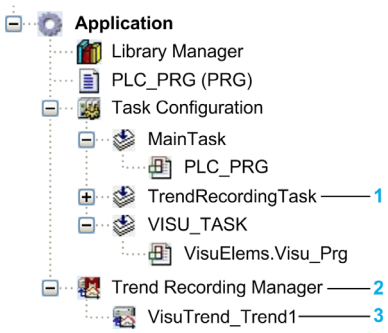
この機能はすべてのコントローラーで対応している訳ではありません。ご使用のコントローラーがトレンド記録に対応しているかは、各コントローラーについては、*プログラミングガイド*を参照してください。

トレンド記録オブジェクト

概要

ツールツリーにトレンド記録オブジェクトを挿入するには、**アプリケーション**ノードを選択し、緑色のプラスボタンをクリックして、**Add Other Objects → トレンド記録 ...** コマンドを実行します。**ツールツリーのトレンド記録**タイプのオブジェクトをダブルクリックして**トレンド記録エディター**を開きます。

トレンド記録オブジェクトを含む**ツールツリー**



1 **タスク**タイプの **TrendRecordingTask**
2 **トレンド記録マネージャー**
3 **トレンド記録**タイプのオブジェクト < トレンド記録の名前 >

トレンド記録タスクオブジェクト

トレンド記録タスクオブジェクトは、**トレンド記録マネージャー**が実行されるタスクによってアプリケーションを拡張します。**ツールツリー**では、**アプリケーション**ノードごとに 1 つの**トレンド記録マネージャー**のサブノードのみ使用できます。タスクは次のプログラムを呼び出します。VisuTrendStorageAccess.GlobalInstances.g_TrendRecordingManager.CyclicCall.

ビジュアライゼーションにトレンドを追加すると、**タスク設定**ノードは**トレンド記録タスク**オブジェクトによって拡張されます。

トレンド記録マネージャーオブジェクト

トレンド記録マネージャーオブジェクトは、データベースを利用した長期的なデータ保存機能によってアプリケーションを拡張します。**ツールツリー**では、**アプリケーション**ノードごとに 1 つの**トレンド記録マネージャー**のサブノードのみ使用できます。

トレンド記録タイプのオブジェクト

トレンド記録タイプのオブジェクトは、**Place data in a database** 機能によってアプリケーションを拡張します。トレース機能でデータを取得します。**アプリケーション**ノードの下には、トレンド記録オブジェクトをいくつでも置くことができます。**トレンド記録エディター**でトレンド記録を設定します。

26.2
トレンド記録設定

このセクションについて

このセクションには次の項目が含まれています。

トレンド記録設定

概要

ツールツリーのトレンド記録オブジェクトをダブルクリックするか、**視覚化 → Edit trend recording** コマンドを実行して**トレンド記録エディター**を開きます。

エディターにはツリービュー構成が含まれており、ツリービューでの選択に応じて**レコード設定**、または**変数設定**が表示されます。

ツリービュー構成

トレンドのツリービューには、設定されているトレンド変数が表示され、設定への移動ができます。一番上のノードにはトレンド記録の名前が表示されます。この項目をクリックすると、関連付けられている**レコード設定**が開きます。一番上のノードの下には、値が取得するトレンド変数ごとにノードがあります。項目をクリックすると、関連付けられている**変数設定**が開きます。

記録の設定

概略

トレンド記録エディター のツリービューで一番上のノードを選択すると、**記録の設定**を含む**トレンド記録** ダイアログボックスが開きます。

トレース (*426* ページ) にも使われるランタイムコンポーネント機能を使用し、データの記録が行われます。ここに表示される設定は同じものです。トリガー設定に関連するパラメーターは、トレンドの記録中にな必要とされないため、グレー表示されます。

記録の設定と変更

Trend Recording Configuration ダイアログボックスの **Record Settings** には、次の設定要素があります。

要素	詳細
タスクパラメーター	リストからデータ取得を実行するタスクを選択します。通常、トレンド記録はメインプログラムと同じタスクで実行されます (例 : PLC_PRG)。
Record condition パラメーター	BOOL 型の変数、ビットまたはプロパティを参照します。値が TRUE の場合、データ取得が実行されます。この設定はオプションです。
コメントパラメーター	コメントテキストを入力します。
精度パラメーター	タイムスタンプが取得される分解能。 トレンドタスクの周期時間が 1 ミリ秒以下の場合は、タイムスタンプの分解能を 1 マイクロ秒に設定します。
トレンドストレージ ... ボタン	このボタンをクリックすると、**Trend storage** ダイアログボックスが開きます。ランタイムでのトレンド記録の動作に関する追加パラメーターを設定することができます。
高度な設定 ... ボタン	このボタンをクリックすると、**Advanced Trend Settings** ダイアログボックスが開きます (*444* ページ)。データの取得と保存に関するパラメーターを設定できます。

トレンドストレージダイアログボックス

Record Settings ダイアログボックスの **Trend Storage...** ボタンをクリックして、**Trend Storage** ダイアログボックスを開きます。

Trend Storage ダイアログボックスには、トレンド記録中にデータをバッファー処理するための設定が含まれています。

パラメーター	詳細
変数の最大数	データベースで管理されるトレンド変数の最大数を定義します。この値を変更すると、EcoStruxure Machine Expert は、次回のコントローラーへのログイン時にダウンロードを強制することによって、コントローラー上のデータベースを更新します。データベースはコントローラーからアップロードされ、データベースの変数はこの新しい設定に従って増減された後、コントローラーにダウンロードされます。
Store every N milliseconds	記録されたデータがデータベースに永続的に格納される前に一時的にローカルバッファーに格納される時間間隔を定義するミリ秒数です。時間間隔は、内部的にタスクサイクル数に変換されます。タスクサイクルの期間はタスク設定で定義されます。 高い値を指定すると、ランタイムのパフォーマンスが向上します。このデメリットは、値が大きいほど、コントローラーの状態が変化した場合に失われる可能性のあるデータ量が増加することです。低い値はこのリスクを軽減します。このデメリットは、トレンドの視覚化の制御が大量のデータによって低下することです。
許容範囲	記録を制限するオプションを有効にします。

パラメーター		詳細
	No Limit	保存されるトレンド記録データの量は制限されません。 この設定を使用すると、定期的に管理されていない場合、最終的にコントローラー内のファイルスペースが使い果たされるため、このオプションはテスト目的でのみ使用してください。
	Maximum number of records	このオプションは、データベースに格納されるデータレコードの最大数を定義します。各データレコードは、タイムスタンプとトレンド変数の値で構成されています。 **注記：**この制限に達すると、トレンド記録は停止します。
	Maximum storage size	このオプションは、トレンド格納用に予約されている最大メモリーサイズを定義します。このメモリー内に格納できるレコード数は、個々のレコードのサイズによって異なります。 リストから適切な単位 (キロバイト (Kb)、メガバイト (Mb)、またはギガバイト (GB)) を選択します。 **注記：**トレンド格納用に予約されているメモリーがいっぱいの場合、トレンド記録は継続し、最も古いレコードが上書きされます。

Advanced Trend Settings ダイアログボックス

Record Settings ダイアログボックスの**高度な設定 ...** ボタンをクリックして、**Advanced Trend Settings** ダイアログボックスを開きます。

Advanced Trend Settings ダイアログボックスでは、ランタイムでのトレンド記録の動作に関連するその他のパラメーターを設定できます。

パラメーター	詳細
測定頻度 (サイクル数)	完了したサイクル数に応じて、ランタイムシステムによってデータが取得される頻度を定義します。 時間間隔は、タスク構成の設定を使用して頻度に応じて計算されます。従って、計算は少なくともタスク周期時間が設定されている場合にのみ実行できます。結果は入力フィールドの右側に表示されます。例 : **1h1m1s1ms** 初期値は、**1** です。これは、データがサイクルごとに取得されることを意味します。
Additional Runtime Buffer for	ランタイムシステムがデータの記録に追加の「オーバーフロー」バッファーを使用できる時間をミリ秒単位で定義します。 例 : **1000 ms** **注記：**この追加バッファーは、トレンドデータの保存中に遅延が発生した場合にデータの損失を回避するのに役立ちます。

変数の設定

概要

Trend Recording Editor のツリービューでトレンド変数を選択すると、**変数設定**付き **Trend Recording** ダイアログボックスが開きます。記録する変数とパラメーター、およびとその表示方法を設定することができます。

変数設定付き **Trend Configuration** ダイアログボックスは トレース用のダイアログボックスに対応しています (*424* ページ)。トレンド変数は、ウィンドウの左側にツリー構造で表示されます。先頭ノードにはトレンド名が付いています。

変数の設定と変更

変数設定を選択するには、左側のツリービューでその変数を選択します。現在の設定がダイアログボックスの右側に表示されます。後から変数設定を変更するには、ツリービューで変数の項目を選択し、**変数設定**ダイアログボックスを再度使用します。

パラメーター		詳細
変数		値を取得するリストから IEC 変数を選択してください。変数は有効な型である必要があります。
パラメーター		値を取得するリストからパラメーターを選択してください。パラメーターは有効な型である必要があります。
Show variable name		**Show variable name** オプションが選択されている場合は、IEC 変数の名前がランタイムモードのトレンド図の視覚化によって、単独で、または**説明**の後の括弧内に表示されます。 このオプションが選択されていない場合は、IEC 変数の名前は表示されますが、**説明**の後の括弧内には表示されません。
詳細		ビジュアライゼーションを使用しているユーザーがトレンド図の変数を選択した際に表示されるツールチップのテキストを入力します。このテキストは、ローカライズ可能な GlobalTextList オブジェクトに挿入されます。 **Show variable name** オプションが選択されている場合、ツールチップには括弧内に表示された変数名が追加されます。 例 : Sensor A (PLC_PRG.iSensor_A) **説明**ボックスにテキストを入力しない場合は、**Show variable name** がデフォルトで選択され、変数名のみが表示されます。 例 : PLC_PRG.iSensor_A 凡例がトレンドに割り当てられている場合、トレンド変数は凡例内でラベル付けされ、トレンドがここで設定されたとおりに表示されます。
グラフの色		トレンド図で変数が表示される色。 をクリックして色選択のダイアログを開きます。
線の種類		取得した値を線グラフとして表示する方法を定義します。データが大きい場合は、**線**オプションを使用します。
	ライン	値が繋がれ線を形成します。
	ステップ	値が階段状に繋がれます。
	なし	値は繋がれません。
点の種類		取得した値を点グラフとして表示する方法を定義します。データが大きい場合は、**なし**オプションを使用します。
	ドット	値は点で表示されます。
	クロス	値は十字で表示されます。
	なし	値は表示されません。
最小警告をアクティブ化		このオプションが有効な場合、変数値が下限値よりも低いとメッセージが表示されます。
クリティカル下限値		トレンド変数の値が制限を下回ると、変数は制限を下回る値に対して定義された**色**で表示されます。
最小警告色		変数が下回るときに表示される色。
最大警告をアクティブ化		このオプションが有効な場合、変数値が上限値よりも高いとメッセージが表示されます。

パラメーター	詳細
クリティカル上限値	トレンド変数の値が制限を上回ると、変数は制限を上回る値に対して定義された**色**で表示されます。
最大警告色	変数が上回るときに表示される色。

トレンド変数は、型が STRING、WSTRING、または ARRAY でない限り、IEC 変数、プロパティー、リファレンス、ポインターの内容、ARRAY 要素、または列挙型にすることができます

トレンドデータの記録設定手順

概要

次の手順を行い、トレンド記録の設定をします。
- タスクの割り当て
- トレンド変数の挿入
- トレンド変数の削除
- トレンドメモリーの設定
- トレンドの詳細設定

タスクの割り当て

トレンドの記録が実行されるタスクを定義します。通常、トレンド記録はメインプログラムと同じタスクで実行されます (例 : PLC_PRG)。

手順	手順内容
1	トレンド記録オブジェクトを**ツールツリー**でダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	ツリービューで一番上のノードを選択します。 **結果** : **レコード設定**がダイアログボックスの右側に表示されます。
3	**タスク** リストを開き関連付けられたアプリケーションを実行するタスクを選択します。

トレンド変数の挿入

トレンド変数の数には制限があり、**Trend storage** ダイアログボックスで設定します。

手順	手順内容
1	トレンドオブジェクトを**ツールツリー**でダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	ツリービュー下の**変数の追加**コマンドを実行します。 **結果** : 新しい変数が挿入され、その設定が表示されます。
3	**変数**リストから有効な IEC 変数を入力するか、入力**パラメーター**を選択して有効なパラメーターを入力します。 **結果** : 信号がトレンド変数に割り当てられます。
4	トレンド変数の表示を設定します。
5	トレンド変数の色を設定します。

トレース設定の**変数設定**も参照してください (*424* ページ)。

トレンド変数の削除

手順	手順内容
1	トレンドオブジェクトを**ツールツリー**でダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	ツリービューでトレンド変数をクリックします。 **結果** : 変数が選択されます。
3	ツリービュー下の**変数の削除**コマンドを実行するか、**削除**キーを押します。

トレンドメモリーの設定

手順	手順内容
1	トレンドオブジェクトを**ツールツリー**でダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	ツリービューで一番上のノードを選択します。 **結果** : **レコード設定**がダイアログボックスの右側に表示されます。
3	**Trend storage...** ボタンをクリックします。 **結果** : **Trend Storage** ダイアログボックスが開きます (*443* ページ)。
4	設定の調節します。

トレンドの詳細設定

手順	手順内容
1	トレンドオブジェクトを**ツールツリー**でダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	ツリービューで一番上のノードを選択します。 **結果** : **レコード設定**がダイアログボックスの右側に表示されます。
3	**詳細 ...** ボタンをクリックします。 **結果** : **Advanced Trend Settings** ダイアログボックスが開きます。
4	設定の調節します。

26.3
トレンドデータの記録

記録処理の開始

前提条件

記録処理を開始するには、**Trend recording** タイプに設定されたオブジェクトが必要です。

トレンド記録の開始

手順	手順内容
1	コントローラーを開始します。
2	トレンド記録を含むアプリケーションを選択し、**アプリケーション → ログイン**コマンドを実行します。
3	アプリケーションを起動させます。

コントローラーを停止したりダウンロードを実行する (**オンライン変更** コマンドを実行 *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*) 前に、トレンド記録処理を停止してトレンドデータをデータベースに保存します。

注記
データの損失 コントローラーの状態を実行状態から変更する前に、トレンドファンクションを停止しトレンドデータを保存します。 **上記の指示に従わないと、物的損害を負う可能性があります。**

条件付トレンド記録の開始

前提条件 : トレンド設定が完了している。

手順	手順内容
1	トレンドオブジェクトをダブルクリックします。 **結果** : **Trend Recording Editor** が開きます。
2	設定のツリービューで一番上のノードを選択します。 **結果** : **レコード設定**がダイアログボックスの右側に表示されます。
3	ブール型変数を **Record Condition** フィールドに割り当てます。 ブール型変数でデータの記録を制御します。
4	トレンド記録を含むアプリケーションを選択し、**アプリケーション → ログイン**コマンドを実行します。
5	アプリケーションを起動させます。

第 27 章
単位の変換

この章について

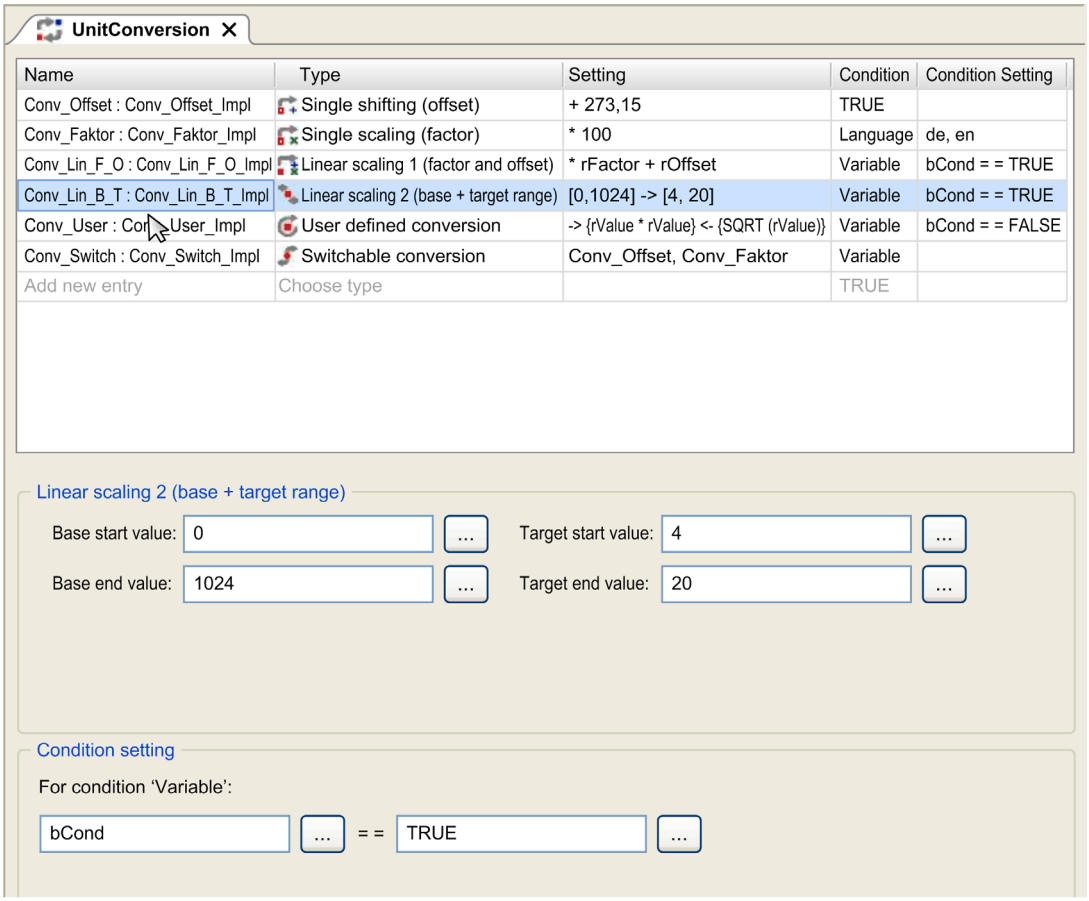
この章には次の項目が含まれています。

単位変換の設定

概要

UnitConversion 設定エディターで、視覚化要素内および IEC エディター内の変数に使用できる変換を定義します。**アプリケーションツリー**で、エディターを **UnitConversion** オブジェクトとして追加します。そこで名前を変更することができます。

UnitConversion エディター



プロジェクトに対してさまざまな変換を定義できます。それぞれに**名前**を入力し、**タイプ**選択します。それぞれの変換式が**設定**列に自動的に表示されます。変換を実行するタイミングを決定するために**条件**を入力します。選択した**タイプ**に応じて、表の下の領域に特定のパラメーターを定義します。

エディターには、次の変換タイプがあります。

- オフセットを使った計算 : **単一シフト (オフセット)** (*453* ページ)
- 係数を使った計算 : **単一スケーリング (係数)** (*453* ページ)
- 係数およびオフセットを使った計算 : **線形スケーリング 1 (係数およびオフセット)** (*453* ページ)
- 基準範囲と目標範囲を使った計算 : **線形スケーリング 2 (ベース + ターゲット範囲)** (*453* ページ)
- ユーザー定義式を使った計算 : **ユーザー定義変換** (*454* ページ)
- 変数値により異なる計算 : **切り替え可能変換** (*454* ページ)

IEC エディターでの単位変換の使用法については、*IEC エディターの使用法* の章を参照してください (*455* ページ)。

視覚化での単位変換の使用方法については、EcoStruxure Machine Expert オンラインヘルプの *視覚化* の部分にある *単位変換の使用* の章を参照してください。

条件

計算が実行されるときの条件を 3 つ選択できます。

条件	詳細
TRUE	変換は常に実行されます。
言語	視覚化の言語 (VisuElems.CurrentLanguage 変数の値)
変数	変数の値によって変換が実行されます。比較値は、定数、変数、または IEC 式にすることができます。

単一シフト (オフセット)

入力値にオフセットを追加する場合にこの変換を使用します。

(出力値 = 入力値 + オフセット)

パラメーター	詳細
オフセット	値、または変数

単一スケーリング (係数)

入力値を係数で掛ける場合にこの変換を使用します。

(出力値 = 入力値 * 係数)

パラメーター	詳細
係数	値、または変数

線形スケーリング 1 (係数およびオフセット)

入力値を係数で掛け、オフセットを追加する場合にこの変換を使用します。

(出力値 = (入力値 * 係数) + オフセット)

パラメーター	詳細
係数	値、または変数
オフセット	値、または変数

線形スケーリング 2 (ベース + ターゲット範囲)

値を入力と出力範囲によって定義したい場合にこの変換を使用します。係数とオフセットは内部で計算されます。

パラメーター	詳細
Base start value	入力範囲の低い値
Base end value	入力範囲の高い値
Target start value	出力範囲の低い値
Target end value	出力範囲の高い値

例

Base start value: 0
Base end value: 1024
Target start value: 4.0
Target end value: 20.0

ユーザー定義変換

入力値をユーザー定義式で計算する場合にこの変換を使用します。変数 rValue を入力値として使用します。この式では、すべての IEC オペランドが使用できます。

パラメーター	詳細
変換	入力値を変換する式
反転	計算値を元に戻す式

切り替え可能変換

言語、変数の値などによって異なる変換をする場合にこの変換を使用します。すべての定義された変換を使用できます。

IEC エディターでの使用

概要

すべての IEC エディターで単位変換を使用できます。新しい変換ごとに変換を作成すると、ファンクションブロックが自動的に作成されます。

ファンクションブロックは 2 つのメソッドを含みます。

メソッド	詳細
Convert	変換式に従って入力値を計算します。
Reverse	入力値に逆算します。

これらのメソッドはすべての IEC エディターで呼び出せます。

ST の例

この例では、Conv_4_20 が変換名です。

```
VAR
   rValue : REAL;
   rConvertedValue : REAL;
END_VAR
rConvertedValue := Conv_4_20.convert(rValue);
```

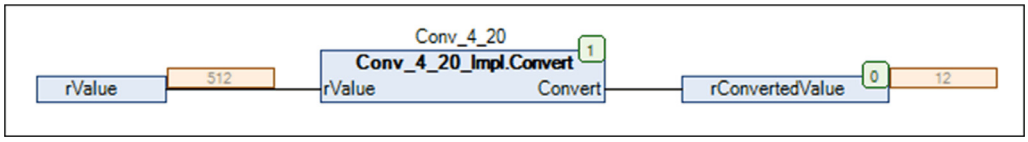
入力値に逆算します。

```
rValue := Conv_4_20.reverse(rConvertedValue);
```

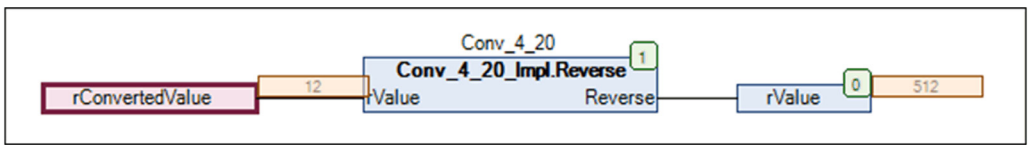
CFC の例

CFC エディターで、要素の上に変換のインスタンス名を入力します。要素でメソッドが定義されます。

CFC メソッド Convert での例



CFC メソッド Reverse での例



第 30 章
Cam モーションエディター

この章について

この章には次のセクションが含まれています。

30.1
Cam モーションエディター - 一般情報

このセクションについて

このセクションには次の項目が含まれています。

一般情報

概略

cam モーションエディターは、cam ダイアグラムを作成およびパラメーター化し、FB_MultiCam / MC_Cam_ID ファンクションブロックに割り当てる際に使用します。作成したダイアグラムを Motion Sizer で使用してドライブのサイズ設定などを行うことができます。

cam ダイアグラムは、VDI 2143 規格で定義された運動法則に基いています。これらのダイアグラムを使用して、マスター軸 ϕ の位置に対する軸の動きを設計することが可能です。また Cam ダイアグラムは、仮想ラインシャフトの動きに追従するサーボ軸の動きをモデル化する際にも用いられます。このラインシャフトの角度は、ϕ で表されます。

軸の異なる一連の動作に対応する各 cam ダイアグラムのセグメントを定義することができます。

Cam オブジェクトの追加

概略

プロジェクトに cam オブジェクトを追加するには、**ツールツリー**内の**アプリケーション**ノードを右クリックし、コンテキストメニューから**オブジェクトの追加 → CamDiagram...** コマンドを実行します。

結果 : **CamDiagram** ノードが挿入されます。初期設定では、ダイアグラムオブジェクトの子オブジェクトである**セグメント**サブノードが含まれています。

Cam オブジェクトのモーションエディターを開く

手順

cam ダイアグラムまたは cam セグメントのモーションエディターを開くには、**ツールツリー**内のノードをダブルクリックします。

条件	結果
CamDiagram ノードをダブルクリックした場合	複数のタブ付きエディタービューに**モーションエディター**タブが開きます。
Segment ノードをダブルクリックした場合	複数のタブ付きエディタービューに**モーションエディター**タブが開き、ダイアグラムの選択したセグメントが緑色の背景で強調表示されます。

複数のモーションエディターを表示する

同時に複数のモーションエディタービューを表示し、相互依存関係にあるモーションを比較することができます。その際は、**ウインドウ → 新規水平タブグループ**、もしくは**ウインドウ → 新規垂直タブグループ**のコマンドを実行します。

30.2
Cam データから IEC プログラムコードを生成

このセクションについて

このセクションには次の項目が含まれています。

Cam データをファンクションブロックに使用

概略

Cam ダイアグラムは、cam データの構造体 (その構造体に対応する数の cam ポイントで構成) を生成するためのベースとして使用されます。

これらのデータ構造体を使用して、FB_MultiCam / MC_Cam_ID ファンクションブロック、または同等の機能を持つ POU をパラメーター化することができます。

Cam データは、アプリケーションプログラム内で cam データの構造体を処理しているファンクションブロック / モジュールの初期化に使用されます。

対象ファンクションブロックの指定

cam モーションエディターの **設定**タブには、既存の cam データ構造体を対象の構造体として指定することのできる入力フィールドがあります。対象の構造体が定義されている場合、cam データはプログラム初期化時に自動的にその構造体に書き込まれます。式の判定が不可能な場合は、診断メッセージが表示されます。

定義された対象のデータ構造体が CommonMotionType か PacDriveLib のどちらであるかを EcoStruxure Machine Expert が検出し、それに応じたパラメーター化が実行されます。

データ構造体	ライブラリー
ST_Multicam	PacDriveLib
FB_MultiCam / MC_Cam_ID	CommonMotionType

Cam データ構造体用データの生成

プロジェクトがビルドされると (**Build** メニューまたはショートカット **F11** の**ビルド**コマンドから実行)、IEC ソースコードが生成され、指定した cam データ構造体に書き込まれます。

cam データ構造体のパラメーター化の命令は、以下の形式で生成されます。

```
MyMultiCam.MultiCamData.diNumberOfCamPoints := 3;
MyMultiCam.MultiCamData.lrYPeriod := 0;
MyMultiCam.MultiCamData.astCamPoint[0].lrX := 0;
MyMultiCam.MultiCamData.astCamPoint[0].lrY := 100;
MyMultiCam.MultiCamData.astCamPoint[0].lrM := 0;
MyMultiCam.MultiCamData.astCamPoint[0].lrK := 0;
MyMultiCam.MultiCamData.astCamPoint[0].lrLambda := 0.5;
MyMultiCam.MultiCamData.astCamPoint[0].lrC := 1;
MyMultiCam.MultiCamData.astCamPoint[0].etCamType := ET_CAMTYPE.Poly5Com;
MyMultiCam.MultiCamData.astCamPoint[1].lrX := 100;
MyMultiCam.MultiCamData.astCamPoint[1].lrY := 0;
MyMultiCam.MultiCamData.astCamPoint[1].lrM := 0;
MyMultiCam.MultiCamData.astCamPoint[1].lrK := 0;
MyMultiCam.MultiCamData.astCamPoint[1].lrLambda := 0.5;
MyMultiCam.MultiCamData.astCamPoint[1].lrC := 1;
MyMultiCam.MultiCamData.astCamPoint[1].etCamType := ET_CAMTYPE.SimplSin;
```

この例では、MyMultiCam.MultiCamData が対象のデータ構造体として設定されています。

データは、オフラインモードで編集された cam ダイアグラムを基に生成されます。オンラインモードで更新はされません。

CamPoints の最大数

各 cam セグメントは、セグメントの左端に位置するデータを指定する CamPoint によって定義されます。cam セグメントの右端の CamPoint は、次の cam セグメントの左端の CamPoint に対応しています。cam ダイアグラムの終点を識別するために、最後の cam segment は右端を独自の CamPoint で定義します。

1 つの cam データ構造体に対し、最大 32 の CamPoints が利用可能です。セグメント数が 32 以上の cam ダイアグラムがある対象構造体が選択された場合は、診断メッセージが表示されます。32 セグメント以上の軸のモーションを設計する際は、複数の cam ダイアグラムに分割する必要があります。

注記：複数の cam ダイアグラムが組み込まれている場合、Motion Sizer でドライブのサイズ設定を行う際に、Motion Sizer は全てのサブ cam ダイアグラムを含む全ての cam 曲線を使用します。

ファンクションブロックの初期化

設定された cam データを対象の構造体に入れるため、起動時に 1 回初期化コードが実行されます。初期化タスクは、ユーザーによる初期化処理の完了後、初期化コードとして呼び出される POU プログラムで実行されます。

Cam ダイアグラムのソースコードのコピー

概略

cam モーションエディターの**生成した IEC ソースコード**タブに、cam ダイアグラム用に生成された読み取り専用のソースコードが表示されます。cam ダイアグラムが変更されるたびに、コードが再生成されます。

IEC コードをコピーし、プログラムの別用途に利用することができます。

30.3
Cam モーションエディターのオンラインビューとファンクション

Cam モーションエディターのオンラインビューとファンクション

オンラインモードでの　Cam モーションエディター

cam モーションエディターが有効で表示されている状態で EcoStruxure Machine Expert のオンラインモードを有効にすると、cam モーションエディターもオンラインモードに切り替わります。ただし、オフラインのデータが表示されたままになります。

オンラインモードが有効である間は、cam ダイアグラムの値を変更することはできません。

- セグメントを選択することはできません。
- セグメントデータを変更することはできません。
- 対象の変数を指定することはできません。
- ダイアグラムの軸の割り当てを変更することをはできません。

上記の通りですが、他のタブを選択して cam ダイアグラムのオフライン情報を表示することは可能です。

オンラインモード中に別の cam モーションエディターを選択すると、対応する cam データ構造体のオフラインデータが表示されます。

オンライン変更

cam モーションエディターは、**オンライン変更** ファンクション (*205* ページ) に対応しています。

コントローラーに cam データ構造体をダウンロードした後でオフラインモードに切り替えると、個々のセグメントの編集が可能になります。

再度オンラインモードに切り替えると変更箇所が検出され、ログインしてオンライン変更を実行 (*203* ページ) するかを尋ねられます。**OK** をクリックして確定すると、cam データ構造体が更新されます。

変更を反映させるためには、FB_MultiCam / MC_Cam_ID ファンクションブロックを変更された cam データ構造体に考慮した適切なモードに切り替える必要があります。

その際は、**デバッグ → 強制値** コマンドで入力パラメーター (例 : iq_xNewCam...) を設定します。詳細は、PD_PacDriveLib の FB_MultiCam 、もしくは CommonMotionType ライブラリーガイド内 FB_MultiCam / MC_Cam_ID に記載の説明を参照してください。

30.4
連続していない位置の経路

位置が連続していない経路

概略

cam データ構造体を使用して FB_MultiCam / MC_Cam_ID ファンクションブロックをパラメーター化します。このデータ構造体は、位置が連続していない経路をマッピングすることはできません。従って、パラメーター化された経路に矛盾がある場合その cam 経路に沿って FB_MultiCam / MC_Cam_ID ファンクションブロックが動くことはありません。

無効なパラメーター化を避けるために、位置に矛盾がないかを検証します。

検証は、cam segment の始点と、その直前のセグメントの終点との位置を比較して行われます。無効なパラメーター化があるたびに、コンパイルエラーが生成されます。

30.5
ダイアログボックス

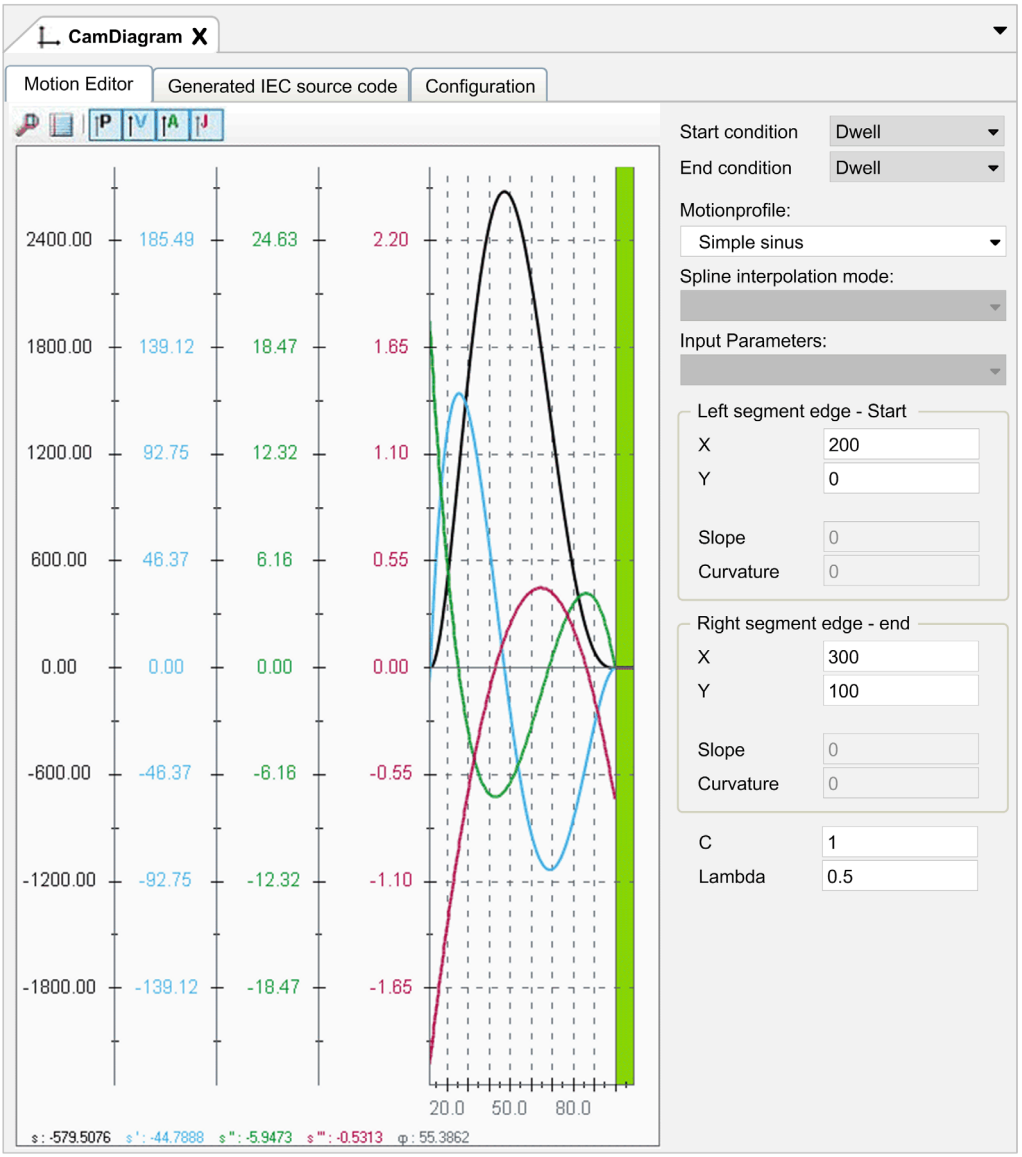
このセクションについて

このセクションには次の項目が含まれています。

モーションエディター

モーションエディター タブ

モーションエディタータブの図



項目	詳細	値の範囲
CamDiagram	このタブの左側のビューには、**ツールツリー**でダブルクリックしたCamDiagram の個々のセグメントが表示されます。 選択した cam segment は緑色で強調表示されます。cam segment を選択するには、**ツールツリー**ビュー内のセグメントをダブルクリックするか、**モーションエディター**タブ内のセグメントをクリックします。	–
開始条件	選択した cam segment の始点に適用される条件。	• **ドエル** • **速度** • **戻る** • **モーション**

項目	詳細	値の範囲
終了条件	選択した cam segment の終点に適用される条件。	● **ドエル** ● **速度** ● **戻る** ● **モーション**
モーションプロファイル	セグメントのプロファイルは運動法則で説明されます。利用可能なモーションプロファイルは、選択した **開始条件** と **終了条件**によって決まります。*開始 / 終了条件の組み合わせにより利用できるモーションプロファイル* (*472* ページ) の表を参照してください。	● **直線** ● **二次放物線** ● **5 次多項式** ● **単純な正弦** ● **変形正弦** ● **変更された加速台形** ● **5 次の一般多項式** ● **正弦 - 直線の組み合わせ** ● ユーザー定義モーションプロファイル
スプライン補間モード	このパラメーターは、ユーザー定義のモーションプロファイルにのみ利用可能です。また、プロファイルの始点と終点のエッジ条件の解決方法を定義します。	● **自然スプライン** ● **境界線の勾配底部のスプライン** ● **周期的スプライン**
入力パラメーター	設定するパラメーターをリストから選択します。**結果** : 対応するテキストフィールドが編集可能になります。残りのパラメーターは EcoStruxure Machine Expert によって算出されます。	表にリストされる値は、選択する**モーションプロファイル**によって異なります。以下にある、各運動法則で使用できるパラメーターの表を参照してください。
左セグメント端 - 開始 セクション		
X	選択した cam segment の始点の x 座標。 セグメントの始点における仮想ラインシャフトの角度を表します。	10 進数 単位 : ϕ
Y	選択した cam segment の始点の y 座標。 セグメントの始点における軸の位置を表します。これが位置 x での s の値です。	10 進数 単位 : 位置
勾配	選択した cam segment の始点における位置曲線の勾配。これが s' の値になります。	10 進数 単位 : 位置 / ϕ
曲率	選択した cam segment の始点における位置曲線の曲率。これが s'' の値になります。	10 進数 単位 : 位置 / ϕ^2
右セグメント端 - 終了セクション		
X	選択した cam segment の終点における x 座標。 セグメントの終点における仮想ラインシャフトの勾配を表します。	10 進数 単位 : ϕ
Y	選択した cam segment の終点における y 座標。 セグメントの終点における軸の位置を表します。これが位置 x での s の値です。	10 進数 単位 : 位置
勾配	選択した cam segment の終点における位置曲線の勾配。これが s' の値です。	10 進数 単位 : 位置 / ϕ
曲率	選択した cam segment の終点における位置曲線の曲率。これが s'' の値です。	10 進数 単位 : 位置 / ϕ^2

項目	詳細	値の範囲
C	選択した cam segment で速度変更に消費される割合を定義します。 値 1 は、セグメント中に等速フェーズがないことを示します。加速および減速処理は、セグメント全体を通して実行されます。 値が 0.5 の場合は、加速および減速処理がセグメントの時間の半分を使って行われることを示します。セグメントの残り半分、加速と減速の変曲点は、等速フェーズ用に確保されます。	0...1
ラムダ	選択した cam segment 内の x 軸 (φ) の、変曲点 (**C**=1 のとき) または等速フェーズ (**C**<1 のとき) の位置を定義します。**ラムダ** は、セグメント内の仮想ラインシャフト角度の加速と減速の比率を表します。 加速フェーズのみで構成されるセグメント (**開始条件** = **ドエル**、**終了条件** = **速度**) では、**ラムダ**の値は 1 に固定されています。これは、速度変更の時間の 100% が加速に使われ、0% が減速に使われることを意味しています。 減速フェーズのみで構成されるセグメント (**開始条件** = **速度** , **終了条件** = **ドエル**) では、**ラムダ** の値は 0 に固定されています。これは、速度変更の時間の 0% が加速に使われ、100% が減速に使われることを意味しています。 加速フェーズと減速フェーズのみで構成されるセグメント (**開始条件** = **ドエル** , **終了条件** = **ドエル**) では、**ラムダ**の値は 0 より大きく、1 未満です。値が 0.5 のときは、加速フェーズと減速フェーズが x 軸 (φ) 上で同じ長さになっていることを意味しています。全体の長さの 50% が速度変更に使われています。値が 0.1 のときは、x 軸 (φ) 上の加速フェーズと減速フェーズの長さの 10% が減速に使われていることを意味しています。90% が加速に使われています。	0...1

特定の運動法則に関する追加情報

名称	詳細
C	モーション曲線の　Cam 率。 有効間隔 : 入力フィールドの値は、0 より大きく 1 以下にしてください。 例 : **C** = 1: 経路は曲がっている、もしくは cam です。 **C** = 0.001: 経路は直線とほぼ完全に一致しています。 **C** = 0.4: 経路の直線率は 60% で、cam 率は 40% です。
ラムダ	変曲点の位置。 ドエル / ドエルの運動法則における有効な間隔内の変曲点 : 入力フィールドの値は、0 以上 1 以下にしてください。 **ラムダ** = 0.5 の場合、変曲点は cam セグメントのちょうど中央にあります。 **ラムダ** = 0.00001 の場合、変曲点はほぼ左端にあります。 ドエル / 速度、もしくは速度 / ドエルの運動法則ではほとんどの場合、**ラムダ** は正確に 1 または 0 で、ちょうど右端または左端にあります。

開始条件と終了条件の組み合わせに利用可能なモーションプロファイル

境界条件 (開始条件、終了条件) の組み合わせによって、以下の**モーションプロファイル**を選択することができます。

範囲	**ドエル (v=0, a=0)**	**速度 (v<>0, a=0)**	**戻る (v=0, a<>0)**	**モーション (v<>0, a<>0)**
ドエル (v=0, a=0)	直線 二次放物線 5 次多項式 単純な正弦 変形正弦 変更された加速台形 5 次の一般多項式	二次放物線 5 次多項式 単純な正弦 変形正弦 変更された加速台形 5 次の一般多項式 (ラムダ = 1)	5 次の一般多項式	5 次の一般多項式
速度 (v<>0, a=0)	二次放物線 5 次多項式 単純な正弦 変形正弦 変更された加速台形 5 次の一般多項式 (Lambda = 0)	直線 5 次の一般多項式	5 次の一般多項式	5 次の一般多項式
戻る (v=0, a<>0)	5 次の一般多項式		正弦と直線の組み合わせ 5 次の一般多項式	5 次の一般多項式
モーション (v<>0, a<>0)	5 次の一般多項式			

二次放物線、5 次多項式、正弦、変形正弦、変更された加速台形の運動法則のパラメーター

以下の表には、**二次放物線**、**5 次多項式**、**正弦曲線**、**変形正弦**、**変更された加速台形**の運動法則に対応したパラメーターと、選択した**開始条件** と**終了条件**を基に得られる値が記載されています。

パラメーター	**開始条件 - 終了条件**		
	ドエル - ドエル	**ドエル - 速度**	**速度 - ドエル**
X 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力、または EcoStruxure Machine Expert が算出	ユーザー入力、または EcoStruxure Machine Expert が算出
勾配 始点	0 に固定	0 に固定	ユーザー入力、または EcoStruxure Machine Expert* が算出
曲率 始点	0 に固定	0 に固定	0 に固定
X 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力、または EcoStruxure Machine Expert が算出	ユーザー入力 *、または EcoStruxure Machine Expert が算出
勾配 終点	0 に固定	ユーザー入力、または EcoStruxure Machine Expert が算出 *	0 に固定
曲率 終点	0 に固定	0 に固定	0 に固定
C	ユーザー入力 (0...1)	ユーザー入力 * (0...1) 、または EcoStruxure Machine Expert が算出	ユーザー入力 * (0...1) 、または EcoStruxure Machine Expert が算出
ラムダ	ユーザー入力 (0...1)	1 に固定	0 に固定
* 初期設定では、4 つのパラメーターのうち 1 つは EcoStruxure Machine Expert が算出しますが、3 つは編集可能です。パラメーターの横にあるオプションをクリックして、これを変更することができます。			

直線の運動法則のパラメーター

以下の表には、**直線**の運動法則に対応したパラメーターと、選択した**開始条件**と**終了条件**を基に得られる値を記載しています。

パラメーター	**開始条件 - 終了条件**	
	ドエル - ドエル	**速度 - 速度**
X 軸 始点	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力、または EcoStruxure Machine Expert が算出
勾配 始点	0 に固定	ユーザー入力、または EcoStruxure Machine Expert が算出 *
曲率 始点	0 に固定	0 に固定
X 軸 終点	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力、または EcoStruxure Machine Expert が算出
勾配 終点	0 に固定	**勾配** 始点のパラメーターと同じ値が使用されます。
曲率 終点	0 に固定	0 に固定
* 初期設定では、**勾配** の始点パラメーターは EcoStruxure Machine Expert が算出しますが、**Y** 軸始点および **Y** 軸終点パラメーターは編集可能です。パラメーターの横にあるオプションをクリックして、これを変更することができます。		

正弦曲線と直線の組み合わせの運動法則のパラメーター

正弦 - 直線の組み合わせの運動法則は、**開始条件 → 戻る**と**終了条件 → 戻る**にのみ対応しています。利用可能なパラメーターは以下の表に記載のとおりです。

パラメーター	**開始条件 - 終了条件**
	戻る - 戻る
X 軸 始点	ユーザー入力
Y 軸 始点	ユーザー入力
勾配 始点	ユーザー入力、または EcoStruxure Machine Expert が算出
曲率 始点	ユーザー入力、または EcoStruxure Machine Expert が算出
X 軸 終点	ユーザー入力
Y 軸 終点	ユーザー入力
勾配 終点	ユーザー入力、または EcoStruxure Machine Expert が算出
曲率 終点	ユーザー入力、または EcoStruxure Machine Expert が算出
C	ユーザー入力、または EcoStruxure Machine Expert が算出
ラムダ	ユーザー入力、または EcoStruxure Machine Expert が算出
* **入力パラメーター**のリストを使用して 6 つのパラメーターのうち 2 つを編集可能として選択します。残りの 4 つのパラメーターは、EcoStruxure Machine Expert が算出します。	

5 次の一般多項式の運動法則のパラメーター

5 次の一般多項式の運動法則が対応する**開始条件** - **終了条件**の値の組み合わせは 15 あるため、この運動法則には異なるパラメーター表が 4 つ用意されています。

以下の表は、**5 次の一般多項式**の運動法則に対応したパラメーターと、選択した**開始条件 = ドエル**を基に得られる値、および**終了条件**の値を記載しています。

パラメーター	**開始条件 - 終了条件**			
	ドエル - ドエル	**ドエル - 速度**	**ドエル - モーション**	**ドエル - 戻る**
X 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 始点	0 に固定	0 に固定	0 に固定	0 に固定
曲率 始点	0 に固定	0 に固定	0 に固定	0 に固定

パラメーター	開始条件 - 終了条件			
	ドエル - ドエル	ドエル - 速度	ドエル - モーション	ドエル - 戻る
X 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 終点	0 に固定	ユーザー入力	ユーザー入力	0 に固定
曲率 終点	0 に固定	0 に固定	ユーザー入力	ユーザー入力

以下の表には、**5 次の一般多項式**の運動法則に対応したパラメーターと、選択した**開始条件** = **モーション**を基に得られる値、および**終了条件**の値が記載されています。

パラメーター	開始条件 - 終了条件			
	モーション - ドエル	モーション - モーション	モーション - 戻る	モーション - 速度
X 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
曲率 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
X 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 終点	0 に固定	ユーザー入力	0 に固定	ユーザー入力
曲率 終点	0 に固定	ユーザー入力	ユーザー入力	0 に固定

以下の表には、**5 次の一般多項式**の運動法則に対応したパラメーターと、選択した**開始条件** = **戻る** を基に得られる値、および**終了条件**の値が記載されています。

パラメーター	開始条件 - 終了条件			
	戻る - ドエル	戻る - モーション	戻る - 戻る	戻る - 速度
X 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 始点	0 に固定	0 に固定	0 に固定	0 に固定
曲率 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
X 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 終点	0 に固定	ユーザー入力	0 に固定	ユーザー入力
曲率 終点	0 に固定	ユーザー入力	ユーザー入力	0 に固定

以下の表には、**5 次の一般多項式**の運動法則に対応したパラメーターと、選択した**開始条件** = **速度** を基に得られる値、および**終了条件**の値が記載されています。

パラメーター	開始条件 - 終了条件			
	速度 - ドエル	速度 - モーション	速度 - 戻る	速度 - 速度
X 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 始点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
曲率 始点	0 に固定	0 に固定	0 に固定	0 に固定
X 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
Y 軸 終点	ユーザー入力	ユーザー入力	ユーザー入力	ユーザー入力
勾配 終点	0 に固定	ユーザー入力	0 に固定	ユーザー入力
曲率 終点	0 に固定	ユーザー入力	ユーザー入力	0 に固定

IEC ソースコードの生成

生成した IEC ソースコード タブ

生成した IEC ソースコードタブには、**設定** タブ内で指定した対象の構造体、および cam ダイアグラムに基づいて自動生成されたプログラムコードが表示されます。

対象の構造体 (cam データ構造体) を定義すると、プログラムの初期化中に cam データがその構造体に自動的に書き込まれます。

対象の構造体が定義されていない場合は、このコードをコピーし ### を変数名に置き換えることで、対象の構造体を手動で設定することも可能です。

このタブに表示されるプログラムコードは、読み取り専用です。cam ダイアグラムが変更されると、コードは再生成されます。

設定

設定 タブ

設定 タブでは、生成する対象の構造体を選択し、プロパティを定義することができます。

設定 タブでは、以下を実行できます。
- ソースコード生成用構造体の選択
- 構造体のプロパティの定義
- カムダイアグラムの軸への割り当て (EDESIGN プロジェクトのみ)

項目	詳細
対象技術	**対象技術** セクションでは、ソースコードの対象構造体 (cam データ構造体) を選択することができます。 構造体を選択、または選択ボタン (...) をクリックし**入力アシスタント**のダイアログボックスを開きます。 対象の構造体が定義されると、アプリケーションプログラムの起動時にこの対象の構造体に cam データが書き込まれます。 式の判定が不可能な場合は、診断メッセージが表示されます。 定義した構造体から、データ構造体が CommonMotionType と PacDriveLib、どちらの構造体であるかが検出され、それに応じて設定が実行されます。
MultiCam 設定	**MultiCam 設定** セクションでは、次のパラメーターの値を設定できます。 ● **ユーザーテーブル ID**: 正の整数 (0 を含む) にすることができる。 ● **Y - 周期** : 正の浮動小数点数 (0 を含む) にすることができる。 どちらのパラメーターも、デフォルト値は 0 です。
軸の割り当て EDESIGN プロジェクトにのみ利用可能	**軸の割り当て** セクションでは、EDESIGN プロジェクトにおける cam ダイアグラムの軸の割り当てを設定することができます。 プロジェクトで利用可能な軸が全て表示されたリストの中から、軸を選びます。 各ダイアグラムに割り当てることのできる軸は 1 つです。デフォルトでは、割り当てはありません。

第 VII 部
プログラミングリファレンス

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
31	変数宣言	479
32	データ型	559
33	プログラミングのガイドライン	583
34	演算子	593
35	オペランド	677

第 31 章
変数宣言

この章について

この章には次のセクションが含まれています。

31.1
宣言

このセクションについて

このセクションには次の項目が含まれています。

一般情報

概要

以下で変数を宣言できます。
- **ソフトウェアカタログの変数**ビュー (*29* ページ)
- POU の**宣言エディター** (*342* ページ)
- **自動宣言**ダイアログボックス (*487* ページ)
- GVL エディター

宣言する変数の種類 (表形式の宣言エディターでは、**スコープ**という名前) は、1 つまたは複数の変数の宣言を囲むキーワードによって指定します。テキスト宣言エディター (*342* ページ) では、共通の変数宣言を VAR と END_VAR で囲みます。

他の変数宣言のスコープについては、以下を参照してください。
- VAR_INPUT
- VAR_OUTPUT
- VAR_IN_OUT
- VAR_GLOBAL
- VAR_TEMP
- VAR_STAT
- VAR_EXTERNAL
- VAR_CONFIG

変数型のキーワードは、属性キーワード (*496* ページ) によって補足されます。

例 : RETAIN (VAR_INPUT RETAIN)

構文

変数宣言の構文 :

<Identifier> {AT <address>}:<data type> {:=<initialization>};

中括弧 {} 内はオプションです。

識別子

識別子は変数の名前です。

識別子を定義する際には、次の事項を考慮してください。
- スペースや特殊文字は使用できません。
- 大文字と小文字の区別なし : VAR1、Var1、および var1 はすべて同じ変数です。
- アンダースコア文字の認識 : A_BCD と AB_CD は 2 つの異なる識別子と見なされます。1 行に複数のアンダースコアは使用できません。
- 長さに制限はありません。
- 複数の使用に関する推奨事項 (次の項を参照)

また、識別子の命名に関する推奨事項 (*483* ページ) の章に記載されている推奨事項を確認してください。

識別子の複数使用 (名前空間)

以下に、識別子の複数使用に関する規制を示します。
- キーワードと同一の識別子を作成しないでください。
- ローカルでは識別子の重複使用をしないでください。
- グローバルでは、識別子を複数使用できます。ローカル変数は、グローバル変数と同じ名前に出来ます。この場合、POU 内のローカル変数が優先されます。
- グローバル変数リスト (GVL) で定義された変数は、別のグローバル変数リスト (GVL) で定義された変数と同じ名前にできます。このコンテキストでは、次の IEC 61131-3 拡張機能を確認してください。
  - グローバルスコープ演算子 : ドット (.) で始まるインスタンスパスはグローバルスコープを開きます。そのため例えば、ivar というグローバル変数と同じ名前のローカル変数がある場合、.ivar はグローバル変数を示します。
  - グローバル変数リスト (GVL) の名前は、このリストに含まれる変数の名前空間として使用できます。異なるグローバル変数リスト (GVL) に、同じ名前の変数を宣言することができます。これらの変数は、変数名の前にリスト名を付けることでアクセスすることができます。
    例
    globlist1.ivar := globlist2.ivar;
    (globlist2 の * ivar は、GVL globlist1 * の ivar にコピーされます)

- 付属のライブラリーのグローバル変数リスト内に定義された変数は、<library namespace>.<name of GVL>.<variable> の構文でアクセスできます。
  例 :
  globlist1.ivar := lib1.globlist1.ivar
  (* ライブラリー lib1 にある globlist2 の ivar が、GVL globlist1 の ivar にコピーされます *)

- またライブラリーは、**ライブラリーマネージャー**を介して組み込まれるときに名前空間が定義されます。従って、<library namespace>.<modulename|variablename> を用いてライブラリーモジュールまたは変数にアクセスできます。入れ子になったライブラリーの場合は、関係するすべてのライブラリーの名前空間を連続して記述しなければならないことに注意してください。
  例 : Lib1 が Lib0 によって参照される場合は、Lib0.Lib1.fun が Lib1 の一部であるモジュール fun にアクセスします :
  ivar := Lib0.Lib1.fun(4, 5); (* 戻り値の fun が、プロジェクト内の ivar にコピーされます *)
  注記 : 参照先ライブラリー Lib の **プロパティ**ダイアログボックスで**この参照がプロジェクトに直接含まれているかのようにすべての IEC シンボルをプロジェクトに発行する**チェックボックスを有効にすると、Lib0.fun. を介して直接 fun モジュールにアクセスすることも可能です。

- **アプリケーションツリーのグローバル**ノードの GVL または POU で宣言された変数には、"__P0OL.". 演算子を前に付けることでアクセスできます。

AT <address>

キーワード AT を使用して、変数を直接アドレスの定義 (*489* ページ) にリンクすることができます。

ファンクションブロックでは、不完全なアドレスステートメントで変数を指定することもできます。そのような変数をローカルインスタンスで使用するには、変数設定内でその変数に対する項目が入力されている必要があります。

型

有効なデータ型 (*560* ページ)。:=< initialization> (*486* ページ) によりオプションとして拡張も可能です。

Pragma 指令

さまざまな目的のためにコード生成に影響を与える必要がある場合、オプションでオブジェクトの宣言部分に pragma 指令 (*505* ページ) を追加できます。

ヒント

変数の自動宣言 (*487* ページ) も可能です。

宣言の入力を短時間でするには、ショートカットキーモード (*488* ページ) を使用します。

識別子の名称に関する推奨事項

概要

識別子は以下の時に定義されます。

- 変数 (変数名) の宣言時
- ユーザー定義データ型の宣言時
- POU (ファンクション、ファンクションブロック、プログラム) の作成時

識別子 (変数宣言の *一般情報* (*481* ページ) を参照してください) を定義する際に注意すべき一般的な項目に加え、次の推奨事項に従って可能な限り固有な名前をつけてください。

- 変数名 (*483* ページ)
- ライブラリーの変数名 (*484* ページ)
- ライブラリーのユーザー定義データ型 (DUT) (*485* ページ)
- ファンクション、ファンクションブロック、プログラム (POU) 、アクション (*485* ページ)
- ライブラリーの POU (*485* ページ)

変数名

アプリケーションやライブラリー内での変数の命名については、可能な限りハンガリアン記法に従ってください。

それぞれの変数に短い意味のある名前を付けます。これがベースになる名前として使用されます。ベース名の各単語の文字は大文字にします。残りの文字は小文字を使用してください (例 : FileSize)。

データ型	下限	上限	情報の内容	接頭辞	コメント
BOOL	FALSE	TRUE	1 ビット	x*	–
				b	予約済み
BYTE	–	–	8 ビット	by	ビット列、算術演算用ではない
WORD	–	–	16 ビット	w	ビット列、算術演算用ではない
DWORD	–	–	32 ビット	dw	ビット列、算術演算用ではない
LWORD	–	–	64 ビット	lw	算術演算用ではない
SINT	-128	127	8 ビット	si	–
USINT	0	255	8 ビット	usi	–
INT	-32,768	32,767	16 ビット	i	–
UINT	0	65,535	16 ビット	ui	–
DINT	-2,147,483,648	2,147,483,647	32 ビット	di	–
UDINT	0	4,294,967,295	32 ビット	udi	–
LINT	-2^{63}	$2^{63}-1$	64 ビット	li	–
ULINT	0	$2^{64}-1$	64 ビット	uli	–
REAL	–	–	32 ビット	r	–
LREAL	–	–	64 ビット	lr	–
STRING	–	–	–	s	–
WSTRING	–	–	–	ws	–
TIME	–	–	–	tim	–
TIME_OF_DAY	–	–	–	tod	–
DATE_AND_TIME	–	–	–	dt	–
DATE	–	–	–	date	–
ENUM	–	–	16 ビット	e	–
POINTER	–	–	–	p	–
ARRAY	–	–	–	a	–

* ブール型変数の場合、BYTE 型と区別するため、および IEC プログラマー (%IX0.0 を参照してください) の認識と対応するために、接頭辞として x が選択されます。

簡易宣言

簡易宣言の例

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

入れ子になった宣言

接頭辞が宣言の順に相互に関連付けられている入れ子になった宣言の例

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

ファンクションブロックインスタンスおよびユーザー定義データ型の変数

ファンクションブロックインスタンスおよび、ファンクションブロックまたは接頭辞としてのデータ型名を取得するユーザー定義データ型の変数 (例 : sdo)。

例

```
cansdoReceivedTelegram: CAN_SDOTelegram;
TYPE CAN_SDOTelegram :    (* prefix: sdo *)
STRUCT
  wIndex:WORD;
  bySubIndex:BYTE;
  byLen:BYTE;
  aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

ローカル定数

ローカル定数 (c) は、接頭辞 c とそれに付帯するアンダースコアで始まり、型の接頭辞と変数名が続きます。

例

```
VAR CONSTANT
  c_uiSyncID: UINT := 16#80;
END_VAR
```

グローバル変数とグローバル定数

グローバル変数は接頭辞 g_ で始まり、グローバル定数には接頭辞 gc_ が付きます。

例

```
VAR_GLOBAL
  g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
  gc_dwExample: DWORD;
END_VAR
```

ライブラリーの変数名

構造

基本的に、変数名については上記の説明を参照してください。お使いのアプリケーションコードの変数にアクセスする場合は、ライブラリーの名前空間を接頭辞として使用します。

例

```
g_iTest: INT; (declaration)
CAN.g_iTest (implementation, call in an application program
```

ライブラリーのユーザー定義データ型 (DUT)

構造

各データ型の構造体名は、その構造体を表す短い説明で構成されます (例えば、SDOTelegram)。

例 (名前空間が CAL のライブラリー)

```
TYPE Day :(
MONDAY,
TUESDAY,
WEDNESDAY,
THURSDAY,
FRIDAY,
SATURDAY,
SUNDAY);
```

宣言 :

```
eToday: CAL.Day;
```

アプリケーションでの使用 :

```
IF eToday = CAL.Day.MONDAY THEN
```

注記：ライブラリーで宣言された DUT や列挙型を使用する場合には、名前空間の使用を検討してください。

ファンクション、ファンクションブロック、プログラム (POU) 、アクション

ファンクション、ファンクションブロック、およびプログラムの名前の前には POU を短く表した POU 名が付いています (例えば、SendTelegram)。変数と同様に、POU 名の単語の最初の文字は常に大文字、その他の文字は小文字にします。POU の名前は動詞と名詞で構成することが推奨されています。

例

```
FUNCTION_BLOCK SendTelegram (* prefix: canst *)
```

宣言部分に、POU の簡単な説明をコメントとして記述します。さらに、入力と出力にもコメントを付けます。ファンクションブロックの場合は、名前の直後に設定したインスタンスに関連する接頭辞を挿入します。

アクション

アクションには、接頭辞を付けません。内部でのみ呼び出されるアクション、すなわち POU 自身によって呼び出されるアクションのみが prv_ から始まります。

ライブラリーの POU

構造

メソッド名を作成する場合には、アクションと同じ規則が適応されます。メソッドの利用可能な入力に、英語のコメントを入力します。宣言にメソッドの簡単な説明を追加します。インターフェイス名は、英文字 I で始めます。例えば、ICANDevice。

注記：ライブラリー内で宣言された POU を使用する場合は、名前空間の使用を検討してください。

変数の初期化

デフォルトの初期値

デフォルトの初期値はすべての宣言に対して 0 ですが、各変数およびデータ型の宣言にユーザー定義の初期値を追加することができます。

ユーザー定義の初期値

ユーザー定義の初期値は代入演算子 := によって行われ、有効な任意の ST 式にすることができます。そのため、定数値および他の変数またはファンクションを使用して初期値を定義することができます。別の変数の初期化に使用される変数自体がすでに初期化されていることを確認してください。

有効な変数の初期化の例

```
VAR
var1:INT := 12;             * Integer variable with initial value of 12. *
x : INT := 13 + 8;          * Integer value defined an expression with literal values.*
y : INT := x + fun(4);       * Integer value defined by an expression
containing a function call. NOTE: Be sure that any variables used in
the variable initialization have already been defined. *
z : POINTER TO INT := ADR(y); * POINTER is not described by the IEC61131-3:
Integer value defined by an address function; NOTE: The pointer will
not be initialized if the declaration is modified online. *
END_VAR
```

詳細

詳細については、次の説明を参照してください。

- 配列 (*574* ページ) の初期化
- 構造体 (*577* ページ) の初期化
- subrange 型 (*581* ページ) の初期化

注記：グローバル変数リスト (GVL) の変数は、POU のローカル変数の前に初期化されます。

注記：SoMachine バージョン 4.0 以降、ファンクションブロック内の変数は次の順序で初期化されます。最初に、宣言の順序に従って定数を宣言し、次に宣言の順序に従って他の変数を宣言します。

初期化順序の詳細については、属性 Attribute global_init_slot を参照してください (*528* ページ)。

宣言

宣言型

変数は、テキスト宣言エディターまたは表形式宣言エディター (*342* ページ) を用いて手動で宣言するか、またはこの章で説明するように自動で宣言することもできます。

自動宣言

オプションダイアログボックスで**テキストエディター → 編集**カテゴリーを定義すると、まだ宣言されていない文字列がエディターの実装部分に入力されて ENTER キーが押されると**自動宣言**ダイアログボックスが開きます。このダイアログボックスは変数の宣言 (*481* ページ) を支援します。

手動による自動宣言

自動宣言ダイアログボックスを手動で開くには、以下の処理のいずれかを行います。

- 初期設定では**編集**メニューにある**自動宣言**コマンドを実行します。
- または、SHIFT+F2 キーを押して自動宣言コマンドを実行します。

自動宣言 ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を開く前にすでに宣言されている変数を選択すると、この変数の宣言を編集できます。

ショートカットキーモード

概要

宣言エディター (*342* ページ) および宣言を実行する他のエディターはショートカットキーモードをサポートしています。

宣言行を終了するときに、CTRL+ENTER を押してショートカットキーモードを有効にします。

宣言を完全に入力する代わりにショートカットキーを使用することができます。

対応しているショートカットキー

次のショートカットキーに対応しています。

- 行の最後の識別子までのすべての識別子は、宣言変数識別子になります。
- この宣言型は行の最後の識別子によって決まります。

これに関連して、次の置換が実行されます。

B または BOOL	は、右のように置換されます。	BOOL
I または INT		INT
R または REAL		REAL
S または string		STRING

- これらのルールでは型が確立されない場合は、自動的に BOOL 型になり、最後の識別子は型として識別されません (例 1 を参照してください)。
- 宣言の型に応じて、すべての定数が初期化または文字列に置換されます (例 2 または 3 を参照してください)。
- アドレス (%MD12 など) は AT キーワードによって拡張されます (例 4 を参照してください)。
- セミコロン (;) の後のテキストはコメントになります (例 4 を参照してください)。
- この行の他の文字はすべて無視されます (例えば、例 5 のエクスクラメーションマークを参照してください)。

例

例の番号	ショートカットキー	結果の宣言
1	A	A: BOOL;
2	A B I 2	A, B: INT := 2;
3	sX S 2; A string	sX:STRING(2); // A string
4	X %MD12 R 5; Real Number	X %MD12 R 5 Real Number
5	B !	B: BOOL;

AT 宣言

概要

コントローラー設定 (デバイスエディター) にあるデバイスの **I/O マッピング**ビューで変数にアドレスを割り当て、プロジェクト変数を特定のアドレスにリンクさせることができます。または、このアドレスを変数の宣言に直接入力することもできます。

構文

< 識別子 >AT < アドレス > : < データ型 >

有効なアドレスを、キーワード AT の後に続けます。詳細については、*アドレス*の詳細 (*692* ページ) を参照してください。バイトアドレス指定モードの場合に起こる可能性のあるオーバーラップに注意してください。

この宣言により、意味のある名前をアドレスに割り当てることができます。受信または送信信号に関する変更は、単一の場所 (例えば、宣言内) でのみ行うことができます。

アドレスに割り当てる変数を選択する際には、次の点に注意してください。

- 入力が必要な変数には、書き込みによるアクセスはできません。コンパイラーはこれを検知してエラーを検出します。
- AT 宣言はローカル変数またはグローバル変数でのみ使用できます。POU の入力変数と出力変数では使用できません。
- 保持変数リスト内では、AT 宣言はできません。
- AT 宣言が構造体またはファンクションブロックのメンバーで使用された場合、すべてのインスタンスはこの構造体 / ファンクションブロックと同じメモリーロケーションにアクセスします。これは、C 言語などの古典的なプログラミング言語の静的変数に対応します。
- 構造体のメモリーレイアウトは対象先によっても決定されます。

例

```
xCounterHeat7 AT %QX0.0: BOOL;
xLightCabinetImpulse AT %IX7.2: BOOL;
xDownload AT %MX2.2: BOOL;
```

メモ

ブール型変数が BYTE 、WORD または DWORD アドレスに割り当てられた場合、オフセットの後の最初のビットだけではなく、TRUE または FALSE で 1 バイトを使用します。

入力、出力、およびメモリーデータ (AT %I、%Q、および %M による宣言) のメモリーサイズは、PacDrive コントローラー (PacDrive LMC Eco、PacDrive LMC Pro/Pro2) 用に、対象デバイスによって事前定義されており、アプリケーションオブジェクトのプロパティで上書きできます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

キーワード

概要

エディターにキーワードを大文字で入力してください。

次の文字列はキーワードとして予約されています。これらは変数の識別子または POU の識別子として使用ができません。

- -
- &
- (
-)
- *
- ,
- .
- ..
- /
- :
- :=
- ;
- [
-]
- ^
- __CATCH
- __CHECKLICENSE
- __CHECKLICENSEBIT
- __DELETE
- __ENDTRY
- __FINALLY
- __ISVALIDREF
- __MEMORYBARRIER
- __NEW
- __QUERYINTERFACE
- __QUERYPOINTER
- __THROW
- __TRY
- __XADO
- |
- +
- <
- <=
- <>
- =
- >=
- >
- =>
- ABS
- ACOS
- ADR
- AND
- AND_THEN
- ASIN
- ATAN
- BITADR
- BY
- CASE
- CONTINUE
- COS
- DO
- ELSE
- ELSIF
- END_CASE
- END_FOR

- END_IF
- END_REPEAT
- END_WHILE
- EXIT
- EXP
- EXPT
- FALSE
- FOR
- IF
- INI
- LIMIT
- LN
- LOG
- LOWER_BOUND
- MAX
- MIN
- MOD
- MOVE
- MUX
- NOT
- OR
- OR_ELSE
- R=
- REF=
- REPEAT
- RETURN
- ROL
- ROR
- S=
- SEL
- SHL
- SHR
- SIN
- SIZEOF
- SQRT
- SUPER
- TAN
- THEN
- THIS
- TO
- TRUE
- TRUNC
- TRUNC_INT
- UNTIL
- UPPER_BOUND
- WHILE
- XOR

さらに、**入力アシスタント**にリストされている変換演算子もキーワードとして扱われます。

31.2
変数型

このセクションについて

このセクションには次の項目が含まれています。

変数型

概略

この章では、次の変数型の詳細について説明します。
- VAR ローカル変数 (*493* ページ)
- VAR_INPUT 入力変数 (*493* ページ)
- VAR_OUTPUT 出力変数 (*493* ページ)
- VAR_IN_OUT 入力変数および出力変数 (*494* ページ)
- VAR_GLOBAL グローバル変数 (*494* ページ)
- VAR_TEMP 一時変数 (*494* ページ)
- VAR_STAT 静的変数 (*494* ページ)
- VAR_EXTERNAL 外部変数 (*495* ページ)
- VAR_INST インスタンス変数 (*495* ページ)

ローカル変数 - VAR

キーワード VAR と END_VAR の間で、POU のすべてのローカル変数が 宣言されます (*481* ページ)。これらには外部接続が無く、外部から書き込むことができません。

VAR への属性 (*496* ページ) の追加を検討してください。

例

```
VAR
 iLoc1:INT; (* 1. Local Variable*)
END_VAR
```

入力変数 - VAR_INPUT

キーワード VAR_INPUT と END_VAR の間で、すべての変数が POU の入力変数として機能するように宣言されます (*481* ページ)。これは、呼び出し位置で呼び出しとともに変数の値が与えられることを意味します。

属性 (*496* ページ) の追加を検討してください。

例

```
VAR_INPUT
 iIn1:INT (* 1. Inputvariable*)
END_VAR
```

出力変数 - VAR_OUTPUT

キーワード VAR_OUTPUT と END_VAR の間で、すべての変数が POU の出力変数として機能するように宣言されます。これは、これらの値が呼び出しを行う POU に戻るということを意味します。

VAR_OUTPUT への属性 (*496* ページ) の追加を検討してください。

例

```
VAR_OUTPUT
 iOut1:INT; (* 1. Outputvariable*)
END_VAR
```

ファンクションとメソッドの出力変数 :

IEC 61131-3 ドラフト 2 に従って、ファンクション (およびメソッド) は追加の出力を持つことができます。次の例のように、ファンクションの呼び出しでそれらを割り当てることができます。

例

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

入出力変数 - VAR_IN_OUT

キーワード VAR_IN_OUT と END_VAR の間で、すべての変数が POU の入力変数および出力変数として機能するように宣言されます (*481* ページ)。

注記：IN_OUT 型の変数では、転送された変数の値が変更されます (ポインターとして転送、リファレンスによる呼び出し)。これは、そのような変数の入力値は定数にできないことを意味します。そのため、ファンクションブロックの VAR_IN_OUT 変数は <FBinstance>.<InOutVariable> を介しても外部から直接読み書きできません。

注記：ビット型のシンボル (ビット型のアドレス上にある %MXaa.b または BOOL 変数など) をファンクションブロックの BOOL 型 VAR_IN_OUT パラメーターに割り当てないでください。そのような割り当てが検出された場合、**メッセージ** の **ビルド** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* に検出されたエラーとして表示されます。

例

```
VAR_IN_OUT
 iInOut1:INT; (* 1. Inputoutputvariable *)
END_VAR
```

グローバル変数 - VAR_GLOBAL

通常の変数、定数、またはグローバル変数としてプロジェクト全体で認識されている外部変数または残存変数を宣言できます。グローバル変数を宣言するには、グローバス変数リスト (GVL) を使用します。**オブジェクトの追加** コマンドを実行して GVL を追加できます (初期設定では、**プロジェクト** メニュー内)。

キーワード VAR_GLOBAL と END_VAR の間で、変数をローカルで宣言します。

VAR_GLOBAL への属性 (*496* ページ) の追加を検討してください。

変数は、前のドットによってグローバル変数として認識されます。例えば、.iGlobVar1

変数名の複数使用、グローバルスコープ演算子 dot (.)、および名前空間についての詳細は、*グローバルスコープ演算子* (*673* ページ) の章を参照してください。

グローバル変数は、グローバル変数リスト (GVL) でのみ宣言できます。これらはプロジェクト内のグローバル変数リストを管理します。**オブジェクトの追加** コマンドを実行して GVL を追加できます (初期設定では、**プロジェクト** メニュー内)。

注記：POU で、グローバル変数と同じ名前でローカルに定義された変数は、POU 内で優先されます。

注記：グローバル変数は POU のローカル変数より先に初期化されます。

一時変数 - VAR_TEMP

この機能は、IEC 61131-3 格規の拡張機能です。

一時変数は POU の各呼び出しで (再) 初期化されます。VAR_TEMP 宣言はプログラムおよびファンクションブロック内でのみ可能です。これらの変数は、プログラム POU またはファンクションブロックの本体内でのみアクセス可能です。

キーワード VAR_TEMP と END_VAR の間で、変数をローカルで宣言します。

注記：VAR の代わりに VAR_TEMP を使用して、POU に必要なメモリー領域 (例えば、変数が一時的にのみ使用される場合はファンクションブロック内) を減らすことができます。

静的変数 - VAR_STAT

この機能は、IEC 61131-3 格規の拡張機能です。

静的変数は、ファンクションブロック、メソッド、およびファンクション内で利用できます。キーワード VAR_STAT と END_VAR の間でローカルに宣言します。それらは、各 POU の最初の呼び出しで初期化されます。

グローバル変数のように、静的変数は宣言された POU から抜けた後も値を保持します。それらは宣言された POU 間で共有されます (例えば、複数のファンクションブロックインスタンス、ファンクション、またはメソッドが同じ静的変数を共有)。例えば、ファンクションでファンクション呼び出しをカウントするカウンターとして使用します。

VAR_STAT への属性 (*496* ページ) の追加を検討してください。

外部変数 - VAR_EXTERNAL

POU にインポートされるグローバル変数です。

キーワード VAR_EXTERNAL と END_VAR の間でローカルに、またグローバル変数リスト (GVL) でも宣言されます。この宣言とグローバル変数宣言は同一にしてください。グローバル変数が存在しない場合は、メッセージが表示されます。

注記：変数を外部変数として定義する必要はありません。これらのキーワードによって、IEC 61131-3 との互換性を維持します。

例

```
VAR_EXTERNAL
 iVarExt1:INT; (* 1st external variable *)
END_VAR
```

インスタンス変数 - VAR_INST

VAR_INST 属性を使用してメソッドの変数をインスタンス変数として宣言すると、この変数はメソッドのスタックではなくファンクションブロックインスタンスのスタックに格納されます。そのため、ファンクションブロックインスタンスの他の変数と同じように動作し、メソッドが呼び出されたときに再初期化されません。

VAR_INST 変数はメソッドのみで使用可能です。メソッド内でのみそのような変数にアクセスできます。CONST、RETAIN などの属性は、宣言には使用できません。変数の値は、メソッドの宣言部分で監視できます。

例

```
METHOD meth_last : INT
VAR_INPUT
  iVar : INT;
END_VAR
VAR_INST
  iLast : INT := 0;
END_VAR
meth_last := iLast;
iLast := iVar;
```

変数型の属性キーワード

概要

次の属性キーワードを変数型の宣言 (*481* ページ) に追加して、スコープを指定します。

- RETAIN: *保持変数* (*496* ページ) を参照してください。
- PERSISTENT: *永続変数* (*497* ページ) を参照してください。
- CONSTANT: *定数 - CONSTANT* (*497* ページ) および *型属性付加リテラル* (*498* ページ) を参照してください。

残存変数 - RETAIN 、PERSISTENT

残存変数は、通常のプログラム実行期間を通してその値を保持することができます。変数を保持変数、またはより厳密に永続変数として宣言します。

この宣言によって、リセット、ダウンロード、またはコントローラーの再起動のときの残存変数の保持の程度を決定します。アプリケーションでは、主に両方の残存フラグの組み合わせが使用されます (*永続変数* (*497* ページ) を参照してください)。

注記 : VAR PERSISTENT 宣言は VAR PERSISTENT RETAIN または VAR RETAIN PERSISTENT と同じように解釈されます。

注記 : **すべてのインスタンスパスを追加** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用して、宣言された変数を**永続リスト**オブジェクトとして受け取ります。

保持変数

保持変数として宣言された変数は、不揮発性メモリー領域に格納されます。この種の変数を宣言するには、POU の宣言部分でキーワード RETAIN を使用するか、またはグローバル変数リストを使用します。

例

```
VAR RETAIN
 iRem1 : INT; (* 1. Retain variable*)
END_VAR
```

通常のコントローラーの電源切断 (または **オンライン**コマンドの**ウォームリセット**実行時) と同様に、コントローラーの予期しないシャットダウン後でも、保持変数はその値を保持します。プログラムの再起動後、保持されている値はさらに処理されます。他の (保持変数でない) 変数は初期値、またはデフォルトの初期値 (初期値が宣言されていない場合) で新たに初期化されます。

例えば、生産機械で個数をカウントするなどの停電後も続行する必要がある処理の場合に、保持されている値を使用できます。

ただし、保持変数は**オンラインの元値のリセット**コマンドを実行した場合、再び初期化されます。また永続変数とは対照的に、**オンラインのコールドリセット**コマンド、またはアプリケーションのダウンロードの実行中も再初期化されます。

注記 : VAR RETAIN とし定義された特定の変数のみが不揮発メモリーに格納されます。ただし、VAR RETAIN と定義されたファンクションのローカル変数は不揮発性メモリーに格納されません。ファンクションでローカルに VAR RETAIN を定義しても効果はありません。

保持プログラムセクション (VAR_RETAIN) のシステム設定ライブラリーからインターフェイスまたはファンクションブロックを使用すると、コントローラーの動作不能を引き起こすシステム例外が発生し、再起動が必要になる可能性があります。

⚠ 警告

装置の意図しない動作

- 保持プログラムセクション (VAR_RETAIN) の SystemConfigurationItf ライブラリーからインターフェイスを使用しないでください。
- 保持プログラムセクション (VAR_RETAIN) の SystemConfiguration ライブラリーからファンクションブロックを使用しないでください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記 : SystemConfigurationItf および SystemConfiguration ライブラリーは、PacDrive コントローラー (PacDrive LMC Eco, PacDrive LMC Pro/Pro2) でのみ使用できます。

永続変数

永続変数は、キーワード PERSISTENT (VAR_GLOBAL PERSISTENT) で識別されます。永続変数は**オンライン**コマンドの**元値をリセット**を実行した場合にのみ再初期化されます。保持変数とは対照的に、永続変数はダウンロード後にもその値を保持します。

注記：AT 宣言は VAR PERSISTENT と組み合わせて使用しないでください。

アプリケーション例

停電やダウンロード後もカウントを続行する必要がある稼動時間のカウンター。残存変数の動作に関する概要表 (*497* ページ) を参照してください。

永続変数はオブジェクトタイプの永続変数の特別なグローバルリスト内でのみ宣言でき、このグローバルリストはアプリケーションに割り当てられています。このようなリストは、アプリケーションに 1 つだけ追加できます。

注記：VAR_GLOBAL PERSISTENT の宣言は、VAR_GLOBAL PERSISTENT RETAIN または VAR_GLOBAL RETAIN PERSISTENT 宣言と同じ効果があります。

保持変数のように、永続変数は別のメモリー領域に格納されます。

例

```
VAR_GLOBAL PERSISTENT RETAIN
  iVarPers1 : DINT; (* 1. Persistent+Retain Variable App1 *)
  bVarPers : BOOL;  (* 2. Persistent+Retain Variable App1 *)
END_VAR
```

注記：永続変数は**永続変数リスト**オブジェクト内でのみ宣言できます。永続変数が他の場所で宣言されていると保持変数のように動作し、**メッセージ**ビューに検出された**ビルド**エラーとして表示されます (保持変数は、グローバル変数リスト内または POU 内で宣言できます)。

アプリケーションの再読み込みのたびに、コントローラーの永続変数リストがプロジェクトの永続変数リストと照合されます。コントローラー上のリストはアプリケーションで識別されます。一致していない場合は、アプリケーションのすべての永続変数 *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を再初期化するよう求められます。リスト内の既存の宣言の名前の変更や削除、またはその他の変更によって不一致が生じる場合があります。

注記：永続変数リストの宣言部分の変更と、再初期化の結果による影響を慎重に考慮してください。

新しい宣言は、リストの最後にのみ追加できます。ダウンロード中、これらは新しい宣言として検出され、完全なリストの再初期化は必要ありません。変数の名前またはデータ型を変更すると、これは新しい宣言として処理され、次のオンライン変更またはダウンロードの際に変数が再初期化されます。

残存変数の動作

残存変数の動作についての詳細は 各コントローラーについては、*プログラミングガイド*を参照してください。

定数 - CONSTANT

定数はキーワード CONSTANT によって識別されます。定数はローカルまたはグローバルに宣言できます。

構文

```
VAR CONSTANT<identifier>:<type> := <initialization>;END_VAR
```

例

```
VAR CONSTANT
 c_iCon1:INT:=12; (* 1. Constant*)
END_VAR
```

可能な定数のリストについては、*オペランド*の章 (*677* ページ) を参照してください。

属性付加リテラル

基本的には、IEC 定数を使用すると可能な限り小さいデータ型が使用されます。別のデータ型を使用する必要がある場合は、定数を明示的に宣言する必要なしに型属性付加リテラルによって実現できます。そのために、定数は型を決定する接頭辞付きになります。

構文

<type>#<literal>;

<type>	データ型を指定します。 入力可能なエントリー : BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL データ型は大文字で書きます。
<literal>	定数を指定します。 <type> で指定したデータ型に適合するデータを入力します。

例

iVar1:=DINT#34;

データの損失なしに定数を対象の型に変換できない場合は、メッセージが発行されます。

通常の定数が使用できる場所であれば型属性付加リテラルを使用できます。

オンラインモードの定数

初期設定 **Replace constants** (**ファイル → プロジェクト設定 → コンパイルオプション**) が有効である限り、オンラインモードの宣言ビューまたはウォッチビューの**値**の列にある定数の値の前に記号  が付きます。この場合、強制や書き込みなどではアクセスできません。

変数設定 - VAR_CONFIG

概要

デバイス I/O 上の処理イメージのファンクションブロック変数をマップするには、変数設定を使用します。これにより、すでにファンクションブロック変数の宣言にあるアドレスを明確に指定する必要がなくなります。この場合、グローバルな VAR_CONFIG リストにあるすべてのファンクションブロックインスタンスに明確なアドレス (*692* ページ) が一元的に割り当てられます。

そのため、キーワード VAR と END_VAR の間に宣言されたファンクションブロック変数に不完全なアドレスを割り当てることができます。これらのアドレスを識別するためにアスタリスクを使用します。

識別子の構文

< 識別子 >AT %<I|Q>* : < データ型 >

定義された不完全なアドレスの使用例

```
FUNCTION_BLOCK locio
VAR
 xLocIn AT %I*: BOOL := TRUE;
 xLocOut AT %Q*: BOOL;
END_VAR
```

この例では 2 つのローカル I/O 変数、ローカル入力変数 (%I*) およびローカル出力変数 (%Q*) が定義されています。

次のように、グローバル変数リスト (GVL) の変数設定でアドレスを定義します。

手順	手順内容
1	**オブジェクトの追加**コマンドを実行します。
2	**デバイスツリー**に**グローバル変数リスト** (GVL) オブジェクトを追加します。
3	キーワード VAR_CONFIG と END_VAR の間に明確なアドレスでインスタンス変数の宣言を入力します。

アドレスを定義するときは、次の点に注意してください。

- インスタンス変数はインスタンスの絶対パスで指定し、各 POU とインスタンス名はピリオドで区切ります。
- 宣言には、ファンクションブロックで指定された不完全なアドレス (%I*、%Q*) のクラス (入力 / 出力) に対応するクラスのアドレスを入力します。
- データ型がファンクションブロックの宣言と一致していることを確認します。

インスタンス変数パスの構文

< インスタンス変数パス > AT %<I|Q><location> :

インスタンスが存在しないためにインスタンスパスが無効である変数を設定すると、エラーが検出されたとして表示されます。不完全なアドレスに割り当てられたインスタンス変数に明確なアドレス設定が存在しない場合にも、エラーが検出されます。

変数設定の例

プログラムでファンクションブロック locio (前の例を参照) が次のように定義がされたとします。

```
PROGRAM PLC_PRG
VAR
locioVar1: locio;
locioVar2: locio;
END_VAR
```

グローバル変数リスト内の修正された変数構成は次のようになります。

```
VAR_CONFIG
PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;
PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

注記：直接マップされた I/O の変更はすぐに処理イメージに表示されるのに対して、VAR_CONFIG によってマップされた変数の変更はそのタスクが終了するまで表示されません。

注記：グローバル変数リストでは、キーワード VAR_GLOBAL および VAR_CONFIG は排他的にのみ使用できます。

31.3
メソッド型

FB_Init、FB_Reinit、および FB_Exit メソッド

メソッドの一般的な目的

メソッド FB_Init および FB_Reinit を明示的に使用して、ファンクションブロック変数の初期化およびファンクションブロック終了時の動作を操作することができます。

この章では、メソッド、および変数の初期化を必要とするさまざまな条件におけるメソッドの適用と効果について説明します。

FB_Init

デフォルトでは、FB_Init メソッドは暗黙的に使用可能です。これは、EcoStruxure Machine Expert がファンクションブロックまたは構造体を初期化するために使用されます。

初期化を操作するために、与えられたデフォルトの初期化コードを拡張して FB_Init メソッドを明示的に宣言することもできます。これにより、戻り値を評価することができます。

FB_Reinit

FB_Reinit メソッドは明示的に宣言する必要があります。

FB_Reinit メソッドが使用可能な場合は、対応するファンクションブロックのインスタンスがコピーされた後 (ファンクションブロック宣言の変更後のオンライン変更中 (*501* ページ)) に呼び出されます。新しいインスタンスモジュールを再初期化します。戻り値は評価されません。元のファンクションブロックを再初期化するためには、そのファンクションブロックに対して明示的に FB_Reinit を呼び出します。これにより、戻り値を評価することができます。

FB_Exit

FB_Exit メソッドは明示的に宣言する必要があります。

実装がある場合、コントローラーがファンクションブロックのインスタンスのコードを削除する前にメソッドが呼び出されます (暗黙的な呼び出し)。戻り値は評価されません。

次の段落では、さまざまな動作条件でのメソッドの使用例を示します。

最初のダウンロード

デフォルト状態のコントローラーにアプリケーションをダウンロードすると、変数のメモリーロケーションは必要に応じた初期状態に設定されます。ファンクションブロックのデータ領域は必要に応じた値に設定されています。アプリケーションのプログラムコードでファンクションブロックの FB_Init を明示的に実装することで、このプロセスを操作することができます。

メソッドパラメーター bInCopyCode が FALSE、bInitRetains が TRUE に設定されている場合、最初のダウンロードが実行されていることを示します。

オンライン変更

オンライン変更 コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を実行すると、メソッド FB_Exit、FB_Init、および FB_Reinit を使用して、ファンクションブロックの初期化を操作することができます。

オンライン変更中、オフラインモードでアプリケーションに加えられた変更はコントローラーにダウンロードされます。ファンクションブロックのインスタンスは、新しいインスタンスによって次のように更新されます。

宣言部分ではなく、ファンクションブロックの実装部分にのみ変更を加えた場合、データ領域は置き換えられません。メソッド FB_Init、FB_Reinit、および FB_Exit は呼び出されません。

ファンクションブロックの宣言部に変更を加えた場合は、**オンライン変更** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* が実行されたときに、FB_Reinit パラグラフ (*501* ページ) に記載されているコピー処理が実行されます。前回のダウンロード以降に変更されたオブジェクトのリストが **Application Information** ダイアログボックスに表示されます (*205* ページ)。このダイアログボックスを開くには、ダイアログボックスの **詳細 ...** ボタンをクリックし、**オンライン変更してログイン**オプションを選択します。

FB_Init および FB_Reinit メソッドのパラメーター bInCopyCode が TRUE、bInitRetains が FALSE に設定されている場合、オンライン変更が実行されていることを示します。

オンライン変更中に実行される呼び出し

オンライン変更コマンドを実行することによって、アドレスの内容を変更できます。

⚠ 注意

無効なポインター

アドレスにポインターを使用し、オンライン変更コマンドを実行するときにはポインターの有効性を確認してください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

オンライン変更中は、次の呼び出しが連続して実行されます。

手順	手順内容	コメント
1	FB_Exit	old_inst.FB_Exit(bInCopyCode := TRUE); コピー処理が始まる前に、クリーンアップ処理を開始するために FB_Exit が呼び出されます。 次のコピー処理のためにデータを準備し、新しいインスタンスの状態に影響を与えます。 アプリケーションの他の部分に、メモリー内で実行される位置の変更について通知されます。 POINTER 型、または REFERENCE 型の変数は、オンラインでの変更中も値を保持し、プロセスが完了した後で必要なメモリーロケーションを参照しなくなる可能性があることに注意してください。 INTERFACE 型の変数はオンライン変更中に適応されます。ソケット、ファイル、または他のハンドルなどの外部リソースは、新しいインスタンスによって採用される可能性があります。システムリソースがオンライン変更のコピー処理の影響を受けない場合、オンライン変更中に個別の処理を必要としないことが多いです。必要ならば、これは Init または Reinit の実装で処理されなければなりません。
2	FB_Init	new_inst.FB_Init(bInitRetains := FALSE, bInCopyCode := TRUE); FB_Init は、オンライン変更プロセス中に特定の操作を実行するために使用できます。 これらは、例えば、新しいメモリーロケーションで変数を適切に初期化したり、特定の変数の新しい位置に関する情報をアプリケーションの他の部分に提供したりします。
3	コピー操作 copy	copy(&old_inst, &new_inst); 既存の値は変わりません。その目的により、古いインスタンスから新しいインスタンスにコピーされます。
4	FB_Reinit	new_inst.FB_Reinit(); FB_Reinit メソッドがコピー操作の後に呼び出されます。ファンクションブロックのインスタンスの変数を定義済みの値に設定します。 例えば、新しいメモリーロケーションで変数を適切に初期化したり、特定の変数の新しい位置に関する情報をアプリケーションの他の部分に提供したりできます。 ファンクションブロックが元の状態にリセットされるときは、アプリケーションからいつでもメソッドを呼び出すことができるため、FB_Reinit メソッドをオンラインでの変更とは無関係に実装します。

注記：ファンクションブロックの変数に pragma (*542* ページ) {attribute no_copy} を追加した場合、この変数はオンライン変更中にコピーされません。初期化のみされます。

更新されたアプリケーションのダウンロード

すでにアプリケーションを実行しているコントローラーにアプリケーションをダウンロードすると、既存のアプリケーションが置き換えられます。FB_Exit メソッドを使用して、例えば、定義済みの状態を外部リソース (ソケットやファイルハンドルなど) に割り当てることができます。

メソッドパラメーター bInCopyCode が FALSE、bInitRetains が FALSE に設定されている場合、更新されたアプリケーションのダウンロードが実行されていることを示します。

アプリケーションの起動

アプリケーションのタスクの最初のサイクルが実行される前に、初期割り当てが処理されます。

例 :

```
T1 : TON := (PT:=t#500ms);
```

割り当ては FB_Init 呼び出された後に実行されます。これらの割り当ての影響を検証できるようにするには、{attribute call_after_init} pragma (*519* ページ) をファンクションブロックとファンクションブロックのメソッド (例えば、呼び出された MyInit) に追加します。この属性を、ファンクションブロックの宣言部および対応するメソッドの宣言部の上に挿入します。この属性を {attribute call_after_init} pragma を使用している他のファンクションブロックを拡張するファンクションブロックにも付加します。対応するメソッドに同じ名前、同じシグニチャー、および同じ属性を割り当てることを推奨します。これを実現するには、SUPER^.MyInit を呼び出します。選択したメソッド名 (FB_Init、FB_Reinit、および FB_Exit を除く) を選択します。このメソッドは、初期割り当てが処理された後、アプリケーションのタスクが開始される前に呼び出されます。

注記 : 明示的に定義された初期化コードが実行されると、ファンクションブロックは暗黙の初期化コードによってすでに完全に初期化されています。このため、SUPER^.FB_Init の呼び出しは許可されていません。

FB_Init メソッドのインターフェイス

```
METHOD FB_Init : BOOL
VAR_INPUT
  bInitRetains : BOOL; // TRUE: the retain variables are initialized (reset warm /reset cold)
  bInCopyCode : BOOL; // TRUE the instance will be copied to the copy-code afterward (online change)
END_VAR
```

戻り値は使用されません。

FB_Init メソッドでは、追加のファンクションブロック入力を宣言できます。入力をファンクションブロックインスタンスの宣言で割り当てます。

例 : serialdevice ファンクションブロック用 FB_Init メソッド

```
METHOD PUBLIC FB_Init : BOOL
VAR_INPUT
   bInitRetains : BOOL; // Initialization of the retain variables
   bInCopyCode : BOOL; // Instance is copied to copy-code
   iCOMnum : INT; // additional input: number of the COM interface that is to be observed
END_VAR
```

serialdevice ファンクションブロックのインスタンス化

```
   com1: serialdevice (iCOMnum:=1);
   com0: serialdevice (iCOMnum:=0);
```

FB_Reinit メソッドのインターフェイス

```
METHOD FB_Reinit : BOOL
```

FB_Exit メソッドのインターフェイス

bInCopyCode. パラメーターは必須です。

```
METHOD FB_Exit : BOOL
VAR_INPUT
bInCopyCode : BOOL; // TRUE: the exit method is called in order to leave the instance which will be copied afterwards (online change).
END_VAR
```

派生ファンクションブロック

ファンクションブロックが別のファンクションブロックから派生している場合、派生ファンクションブロックの FB_Init は、ベースファンクションブロックの FB_Init メソッドと同じパラメーターを定義する必要があります。ただし、インスタンスの特別な初期化を実装するためにさらにパラメーターを追加することができます。

FB_Exit および FB_Init の派生ファンクションブロックの呼び出し順序の例

このリストで指定されている POU について、次のことが想定されています。SubFB EXTENDS MainFB および SubSubFB EXTENDS SubFB:

手順	手順内容
1	fbSubSubFb.FB_Exit(...);
2	fbSubFb.FB_Exit(...);
3	fbMainFb.FB_Exit(...);
4	fbMainFb.FB_Init(...);
5	fbSubFb.FB_Init(...);
6	fbSubSubFb.FB_Init(...);

31.4
Pragma 指令

このセクションについて

このセクションには次の項目が含まれています。

Pragma 指令

概要

pragma 指令は、コンパイルまたはプリコンパイル (プリプロセッサー) 処理に関する変数のプロパティに影響を与えるために使用します。これは、pragma がコード生成に影響することを意味します。

注記：使用できる pragmas は、C プリプロセッサー指令の 1:1 実装ではないことを考慮してください。この指令は通常の命令文として処理されるため、命令文の位置でのみ使用できます。式の中、またエディターの宣言部の中で使用できません。

pragma によって変数を初期化するか、監視、パラメーターリストに追加、シンボルリスト (*555* ページ) に追加、または**ライブラリーマネージャー**に表示させるかを決定できます。ビルド処理中にメッセージの出力を強制できます。条件付き pragmas を使用することで、特定の条件に応じて変数の処理方法を変えて定義もできます。これらの pragmas を特定のオブジェクトのコンパイルプロパティの定義として入力することもできます。

pragma は、実装または宣言エディター行の個別の行に、または補足的テキストと共に使用できます。FBD/LD/IL エディターでは、**Insert Label** コマンドを実行して、pragma によってできたテキストフィールドのデフォルテキスト Label: を置き換えます。pragma だけでなくラベルも設定する場合は、最初に pragma を挿入してからラベルを挿入します。

pragma 指令は中括弧で囲みます。

構文

{ <instruction text> }

変数名のすぐ後ろに開き括弧を置くことができます。開き括弧と閉じ括弧は同じ行内にしてください。

条件付 pragma の正しい位置

```
{IF defined(abc)}
IF x =abc THEN
  {IF defined(cde)}
    y := 12;
  {ELSE}
    y :=13;
  {END_IF}
END_IF
{ELSE}
IF x = 12 THEN
  {IF defined(cde)}
    y := 12;
  {ELSE}
    y :=13;
  {END_IF}
END_IF
```

条件付 pragma の間違った位置

注記：この負の例で示されている位置には条件付き pragma を**使用しないでください**。

```
{IF defined(abc)}
IF x = abc THEN
{ELSE}
IF x = 12 THEN
{END_IF}
y := {IF defined(cde)} 12; {ELSE} 13; {END_IF}
END_IF
```

詳細

pragma のタイプおよび内容に応じて、pragma は次の条件のいずれかが満たされるまで、後に続くすべての命令文を順番に処理します。

- 適切な pragma によって終了
- 同じ pragma が異なるパラメーターで実行
- コードの最後に到達

ここでのコードという用語は、宣言部、実装部、グローバル変数リスト、または宣言型を指します。

注記：Pragma 指令は、大文字と小文字を区別します。

コンパイラーが命令文の意味を解釈できない場合、pragma はコメントとして処理され、スキップされます。

以下の pragma タイプを参照してください。

- *Pragmas メッセージ* (*508* ページ)
- *旧属性* (*550* ページ)
- *Pragmas 属性* (*516* ページ)
- *条件付き Pragmas* (*509* ページ)
- *領域 Pragmas* (*514* ページ)
- *シンボル属性* (*555* ページ)

メッセージ Pragmas

概要

メッセージ pragmas は、プロジェクトのコンパイル (ビルド) 中に**メッセージ**ビュー (初期設定では**編集**メニュー内) へのメッセージの出力を強制します。

POU のテキストエディターの既存の行または別の行に pragma 指令を挿入できます。プロジェクトのコンパイル時、実装コードの現在定義されていない領域にあるメッセージ pragmas は考慮されません。詳細については、*条件付き Pragmas* (*509* ページ) の章の定義済み (識別子) の説明の例を参照してください。

メッセージ Pragmas のタイプ

メッセージ pragmas は 4 種類あります。

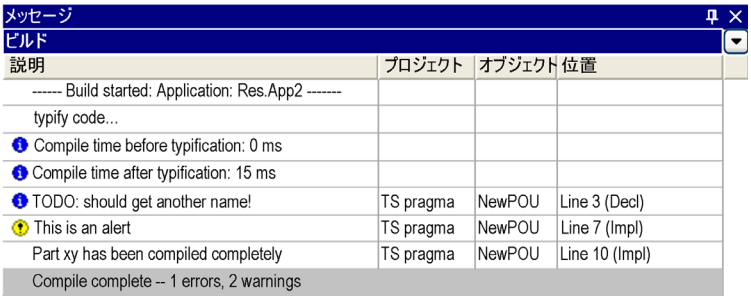
Pragma	アイコン	メッセージタイプ
{text 'text string'}	–	テキストタイプ 指定した文字列が表示されます。
{info 'text string'}		情報 指定した文字列が表示されます。
{warning digit 'text string'}		警告タイプ 指定した文字列が表示されます。 グローバルな旧 pragma (*550* ページ) とは異なり、警告はローカルで明示的に定義されます。
{error 'text string'}		エラータイプ 指定した文字列が表示されます。

注記：タイプ情報、警告、および検出されたエラーのメッセージの場合、**Next Message** コマンドを実行することでメッセージのソース位置、すなわち POU 内の pragma の位置に到達できます。これはテキスト型ではできません。

ST エディターの宣言と実装の例

```
VAR
ivar : INT; {info 'TODO: should get another name'}
bvar : BOOL;
arrTest : ARRAY [0..10] OF INT;
i:INT;
END_VAR
arrTest[i] := arrTest[i]+1;
ivar:=ivar+1;
{warning 'This is an alert'}
{text 'Part xy has been compiled completely'}
```

メッセージビューの出力



条件付き Pragmas

概要

ExST (拡張 ST) 言語は、プリコンパイルまたはコンパイル処理でコード生成に影響を与える条件付き *Pragma 指令* (*506* ページ) に対応しています。

注記：宣言部には条件付き pragma を使用しないでください。それらは無視されます。

コンパイルの対象としてみなされる実装コードは、次の条件によって異なります。
- 特定のデータ型または変数が宣言されているか。
- 型または変数に特定の属性があるか。
- 変数には特定のデータ型があるか。
- 特定の POU またはタスクが使用可能であるか、または呼び出しツリーの一部であるか、など ...

注記：POU ツリーで宣言された POU または GVL では、アプリケーションで宣言された {define...} は使用できません。アプリケーションの定義は、各アプリケーションの下に挿入されたインターフェイスにのみ影響します。

{define identifier string}	前処理の間、トークン文字列が空でない (許可され明確に定義されている) 場合、後に続く識別子のインスタンスはすべて指定したトークンのシーケンスに置き換えられます。識別子はオブジェクトの終わり、または {undefine} 指令で未定義になるまで定義されたままスコープ内に残ります。条件付きコンパイル (*509* ページ) で使用されます。
{undefine identifier}	identifier のプリプロセッサー定義 ({define} により定義、この表の 1 行目を参照) が削除されます。そのため識別子は未定義です。指定した識別子が現在定義されていない場合、この pragma は無視されます。
{IF expr} ... {ELSIF expr} ... {ELSE} ... {END_IF}	これらは、条件付きコンパイル用の pragmas です。指定した式 exprs がコンパイル時に定数である必要がある場合、式の内の 1 つが 0 以外の値に評価されるまで表示される順序で評価されます。正常に完了した指令に関連付けられた文字列は前処理され、通常通りコンパイルされます。その他は無視されます。各セクションの順序は決められています。ただし、elsif および else 部はオプションです。また、elsif 部は任意で複数記述できます。 定数 expr 内には、複数の条件付きコンパイル演算子 (*509* ページ) を使用できます。
<expr>	条件付きコンパイル pragma ({if} または {elsif}) (前の表を参照) の定数式 expr 内では、複数の演算子を使用できます。これらの演算子は {undefine} または {define} で再定義されていない場合があります。

これらの式は**コンパイラー定義**で {define} によって定義された定義と同じように使用できます。オブジェクトの**プロパティ**ダイアログボックス (**表示 → プロパティ → ビルド**) のテキストフィールド。

条件付きコンパイル演算子

次の演算子に対応しています。
- defined (identifier) (*510* ページ)
- defined (variable:variable) (*510* ページ)
- defined (type:identifier) (*510* ページ)
- defined (pou:pou-name) (*510* ページ)
- hasattribute (pou: pou-name, attribute) (*511* ページ)
- hasattribute (variable: variable, attribute) (*511* ページ)
- hastype (variable:variable, type-spec) (*512* ページ)
- hasvalue (define-ident, char-string) (*512* ページ)
- NOT operator (*513* ページ)

- AND operator（*513* ページ）
- OR operator（*513* ページ）
- operator（*513* ページ）

defined (identifier)

この演算子は、識別子が {define} 命令で定義され、その後 {undefine} 命令で未定義にされていない場合に式の値を TRUE にします。それ以外の場合、値は FALSE です。

defined（identifier）**の例**：

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。識別子 pdef1 は App2 で定義されていますが、App1 では定義されていません。

```
{IF defined (pdef1)}
   (* this code is processed in App1 *)
   {info 'pdef1 defined'}
   hugo := hugo + SINT#1;
{ELSE}
   (* the following code is only processed in application App2 *)
   {info 'pdef1 not defined'}
   hugo := hugo - SINT#1;
{END_IF}
```

さらに、メッセージ pragma の例（*508* ページ）も含まれています。

pdef1 が定義されているためアプリケーションがコンパイルされると、pdef1 defined の情報のみが**メッセージ**ビューに表示されます。pdef1 が定義されていないと、**pdef1 not defined** というメッセージが表示されます。

defined (variable:variable)

変数に適用すると、この特定の変数が現在のスコープ内で宣言されている場合は値が TRUE になります。それ以外の場合は、FALSE です。

defined（variable:variable）**の例**：

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。変数 g_bTest は App2 で宣言されていますが、App1 では宣言されていません。

```
{IF defined (variable:g_bTest)}
(* the following code is only processed in application App2 *)
g bTest := x > 300;
{END_IF}
```

defined (type:identifier)

型の識別子に適用すると、その特定の名前の型が宣言されている場合は値が TRUE になります。それ以外の場合は、FALSE です。

defined（type:identifier） **の例**：

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。データ型 DUT は App2 で定義されていますが、App1 では定義されていません。

```
{IF defined (type:DUT)}
(* the following code is only processed in application App1 *)
bDutDefined := TRUE;
{END_IF}
```

defined (pou:pou-name)

POU 名に適用すると、その特定の POU 名の POU またはアクションが定義されている場合は値が TRUE になります。それ以外の場合は、FALSE です。

defined（pou: pou-name）**の例**：

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。POU CheckBounds は App2 では使用可能ですが、App1 では使用できません。

```
{IF defined (pou:CheckBounds)}
(* the following code is only processed in application App1 *)
arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
{ELSE}
(* the following code is only processed in application App2 *)
arrTest[i] := arrTest[i]+1;
{END_IF}
```

hasattribute (pou: pou-name, attribute)

POU に適用すると、この特定の attribute が POU の宣言部の 1 行目に指定されている場合は値が TRUE になります。

hasattribute (pou: pou-name, attribute) **の例**:

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。ファンクション fun1 は App1 および App2 で宣言されていますが、App1 にのみ vision 属性があります。

App1 で fun1 を定義 :

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

App2 で fun1 を定義 :

```
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

Pragma 指令

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in application App1 *)
ergvar := fun1 ivar);
{END_IF}
```

hasattribute (variable: variable, attribute)

変数に適用すると、この特定の属性が変数の宣言の前の行で {attribute} 命令によって指定されてる場合は値が TRUE になります。

hasattribute (variable: variable, attribute) **の例**:

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。変数 g_globalInt は App1 および App2 で使用されていますが、App1 にのみ DoCount 属性があります。

App1 で g_globalInt を定義

```
VAR_GLOBAL
{attribute 'DoCount'}
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

App2 で g_globalInt を定義

```
VAR_GLOBAL
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

Pragma 指令

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
(* the following code line will only be processed in App1, because there variable g_globalInt has got the attribute 'DoCount' *)
g_globalInt := g_globalInt + 1;
{END_IF}
```

hastype (variable:variable, type-spec)

変数に適用すると、この特定の変数に指定された type-spec がある場合は値が TRUE になります。それ以外の場合は、FALSE です。

type-spec に使用できるデータ型

- LREAL
- REAL
- LINT
- DINT
- INT
- SINT
- ULINT
- UDINT
- UINT
- USINT
- TIME
- LWORD
- DWORD
- WORD
- BYTE
- BOOL
- STRING
- WSTRING
- DATE_AND_TIME
- DATE
- TIME_OF_DAY

演算子 hastype (variable: variable, type-spec) **の例**:

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。変数 g_multitype は、App1 では LREAL 型、App2 では STRING 型で宣言されています。

```
{IF (hastype (variable: g_multitype, LREAL))}
(* the following code line will be processed only in App1 *)
g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
(* the following code line will be processed only in App2 *)
g_multitype := 'this is a multitalent';
{END_IF}
```

hasvalue (define-ident, char-string)

定義 (define-ident) が定義され、指定された値 (char-string) がある場合、その値は TRUE です。それ以外の場合は、FALSE です。

hasvalue (define-ident, char-string) **の例**:

前提条件 : 変数 test はアプリケーション App1 および App2 で使用されています。App1 では値 1 を取得し、App2 では値 2 を取得します。

```
{IF hasvalue(test,'1')}
(* the following code line will be processed in App1, because there variable test has value 1 *)
x := x + 1;
{ELSIF hasvalue(test,'2')}
(* the following code line will be processed in App1, because there variable test has value 2 *)
x := x + 2;
{END_IF}
```

NOT operator

operator の逆の値が TRUE の場合にこの式の値が TRUE になります。operator は、この章で説明されている演算子のいずれかにすることができます。

NOT operator **の例**:

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。POU PLC_PRG1 は、App1 および App2 で使用されます。POU CheckBounds は、App1 でのみ使用できます。

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
(* the following code line is only executed in App2 *)
bANDNotTest := TRUE;
{END_IF}
```

AND operator

両方の演算子が TRUE の場合にこの式の値が TRUE になります。operator は、この表のリストにある演算子のいずれかにすることができます。

AND operator **の例**:

前提条件 :App1 および App2 の 2 つのアプリケーションがあります。POU PLC_PRG1 は App1 および App2 で使用されます。POU CheckBounds は、App1 でのみ使用できます。

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
(* the following code line will be processed only in applications App1, because only there "PLC_PRG1" a
nd "CheckBounds" are defined *)
bORTest := TRUE;
{END_IF}
```

OR operator

演算子内の 1 つが TRUE の場合、式は TRUE になります。operator は、この章で説明されている演算子のいずれかにすることができます。

OR operator **の例**:

前提条件 : POU PLC_PRG1 はアプリケーション App1 および App2 で使用されています。POU CheckBounds は、App1 でのみ使用できます。

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
(* the following code line will be processed in applications App1 and App2, because both contain at least
 one of the POUs "PLC_PRG1" and "CheckBounds" *)
bORTest := TRUE;
{END_IF}
```

(operator)

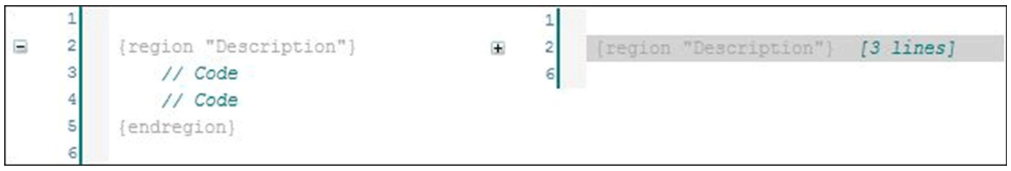
(operator) 演算子を中括弧で囲みます。

領域 Pragmas

概要

テキストエディターで複数の行を 1 つのブロックにまとめるには、領域 pragmas を使用します。ブロックに名前をつけることができます。領域 pragmas はネストができます。

この図は、拡張ビューと縮小ビューに領域 pragma を含むプログラムコードを示しています。



領域 pragmas は、ST エディターおよび宣言エディター使用できます。**ツール → オプション → Syntax Highlighting** ダイアログボックスで、構文の強調表示を個々の要件に合わせることができます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

31.5
属性 Pragmas

このセクションについて

このセクションには次の項目が含まれています。

属性 Pragmas

概要

コードを生成するコンパイルまたはプリコンパイルに影響させるために、属性 pragmas (*506* ページ) をシグネチャーに割り当てることができます。

条件付き pragmas (*509* ページ) と組み合わせて使用できるユーザー定義属性 (*517* ページ) もあります。

属性は宣言部の中で定義されます。宣言部を持たない動作オブジェクトおよび遷移オブジェクトに対しては例外が発生します。実装部分の先頭で属性を定義できます。

また、以下の定義済み標準属性 pragmas もあり ます。

- attribute displaymode (*525* ページ)
- attribute ExpandFully (*527* ページ)
- attribute global_init_slot (*528* ページ)
- attribute hide (*530* ページ)
- attribute hide_all_locals (*531* ページ)
- attribute initialize_on_call (*532* ページ)
- attribute init_namespace (*533* ページ)
- attribute init_On_Onlchange (*533* ページ)
- attribute instance-path (*534* ページ)
- attribute linkalways (*535* ページ)
- attribute monitoring (*536* ページ)
- attribute no_check (*541* ページ)
- attribute no_copy (*542* ページ)
- attribute no-exit (*543* ページ)
- attribute noinit (*544* ページ)
- attribute no_virtual_actions (*546* ページ)
- attribute obsolete (*550* ページ)
- attribute pack_mode (*551* ページ)
- attribute qualified_only (*552* ページ)
- attribute reflection (*553* ページ)
- attribute subsequent (*554* ページ)
- attribute symbol (*555* ページ)
- attribute warning disable (*556* ページ)

ユーザー定義属性

概要

任意のユーザー定義属性 pragma またはアプリケーション定義属性 pragmas を、POU、型の宣言、または変数に割り当てることができます。条件付 pragmas (*509* ページ) を使用してコンパイルする前に属性を照会できます。

構文

{attribute 'attribute'}

この pragma 指令は、後に続く POU 宣言または変数宣言に対して有効です。

ユーザー定義属性を以下に割り当てることができます。

- POU またはアクション
- 変数
- データ型

POU およびアクションの例

ファンクション fun1 の属性 vision

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

変数の例

変数 ivar の属性 DoCount

```
PROGRAM PLC_PRG
VAR
{attribute 'DoCount'};
ivar:INT;
bvar:BOOL;
END_VAR
```

型の例

データ型 DUT_1 の属性 aType

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
a:INT;
b:BOOL;
END_STRUCT
END_TYPE
```

条件付き Pragmas の使い方については、*条件付き Pragmas* (*509* ページ) の章を参照してください。

Attribute call_after_global_init_slot

概要

宣言部分の上の行にこの属性を含むすべてのファンクションとプログラムは、グローバルの初期化 (GlobalInit) の後に呼び出されます。呼び出し順序は属性値によって決まります。

注記：この属性を含むファンクションまたはメソッドで VAR_INPUT 宣言が使用されていると、コンパイルエラーが (コード生成中に) 検出されます。その理由は、オンライン変更中にファンクションが暗黙的に呼び出されたときに、入力変数が不明なためです。

構文

{attribute 'call_after_global_init_slot' := '<slot>'}

<slot> を呼び出しシーケンス内の優先順位を定義する整数値に置き換えます。値が小さいほど、呼び出し優先順位は高くなります。属性に同じ値を持つシグネチャーがいくつかある場合、それらの初期化の順序は未定義のままです。

メソッドに属性が指定されている場合は、該当するファンクションブロックのすべてのインスタンスに対して呼び出されます。すべてのインスタンスは指定されたスロット内で呼び出されます。ただし、インスタンス自体の順序を制御することはできません。

Attribute call_after_init

概要

pragma {attribute call_after_init} を使用して、ファンクションブロックインスタンスの初期化後に暗黙的に呼び出されるメソッドを定義します。性能に関する理由から、属性をファンクションブロック自体と呼び出すインスタンスメソッドの両方に設定します。このメソッドは、FB_Init (*503* ページ) の後およびインスタンス宣言に初期化式の変数値を適用した後に呼び出されます。

注記：この属性を含むメソッドで VAR_INPUT 宣言が使用されていると、コンパイルエラーが検出されます。その理由は、オンライン変更中にメソッドが暗黙的に呼び出されたときに、入力変数が不明なためです。

構文

```
{attribute 'call_after_init'}
```

例

定義は以下のとおりです。

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <functionblock definition>
{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

宣言は以下のようになります。

```
inst : FB := (in1 := 99);
```

コード処理の順序は次のようになります。

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

よって、FB_Afterinit でユーザー定義の初期化を操作できます。

Attribute call_after_online_change_slot

概要

宣言部分の上の行にこの属性を含むすべてのファンクションとプログラムは、オンライン変更の後に呼び出されます。呼び出し順序は属性値によって決まります。

注記：この属性を含むファンクションまたはメソッドで VAR_INPUT 宣言が使用されていると、コンパイルエラーが (コード生成中に) 検出されます。その理由は、オンライン変更中にファンクションが暗黙的に呼び出されたときに、入力変数が不明なためです。

構文

{attribute 'call_after_online_change_slot' := '<slot>'}

<slot> を呼び出しシーケンス内の優先順位を定義する整数値に置き換えます。値が小さいほど、呼び出し優先順位は高くなります。複数のモジュールがその属性に対して同じ優先順位値を持つ場合、それらが呼び出される順序は未定義のままです。

メソッドに属性が指定されている場合は、該当するファンクションブロックのすべてのインスタンスに対して呼び出されます。すべてのインスタンスは指定されたスロット内で呼び出されます。ただし、インスタンス自体の順序を制御することはできません。

注記：アプリケーションはオンラインでの変更中は実行できないため、この状況で実行される各コードはジッターを引き起こす可能性があります。このため、実行するコードを最小限に抑えます。

Attribute call_before_global_exit_slot

概要

宣言部分の上の行にこの属性を含むすべてのファンクションとプログラムは、GlobalExit の後に呼び出されます。GlobalExit は、新しいダウンロードの前、リセット時、またはオンラインでの変更中に実行され、FB_exit メソッドが提供されているモジュールに影響します。呼び出し順序は属性値によって決まります。

注記：この属性を含むファンクションまたはメソッドで VAR_INPUT 宣言が使用されていると、コンパイルエラーが (コード生成中に) 検出されます。その理由は、オンライン変更中にファンクションが暗黙的に呼び出されたときに、入力変数が不明なためです。

構文

{attribute 'call_before_global_exit_slot' := '<slot>'}

<slot> を呼び出しシーケンス内の優先順位を定義する整数値に置き換えます。値が小さいほど、呼び出し優先順位は高くなります。複数のモジュールがその属性に対して同じ優先順位値を持つ場合、それらが呼び出される順序は未定義のままです。

メソッドに属性が指定されている場合は、該当するファンクションブロックのすべてのインスタンスに対して呼び出されます。すべてのインスタンスは指定されたスロット内で呼び出されます。ただし、インスタンス自体の順序を制御することはできません。

Attribute call_on_type_change

概要

A が参照するファンクションブロック B や C などのデータ型が変更されたときにこのメソッドが呼び出されるようにするには、ファンクションブロック A のメソッドに Attribute call_on_type_change pragma を追加します。ファンクションブロックは、ポインター (*569* ページ) またはリファレンス (*568* ページ) によって参照できます。

構文

{attribute 'call_on_type_change':= '< 最初の参照先ファンクションブロックの名前 >|<2 番目の参照先ファンクションブロックの名前 >|<n 番目の参照先ファンクションブロックの名前 >'}

メソッド宣言の最初の行の上に、Attribute call_on_type_change を挿入します。

例

リファレンスありのファンクションブロックの例

```
FUNCTION_BLOCK FB_A
...
VAR
   var_pt: POINTER TO FB_B;
   var_ref: REFERENCE TO FB_C;
END_VAR
...
```

参照されているファンクションブロック FB_B および FB_C でデータタイプが変更されたときに呼び出されるメソッドの例

```
{attribute 'call_on_type_change' := 'FB_B,
FB_C'}
METHOD METH_react_on_type_change : INT
VAR_INPUT
...
```

Attribute const_replaced, Attribute const_non_replaced

概要

コンパイラーオプション **Replace constants** を明示的に有効にする場合は、グローバル定数の宣言に pragma {attribute 'const_replaced'} を挿入します。これにより、**シンボル設定**で定数が使用可能になります。

それに対応して、コンパイラーオプション **Replace constants** を無効にするために、pragma {attribute 'const_non_replaced'} を挿入できます。

Replace constants オプションは、**プロジェクト設定 → Compile options** ダイアログボックスでプロジェクト全体に対して事前定義されています *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

構文

```
{attribute 'const_replaced'}
```

```
{attribute 'const_non_replaced'}
```

例

Replace constants は pragma によって無効にされるため、iTestCon および bTestCon は**シンボル設定**使用できます。

```
VAR_GLOBAL CONSTANT
   {attribute 'const_non_replaced'}
   iTestCon    :    INT  := 12;
   {attribute 'const_non_replaced'}
   bTestCon    :    BOOL := TRUE;
   rTestCon    :    REAL := 1.5;
END_VAR
```

```
VAR_GLOBAL
   iTestVar    :    INT  := 12;
   bTestVar    :    BOOL := TRUE;
END_VAR
```

Attribute 'dataflow'

概要

The pragma {attribute 'dataflow'} を使用すると、FBD/LD/IL エディターでファンクションブロックを処理するときにデータフローを制御できます。この属性は、次または前のファンクションブロックへの接続として使用されるファンクションブロックの入力または出力を定義します。

ファンクションブロックの宣言で、この属性を入力 1 つと出力 1 つにのみ割り当てることができます。

{attribute 'dataflow'} のないファンクションブロックの場合、データフローは次のように自動的に決定されます。

接続は、同じタイプの出力と入力の間に確立されます。ファンクションブロックの最上位の入力変数と出力変数が最初に使用されます。同じデータ型を持つ変数がない場合は、一番上の出力が次のファンクションブロックの一番上の入力に接続されます。

また、ポインターを使用してファンクションブロックのコネクターピンを他の位置に接続することによって、エディター内の制御フローを変更することもできます。詳細については、*要素の挿入*、*配置*、*および置換* の説明を参照してください (*241* ページ)。

構文

```
{attribute 'dataflow'}
```

例

FB と前のファンクションブロックは入力変数 i1 で繋がれます。FB と次のファンクションブロックは出力変数 outRes1 で繋がれます。

```
FUNCTION_BLOCK FB
VAR_INPUT
        r1 : REAL;
        {attribute 'dataflow'}
        i1 : INT;
        i2 : INT;
        r2 : REAL;
END_VAR
VAR_OUTPUT
        {attribute 'dataflow'}
        outRes1 : REAL;
        out1 : INT;
        g1 : INT;
        g2 : REAL;
END_VAR
```

Attribute displaymode

概要

pragma {attribute displaymode} を使用して、1 つの変数の表示モードを定義します。この設定により、サブメニュー **Display Mode** (初期設定では **オンライン**メニュー) のコマンドで実行されるすべての監視変数の表示モード用グローバル設定が上書きされます。

変数宣言を含む行の上の行に pragma を配置します。

構文

{attribute 'displaymode':=<displaymode>}

以下の定義を使用できます。

- バイナリー形式で表示

{attribute 'displaymode':='bin'}
{attribute 'displaymode':='binary'}

- 10 進数形式で表示

{attribute 'displaymode':='dec'}
{attribute 'displaymode':='decimal'}

- 16 進数形式で表示

{attribute 'displaymode':='hex'}
{attribute 'displaymode':='hexadecimal'}

例

```
VAR
  {attribute 'displaymode':='hex'}
  dwVar1: DWORD;
END_VAR
```

Attribute estimated-stack-usage

概要

pragma {attribute 'estimated-stack-usage' := '<allowed stack size in bytes>'} は、アクティブスタックチェックを持つランタイムシステムが、スタック領域に十分なスペースがないことを示すメッセージを発行しないようにするのに役立ちます。このチェックはコード生成中に実行されます。再帰的なメソッドには、メッセージ数を減らすことができます。

構文

{attribute 'estimated-stack-usage' := '<allowed stack size in bytes>'}

位置の挿入

該当するメソッドの宣言部の METHOD　宣言の上の行にこの pragma を挿入します。

例

```
{attribute 'estimated-stack-usage' := '99'}
METHOD xxMETH : INT
VAR_INPUT
END_VAR
   VAR next: AFB;
   big: ARRAY[0..100] OF DINT;
END_VAR
next.xxMETH();
```

Attribute ExpandFully

概要

pragma {attribute 'ExpandFully'} を使用して、参照されたビジュアライゼーションの入力変数として使用される配列のすべてのメンバーを **Visualization Properties** ダイアログボックス内でアクセスできるようにします。

構文

{attribute 'ExpandFully'}

例

ビジュアライゼーション visu をビジュアライゼーション visu_main 内のフレームに挿入することを目的としています。

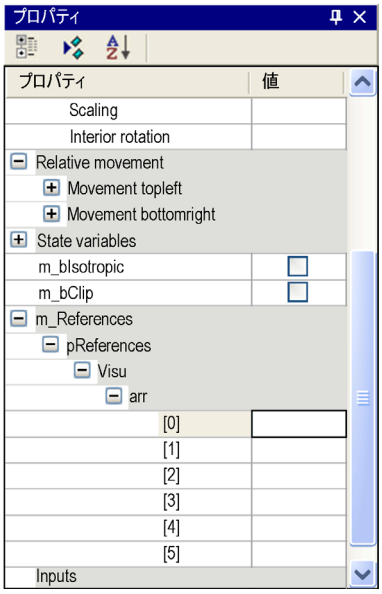
入力変数 arr は、visu のインターフェイスエディターで定義され、後に visu_main にあるフレームの**プロパティ**ダイアログボックスで割り当てるために使用できます。

プロパティダイアログボックスから使用可能な配列の特定要素を取得するには、visu のインターフェイスエディターで arr の直前に属性 ExpandFully を挿入します。

visu のインターフェイスエディターでの宣言

```
VAR_INPUT
{attribute 'ExpandFully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```

visu_main のフレームの**プロパティ**ダイアログボックスの結果



Attribute global_init_slot

概要

pragma {attribute 'global_init_slot'} は、POU またはグローバル変数リストの初期化シーケンスを定義します。

リスト内の変数 (GVL または POU) は上から下に向かって初期化されます。

使用可能なグローバル変数リストが複数ある場合、初期化の順序は定義されていません。

初期化の順序は、1、'hello'、3.6 などのリテラル値、または基本データ型の定数には関係ありません。ただし、異なるリスト間に依存関係がある場合は、初期設定の順序を定義する必要があります。これを実現するには、属性 attribute global_init_slot を使用して、定義済みの初期化スロットを GVL、または POU に割り当てることができます。

構文

{attribute 'global_init_slot' := '< スロット >'}

< スロット > に初期化の順序を定義する値の整数を代入します。POU (プログラムファンクションブロック) のデフォルト値は 50,000 です。GVL のデフォルト値は 49,990 です。値を小さくすると先に初期化されます。属性 global_init_slot に同じ値を持つ複数の POU または GVL の場合、それらの初期化の順序は未定義のままです。これは、**メッセージ** ビューの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **ビルド**カテゴリーで検出されたプログラムエラーとして示されます。

pragma {attribute 'global_init_slot'} は、GVL または POU 全体で有効なので VAR_GLOBAL または POU 宣言の上に位置しなければなりません。

例

この例では、2 つのグローバル変数リスト GVL_1 と GVL_2、および両方のリストの変数を使用するプログラム PLC_PRG を含みます。

GVL_1 は変数 B を使用して、GVL_2 で値 1000 で初期化された変数 A を初期化します。

GVL_1:

```
VAR_GLOBAL //49990
  A : INT := GVL_2.B*100;
END_VAR
```

GVL_2:

```
VAR_GLOBAL //49990
    B : INT := 1000;
  C : INT := 10;
END_VAR
```

PLC_PRG:

```
PROGRAM PLC_PRG //50000
VAR
    ivar: INT := GVL_1.A;
    ivar2: INT;
END_VAR
ivar:=ivar+1;
ivar2:=GVL_2.C;
```

この例をビルドすると、GVL_2 が初期化される前に GVL_2.B が GVL_1.A の初期化に使用されるため、**メッセージ** ビューの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **ビルド**カテゴリーでプログラミングエラーが発生します。これを避けるには、初期化のシーケンスで attribute global_init_slot を使用して GVL_2 を GVL_1 の前に置きます。

プログラムの初期に初期化をするには GVL_2 は 49989 またはそれ以下のスロット値をもつ必要があります。

GVL_2:

```
{attribute 'global_init_slot' := '100'}
VAR_GLOBAL
  B : INT := 1000;
  C : INT := 10;
END_VAR
```

両方 GVL は、いずれの場合もプログラム前に初期化されているので、pragma なし PLC_PRG の実装部分で GVL_2.C を使用することができます。

Attribute hide

概要

pragma {attribute hide} によって、変数またはシグネチャー全体がコンポーネントリストの機能 (*558* ページ) または入力アシスタントに表示されないようにします。pragma に続く変数のみが非表示になります。

構文

{attribute 'hide'}

すべてのローカル変数およびシグネチャーを非表示にするには、attribute hide_all_locals (*531* ページ) を使用します。

例

ファンクションブロック myPOU は、次の属性を使用して実装されます。

```
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
{attribute 'hide'}
a_invisible: BOOL;
a_visible: BOOL;
END_VAR
VAR_OUTPUT
b:INT;
END_VAR
```

メインプログラムでは、ファンクションブロック myPOU のインスタンスが 2 つ定義されます。

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

入力値を POU1 に割り当てると、PLC_PRG の実装部で POU1 を入力する際に機能するコンポーネントリスト (*558* ページ) は、入力変数 a および a_visible (および出力変数 b) を表示します。隠れ入力変数 a_invisible は、表示されません。

Attribute hide_all_locals

概要

pragma {attribute 'hide_all_locals'} によって、シグネチャーのすべてのローカル変数がコンポーネントリストの機能 (*558* ページ) または入力アシスタントに表示されないようにします。この属性は、hide (*530* ページ) 属性をローカル変数の各項目に割り当てることと同じです。

構文

```
{attribute 'hide_all_locals'}
```

例

ファンクションブロック myPOU は、次の属性を使用して実装されます。

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
END_VAR
VAR_OUTPUT
b:BOOL;
END_VAR
VAR
c,d:INT;
END_VAR
```

メインプログラムでは、ファンクションブロック myPOU のインスタンスが 2 つ定義されます。

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

入力値を POU1 に割り当てると、PLC_PRG の実装部で POU1 を入力する際に機能するコンポーネントリスト (*558* ページ) は、変数 a および b を表示します。ローカルの隠れ変数 c または d は、表示されません。

Attribute initialize_on_call

概要

pragma {attribute initialize_on_call} を入力変数に追加できます。この属性のファンクションブロックの入力は、ファンクションブロックが呼び出されるたびに初期化されます。入力がポインターを要求しているがオンライン変更によってこのポインターが削除されている場合、入力は NULL にセットされます。

構文

{attribute 'initialize_on_call'}

Attribute init_namespace

概要

ライブラリーの pragma {attribute init_namespace} で宣言されている STRING または WSTRING 型の変数は、ライブラリーの現在の名前空間で初期化されます。詳細については、ライブラリー管理 *(EcoStruxure Machine Expert, ファンクションおよびライブラリーユーザーガイド 参照)* の詳細を参照してください。

構文

```
{attribute 'init_namespace'}
```

例

ファンクションブロック POU には、必要な属性がすべてあります。

```
FUNCTION_BLOCK POU
VAR_OUTPUT
{attribute 'init_namespace'}
myStr: STRING;
END_VAR
```

メインプログラム PLC_PRG で、ファンクションブロック POU のインスタンス fb が定義されています。

```
PROGRAM PLC_PRG
VAR
fb:POU;
newString: STRING;
END_VAR
newString:=fb.myStr;
```

変数 myStr は現在の名前空間で初期化されます (例、MyLib.XY)。この値は、メインプログラムで newString に代入されます。

Attribute init_On_Onlchange

概要

オンライン変更で変数を初期化するには、次の手順に沿って進めます。

手順	手順内容
1	pragma {attribute 'init_on_onlchange'} を変数に使用して、オン変更 (*205* ページ) のたびにこの変数を初期化します。
2	アプリケーションの**プロパティ** ダイアログボックスの *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **ビルド**タブを開き、**Compiler defines** ボックスに文字列 no_fast_online_change を入力します。

構文

```
{attribute 'init_on_onlchange' }
```

位置の挿入

変数の宣言部の先頭行にこの pragma を挿入します。

Attribute instance-path

概要

pragma {attribute instance-path} をローカル文字列変数に追加できます。このローカル文字列変数は、この文字列変数が属する POU の**アプリケーションツリー**パスで初期化されます。この pragma を指定するには、対応する POU に attribute reflection (*553* ページ)、その文字列変数に追加の属性 noinit (*544* ページ) が使用されていることを前提としています。

構文

```
{attribute 'instance-path'}
```

例

以下のファンクションブロック POU には属性 'reflection' があるとします。

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
{attribute 'instance-path'}
{attribute 'noinit'}
str: STRING;
END_VAR
```

メインプログラム PLC_PRG 内で、ファンクションブロック POU のインスタンス myPOU が呼ばれます。

```
PROGRAM PLC_PRG
VAR
myPOU:POU;
myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

インスタンス myPOU の初期化後、文字列変数 str が、インスタンス myPOU のパスに代入されます。例 : PLC.Application.PLC_PRG.myPOU。このパスは、メインプログラム内で変数 myString に代入されます。

注記：文字列変数の長さは任意に定義できます (>255 でも可)。ただし、その長さより短い文字列変数に代入されると、文字列は (末尾から) 切り捨てられます。

Attribute linkalways

概要

pragma {attribute 'linkalways'} を使用して POU、またはグローバル変数リストをコンパイラー用にマーク付けすることで常にコンパイル情報に含まれるようにします。その結果、このオプションのあるオブジェクトは常にコンパイルされてコントローラーにダウンロードされます。コンパイラーオプション **Link always** も同じ効果があります。

構文

{attribute 'linkalways'}

シンボル設定エディターを使用すると、マークされた POU がシンボル設定用に選択可能な変数のベースとして使用されます。

例

グローバル変数リスト GVLMoreSymbols は、属性 'linkalways' を使用して実装されます。

```
{attribute 'linkalways'}
VAR_GLOBAS
g_iVar1: INT;
g_iVar2: INT;
END_VAR
```

このコードでは、GVLMoreSymbols のシンボルが **シンボル設定**で選択できるようになります。

Attribute monitoring

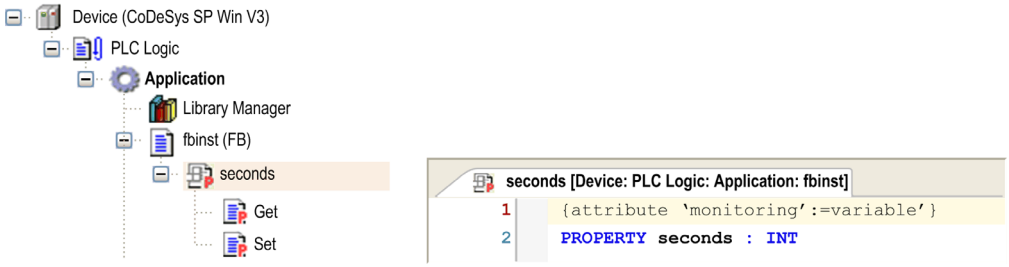
概要

この属性 pragma によって、IEC エディターのオンラインビューまたはウォッチリストで監視されているプロパティおよびファンクション呼び出しの結果を取得できます。

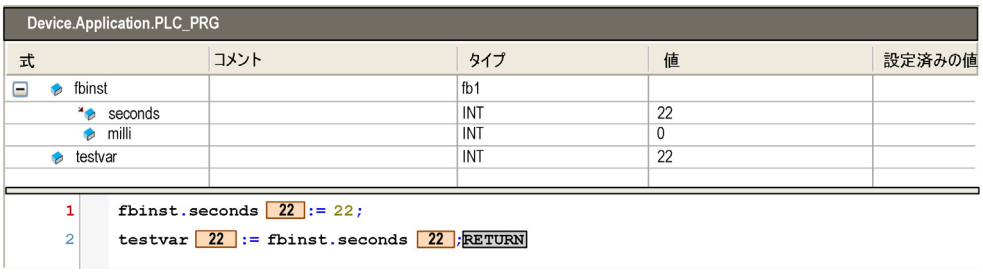
プロパティの監視

pragma をプロパティ定義の上の行に追加します。その結果、プロパティまたはウォッチリストを使用する POU のオンラインビューにプロパティの変数の名前、型、および値が表示されます。そこに、プロパティに属する変数を強制するための設定済みの値を入力することもできます。

変数監視用に準備されたプロパティの例



監視ビューの例



プロパティ変数の現在値の監視

プロパティ変数の現在値を監視する方法は 2 種類あります。特定の使用方法において、実際に必要な値を取得するために適している属性を慎重に検討してください。これは、変数の処理がプロパティ内で実行されるかによって異なります。

1.Pragma {attribute 'monitoring':='variable'}

アプリケーションが set または get メソッドを呼び出すたびに現在のプロパティの値を取得する暗黙の変数がプロパティ用に作成されます。この暗黙の変数に格納されている最新の値が監視されます。

構文

{attribute 'monitoring':='variable'}

2.Pragma {attribute 'monitoring':='call'}

この属性は、単純なデータ型またはポインターを返すプロパティに対してのみ使用できます。構造体型では使用できません。

監視される値は、プロパティの直接呼び出しによって読み書きされます。ランタイムシステムの監視サービスがプロパティの実装部分を含むプロパティファンクションの Get または Set メソッドを実行します。

注記：中間変数 (*1. Pragma*) を使用する代わりに、このモニタリングタイプを選択する場合、プロパティ内で実装されている操作に起因して起こりうる問題を考慮してください。

注記：monitoring pragma もシンボルの設定 (*459* ページ) によって評価されます。値 variable が指定されている場合は、プロパティに対する読み取りアクセスのみがシンボル設定で使用できます。

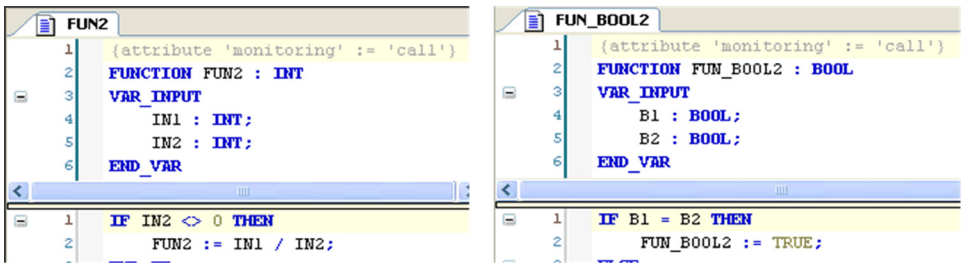
構文

{attribute 'monitoring':='call'}

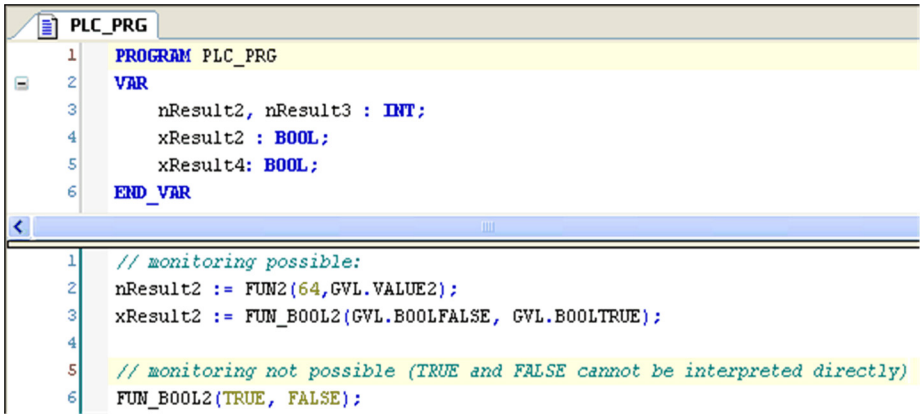
ファンクション呼び出しの結果の監視

ファンクション呼び出しの監視は、4 バイトの数値 (例えば、INT、SHORT、LONG) として解釈できる定数に対して使用できます。他の入力パラメーター (例えば、BOOL) の場合は、定数パラメーターの代わりに変数を使用します。pragma {attribute 'monitoring':='call'} をファンクション宣言の上の行に追加します。POU で変数がファンクション呼び出しの結果に代入され、その POU のオンラインビューにあるテキストエディタービューでこの変数を監視できます。同様に、変数をウォッチリストに追加することもできます。ウォッチビューで提供された変数をすぐに取得するには、**ウォッチを追加**コマンドを実行します。

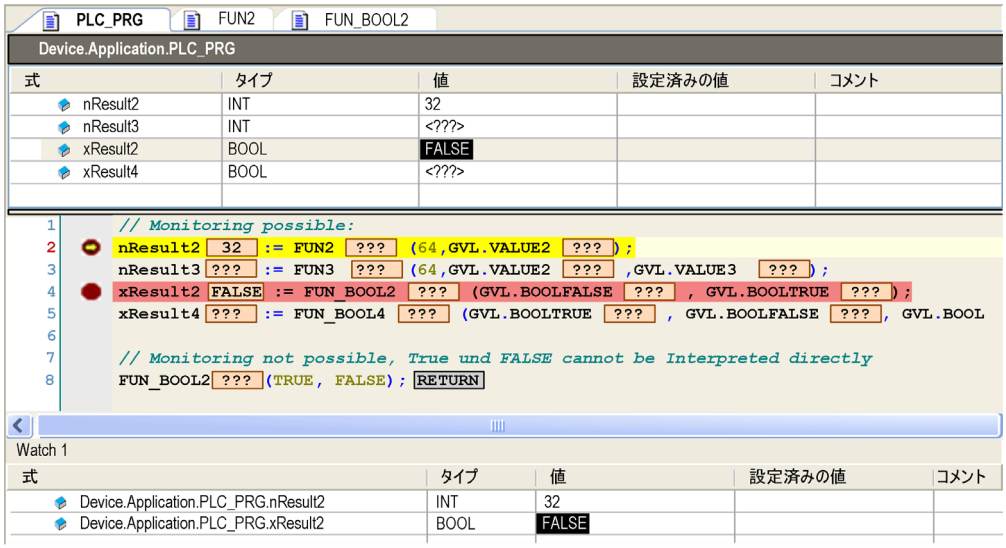
例 1: ファンクション FUN2 および FUN_BOOL2、属性 'monitoring' 付き



例 2: POU プログラム内のファンクション FUN2 および FUN_BOOL2 の呼び出し



例 3: オンラインモードでのファンクション呼び出し



外部ファンクションの暗黙的呼び出しによる変数の監視

外部ファンクションの暗黙的呼び出しによる変数の監視の場合は、以下の条件を満たす必要があります。

- ファンクションが、{attribute 'monitoring' := 'call'} でマーク付けされている。
- ファンクションが、**Link Always** としてマーク付けされている。
- 変数が、{attribute 'monitoring_instead' := 'MyExternalFunction(a,b,c)'}. でマーク付けされている。
- 値 a,b,c が整数値であり、呼び出すファンクションの入力パラメーターと一致する。

注記：ファンクションの強制または書き込みには対応していません。内部強制フラグとして機能する特定のファンクションに入力パラメーターを追加することで、暗黙的に強制を実装できます。

注記：コンパクトなランタイムシステムでは、ファンクションの監視はできません。

Attribute namespace

概要

pragma {attribute namespace} は、attribute symbol (*555* ページ) と組み合わせてプロジェクト変数の名前空間を再定義します。GVL またはプログラムのような POU 全体に指定できますが、特定の変数には指定できません。関連する変数は新しい名前空間の定義と共にシンボルファイルにエクスポートされ、このファイルのダウンロード後にコントローラーで使用可能になります。

これにより、元々異なる名前空間をもつ POU またはビジュアライゼーションから変数にアクセスできます。例えば、以前の EcoStruxure Machine Expert ビジュアライゼーションをそれ以降の EcoStruxure Machine Expert 環境で実行することもできます。

詳細については、シンボル設定の説明を参照してください。ダウンロード時またはプロジェクトのオンライン変更時に新しいシンボルファイルが作成されます。ファイルはアプリケーションと共にコントローラーにダウンロードされます。

構文

```
{attribute 'namespace' := '<namespace>'}
```

プログラム変数の名前空間の置換の例

```
{attribute 'namespace':='prog'}
PROGRAM PLC_PRG
VAR
  {attribute 'symbol' := 'readwrite'}
  iVar:INT;
  bVar:BOOL;
END_VAR
```

例えば、以前に iVar が App1.PLC_PRG.ivar によってアクセスされていた場合、現在は prog.ivar からアクセスできます。

その他の置換例

元の名前空間	変数	名前空間の置換	現在のプロジェクト内での変数へのアクセス
App1.Lib2.GVL2	Var07	{attribute 'namespace':=''}	.Var07
App1.GVL2	Var02	{attribute 'namespace':='Ext''}	Ext.Var02
App1.GVL2.FB1	Var02	{attribute 'namespace':='App1.GVL2''}	App1.GVL2.Var02

表に示されている置換によって、シンボルファイルに以下が入力されます。

```
<NodeList>
 <Node name="">
    <Node name="Var07" type="T_INT" access="ReadWrite">
 </Node>
</NodeList>
<NodeList>
 <Node name="Ext">
  <Node name="Var02 " type="T_INT" access="ReadWrite"></Node>
 </Node>
</NodeList>
<NodeList>
 <Node name="App1">
  <Node name="GVL2">
   <Node name="Var02 " type="T_INT" access="ReadWrite"></Node>
    </Node>
 </Node>
</NodeList>
```

Attribute no_assign

概要

pragma {attribute 'no assign'} をファンクションブロックの宣言部の先頭行に挿入してください。これにより、ファンクションブロックのインスタンスが同じファンクションブロックの別のインスタンスに割り当てられた場合にコンパイルエラーが発生します。例えば、ファンクションブロックにポインターが含まれている場合は、このような割り当てを避けたい場合があります。値が割り当てられるときにコピーされるので、問題を引き起こす可能性があります。

注記：内部ポインターをもつファンクションブロックで {attribute 'no assign'} を使用すると、ファンクションブロックのインスタンスが同じファンクションブロックの別のインスタンスに割り当てられないようになります。

ポインターをもつファンクションブロックインスタンスの割り当て

この例では、fb_exit を実行したときにファンクションブロックインスタンスの値の割り当てによって問題が発生します。

```
VAR_GLOBAL
inst1 : TestFB;
   awsBufferLogFile : ARRAY [0..9] OF WSTRING(66);(* Area: 0, Offset: 0x1304 (4868)*)
   LogFile : SEDL.LogRecord := (sFileName := 'LogFile.log', pBuffer := ADR(awsBufferLogFile), udiMax
EntriesFile := UDINT#10000, udiMaxBuffered := UDINT#10, uiLineSize := UINT#64, wsSep := " ", xCircu
lar := TRUE, siDateFormat := SINT#0, siTimeFormat := SINT#0);
END_VAR
```

```
PROGRAM PLC_PRG
VAR
   inst2 : TestFB := inst1;
   LogFileNew : LogRecord := LogFile;
END_VAR
```

この場合、LogRecord がポインターのリストを管理します。fb_exit が適用される場合、それらに対して異なる動作が実行されます。ファンクションブロックインスタンスを割り当てると、fb_exit が 2 回実行され、問題が発生します。これを防ぐには、ファンクションブロック TestFB の宣言に no_assign 属性を追加します。

```
{attribute 'no_assign'}
FUNCTION_BLOCK TestFB
VAR_INPUT
...
```

次のコンパイルエラーが報告されます。

C0328: Assignment not allowed for type TestFB

C0328: Assignment not allowed for type LogRecord

Attribute no_check

概要

暗黙的チェック用のための POU の呼び出しを抑制するために、POU の宣言に pragma {attribute 'no_check'} を追加できます。チェック機能によってパフォーマンスに影響がある場合があるため、頻繁に呼び出される POU、またはすでに承認されている POU にはこの属性を指定します。

注記：この属性は、POU の子オブジェクトにも自動的に影響します。

例 : この属性を持つプログラムでは、チェック機能は実行されません。このプログラムに割り当てられている動作に対しても実行されません。

構文

{attribute 'no_check'}

Attribute no_copy

概要

一般的にオンライン変更では、例えば、POU のようにインスタンスの再割り当てが必要です。このインスタンス内の変数の値はコピーされます。

ただし、pragma {attribute no_copy} が変数に追加されている場合は、オンライン変更によるこの変数のコピーは実行されず、その代わりにこの変数が初期化されます。これは、オンライン変更によって実際にシフトされた変数 (従って、アドレスが変更された変数) を指すローカルポインター変数の場合に合理的です。

構文

{attribute 'no_copy'}

Attribute no-exit

概要

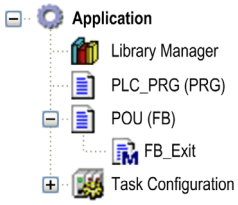
ファンクションブロックに FB_exit メソッド (*501* ページ) がある場合、pragma {attribute no-exit} をファンクションブロックに割り当てることで特別なインスタンスの呼び出しを抑制できます。

構文

{attribute 'no-exit'}

例

FB_Exit メソッドが POU という名前のファンクションブロックに追加されたとします。



メインプログラム PLC_PRG では、POU タイプの 2 つの変数がインスタンス化されます。

```
PROGRAM PLC_PRG
VAR
POU1 : POU;
{attribute 'no-exit'}
POU2 : POU;
END_VAR
```

POU1 で bInCopyCode 変数が TRUE になると、FB_Exit メソッドが呼び出され、後でコピーされるインスタンスを終了します (オンライン変更)。FB_Exit メソッドは、ファンクションブロックインスタンスの POU2 のコンテキストでは呼び出されません。

Attribute no_init

概要

pragma {attribute no_init} の変数は、暗黙的には初期化されません。pragma は、続いて宣言される変数に属します。

構文

{attribute 'no_init'}

以下も可

{attribute 'no-init'}
{attribute 'noinit'}

例

```
PROGRAM PLC_PRG
VAR
A : INT;
{attribute 'no_init'}
B : INT;
END_VAR
```

関連するアプリケーションでリセットが実行された場合、整数の変数 A は暗黙的に 0 で再度初期化されるが、変数 B は現在割り当てられている値を保持します。

Attribute no_instance_in_retain

概要

pragma {attribute 'no_instance_in_retain'} は、特定のファンクションブロックのインスタンスが保持メモリー領域に格納されるのを回避するのに役立ちます。

ファンクションブロックの宣言部の先頭行に挿入してください。これには、POU のインスタンスが RETAIN 変数として宣言されている場合にメッセージが作成されるという効果があります。

構文

{attribute 'no_instance_in_retain'}

Attribute no_virtual_actions

概要

この属性は、SFC で実装された基本ファンクションブロックから派生し、基本クラスのメイン SFC ワークフローを使用しているファンクションブロックに対して有効です。ここで呼び出されるアクションは、メソッドと同じ仮想動作を示します。つまり、基本クラスのアクションは、派生クラスに関連する特定の実装によって上書きされる場合があります。

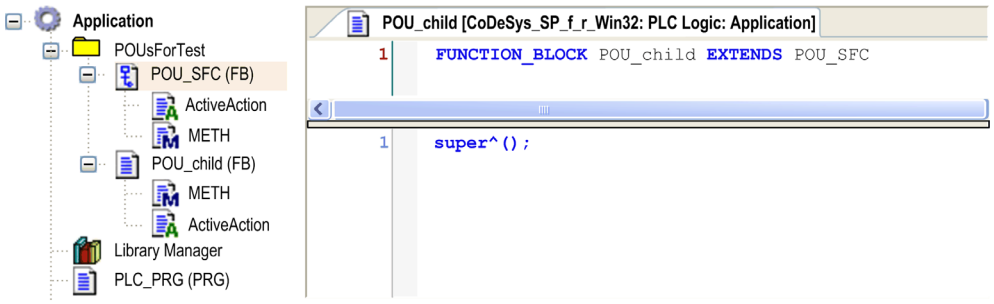
基本クラスのアクションが上書きされないようにするには、基本クラスに pragma {attribute 'no_virtual_actions'} を指定します。

構文

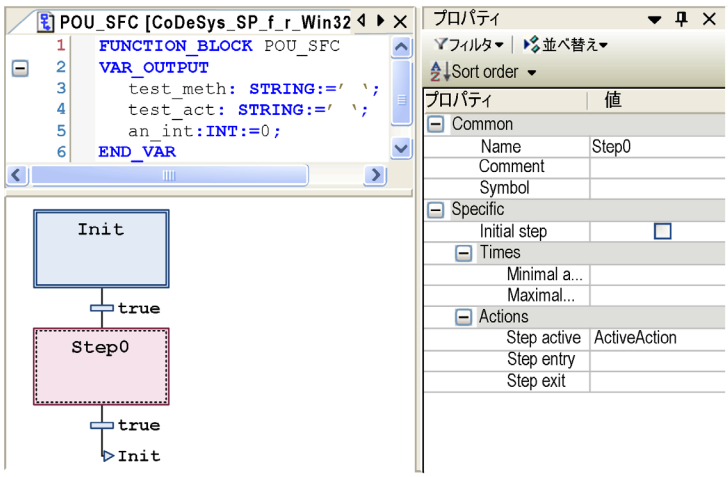
{attribute 'no_virtual_actions'}

例

次の例では、ファンクションブロック POU_SFC にファンクションブロック POU_child によって拡張される基本クラスがあります。



キーワード SUPER を使用して、派生クラス POU_child が SFC で実装された基本クラスのワークフローを呼び出します。



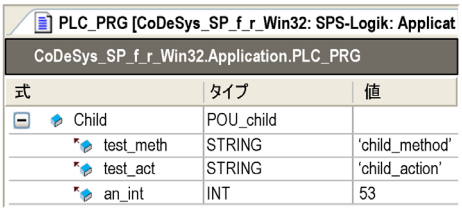
このワークフローの典型的な実装は、初期ステップに限定されます。この後に、出力変数の割り当てに関係するステップアクション ActiveAction に関連するステップが 1 つ続きます。

```
an_int:=an_int+1;          // counting the action calls
test_act:='father_action'; // writing string variable test_act
METH();                    // Calling method METH for writing string variable test_meth
```

派生クラス POU_child の場合、ActiveAction の特定の実装によって上書きされます。これは、文字列 'father_action' の代わりに 'child_action' を変数 test_act に代入する点で元とは異なります。

同様に、基本クラス内で文字列 'father_method' を変数 test_meth に代入する METH メソッドは、代わりに test_meth を 'child_method' に代入するように上書きされます。

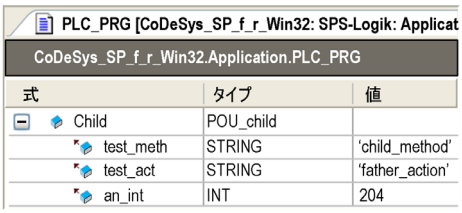
メインプログラム PLC_PRG は、繰り返し Child (POU_child のインスタンス) の呼び出しを実行します。予想されるように、出力文字列の実際の値は、派生クラスのアクションとメソッドの呼び出しを示します。



基本クラスの前に属性 'no_virtual_actions' がある場合は、異なる動作になります。

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK POU_SFC...
```

METH メソッドが派生クラス内での実装によって上書きされる一方、ステップアクションの呼び出しは基本クラスのアクション ActiveAction の呼び出しになります。従って、test_act には文字列 'father_action' が代入されます。



Attribute pingroup

概要

入力ピンまたは出力ピン (パラメーター) をグループ化するために、ファンクションブロックの宣言に pragma {attribute 'pingroup' := '<groupname>'} を挿入します。次に、FBD および LD エディターのそれぞれのボックスで、各ピングループを表示、折りたたみ、または展開できます。複数のグループが可能であり、それらは名前によって区別されます。ボックスごとの特定の状態 (折りたたまれた / 展開された) はプロジェクトオプションに保存されます。

属性 pingroup のない入力と出力は、常にグループの上に表示されます。

構文

```
{attribute 'pingroup' := '<groupname>'}
```

例

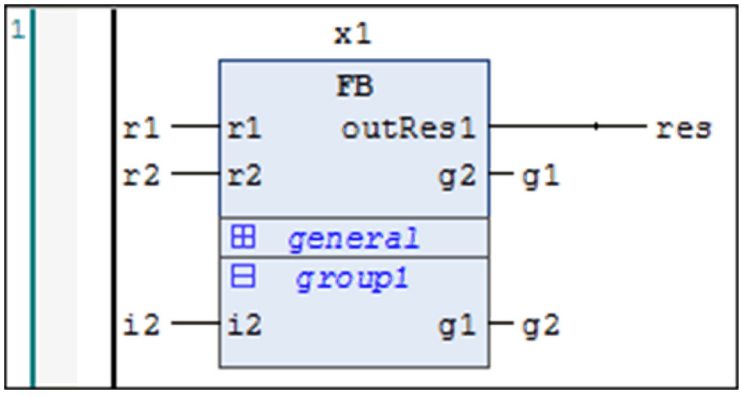
2 つのグループが定義されます。

- general (i1, out1)
- group1 (i2, g1)

r1, r2, outRes1 および g2 は常に表示されます。

```
FUNCTION_BLOCK FB
VAR_INPUT
  r1 : REAL;
  {attribute 'pingroup' := 'general'}
  i1 : INT;
  {attribute 'pingroup' := 'group1'}
  i2 : INT;
  r2 : REAL;
END_VAR
VAR_OUTPUT
  outRes1 : REAL;
  {attribute 'pingroup' := 'general'}
  out1 : INT;
  {attribute 'pingroup' := 'group1'}
  g1 : INT;
  g2 : REAL;
END_VAR
```

FBD エディターのピングループ



Attribute pin_presentation_order_inputs/outputs

概要

pragmas は、ファンクションブロックの入力と出力がグラフィック言語エディターに表示される順序を定義します。

構文

{attribute 'pin_presentation_order_inputs' :=
'<input_k>,<input_l>,*,<input_m>'}

{attribute 'pin_presentation_order_outputs' :=
'<output_k>,<output_l>,*,<output_m>'}

* 文字は、入力パラメーターまたは出力パラメーターの並び替え済みリストの先頭と末尾の間の区切り文字です。区切り文字は、未定義の入力パラメーターまたは出力パラメーターに置き換えられます。区切り文字が利用できない場合は、pragma で明示的に定義されていない入力または出力パラメーターが並べ替え済みリストの末尾に追加されます。

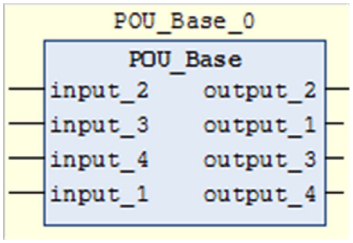
ファンクションブロックの宣言部の先頭行に pragmas が挿入されます。

注記：pragmas の `attribute 'pin_presentation_order_inputs`、および `attribute 'pin_presentation_order_outputs` は pragma の `pingroup` が使用される場合は、評価されません。

例

```
{attribute 'pin_presentation_order_inputs' :=
'input_2,*,input_1'}
{attribute 'pin_presentation_order_outputs' :=
'output_2, output_1}
FUNCTION_BLOCK POU_BASE
VAR_INPUT
   input_1 : BOOL;
   input_2 : INT;
   input_3 : INT;
   input_4 : INT;
END_VAR
VAR_OUTPUT
   output_1 : BOOL;
   output_2 : INT;
   output_3 : INT;
   output_4 : BOOL;
END_VAR
```

この pragma 定義では、POU_Base ファンクションブロックの入力および出力ピン順序が以下のようになります。



Attribute obsolete

概要

obsolete pragma をデータ型定義に追加することで、プロジェクト内で各データ型 (構造体、ファンクションブロックなど) が使用されている場合に、ビルド中にユーザー定義の警告を発生させることができます。従って、データ型がもう使用されていないことを通知できます。

ローカルで使用されるメッセージ pragma (*508* ページ) とは異なり、この警告は定義内で定義されるため、データ型のすべてのインスタンスに対してグローバルです。

この pragma 指令は現在の行に対して有効です。また、別の行に置かれている場合は次の行に対して有効です。

構文

```
{attribute 'obsolete' := 'user-defined text'}
```

例

旧 pragma がファンクションブロック fb1 の定義に挿入されています。

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
i:INT;
END_VAR
...
```

宣言内で fb1 がデータ型として使用されている場合 (例、fbinst: fb1;)、プロジェクトがビルドされると次の警告が出力されます。

'datatype fb1 not valid'

Attribute pack_mode

概要

pragma {attribute 'pack_mode'} は、割り当て中にパッキングされるデータ構造のモードを定義します。データ構造の上に属性を設定します。構造全体のパッキングに影響します。

構文

{attribute 'pack_mode' := '< 値 >'}

プレースホルダー < 値 > には以下の値が入ります。

pack_mode	関連付けられたパッキングメソッド
0	整列
1	1 バイト整列
2	2 バイト整列
4	4 バイト整列
8	8 バイト整列

構造体によっては、個々のモードのメモリーマッピングに違いがない場合があります。例えば、pack_mode = 4 の構造体のメモリー割り当ては pack_mode = 8 と一致します。

構造体が配列に結合されている場合は、各構造体の末尾にバイトが追加されて、次の構造体が整列されます。

例

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
STRUCT
 Enable: BOOL;
 Counter: INT;
 MaxSize: BOOL;
 MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

データ型 myStruct の変数は整列されてインスタンス化されます。

その要素 Enable のアドレスが 0x0100 の場合、要素 Counter はアドレス 0x0101、MaxSize は 0x0103、そして MaxSizeReached は 0x0104 と続きます。

pack_mode=2 の場合、Counter は 0x0102、MaxSize は 0x0104、そして MaxSizeReached は 0x0106 にあります。

Attribute qualified_only

概要

pragma {attribute 'qualified_only'} がグローバル変数リストの先頭に割り当てられている場合、このリストの変数はグローバル変数名 (例えば、gvl.g_var) を使用してのみアクセスできます。これは、列挙型の変数に対しても機能します。ローカル変数との名前の不一致を避けるために役立ちます。

構文

{attribute 'qualified_only'}

例

以下のグローバル変数リスト (GVL) に属性 'qualified_only' があるとします。

```
{attribute 'qualified_only'}
VAR_GLOBAL
iVar:INT;
END_VAR
```

POU PLC_PRG 内では、この例で示されているようにグローバル変数は接頭辞 GVL を付けて呼び出す必要があります。

```
GVL.iVar:=5;
```

以下の変数の不完全な呼び出しは、エラーとして検出されます。

```
iVar:=5;
```

Attribute reflection

概要

pragma {attribute 'reflection'} はシグネチャーに指定されます。パフォーマンス上の理由から、この属性は POU が instance-path 属性 (*534* ページ) をもつために必須です。

構文

```
{attribute 'reflection'}
```

例

attribute instance-path の例 (*534* ページ) を参照してください。

Attribute subsequent

概要

pragma {attribute 'subsequent'} は、変数を強制的にメモリーの 1 つの場所に連続的に割り当てます。リストが変更された場合、リスト全体が新しい場所に割り当てられます。この pragma は、プログラムおよびグローバル変数リスト (GVL) で使用します。

構文

{attribute 'subsequent'}

注記：リストの変数の 1 つが RETAIN である場合、リスト全体が RETAIN メモリーに配置されます。

注記：プログラムで属性 subsequent をもつ VAR_TEMP は、コンパイラーエラーとして検出されます。

Attribute symbol

概要

pragma {attribute 'symbol'} は、シンボル設定で処理する変数を定義します。

変数に対して次のエクスポート処理が実行されます。
- 変数はシンボル設定でシンボルとして公開されます。
- 変数はプロジェクトディレクトリー内の XML ファイルにエクスポートされます。
- 変数は対象システムの外部アクセス用 (OPC サーバーなど) に使用できる非表示ファイルにエクスポートされます。

属性付きの変数は、属性が設定されていない場合やシンボル設定エディターで表示されない場合でも、コントローラーにダウンロードされます。

注記：シンボル設定は**ツールツリー**の各アプリケーションでオブジェクトとして使用できる必要があります。

構文

{attribute 'symbol' := 'none' | 'read' | 'write' | 'readwrite'}

プログラムやグローバル変数リストからのシンボルにのみアクセスできます。シンボルにアクセスするには、シンボル名を指定します。

特定の変数に pragma 定義を割り当てたり、プログラムで宣言されたすべての変数をまとめて割り当てることができます。
- 1 つの変数に対して有効にするには、pragma を変数宣言の前の行に記述します。
- プログラムの宣言部分に含まれるすべての変数に対して有効にするには、宣言エディターの最初の行に pragma を記述します。この場合、pragma を明示的に追加することによって特定の変数の設定も変更できます。

次の pragma パラメーターでシンボルへのアクセスを定義します。
- 'none'
- 'read'
- 'write'
- 'readwrite'

パラメーターが定義されていない場合、デフォルトの 'readwrite' が有効になります。

例

次の設定では、変数 A および B が読み書きアクセスでエクスポートされます。変数 D は読み込みアクセスでエクスポートされます。

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
A : INT;
B : INT;
{attribute 'symbol' := 'none'}
C : INT;
{attribute 'symbol' := 'read'}
D : INT;
END_VAR
```

Attribute warning disable

概要

pragma warning disable を使用して、警告を抑制できます。警告表示を有効にするには、pragma warning restore を使用します。

構文

{warning disable <compiler ID>}

コンパイラー ID: コンパイラーが検出するすべての警告とエラーには、説明の最初に表示される固有の ID があります。

コンパイラーメッセージの例

```
------ Build started: Application: Device.Application ------
typify code ...
C0196: Implicit conversion from unsigned Type 'UINT' to signed Type 'INT' : possible change of sign
Compile complete -- 0 errors
```

例

```
VAR
   {warning disable C0195}
   test1 : UINT := -1;
   {warning restore C0195}
   test2 : UINT := -1;
END_VAR
```

この例では、test2 に対しての警告が検出されます。test1 に対しての警告は検出されません。

Attribute enable_dynamic_creation

概要

ファンクションブロックに __NEW 演算子 (*664* ページ) を使用するには、pragma enable_dynamic_creation が必要です。

構文

{attribute 'enable_dynamic_creation'}

ファンクションブロックの宣言部の先頭行に pragma を挿入します。

31.6
Smart Coding 機能

Smart Coding

概要

識別子 (変数やファンクションブロックインスタンスなど) を入力できるところ (IEC 61131-3 の言語エディター内や**ウォッチ、トレース、ビジュアライゼーション** ウィンドウ内) では、smart coding 機能が使用できます。**ツール → オプション**ダイアログボックスの**スマートコーディング**で、この機能をカスタマイズできます。

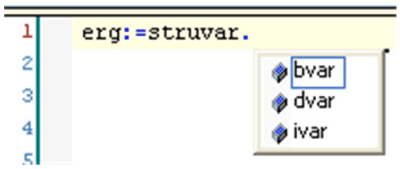
識別子の挿入における対応

smart coding 機能は正しい識別子を挿入する支援をします。

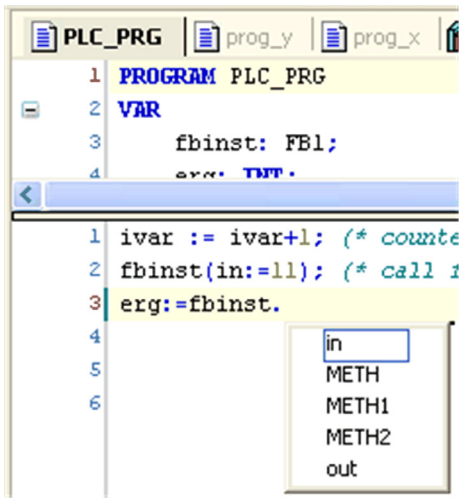
- グローバル識別子が挿入される場所に識別子の代わりにドット (.) を挿入すると、選択ボックスが表示されます。現在使用できるグローバル変数がリスト表示されます。その要素の 1 つを選択し、RETURN キーを押してドットの後に挿入します。リストの項目をダブルクリックすることでも要素を挿入できます。
- ファンクションブロクインスタンスまたは構造体変数の後にドット (.) を入力すると、選択ボックスが表示されます。対応するファンクションブロックまたは構造体の要素の入力および出力変数がリスト表示されます。RETURN キーを押すか、リストの項目をダブルクリックして挿入する要素を選択することができます。
- ST エディターで任意の文字列を入力し CTRL + SPACE を押すことで、選択ボックスが表示されます。プロジェクト内の POU と使用できるグローバル変数がリスト表示されます。指定された文字列で始まるリストの最初の項目が選択されます。RETURN キーを押して、プログラムに挿入します。

例

smart coding 機能は構造体の要素を提供します。



smart coding 機能はファンクションブロックの要素を提供します。



第 32 章
データ型

この章について

この章には次のセクションが含まれています。

セクション	項目	参照ページ
32.1	一般情報	560
32.2	標準データ型	561
32.3	IEC 規格への拡張	565
32.4	ユーザー定義データ型	572

32.1
一般情報

データ型

概要

EcoStruxure Machine Expert のプログラミングでは、標準データ型 (*561* ページ)、ユーザー定義データ型 (*572* ページ)、またはファンクションブロックのインスタンスを使用できます。各識別子はデータ型に割り当てられます。このデータ型によって予約されるメモリー領域の量と格納する値の型が指定されます。

32.2
標準データ型

標準データ型

概略

EcoStruxure Machine Expert は IEC61131-3 規格で記述されるすべてのデータ型 (*560* ページ) に対応しています。

この章では、次のデータ型を説明しています。
- BOOL (*561* ページ)
- 整数型 (*561* ページ)
- REAL / LREAL (*562* ページ)
- STRING (*562* ページ)
- WSTRING (*562* ページ)
- Time データ型 (LTIME) (*563* ページ)
- ANY と ANY_<type> (*563* ページ)

さらに、一部の標準拡張データ型 (*565* ページ) に対応しています。またユーザー定義の型 (*572* ページ) も定義することができます。

BOOL

BOOL 型変数は TRUE (1) または FALSE (0) の値を持ちます。8 ビットのメモリー領域が予約されます。

詳細については、*BOOL 定数* (*679* ページ) の章を参照してください。

注記 : 暗黙的チェックを使用して、変数型の変換を検証できます (*暗黙的チェック用 POU* (*153* ページ) の章を参照してください)。

Integer

次の表は使用できる整数データ型のリストです。各型で値の範囲が異なります。次の範囲制限が適用されます。

データタイプ	下限	上限	メモリー領域
BYTE	0	255	8 ビット
WORD	0	65,535	16 ビット
DWORD	0	4,294,967,295	32 ビット
LWORD	0	2^{64}-1	64 ビット
SINT	-128	127	8 ビット
USINT	0	255	8 ビット
INT	-32,768	32,767	16 ビット
UINT	0	65,535	16 ビット
DINT	-2,147,483,648	2,147,483,647	32 ビット
UDINT	0	4,294,967,295	32 ビット
LINT	-2^{63}	2^{63}-1	64 ビット
ULINT	0	2^{64}-1	64 ビット

注記 : 大きい型から小さい型への変換は、情報の損失を引き起こす可能性があります。

詳細については、整数の定数 (*684* ページ) の説明を参照してください。

注記 : 暗黙的チェックを使用して、変数型の変換を検証できます (*暗黙的チェック用 POU* (*153* ページ) の章を参照してください)。

REAL / LREAL

データ型 REAL および LREAL は浮動小数点型です。これらは有理数を表します。

REAL および LREAL データ型の特徴

データ型	下限	上限	メモリー領域
REAL	-3.402823e+38	3.402823e+38	32 ビット
LREAL	-1.7976931348623158e+308	1.7976931348623158e+308	64 ビット

注記：データ型 LREAL の扱いについては、対象デバイスによって異なります。コンパイル中に 64 ビット型 LREAL が REAL に変換されるか (情報が失われる可能性があります)、保持されるかは対応するマニュアルを参照してください。

注記：REAL または LREAL が SINT, USINT, INT, UINT, DINT, UDINT, LINT、または ULINT に変換され実数の値が整数の値の範囲外になった場合、結果は対象システムに応じて未定義になります。この場合でも例外が可能です。対象に依存しないコードを取得するには、アプリケーションで範囲超過の処理をしてください。REAL/LREAL の値が整数値の範囲内の場合、すべてのシステムで同じように変換されます。

i1 := r1; の代入をすると、エラーが検出されます。従って、次のような変換演算子 (*631* ページ) を使用する場合、前の注記が適用されます。

```
i1 := REAL_TO_INT(r1);
```

詳細については、REAL/LREAL 定数 (オペランド) (*685* ページ) を参照してください。

注記：暗黙的チェックを使用して、変数型の変換を検証できます (*暗黙的チェック用 POU* (*153* ページ) の章を参照してください)。

STRING

STRING データ型変数には任意の文字列を含めることができます。宣言にサイズを記述することで、変数のために予約されるメモリー領域を決定します。サイズの記述は文字列の文字数を表し、括弧または角括弧で囲みます。サイズの指定がない場合、初期値の 80 文字が使用されます。

一般に、文字列の長さは制限されません。ただし、文字列ファンクションで処理できる文字列の長さは 1 ~ 255 文字です。変数データ型に対して長すぎる文字列で変数を初期化した場合、それに対応して文字列が右から左へ切り捨てられます。

注記：STRING 型の変数に対して必要なメモリー領域は 1 文字あたり 1 バイト + 追加の 1 バイトです。"STRING[80]" の宣言では 81 バイトが必要になります。

35 文字の文字列宣言の例

```
str:STRING(35):='This is a String';
```

詳細については、WSTRING および STRING 定数 (オペランド) (*686* ページ) を参照してください。

注記：暗黙的チェックを使用して、変数型の変換を検証できます (*暗黙的チェック用 POU* (*153* ページ) の章を参照してください)。

WSTRING

WSTRING データ型は、STRING 型 (ASCII) と違って、Unicode 形式で解釈し、および各文字につき 2 バイト、余分なメモリー領域が 2 バイト (STRING の場合はそれぞれ 1 バイトのみ) 必要です。

ライブラリー standard64.lib は、WSTRING 文字列用のファンクションを提供します。

WSTRING の文字数は、含まれている文字によって異なります。WSTRING のサイズが 10 の場合、WSTRING の長さは最大 10 WORDS になります。Unicode の一部の文字は、コーディングに複数の WORDS が必要なので、文字数が WSTRING の長さ (この場合は 10) に対応する必要がありません。このデータ型は、0 で終了するため、1 WORD の追加メモリーが必要です。

サイズが定義されていない場合は、80 WORDS プラス 1 (終了文字 0 用) が割り当てられます。

例

```
wstr:WSTRING:="This is a WString";
wstr10 : WSTRING(10) := "1234567890";
```

詳細については、次の説明を参照してください。

- STRING (*562* ページ)
- STRING 定数 (*686* ページ) (オペランド)

Time データ型

データ型 TIME, TIME_OF_DAY (TOD)、DATE、および DATE_AND_TIME (DT) は DWORD のように内部で処理されます。TIME および TOD の時間の単位はミリ秒です。TOD の時間は 12:00 A.M に開始されます。DATE および DT の時間の単位は秒で、1970 年 1 月 1 日 12:00 A.M に開始されます。

LTIME

LTIME が高分解能タイマーの時間ベースとして対応しています。LTIME のサイズは 64 ビット、分解能はナノ秒です。

LTIME の構文

LTIME#< 時間の宣言 >

時間の宣言には、TIME 定数で使用されている時間単位と次の単位を含めることができます。
- us: マイクロ秒
- ns: ナノ秒

LTIME の例

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns

TIME サイズ 32 ビットと分解能ミリ秒 (*563* ページ) の比較。

詳細については、次の説明を参照してください。
- データタイプ (*560* ページ)
- TIME 定数 (*680* ページ)
- DATE 定数 (*681* ページ)
- DATE_AND_TIME 定数 (*682* ページ)
- TIME_OF_DAY 定数 (*683* ページ)

注記 : 暗黙的チェックを使用して、変数型の変換を検証できます (*暗黙的チェック用 POU* (*153* ページ) の章を参照してください)。

ANY / ANY_<type>

ファンクションの実装で、ファンクション入力の 1 つ (VAR_INPUT) が一般的な IEC データ型 (ANY または ANY_<type>) を持つ場合、呼び出しパラメーターのデータ型は一意として定義されません。このファンクションには、さまざまなデータ型の変数を渡すことができます。渡された値とその型は、定義済みの構造体を介してファンクション内で要求できます。

ファンクション入力に基本データ型の使用を許可する汎用 IEC データ型

汎用データ型の階層			基本データ型
ANY	ANY_BIT	–	BOOL, BYTE, WORD, DWORD, LWORD
	ANY_DATE	–	DATE_AND_TIME, DATE, TIME_OF_DAY
	ANY_NUM	ANY_REAL	REAL, LREAL
		ANY_INT	USINT, UINT, UDINT, ULINT SINT, INT, DINT, LINT
	ANY_STRING	–	STRING, WSTRING

例

```
FUNCTION ANYBIT_TO_BCD : DWORD
  VAR_INPUT
    value : ANY_BIT;
  END_VAR
```

ファンクション ANYBIT_TO_BCD が呼び出されると、データ型 B BOOL, BYTE, WORD, DWORD、または LWORD の変数をパラメーターとしてファンクションに渡すことができます。

定義済みの構造体

コードのコンパイル時に、ANY データ型は内部的に次の構造体に置き換えられます。

```
TYPE AnyType :
STRUCT
      // the type of the actual parameter
   typeclass : __SYSTEM.TYPE_CLASS ;
      // the pointer to the actual parameter
   pvalue : POINTER TO BYTE;
      // the size of the data, to which the pointer points
   diSize : DINT;
END_STRUCT
END_TYPE
```

実際の呼び出しパラメーターは、ランタイムで構造体要素を割り当てます。

例

このコード例は、渡された 2 つの変数が同じ型と同じ値を持つかどうかを比較します。

```
FUNCTION Generic_Compare : BOOL
VAR_INPUT
   any1 : ANY;
   any2 : ANY;
END_VAR
VAR
   icount: DINT;
END_VAR

Generic_Compare := FALSE;
IF any1.typeclass <> any2.typeclass THEN
   RETURN;
END_IF
IF any1.diSize <> any2.diSize THEN
   RETURN;
END_IF
// Byte comparison
FOR icount := 0 TO any1.diSize-1 DO
   IF any1.pvalue[iCount] <> any2.pvalue[iCount] THEN
      RETURN;
   END_IF
END_FOR
Generic_Compare := TRUE;
RETURN;
// END_FUNCTION
```

__VARINFO 演算子 (*672* ページ) の説明も参照してください。

32.3
IEC 規格への拡張

概要

この章では、IEC 61131-3 規格に加えて、EcoStruxure Machine Expert で対応しているデータ型をリスト表示します。

このセクションについて

このセクションには次の項目が含まれています。

UNION

概要

IEC 61131-3 規格への拡張として、ユーザー定義型で UNION を宣言できます。

UNION の要素は同じオフセットを持ちます。つまり、同じ格納場所を占有します。そのため、次の例で示すような UNION の定義をすると name.a への代入は name.b も操作します。

例

```
TYPE name: UNION
a : LREAL;
b : LINT;
END_UNION
END_TYPE
```

BIT

概要

BIT データ型は、構造体 (*577* ページ) 内の特定の変数のデータ型にのみ使用できます。利用できる値は TRUE (1) と FALSE (0) です。

BIT 要素は 1 ビットのメモリー領域を消費し、構造体の 1 ビットを名前でアドレス指定することができます (詳細については、*構造体のビットアクセス* (*578* ページ) の項を参照してください)。次々に宣言されるビット要素が、バイト単位で結合されます。いかなる場合も 8 ビットが予約されている BOOL 型 (*561* ページ) とは異なり、メモリーの使用領域は最適化されます。ただし、ビットへのアクセスには、より多くの時間がかかります。そのため、複数のブール値の情報をコンパクトな形式で保存したい場合は BIT データ型を使用してください。

リファレンス

概要

このデータ型は IEC 61131-3 規格への拡張に使用できます。

リファレンスは、メモリー内の他の場所にあるオブジェクト (変数) のアドレスを格納します。この点において動作はポインターと同じです。

ポインターとは対照的に、構文に関しては、変数はオブジェクトのように動作します。さらに、REFERENCE として宣言された変数には、POINTERS: と比較して以下の利点があります。

- 参照先オブジェクトの内容にアクセスするために、リファレンスを明示的に (^ で) 逆参照する必要はありません。
- ファンクション / ファンクションブロック / メソッドの入力パラメーターに値を渡す場合、以下が適用されます。入力が REFERENCE TO 〈データ型〉として宣言されている場合、対応する〈データ型〉の変数を渡すことができます (ptrInput :=ADR (変数) の代わりに refInput := 変数)。
- コンパイラーは、同じデータ型のリファレンスが互いに割り当てられていることを確認します。

詳細は、REF 演算子の割り当て の説明を参照してください (*331* ページ)。

構文

< 識別子 > : REFERENCE TO < データ型 >

宣言の例

```
A : REFERENCE TO DUT;
B : DUT;
C : DUT;
A REF= B; // corresponds to A := ADR(B);
A := C; // corresponds to A^ := C;
```

注記 : REFERENCE TO REFERENCE、ARRAY OF REFERENCE、または POINTER TO REFERENCE のようなリファレンスを宣言することはできません。

有効なリファレンスをチェック

__ISVALIDREF 演算子を使用して、リファレンスが有効な値 (0 以外の値) であるかを検証します。

構文

```
<boolean variable> := __ISVALIDREF(identifier, declared with type <REFERENCE TO <datatype>);
<boolean variable> will be TRUE, if the reference points to a valid value, FALSE if not.
```

例

宣言

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
```

実装

```
ivar := ivar +1;
ref_int REF= ivar;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);   (* will be TRUE, because ref_int points to ivar, which is unequal 0 *)
testref := __ISVALIDREF(ref_int0); (* will be FALSE, because ref_int is set to 0 *)
```

ポインター

概要

IEC 61131-3 規格への拡張として、ポインターを使用できます。

ポインターは、アプリケーションプログラムの実行中にアドレスを保存します。ポインターは、任意のデータ型 (*560* ページ) の変数を指すことができます。暗黙的なポインターモニターファンクションの使用については、次の *CheckPointer ファンクション* (*570* ページ) の項で詳しく説明します。

ポインター宣言の構文

<identifier>: POINTER TO <data type>;

ポインターのデリファレンスとは、現在指しているアドレスに格納されている値を取得することを意味します。ポインター識別子の後にコンテンツ演算子 ^ (ASCII キャレットおよび曲折アクセント記号) (*628* ページ) を追演加することで、ポインターをデリファレンスできます。次の例の `pt^` を参照してください。

ADR address operator (*627* ページ) を使用して、変数アドレスをポインターに割り当てることができます。

例

```
VAR
  pt:POINTER TO INT;  (* of pointer pt *)
var_int1:INT := 5;  (* declaration of variables var_int1 and var_int2 *)
  var_int2:INT;
END_VAR
pt := ADR(var_int1); (* address of var_int1 is assigned to pointer pt *)
var_int2:= pt^;      (* value 5 of var_int1 gets assigned to var_int2 via dereferencing of pointer pt; *)
```

ファンクションポインター

EcoStruxure Machine Expert ではファンクションポインターにも対応しています。これらのポインターは外部ライブラリーに渡すことができますが、プログラミングシステムのアプリケーション内でファンクションポインターを呼び出すことはできません。CALLBACK ファンクション (SYSTEM LIBRARY ファンクション) を登録するための RUNTIME ファンクションは、ファンクションポインターを要求します。そして、登録が要求された CALLBACK に応じて、それぞれのファンクションはランタイムシステムによって暗黙的に呼び出されます (例えば、STOP 時)。システムコール (ランタイムシステム) を有効にするためには、ファンクションオブジェクトの各プロパティ (デフォルトでは、**表示 → プロパティ ... → ビルド**) を設定してください。

ADR 演算子 (*627* ページ) をファンクション名、プログラム名、ファンクションブロック名、およびメソッド名に使用できます。ファンクションはオンライン変更後に移動できるので、結果はファンクションのアドレスではなく、ファンクションへのポインターのアドレスです。このアドレスは、ファンクションが対象上に存在する限り有効です。

オンライン変更コマンドを実行することによって、アドレスの内容を変更できます。

⚠ 注意

無効なポインター

アドレスにポインターを使用し、オンライン変更コマンドを実行するときにはポインターの有効性を確認してください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

ポインターへのインデックスアクセス

IEC 61131-3 規格への拡張として、POINTER 型、STRING (*562* ページ) 型、および WSTRING (*562* ページ) 型の変数にインデックスアクセス [] が使用できます。

- pint[i] は基本データ型を返します。
- ポインターへのインデックスアクセスは算術演算です。
  インデックスアクセスがポインター型の変数で使用されている場合、オフセット pint[i] は (pint + i * SIZEOF(base type))^ に等しいです。また、インデックスアクセスはポインターに対する暗黙的デリファレンスを実行します。結果の型は、ポインターの基本型です。
  pint[7] does not equate to (pint + 7)^ を考慮してください。
- インデックスアクセスが STRING 型の変数で使用されている場合、結果はオフセット index-expr にある文字になります。結果は BYTE 型です。str[i] は文字列の i 番目の文字を SINT (ASCII) で返します。
- インデックスアクセスが WSTRING 型の変数で使用されている場合、結果はオフセット index-expr にある文字になります。結果は WORD 型です。wstr[i] は文字列の i 番目の文字を INT (Unicode) で返します。

注記：リファレンス (*568* ページ) も使用できます。ポインターとは異なり、リファレンスは値に直接影響します。

CheckPointer ファンクション

ランタイム中にポインターをモニターするには、暗黙的にモニターする CheckPointer ファンクションを使用できます。必要に応じて調整できます。これを実現するには、**暗黙的チェック用 POU** オブジェクト (*153* ページ) をアプリケーションに追加してください。**Pointer Checks** カテゴリーに関連するチェックボックスを有効にします。

注記：渡されたポインターが有効なメモリーアドレスを参照しているかどうか、および参照されているメモリー領域の位置合わせがポインターが指す変数のデータ型に適合するかどうかを確認するために、機器の試運転中に CheckPointer ファンクションを実装する必要があります。

CheckPointer は REFERENCE TO タイプの変数を同様に監視します。

注記：THIS ポインターの暗黙的なチェックファンクションの呼び出しはありません。

テンプレート

宣言部 :

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

呼び出されると、次の入力パラメーターがファンクションに提供されます。

- ptToTest: ポインターの対象アドレス
- iSize: 参照される変数のサイズ。iSize のデータ型は整数と互換性があり、ポインターアドレスに格納される可能性のある最大データサイズより大きくしてください。
- iGran: アクセス粒度は参照される変数で使用される最大の非構造体データ型です。iGran のデータ型は整数との互換性が必要です。
- bWrite: アクセスのタイプ (TRUE= 書き込みアクセス、FALSE= 読み込みアクセス)。bWrite のデータ型は BOOL にしてください。

PacDrive コントローラー用の CheckPointer ファンクションの実装

PacDrive コントローラー (PacDrive LMC Eco / PacDrive LMC Pro/Pro2) 用の CheckPointer ファンクション の実装は、システムランタイム例外を生成し、メモリーアドレスまたはアライメントが無効な場合にメッセージロガーに呼び出しスタックを書き込むことです。

実装部 :

```
CheckPointer := ptToTest;
IF ptToTest = 0 THEN
    FC_DiagMsgWrite(4, 'CP = 0');
    FC_SysUserCallStack(0);
ELSE
    CheckPointer := ptToTest;
END_IF
```

オプティマイズコントローラー用の CheckPointer ファンクションの実装

オプティマイズコントローラー (例えば、Modicon M241 Logic Controller) 用の CheckPointer ファンクションの実装は、渡されたポインターを返すことです。

実装部 (未完成):

```
// No standard way of implementation. Fill your own code here
CheckPointer := ptToTest;
```

チェックの結果が正の場合、変更されていない入力ポインターが返されます (ptToTest)。

32.4
ユーザー定義データ型

このセクションについて

このセクションには次の項目が含まれています。

定義されたデータ型

概要

標準データ型に対しては、プロジェクト内で特殊なデータ型を定義することができます。

それらは、**POU ツリー**、**デバイスツリー**、または POU の宣言部分で DUT (データユニットタイプ) オブジェクトを作成することで宣言できます。

そのオブジェクトをできるだけ固有なものにするためには、オブジェクトの名前に関する推奨事項 (*483* ページ) を参照してください。

次のユーザー定義データ型を参照してください。

- 配列 (*574* ページ)
- 構造体 (*577* ページ)
- 列挙型 (*579* ページ)
- サブレンジタイプ (*581* ページ)
- リファレンス (*568* ページ)
- ポインター (*569* ページ)

配列

概要

基本データ型として、1 次元、2 次元、3 次元のフィールド (配列) に対応しています。POU の宣言部とグローバル変数リストの両方に配列を定義できます。暗黙的境界チェック (*576* ページ) も使用できます。配列は定義済みの長さ、または可変長で宣言できます。

可変長のデータ型 ARRAY は、ファンクションブロック、メソッド、およびファンクションの VAR_IN_OUT 変数に対してのみ宣言できます。この配列の下限と上限を取得するには、演算子 LOWER_BOUND(<array name>,<dim>) と UPPER_BOUND(<array name>,<dim>) を使用します。

定義済みの長さの配列宣言の構文

< 配列 _ 名 > : ARRAY [<ll1>..<ul1>,<ll2>..<ul2>,<ll3>..<ul3>] OF < 要素 . タイプ >

ll1, ll2, ll3 はフィールドの範囲の下限値です。

ul1, ul2 および ul3 はフィールドの範囲の上限値です。

範囲の値は整数型にしてください。

定義済みの長さの配列宣言の例

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

可変長の配列宣言の構文

< 配列 名 > : ARRAY [*|, *|, *] OF < データ型 >;

可変長の配列宣言の例

```
FUNCTION SUM: INT; // Onedimensional
arrays of variable lengths can be
passed to this addition function.
VAR_IN_OUT
A: ARRAY [*] OF INT;
END_VAR
VAR
  i, sum2 : DINT;
END_VAR
sum2:= 0;
FOR i:= LOWER_BOUND(A,1) TO UPPER_BOUND(A,
1) // The length of the respective array is
determined.
  sum2:= sum2 + A[i];
END_FOR;
SUM:= sum2;
```

配列の初期化

配列の初期化の例

```
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 *)
```

構造体配列の初期化の例

構造体の定義

```
TYPE STRUCT1
STRUCT
 p1:int;
 p2:int;
 p3:dword;
END_STRUCT
END_TYPE
```

配列の初期化

```
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];
```

値が事前に割り当てられていない要素は、基本型のデフォルトの初期値で初期化されます。前の例で、要素 arr1[3]...arr1[10] は 0 で初期化されます。

FB_Init の追加パラメーターを持つファンクションブロックの配列の初期化の例

2 つのパラメーターを持つファンクションブロック FB とメソッド FB_Init の例

```
FUNCTION_BLOCK FB
VAR
   _nId : INT;
   _lrIn : LREAL;
END_VAR
METHOD FB_Init : BOOL
VAR_INPUT
   bInitRetains : BOOL;
   bInCopyCode : BOOL;
   nId : INT;
   lrIn : LREAL;
END_VAR
_nId := nId;
_lrIn := lrIN;
```

初期化を伴う配列宣言の例

```
PROGRAM PLC_PRG
VAR
  inst : FB(nId := 11, lrIn := 33.44);
   ainst : ARRAY [0..1, 0..1] OF FB[(nId :=
12, lrIn := 11.22), (nId := 13, lrIn :=
22.33), (nId := 14, lrIn := 33.55),(nId := 15,
lrIn := 11.22)];
END_VAR
```

配列要素へのアクセス

2 次元配列では、次のように要素にアクセスします。

< 配列名 >[Index1,Index2]

例

Card_game [9,2]

配列範囲のチェックファンクション

ランタイム中に配列要素プロパティにアクセスするためには、CheckBounds ファンクションを有効にする必要があります。ファンクションの挿入の詳細については、**暗黙的チェック用 POU** ファンクション (*153* ページ) の説明を参照してください。

このチェックファンクションは、適切な方法 (例えば、検出されたエラーフラグを設定することや、インデックスを調整すること) によって境界違反を処理する必要があります。ファンクションは ARRAY 型の変数が割り当てられるとすぐに暗黙的に呼び出されます。

CheckBounds ファンクションの使用の例

チェックファンクションのデフォルト実装は次のとおりです。

宣言部 :

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
index, lower, upper: DINT;
END_VAR
```

実装部 :

```
// Implicitly generated code : Only an Implementation suggestion
IF  index < lower THEN
CheckBounds := lower;
ELSIF  index > upper THEN
CheckBounds := upper;
ELSE
CheckBounds := index;
END_IF
```

呼び出されると、ファンクションは次の入力パラメーターを取得します。

- index: フィールド要素インデックス
- lower: フィールド範囲の下限値
- upper: フィールド範囲の上限値

インデックスが範囲内にある限り、戻り値はインデックスの値になります。それ以外の場合、範囲違反に応じて、範囲の上限値または下限値のいずれかが返されます。

配列 a の上限値を超過

次の例では、配列 a の上限値を超えています。

```
PROGRAM PLC_PRG
VAR
 a: ARRAY[0..7] OF BOOL;
   b: INT:=10;
END_VAR
a[b]:=TRUE;
```

この例では、代入の前に暗黙的に呼ばれた CheckBounds ファンクションが、インデックスの値を 10 から上限値の 7 に変更します。そのため、値 TRUE は配列の要素 a[7] に代入されます。これは、配列範囲外のアクセスを CheckBounds ファンクションで修正する方法です。

構造体

概要

オブジェクトの追加ダイアログボックスで、DUT (データタイプユニット) オブジェクトとしてプロジェクト内で構造体を作成します。

キーワード TYPE と STRUCT で始まり、END_STRUCT と END_TYPE で終わります。

構文

```
TYPE <structurename>:
STRUCT
   <declaration of variables 1>
   ...
   <declaration of variables n>
END_STRUCT
END_TYPE
```

<structurename> はプロジェクト全体を通して認識され、標準データ型のように使用できる型です。

入れ子構造が許可されています。唯一の制限は、変数にアドレスを割り当てられないことです (AT 宣言ができません)。

例

Polygonline という名前の構造体の定義の例 :

```
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

例

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

変数による初期化はできません。構造体の配列の初期化の例については、*配列* (*574* ページ) を参照してください。

構造体の要素へのアクセス

次の構文を使用して、構造体の要素にアクセスできます。

< 構造体の名前 >.< コンポーネントの名前 >

前の構造体 Polygonline の例では、要素 Start に Poly_1.Start でアクセスできます。

構造体内のビットアクセス

データ型 BIT (*567* ページ) は構造体内だけで宣言できる特殊なデータ型です。BIT は 1 ビットのメモリー領域を消費し、構造体の 1 ビットを名前でアドレス指定することができます。

```
TYPE <structurename>:
STRUCT
   <bitname bit1> : BIT;
   <bitname bit2> : BIT;
   <bitname bit3> : BIT;
   ...
   <bitname bitn> : BIT;
END_STRUCT
END_TYPE
```

次の構文を使用して、構造体の要素 BIT にアクセスできます。

< 構造体の名前 >.< ビット名 >

注記 : BIT 変数ではリファレンスやポインターの使用はできません。さらに、BIT 変数は配列内で使用はできません。

構造体の比較

EQ 演算子の場合 (*624* ページ)、ターゲットシステムが構造化データ型をサポートしている場合は、STRUCT (構造体) 型のオペランドを比較することができます。

列挙型

概要

列挙型は、カンマで区切られた文字列定数で構成されたユーザー定義の型です。これらの定数は、列挙値と呼ばれます。列挙値は、プロジェクト内のグローバル定数の識別子です。

列挙値が POU 内で宣言されていても、プロジェクトのすべての領域でグローバルに認識されます。

列挙型は **オブジェクトの追加** ダイアログボックスで、DUT オブジェクトとしてプロジェクト内に作成されます。

注記： ローカル列挙宣言ができるのは、TYPE 内だけです。

列挙型を宣言するための構文

TYPE <enum identifier>: (<enum_0>|:=<value>,<enum_1>|:=<value>, ...,<enum _n>|:=<value>)|<base data type>|:=<default value>;
END_TYPE

以下の要素を追加で使用することができます。

- 単一列挙値の初期化
- 〈base data type〉の定義 (*IEC 61131-3 規格の 2 番目の拡張*参照 (*580* ページ))。
- すべてのコンポーネントを初期化するためのデフォルト値の定義

列挙型を使用して変数を宣言するための構文

<variable identifier> : <enum identifier>| := <initialization value>

A variable of type 〈enum identifier〉型の変数は 〈enum_..〉 を取ることができます。

初期化

条件	結果
• 列挙の宣言で < デフォルト値 > を定義していない場合（以下の例を参照）列挙型の宣言で 〈default value〉 を定義していない場合 (以下の例を参照)、さらに • 〈enum identifier〉型 の変数の宣言に対して明示的な初期化値を定義していない場合	変数は最初の列挙型コンポーネントの値で初期化されます。列挙宣言内でコンポーネントが値 0 で初期化されていない限り、これが適用されます。この場合、変数も値 0 で初期化されます。

コンポーネントの明示的初期化値を使用した列挙型の例

列挙型の定義

TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10);
  (* The initial value for each of the colors is red 0, yellow 1, green 10 *)
END_TYPE

この宣言では、最初の 2 つのコンポーネントはデフォルトの初期値を取得します。

red = 0, yellow = 1

3 番目のコンポーネントの初期値は明示的に定義されています。

green = 10

実装での使用

TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=10; (* The value of the TRAFFIC_SIGNAL1 is "green" *)
FOR i:= red TO green DO
  i := i + 1;
END_FOR;

定義されたデフォルト値を使用した列挙型の例

列挙型の定義

```
TYPE COLOR :
(
   red,
   yellow,
   green
) := green;
END_TYPE
```

実装での使用

```
c1 : COLOR;
c2 : COLOR := yellow;
```

この場合、変数 c1 は値 green で初期化されます。初期値 yellow が c2 に定義されます。

IEC 61131-3 規格への第 1 拡張

列挙定数へのアクセスを明確にするために列挙型の名前を使用できます (スコープ演算子 (*674* ページ) と同様に)。

そのため、異なる列挙型で同じ定数を使用できます。

例

同じ名前のコンポーネントを持つ 2 つの列挙型の定義

```
TYPE COLORS_1: (red, blue);
END_TYPE
TYPE COLORS_2: (green, blue, yellow);
END_TYPE
```

1 つのブロック内で、異なる列挙型の同じ名前のコンポーネントの使用

```
colorvar1 : COLORS_1;
colorvar2 : COLORS_2;

(* valid: *)
colorvar1 := COLORS_1.blue;
colorvar2 := COLORS_2.blue;

(* invalid: *)
colorvar1 := blue;
colorvar2 := blue;
```

IEC 61131-3 規格への第 2 拡張

列挙型の基本データ型を明示的に指定できます。初期値は INT です。

列挙型の明示的なその他の基本データ型の例

```
TYPE COLORS_2 : (yellow, blue, green:=16#8000)DINT;
END_TYPE
```

注記：属性 strict は TYPE 宣言の上の行でプロジェクトに追加した各列挙型に自動的に割り当てられます。これにより、次の場合にコンパイル処理中にエラーが検出されます。

- 列挙型の変数による算術演算
- 列挙型の変数への定数値の割り当て。定数は列挙値に対応しません。
- 列挙型の変数への非定数値の代入。非定数は列挙型以外のデータ型を持ちます。

属性を明示的に追加または削除できます。

構文 : {attribute 'strict'}

サブレンジ型

概要

サブレンジ型は、値の範囲が基本データ型のサブセットになるユーザー定義の型 (*573* ページ) です。暗黙的な境界範囲チェック (*581* ページ) も使用できます。

宣言は DUT オブジェクトで行うことができますが、サブレンジ型で変数を直接宣言することもできます。

構文

DUT オブジェクトとしての宣言の構文 :

TYPE <name>: <Inttype> (<ug>..<og>) END_TYPE;

<name>	有効な IEC 識別子
<inttype>	次のデータ型のいずれか。SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD)
<ug>	基本型と互換性のある定数。範囲の下限を設定します。 その下限自体も範囲に含まれます。
<og>	基本型と互換性のある定数。範囲の上限を設定します。 その上限自体も範囲に含まれます。

例

```
TYPE
 SubInt : INT (-4095..4095);
END_TYPE
```

サブレンジ型の変数の直接宣言

```
VAR
 i : INT (-4095..4095);
 ui : UINT (0..10000);
END_VAR
```

サブレンジ型 (宣言または実装の) に代入される値が範囲と一致しない場合 (例えば、上の宣言の例では i:=5000)、メッセージが発行されます。

範囲のチェックファンクション

ラインタイム中にサブレンジ型の範囲制限の確認をするためには、CheckRangeSigned,、CheckLRangeSigned または、CheckRangeUnsigned、CheckLRangeUnsigned ファンクションををアプリケーションで利用できるようにする必要があります。ファンクションの挿入の詳細については、**暗黙的チェック用 POU** ファンクション (*153* ページ) の説明を参照してください。

このチェックファンクションの目的は、サブレンジの違反を適切に処理することです (例えば、エラーフラグを設定するか値を変更するなど)。このファンクションは、サブレンジ型の変数が割り当てられるとすぐに暗黙的に呼び出されます。

⚠ 警告

装置の意図しない動作

暗黙的チェックファンクションの宣言部分は変更しないでください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

例

符号付サブレンジ型に属する変数の割り当てには、CheckRangeSigned への暗黙の呼び出しが必要です。値を許容範囲にトリミングするファンクションのデフォルトの実装は、次のように提供されます。

宣言部 :

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
value, lower, upper: DINT;
END_VAR
```

実装部 :

```
// Implicitly generated code : Only an Implementation suggestion
IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
END_IF
```

呼び出されると、ファンクションは次の入力パラメーターを取得します。

- value: 範囲の型に代入する値
- lower: 範囲の下限
- upper: 範囲の上限

代入される値が有効範囲内にある限り、ファンクションの戻り値として使用されます。それ以外の場合、範囲違反に応じて、範囲の上限または下限のいずれかが返されます。

i:=10*y は暗黙的に次のように置き換えられます。

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

例えば、y の値が 1000 である場合、変数 i に 10*1000=10000 (元の実装で提供されている) は代入されず、範囲の上限 4095 が代入されます。

CheckRangeUnsigned ファンクションも同様です。

注記：ファンクションのいずれも存在しない場合、ランタイム中にサブレンジ型の型確認は実行されません。この場合、任意の DINT/UDINT の値をサブレンジ型 DINT/UDINT の変数に割り当てることができます。任意の LINT/ULINT の値をサブレンジ型 LINT/ULINT の変数に割り当てることができます。

第 33 章
プログラミングのガイドライン

この章について

この章には次のセクションが含まれています。

33.1
命名規則

一般情報

識別子名の作成

各識別子 (基本名) は英語で関連のある短い記述にしてください。基本名は内容を表すようにします。基本名の各単語の先頭文字は大文字にします。残りの文字は小文字で表記します (例 e: FileSize)。基本名にスコープおよびプロパティを示すための接頭辞が付きます。

可能な限り、識別子は 20 文字以下にしくてください。この値はガイドラインです。必要に応じて数を上下に調整できます。

標準用語の略語 (TP、JK-FlipFlop など) を使用する場合、名前に大文字を 3 つ以上含めないでください。

大文字と小文字の区別

IEC プログラムで識別子を使用する際は読みやすさを向上させるために、特に接頭辞の大文字と小文字の区別を考慮してください。

注記：コンパイラーは大文字と小文字の区別をしません。

有効な文字

識別子には次の文字、番号、および特殊文字のみを使用してください。

0...9、A...Z、a...z

接頭辞の表示を明確にするために、区切り文字としてアンダースコアを使用します。それぞれの接頭辞セクションで構文を説明します。

基本名にアンダースコアは使用しないでください。

例

推奨されている識別子	推奨されていない識別子
diState	diSTATE
xInit	x_Init
diCycleCounter	diCyclecounter
lrRefVelocity	lrRef_Velocity
c_lrMaxPosition	clrMaxPosition
FC_PidController	FC_PIDController

33.2
接頭辞

このセクションについて

このセクションには次の項目が含まれています。

接頭辞

概要

接頭辞はファンクションで名前を割り当てるために使用されます。

次の接頭辞が使用できます。

接頭辞	用途	構文	例
スコープ接頭辞 (*588* ページ)	変数および定数のスコープ	[スコープ]_[識別子]	G_diFirstUserFault
データ型接頭辞 (*589* ページ)	変数および定数のデータ型の識別	[タイプ][識別子]	xEnable
プロパティ接頭辞 (*590* ページ)	変数および定数のプロパティの識別	[プロパティ]_[識別子]	c_iNumberOfAxes
POU 接頭辞 (*591* ページ)	POU がファンクション、ファンクションブロック、またはプログラムとして実行される場合の識別	[POU]_[識別子]	FB_VisuController
名前空間接頭辞 (*592* ページ)	ライブラリーで宣言された POU、定数、変数、およびデータ型用	[名前空間].[識別子]	TPL.G_dwErrorCode

接頭辞の順序

概要

識別子はスコープ接頭辞とタイプ接頭辞が含まれます。変数のプロパティによってプロパティ接頭辞を使用してください (例えば、定数)。追加の名前空間接頭辞はライブラリー用に使用します。

必須順序

次の順序は必須です。

スコープ][プロパティ][_][タイプ][識別子]

アンダースコア (_) で、スコープ接頭辞およびプロパティ接頭辞とタイプ接頭辞を区切ります。

例

```
Gc_dwErrorCode   : DWORD;
diCycleCounter   : DINT;
```

追加の名前空間接頭辞はライブラリーに使用します。

[名前空間].[スコープ][プロパティ][_][タイプ][識別子]

例

```
ExampleLibrary.Gc_dwErrorCode
```

独立プログラム構成単位 (POU)

アンダースコアを挿入し、プログラム構成単位 (ファンクション、ファンクションブロック、プログラム) 接頭辞と識別子を区切ります。

[POU][_][識別子]

例

```
FB_MotionCorrection
```

追加の名前空間接頭辞はライブラリー用に使用します。

[名前空間].[POU]_[識別子]

ドット (.) よって POU 接頭辞と名前空間接頭辞を区切ります。

例

```
ExampleLibrary.FC_SetError()
```

依存プログラム構成単位 (POU)

メソッド、アクション、およびプロパティは依存 POU と見なされます。これらは独立 POU より下のレベルで使用されます。

メソッドおよびアクションには接頭辞がありません。

プロパティには戻り値のタイプ接頭辞が付きます。

例

```
PROPERTY lrVelocity : LREAL
```

スコープ接頭辞

概要

スコープ接頭辞は変数と定数のスコープを示します。ローカル変数、グローバル変数、または定数のいずれかになります。

グローバル変数は大文字 G_ で示され、グローバル定数にはプロパティ接頭辞 c が追加されます。(どちらも続けてアンダースコアを付けます)。

注記：さらに、ライブラリーのグローバル変数と定数をライブラリーの名前空間で識別します。

スコープ接頭辞	タイプ	用途	例
接頭辞なし	VAR	ローカル変数	xEnable
G_	VAR_GLOBAL	グローバル変数	G_diFirstUserFault
Gc_	VAR_GLOBAL CONSTANT	グローバル定数	Gc_dwErrorCode

例

```
VAR_GLOBAL CONSTANT
   Gc_dwExample : DWORD := 16#0000001A;
END_VAR
```

名前空間 INF でライブラリーのグローバル変数へアクセス :

```
INF.G_dwExample := 16#0000001A;
```

データ型接頭辞

標準データ型

データ型接頭辞は変数と定数のデータ型を識別します。

注記：データ型接頭辞はポインター、参照、配列などの複合体でも構いません。ポインターまたは配列が最初に記述され、その後にポインター型または配列型の接頭辞が続きます。

IEC 61131-3 標準データ型接頭辞、また標準拡張の接頭辞が表にリストされています。

データ型接頭辞	タイプ	使用 (メモリーロケーション)	例
x	BOOL	boolean (8 ビット)	xName
by	BYTE	bit sequence (8 ビット)	byName
w	WORD	bit sequence (16 ビット)	wName
dw	DWORD	bit sequence (32 ビット)	dwName
lw	LWORD	bit sequence (64 ビット)	lwName
si	SINT	short integer (8 ビット)	siName
i	INT	integer (16 ビット)	iName
di	DINT	double integer (32 ビット)	diName
li	LINT	long integer (64 ビット)	liName
uli	ULINT	long integer (64 ビット)	uliName
usi	USINT	short integer (8 ビット)	usiName
ui	UINT	integer (16 ビット)	uiName
udi	UDINT	double integer (32 ビット)	udiName
r	REAL	floating-point number (32 ビット)	rName
lr	LREAL	doubled floating-point number (64 ビット)	lrName
dat	DATE	date (32 ビット)	datName
t	TOD	time (32 ビット)	tName
dt	DT	date and time (32 ビット)	dtName
tim	TIME	duration (32 ビット)	timName
ltim	LTIME	duration (64 ビット)	ltimName
s	STRING	character string ASCII	sName
ws	WSTRING	character string unicode	wsName
p	pointers	pointer	pxName
r	reference	reference	rxName
a	array	field	axName
e	enumeration	list type	eName
st	struct	structure	stName
if	interface	interface	ifMotion
ut	union	union	uName
fb	function block	function block	fbName

例

```
piCounter: POINTER TO INT;
aiCounters: ARRAY [1..22] OF INT;
paiRefCounter: POINTER TO ARRAY [1..22] OF INT;
apstTest   : ARRAY[1..2] OF POINTER TO ST_MotionStructure;
rdiCounter : REFERENCE TO DINT;
ifMotion   : IF_Motion;
```

プロパティ接頭辞

概要

プロパティ接頭辞は変数と定数のプロパティを識別します。

接頭辞タイプ	用途	構文	例
c_	VAR CONSTANT	ローカル定数	c_xName
r_	VAR RETAIN	保持型変数 RETAIN	r_xName
p_	VAR PERSISTENT	保持型変数 PRESISTENT	p_xName
rp_	VAR PERSISTENT	保持型変数 RETAIN PRESISTENT	rp_xName
i_	VAR_INPUT	POU の入力パラメーター	i_xName
q_	VAR_OUTPUT	POU の出力パラメーター	q_xName
iq_	VAR_IN_OUT	POU の入出力パラメーター	iq_xName
ati_	AT %IX x.y AT %IB z AT %IW k	IEC 入力領域に書き込まれる入力変数	ati_x0_0MasterEncoderInitOK
atq_	AT %QX x.y AT %QB z AT %QW k	IEC 入力領域に書き込まれる出力変数	atq_w18AxisNotDone
atm_	AT %MX x.y AT %MB z AT %MW k	IEC マーカー領域に書き込まれるマーカー変数	atm_w19ModuleNotReady

注記：

- 定数を RETAIN または PERSISTENT として宣言しないでください。
- POU 内で RETAIN 変数を宣言しないでください。これにより、RETAIN メモリー領域で POU が管理されます。

AT- 宣言された変数の例

AT- 宣言された変数の名前は対象の変数型も含みます。タイプ接頭辞のように使用されます。

```
ati_xEncoderInit AT %IX0.0 : BOOL;
atq_wAxisNotDone AT %QW18 : WORD;
atm_wModuleNotReady AT %MW19 : WORD;
```

注記：コントローラー設定 (デバイスエディター) のデバイスマッピングダイアログで変数にアドレスを割り当てることもできます。デバイスでこのダイアログが提供されいるかは、デバイスのドキュメントに記載されています。

POU 接頭辞

概要

次のプログラム構成単位 (POU) は IEC 61131-3 で定義されています。

- ファンクション
- ファンクションブロック
- プログラム
- データ構造体
- リストタイプ
- 共用体
- インターフェイス

EcoStruxure Machine Expert では作成およびテストケースの管理用に追加の POU があります。

- テストケース
- テストシリーズ
- テストリソース

識別子は POU 接頭辞と出来る限り短い名前で作成します (例えば、FB_GetResult)。変数のように、基本名の各単語の先頭文字は大文字にします。残りの文字は小文字で表記します。動詞と名詞を複合した POU 名を作成します。

接頭辞は名前の前にアンダースコアを付けて表記し、次の表に基づいた POU のタイプを識別します。

POU 接頭辞	タイプ	用途	例
SR_	PROGRAM	プログラム	SR_FlowPackerMachine
FB_	FUNCTION_BLOCK	ファンクションブロック	FB_VisuController
FC_	FUNCTION	ファンクション	FC_SetUserFault
ST_	STRUCT	データ構造体	ST_StandardModuleInterface
ET_	Enumeration	リストタイプ	ST_StandardModuleInterface
UT_	UNION	共用体	UT_Values
IF_	INTERFACE	インターフェイス	IF_CamProfile
TC_	Testcase*	テストケース	TC_MultiCam
TS_	TestSeries*	テストシリーズ	TS_Robotic
TR_	Test resources*	テストリソース	TR_Axes
*Schneider Electric ETest プラグインによって導入された IEC 識別子の拡張			

名前空間接頭辞

概要

ライブラリーの名前空間は**ライブラリーマネージャー**に表示できます。名前空間には短い頭文字 (PacDriveLib -> PDL) を使用してください。ライブラリーのデフォルト名前空間は変更しないでください。

独自に開発したライブラリー用の名前空間を予約するためには、Schneider Electric の担当者にお問い合わせください。

例

ファンクション FC_DoSomething() はライブラリー TestlibraryA (名前空間は TLA) と TestlibraryB (名前空間は TLB) にあります。それぞれのファンクションには名前空間接頭辞でアクセスします。

プロジェクトにどちらのライブラリーも存在する場合、次の呼び出しでコンパイル中にエラーが検出されます。

```
FC_DoSomething();
```

この場合、どちらの POU が呼び出されたのかを明確に定義する必要があります。

```
TLA.FC_DoSomething();
TLB.FC_DoSomething();
```

第 34 章
演算子

概要

EcoStruxure Machine Expert はすべての IEC 演算子に対応しています。標準ファンクションとは異なり、これらの演算子はプロジェクト全体を通して暗黙的に認識されます。

IEC 演算子以外に、規格では規定されていない次の演算子にも対応しています。

- ANDN
- ORN
- XORN
- SIZEOF (算術演算子 (*594* ページ) を参照してください)
- ADR
- BITADR
- コンテンツ演算子 (アドレス演算子 (*626* ページ) を参照
- 一部のスコープ演算子 (*673* ページ)

この章について

この章には次のセクションが含まれています。

34.1
算術演算子

概要

IEC1131-3 規格で規定されている次の演算子が使用できます。

- ADD
- MUL
- SUB
- DIV
- MOD
- MOVE

さらに、次の標準拡張演算子があります。

- SIZEOF

結果の値が結果変数に使用されるデータ型の範囲を超える場合に、算術演算のオーバーフローが起こる可能性があることを考慮してください。これにより、高い値ではなく低い値が機械に書き込まれるか、またはその逆が起こる可能性があります。

⚠ 警告

装置の意図しない動作

算術のオーバーフローを避けるために、算術演算で使用されるオペランドと結果を常に確認してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

このセクションについて

このセクションには次の項目が含まれています。

ADD

概要

変数の加算に使用する IEC 演算子。

使用できる型
- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME
- TIME_OF_DAY(TOD)
- DATE
- DATE_AND_TIME(DT)

TIME データ型の場合、次の組み合わせができます。
- TIME+TIME=TIME
- TOD+TIME=TOD
- DT+TIME=DT

FBD/LD エディターでは、ADD 演算子は拡張可能なボックスです。これは、一連の連結された ADD ボックスの代わりに、複数の入力がある 1 つのボックスを使用できることを意味します。入力を追加するには、**入力の挿入**コマンドを使用します。数に制限はありません。

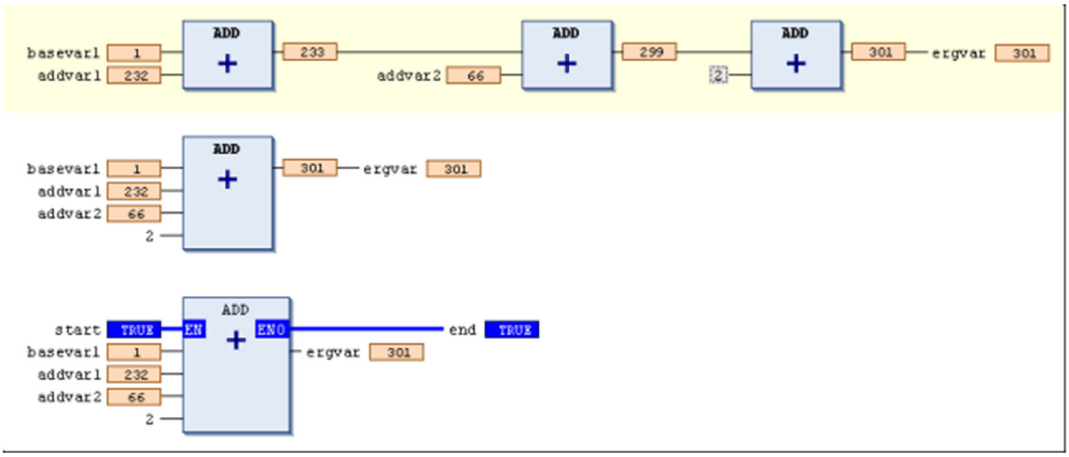
IL の例

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

ST の例

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

FBD の例



1. 一連の ADD ボックス
2. 拡張された ADD ボックス
3. EN/ENO パラメーター付き ADD ボックス

MUL

概要

変数の乗算に使用する IEC 演算子。

使用できる型
- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME

TIME 変数は整数変数で乗算することができます。

FBD/LD エディターでは、MUL 演算子は拡張可能なボックスです。これは、一連の連結された MUL ボックスの代わりに、複数の入力がある 1 つのボックスを使用できることを意味します。入力を追加するには、**入力の挿入**コマンドを使用します。数に制限はありません。

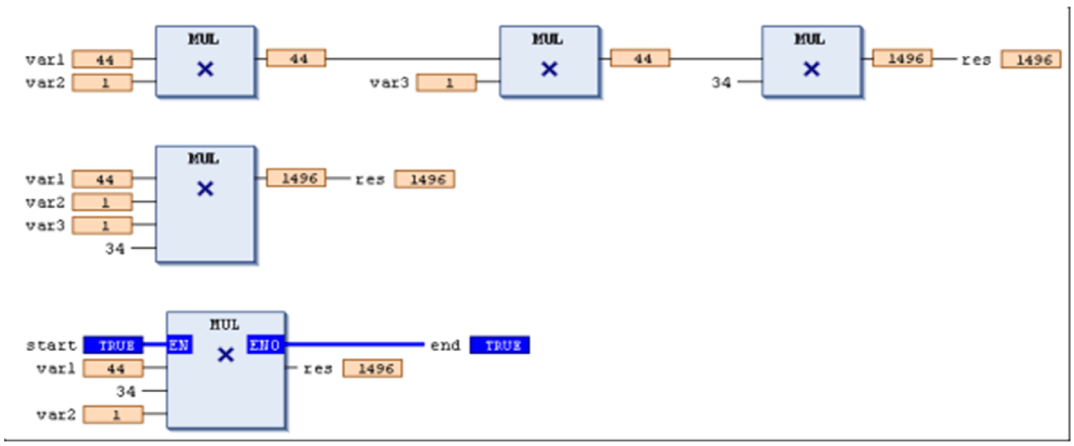
IL の例

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

ST の例

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

FBD の例



1. 一連の MUL ボックス
2. 拡張された MUL ボックス
3. EN/ENO パラメーター付き MUL ボックス

SUB

概要

変数を別の変数で減算する IEC 演算子。

使用できる型
- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME
- TIME_OF_DAY(TOD)
- DATE
- DATE_AND_TIME(DT)

TIME データ型の場合、次の組み合わせができます。
- TIME–TIME=TIME
- DATE–DATE=TIME
- TOD–TIME=TOD
- TOD–TOD=TIME
- DT-TIME=DT
- DT-DT=TIME

負の値の TIME 値は定義されないことを考慮してください。

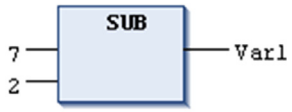
IL の例

```
LD    7
SUB   2
ST    Var1
```

ST の例

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

FBD の例



DIV

概要

変数を別の変数で除算する IEC 演算子。

使用できる型
- 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することができます。

IL の例

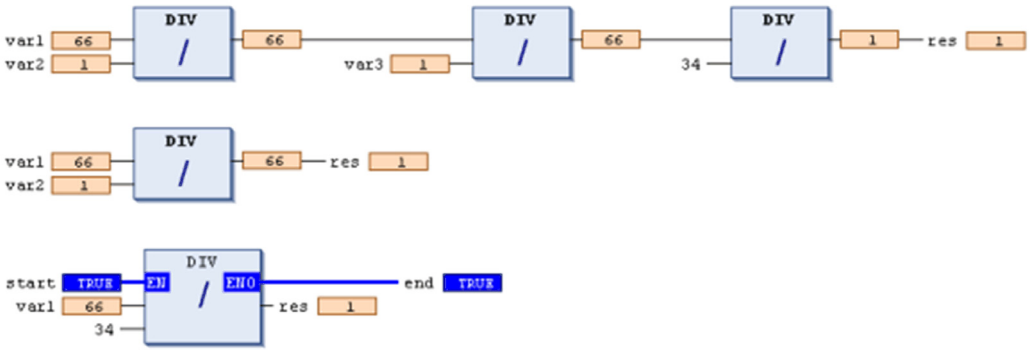
(Var1 の結果は 4 です。)

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

ST の例

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

FBD の例



1. 一連の DIV ボックス
2. 単体の DIV ボックス
3. EN/ENO パラメーター付き DIV ボックス

異なる対象システムでは、ゼロ除算エラーに関して異なる動作をする可能性があります。それによりコントローラーが HALT を引き起こしたり、検出されなくなる場合があります。

⚠ 警告

装置の意図しない動作

本書に記載されているチェックファンクションを使用するか、プログラミングコードでゼロ除算を避けるためのチェックプログラムを記述してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：暗黙的チェックファンクションについての詳細は、*暗黙的チェック用 POU* (*153* ページ) の章を参照してください。

チェックファンクション

次のチェックファンクションを使用して除数の値を確認し、0 による除算を避け、必要に応じてそれらを適合させることができます。

- CheckDivDInt
- CheckDivLint
- CheckDivReal
- CheckDivLReal

ファンクションの挿入の詳細については、**暗黙的チェック用 POU** ファンクション (*153* ページ) の説明を参照してください。

チェックファンクションは、アプリケーションコード内の各除算の前に自動的に呼び出されます。

CheckDivReal ファンクションの実装については、次の例を参照してください。

CheckDivReal ファンクションのデフォルト実装

宣言部分

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
 divisor:REAL;
END_VAR
```

実装部分 :

```
// Implicitly generated code : only an suggestion for implementation
IF divisor = 0 THEN
 CheckDivReal:=1;
ELSE
 CheckDivReal:=divisor;
END_IF;
```

DIV 演算子は除数として CheckDivReal ファンクションの出力を使用します。次の例では、0 による除算は禁止されているため、0 初期化された除数 d は、除算が実行される前に CheckDivReal により 1 に変更されます。そのため、除算の結果は 799 になります。

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL;
END_VAR
erg:= v1 / d;
```

MOD

概要

変数を別の変数で剰余演算する IEC 演算子。

使用できる型
- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT

このファンクションの結果は除算の整数の余りです。

対象システムが異なると、ゼロ除算に関して異なる動作をする可能性があります。それによりコントローラーが HALT を引き起こしたり、検出されなくなる場合があります。

⚠ 警告

装置の意図しない動作

本書に記載されているチェックファンクションを使用するか、プログラミングコードでゼロ除算を避けるためのチェックプログラムを記述してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：暗黙的チェックファンクションについての詳細は、*暗黙的チェック用 POU* (*153* ページ) の章を参照してください。

IL の例

Var1 の結果は 1 です。

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

ST の例

```
var1 := 9 MOD 2;
```

FBD の例

MOVE

概要

変数を適切なデータ型の別の変数に代入する IEC 演算子。

MOVE 演算子はすべてのデータ型が使用できます。

MOVE はグラフィックエディター FBD、LD、CFC のボックスとして利用できるため、EN/ENO (ロック解除) ファンクションを変数の代入にも適用することができます。

EN/ENO ファンクションと組み合わせた CFC の例

en_i が TRUE の場合のみ var1 が var2 に代入されます。

IL の例

結果 : var2 が var1 の値を取得します。

```
LD    var1
MOVE
ST    var2
```

次でも同じ結果が得られます。

```
LD    var1
ST    var2
```

ST の例

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

次でも同じ結果が得られます。

```
ivar2 := ivar1;
```

SIZEOF

概要

この算術演算子は、IEC 61131-3 規格で指定されていません。

この演算を使用して、指定された変数 x に必要なバイトの数を測定できます。

SIZEOF 演算子は符号なしの値を返します。戻り値の型は測定した変数 x のサイズに適合されます。

SIZEOF (x) の戻り値	測定したサイズで暗黙的に使用される定数のデータ型
0 ≤ x のサイズ < 256	USINT
256 ≤ x のサイズ < 65,536	UINT
65,536 ≤ x のサイズ < 4,294,967,296	UDINT
4,294,967,296 ≤ x のサイズ	ULINT

ST の例

```
var1 := SIZEOF(arr1);  (* d.h.: var1:=USINT#10; *)
```

IL の例

結果は 10 です。

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

34.2
ビット文字列演算子

概要

IEC1131-3 規格に対応した次のビット文字列演算子が使用できます。
- AND (*604* ページ)
- OR (*605* ページ)
- XOR (*606* ページ)
- NOT (*607* ページ)

次の演算子は規格では指定されていません。また、使用もできません。
- ANDN
- ORN
- XORN

ビット文字列演算子は、対応する 2 つまたは複数のオペランドを比較します。

このセクションについて

このセクションには次の項目が含まれています。

項目	参照ページ
AND	604
OR	605
XOR	606
NOT	607

AND

概要

ビットオペランドのビット単位の AND 演算に使用する IEC ビット文字列演算子。

入力ビットがすべてが 1 の場合、結果のビットは 1 になります。その他の場合は 0 になります。

使用できる型
- BOOL
- BYTE
- WORD
- DWORD
- LWORD

IL の例

Var1 の結果は 2#1000_0010 です。

```
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

概要

ビットオペランドのビット単位の OR 演算に使用する IEC ビット文字列演算子。

入力ビットのいずれかが 1 の場合、結果のビットは 1 になります。その他の場合は 0 になります。

使用できる型
- BOOL
- BYTE
- WORD
- DWORD
- LWORD

IL の例

var1 の結果は 2#1001_1011 です。

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

概要

ビットオペランドのビット単位の XOR 演算に使用する IEC ビット文字列演算子。

入力ビットの 1 つだけが 1 の場合、結果のビットは 1 になります。両方が 1 または 0 の場合は結果のビットは 0 になります。

使用できる型
- BOOL
- BYTE
- WORD
- DWORD
- LWORD

注記：XOR では入力を追加することができます。2 つ以上の入力がある場合、最初の 2 つの入力に対して XOR 演算が実行されます。次にその結果と 3 つ目の入力の組み合わせで XOR 演算され、同じように続きます。これは、入力数が奇数の場合に結果のビット = 1 になるという効果があります。

IL の例

結果は 2#0001_1001 です。

Var1:BYTE;

```
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

概要

ビットオペランドのビット単位の NOT 演算に使用する IEC ビット文字列演算子。

結果のビットは、対応する入力ビットが 0 の場合は 1 になり、1 の場合は 0 になります。

使用できる型
- BOOL
- BYTE
- WORD
- DWORD
- LWORD

IL の例

Var1 の結果は 2#0110_1100 です。

```
Var1:BYTE;
LD    2#1001_0011
NOT
ST    var1
```

ST の例

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

FBD の例

34.3
ビットシフト演算子

このセクションについて

このセクションには次の項目が含まれています。

SHL

概要

オペランドをビット単位で左にシフトさせる IEC 演算子。

erg:= SHL (in, n)

in: 左にシフトするオペランド

n: in を左へシフトさせるビットの数

注記：n がデータ型の幅を超える場合、BYTE、WORD、DWORD および LWORD オペランドがどのように埋められるかは対象システムにより異なります。一部はゼロ (0) で埋められ、他は n MOD 〈register width〉で埋められます。

注記：算術演算に考慮されるビットの量は、入力変数のデータ型により異なります。入力変数が定数の場合は、可能な限り小さいデータ型が考慮されます。出力変数のデータ型は、算術演算に影響しません。

例

次の例の 16 進数表記による erg_byte および erg_word の異なる結果を参照してください。in_byte および in_word の入力変数の値が同じでも、結果は入力変数のデータ型 (BYTE または WORD) によって異なります。

ST の例

```
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); (* Result is 16#14, 2#00010100 *)
erg_word:=SHL(in_word,n); (* Result is 16#0114, 2#0000000100010100 *)
```

FBD の例

IL の例

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

SHR

概要

オペランドをビット単位で右にシフトさせる IEC 演算子。

erg:= SHR (in, n)

in: 右にシフトするオペランド

n: in を右へシフトさせるビットの数

注記：n がデータ型の幅を超える場合、BYTE、WORD、DWORD および LWORD オペランドがどのように埋められるかは対象システムにより異なります。一部はゼロ (0) で埋められ、他は n MOD <register width> で埋められます。

例

次の例では、16 進数表記の入力変数 (BYTE または WORD) の型による算術演算の結果を示しています。

ST の例

```
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); (* Result is 16#11, 2#00010001 *)
erg_word:=SHR(in_word,n); (* Result is 16#0011, 2#0000000000010001 *)
```

FBD の例

IL の例

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

ROL

概要

オペランドをビット単位で左に回転させる IEC 演算子。

erg:= ROL (in, n)

使用できるデータ型
- BYTE
- WORD
- DWORD
- LWORD

in は 1 ビット左へ n 回シフトし、左端のビットが右から再挿入されます。

注記：算術演算に考慮されるビットの量は、入力変数のデータ型により異なります。入力変数が定数の場合は、可能な限り小さいデータ型が考慮されます。出力変数のデータ型は、算術演算に影響しません。

例

次の例の 16 進数表記による erg_byte および erg_word の異なる結果を参照してください。in_byte および in_word の入力変数の値が同じでも、結果は入力変数のデータ型 (BYTE または WORD) によって異なります。

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); (* Result is 16#15 *)
erg_word:=ROL(in_word,n); (* Result is 16#0114 *)
```

FBD の例

IL の例

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

ROR

概要

オペランドをビット単位で右に回転させる IEC 演算子。

erg:= ROR (in, n)

使用できるデータ型
- BYTE
- WORD
- DWORD
- LWORD

in は 1 ビットを右へ n 回シフトし、右端のビットは左から再挿入されます。

注記：算術演算のために検出されるビットの数は入力変数のデータ型により異なります。入力変数が定数の場合、可能な限り小さいデータ型が検出されます。出力変数のデータ型は、算術演算に影響しません。

例

次の例の 16 進数表記による erg_byte および erg_word の異なる結果を参照してください。in_byte および in_word の入力変数の値が同じでも、結果は入力変数のデータ型 (BYTE または WORD) によって異なります。

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); (* Result is 16#51 *)
erg_word:=ROR(in_word,n); (* Result is 16#4011 *)
```

FBD の例

IL の例

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

34.4
選択演算子

概要

選択操作は変数で実行することもできます。

本書で提供される例は、定数を演算子として使用する次のものに限定されています。

- SEL (*614* ページ)
- MAX (*615* ページ)
- MIN (*616* ページ)
- LIMIT (*617* ページ)
- MUX (*618* ページ)

このセクションについて

このセクションには次の項目が含まれています。

SEL

概要

バイナリー選択用の IEC 選択演算子。

G は IN0 または IN1 のどちらが OUT に割り当てられるかを決定します。

OUT := SEL(G, IN0, IN1) は次を意味します。

OUT := IN0; G=FALSE の場合

OUT := IN1; G=TRUE の場合

使用できるデータ型

IN0, ..., INn と OUT は同じデータ型にできます。特にユーザー定義データ型を使用してる場合は、これらの位置に同じデータ型の変数が使用されていることを確認してください。コンパイラーは型の同一性を検証し、コンパイラーエラーを返します。インタフェース変数へのファンクションブロックインスタンスの割り当てはサポートされていません。

G: BOOL

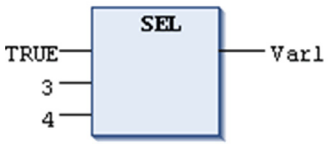
IL の例

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

ST の例

```
Var1:=SEL(TRUE,3,4); (* result is 4 *)
```

FBD の例



メモ

注記：G が TRUE の場合、IN0 より前の式は処理されません。G が FALSE の場合、IN1 より前の式は処理されません。

MAX

概要

最大値ファンクションを実行する IEC 選択演算子。

MAX 演算子は 2 つの値のうち大きい値を返します。

OUT := MAX(IN0, IN1)

IN0、IN1 および OUT は任意の変数型にできます。

IL の例

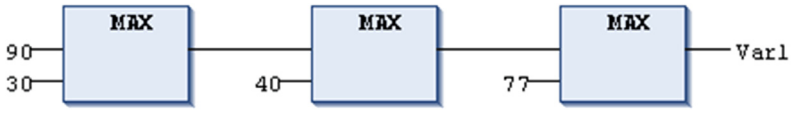
結果は 90 です。

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

ST の例

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

FBD の例



MIN

概要

最小値ファンクションを実行する IEC 選択演算子。

MIN 演算子は 2 つの値のうち小さい方を返します。

OUT := MIN(IN0, IN1)

IN0、IN1 および OUT は任意の変数型にできます。

IL の例

結果は 30 です。

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

ST の例

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

FBD の例

LIMIT

概要

制限ファンクションを実行する IEC 選択演算子。

OUT := LIMIT(Min, IN, Max) means:
OUT := MIN (MAX (IN, Min), Max)

Max は結果の上限値で、Min は下限値です。値 IN が上限値 Max を超えた場合、LIMIT は Max を返します。IN が Min を下回る場合、結果は Min になります。

IN および OUT は任意の変数型にできます。

IL の例

結果は 80 です。

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

ST の例

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

MUX

概要

多重化演算用の IEC 選択演算子。

OUT := MUX(K, IN0,...,INn)) は次を意味します。

OUT := INk

IN0, ...,INn と OUT は同じデータ型にできます。特にユーザー定義データ型を使用してる場合は、これらの位置に同じデータ型の変数が使用されていることを確認してください。コンパイラーは型の同一性を検証し、コンパイラーエラーを返します。インタフェース変数へのファンクションブロックインスタンスの割り当てはサポートされていません。

K は BYTE、WORD、DWORD、LWORD、SINT、USINT、INT、UINT、DINT、LINT、ULINT または UDINT である必要があります。

MUX は値のグループから K 番目の値を選択します。

IL の例

結果は 30 です。

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

ST の例

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

注記：INk 以外の入力より前にある式は、実行時間の削減のため処理されません。シミュレーションモードでのみ、すべての式が実行されます。

34.5
比較演算子

概要

IEC1131-3 規格に準拠した次の演算子が使用できます。

- GT (*620* ページ)
- LT (*621* ページ)
- LE (*622* ページ)
- GE (*623* ページ)
- EQ (*624* ページ)
- NE (*625* ページ)

このセクションについて

このセクションには次の項目が含まれています。

項目	参照ページ
GT	620
LT	621
LE	622
GE	623
EQ	624
NE	625

GT

概要

比較 (より大きい) ファンクションを実行する比較演算子。

GT 演算子は第 1 オペランドの値が第 2 オペランドの値より大きい場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

IL の例

結果は FALSE です。

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

ST の例

```
VAR1 := 20 > 30;
```

FBD の例

LT

概要

比較 (より小さい) ファンクションを実行する比較演算子。

LT 演算子は第 1 オペランドの値が第 2 オペランドの値より小さい場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

IL の例

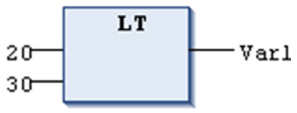
結果は TRUE です。

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

ST の例

```
VAR1 := 20 < 30;
```

FBD の例



LE

概要

比較 (以下) ファンクションを実行する比較演算子。

LE 演算子は第 1 オペランドの値が第 2 オペランドの値より小さいか等しい場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

IL の例

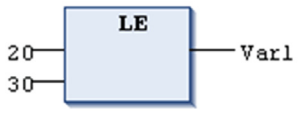
結果は TRUE です。

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

ST の例

```
VAR1 := 20 <= 30;
```

FBD の例



GE

概要

比較 (以上) ファンクションを実行する比較演算子。

GE 演算子は第 1 オペランドの値が第 2 オペランドの値より大きいか等しい場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

IL の例

結果は TRUE です。

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

ST の例

```
VAR1 := 60 >= 40;
```

FBD の例

EQ

概要

比較 (等しい) ファンクションを実行する比較演算子。

EQ 演算子はオペランドが等しい場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

IL の例

結果は TRUE です。

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

ST の例

```
VAR1 := 40 = 40;
```

FBD の例

NE

概要

比較 (等しくない) ファンクションを実行する比較演算子。

NE 演算子はオペランドが等しくない場合に TRUE を返すブール演算子です。

オペランドは任意の基本データ型にできます。

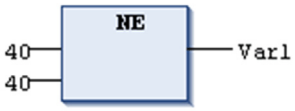
IL の例

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

ST の例

```
VAR1 := 40 <> 40;
```

FBD の例



34.6
アドレス演算子

このセクションについて

このセクションには次の項目が含まれています。

ADR

概要

このアドレス演算子は、規格 IEC 61131-3 では指定されていません。

ADR はアドレス (*692* ページ) の引数を DWORD 型で返します。このアドレスはプロジェクト内のポインター (*569* ページ) に割り当てることができます。

注記：EcoStruxure Machine Expert では、ファンクション名、プログラム名、ファンクションブロック名、およびメソッド名に ADR 演算子を使用できます。

ポインター (*569* ページ) の章を参照して、ファンクションポインターを外部ライブラリーに渡すことができることを考慮してください。ただし、EcoStruxure Machine Expert ではファンクションポインターを呼び出すことはできません。システムコール (ランタイムシステム) を有効にするためには、ファンクションオブジェクトの各オブジェクトのプロパティ (メニューの **表示 → プロパティ ... → ビルド**) を設定してください。

ST の例

```
dwVar:=ADR(bVAR);
```

IL の例

```
LD    bVar
ADR
ST    dwVar
```

オンライン変更の考慮事項

オンライン変更コマンドを実行すると変数をメモリー内の別の場所に移動できます。コピーが必要な場合は、オンライン変更中に表示されます。

変数のシフトにより、POINTER 変数が無効なメモリーを指す可能性があります。ポインターがサイクル間で保持されず、各サイクルで再割り当てされることを確認してください。

⚠ 警告

装置の意図しない動作

POU で最初に使用する前、およびそれに続くサイクルごとに POINTER TO タイプ変数の値を割り当ててください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：ファンクションおよびメソッドの POINTER TO 変数はファンクションの呼び出し元に返したり、グローバル変数に渡さないでください。

コンテンツ演算子

概要

このアドレス演算子は、規格 IEC 61131-3 では指定されていません。ポインター識別子の後にコンテンツ演算子 ^ (ASCII キャレットおよび曲折アクセント記号) を追加することで、ポインターをデリファレンスできます。

ST の例

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

オンライン変更の考慮事項

オンライン変更コマンドを実行することによって、アドレスの内容を変更できます。

⚠ 注意

無効なポインター

アドレスにポインターを使用し、オンライン変更コマンドを実行するときにはポインターの有効性を確認してください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

BITADR

概要

このアドレス演算子は、規格 IEC 61131-3 では指定されていません。

BITADR はセグメント内のビットオフセットを DWORD 型で返します。オフセット値は対象のオプション設定で**バイトアドレス指定**が有効であるかによって異なります。

DWORD の最上位のニブル (4 ビット) はメモリー領域を示します。

メモリー : 16x40000000

入力 : 16x80000000

出力 : 16xC0000000

ST の例

```
VAR
      var1 AT %IX2.3:BOOL;
      bitoffset: DWORD;
END_VAR
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 16x80000013, if byte addressing=FALSE: 1
6x80000023 *)
```

IL の例

```
LD    Var1
BITADR
ST    bitoffset
```

オンライン変更の考慮事項

オンライン変更コマンドを実行することによって、アドレスの内容を変更できます。

⚠ 注意
無効なポインター アドレスにポインターを使用し、オンライン変更コマンドを実行するときにはポインターの有効性を確認してください。 **上記の指示に従わないと、傷害または物的損害を負う可能性があります。**

34.7
呼び出し演算子

CAL

概要

ファンクションブロックまたはプログラムを呼び出す IEC 演算子。

CAL を IL 内で使用して、ファンクションブロックインスタンスの呼び出します。ファンクションブロックインスタンスの名前の直後の括弧に、入力変数として使用する変数を記述します。

例

入力変数 Par1 および Par2 がそれぞれ 0 と TRUE のファンクションブロックのインスタンス Inst を呼び出します。

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

34.8
型変換演算子

このセクションについて

このセクションには次の項目が含まれています。

データ型変換ファンクション

概要

大きい型から小さい型へ暗黙的に変換することはできません。(例えば、INT から BYTE へ、DINT から WORD へ)。これを実現するには、特殊な型変換を実行する必要があります。基本的には基本型から他の基本型に変換することができます。

構文

型変換 : <elem.type1>_TO_<elem.type2>

オーバーロード変換 : TO_<elem.type2>

注記： ...TO_STRING 変換では、文字列は左詰めで生成されます。定義されている長さが短過ぎる場合、右端が切り捨てられます。

次の型変換に対応しています。

- BOOL_TO 変換 (*633* ページ)
- TO_BOOL 変換 (*635* ページ)
- 整数型間の変換 (*637* ページ)
- REAL_TO-/ LREAL_TO 変換 (*638* ページ)
- TIME_TO/TIME_OF_DAY 変換 (*639* ページ)
- DATE_TO/DT_TO 変換 (*640* ページ)
- STRING_TO 変換 (*641* ページ)
- TRUNC (*642* ページ) (DINT へ変換)
- TRUNC_INT (*643* ページ)
- ANY_NUM_TO_< 数値データ型 >

BOOL_TO 変換

定義

BOOL 型を他の型に変換する IEC 演算子。

構文

BOOL_TO_< データ型 >

変換結果

数値型および文字列型の変換結果はオペランドの状態によって異なります。

オペランドの状態	数値型の結果	文字列型の結果
TRUE	1	TRUE
FALSE	0	FALSE

ST の例

ST の例と変換結果

例	結果
i:=BOOL_TO_INT(TRUE);	1
str:=BOOL_TO_STRING(TRUE);	'TRUE'
t:=BOOL_TO_TIME(TRUE);	T#1ms
tof:=BOOL_TO_TOD(TRUE);	TOD#00:00:00.001
dat:=BOOL_TO_DATE(FALSE);	D#1970
dandt:=BOOL_TO_DT(TRUE);	DT#1970-01-01-00:00:01

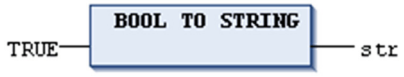
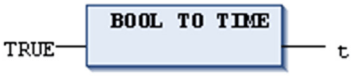
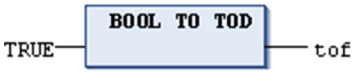
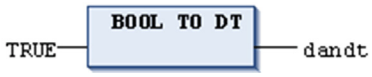
IL の例

IL の例と変換結果

例	結果
LD TRUE BOOL_TO_INT ST i	1
LD TRUE BOOL_TO_STRI... ST str	'TRUE'
LD TRUE BOOL_TO_TIME ST t	T#1ms
LD TRUE BOOL_TO_TOD ST tof	TOD#00:00:00.001
LD FALSE BOOL_TO_DATE ST dandt	D#1970-01-01
LD TRUE BOOL_TO_DT ST dandt	DT#1970-01-01-00:00:01

FBD の例

FBD の例と変換結果

例	結果
	1
	'TRUE'
	T#1ms
	TOD#00:00:00.001
	D#1970-01-01
	DT#1970-01-01-00:00:01

TO_BOOL 変換

定義

変数型を BOOL に変換する IEC 演算子。

構文

<data type>_TO_BOOL

変換結果

オペランドが 0 以外の場合、結果は TRUE となります。オペランドが 0 の場合、結果は FALSE となります。

オペランドが TRUE の場合、STRING 型の変数の結果は TRUE です。それ以外の場合は FALSE になります。

ST の例

ST の例と変換結果

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

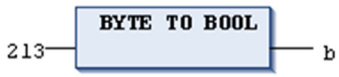
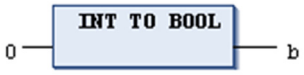
IL の例

IL の例と変換結果

例	結果
LD 213 BYTE_TO_BOOL ST b	TRUE
LD 0 INT_TO_BOOL ST b	FALSE
LD T#5ms TIME_TO_BOOL ST b	TRUE
LD 'TRUE' STRING_TO_BOOL ST b	TRUE

FBD の例

FBD の例と変換結果

例	結果
	TRUE
	FALSE
	TRUE
	TRUE

整数型間の変換

定義

整数型を別の型に変換します。

構文

<INT data type>_TO_<INT data type>

整数データ型の詳細については、*標準データ型* (*561* ページ) 章を参照してください。

変換結果

変換する数値が範囲外の場合、数値の最初のバイトは無視されます。

ST の例

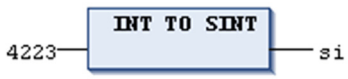
si := INT_TO_SINT(4223); (* Result is 127 *)

整数 4223 (16 進数では 16#107f) を SINT 変数として保存した場合、127 (16 進数では 16#7f) と表示されます。

IL の例

```
LD          4223
INT_TO_SINT
ST          si
```

FBD の例



REAL_TO / LREAL_TO 変換

定義

変数型 REAL または LREAL を異なる型に変換する IEC 演算子。

値は最も近い整数に切り上げまたは切り捨てられ、新しい変数型に変換されます。

次の変数型は例外です。
- STRING
- BOOL
- REAL
- LREAL

変換結果

REAL または LREAL が SINT、USINT、INT、UINT、DINT、UDINT、LINT 、または ULINT に変換され実数の値が整数の範囲外の場合、結果は定義されず、コントローラーの例外が発生する可能性があります。

注記：アプリケーションによって任意の範囲のオーバーフローを検証し、変換を実行する前に、REAL および LREAL の値が対象の整数の範囲内にあるかを確認します。

STRING 型に変換する際は、合計桁数が 16 に制限されていうことを考慮してください。(L)REAL 数字の桁数が多い場合、16 番目の数字は丸められます。STRING の長さが短く定義されている場合、右端が切り捨てられます。

ST の例

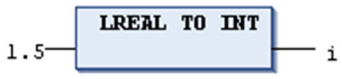
ST の例と変換結果 :

例	結果
i := REAL_TO_INT(1.5);	2
j := REAL_TO_INT(1.4);	1
i := REAL_TO_INT(-1.5);	-2
j := REAL_TO_INT(-1.4);	-1

IL の例

```
LD          2.75
REAL_TO_INT
ST          i
```

FBD の例



TIME_TO/TIME_OF_DAY 変換

定義

変数型 TIME または TIME_OF_DAY を異なる型に変換する IEC 演算子。

構文

TIME_TO_< データ型 >

TOD_TO_< データ型 >

変換結果

時間の単位はミリ秒 (午前 12:00 から始まる TIME_OF_DAY 変数) で DWORD 内部に保存されます。その後でこの値が変換されます。

STRING 型の場合、結果は時間定数です。

ST の例

ST の例と変換結果

例	結果
`str := TIME_TO_STRING(T#12ms);`	'T#12ms'
`dw := TIME_TO_DWORD(T#5m);`	300000
`si := TOD_TO_SINT(TOD#00:00:00.012);`	12

IL の例

IL の例と変換結果

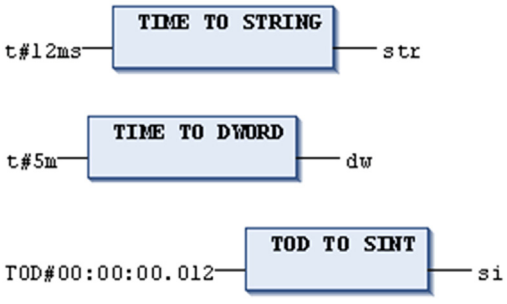
例	結果

`LD T#12ms`
`TIME_TO_STRI...`
`ST str`
	'T#12ms'

`LD T#300000ms`
`TIME_TO_DWORD`
`ST dw`
	300000

`LD TOD#00:00:00.012`
`TIME_TO_SINT`
`ST si`
	12

FBD の例



DATE_TO/DT_TO 変換

定義

変数型 DATE または DATE_AND_TIME を異なる型に変換する IEC 演算子。

構文

DATE_TO_< データ型 >

DT_TO_< データ型 >

変換結果

日付は 1970 年 1 月 1 日以降の秒単位で、DWORD 内部に格納されます。その後でこの値が変換されます。

STRING 型変数の結果は日付定数です。

ST の例

ST の例と変換結果

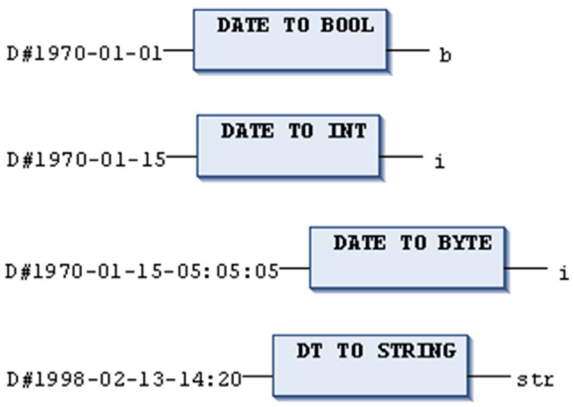
例	結果
b := DATE_TO_BOOL(D#1970-01-01);	FALSE
i := DATE_TO_INT(D#1970-01-15);	29952
byt := DT_TO_BYTE(DT#1970-01-15-05:05:05);	129
str := DT_TO_STRING(DT#1998-02-13-14:20);	'DT#1998-02-13-14:20'

IL の例

IL の例と変換結果

例	結果
LD D#1970-01-01 DATE_TO_BOOL ST b	FALSE
LD D#1970-01-01 DATE_TO_INT ST i	29952
LD D#1970-01-15-05:05: DATE_TO_BYTE ST byt	129
LD D#1998-02-13-14:20 DATE_TO_STRI... ST str	'DT#1998-02-13-14:20'

FBD の例



STRING_TO 変換

定義

変数型 STRING を異なる型に変換する IEC 演算子。

構文

STRING_TO_< データ型 >

値の指定

IEC61131-3 規格に対応した STRING 型のオペランドを指定します。値は変換後のデータ型の有効な 定数 (リテラル) (*497* ページ) に対応している必要があります。これは、指数値、無限値、接頭辞、グループ化文字 (″_″)、およびカンマの指定に適用されます。数字桁の後に文字を追加することができます。例えば、23xy。数字の前に文字は使用できません。

オペランドは変換後のデータ型の有効な値にしてください。

注記：オペランドのデータ型が対象の型に一致しない、または値が変換後のデータ型の範囲を超える場合、結果はプロセッサー型により異なるため定義されません。
大きい型から小さい型への変換は、情報の損失を引き起こす可能性があります。

⚠ 注意

データの損失

データ型が一致しない変換や、変換される値が変換後のデータ型より大きい場合は、結果がアプリケーション内で有効であることを確認してください。

上記の指示に従わないと、傷害または物的損害を負う可能性があります。

IL の例

例	変換結果
LD 'TRUE' STRING_TO_BOOL ST b	TRUE

ST の例

例	変換結果
b := STRING_TO_BOOL('TRUE');	TRUE
w := STRING_TO_WORD('abc34');	0
w := STRING_TO_WORD('34abc');	34
t := STRING_TO_TIME('T#127ms');	T#127ms
r := STRING_TO_REAL('1.234');	1.234
bv := STRING_TO_BYTE('500');	244

FBD の例

例	変換結果
	TRUE

TRUNC

定義

REAL を DINT に変換する IEC 演算子。値の整数部分が使用されます。

入力値が DINT または INT で表すことができない場合、ファンクションの結果は定義されません。そのような入力値の動作はプラットフォームによって異なります。

ST の例

ST の例と変換結果

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

IL の例

```
LD      1.9
TRUNC
ST      diVar
```

TRUNC_INT

定義

REAL を INT に変換する IEC 演算子。値の整数部分が使用されます。

入力値が DINT または INT で表すことができない場合、ファンクションの結果は定義されません。そのような入力値の動作はプラットフォームによって異なります。

ST の例

ST の例と変換結果

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

IL の例

```
LD          1.9
TRUNC_INT
ST          iVar
```

ANY_...._TO 変換

定義

任意のデータ型、具体的には数値データ型から別のデータ型への変換。どの型の変換においても、変換を成功させるためにオペランドのサイズを考慮してください。

構文

ANY_NUM_TO_< 数値データ型 >

ANY_TO_< 任意のデータ型 >

例

REAL データ型の変数を INT に変換します。

```
re : REAL := 1.234;
i : INT := ANY_TO_INT(re)
```

TO_<xxx> 変換

定義

変数の別のタイプ型への変換入力タイプを明示的に指定しないでください (オーバーロード変換処理)。

データ型変換ファンクション (*632* ページ) の説明も参照してください。

構文

TO_<data type>

ST の例

REAL データ型の変数を INT に変換します。

```
VAR
  iVar: INT;
  bVar: BOOL;
  strVar: STRING;
        rVar: REAL;
END_VAR
```

```
wVar:=TO_WORD('123')    (* result is 123 *)
bVar:=TO_BOOL(1);       (* result is TRUE *)
strVar:=TO_STRING(342); (* result is '342' *)
iVar_=TO_INT(4.22);     (* result is 4 *)
```

34.9
数値ファンクション

概要

この章では、利用可能な IEC 数値演算子の説明をします。これらの演算子は IEC 61131-3 規格で指定されています。

このセクションについて

このセクションには次の項目が含まれています。

ABS

定義

値の絶対値を返す IEC 数値演算子。

入出力は任意の数値の基本データ型にすることができます。

IL の例

i の結果は 2 です。

```
LD      –2
ABS
ST      i
```

ST の例

```
i:=ABS(–2);
```

FBD の例

SQRT

定義

値の平方根を返す IEC 数値演算子。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型にしてください。

IL の例

q の結果は 4 です。

```
LD      16
SQRT
ST      q
```

ST の例

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

FBD の例

LN

定義

基準値 10 の自然対数を返す IEC 数値演算子。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型にしてください。

IL の例

q の結果は 3.80666 です。

```
LD      45
LN
ST      q
```

ST の例

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

FBD の例

LOG

定義

基準値 10 の対数を返す IEC 数値演算子。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型にしてください。

IL の例

q の結果は 2.49762 です。

```
LD      314.5
LOG
ST      q
```

ST の例

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

FBD の例

EXP

定義

指数ファンクションを返す IEC 数値演算子。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型にしてください。

IL の例

q の結果は 7.389056099 です。

```
LD      2
EXP
ST      q
```

ST の例

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

FBD の例

SIN

定義

サインの角度を返す IEC 数値演算子。

ラジアン単位の角度を定義する入力は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型にしてください。

IL の例

q の結果は 0.479426 です。

```
LD      0.5
SIN
ST      q
```

ST の例

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

FBD の例

COS

定義

コサインの角度を返す IEC 数値演算子。

角度を分 (角度) 単位で定義する入力は、出力変数が REAL または LREAL 型である任意の数値基本データ型にできます。

IL の例

q の結果は 0.877583 です。

```
LD      0.5
COS
ST      q
```

ST の例

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

FBD の例

TAN

定義

タンジェントの値を返す IEC 数値演算子。値は角度の分単位で計算されます。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型でなければなりません。

IL の例

q の結果は 0.546302 です。

```
LD      0.5
TAN
ST      q
```

ST の例

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

FBD の例

ASIN

定義

アークサイン (サインの逆関数) の値を返す IEC 数値演算子。値は角度の分単位で計算されます。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型でなければなりません。

IL の例

q の結果は 0.523599 です。

```
LD      0.5
ASIN
ST      q
```

ST の例

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

FBD の例

ACOS

定義

アークコサイン (コサインの逆関数) の値を返す IEC 数値演算子。値は角度の分単位で計算されます。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型でなければなりません。

IL の例

q の結果は 1.0472 です。

```
LD      0.5
ACOS
ST      q
```

ST の例

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

FBD の例

ATAN

定義

アークタンジェント (タンジェントの逆関数) の値を返す IEC 数値演算子。値は角度の分単位で計算されます。

入力変数は任意の数値の基本データ型にすることができますが、出力変数は REAL または LREAL 型でなければなりません。

IL の例

q の結果は 0.463648 です。

```
LD      0.5
ATAN
ST      q
```

ST の例

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

FBD の例

EXPT

定義

他の変数によって変数をべき乗する IEC 数値演算子。

OUT = IN1 の IN2 乗

入力変数は数値の基本データ型 (SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, and LWORD) にすることができますが、出力変数は REAL または LREAL 型でなければなりません。

次の場合、この関数の結果は定義されません。
- 基準値が負である。
- 基準値が 0 または指数値が 0 以下である。

このような入力値の動作はプラットフォームによって異なります。

IL の例

結果は 49 です。

```
LD          7
EXPT          2
ST          Var1
```

ST の例

```
var1:=EXPT(7,2);
```

FBD の例

34.10
IEC 拡張演算子

このセクションについて

このセクションには次の項目が含まれています。

IEC 拡張演算子

概要

IEC 演算子に加え、EcoStruxure Machine Expert では IEC 拡張演算子にも対応しています。

- ADR (*627* ページ)
- BITADR (*629* ページ)
- SIZEOF (*602* ページ)
- __DELETE (*661* ページ)
- __ISVALIDREF (*663* ページ)
- __NEW (*664* ページ)
- __QUERYINTERFACE (*666* ページ)
- __QUERYPOINTER (*667* ページ)
- スコープ演算子 (*673* ページ)

__DELETE

定義

この演算子は、IEC 61131-3 規格では指定されていません。

互換性については、Compatibility and Migration User Guide *(EcoStruxure Machine Expert Compatibility and Migration, User Guide 参照)* の EcoStruxure Machine Expert/CoDeSys コンパイラーバージョンのマッピングテーブルを参照してください。

__DELETE 演算子は、前に __NEW 演算子 (*664* ページ) を介して割り当てられたオブジェクトのメモリーを解放します。

__DELETE に戻り値はありません。実行後はオペランドに 0 がセットされます。

アプリケーションのビルドオプション ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* (**ビュー → プロパティ ... → アプリケーションのビルドオプション**) の **動的メモリー割り当ての使用**オプションを有効にしてください。ご使用のコントローラーでこのオプションが使用可能かどうかは、各コントローラーについては、*プログラミングガイド*を参照してください。

構文

__DELETE (< ポインター >)

ポインターがファンクションブロックへのポインターである場合、ポインターに NULL がセットされる前に専用メソッド FB_Exit が呼び出されます。

注記：基本のファンクションブロックのデータ型ではなく、派生したファンクションブロックの正しいデータ型を使用してください。変数型 POINTER TO BaseFB は使用しないでください。これは、基本となるファンクションブロックが FB_Exit ファンクションを実行しない場合、__DELETE(pBaseFB) の使用後に FB_Exit が呼ばれないようするためです。

例

次の例では、ファンクションブロック FBDynamic が POU PLC_PRG の _NEW を介して直接割り当てられます。そうすることで、DUT が割り当てられている FB_Init メソッドが呼び出されます。__DELETE が PLC_PRG からのファンクションブロックポインターで呼び出されると、次に割り当てられた内部タイプを解放する FB_Exit が呼び出されます。

```
FUNCTION_BLOCK FBDynamic
VAR_INPUT
in1, in2 : INT;
END_VAR
VAR_OUTPUT
out : INT;
END_VAR
VAR
test1 : INT := 1234;
_inc : INT := 0;
_dut : POINTER TO DUT;
END_VAR
out := in1 + in2;


METHOD FB_Exit : BOOL
VAR_INPUT
bInCopyCode : BOOL;
END_VAR
__Delete(_dut);


METHOD FB_Init : BOOL
VAR_INPUT
bInitRetains : BOOL;
bInCopyCode : BOOL;
END_VAR
_dut := __NEW(DUT);
```

```
METHOD INC : INT
VAR_INPUT
END_VAR
_inc := _inc + 1;
INC := _inc;


PLC_PRG(PRG)
VAR
pFB : POINTER TO FBDynamic;
bInit: BOOL := TRUE;
bDelete: BOOL;
loc : INT;
END_VAR
IF (bInit) THEN
pFB := __NEW(FBDynamic);
bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
pFB^(in1 := 1, in2 := loc, out => loc);
pFB^.INC();
END_IF
IF (bDelete) THEN
__DELETE(pFB);
END_IF
```

__ISVALIDREF

定義

この演算子は、IEC 61131-3 規格では指定されていません。

リファレンスが有効な値を指しているかを確認することができます。

使用方法と例については、リファレンス (*568* ページ) データ型の説明を参照してください。

__NEW

定義

この演算子は、IEC 61131-3 規格では規定されていません。

__NEW 演算子は、ファンクションブロックインスタンスおよび標準データ型の配列を割り当てます。この演算子は、適切に型付けされたポインターをオブジェクトに返します。オペレーターが割り当て内で使用されていない場合は、メッセージが表示されます。

__NEW 演算子を使用するためには、**アプリケーションのビルドオプション** ビュー *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* (**ビュー → Properties... → アプリケーションのビルドオプション**) の **動的メモリー割り当ての使用**のオプションを有効にしてください。ご使用のコントローラーでこのオプションが使用可能かどうかは、各コントローラーについては、*プログラミングガイド*を参照してください。

メモリーが割り当てられていない場合、__NEW は 0 を返します。

割り当てを解除するには、__DELETE を使用してください。

構文

__NEW (< タイプ >, [< サイズ >]

この演算子は指定されたタイプ <type> の新しいオブジェクトを作成し、<type> のポインターを返します。オブジェクトの初期化は作成後に呼び出されます。0 が返された場合、この演算子は正常に完了していません。

注記：基本のファンクションブロックのデータ型ではなく、派生したファンクションブロックの正しいデータ型を使用してください。変数型 POINTER TO BaseFB は使用しないでください。これは、基本となるファンクションブロックが FB_Exit ファンクションを実行しない場合、__DELETE(pBaseFB) の使用後に FB_Exit が呼ばれないようするためです。

<type> がスカラの場合、オプションのオペランド <length> を設定してください。この演算子は設定された長さのスカラ型の配列を作成します。

例

pScalarType := __New(ScalarType, length);

注記：__NEW で作成されたファンクションブロックは固定メモリー領域を持っています。オンライン変更によってデータレイアウトを変更することはできません。したがって、新しい変数を追加または削除したり、タイプを変更したりすることはできません。

そのため、ライブラリーがないファンクションブロック (それらは変更できないため) と enable_dynamic_creation 属性をもつファンクションブロックのみが __NEW 演算子で使用できます。このフラグでファンクションブロックを変更しコードのコピーが必要になる場合、メッセージが生成されます。

注記：メモリー割り当てのコードは再入力不可能である必要があります。

セマフォ (SysSemEnter) は、同時に 2 つのタスクがメモリー割り当てすることを避けるために使用されます。そのため、__New を広範囲で使用すると大きなジッタを生成する可能性があります。

スカラ型の場合の例

```
TYPE DUT :
   STRUCT
      a,b,c,d,e,f : INT;
   END_STRUCT
END_TYPE
PROGRAM PLC_PRG
VAR
   pDut : POINTER TO DUT;
   bInit: BOOL := TRUE;
```

```
    bDelete: BOOL;
END_VAR
IF (bInit) THEN
    pDut := __NEW(DUT);
    bInit := FALSE;
END_IF
IF (bDelete) THEN
    __DELETE(pDut);
END_IF
```

ファンクションブロックの場合の例

```
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FBDynamic
VAR_INPUT
    in1, in2 : INT;
END_VAR
VAR_OUTPUT
    out : INT;
END_VAR
VAR
    test1 : INT := 1234;
    _inc : INT := 0;
    _dut : POINTER TO DUT;
END_VAR
out := in1 + in2;
```

```
PROGRAM_PLC_PRG
VAR
    pFB : POINTER TO FBDynamic;
    loc : INT;
    bInit: BOOL := TRUE;
    bDelete: BOOL;
END_VAR
```

```
IF (bInit) THEN
    pFB := __NEW(FBDynamic);
    bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
    pFB^(in1 := 1, in2 := loc, out => loc);
    pFB^.INC();
END_IF
IF (bDelete) THEN
    __DELETE(pFB);
END_IF
```

配列の場合の例

```
PLC_PRG(PRG)
VAR
    bInit: BOOL := TRUE;
    bDelete: BOOL;
    pArrayBytes : POINTER TO BYTE;
    test: INT;
    parr : POINTER TO BYTE;
END_VAR
IF (bInit) THEN
    pArrayBytes := __NEW(BYTE, 25);
    bInit := FALSE;
END_IF
IF (pArrayBytes <> 0) THEN
    pArrayBytes[24] := 125;
    test := pArrayBytes[24];
END_IF
IF (bDelete) THEN
    __DELETE(pArrayBytes);
END_IF
```

__QUERYINTERFACE

定義

この演算子は、IEC 61131-3 規格では規定されていません。

ラインタイムで、__QUERYINTERFACE によってインターフェイスリファレンスの型変換ができます。この演算子は、結果を BOOL 型で返します。TRUE は変換が正常に実行されたことを意味します。

構文

__QUERYINTERFACE(<ITF_Source>, < ITF_Dest>

第 1 オペランドは、目的のタイプのインターフェイスリファレンスまたはファンクションブロックインスタンスです。第 2 オペランドはインターフェイスリファレンスです。__QUERYINTERFACE の実行後、ITF から参照されているオブジェクトがインターフェイスを実装している場合、ITF_Dest は目的のインターフェイスへのリファレンスを保持します。この場合、変換が成功し演算子の結果は TRUE を返します。それ以外の場合、この演算子は FALSE を返します。

明示的な変換の前提条件は、ITF_Source と ITF_Dest がインターフェイス __System.IQueryInterface の拡張であることです。このインターフェイスは暗黙的に提供され、ライブラリーを必要としません。

例

ST の例

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD
INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD
INTERFACE ItfDerived2 EXTENDS ItfBase
METHOD mderived2 : BOOL
END_METHOD
FUNCTION_BLOCK FB1 IMPLEMENTS ItfDerived1
METHOD mbase : BOOL
   mbase := TRUE;
END_METHOD
METHOD mderived1 : BOOL
   mderived1 := TRUE;
END_METHOD
END_FUNCTION_BLOCK
FUNCTION_BLOCK FB2 IMPLEMENTS ItfDerived2
METHOD mbase : BOOL
   mbase := FALSE;
END_METHOD
METHOD mderived2 : BOOL
   mderived2 := TRUE;
END_METHOD
END_FUNCTION_BLOCK
PROGRAM POU
VAR
   inst1 : FB1;
   inst2 : FB2;
   itfbase1 : ItfBase := inst1;
   itfbase2 : ItfBase := inst2;
   itfderived1 : ItfDerived1 := 0;
   itfderived2 : ItfDerived2 := 0;
   bTest1, bTest2, xResult1, xResult2: BOOL;
END_VAR
xResult1 := __QUERYINTERFACE(itfbase1, itfderived1); // xResult = TRUE, itfderived1 <> 0
                              // references the instance inst1
xResult2 := __QUERYINTERFACE(itfbase1, itfderived2); // xResult = FALSE, itfderived2 = 0
xResult3 := __QUERYINTERFACE(itfbase2, itfderived1); // xResult = FALSE, itfderived1 = 0
xResult4 := __QUERYINTERFACE(itfbase2, itfderived2); // xResult = TRUE, itfderived2 <> 0
                              // references the instance inst2
```

__QUERYPOINTER

定義

この演算子は、IEC 61131-3 規格では指定されていません。

ランタイムでは、__QUERYPOINTER は型指定されていないポインターへのインターフェイスリファレンスを割り当てます。この演算子は、BOOL 型で結果を返します。TRUE は変換が正常に実行されたことを意味します。

構文

__QUERYPOINTER (<ITF_Source>, < Pointer_Dest>

第 1 オペランドは、目的のタイプのインターフェイスリファレンスまたはファンクションブロックインスタンスです。第 2 オペランドは型指定されていないポインターです。__QUERYPOINTER の実行後、Pointer_Dest は目的のインターフェイスへのリファレンスのアドレスを保持します。この場合、変換が成功し演算子の結果は TRUE を返します。それ以外の場合、この演算子は FALSE を返します。Pointer_Dest はタイプ指定されていなく、任意の型にできます。プログラマーは実際の型を必ず確認してください。例えば、インターフェイスで型コードを返すメソッドを提供するなどです。

明示的な変換の前提条件は、ITF_Source がインターフェイス __System.IQueryInterface の拡張であることです。このインターフェイスは暗黙的に提供され、ライブラリーを必要としません。

例

```
TYPE KindOfFB
   (FB1 := 1, FB2 := 2, UNKOWN := -1);
END_TYPE
INTERFACE Itf EXTENDS __System.IQueryInterface
METHOD KindOf : KindOfFB
END_METHOD
FUNCTION_BLOCK F_BLOCK_1 IMPLEMENTS ITF
METHOD KindOf : KindOfFB
   KindOf := KindOfFB.FB1;
END_METHOD
FUNCTION_BLOCK F_BLOCK_2 IMPLEMENTS ITF
METHOD KindOf : KindOfFB
   KindOf := KindOfFB.FB2;
END_METHOD
FUNCTION CAST_TO_ANY_FB : BOOL
VAR_INPUT
   itf_in : Itf;
END_VAR
VAR_OUTPUT
   pfb_1: POINTER TO F_BLOCK_1 := 0;
   pfb_2: POINTER TO F_BLOCK_2 := 0;
END_VAR
VAR
   xResult1, xResult2 : BOOL;
END_VAR
IF itf_in <> 0
   CASE itf_in.KindOf OF
      KindOfFB.FB1:
         xResult1 := __QUERYPOINTER(itf_in, pfb_1);
      KindOfFB.FB2 THEN
         xResult2 := __QUERYPOINTER(itf_in, pfb_2);
   END_CASE
END_IF
CAST_TO_ANY_FB := xResult1 OR xResult2;
```

AND_THEN

定義

この演算子は、IEC 61131-3 規格では指定されていません。構造化テキスト (ST) でのプログラミングにのみ許可されています。

AND_THEN は、短絡モードで BOOL 型と BIT 型のオペランドの AND オペレーションを実行します。これには次のような影響があります。

すべてのオペランドが TRUE の場合、オペレーションの結果は TRUE です。それ以外の場合は FALSE です。

1 つのオペランドが FALSE の場合、他の演算子の式は評価されません (遅延評価)。この点で、AND_THEN 演算子は、IEC-61131-3 規格で定義されている AND 演算子とは異なります。AND は常にすべての式を評価します (*604* ページ)。

これとは対照的に、スタンダードの IEC オペランド AND を使用すると、常にすべてのオペランドが評価されます (*604* ページ)。

OR_ELSE

定義

この演算子は、IEC 61131-3 規格では指定されていません。構造化テキスト (ST) でのプログラミングにのみ許可されています。

OR_ELSE は、短絡モードで BOOL 型と BIT 型のオペランドの AND オペレーションを実行します。これには次のような影響があります。

少なくとも 1 つのオペランドが TRUE の場合、オペレーションの結果は TRUE です。それ以外の場合は FALSE です。

1 つのオペランドが TRUE の場合、他の演算子の式は評価されません (遅延評価)。この点で、OR_ELSE 演算子は、IEC-61131-3 規格で定義されている OR 演算子とは異なります。OR は常にすべての式を評価します (*605* ページ)。

例

```
 VAR
  bEver: BOOL;
    bX1: BOOL;
  dw: DWORD := 16#000000FF;
END_VAR
bEver := FALSE;
bX := dw.8 OR_ELSE dw.1 OR_ELSE dw.1 OR_ELSE (bEver := TRUE);
```

dw.8 は FALSE、dw.1 は TRUE なので、おペレーションの結果 (bX) は TRUE です。ただし、3 番目の入力の式は実行されず、bEver は FALSE のままです。代わりにスタンダードの OR オペレーションを使用した場合 (*605* ページ)、bEver は TRUE に設定されています。

__TRY, __CATCH, __FINALLY, __ENDTRY

定義

これらの演算子は、IEC 61131-3 規格では指定されていません。

__TRY, __CATCH, __FINALLY, __ENDTRY 演算子は、IEC コードでの例外の特定の処理に使用されます。これらの演算子を使用すると、エラーが検出された場合に特定の文を実行できます。さらに、例外が検出されてもプログラムは (通常のように) 停止しません。

構文

```
__TRY
<statements_try>
__CATCH(exec)
<statements_catch>
__FINALLY
<statements_finally>
__ENDTRY
<statements_next>
```

ファンクション

<statements_try> の実行中に例外が検出された場合 (そこから呼び出されたファンクションでも)、<statements_catch> が実行されます。そのようにすると、他の例外が検出された場合のように実行は停止しません。ただし、ランタイムシステムには、範囲オフセットおよび検出された例外の種類に関する情報を含むログメッセージがあります。

<statements_catch> を実行した後、<statements_finally> が自動的に実行され (プログラムされている場合)、次に <statements_next> が実行されます。

<exception> 変数は __SystemExceptionCode タイプである必要があります。

例

__TRY のステートメントが例外を生成した場合、プログラムの実行は停止されず、__CATCH のステートメントが実行されます。これは exc ファンクションが実行されることを意味します。その後 __ENDTRY のステートメントが実行されます。

```
FUNCTION Tester : UDINT
VAR_INPUT
   count : UDINT;
END_VAR
VAR_OUTPUT
   strExceptionText : STRING;
END_VAR
VAR
   exc : __SYSTEM.ExceptionCode;
END_VAR
__TRY
Tester := tryFun(count := count, testcase := g_testcase);
//This statement is tested. If it produces an exception, then the statement in __CATCH is executed first,
and then the statement in __FINALLY.
__CATCH(exc)
HandleException(exc, strExceptionText => strExceptionText);
__FINALLY
GVL.g_count := GVL.g_count + 2;
__ENDTRY
```

プログラムされた例外処理にもかかわらずエラー位置で実行を停止するには、**Stop execution on handled exceptions** コマンド *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* を使用します。

__System.Exception タイプ

```
TYPE ExceptionCode:
( RTSEXCPT_UNKNOWN                          := 16#FFFFFFFF,
RTSEXCPT_NOEXCEPTION                        := 16#00000000,
RTSEXCPT_WATCHDOG                           := 16#00000010,
RTSEXCPT_HARDWAREWATCHDOG                   := 16#00000011,
RTSEXCPT_IO_CONFIG_ERROR                    := 16#00000012,
RTSEXCPT_PROGRAMCHECKSUM                    := 16#00000013,
RTSEXCPT_FIELDBUS_ERROR                     := 16#00000014,
RTSEXCPT_IOUPDATE_ERROR                     := 16#00000015,
RTSEXCPT_CYCLE_TIME_EXCEED                  := 16#00000016,
RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED         := 16#00000017,
RTSEXCPT_UNRESOLVED_EXTREFS                 := 16#00000018,
RTSEXCPT_DOWNLOAD_REJECTED                  := 16#00000019,
RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR  := 16#0000001A,
RTSEXCPT_LOADBOOTPROJECT_FAILED             := 16#0000001B,
RTSEXCPT_OUT_OF_MEMORY                      := 16#0000001C,
RTSEXCPT_RETAIN_MEMORY_ERROR                := 16#0000001D,
RTSEXCPT_BOOTPROJECT_CRASH                  := 16#0000001E,
RTSEXCPT_BOOTPROJECTTARGETMISMATCH          := 16#00000021,
RTSEXCPT_SCHEDULEERROR                      := 16#00000022,
RTSEXCPT_FILE_CHECKSUM_ERR                  := 16#00000023,
RTSEXCPT_RETAIN_IDENTITY_MISMATCH           := 16#00000024,
RTSEXCPT_IEC_TASK_CONFIG_ERROR              := 16#00000025,
RTSEXCPT_APP_TARGET_MISMATCH                := 16#00000026,
RTSEXCPT_ILLEGAL_INSTRUCTION                := 16#00000050,
RTSEXCPT_ACCESS_VIOLATION                   := 16#00000051,
RTSEXCPT_PRIV_INSTRUCTION                   := 16#00000052,
RTSEXCPT_IN_PAGE_ERROR                      := 16#00000053,
RTSEXCPT_STACK_OVERFLOW                     := 16#00000054,
RTSEXCPT_INVALID_DISPOSITION                := 16#00000055,
RTSEXCPT_INVALID_HANDLE                     := 16#00000056,
RTSEXCPT_GUARD_PAGE                         := 16#00000057,
RTSEXCPT_DOUBLE_FAULT                       := 16#00000058,
RTSEXCPT_INVALID_OPCODE                     := 16#00000059,
RTSEXCPT_MISALIGNMENT                       := 16#00000100,
RTSEXCPT_ARRAYBOUNDS                        := 16#00000101,
RTSEXCPT_DIVIDEBYZERO                       := 16#00000102,
RTSEXCPT_OVERFLOW                           := 16#00000103,
RTSEXCPT_NONCONTINUABLE                     := 16#00000104,
RTSEXCPT_PROCESSORLOAD_WATCHDOG             := 16#00000105,
RTSEXCPT_FPU_ERROR                          := 16#00000150,
RTSEXCPT_FPU_DENORMAL_OPERAND               := 16#00000151,
RTSEXCPT_FPU_DIVIDEBYZERO                   := 16#00000152,
RTSEXCPT_FPU_INEXACT_RESULT                 := 16#00000153,
RTSEXCPT_FPU_INVALID_OPERATION              := 16#00000154,
RTSEXCPT_FPU_OVERFLOW                       := 16#00000155,
RTSEXCPT_FPU_STACK_CHECK                    := 16#00000156,
RTSEXCPT_FPU_UNDERFLOW                      := 16#00000157,
RTSEXCPT_VENDOR_EXCEPTION_BASE              := 16#00002000
RTSEXCPT_USER_EXCEPTION_BASE                :=16#00010000 )
UDINT ; END_TYPE
```

__VARINFO

定義

この演算子は、IEC 61131-3 規格では指定されていません。

ファンクション

__VARINFO 演算子はプロジェクトの変数情報をランタイムで提供します。その情報はデータ構造体として __SYSTEM.VAR_INFO データ型の変数に格納されます。

例

ランタイムでは、MyVarInfo 変数には MyVar 変数に関する情報が含まれています。

```
//Declaration
VAR
  MyVarInfo: __SYSTEM.VAR_INFO
  MyVAR: INT;
END_VAR

//Program code
MyVarInfo:= __VARINFO (MyVar);
```

SYSTEM.VAR_INFO タイプ

__SYSTEM.VAR_INFO データ型の変数に要素が含まれます。

要素	詳細
ByteAddress	変数のアドレス。
ByteOffset	オフセット (バイト)。
Area	メモリー領域の数。
BitNr	バイト中のビットの数。 ビットタイプではない場合、値は -1 です。
BitSize	変数のサイズ (ビット)。
BitAddress	変数のビットアドレス。
TypeClass	変数のデータ型クラス。
TypeName	変数のデータ型。
NumElements	配列用 : 配列要素の数。
BaseTypeClass	配列用 : ベースデータ型のデータタイプクラス。
ElemBitSize	配列用 : 配列要素のビットサイズ。
MemoryArea	メモリー領域情報 : メモリー、入力、出力、保持、グローバル、ローカル。
Symbol	変数名。
Comment	コメント。

スコープ演算子

定義

IEC 演算子の拡張では、プロジェクトのスコープ内で変数またはモジュール名を複数回使用する場合、変数またはモジュールへのアクセスを明確にする方法がいくつかあります。

以下のスコープ演算子を使用できます。

- グローバルスコープ演算子
- グローバル変数リスト名
- 列挙型の名前
- ライブラリー名前空間
- グローバルノード演算子

グローバルスコープ演算子

ドット (.) で始まるインスタンスパスはグローバルスコープ (名前空間) を開きます。そのため、グローバル変数 .<varname> と同じ名前のローカル変数がある場合、<varname> はグローバル変数を参照します。

グローバル変数リスト名

グローバル変数リストの名前は、このリストに入っている変数の名前空間として使用できます。従って、異なるグローバル変数リストに同じ名前の変数を宣言することができ、変数名の前に <global variable list name>. を付けることで、目的の変数にアクセスができます。

構文

< グローバル変数リスト >.< 変数 >

例

グローバル変数リスト globlist1 および globlist2 はそれぞれ変数名 varx を含んでいます。次の行では、globlist2 の varx は globlist1 の varx にコピーされます。

```
globlist1.varx := globlist2.varx;
```

複数のグローバル変数リストで宣言された変数名が、グローバル変数リスト名なしで参照されている場合はメッセージが生成されます。

ライブラリー名前空間

POU へのアクセスを一意にするために、ドットで区切られた接頭辞としてライブラリーの名前空間を POU に追加できます。初期設定では、ライブラリーの namespace はライブラリー名と同じです。

例 : LIB_A.FB_A

構文

< ライブラリーの名前空間 >.< ライブラリー POU>

例

プロジェクトに含まれるライブラリーに FB_A があり、プロジェクト内でローカルに定義された FB_A もある場合は、その POU と区別するために、ライブラリーのファンクションブロックに LIB_A.FB_A という名前を割り当てることができます。

```
var1 := FB_A(in := 12); // Call of the project function block FB_A
var2 := LIB_A.FB_A(in := 22); // Call of the library function block FB_A
```

次のいずれかで別の名前を名前空間に定義することもできます。**プロジェクト情報**でライブラリープロジェクトを作成した際に、**プロジェクト情報**で定義する (**プロジェクト** メニューの初期設定)。または、後で**ライブラリーマネージャー**に含まれているライブラリーの**プロパティ**ダイアログボックスで定義する。

列挙型の名前

列挙型定数へのアクセスを明確にするために列挙型の型名を使用できます。そのため、異なる列挙型で同じ定数を使用できます。

列挙型名はドット (.) で区切られた定数名の前に付けてください。

構文

< 列挙型の名前 >.< 定数の名前 >

例

定数 Blue は列挙型 Colors と列挙型 Feelings の要素です。

```
color := Colors.Blue; // Access to enum value Blue in type Colors
feeling := Feelings.Blue; // Access to enum value Blue in type Feelings
```

グローバルノード演算子

アプリケーションツリーのグローバルノードの GVL または POU で宣言された変数には、"__POOL.". 演算子を前に付けることでアクセスできます。

34.11
初期化演算子

INI 演算子

概要

注記： INI 演算子は旧式の演算子です。FB_init メソッドが INI 演算子のの代わりになります。FB_init メソッドの詳細については、FB_init, FB_reinit メソッド (*503* ページ) 章を参照してください。ただし、EcoStruxure Machine Expert の以前のバージョンのプロジェクトとの互換性を保つために INT 演算子を引き続き使用することができます。

POU で使用されているファンクションブロックインスタンスで、RETAIN 変数を初期化するために INI 演算子を使用します。

INT 演算子をブール型変数に代入します。

構文

< ブール型変数 > := INI(<FB- インスタンス , TRUE|FALSE)

演算子の 2 番目のパラメーターを TRUE にした場合、ファンクションブロック FB 内で定義されているすべての RETAIN 変数が初期化されます。

ST の例

fbinst は、RETAIN 変数 retvar が定義されているファンクションブロック fb のインスタンスです。

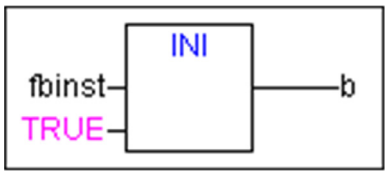
POU で宣言

```
fbinst:fb;
b:bool;
```

実装部分

```
b := INI(fbinst, TRUE);
ivar:=fbinst.retvar (* => retvar gets initialized *)
```

FBD での Call 演算子の例



第 35 章
オペランド

この章について

この章には次のセクションが含まれています。

35.1
定数

このセクションについて

このセクションには次の項目が含まれています。

BOOL 定数

概要

BOOL 定数は論理値 TRUE または FALSE です。

データ型 BOOL (*561* ページ) を参照してください。

TIME 定数

概要

TIME 定数は標準タイマーモジュールを動作させるために使用します。時定数 TIME のサイズは 32 ビット、IEC 61131-3 規格に対応しています。さらに、LTIME は規格への拡張によって高分解能タイマー用の時間ベースに対応しています。LTIME のサイズは 64 ビット、分解能はナノ秒です。

ライブラリー standard64.lib は、WSTRING 文字列用のファンクションを提供します。

TIME 定数の構文

t#< 時間の宣言 >

t# の代わりに次を使用することもできます。
- T#
- time
- TIME

TIME の宣言に次の時間単位を含むことができます。すべてを使用する必要はありませんが、次の順序で使用してください。
- d: 日付
- h: 時間
- m: 分
- s: 秒
- ms: ミリ秒

ST 代入における TIME 定数の正しい使用例

例	詳細
TIME1 := T#14ms;	–
TIME1 := T#100S12ms;	(* 最上位の要素は制限を越えてもかまいません *)
TIME1 := t#12h34m15s;	–

間違った使用例

例	詳細
TIME1 := t#5m68s;	(* 下位の要素が制限を越えています *)
TIME1 := 15ms;	(*T# がありません *)
TIME1 := t#4ms13d;	(* 入力の順序が正しくありません *)

LTIME 定数の構文

LTIME#< 時間の宣言 >

時間の宣言には、TIME 定数で使用できる時間単位の他に以下を含むことができます。
- us: マイクロ秒
- ns: ナノ秒

ST 代入における LTIME 定数の正しい使用例

LTIME1 := LTIME#1000d15h23m12s34ms2us44ns
LTIME1 := LTIME#3445343m3424732874823ns

詳細については、TIME データ型 (*563* ページ) の説明を参照してください。

DATE 定数

概要

これらの定数は日付を入力するために使用します。

構文

d#< 日付宣言 >

d# の代わりに次を使用することもできます。

- D#
- date
- DATE

日付の宣言は書式 < 年 - 月 - 日 > で入力してください。

DATE の値は、内部で 1979 年 01 月 01 日 00:00 からの秒単位の時間を含む DWORD の値として扱われます。

例

```
DATE#1996-05-06
d#1972-03-29
```

詳細については、TIME データ型 (*563* ページ) の説明を参照してください。

DATE_AND_TIME 定数

概要

DATE_AND_TIME 定数は、DATE 定数および TIME_OF_DAY 定数を組み合わせた形式です。

構文

dt#< 日付と時間宣言 >

dt# の代わりに次を使用できます。

- DT#
- date_and_time
- DATE_AND_TIME

日付と時間の宣言を書式 < 年 - 月 - 日 - 時 : 分 : 秒 > で入力します。

秒を実数として入力できます。これにより秒数を指定できます。

DATE_AND_TIME の値は、内部で 1979 年 01 月 01 日 00:00 からの秒単位の時間を含む DWORD の値として扱われます。

例

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

詳細については、TIME データ型 (*563* ページ) の説明を参照してください。

TIME_OF_DAY 定数

概要

この型の定数を使用して、1 日の時間を格納します。

構文

tod#< 時間の宣言 >

tod# の代わりに次を使用することもできます。

- TOD#
- time_of_day#
- TIME_OF_DAY#

時間の宣言を書式 < 時 : 分 : 秒 > で入力してください。

秒を実数として入力できます。これにより秒数を指定できます。

TIME_OF_DAY の値は、内部で 0:00 時間からのミリ秒単位の時間を含む DWORD の値として扱われます。

例

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

詳細については、TIME データ型 (*563* ページ) の説明を参照してください。

NUMBER 定数

概要

NUMBER の値は、2 進数、8 進数、10 進数、および 16 進数で表示されます。10 進数以外の整数値は、基数に続いて整数定数の前に数字記号 (#) を付けて表します。16 進数の数字 10...15 の値は、A...F の文字を表します。

数字にアンダースコアを含むこともできます。

例

14	(10 進数)
2#1001_0011	(2 進数)
8#67	(8 進数)
16#A	(16 進数)

これらの NUMBER の値は、次の型になります。
- BYTE
- WORD
- DWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- REAL
- LREAL

大きな変数型から小さな変数型への暗黙の型変換は許可されていません。これは、DINT 変数を INT 変数のとして使用できないことを意味します。型変換ファンクション (*632* ページ) を使用してください。

REAL/LREAL 定数

概要

REAL および LREAL 定数は小数値を指定でき、指数関数で表されます。小数点は米国の標準書式を使用してください。

例

7.4	7,4 の代わり
1.64e+009	1,64e+009 の代わり

文字列定数

概要

文字列定数は任意の文字列です。STRING (*562* ページ) 定数には一重引用符を前後に付けます。WSTRING (*562* ページ) 定数には二重引用符を前後に付けます。文字列は、ISO/IEC 8859-1 で指定されている文字セットに従ってコード化されます。空白や特殊文字 (ウムラウト記号やアクセント記号のようなさまざまな言語用の特殊文字) も入力できます。

文字列では、ドル記号 ($) とそれに続く 2 つの 16 進数の組み合わせは、ISO/IEC 8859-1 のコーディングに従って 16 進数コードとして解釈されます。コードは ASCII コードに対応しています。また、次の表に示されている特別なケースにも注意してください。

16 進数コード

ドル記号で始まる文字の組み合わせで、16 進数コードとして解釈されます。

$ コード含む文字列	解釈
'$<8-bit code>'	8 ビットコード : ISO/IEC 8859-1 に従って解釈される 2 桁の 16 進数。
'$41'	A
'$9A'	©
'$40'	@
'$0D'	制御文字 : 改行 ('$R' に対応)
'$0A'	制御文字 : ニューライン ('$L'、および '$N' に対応)

STRING の特別なケース

ドル記号で始まる文字の組み合わせで、特定の意味をもちます。

$ コード含む文字列	解釈
'$L', '$l'	制御文字 : ラインフィード ('$0A' に対応)
'$N', '$n'	制御文字 : ニューライン ('$0A' に対応)
'$P' '$p'	制御文字 : 書式送り
'$R', '$r'	制御文字 : 改行 ('$0D' に対応)
'$T', '$t'	制御文字 : タブ
'$$'	ドル記号 $
'$''	単一引用符 : '

例

STRING の定数宣言 :

```
VAR CONSTANT
   constA : STRING := 'Hello world';
   constB : STRING := 'Hello world $21'; // Hello world!
END_VAR
```

WSTRING 宣言の例 :

```
wstr:WSTRING:="This is a WString";
wstr10 : WSTRING(10) := "1234567890";
```

型属性付加定数 / 型属性付加リテラル

概要

基本的には、IEC 定数を使用すると可能な限り小さいデータ型が使用されます。ただし、LREAL が常に使用される REAL/LREAL 定数は例外です。別のデータ型を使用する必要がある場合は、定数を明示的に宣言する必要がある場合を除いて、型属性付加リテラル (型属性付加定数) を使用してください。このためには、型を決める接頭辞をもった定数にします。

構文

<Type>#<Literal>

<Type> は目的のデータ型です。使用できるデータ型は次のとおりです。

- BOOL
- SINT
- USINT
- BYTE
- INT
- UINT
- WORD
- DINT
- UDINT
- DWORD
- REAL
- LREAL

データ型は大文字で書きます。

<Literal> に定数を指定します。入力するデータは <Type> で指定したデータ型に適合させてください。

例

var1:=DINT#34;

定数をデータの損失なく対象の型に変換できない場合、メッセージが生成されます。

通常の定数が使用できる場所であれば型属性付加リテラルを使用できます。

35.2
変数

このセクションについて

このセクションには次の項目が含まれています。

変数

概要

変数は POU の宣言部分、グローバル変数リスト (GVL)、永久変数リスト、またはデバイスの I/O マッピングでローカルに宣言できます。

変数の識別子およびさまざまな使用に関しての規則を含む変数の宣言の詳細については、*変数の宣言* (*479* ページ) の章を参照してください。

変数が使用できるデータ型 (*559* ページ) によって異なります。

入力アシスタントから利用できる変数にアクセスできます。

配列、構造体、および POU のアクセス変数

次の表は配列、構造体、および POU にアクセスするための構文がリストされています。

構文	アクセス先
< 配列名 >[インデックス 1, インデックス 2]	2 次元配列 (*574* ページ) の要素
< 構造体名 >.< 変数名 >	構造体 (*577* ページ) 変数
< ファンクションブロック名 >.< 変数名 >	ファンクションブロックおよびプログラム変数

変数のビットアドレス指定

概要

整数変数では、それぞれのビットにアクセスできます。そのためには、変数にアドレス指定するビットのインデックスをドットで区切って付け加えます。ビットインデックスには任意の定数を使用できます。インデックスの基準は 0 です。

構文

< 変数名 >.< ビットインデックス >

例

```
a : INT;
b : BOOL;
...
a.2 := b;
```

変数の 3 番目のビット a に変数 b の値が代入されます。この場合、変数 a は 3 です。

変数のビット幅よりインデックスが大きい場合、次のメッセージが生成されます。

インデックス '<n>' は変数 '<var>' の有効な範囲を超えています！

次のデータ型の変数にビットアドレス指定ができます。

- SINT
- INT
- DINT
- USINT
- UINT
- UDINT
- BYTE
- WORD
- DWORD

ビットアドレスができないデータ型の場合、次のメッセージが生成されます。

直接インデックスには無効なデータ型 '<type>' です

VAR_IN_OUT 変数にビットアクセスを割り当てないでください。

グローバル定数を使用したビットアクセス

ビットインデックスを定義するグローバル定数を宣言している場合、この定数はビットアクセスに使用できます。

グローバル定数と変数によるビットアクセスの例 :

1. グローバル変数リストでグローバル定数の宣言

変数 enable でアクセスするビットを定義します :

```
VAR_GLOBAL CONSTANT
   enable:int:=2;
END_VAR
```

2. 整数変数のビットアクセス

POU で宣言 :

```
VAR
   xxx:int;
END_VAR
```

ビットアクセス :

```
xxx.enable := true; (* -> the third bit in variable xxx will be set TRUE *)
```

BIT データ型のビットアクセス

BIT データ型は構造体でのみ可能な特殊なデータ型です。詳細については、構造体のビットアクセス (*578* ページ) を参照してください。

例 : BIT データ型のビットアクセス

構造体の宣言

```
TYPE ControllerData :
STRUCT
   Status_OperationEnabled : BIT;
   Status_SwitchOnActive : BIT;
   Status_EnableOperation : BIT;
   Status_Error : BIT;
   Status_VoltageEnabled : BIT;
   Status_QuickStop : BIT;
   Status_SwitchOnLocked : BIT;
   Status_Warning : BIT;
END_STRUCT
END_TYPE
```

POU で宣言

```
VAR
   ControllerDrive1:ControllerData;
END_VAR
```

ビットアクセス

```
ControllerDrive1.Status_OperationEnabled := TRUE;
```

35.3
アドレス

直接アドレス

概要

EcoStruxure Machine Expert で指定された直接アドレスには以下の情報が含まれます。

- メモリーロケーションに関する情報
- メモリーフォーマット (サイズ)
- メモリーロケーションのオフセットオフセットは整数で指定されます。ビットアドレスの場合は、ドットとビットの位置を表す数字が続きます。

構文

%< メモリー領域接頭辞 >< サイズ接頭辞 >< 数字 |. 数字 |. 数字 ....>

次のメモリー領域接頭辞に対応しています。

I	入力 (入力ドライバーによる物理的な入力)
Q	出力 (出力ドライバーを介した物理的な出力)
M	メモリーロケーション

次のサイズ接頭辞に対応しています。

X	シングルビット
None	シングルビット
B	バイト (8 ビット)
W	バイト (16 ビット)
D	バイト (32 ビット)

例

アドレスの例	詳細
%QX7.5	出力ビット 7.5
%Q7.5	
%IW215	入力ワード 215
%QB7	出力ビット 7
%MD48	メモリーロケーション 48 のダブルワード
ivar AT %IW0: WORD;	アドレスの割り当てを含む変数宣言の例 詳細については、AT 宣言の章 (*489* ページ) を参照してください。

注記：入力、出力、およびメモリーデータ (AT %I、%Q、および %M による宣言) のメモリーサイズは、PacDrive コントローラー (PacDrive LMC Eco、PacDrive LMC Pro/Pro2) 用に、対象デバイスによって事前定義されており、アプリケーションオブジェクトのプロパティで上書きできます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。

バイトアドレス指定モードとワードアドレス指定モード

バイトアドレス指定モード、またはワードアドレス指定モードを使用するデバイス。

例

モード	例
バイトアドレス指定	ADR(%IW1) = ADR(%IB1)
ワードアドレス指定	ADR(%IW1) = ADR(%IB2)

ドットの後に続く番号であるビットアドレスの 2 番目の要素の範囲は、次のとおりです。

- バイトアドレス指定モード : 0...7
- ワードアドレス指定モード : 0...15

またビットアドレスの処理についても、デバイスを異なる設定にできます。その後、EcoStruxure Machine Expert コンパイラーによってそれに対応するように解釈されます。

例 : バイトアドレス指定のデバイスでは、ビットアドレス %IX2.5 はバイト 2 (IB2) をアドレス指定します。しかしながら、ワードアドレス指定のデバイスでは、メモリーの異なる位置を参照するワード 2 をアドレス指定します。

注記 : 明示的にシングルビットアドレスが指定されていない場合、ブール型の値はバイト単位で割り当てられます。例 : varbool1 AT %QB7 の値の変更は QX0.0 から QX0.7 、までの範囲に影響します。

35.4
ファンクション

ファンクション

概要

ST ではオペランドとしてファンクションの呼び出しを使用できます。

例

Result := Fct(7) + 3;

ファンクションとその宣言の一般的な説明については、このプログラミングガイドの *プログラム構成単位 (POU)* セクションにある *ファンクション*の説明を参照してください (*141* ページ)。

TIME() ファンクション

このファンクションはシステムが起動後に経過した時間 (ミリ秒単位) を返します。

データ型は TIME です。

IL の例

```
TIME
ST systime  (* Result for example: T#35m11s342ms *)
```

ST の例

```
systime:=TIME();
```

第 VIII 部
EcoStruxure Machine Expert テンプレート

このパートについて

このパートには次の章が含まれています。

第 36 章
テンプレートについての一般情報

36.1
EcoStruxure Machine Expert テンプレート

このセクションについて

このセクションには次の項目が含まれています。

EcoStruxure Machine Expert テンプレートについての一般情報

概要

EcoStruxure Machine Expert は、EcoStruxure Machine Expert プロジェクトで開発された専用の制御やビジュアライゼーションの機能を他の EcoStruxure Machine Expert プロジェクトで簡単に利用できるようにするためのテンプレートがあります。これらは、異なる EcoStruxure Machine Expert プロジェクトでのフィールドデバイスやアプリケーション機能の使用の標準化に役立ちます。

次の種類のテンプレートを使用できます。

- デバイステンプレート : 単一のフィールドデバイスまたは I/O モジュールに関連するテンプレート。
- ファンクションテンプレート : 上位レベルのアプリケーション機能に関連するテンプレート。

EcoStruxure Machine Expert はさまざまなテンプレートがありますが、他のプロジェクトでも使用したい機能の独自のテンプレートを作成することもできます。

独自のテンプレートの作成

EcoStruxure Machine Expert テンプレートを作成するには次の手順が必要です。

手順	手順内容
1	EcoStruxure Machine Expert プロジェクト内で機能を作成し、適切なハードウェアまたはシミュレーションでテストを行います。
2	作成した機能をテンプレートライブラリーに保存します。
3	別の EcoStruxure Machine Expert プロジェクトを開きテンプレートラブラリーからテンプレートを選択してその機能を使用できるようにします。

一般的な注意

EcoStruxure Machine Expert テンプレートを使用する際、次の点に注意してください。

- テンプレートはコントローラー固有ではないのでどのコントローラーでも使用可能にできます。テンプレート内の機能がテンプレートを追加するコントローラーで実行できるかどうか検証してください。
- テンプレートをインストールした後は、個々の要件に合わせて作成されたオブジェクトを自由に調整することができます。
- 同じコントローラーデバイスに 1 つのテンプレートを複数回設置することができます。オブジェクトを作成すると、名前の重複を避けるために名前が設置中に自動的に変更されます。詳細については *テンプレートからデバイスを追加の章のオブジェクトの命名セクション*を参照してください (*709* ページ)。
- テンプレートは変数の直接表記 (例 : %IX2.0) の使用をサポートしていません。
  不完全なアドレス指定 (例 : %I*) の変数の直接表記を使用することはできます。詳細は *変数設定 - VAR_CONFIG* (*499* ページ) の章を参照してください。

注記 : このような形式の直接アドレス指定用のプレースホルダーがありますが、プログラムでの使用は避けできるだけシンボルアドレス指定をするようにします。

対応 I/O モジュール

EcoStruxure Machine Expert テンプレートには次の I/O モジュールを入れることができます。

- TM2
- TM3
- TM5

対応フィールドバス

EcoStruxure Machine Expert テンプレートには次のフィールドバスにリンクされたフィールドデバイスを入れることができます。

- CANopen
- Modbus シリアル回線 (Modbus IOScanner)
- Modbus TCP IO Scanner
- SoftMotion General Drive Pool (LMC058)
- CANmotion
- Ethernet/IP Scanner
- Sercos3

EcoStruxure Machine Expert テンプレートの管理

概要

次の項では、デバイステンプレートまたはファンクションテンプレートを新規作成または既存のものを変更し、他のパソコンに転送するためにファイルとして保存する方法について説明します。

テンプレートライブラリー

テンプレートライブラリーには複数のデバイステンプレートまたはファンクションテンプレートがあります。

書き込み保護

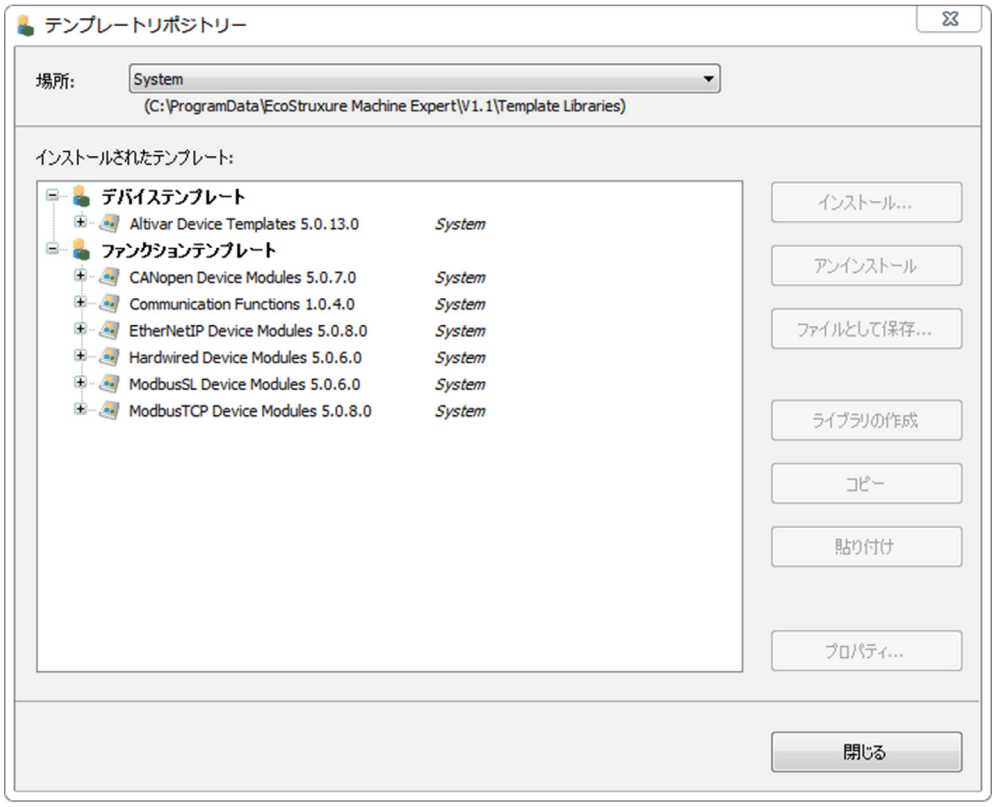
EcoStruxure Machine Expert に付属で含まれる標準テンプレートライブラリーは書き込み保護がされているため、削除または名前の変更はできません。

注記：書き込み保護のあるライブラリーを変更 (個々のテンプレートのインストールまたは名前の変更) することはできませんが、完全にアンインストールすることは可能です。

テンプレートの管理

EcoStruxure Machine Expert のデバイステンプレートとファンクションテンプレートの管理をするには、EcoStruxure Machine Expert Logic Builder の**ツール → テンプレートリポジトリー**を選択します。

テンプレートリポジトリーダイアログボックスが開きます。



場所リストから**インストールされたテンプレート**ボックスに表示するテンプレートのタイプを選択します。

- **< すべての場所 >** がデフォルトして選択されています。すべてのデバイステンプレートとファンクションテンプレートが表示されます。
- **レガシー**: EcoStruxure Machine Expert V3.1 (インストールされている場合) のデバイステンプレートとファンクションテンプレートが表示されます。
- **ユーザー** : ユーザーが作成またはインストールしたデバイステンプレートとファンクションテンプレートのみが表示されます。
- **システム** : EcoStruxure Machine Expert で納品されたデバイステンプレートとファンクションテンプレートが表示されます。

テンプレートライブラリーが格納されたディレクトリーのパスは**場所**フィールドの下に表示されます。

インストールされたテンプレートボックスはインストールされたテンプレートを 2 つのグループ (**デバイステンプレート**、**ファンクションテンプレート**) で表示します。各ライブラリーにはデバイステンプレートまたはファンクションテンプレートのどちらかを入れることができます。

追加テンプレートライブラリーのインストール

次の手順に従って追加のテンプレートライブラリーを追加します。

手順	手順内容
1	**テンプレートリポジトリー**ダイアログボックスの**インストール**ボタンをクリックします。 **結果** : **File open** ダイアログボックスが開きます。
2	インストールしたいテンプレートライブラリーのあるフォルダーを参照します。
3	インストールしたいライブラリーファイルを選択し **OK** をクリックします。 **結果** : 選択したテンプレートライブラリーがインストールされ、**テンプレートリポジトリー**ダイアログボックスにライブラリーにあるデバイステンプレートまたはファンクションテンプレートと共に表示されます。

テンプレートライブラリーの削除

次の手順に従って、テンプレートライブラリーを削除します。

手順	手順内容
1	**テンプレートリポジトリー**ダイアログボックスの**インストールされたテンプレート**リストで、削除したいテンプレートライブラリーを選択します。
2	**アンインストール**ボタンをクリックして選択したテンプレートライブラリーを削除します。 **結果** : 選択したテンプレートライブラリーが削除されます。

テンプレートライブラリーの名前の変更

次の手順に従ってテンプレートライブラリーの名前を変更します。

手順	手順内容
1	**テンプレートリポジトリー**ダイアログボックスの**インストールされたテンプレート**リストで、名前を変更したいテンプレートライブラリーを選択します。
2	変更したいテンプレートライブラリーの名前をクリックします。 **結果** : ボックスが開きます。
3	ボックスに新しい名前を入力し **Enter** を押すかボックスから離れます。 **結果** : テンプレートライブラリーに新しい名前が割り当てられます。

新規テンプレートライブラリーの作成

次の手順に従って新規テンプレートライブラリーを作成します。

手順	手順内容
1	新規テンプレートライブラリーを作成するには、**場所**リストから**ユーザー**オプションまたは **< すべての場所 >** オプションを選択します。
2	デバイステンプレート用の新規テンプレートライブラリーを作成するには、**インストールされたテンプレート**リストで **デバイステンプレート**ノードを選択し**ライブラリーの作成**ボタンをクリックします。 **結果** : **インストールされたテンプレー**リストの**デバイステンプレート**セクションの一番下にデフォルト名の新規テンプレートライブラリーが追加されます。 ファンクションテンプレートの新規テンプレートライブラリーを作成するには、**インストールされたテンプレート**リストで **ファンクションテンプレート**ノードを選択し**ライブラリーの作成**ボタンをクリックします。 **結果** : **インストールされたテンプレー**リストの**ファンクションテンプレート**セクションの一番下にデフォルト名の新規テンプレートライブラリーが追加されます。
3	上記のように新規テンプレートライブラリーの名前を変更し、例えば、下記で説明されているコピー / 貼り付け操作を使用してデバイステンプレートまたはファンクションテンプレートを入れます。

テンプレートライブラリーをファイルとして保存

デバイステンプレートまたはファンクションテンプレートを含むテンプレートライブラリーは EcoStruxure Machine Expert 固有の XML ファイルです。

次の手順に従って他のパソコンで使用できるようにします。

手順	手順内容
1	**インストールされたテンプレート**リストでエクスポートしたいテンプレートライブラリーを選択します。
2	**ファイルとして保存しています ...** ボタンをクリックします。
3	**ファイルの保存**ダイアログボックスで、テンプレートライブラリーファイルを保存したいフォルダーに移動します。
4	テンプレートライブラリーファイルを別のパソコンに転送し、**テンプレートリポジトリー**を使用してインストールします。

テンプレートライブラリーのコピー / 貼り付け操作

テンプレートリポジトリーダイアログではテンプレートライブラリーのコピー / 貼り付け操作にも対応しています。

デバイステンプレートまたはファンクションテンプレートを含むテンプレートライブラリーをコピーするには、**インストールされたテンプレート**リストで項目を選択し**コピー**ボタンをクリックします。

デバイステンプレートまたは**ファンクションテンプレート**ノードを選択し**貼り付け**ボタンをクリックすると、このテンプレートライブラリーのコピーがデフォルト名で**インストールされたテンプレート**リストに貼り付けられます。

デフォルト名を任意の名前に変更します。

テンプレートのコピー / 貼り付け操作

テンプレートリポジトリーダイアログではデバイステンプレートおよびファンクションテンプレートのコピー / 貼り付け操作にも対応しています。

デバイステンプレートまたはファンクションテンプレートをコピーするには、**インストールされたテンプレート**リストでテンプレートライブラリーノードの下にある項目を選択し**コピー**ボタンをクリックします。

ライブライリーが書き込み保護されていない場合、テンプレートをテンプレートライブラリーに貼り付けることができます。

ライブラリーは同じ種類のライブラリーにのみ貼り付けできます。

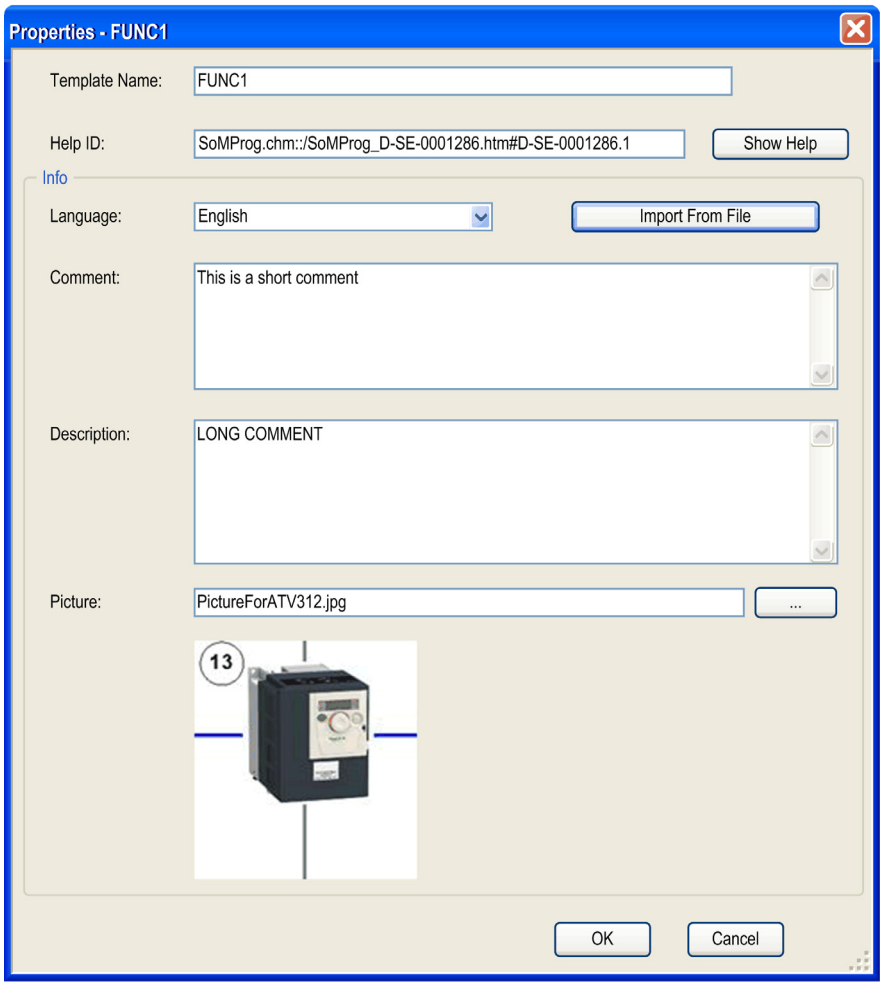
必要に応じてデフォルト名を任意の名前に変更してください。

テンプレートまたはテンプレートライブラリーの詳細情報の追加

テンプレートリポジトリーダイアログボックスでは、テンプレートまたはテンプレートライブラリーの詳細情報を入力することができます。

詳細情報を追加するには、ライブラリーまたはテンプレートライブラリーを**インストールされたテンプレート**リストから選択し**プロパティ ...** ボタンをクリックします。

選択されたライブラリーまたはテンプレートライブラリーの**プロパティ**ダイアログボックスが表示されます。



選択されたライブラリーまたはテンプレートライブラリーが書き込み禁止に設定されていない場合、**プロパティ**ダイアログボックスには以下の編集可能なパラメーターとそれに対応するボタンが表示されます。

項目	詳細
Template Name / **ライブラリ名**ボックス	適用されるライブラリーまたはテンプレートライブラリーの名前を表示します。名前を変更するには、このボックスをクリックして変更したい名前を入力します。
ヘルプ ID ボックス	シュナイダーエレクトリックのテンプレート用またはテンプレートライブラリー用には、オンラインヘルプのそれぞれの説明への参照先が表示されます。 テンプレートのオンラインヘルプドキュメントが利用できる場合は、オンラインヘルプの該当箇所への完全な参照パス、またはオンラインヘルプのインデックスに対応するキーワードを入力します。
ヘルプの表示ボタン	**ヘルプ ID** ボックスで指定されたオンラインヘルプドキュメント、または **ヘルプ ID** ボックスで指定されたキーワードで検索されるオンラインヘルプのインデックスが開きます。

項目		詳細
Info セクション		–
	言語リスト	EcoStruxure Machine Expert のグラフィックユーザーインターフェイスで使用できる言語が表示されます。言語を選択すると言語に依存する**コメント**、**説明**、および**画像**が、選択された言語で表示されます。 言語固有のコンテンツがない場合は、デフォルト言語の**英語**が表示されます。
	ファイルからインポートボタン	標準の**開く**ダイアログボックスが表示されます。言語に依存する**コメント**、**詳細**、および**画像**のローカライズされたコンテンツを含む XML ファイルを参照することができます。この XML ファイルの構成は、例 (*704* ページ) で示されている構成に従ってください。
	コメントボックス	短い文 (例えば、選択されたライブラリーまたはテンプレートライブラリーのコンテンツの概要や目的など) を入力します。この文字列は、EcoStruxure Machine Expert でテンプレートライブラリーを選択した時にツールチップとして表示されます。
	説明ボックス	長い文 (例えば、選択されたライブラリーまたはテンプレートライブラリーのコンテンツの詳細や目的など) を入力します。
	画像パラメーター **...** ボタン	言語固有の画像へのパスを入力します。 **...** ボタンをクリックして画像ファイルを参照することもできます。 サポートされている画像ファイル形式 ● ビットマップ : **.bmp* ● JPEG: **.jpg* ● Graphics interchange format (グラフィックスインターチェンジフォーマット): **.gif* ● アイコン : **.ico* 画像を指定すると、**プロパティ**ダイアログボックスに表示されます。**OK** ボタンをクリックすると、画像がテンプレートに埋め込まれます。

読み取り専用チェックボックスはテンプレートライブラリーのみに使用可能で、選択されたテンプレートライブラリーが読み取り専用状態であるかを示します。ここで、テンプレートライブラリーの状態を変更することはできません。

言語依存要素のローカライズ

以下の構成の XML ファイルをインポートすることで、言語依存の要素、**コメント**、**詳細**、および**画像**をローカライズすることができます。

```
<?xml version="1.0" encoding="UTF-8"?>
<TemplateProperties>
<HelpId>SoMProg.chm::/SoMProg_D-SE-0001286.htm#D-SE-0001286.1</HelpId>
<PropertySet languageId = "en">
<Comment>This is a short description</Comment>
<Description>This is a long description</Description>
<ImageFile>PictureEnglish.jpg</ImageFile>
</PropertySet>
<PropertySet languageId = "de">
<Comment>Kurze Beschreibung</Comment>
<Description>Lange Beschreibung</Description>
<ImageFile>PictureGerman.jpg</ImageFile>
</PropertySet>
</TemplateProperties>
```

第 37 章
デバイステンプレートの管理

37.1
デバイステンプレートの管理

このセクションについて

このセクションには次の項目が含まれています。

デバイステンプレートとは

用語の使用に関する一般情報

以下の説明では、読みやくするためにフィールドデバイスという用語のみが使用されていても、フィールドデバイスと I/O モジュールの両方に適用します。

デバイステンプレートのコンテンツ

デバイステンプレートは、特定のフィールドデバイスまたは I/O に関連しています。次の情報を含みます。

- フィールドバス設定
- 制御ロジック (コントローラープログラミング) (オプション)
- ビジュアライゼーション要素 (ビジュアライゼーションプログラミング) (オプション)

デバイステンプレートの使用

既に使用可能なデバイステンプレートはテンプレートライブラリーに保存されています。各テンプレートライブラリーには、共通のベース (例えば、モーター制御に関連する) をもつ複数のデバイステンプレートの定義が含まれています。

それらを選択して個々の EcoStruxure Machine Expert プロジェクトの要件に適合させることで、あらかじめ設定されたフィールドデバイスを新しく作成することができます。

新規バイステンプレートの作成

あらかじめ設定されたフィールドデバイスを EcoStruxure Machine Expert プロジェクトで再利用できるようにするには、そのフィールドデバイスをデバイステンプレートとして保存します。これにはフィールドデバイスにリンクされたコントローラーのプログラミングとビジュアライゼーションも含みます。

デバイステンプレートのバージョン

デバイステンプレートの作成中には、作成するデバイスのデバイスディスクリプションが実際に存在するかが検証されます。存在しない場合、最新バージョンがあれば自動的にデバイスが最新バージョンに更新されます。

テンプレートからのデバイスの追加

概要

デバイステンプレートは特定のフィールドデバイスに関連し、次の情報を含みます。

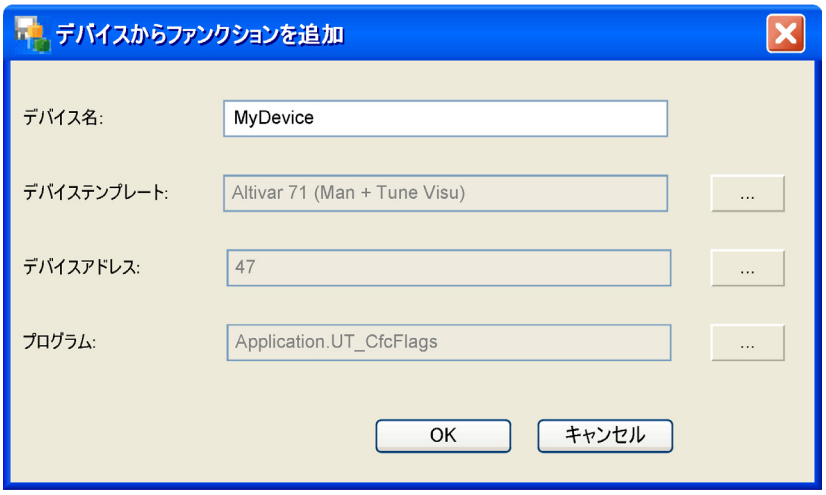
- フィールドバスデバイス設定
- 制御ロジック (コントローラープログラミング) (オプション)
- ビジュアライゼーション要素 (ビジュアライゼーションプログラミング) (オプション)

プロジェクトから独自のデバイステンプレートを作成することができます。詳細については、*デバイステンプレートの作成手順*の章 (*714* ページ) を参照してください。

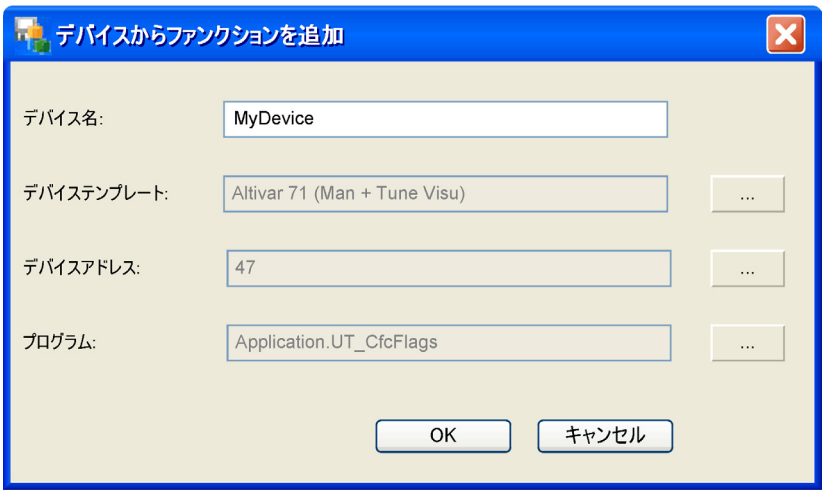
テンプレートからデバイスを追加

EcoStruxure Machine Expert では、2 種類の方法でデバイステンプレートからデバイスを追加できます。

- デバイステンプレートを使用してデバイスをドラッグ & ドロップで作成します。

手順	手順内容
1	ハードウェアカタログの**デバイス & モジュール**ビューを開きます。
2	**デバイス & モジュール**ビューで**デバイステンプレート**オプションを有効にします。 **結果** : **デバイス & モジュール**ビューに EcoStruxure Machine Expert で使用できるフィールドデバイスのテンプレートが表示されます。
3	**デバイス & モジュール**ビューのテンプレートを選択して**デバイスツリー**にドラッグし、適切なコントローラーのサブノードにドロップします。 **結果** : 適切なサブノードは EcoStruxure Machine Expert で強調表示されます。 **結果** : **テンプレートからデバイスを追加**ダイアログボックスが表示されます。 
4	**テンプレートからデバイスを追加**ダイアログボックスで**デバイス名**を設定しフィールドバスに数値アドレスが必要な場合は**デバイスアドレス**も設定します。デバイステンプレートに制御ロジックが含まれている場合は、制御ロジックが挿入されているプログラム (POU) を選択します。
5	**OK** ボタンをクリックします。 **結果** : デバイスは、選択したオプションのビジュアライゼーション画面と制御ロジックを含むデバイステンプレートに従って作成され、パラメーター化されます。

- コンテキストメニューからデバイステンプレートを使用してデバイスを作成します。

手順	手順内容
1	**デバイスツリー**を開きます。
2	フィールドデバイスマネージャーを右クリックし、コンテキストメニューから**テンプレートからデバイスを追加**を実行します。 **結果**：**テンプレートからデバイスを追加**ダイアログボックスが表示されます。 
3	**テンプレートからデバイスを追加**ダイアログボックスで使用する**デバイステンプレート**を選択、**デバイス名**を設定し、フィールドバスに数値アドレスが必要な場合は**デバイスアドレス**も設定します。デバイステンプレートに制御ロジックが含まれている場合は、制御ロジックが挿入されているプログラム (POU) を選択します。
4	**OK** ボタンをクリックします。 **結果**：デバイスは、選択したオプションのビジュアライゼーション画面と制御ロジックを含むデバイステンプレートに従って作成され、パラメーター化されます。

注記：フィールドデバイスの作成処理では元に戻す / やり直す機能は使用できません。

オブジェクトの命名

同じデバイステンプレートをベースとして異なるフィールドデバイスを作成した際に名前が重複するのを避けるために、フィールドデバイスとそれに関連するオブジェクト (FB、ビジュアライゼーション、変数) には以下の命名規則が適用されます。

元のオブジェクト名	結果
ケース 1:	
元のフィールドデバイスの名前を含む場合	オブジェクトのこの部分は新しく作成されたフィールドデバイスの名前に置き換わります。
例:	
フィールドデバイス ATV1 のデバイステンプレートに変数 Var_ATV1_Input が含まれています。	このデバイステンプレートで作成された新しいデバイス Axis1 では、それに対応して新しい変数が Var_Axis1_Input という名前になります。
ケース 2:	
元のフィールドデバイスの名前を含まない場合	新しいデバイスの名前とアンダースコアを元の名前に挿入し、固有の新しい名前を作成します。
例:	
フィールドデバイス ATV1 のデバイステンプレートに変数 Var_Input1 が含まれています。	このデバイステンプレートで作成された新しいデバイス Axis1 では、それに対応して新しい変数が Axis1_Var_Input1 という名前になります。

フィールドデバイスまたは I/O モジュールをベースにしたデバイステンプレートの作成

概要

フィールドデバイスまたは I/O モジュールをベースにデバイステンプレートを作成できます。以下の説明では、読みやくするためにフィールドデバイスという用語のみが使用されていても、フィールドデバイスと I/O モジュールの両方に適用します。

次の項では以下を説明します。

- フィールドデバイスまたは I/O モジュール (ロジックおよびビジュアライゼーションを含む) をテンプレートとして保存するための条件
- デバイステンプレートに保存される情報

フィールドデバイスの前提条件

フィールドデバイスをデバイステンプレートとして保存するには以下の条件を満たさなければなりません。

- フィールドバスは対応フィールドバスリスト (*699* ページ) にリストされたフィールドバスにリンクしている必要があります。
- デバイスタイプが**デバイスリポジトリー**にインストールされている必要があります。

I/O モジュールの前提条件

サポートされた I/O モジュールのみをデバイステンプレートとして保存できます (*699* ページ)。

アプリケーションの前提条件

正常なアプリケーションからのみテンプレートを作成できます。正常とは**ビルド**処理でエラーが検知されないという意味です。

制御ロジックをテンプレートに含める場合の前提条件

制御ロジックをテンプレートを含めるには、制御ロジックにこのフィールドデバイスとデータの交換をするコードセクションがある必要があります。この制御ロジックは実行される (タスクに追加されるか、他のプログラムから呼び出される) 必要があります。それ以外では、**ビルド**コマンドが実行される際に考慮されません。

デバイステンプレートに保存されるデバイス情報

次のフィールドバスデバイスの情報がデバイステンプレートに保存されます。

- デバイス設定
- フィールドデバイスの I/O マッピング
- フィールドデバイスに適切なビジュアライゼーション
- フィールドデバイスとデータを交換する制御ロジック

デバイステンプレートの作成に適したビジュアライゼーション

概要

各デバイステンプレートを 1 つまたは複数の Logic Builder のビジュアライゼーションに関連付けることができます。対応しているビジュアライゼーションのタイプを下記に示します。

対応しているビジュアライゼーション

EcoStruxure Machine Expert では両方のビジュアライゼーションに対応しています。

- シンプルなビジュアライゼーション
- フレームを使用したモジュラーのビジュアライゼーション

フレームを使用したビジュアライゼーションはより柔軟でモジュール性があります。

シンプルなビジュアライゼーション

フレームを使用しないビジュアライゼーションは、I/O デバイス用に作成された単一ビジュアライゼーションオブジェクトが基になっています。

EcoStruxure Machine Expert はビジュアライゼーション要素のプロパティ内で I/O デバイスのデータを参照します。このデバイステンプレートをベースに新しいデバイスを作成すると EcoStruxure Machine Expert はビジュアライゼーション要素のプロパティを変数に直接置き換えます。

フレームを使用したビジュアライゼーション

フレームを使用したビジュアライゼーションは、メイン画面 (フレーム) のあらかじめ定義された領域にモジュールのように組み合わされた多数の小さなビジュアライゼーションを使用して、他のビジュアライゼーションと組み込むことのできるメイン画面で構築されています。

メイン画面では、フレームオブジェクトは入れ物として四角形の物のように配置されます。そのような入れ物に別のビジュアライゼーションを割り当てることができます。

埋め込まれたビジュアライゼーションをインターフェイスで使用することで、ビジュアライゼーション要素に内部的にアクセスできます。

詳細はついては、EcoStruxure Machine Expert オンラインヘルプの **ソフトウェア → CODESYS のビジュアライゼーション**を参照してください。

デバイステンプレートに埋め込まれたビジュアライゼーションを使用するには、各ビジュアライゼーションモジュールの I/O デバイスまたはファンクションブロックへの接続に関するすべての変数の定義を含むインターフェイスを定義します。このデバイステンプレートをベースに新しいデバイスを作成すると、EcoStruxure Machine Expert によって埋め込まれたビジュアライゼーションのすべてのプレースホルダーが作成された I/O デバイス名に従って調整されます。

注記：固有の I/O デバイスにリンクされたフレームおよびファンクションブロックを使用したビジュアライゼーションはすべて、EcoStruxure Machine Expert で検索できるようにライブラリーで定義する必要があります。

デバイステンプレートに制御ロジックを統合するための詳細情報

概要

制御ロジックに、フィールドデバイスとデータを以下のいずれかの方法で交換するコードセクションがある場合、デバイステンプレートにその制御ロジックを含むことができます。

- コードセクションはフィールドデバイスの I/O マッピングで定義された新しい変数を使用。
- コードセクション、およびフィールドデバイスの I/O マッピングは共通の変数を使用。この変数は GVL またはコードセクションが属するアプリケーションにあるコントローラープログラムで定義されます。
  注記：構造体や配列を使用する場合は、それが 1 つのフィールドデバイスのみに関連付けられていることを検証してください。
- コードセクションとフィールドデバイスはデバイス固有の固定変数 (Altivar または Lexium ドライブで使用される axis-ref 変数など) を使用。

コードセクションの相互に接続されている呼び出し

コードセクションは、ファンクションブロック、ファンクション、および演算子の相互に接続された呼び出しのシーケンスで構成されています。

個々の呼び出し間で以下のいずれかの関係が存在する場合、それらは接続されているとみなされます。

- CFC、FBD、および LD で個々の呼び出し間に接続イメージがある。
- 変数が呼び出しの出力と別の呼び出しの入力に接続されている。
- 呼び出し別の呼び出しのパラメーターを使用している。

ファンクションブロックの個別選択

デバイステンプレートに含めるフィールドデバイスとデータを交換するコードセクションに含まれるファンクションブロックを個別に選択することができます。これにより、同じフィールドデバイスに異なるファンクションを提供する別のデバイステンプレートを作成することができます。

注記：ファンクションブロックの種類はライブラリーで定義する必要があります。

デバイステンプレートへの式の追加

ファンクションブロックのパラメーター、ファンクション、または演算子に関連付けられている式、およびこれらの式で使用される変数は自動的にデバイステンプレートに保存されます。

制御ロジック作成の一般的な方法

デバイステンプレートにはシンプルな制御ロジックのみを入れます。

これにより、異なる IEC 言語で作成してもコードセクションは同じように機能します。

注記：複雑な制御ロジックの場合は、ファンクションテンプレートを作成してください。

FBD / LD での制御ロジック作成の方法

他の IEC 言語にはないため、エッジ検出の要素は避けます。

その代わり可能であれば、R_TRIG または F_TRIG ファンクションブロックを使用します。

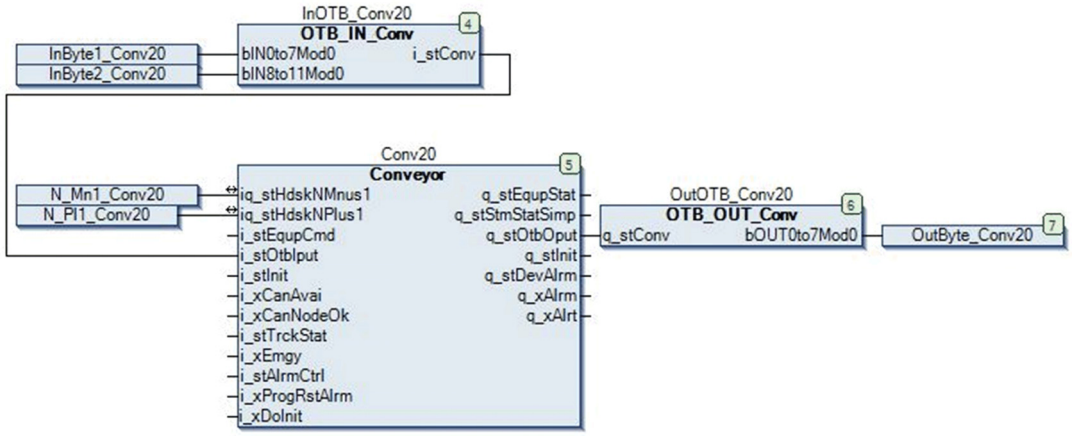
CFC での制御ロジック作成の方法

データフローの位置に従って同じコードセクションに属する CFC 要素を順序付けるには、**Execution Order → Order By Data Flow** コマンドを使用します。これにより、他の IEC 言語との互換性が高くなります。

テンプレートから新しいデバイスが作成される際に変数名が変わり長くなるので、個々の CFC 要素の間にスペース (水平方向に) を入れます。

制御ロジックの例

下図は搬送アプリケーションの Advantys OTB 分散 I/O デバイス用コードセクションの典型的な例です。



コードセクションは次のファンクションブロックで構成されます。

名前	タイプ	ファンクション
InOTB_Conv20	入力ブロック	OTB からのデータを制御ブロックが要求するフォーマットに変換します。
Conv20	制御ブロック	データ処理をします。
OutOTB_Conv20	出力ブロック	制御ブロックからのデータを OTB が要求するフォーマットに変換します。

変数 InByte1_Conv20、InByte2_Conv20 および OutByte_Conv20 は OTB の I/O マッピングで定義します。これは、このコードセクションが OTB とデータ交換をすることを意味します。よって、デバイステンプレートの一部となります。

デバイステンプレート作成手順

概要

次の項では、*フィールドバスをベースにしたデバイステンプレートの作成* (*710* ページ) に記載されている基準を満たすフィールドデバイスを保存する手順を説明します。

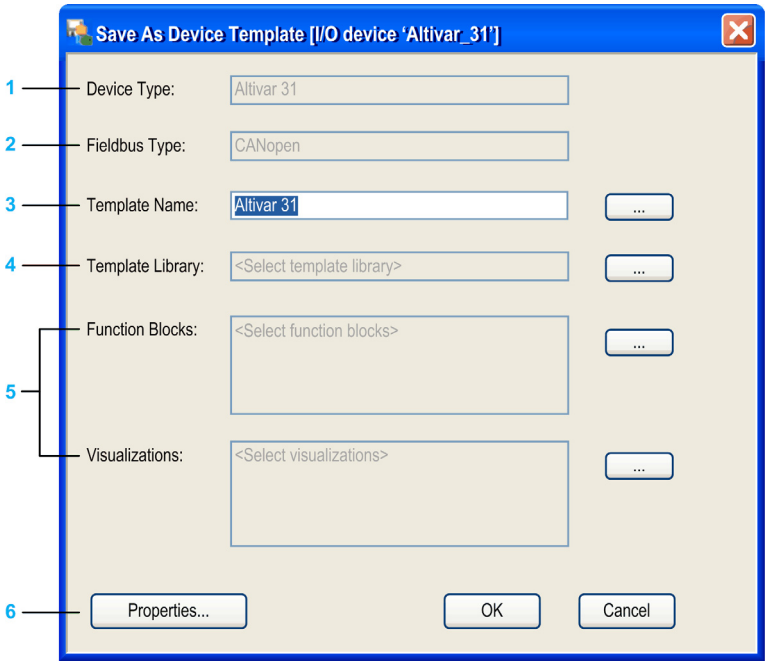
フィールドデバイスをテンプレートとして保存する手順

次の手順に従って既存のフィールドデバイスをデバイステンプレートとして保存します。

手順	手順内容
1	**デバイスツリー**でデバイステンプレートとして保存したいフィールドデバイスを右クリックします。
2	コンテキストメニューから**デバイステンプレートとして保存**を選択します。 **結果** : EcoStruxure Machine Expert は自動的にアプリケーションをビルドします。ビルド処理が正常に完了すると**デバイステンプレートとして保存**ダイアログボックスが表示されます。
3	下記の要領で**デバイステンプレートとして保存**ダイアログボックスで新規デバイステンプレートを定義します。
4	**OK** をクリックし**デバイステンプレートとして保存**ダイアログボックスを閉じて新規デバイステンプレートを作成します。

デバイステンプレートとして保存ダイアログボックス

デバイステンプレートとして保存ダイアログボックスには次のパラメーターあります。



1 デバイステンプレートの基となったフィールドデバイスのタイプが表示されます。
2 フィールドデバイスのフィールドバスタイプが表示されます。
3 作成されるデバイステンプレートの名前 (初期値は元のフィールドデバイスの名前)。
4 デバイステンプレートが追加されるライブラリーを選択します。
5 デバイステンプレートと一緒に保存するファンクションブロックとビジュアライゼーションを選択します。
6 **プロパティ**ボタンでデバイステンプレートの詳細情報を追加します。

新規デバイステンプレート名の定義

Template Name テキストボックスを使用してデバイステンプレートの名前を定義します。

デフォルトでは、このテキストボックスに選択されたフィールドデバイスの名前が表示されます。

このテキストボックスに直接新しい名前を入力するか、このデバイステンプレートを上書きしたい場合は **...** ボタンをクリックしてリストから既存デバイステンプレートを選択します。

テンプレートライブラリーの選択

次の手順に従ってインストールされているテンプレートライブラリーまたは作成したテンプレートライブラリーを選択しデバイステンプレートを保存します。

手順	手順内容
1	**デバイステンプレートとして保存**ダイアログボックスで **テンプレートライブラリー**テキストボックスの右側にある **....** ボタンをクリックします。 **結果** : **テンプレートライブラリーの選択**ダイアログボックスが表示されます。
2	**テンプレートライブラリーの選択**ダイアログボックスに、現在のプロジェクトにインストールされているテンプレートライブラリーまたは作成済みのテンプレートライブラリーがすべて表示されます。書き込み保護がされているテンプレートライブラリーは表示されません。 新規のデバイステンプレートをテンプレートライブラリーに追加するには、適切なテンプレートライブラリーを選択し **OK** をクリックします。

ファンクションブロックの選択

次の手順に従ってデバイステンプレートに加えるファンクションブロックのインスタンスを選択します。

手順	手順内容
1	**デバイステンプレートとして保存**ダイアログボックスで **ファンクションブロック**テキストボックスの右側にある **....** ボタンをクリックします。 **結果** : **ファンクションブロックの選択**ダイアログボックスが表示されます。 **ファンクションブロックの選択**ダイアログボックスにはフィールドデバイス (*712* ページ) の制御ロジックに含まれるすべてのファンクションブロックのインスタンスが表示されます。
2	デバイステンプレートに追加したいファンクションブロックのチェックボックスを有効にします。 または、ルートのノードのチェックボックスを有効にして、このノードの下にあるすべての要素を選択します。
3	**OK** ボタンをクリックします。

ビジュアライゼーションの選択

次の手順に従ってフィールドデバイスに入れるビジュアライゼーションを選択します。

手順	手順内容
1	**デバイステンプレートとして保存**ダイアログボックスで **ビジュアライゼーション**テキストボックスの右側にある **....** ボタンをクリックします。 **結果** : **ビジュアライゼーションの選択**ダイアログボックスが表示されます。 **ビジュアライゼーションの選択**ダイアログボックスにはフィールドデバイスまたは選択されたファンクションブロックの 1 つにリンクされているビジュアライゼーションが表示されます。
2	デバイステンプレートに追加したいビジュアライゼーションのチェックボックスを有効にします。 または、ルートのノードのチェックボックスを有効にして、このノードの下にあるすべての要素を選択します。
3	**OK** ボタンをクリックします。

新規デバイステンプレートに詳細情報を追加

プロパティ ... ボタンをクリックして、新規デバイステンプレートに詳細情報を追加します。**プロパティ**ダイアログボックスが開きます。デバイステンプレートの追加情報を入力できます。デバイステンプレートとテンプレートライブラリーのダイアログボックスは同じですので、テンレプートまたはテンプレートライブラリーの情報の追加の章 (*702* ページ) の説明を参照してください。

第 38 章
ファンクションテンプレートの管理

38.1
ファンクションテンプレートの管理

このセクションについて

このセクションには次の項目が含まれています。

ファンクションテンプレートとは

ファンクションテンプレートのコンテンツ

ファンクションテンプレートは、アプリケーションファンクションに関連付けられた専用の制御およびビジュアライゼーション機能を表します。

ファンクションテンプレートには次の要素があります。

- IEC プログラム、ファンクション、またはファンクションブロック
- ユーザー定義のデータ型 (DUT)
- アプリケーションファンクションで使用されているフィールドデバイスまたは I/O モジュール
- アプリケーションファンクションを視覚化するために使用されているビジュアライゼーション
- グローバル変数リスト
- 他のアプリケーションファンクションと共有可能なグローバル変数
- テキストリスト、およびイメージプール
- トレース
- CAM テーブル (CANmotion 用)
- I/O チャンネルにマップされる I/O 変数
- テンプレートパラメーター

ファンクションテンプレートの使用

使用可能なファンクションテンプレートはテンプレートライブラリーに保存されています。各テンプレートライブラリーには、共通のベース (例えば、すべてがパッケージアプリケーションに関連) をもつ複数のファンクションテンプレートがあります。

すぐに使用できるアプリケーションファンクションを作成するために、それらを簡単に選択し、個々の EcoStruxure Machine Expert プロジェクトの要件に適合させることが可能です。

新規ファンクションテンプレートの作成

既に作成されたアプリケーションファンクションを EcoStruxure Machine Expert プロジェクトで再利用できるようにするには、そのアプリケーションファンクションをファンクションテンプレートとして保存します。

ファンクションテンプレートを保存する際に、どのテンプレートライブラリーに保存するかを決定します。

ファンクションテンプレートのバージョン

ファンクションテンプレートの作成時に、作成するデバイスのデバイスディスクリプションが実際に存在するかが検証されます。存在しない場合は、最新バージョンがあれば自動的にデバイスが最新バージョンに更新されます。

テンプレートからファンクションの追加

手順

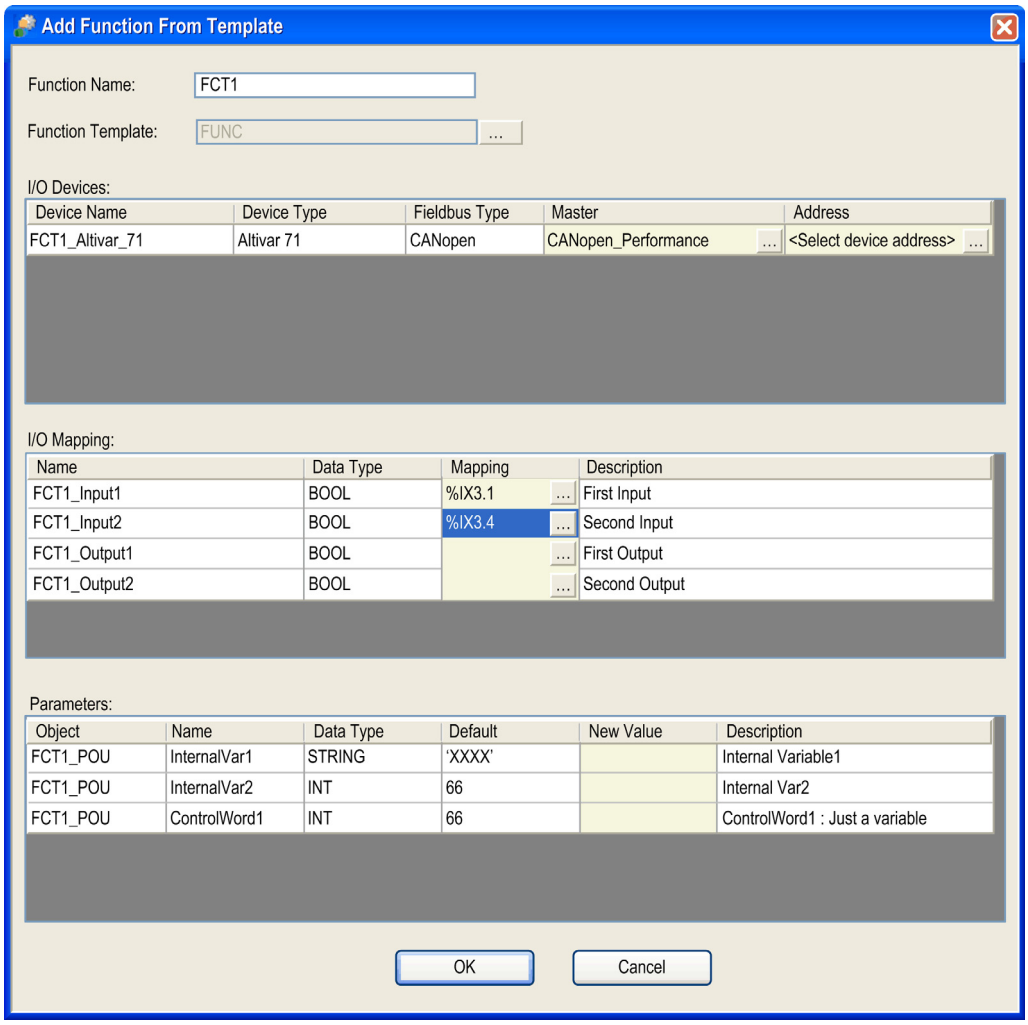
EcoStruxure Machine Expert では、2 種類の方法でファンクションテンプレートからファンクションを追加できます。

次の手順に従ってドラッグ & ドロップでファンクションテンプレートからアプリケーションファクションを追加します。

手順	手順内容
1	**ソフトウェアカタログ**の**マクロ**ビューを開きます。
2	**マクロ**ビューからファンクションテンプレートをドラッグして、ナビゲーターの適切なノードにドロップします。 適切なノードは次の通りです。 ● **アプリケーションツリー**のアプリケーションノード ● **アプリケーションツリー**のアプリケーションノードの下にあるフォルダーノード ● **Functional tree** のコントローラーノード ● **Functional tree** のファンクションモジュールノード **結果** : **テンプレートからファンクションを追加**ダイアログボックスが開きます。

または、ナビゲーターで適切なノードを右クリックしてコンテキストメニューから**テンプレートからファンクションを追加**コマンドを実行して、ファンクションテンプレートからアプリケーションファンクションを追加することもできます。

テンプレートからファンクションを追加ダイアログボックス



テンプレートからファンクションを追加ダイアログボックスには、ファンクションを設定するための次の要素が表示されます。

要素		詳細
ファンクション名テキストボックス		名前を入力します。入力した名前はこのアプリケーションの新しいフォルダーと、そこに含まれる要素の命名に使用されます。
ファンクションテンプレート		**...** ボタンをクリックし**ファンクションテンプレートの選択**ダイアログボックスからファンクションテンプレートを選択します。
I/O デバイステーブル		–
	デバイス名	フィールドデバイス名が表示されます。この名前は変更できません。
	デバイスタイプ	フィールドデバイスのタイプを示します。このセルは変更できません。
	フィールドバスタイプ	フィールドデバイスのフィールドバスタイプを示します。このセルは変更できません。
	マスター	フィールドデバイスが接続されているフィールドバスマスターが表示されます。複数のマスターが存在する場合は、リストから選択することができます。
	アドレス	初期設定では空です。数値アドレスが必要なフィールドバスのフィールドデバイス (Modbus シリアル回線、および CANopen) は、フィールドの右側にある **...** ボタンをクリックしてアドレスを指定します。
I/O マッピングテーブル		ファンクションテンプレートの一部である I/O 変数をリスト表示します。既存のデバイスやモジュールの I/O チャンネルにマップできます。
	名前	I/O チャンネルでマップする必要がある I/O 変数の名前が表示されます。
	データ型	I/O 変数がマップする I/O チャンネルのデータ型が表示されます。
	マッピング	**...** ボタンをクリックして **I/O マッピングの選択**ダイアログボックスを開きます。選択した変数をマップする I/O チャンネルを選択できます。 変数が I/O チャンネルにマップされると、変数がマップされた I/O チャンネルの入力または出力アドレスが**マッピング**フィールドに表示されます。
	詳細	I/O 変数の詳細が表示されます。
パラメーターテーブル		ファンクションテンプレートに含まれているテンプレートパラメーターがリスト表示されます。
	オブジェクト	変数が定義された GVL またはプログラムの名前が表示されます。このフィールドは変更できません。
	名前	変数名が表示されます。このセルは変更できません。
	データ型	変数のデータ型が表示されます。このセルは変更できません。
	デフォルト	変数のデフォルト値が表示されます。これは、テンプレートが作成された時の変数の初期値です。このセルは変更できません。
	新しい値	新しい値を変数に割り当てる場合、このセルを編集します。このセルを空にするとこの変数には**デフォルト値**が使用されます。 データ型にあった有効な値を入力します。
	詳細	変数の説明を表示します。
OK ボタン		**OK** ボタンをクリックして設定を確定します。 **結果** : EcoStruxure Machine Expert は設定が有効であるか検証し、**アプリケーション**ノードの下に新しいアプリケーションファンクションを個別のノードとして挿入するか、検出したエラーメッセージを表示します。

I/O マッピングの選択ダイアログボックス

I/O マッピングの選択ダイアログボックスは**テンプレートからファンクションを追加**ダイアログボックスで選択された変数を I/O チャンネルにマップするために使用します。

使用できる I/O チャンネルが**デバイスツリー**のようなツリー構造で表示されます。ルートのノードはコントローラーです。新しい変数のデータ型に一致する I/O チャンネルのみが表示されます。

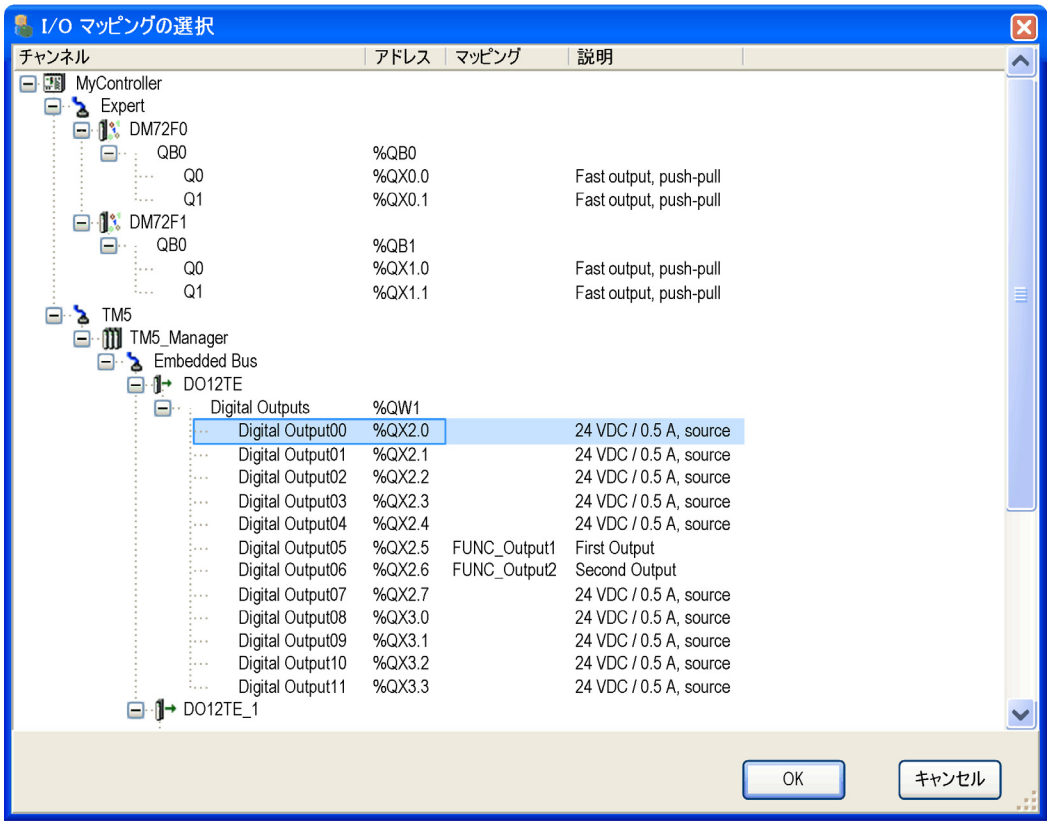
2 つのデータ型は、型名が同一であるか、基本的な IEC データ型で同じサイズであれば互換性があります。

例：

UINT --> INT 互換性あり

UDINT --> INT 互換性なし

プラス記号をクリックするとサブノードが表示されます。



I/O マッピングの選択ダイアログボックスには次の欄があります。

欄	詳細
チャンネル	ツリー構造が表示されます。各デバイスはデバイス名とデバイスアイコンで表示されます。各 I/O チャンネルはチャンネル名で表示されます。
アドレス	I/O チャンネルに対応する入力 / 出力アドレスが表示されます。
マッピング	I/O チャンネルで現在マップされている I/O 変数が表示されます。
詳細	I/O チャンネルの説明が表示されます。

変数を I/O チャンネルにマップする際には以下を考慮します。

- ファンクションテンプレートで提供されるすべての変数を I/O チャンネルにマップします。
- ファンクションテンプレートの I/O 変数を、すでにマッピングされている I/O チャンネルにマップすることができます。既存のマッピングは上書きされます。
- 同じ I/O チャンネル上に複数の変数を割り当てるようなマッピングはできません。

作成されたオブジェクト

ファンクションテンプレートによってプロジェクトに次のオブジェクトが作成されます。

オブジェクト	詳細
ルートフォルダー	**デバイスツリー**の**アプリケーション**ノードの下に**ファンクション名**テキストボックスで定義された名前のフォルダーが作成されます。
フィールドデバイス	ファンクションテンプレートに含まれるフィールドデバイスは、命名規則に沿った名前で作成されフィールドバスマスターに接続されます。必要に応じて I/O マッピングが自動的に調節されます。
ナビゲーターでルートフォルダーのサブノードとして使用可能なオブジェクト	ファンクションテンプレートに含まれるオブジェクトは、命名規則に沿った名前でそれぞれのナビゲーター (**デバイスツリー** , **アプリケーションツリー** , **ツールツリー**) のルートフォルダーの下に作成されます。オブジェクトのプロパティは自動的に調整されます。
タスク設定	タスク設定はファンクションテンプレートで必要な場合に調整されます。
グローバル変数リスト	ファンクションテンプレートに含まれるグローバル変数リストは命名規則に沿った名前でルートフォルダーの下に作成されます。
外部変数	ファンクションテンプレートに属していないグローバル変数リストのグローバル変数は、下記のように元のグローバル変数リストに再保存されます。 ● 元の名前のグローバル変数リストがアプリケーションの下に存在しない場合は、自動的に作成されます。 ● 元の名前のグローバル変数がグローバル変数リストに存在しない場合は、自動的に作成されます。 グローバル変数の型が正しくない場合、EcoStruxure Machine Expert はエラー検出メッセージを表示します。
保持変数	保持変数は下記のようにアプリケーションの変数リストに再保存されます。 ● 保持変数リストがアプリケーションの下に存在しない場合は、元の名前で自動的に作成されます。 ● 元の名前の変数が保持変数リストに存在しない場合は、自動的に作成されます。 保持変数の型が正しくない場合、EcoStruxure Machine Expert はエラー検出メッセージを表示します。
外部オブジェクト	ファンクションテンプレートに含まれていないが、ファンクションテンプレートによって参照されているオブジェクト (ファンクションブロックや DUT など) は、以下のように処理されます。 ● オブジェクトが存在しない場合は作成されます。 ● オブジェクトが既に存在していて変更されていない場合は、変更されません。 ● オブジェクトがすでに存在していて変更が検出された場合は、**Messages** ビューにエラーが報告されます。検出された変更に関する詳細情報を表示するには、**メッセージ**ビューで項目をクリックします。

ファンクションテンプレートのインストールで作成されたオブジェクトはすべて **メッセージ** ウィンドウにリスト表示されます。

オブジェクトの命名

同じファンクションテンプレートを同じコントローラーデバイスで複数回インスタンス化すると、名前の重複を避けるために、以下の命名規則がアプリケーションファンクションおよびそれに関連するオブジェクトに適用されます。

元のオブジェクト名	結果
ケース 1:	
アプリケーションファンクションの名前を含む。	オブジェクトのこの部分は新しく作成されたアプリケーションファンクションの名前で置き換わります。
例 :	
元のアプリケーションのテンプレート Axis は Axis_Init というプログラムを含みます。	このテンプレートで新しいアプリケーションファンクション Axis1 を作成すると、新しいプログラムの名前は Axis1_Init になります。
ケース 2:	
アプリケーションファンクションの名前を含まない。	新しいアプリケーションファンクションの名前とアンダースコアを元の名前に挿入し、固有の新しい名前を作成します。

元のオブジェクト名	結果
例:	
元のアプリケーション Axis は InitProg というプログラムを含みます。	このファンクションテンプレートで新しいアプリケーションファンクション Axis1 を作成すると、新しいプログラムの名前は Axis1_InitProg になります。

注記：アプリケーションファンクションになるべく短い名前を使用すると、名前がすべて表示されます。

ファンクションテンプレートのベースとしてのアプリケーションファンクション

概要

次の項では以下を説明します。
- アプリケーションファンクションと関連するフィールドデバイス、I/O モジュールおよびビジュアライゼーションをファンクションテンプレートとして保存するための条件。
- ファンクションテンプレートに保存される情報。

アプリケーションの前提条件

正常なアプリケーションからのみテンプレートを作成できます。正常とはビルド処理でエラーが検知されないという意味です。

アプリケーションファンクションをファンクションテンプレートとして保存する前提条件

アプリケーションファンクションをファンクションテンプレートとして保存するには、アプリケーションファンクションのプログラムが実行されることが前提条件となります。

次のいずれかの基準を満たす必要があります。
- タスクに追加される。
- 別のプログラムから呼び出される。

それ以外では、**ビルド**コマンドを実行する際に考慮されません。

ファンクションテンプレートの I/O 変数

I/O 変数とはフィールドデバイスの I/O チャンネルにマップされた変数です。次の条件が適用される場合はファンクションテンプレートに保存されます。
- I/O 変数がファンクションテンプレートに含まれるプログラムまたはビジュアライゼーションで使用されている場合。
- I/O 変数がマップされているフィールドデバイスまたは I/O モジュールをファンクションテンプレートに含めることができない場合。

アプリケーションファンクションがファンクションテンプレートから作成された場合 (*720* ページ)、ファンクションテンプレートに保存された I/O 変数を既存の I/O チャンネルにマップすることができます。

I/O 変数の説明は**テンプレートからファンクションを追加**ダイアログボックスに表示されます。

この説明は次のように作成されます。
- I/O 変数がデバイスエディター **I/O マッピング** タブで作成された場合 (*125* ページ)、説明は I/O チャンネルの説明から取得されます (これは元の説明が変更された場合のみ適用されます)。
- I/O 変数が既存変数の参照用である場合、説明はその変数のコメントから取得されます。

テンプレートのパラメーター

テンプレートのパラメーターは調整可能な初期値をもつ変数です。

例 : デバイスを通信ファンクションブロックを介して使用する場合、デバイスのアドレスをこのファンクションブロックに入力パラメーターとして割り当てる必要があります。このアドレスを設定するには、変数をファンクションブロックに接続しその変数をテンプレートのパラメーターとして定義します。

次の条件が適用される場合、変数をテンプレートパラメーターにすることができます。
- 変数がファンクションテンプレートに含まれるプログラムまたはグローバル変数リスト内で定義されている。
- 変数が単純なデータ型 (BOOL、数値型、STRING 型、単純なデータ型のエイリアス) である。
- 変数の初期値が明示的にリテラル値として定義されている。

これらの条件を満たすすべての変数を、ファンクションテンプレートを保存するときに (*729* ページ) テンプレートパラメーターとして選択することができます。

変数をテンプレートパラメーターとして選択した場合、ファンクションテンプレートから新しいアプリケーションファンクションを作成する (*720* ページ) ときに、この変数の初期値を調整できます。

ファンクションテンプレートに保存されるオブジェクト

次のオブジェクトはファンクションテンプレートに保存されます。

- **アプリケーション**ノードの下、または**ファンクションモデル**ノードの下のサブノードとしてツリー構造で使用可能なオブジェクト。
- 使用されているフィールドデバイス I/O モジュール。
- オブジェクトによって参照されるファンクションブロック、ファンクション、または DUT。

ファンクションテンプレート作成手順

概要

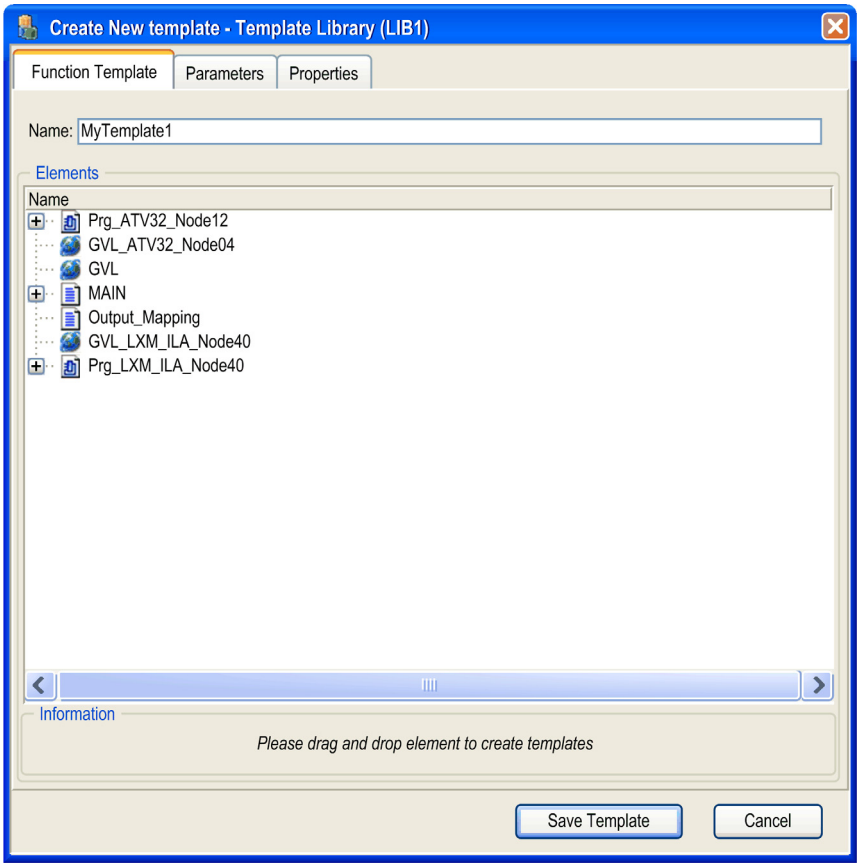
EcoStruxure Machine Expert では 2 種類の方法でファンクションテンプレート作成できます。

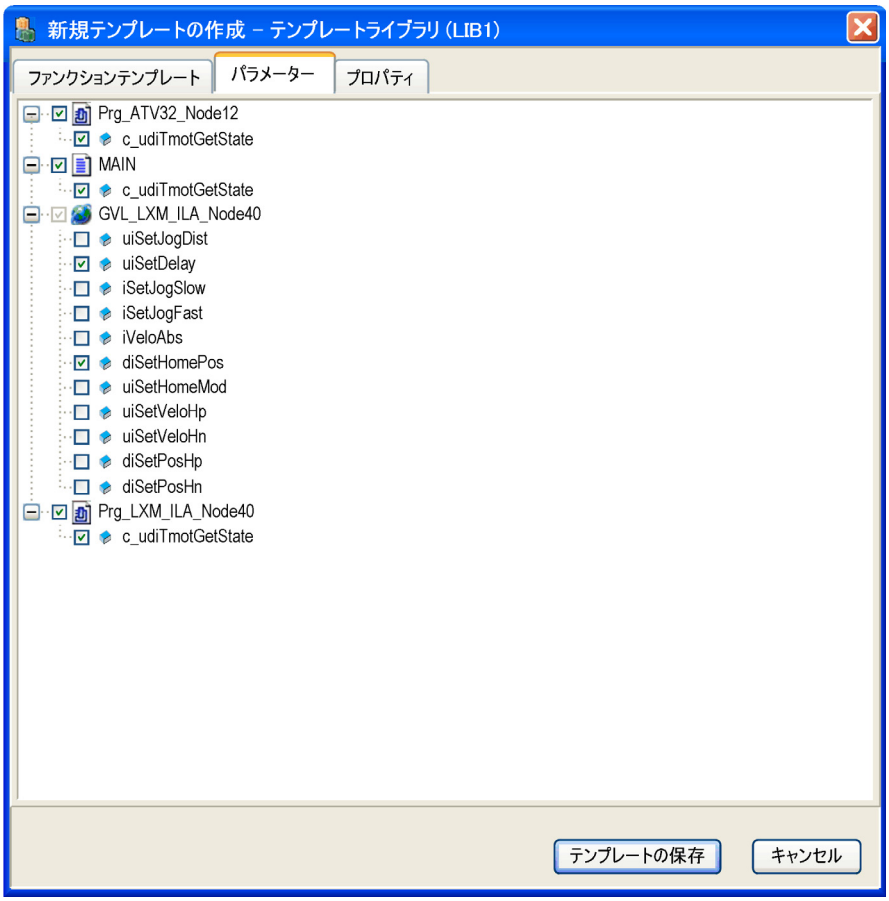
- **マクロ**ビューの**新規テンプレートの作成**ダイアログボックスを使用。
- **アプリケーションツリー**または、**Functional tree** から **Save as Function Template** ダイアログボックスを使用。

次の項では、*ファンクションテンプレートのベースとしてのアプリケーションファンクション* (*725* ページ) に記載されている基準を満たす既存のアプリケーションファンクションをファンクションテンプレートとして保存する手順を説明します。

マクロビューを使用する手順

マクロビューを使用する方法では要素をドラッグ & ドロップして独自のファンクションテンプレートを作成できます。

手順	手順内容
1	**マクロ**ビューで**マイテンプレート**セクションを開きます。
2	**マイテンプレート**ノードを選択して緑色のプラスボタンをクリックします。 **結果 :** 新しいノードが**マイテンプレート**ノードの下にデフォルト名 **LIB1** で挿入されます。
3	**LIB1** ノードを選択して緑色のプラスボタンをクリックします。 **結果 : 新規テンプレートの作成**ダイアログボックスが表示されます。 
4	**新規テンプレートの作成**ダイアログボックスの**ファンクションテンプレート**タブでファンクションテンプレートの**名前**を入力します。 ファンクションテンプレートに入れる要素を**アプリケーションツリー**から**ファンクションテンプレート**タブの**要素**ボックスにドラッグします。このボックスに表示された要素がファンクションテンプレートに挿入されます。 **注記 :** 要素は同じアプリケーションに属している必要があります。

手順	手順内容
5	**新規テンプレートの作成**ダイアログボックスの**パラメーター**タブには**ファンクションテンプレート**タブで選択した要素に含まれる変数が表示されます。  変数リストで、変数またはそのノードのチェックボックスを有効にしてテンプレートパラメーターとして宣言したい変数を選択します。
6	**新規テンプレートの作成**ダイアログボックスの**プロパティ**タブでは、ファンクションテンプレートに詳細な情報を追加することができます。 ファンクションテンプレートのオンラインヘルプのリンクを挿入することができます。このダイアログボックスではローカライズできる文字列の情報を追加できます。またこのファンクションテンプレートを説明する画像を挿入することもできます。これらのパラメーターの詳細については、*テンプレートまたはテンプレートライブラリーの詳細情報の追加* (*702* ページ) の章を参照してください。
7	**テンプレートの保存**をクリックします。

アプリケーションツリー または Functional tree を使用した手順

次の手順に従って既存のアプリケーションファンクションをファンクションテンプレートとして保存します。

手順	手順内容
1	**アプリケーションツリー**のアプリケーションノード、または **Functional tree** のファンクションモデルノードのサブフォルダーを右クリックします。
2	コンテキストメニューから**ファンクションテンプレートとして保存**コマンドを選択します。 **結果 :** EcoStruxure Machine Expert は自動的にアプリケーションをビルドします。ビルド処理が正常に完了すると**ファンクションテンプレートとして保存**ダイアログボックスが表示されます。 
3	新しいテンプレートを下記のように定義します。
4	**OK** をクリックし**ファンクションテンプレートとして保存**ダイアログボックスを閉じ新規ファンクションテンプレートを作成します。 **結果** : EcoStruxure Machine Expert によってファンクションテンプレートが作成可能であるかが検証され、ファンクションテンプレートが正常に作成されたメッセージか、または検知されたエラーが表示されます。

テンプレート名の割り当て

ファンクションテンプレートとして保存ダイアログボックスの**テンプレート名**テキストボックスに、テンプレートライブラリーに格納するファンクションテンプレートの名前を定義します。このテキストボックスにはデフォルトで**アプリケーションツリー**のアプリケーションファンクションが入っているフォルダーの名前が表示されますが、必要に応じて変更することができます。

テンプレートライブラリーの選択

次の手順に従ってインストールされているテンプレートライブラリーまたは新規のテンプレートライブラリーを選択しファンクションテンプレートを保存します。

手順	手順内容
1	**ファンクションテンプレートとして保存**ダイアログボックスで **テンプレートライブラリー**テキストボックスの右側にある **...** ボタンをクリックします。 **結果** : **テンプレートライブラリーの選択**ダイアログボックスが表示されます。
2	**テンプレートライブラリーの選択**ダイアログボックスに、開いているプロジェクトにインストールされているテンプレートライブラリーまたは作成済みのテンプレートライブラリーが表示されます。書き込み保護がされているライブラリーは表示されません。 新規のファンクションテンプレートをテンプレートライブラリーに追加するには、適切なテンプレートライブラリーを選択し **OK** をクリックします。

パラメーターとして変数を選択

ファンクションテンプレートの変数はテンプレートパラメーター (*725* ページ) として定義することができます。

次の手順に従ってファンクションテンプレートの変数をテンプレートパラメーターとして定義します。

手順	手順内容
1	**ファンクションテンプレートとして保存**ダイアログボックスで **パラメーター**テキストボックスの右側にある **...** ボタンをクリックします。 **結果** : **パラメーターとして変数を選択**ダイアログボックスが表示されます。 選択されたアプリケーションで定義された変数が表示されます。
2	個々の変数のチェックボックスを有効にしてファンクションテンプレートのテンプレートパラメーターとして選択します。 または、ルートのノードのチェックボックスを有効にして、このノードの下にあるすべての要素を選択します。
3	**OK** ボタンをクリックします。 **結果** : 選択した変数は**ファンクションテンプレートとして保存**ダイアログボックスの**パラメーター**テキストボックスに表示されます。 これらは**テンプレートからファンクションを追加**ダイアログボックスの**パラメーター**テーブルに表示され、そこでパラメーターの**新しい値**を設定することができます。

既存ファンクションテンプレートの上書き

次の手順に従って既存ファンクションテンプレートに選択したアプリケーションファンクションを上書きします。

手順	手順内容
1	**ファンクションテンプレートとして保存**ダイアログボックスで**テンプレート名**テキストボックスの右側にある **...** ボタンをクリックします。
2	置き換えたい既存のファンクションテンプレートを参照します。
3	置き換えたいファンクションテンプレートを選択します。 **結果** : このファンクションテンプレートの名前が**テンプレート名**テキストボックスに挿入され、**テンプレートライブラリー**テキストボックスにはテンプレートが格納されているテンプレートライブラリーの名前が挿入されます。
4	**OK** をクリックし**ファンクションテンプレートとして保存**ダイアログボックスを閉じ選択したファンクションテンプレートを新しいアプリケーションファンクションに置き換えます。

新規ファンクションテンプレートに詳細情報を追加

プロパティ ... ボタンをクリックして、新規ファンクションンプレートに詳細情報を追加します。**プロパティ**ダイアログボックスが開きます。ファンクションテンプレートの詳細情報を追加することができます。デバイステンプレートとテンプレートライブラリーのダイアログボックスは同じですので、テンレプートまたはテンプレートライブラリーの情報の追加の章 (*702* ページ) の説明を参照してください。

第 IX 部
トラブルシューティングとよくある質問

このパートについて

このパートには次の章が含まれています。

章	章タイトル	参照ページ
39	一般 - トラブルシューティングとよくある質問	733
40	コントローラーへのアクセス - トラブルシューティングとよくある質問	743

第 39 章
一般 - トラブルシューティングとよくある質問

39.1
よくある質問

このセクションについて

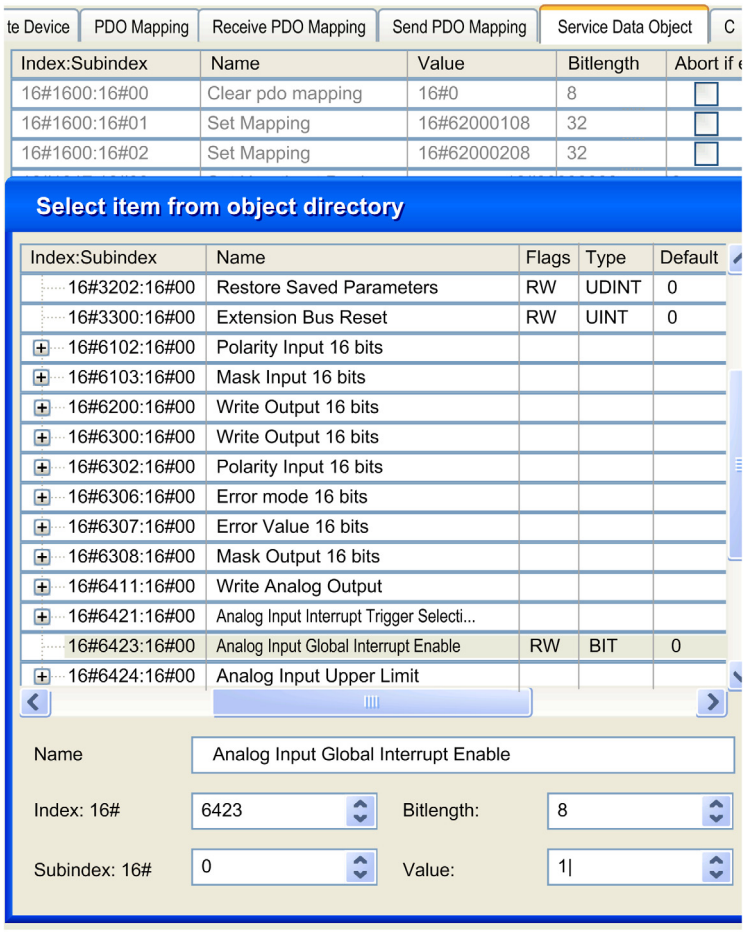
このセクションには次の項目が含まれています。

CANopen でアナログ入力を有効にして設定するにはどうしたらよいですか？

概要

このセクションでは、SDO (Service Data Object、サービスデータオブジェクト) 6423 の値を 1 に設定することで CANopen 標準に従ってアナログ入力を有効にする方法について説明します。

手順

手順	手順内容
1	**デバイスツリー**のアナログ CANopen デバイスノードをクリックします。
2	エディターの **CANopen Remote Device** タブで**エキスパート設定を有効**オプションを有効にします。 **結果** : 追加タブが表示され **Service Data Object** タブに情報が入力されます。
3	**Service Data Object** タブを開き**新規 ...** ボタンをクリックします。 **結果** : **Select item from object directory** ダイアログボックスが表示されます。 
4	オブジェクトのリストから、オブジェクト **6423** を選択し**値**に 1 を入力し **OK** をクリックします。 **結果** : CANopen バス上のアナログ入力送信が有効になります。これでデバイスのハードウェアマニュアルに記載されているように、アナログ値のパラメーターを設定できるようになります。

なぜ時々 EcoStruxure Machine Expert の起動時のパフォーマンスが落ちるのですか？

概要

パソコン構成の他に、EcoStruxure Machine Expert の起動時間が長くなる条件があります。

起動フェーズ	起動時のパフォーマンス
EcoStruxure Machine Expert インストール後の初回起動	EcoStruxure Machine Expert インストール後の初回起動時にパソコン上の作業環境を生成します。これは 1 回のみの実行ですが、初回起動時間に大きな影響を与えます。
再起動後の初回起動	パソコンを再起動すると、EcoStruxure Machine Expert に必要なサービスを Microsoft Windows が起動するためにバックグラウンドで時間を消費するため EcoStruxure Machine Expert の起動時間が通常よりも長くなる可能性があります。これは起動時間に影響し、避けることはできません。
その後の起動	以前にパソコンでシステムを起動している場合、起動時のパフォーマンスは向上します。

ショートカットキーとメニューを管理するにはどうしたらよいですか？

概要

EcoStruxure Machine Expert ソフトウェアのメニューとショートカットキーは、現在の状態、つまり開いているウィンドウまたはエディターによって異なります。

ショートカットキーとメニューを個々の設定に合わせる、または次のセクションで説明されているように EcoStruxure Machine Expert または CoDeSys の標準ショートカットキーとメニューを読み込むことができます。

ショートカットキーとメニューのカスタマイズ

ツール → カスタマイズメニューを使用してショートカットキーとメニューを個々の設定に合わせることができます。

EcoStruxure Machine Expert 標準ショートカットキーとメニューの復元

次の手順に従って EcoStruxure Machine Expert 標準ショートカットキーとメニューを (カスタマイズした後) 復元します。

手順	手順内容
1	**ツール**メニューから**カスタマイズ**コマンドを実行します。 **結果** : **カスタマイズ**ダイアログボックスが表示されます。
2	**カスタマイズ**ダイアログボックスで **Load...** ボタンをクリックします。 **結果** : **Load Menu** ダイアログボックスが表示されます。
3	**Load Menu** ダイアログボックスで *...\Program Files\Schneider Electric\SoMachine Software\Vx.x\LogicBuilder\Settings* フォルダーに移動し、*Standard.opt.menu* ファイルを選択し**開く**をクリックします。 **結果** : **カスタマイズ**ダイアログボックスに 標準 EcoStruxure Machine Expert 設定が表示されます。
4	これらの標準設定を EcoStruxure Machine Expert のグラフィックユーザーインターフェイスに読み込むには **OK** をクリックします。

ショートカットキーとメニューを CoDeSys 標準に設定

次の手順に従って CoDeSys のショートカットキーとメニューを EcoStruxure Machine Expert グラフィックユーザーインターフェイスにインポートします。

手順	手順内容
1	**ツール**メニューから**カスタマイズ**コマンドを実行します。 **結果** : **カスタマイズ**ダイアログボックスが表示されます。
2	**カスタマイズ**ダイアログボックスで **読み込み**ボタンをクリックします。 **結果** : **Load Menu** ダイアログボックスが表示されます。
3	**Load Menu** ダイアログボックスで *...\Program Files\Schneider Electric\SoMachine Software\Vx.x\LogicBuilder\Settings\OriginalCoDeSys* フォルダーに移動し、*Standard.opt.menu* ファイルを選択し**開く**をクリックします。 **結果** : **カスタマイズ**ダイアログボックスに CoDeSys 設定が表示されます。
4	これらの CoDeSys 設定を EcoStruxure Machine Expert のグラフィックユーザーインターフェイスに読み込むには **OK** をクリックします。

注記：EcoStruxure Machine Expert ソフトウェアのメニューとショートカットキーは、現在開いているウィンドウまたはエディターによって異なります。

メニューの拡張

EcoStruxure Machine Expert のメインメニューとコンテキストメニューは折りたたまれた状態または全表示状態のビューで表示されます。折りたたまれたモードでは、使用頻度の低いまたは使用できないコマンドは表示されません。メニュー下部にある矢印メニュー項目 ▼ をクリックすると対応するメニューが展開され、すべてのメニュー項目が表示されます。

全表示モードで常にメニューを表示するには、**ツール → オプション → 機能**ダイアログボックスの**常にフルメニューを表示**オプションを有効にします。

32 ビット オペレーティングシステムで EcoStruxure Machine Expert が使用できるメモリー制限を増やすにはどうすればよいですか？

概要

サイズの大きい EcoStruxure Machine Expert プロジェクトは、32 ビットオペレーティングシステムに負荷が掛かり、メモリー消費に関して技術的に制限される場合があります。これは、32 ビットオペレーティングシステムが EcoStruxure Machine Expert などのユーザー処理に対してメモリーを 2 GB しか提供しないためです。

サイズの大きい EcoStruxure Machine Expert プロジェクトの判別

EcoStruxure Machine Expert プロジェクトの **オブジェクト**総数が多い場合、プロジェクトのサイズは大きいと考えられます。プロジェクトで使用可能な**オブジェクト**、例えば、**デバイス**、**POU**、**アクション**、**DUT**、**グローバル変数リスト**、**ビジュアライゼーション**などは**プロジェクト情報** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の **Statistics** タブに表示されます。ただし、**オブジェクト**の総数のみがプロジェクトサイズの大きさを示しているとは限りません。個々の**オブジェクト**も本質的に大きい可能性があります。

Windows 7 32-Bit オペレーティングシステムの 3 GB 切り替えファンクションを有効にする

Windows 7 32-bit オペレーティングシステムでのユーザー処理のメモリー制限を増やすために、次の手順で 3 GB 切り替えファンクションを有効にできます。

手順	手順内容
1	**スタートメニュー → すべてのプログラム → アクセサリー**に移動します。
2	**コマンドプロンプト**を右クリックし、**管理者として実行**コマンドを実行します。
3	`bcdedit /set IncreaseUserVa 3072` を入力します。
4	コンピューターを再起動します。

Windows 7 32-Bit オペレーティングシステムの 3 GB 切り替えファンクションを無効にする

Windows 7 32-bit オペレーティングシステムでのユーザー処理のメモリー制限を増やすために、次の手順で 3 GB 切り替えファンクションを有効にできます。

手順	手順内容
1	**スタートメニュー → すべてのプログラム → アクセサリー**に移動します。
2	**コマンドプロンプト**を右クリックし、**管理者として実行**コマンドを実行します。
3	`bcdedit /deletevalue IncreaseUserVa` を入力します。
4	コンピューターを再起動します。

EcoStruxure Machine Expert のメモリー消費量を減らすにはどうすればよいですか？

概要

この章では、システム上で EcoStruxure Machine Expert 処理のメモリー消費量を削減するための情報を提供します。

EcoStruxure Machine Expert プロジェクトを分割

EcoStruxure Machine Expert プロジェクトが独立したルートデバイス (コントローラー) などの複数の独立したパーツで構成されている場合は、プロジェクトを分割してルートデバイスごとに独立した EcoStruxure Machine Expert プロジェクトを作成することができます。小さい EcoStruxure Machine Expert プロジェクトはメモリー消費量も少なくなります。

EcoStruxure Machine Expert エディターを閉じる

エディターを開いているとメモリーを消費するので、設定後にはそれぞれの EcoStruxure Machine Expert エディターを閉じます。

EcoStruxure Machine Expert のビルドタイムパフォーマンスを向上するにはどうすればよいですか？

推奨方法

次の推奨方法によって、EcoStruxure Machine Expert を使用しているときのパフォーマンスの低下を避けることができる場合があります。

- パソコンのハードウェアがシステム要件 *(EcoStruxure Machine Expert, Introduction 参照)* を満たしていることを確認します。
- ソリッドステートドライブ (SSD) を使用し十分なメモリー容量があることを確認します。詳細については IT 管理者に問い合わせてください。
- 必要のないコンポーネントを EcoStruxure Machine Expert 設定マネージャー でアンインストールすることを検討します。
- EcoStruxure Machine Expert に統合された Vijeo-Designer をご使用の場合は、自動シンボルエクスポート機能を無効にすることを検討してください。これを行うには、EcoStruxure Machine Expert Logic Builder **オプション → Vijeo-Designer** ダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の**自動シンボルエクスポートの無効**オプションを有効にします。
- この機能をあまり使用していない場合は、EcoStruxure Machine Expert Logic Builder **オプション → FDT オプション**のダイアログボックス *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* **DTM を削除した後に元に戻すを無効にする (性能の最適化)** オプションを無効にすることを検討してください。
- サイズの大きい POU の使用を避けます。EcoStruxure Machine Expert は LD などのグラフィック言語でプログラムされた POU のサイズに制限がありません。ただし、1 つの大きな POU を作成するよりも複数の POU を順番に呼び出すことを推奨します。

シリアル回線の Modbus IOScanner に問題が発生した場合どうすればよいですか？

概要

このセクションでは、シリアル回線の Modbus IOScanner の使用で検知された問題を解決する方法について説明します。

アプリケーションの例外状態

シリアル回線の Modbus IOScanner はコントローラーで設定されていて、Modbus スレーブデバイスの 1 つが切断されています。

アプリケーションのダウンロード後、またはコントローラーのリセット後にアプリケーションが例外状態になる場合は以下の手順に従います。

手順	手順内容
1	ケーブルに問題がないか確認します。
2	コントローラーと Modbus シリアルスレーブ間のケーブルが正確に接続されていることを確認します。
3	コントローラーをリセットします。

シリアル回線で Modbus IOScanner 使用中にエラーを検知

条件	結果
シリアル回線で Modbus IOScanner 使用中にエラーを検知	エラーの発生または原因として検出されたスレーブに関連付けられた xError フラグは TRUE に設定されます。
	通信は停止しません (コントローラーはスレーブに接続しようとします)。
	パラメーター uiNumberOfCommunicatingSlaves が減少し xAllSlavesOK は FALSE に設定されます。

スレーブへの接続が再確立された後、以下を行うためにスレーブの xReset 入力の立上がりが必要となります。

- xError のリセット
- uiNumberOfCommunicatingSlaves の値の更新
- xAllSlavesOK の値の更新

ネットワーク変数リスト (NVL) 通信が中断した場合どうすればよいですか？

問題

オンライン変更後に NVL 通信が中断。

対処方法

対象のコントローラーを再起動します。

第 40 章
コントローラーへのアクセス - トラブルシューティングとよくある質問

この章について

この章には次のセクションが含まれています。

40.1
トラブルシューティング：新規コントローラーへのアクセス

このセクションについて

このセクションには次の項目が含まれています。

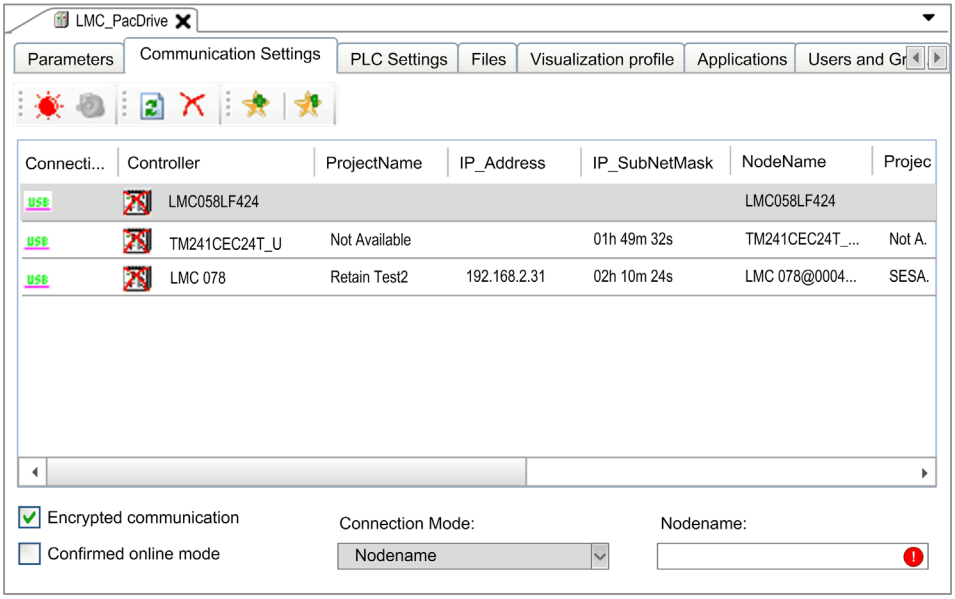
新規コントローラーへのアクセス

概略

新規コントローラーにアクセスするには、コントローラーのネットワーク設定を EcoStruxure Machine Expert PC のネットワークに合わせます。この章では手順を追った例を示します。

例

この例では、IP アドレスが 192.168.1.33 の LMC058 にサブネットマスク 255.255.255.0、IP アドレス 192.168.1.10 の PC からアクセスする手順を示します。

手順	手順内容
1	コントローラーを EcoStruxure Machine Expert が起動している PC に直接接続するか PC のネットワークに Ethernet ケーブルで接続します。
2	EcoStruxure Machine Expert で、デバイスエディターの表示 (*83* ページ) から**通信設定**を開きます。 **結果 :** LMC058 コントローラーがリストに表示されます。 
3	コントローラーの通信設定を変更するには、**通信設定**リストのコントローラーを右クリックし、コンテキストメニューから**通信設定を編集しています ...** コマンドを実行します。 **結果 : 通信設定の編集**ダイアログボックスが開きます。
4	**通信設定の編集**ダイアログボックスでにネットワークで使用可能な **IP アドレス**を入力します。 IP アドレスを設定するときは、次の警告メッセージを参照してください。
5	**OK** をクリックして**通信設定の編集**ダイアログボックスを確認します。
6	**通信設定**ビューでコントローラーに接続します。

ネットワークの各デバイスには固有のアドレスが必要なため、IP アドレスは慎重に管理してください。複数のデバイスに同じ IP アドレスを設定すると、ネットワークおよび関連する機器が意図しない動作をする可能性があります。

⚠ 警告

装置の意図しない動作

- すべてのデバイスが固有のアドレスであることを確認してください。
- システム管理者から IP アドレスを取得してください。
- システムを動作させる前に、デバイスの IP アドレスが固有であることを確認してください。
- ネットワーク上の他の機器に同じ IP アドレスを割り当てないでください。
- Ethernet 通信を含むアプリケーションを複製した後は、IP アドレスを固有のアドレスに更新してください。

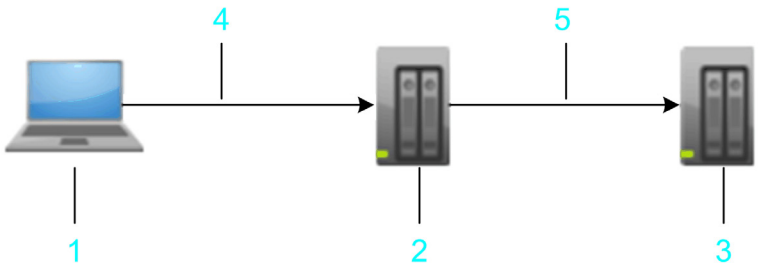
上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

注記：遠隔からのアクセスを避けるためのパラメーターをサポートしているコントローラーもあります (LMC •0•C コントローラーの **RemoteCommunicationAccess** パラメーター)。

IP アドレスを介した接続とアドレス情報

概要

使用される通信プロトコルはコントローラーに接続するための接続の種類に依存しないメカニズムを提供します。例えば、パソコンに USB で接続しているホップコントローラーに Ethernet 経由で接続している対象コントローラーにアクセスすることができます。

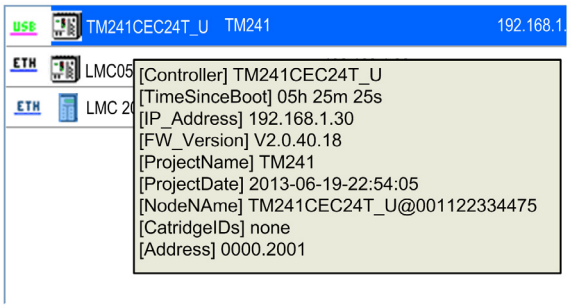


1 パソコン
2 ホップコントローラー
3 対象コントローラー
4 USB
5 Ethernet

アドレス情報

例では、USB は異なるプロトコルを使用します。そのため通常は IP アドレスを使用して対象コントローラーをアドレス指定することはできません。代わりに 1 つまたは複数のホップを介した対象コントローラーへの接続方法が記述されたルーティング情報が使用されます。

このルーティング情報はコントローラーリストの項目のツールチップとして表示されます (以下の例では、**[Address] 0000.2001**)。



注記：このアドレスはコントローラーへの接続方法のみを示しているので、ローカルパソコンやホップコントローラーのネットワークアダプター設定の変更によって変わる場合があります。例えば、ネットワークアダプターを有効 / 無効にした時、またはネットワークアダプターを使用するサービスの開始 / 停止時に変わります。特定の対象コントローラーへのアドレスは、送信パソコンによって異なる場合があります。

ノード名

コントローラーの**ノード名**はシステム内で安定した ID であるため、対象コントローラーを識別するために使用されます。

IP アドレスが**接続モード**として選択されている場合、**ノード名**から情報を取得します。コントローラーによっては (LMC •0•C など) **ノード名**が IP アドレスと共に自動的に作成します。**ノード名**を設定して (FAQ なぜコントローラーが通信設定ビューに表示されないのですか？(*750* ページ) で説明されているように) システムがコントローラーを IP アドレスで検索するようにできます。ノード名に IP アドレスがない場合はコントローラーから IP アドレスを取得します。ただし、すべてのデバイスまたは現在のファームウェアバージョンがこの機能をサポートしている訳ではありません。この場合、**接続モード : ノード名**を使用して接続するか括弧で囲んだ IP アドレスを含むデバイス名を設定します。
例 : **MyDevice (192.168.1.30)**。

40.2
よくある質問 - コントローラーとの接続に問題が発生した場合はどうすればよいですか？

このセクションについて

このセクションには次の項目が含まれています。

よくある質問 - なぜコントローラーと接続できないのですか？

IP アドレスはあっているようなのになぜコントローラーと接続できないのですか？

*新規コントローラーへのアクセス*の章 (*745* ページ) で説明されているように IP アドレスを設定していてもコントローラーに接続できない場合は、サブネットマスクが原因である可能性があります。使用する通信プロトコルは、送信側と受信側のサブネットマスクが同じである必要があるため、コントローラーへ ping が可能であっても接続が確立ができません。

この問題を解決するには、次の手順を実行します。

手順	手順内容
1	EcoStruxure Machine Expert で、デバイスエディターの表示 (*83* ページ) から**通信設定**を開きます。
2	コントローラーの通信設定を変更するには、**通信設定**リストのコントローラーを右クリックし、コンテキストメニューから**通信設定を編集しています ...** コマンドを実行します。 **結果 : 通信設定の編集**ダイアログボックスが開きます。
3	コントローラーに設定された **サブネットマスク**を EcoStruxure Machine Expert のパソコンのサブネットマスクと同一にします。 **例 :** **255.255.0.0** を **255.255.255.0** に変更します。

注記 : 通信設定ダイアログボックスで**接続モード**を変更した後は、選択したコントローラーにアクセスするためにログイン手順を 2 回実行する必要がある場合があります。

なぜコントローラーにログインできないのですか？

アプリケーション (EcoStruxure Machine Expert Logic Builder、コントローラーアシスタントなど) とコントローラーの通信には、実行中の EcoStruxure Machine Expert ゲートウェイが必要です。コントローラーにログインを試みるとアプリケーションは自動的に有効な EcoStruxure Machine Expert ゲートウェイを開始します。EcoStruxure Machine Expert が (Windows) の管理者権限で起動されていない場合は、ゲートウェイの開始は行われません。

この問題を解決するには、次の手順を実行します。

手順	手順内容
1	Windows の通知領域で ゲートウェイ管理コンソール アイコンが、選択されたゲートウェイが停止していることを示す赤で表示されていることを確認します。 
2	ゲートウェイ管理コンソール アイコンを右クリックして**ゲートウェイの開始**コマンドをコンテキストメニューから実行します。 **結果 :** 選択されたゲートウェイのサービスが開始します。
3	Windows の通知領域で ゲートウェイ管理コンソール アイコンが、選択されたゲートウェイが実行されていることを示す緑色で表示されていることを確認します。 
4	もう一度コントローラーにログインします。

なぜコントローラーは通信設定タブに表示されないのですか？

コントローラーと EcoStruxure Machine Expert のパソコン間の通信をクラシックモードを使って確立した場合、クラシックモードに**通信設定** タブが表示されます (*100* ページ)。

コントローラーがクラシックモードの**通信設定**タブに表示されない場合は、次のように、一時的にコントローラーの選択モードの **通信設定**タブに切り替えます (*83* ページ)。

手順	手順内容
1	**ツール → オプション → デバイスエディター** ダイアログボックスを開きます *(EcoStruxure Machine Expert, Menu Commands, Online Help 参照)*。
2	**Communication page → Controller selection mode** 設定を選択し、**OK** をクリックして確認します。

なぜコントローラーは通信設定ビューに表示されないのですか？

コントローラーが、コントローラー選択モード (*83* ページ) の**通信設定**タブに表示されない場合は、2 つの異なるデバイスが同じ**ノード名**に割り当てられている可能性があります。2 つの異なるデバイスに同じ**ノード名**が割り当てられている場合、どちらか 1 つのデバイスのみがリストに表示されます。

ネットワーク上の各デバイスは固有の**ノード名**が必要なため、**ノード名**は慎重に管理してください。複数のデバイスに同じ**ノード名**を設定すると、ネットワークおよび関連する機器が予期しない動作をする可能性があります。

⚠ 警告

装置の意図しない動作

- システムを動作させる前に、すべてのデバイスに固有のノード名が割り当てられていることを確認してください。
- Ethernet 通信を含むアプリケーションを複製した後は、ノード名を固有のノード名に変更してください。

上記の指示に従わないと、死亡、重傷、または物的損害を負う可能性があります。

次の手順に従ってデバイスの**ノード名**を変更します。

手順	手順内容
1	リストで重複した**ノード名**が割り当てられているデバイスを右クリックし、**デバイス名の変更**コマンドをコンテキストメニューから実行します。 **結果 : デバイス名の変更**ダイアログボックスが表示されます。 
2	**デバイス名の変更**ダイアログボックスで固有の**ノード名**を**新規**テキストボックスに入力します。
3	**OK** をクリックして**デバイス名の変更**ダイアログボックスを閉じます。
4	**通信設定**ビューで**更新**ボタンをクリックしデバイスリストを更新します。 **結果 :** 同じ**ノード名**から別のノード名に変更したデバイスがリストに表示されます。
5	重複した**ノード名**が無くなるまで手順 1...4 を繰り返します。

注記 : LMC •0•C などのコントローラーは、プロジェクトのダウンロード後のプロジェクトのデバイス名と IP アドレス (例 : **MyLMC (192.168.1.30)**) で**ノード名**を自動的に作成します。コントローラーで変更があった場合、割り当てられた**ノード名**は自動的に上書きされます。

よくある質問 - なぜパソコンとコントローラーの通信が中断するのですか？

なぜパソコンとコントローラーの通信が中断するのですか？

下記のようにゲートウェイを再起動する必要がある場合があります。

手順	手順内容
1	Windows タスクバーで**ゲートウェイトレイアプリケーション**アイコン  を右クリックします。
2	**ゲートウェイの再起動**をコンテキストメニューから実行します。

付録

付録について

付録には次の章が含まれています。

章	章タイトル	参照ページ
A	ネットワーク通信	755
B	Python スクリプト言語	763
C	移行用コントローラー機能セット	849
D	技術情報保護	853

付録 A
ネットワーク通信

この章について

この章には次の項目が含まれています。

ネットワークトポロジー

概要

EcoStruxure Machine Expert 制御ネットワークは、透過的な通信媒体に対応し、異なるネットワーク間でパケットをルーティングするためにそれ自身の設定 (アドレスの割り当て) をするようにプログラムされたシステムです。ルーティングのメカニズムはネットワークのどのノードでも、つまりリソースが少ないノードであってもパケットを再ルーティングできるほど単純です。そのため、大きなルーティングテーブル、複雑な計算、または実行時の要求は避けられています。

制御ネットワークは階層的に構成されています。つまり、各ノードに 1 つの親ノードと任意の数の子ノードがあります。親のないノードは、最上位ノードです。リング型は許可されていません。つまり、制御ネットワークはツリー構造です。

親と子の関係は、ネットワークセグメントの仕様によって生じます。例えば、ネットワークセグメントはローカル Ethernet またはシリアルのポイントツーポイント接続に相当します。これにより、メインネットワーク (メインネット) とサブネットワーク (サブネット) を区別します。各ノードには多くて 1 つのメインネットワークがあり、それが親です。各ノードには、任意の数のサブネットを設定できます。ノードは、それらすべての親として機能します。

ネットワークセグメントが複数のノードのサブネットとして同時に定義された場合は、ネットワークに複数の親ができます。ただし、各ネットワークセグメントは 1 つの親しかもてないため、設定は無効になります。

アドレス指定とルーティング

概要

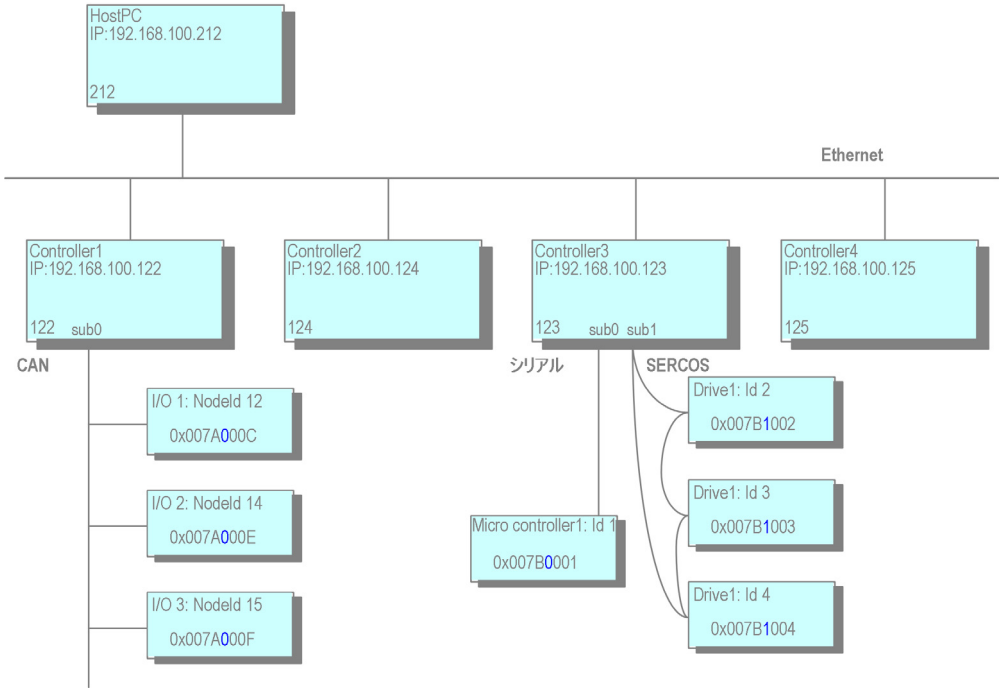
アドレス指定によって、制御ネットワークのトロポジーを固有のアドレスにマップします。ノードアドレス (*759* ページ)、は階層的に構成されます。

各ネットワーク接続において、その各ローカルネットワークで固有にノードを識別するローカルアドレスは、関連するブロックドライバーによって割り当てられます。ノードアドレス全体では、このローカルアドレスの前に親によってローカルネットワークを割り当てられたサブネットインデックスが付きます。さらに、その親のノードアドレスが前に付きます。

デバイスによってサブネットインデックスの長さ (単位はビット) が決まるのに対し、ローカルアドレスの長さはネットワークの種類によって決まります。

メインネットワークのないノードは、アドレス 0 の最上位ノードです。親をもたないメインネットワークのノードも最上位ノードであり、メインネットワークのローカルアドレスに割り当てられます。

例 : メインネットとサブネット



この例では、子ノードのアドレスは 16 進数で表されています。最初の 4 桁は、メインネットの特定の親のアドレスを表します。例えば、PLC1 では 0x007A=122。次のバイト (青色で表示) はサブネットインデックス用に予約されており、ローカルアドレスがそれに続きます。例えば、ノード ID 12 では C=12。

アドレスの構造化により、ルーティングアルゴリズムは比較的効率的に保たれます。例えば、ルーティングテーブルは必要ありません。ローカルで必要な情報は、自身のアドレスおよび親ノードのアドレスです。

それで、ノードはデータパケットを適切に処理できます。

- 目標のアドレスが現在のノードのアドレスと等しい場合、目標のアドレスが受信機であると判断されます。
- 目標アドレスが現在のノードのアドレスで始まる場合、パケットはそのノードの子または子孫用であることを意味し、転送されます。
- または、受信機は現在のノードの子孫ではないことを意味します。パケットは自身の親に転送する必要があります。

相対アドレス指定

相対アドレス指定は特別な機能です。相対アドレス (*760* ページ) は、受信機のノードのノード番号は含まず直接送信機から受信機へのパスを表します。原則は、ファイルシステムの相対パスに類似します。アドレスは、パケットが上に移動する必要があるステップ数、すなわち次の対応する親へのステップ数と目標ノードへの下に続くパスで構成されています。

相対アドレス指定の利点は、全体の制御ネットワーク内でサブツリーが別の場所に移動したときにも、同じサブツリー内の 2 つのノードが通信を継続できることです。このような移動によってノードの絶対アドレスは変わりますが、相対アドレスは有効なままです。

アドレスの決定

ノードは、自身のアドレスを親からのアドレスであるか、または自身が最上位のノードであるかを判断しようとします。そのために、ノードは起動中にブロードキャストメッセージを介してメインネットワークにアドレス判定を送信します。このメッセージが応答されない限りノードは自身を最上位ノードと見なします。ただし、親ノードの検出は続行します。親ノードはアドレス通知で応答します。そこで、ノードは自身のアドレスを完成させ、サブネットに渡します。

アドレス決定は、起動時またはプログラミングパソコンの要求で実行できます。

アドレスの構造

概要

下記は、次のアドレスタイプの構造の詳細です。
- ネットワークアドレス (*759* ページ)
- ノードアドレス (*759* ページ)
- 絶対アドレスと相対アドレス (*760* ページ)
- ブロードキャストアドレス (*761* ページ)

ネットワークアドレス

ネットワークアドレスはネットワークタイプ (例えば、IP アドレス) のアドレスと制御ネットワークの論理アドレスとのマッピングを表しています。このマッピングは各ブロックドライバーによって処理されます。クラス C IP アドレスの Ethernet では、すべてのネットワークデバイスの IP アドレスの最初の 3 バイトが同じです。そのため、IP アドレスの最後の 8 ビットによってブロックドライバーにおける 2 つのアドレス間を明白にマッピングできるので、ネットワークアドレスとして十分です。

ノードには、各ネットワーク接続ごとに個別のネットワークアドレスがあります。このアドレスは、各ネットワーク接続ごとにローカルでのみ固有である必要があるため、異なるネットワーク接続では同じネットワークアドレスがある場合があります。

用語 : 一般的に、ネットワーク接続の記述がないノードのネットワークアドレスは、メインネットワークのネットワークアドレスを参照します。

ネットワークアドレスの長さはビットで指定し、必要に応じてブロックドライバーで選択できます。ネットワークセグメントでは、すべてのノードで同じ長さにしてください。

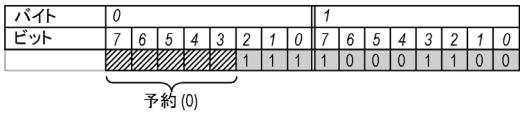
ネットワークアドレスは、次のコーディングと共にバイト型配列として表されます。
- ネットワークアドレスの長さ : n ビット
- 必要なバイト : b = (n + 7) DIV 8
- 最初のバイトの最下位ビット (n MOD 8) および残りすべて (n DIV 8) のバイトがネットワークアドレスに使用されます。

例 - ネットワークアドレス

長さ : 11 ビット

アドレス : 111 1000 1100

ネットワークアドレスコーディングの例



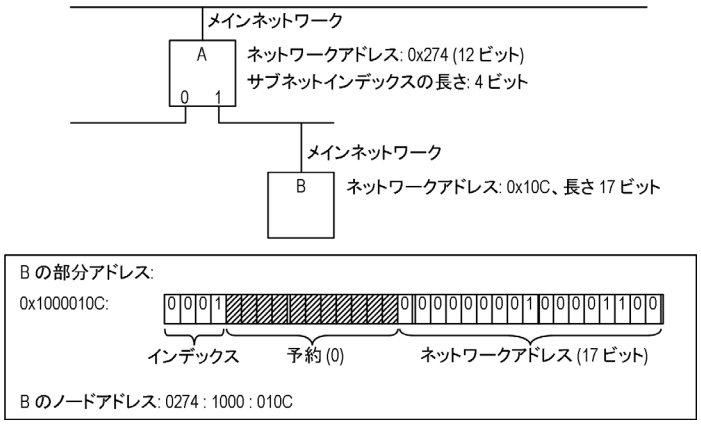
ノードアドレス

ノードアドレスは、制御ネットワークのノードの絶対アドレスを示すため、ツリー全体で固有です。アドレスは最大 15 個のアドレス部で構成され、各アドレス部は 2 バイトで構成されています。下位のノードがネットワーク階層内にあるほど、そのアドレスは長くなります。

ノードアドレスは、ノードのすべての先行ノードの部分アドレスとノード自身で構成されます。各部分アドレスは、複数のアドレス部で構成されています。そのため、長さは常に 2 の倍数です。ノードの部分アドレスは、ノードのメインネットワークにあるノードのネットワークアドレスと親ノードのメインネットワークのサブネットインデックスで形成されています。サブネットインデックスに必要なビットは、親ノードのルーターによって決まります。部分アドレスの長さが 2 バイトの倍数であることを確認するために、サブネットインデックスとネットワークアドレスの間にフィラービットが挿入されます。

特殊な場合 :
- ノードにメインネットワークがない : これは、メインネットワークにサブネットインデックスもネットワークアドレスもないことを意味します。この場合、アドレスは 0x0000 にセットされます。
- メインネットワークはあるが親がないノード : この場合、サブネットインデックスの長さは 0 ビットと仮定されます。部分アドレスはネットワークアドレスに対応しており、必要に応じてフィラービットで補完されます。

例 - ノードアドレス



ノードアドレスは常に 16 進数で表されます。この個別のアドレス部 (それぞれの場合で 2 バイト) はコロン (:) で区切られています。アドレス部のバイトは、区切り文字なしで順番に表示されます (上記の例を参照) これはバイト型配列を表し 16 ビットの値ではないため、アドレス部はリトルエンディアン形式では表示されません。アドレス部の桁が足りないアドレスを手動で入力した場合は、左から順に 0 で埋められます。274 = 0274 読みやすさの向上のため、出力には常に先頭のゼロを含めてください。

絶対アドレスと相対アドレス

2 つのノード間の通信は、相対アドレスまたは絶対アドレスに基づきます。絶対アドレスはノードアドレスと同じです。相対アドレスは、送信機から受信機へのパスを指定します。相対アドレスは、アドレスオフセットと受信機へ降りていくパスで構成されます。

(負の) アドレスオフセットは、パケットが共通の親から再度下に向かって渡される前に、ツリーでパケットが上に向かって渡される必要のあるアドレス部の数を示します。ノードは、複数のアドレス部で構成される部分アドレスを使用できるため、渡される親ノードの数は常にアドレスオフセットに等しいです。これは、親ノード間の区別が明白ではないことを意味します。そのため、通信相手のアドレスの共通な最初の部分が親アドレスとして使用されます。実際の親ノードに関わらず、各アドレス部は上向きのステップでカウントされます。これらの仮定によるエラーは親ノードで検出でき、ノードによって正常に処理される必要があります。

共通の親に到達すると、相対パス (アドレス部の配列) は通常の方法で下向きに続きます。

正式 : 受信機のノードアドレスは、送信機のノードアドレスから最後のアドレスオフセット部を削除し、残りのアドレスに相対パスを追加して形成されます。

例

この例では、文字がアドレス部を表し、ドットで特定のノードが区切られています。ノードは複数のアドレス部を持つことができるため、この例には複数の文字があります。

ノード A: a.bc.d.ef.g

ノード B: a.bc.i.j.kl.m

- 最下位の共通の親アドレス : a.bc
- A から B への相対アドレス : -4/i.j.kl.m (数字の -4 は 4 つのアドレス部 d、e、f、g から生じます。従って、パケットは上に上がる必要があります。)

相対アドレスは、中間ノードを通して各パスで調整されます。それにより、アドレスオフセットも調整されます。これは、常に親ノードによって行われます。ノードがサブネットの 1 つからパケットを受信すると、アドレスオフセットはこのサブネットのアドレス部の長さだけ増加します。

- 新しいアドレスオフセットが 0 より小さい場合、パケットは親ノードに送られます。
- アドレスオフセット 1 が 0 以上の場合、相対アドレスのアドレスオフセットで表された位置のローカルアドレスの子ノードにパケットを送ります。初めに、ノードが正しいアドレスを認識するように、ノードアドレスは子ノードのローカルアドレスの長さ分だけ増加させてください。

共通の親を決定する際に上記のエラーが発生すると、特別な状況が発生します。この場合、「本当の」共通の親アドレスのアドレスオフセットは負ですが、その大きさはパケットの送信元のサブネットの部分アドレスの長さより大きくなります。ノードはこの状態を検出し、前のノードのアドレスと長さの差に基づいて次の子ノードのローカルアドレスを計算して、次のノードが正しい相対アドレスを認識するようにアドレスオフセットを適合させます。また、アドレス部自体は変更されず、アドレスオフセットのみが変更されます。

ブロードキャストアドレス

グローバルとローカルの 2 種類のブロードキャストがあります。グローバルブロードキャストは、制御ネットワーク内のすべてのノードに送信されます。このために空のノードアドレス (長さは 0) が予約されています。

ローカルブロードキャストは、ネットワークセグメントのすべてのデバイスに送信されます。そのため、ネットワークアドレスのすべてのビットは 1 に設定されます。これは相対アドレスと絶対アドレスの両方で可能です。

ブロックドライバーは両方のブロードキャストアドレスを処理できる必要があります。つまり、空のネットワークアドレスおよびすべてのビットが 1 に設定されたネットワークアドレスを解釈し、ブロードキャストとして送信する必要があります。

付録 B
Python スクリプト言語

この章について

この章には次のセクションが含まれています。

B.1
一般情報

このセクションについて

このセクションには次の項目が含まれています。

概要

EcoStruxure Machine Expert での Python スクリプト

EcoStruxure Machine Expert には、開発環境を自動化するための強力なツールとして使用できる Python インタプリタ が含まれています。

EcoStruxure Machine Expert Logic Builder のメインメニューのコマンド **ツール → Scripting → Execute Script File...** を使用して、Python スクリプトファイルを実行できます。

EcoStruxure Machine Expert での Python スクリプトのドキュメントは、いくつかの部分で構成されています。

EcoStruxure Machine Expert *で Python インタプリタにアクセスする* (*767* ページ) の章では、Python インタプリタ of EcoStruxure Machine Expert のあけ方、そこでのスクリプトファイルの実行方法、およびそのそれ以外の方法での対話について説明します。*Logic Builder Shell の使用* (*771* ページ)、*Logic Builder スクリプトイミディエイトビューの使用* (*776* ページ)、および *Logic Builder Shell とスクリプトイミディエイトビューでのキーボードコマンド* (*778* ページ) の章では個々のパーツについての詳細を説明します。

Schneider Electric のスクリプトエンジンの例 (*803* ページ)、および *CoDeSys のスクリプトエンジンの例* (*822* ページ) では、Python スクリプトを使用して EcoStruxure Machine Expert を自動化するさまざまな例を紹介しています。これらの例は、Python スクリプトで使用可能な EcoStruxure Machine Expert 固有の API （Application Programming Interface / アプリケーションプログラミングインターフェイス）の機能を示す共通のスレッドを提供するために示されています。

EcoStruxure Machine Expert *Python API (dir() および inspectapi 使用)* (*779* ページ)、*ベストプラクティス* (*795* ページ)、および EcoStruxure Machine Expert *スクリプト - Python API* (*798* ページ) の章では Python スクリプトの使用方法や提供されている API を探るための汎用的なヒントを提供します。

EcoStruxure Machine Expert オンラインヘルプの EcoStruxure Machine Expert の *プログラミング*のセクションにある *Schneider Electric - スクリプトエンジンクラスライブラリー*、および *スクリプトエンジンプラグイン API リファレンス*のセクションには、Python スクリプトで使用できる EcoStruxure Machine Expert 固有の API の参照情報があります。これらのメンバーは、前述の例で説明した方法で使用できます。

Python 用開発環境の選択

Notepad++、PTVS 付き Microsoft Visual Studio (Python Tools for Visual Studio)、または PyCharm など、Python ファイルを編集するためのさまざまな方法があります。ニーズにあったエディターを選択してください。

次の章も参照してください。

- *Microsoft Visual Studio と PTVS* で Logic Builder Shell *を使用する* (*783* ページ)
- *JetBrains PyCharm* で Logic Builder Shell *を使用する* (*787* ページ)
- *Microsoft Visual Studio Code と Python Extension* で Logic Builder Shell *を使用する* (*790* ページ)

Python V2.x と Python V3.x 間の互換性

EcoStruxure Machine Expert に付属の Python インタプリタ は、Python バージョン 2.7 をベースにした IronPython の実装です。

注記：バージョン 2.x とバージョン 3.x の間で、Python 言語定義にいくつかの大きな変更があります。EcoStruxure Machine Expert に付属の Python インタプリタ は Python V2.x をベースにしていますが、バージョン 3.x の実装と互換性のあるコードを記述できる可能性があります。これには、`from __future__ import print_function` 命令を使用することなどが含まれます。
詳細は、ウェブサイトを参照してください。

- *http://wiki.python.org/*
- *http://docs.python.org/*

新しいファンクションの例 :

```
from __future__ import print_function
from __future__ import division
```

```
# New Python print syntax
print('Hello World!')
# Division
# Python 2 return an integer and rounds off
# Python 3 returns a float
print(17/3)
```

コーディング規約

同じプログラミングプロジェクトにおける異なるプログラマーによる作業を一致させ、作業を容易にするためには、共通のプログラミングスタイルの採用が適しています。EcoStruxure Machine Expert は、*Python コードののスタイルガイドに準拠しています。*EcoStruxure Machine Expert に付属のスクリプトと例は、これらの規則に準拠しています。コードの可読性を向上させるため、同じ規則に従ってコードを作成することを推奨します。

詳細については、*http://www.python.org/dev/peps/pep-0008/* で *Python コードのスタイルガイド*を参照してください。

役立つリンク

詳細は、次のウェブサイトを参照してください :

- Python 公式ウェブページにはチュートリアルおよび言語リファレンスがあります。 *http://docs.python.org/*
- Python 公式ブログ *http://blog.python.org/*
- *初心者のための Python ガイド* *http://wiki.python.org/moin/BeginnersGuide*
- CoDeSys 公式フォーラムには用例と役立つ情報があります。*http://forum.codesys.com/*
- IronPython インタプリタ *http://ironpython.codeplex.com/*。
- PyTools (Visual Studio プラグイン) *http://pytools.codeplex.com/*
- PyCharm (Visual Studio プラグイン) *http://www.jetbrains.com/pycharm*
- PyCharm Debugging の使用方法 : *http://www.jetbrains.com/help/pycharm/2017.1/debugging.html*.
- Visual Studio Code: *http://code.visualstudio.com*.
- Python extension: *http://marketplace.visualstudio.com/items?itemName=donjayamanne.python*.
- *IronPython クックブック* *http://www.ironpython.info/index.php/Contents*.
- 無料 Galileo オンラインブック (ドイツ語のみ) *http://openbook.galileocomputing.de/python/*.

EcoStruxure Machine Expert で Python インタプリタにアクセスする

概略

EcoStruxure Machine Expert は Python スクリプト言語を使用して自動化ができます。

EcoStruxure Machine Expert の Python スクリプト機能はさまざまな方法で公開されています。この表には、さまざまなエントリポイントが一覧表示されています。

場所	入力タイプ	説明 / 使用例	開き方
Logic Builder ユーザーインターフェイス	対話型 (REPL)	EcoStruxure Machine Expert のユーザーインターフェイス内蔵された、コマンドラインインターフェースを備えた対話型 Python シェル。	Logic Builder のメニューコマンド **ビュー → スクリプトイミディエイト** (Logic Builder *スクリプトイミディエイト* (*776* ページ) の使用も参照してください)。
	Python ファイル (*.py)	EcoStruxure Machine Expert のユーザーインターフェイス内から実行するスクリプトファイルを選択します。	Logic Builder のメニューコマンド **ツール → Scripting → Execute Script File...** (Logic Builder *スクリプトイミディエイトビューの使用* (*776* ページ) も参照してください)。
スタンドアロン シェル	REPL	コマンドラインインターフェイスで、スタンドアロン実行 (EcoStruxure Machine Expert のユーザーインターフェイスなし) の対話型 Python シェル。	コマンドライン引数なしで LogicBuilderShell.exe を開きます (*Logic Builder Shell の使用* (*771* ページ) を参照してください)。
	Python ファイル (*.py)	Python スクリプトを Windows コマンドライン、バッチファイル、または同様の方法で実行します。	スクリプトファイルをコマンドライン引数として LogicBuilderShell.exe を開きます (*Logic Builder Shell の使用* (*771* ページ) を参照してください)。
Microsoft Visual Studio + PTVS (Python Tools for Visual Studio)	REPL	Microsoft Visual Studio の対話型 REPL ビューを使用し、Logic Builder Shell で Python コマンドを実行できます。	LogicBuilderShell.exe を Microsoft Visual Studio 内で Python インタプリタ として設定します (refer to the chapter *Microsoft Visual Studio と PTVS で* (*783* ページ) Logic Builder Shell を使用するを参照してください)。
	Python ファイル (*.py)	Visual Studio プロジェクトの Python ファイルは Logic Builder Shell で実行、およびデバッグができます。機能比較表 (*768* ページ) のデバッグに関する考慮事項も参照してください。	
JetBrains PyCharm	REPL	JetBrains PyCharm の Python Console を使用し、Logic Builder Shell で Python コマンドを実行できます。	LogicBuilderShell.exe を PyCharm 内で Python インタプリタ として設定します (JetBrains PyCharm で Logic Builder Shell *を使用する* (*787* ページ) を参照してください)。
	Python ファイル (*.py)	Python ファイルは Logic Builder Shell で実行、およびデバッグができます。	

場所	入力タイプ	説明 / 使用例	開き方
Microsoft Visual Studio Code + Python extension	Python ファイル (*.py)	Python ファイルは Logic Builder Shell で実行、およびデバッグができます。機能比較表 (*768* ページ) のオートコンプリート、およびシンタックスエラーチェック機能に関する考慮事項も参照してください。	LogicBuilderShell.exe を Microsoft Visual Studio Code 内で Python インタプリタ として設定します (refer to the chapter *Microsoft Visual Studio Code と Python extension* で (*790* ページ) Logic Builder Shell を使用するを参照してください)。

次のセクションでは、Python スクリプトがさまざまな使用例で EcoStruxure Machine Expert に統合される方法の概要を説明します。

サードパーティーメーカーの統合開発環境 (IDE) の機能比較

この章に記載されている各サードパーティーメーカーの IDE はそれぞれ異なる機能をサポートしています。この表には、3 つの IDE とそれらがサポートする機能がリストされています。

注記：オートコンプリート (IntelliSense) 機能は、EcoStruxure Machine Expert 固有の API のフィールド、プロパティ、およびメソッドを認識しません。

ファンクション	Microsoft Visual Studio + PTVS	PyCharm	Microsoft Visual Studio Code + Python extension
REPL	X	X	–
Execute	X	X	X
デバッグ	X	X	X
ブレークポイント	X	X	X
ステップイン / ステップオーバー	X	X	X
変数ウォッチ / 式の評価	–	X	X
シンタックスの強調表示	X	X	X
シンタックスエラーチェック	X	X	–
オートコンプリート (IntelliSense)	X	X	部分的 (ローカル要素、およびコードスニペットのみ)
注意事項	–	PyCharm の EcoStruxure Machine Expert インストールで修正を行う必要があります。 デバッグは IronPython で行えますが、エラーメッセージはコンソールに表示されます。	デバッグの実行は Python モジュール機能で停止しますが、継続できます。 オートコンプリート機能は LogicBuilderShell.exe のいくつかのプロセスで開始します。完了すると再び使用できるようになります。 オートコンプリート機能をさらに使用すると、エラーメッセージが拡張子によって引き起こされ、Microsoft Visual Studio Code コード開発コンソールに表示されます（詳細については、**ヘルプ → Toggle Developer Tools** を参照してください)。
X 機能はサポートされています **–** 機能はサポートされていません			

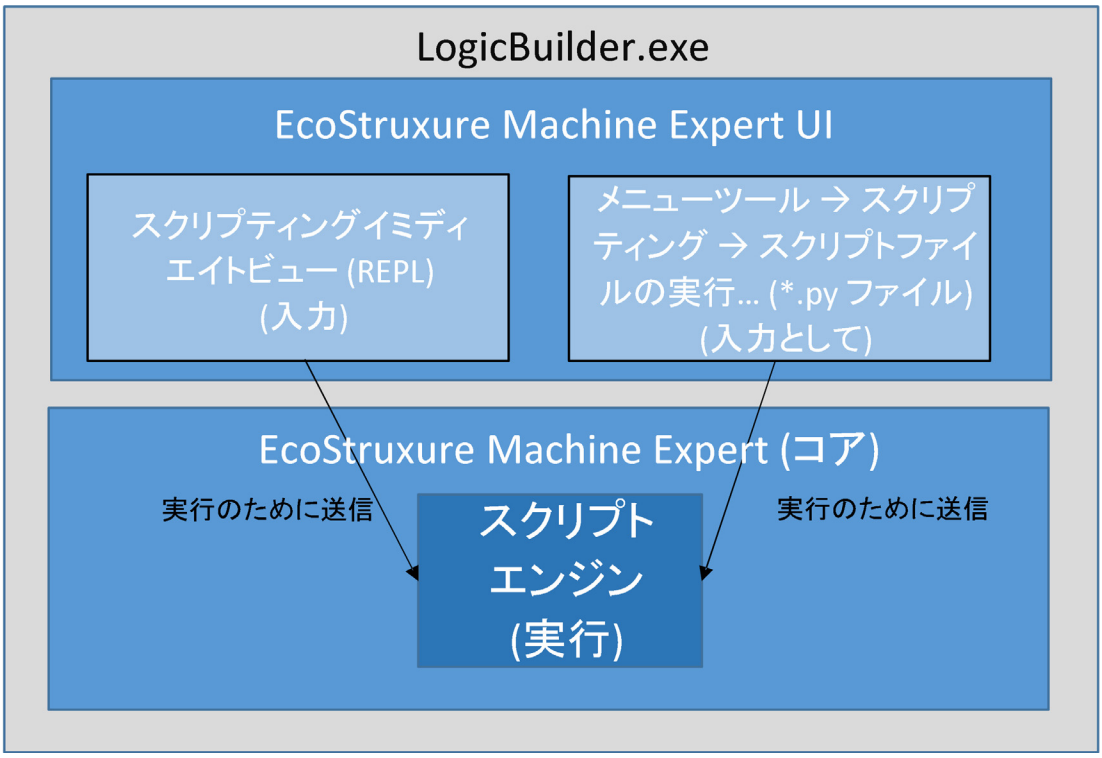
Logic Builder ユーザーインターフェイス

Logic Builder のユーザーインターフェイスでは、**ツール → Scripting → Execute Script File...** コマンドを使用してスクリプトファイルを実行したり、**スクリプトイミディエイト**ビューでスクリプトステートメントを実行できます。

スクリプトイミディエイトビューは EcoStruxure Machine Expert に統合された (Python) インタプリタで、例えばファンクションの起動を可能にします。

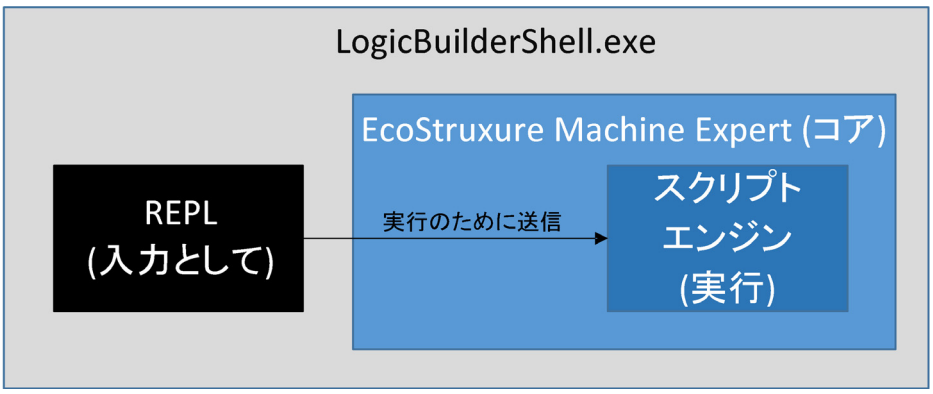
スクリプトイミディエイトビューはインタプリタをホストし、REPL の原則に基づいています。

ブロック図は、EcoStruxure Machine Expert ユーザーインターフェイスがスクリプトエンジンを使用して Python コマンドを実行している様子を示しています。



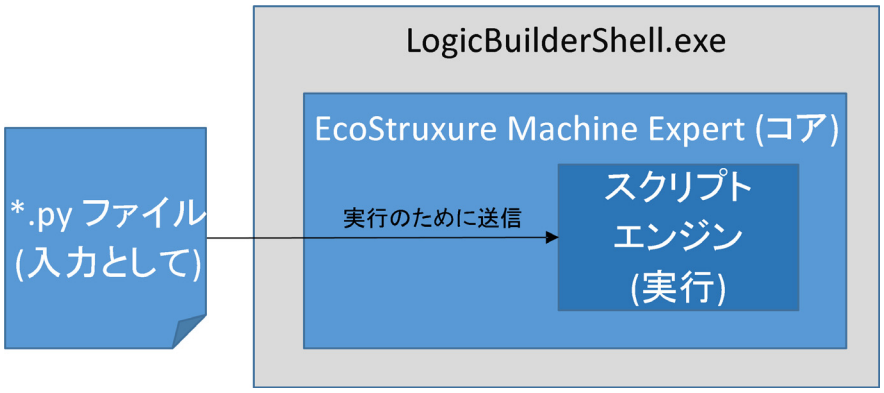
対話型シェルを介したステートメントの実行

ブロック図は、REPL ベースの シェルの対話型インタプリタ が Python ステートメントを実行するためにスクリプトエンジンをどのように使用しているかを示しています。



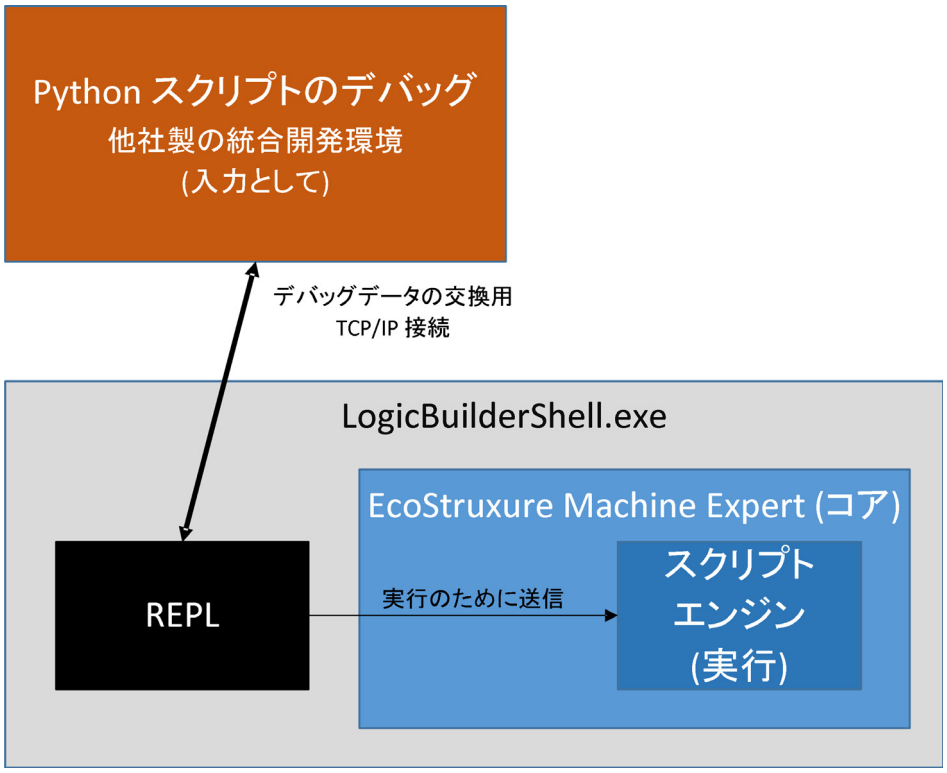
非対話型シェルを介したスクリプトの実行

ブロック図は、(非対話型) シェル がスクリプトを実行するためにスクリプトエンジンをどのように使用しているかを示しています。



サードパーティーメーカーの統合開発環境 (IDE) を使用したスクリプトのデバッグ

ブロック図は、サードパーティーメーカーの統合開発環境 (IDE) (例えば、Visual Studio および PTVS) が Python どのようにスクリプトのデバッグに使用できるかを示しています。



Logic Builder Shell の使用

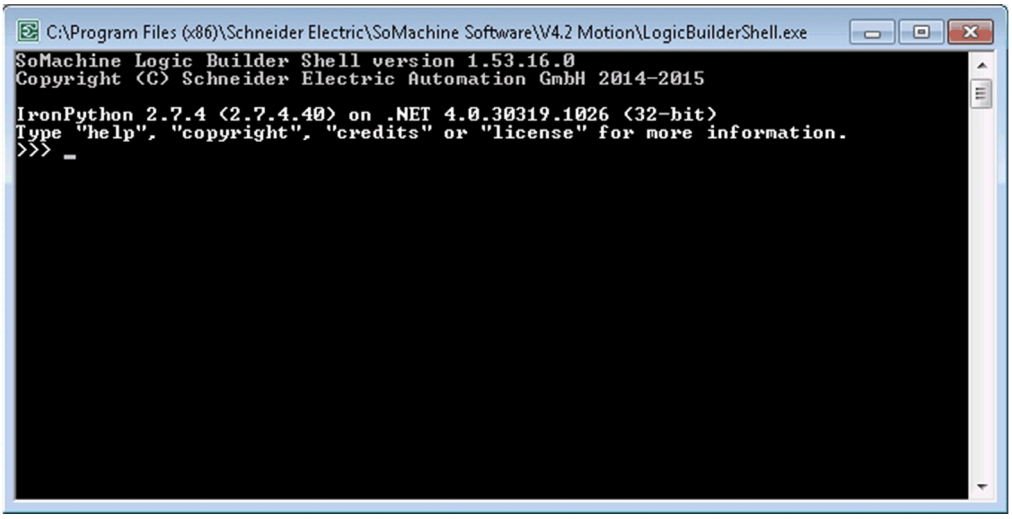
Logic Builder Shell の起動

LogicBuilderShell.exe は EcoStruxure Machine Expert がインストールされているディレクトリーにあります。

Logic Builder Shell は、IronPython ipy.exe に基づいた EcoStruxure Machine Expert REPL を提供します。これは、パソコンにインストールされたときの標準 IronPython パッケージの REPL です。したがって、タブ補完、Python スクリプトのデバッグサポートなど、同じ機能が提供されます。さらに、EcoStruxure Machine Expert Python API へのアクセスが可能です。

LogicBuilderShell.exe をダブルクリックして、コンソールで Logic Builder Shell (引数なし) を起動します。

結果 : 入力プロンプト (>>>) が表示されます。



コマンド

入力プロンプト (>>>) で Python ステートメントを入力し、**入力** を押して実行します。

次の一連のコマンドは、プロジェクトを開き、デバイスノードを検索し、名前を印刷する方法の例を示しています。

```
IronPython 2.7.4 (2.7.4.40) on .NET 4.0.30319.18444 (32-bit)
Type "help", "copyright", "credits" or "license" for more information.
>>> projectName = "C:\\temp\\MyProject.project"
>>> proj = projects.open(projectName)
>>> sercosNode = proj.find("SERCOSIII", True)[0]
>>> print(sercosNode.get_name(False))
SERCOSIII
>>>
```

使い方

LogicBuilderShell.exe [options] [file.py|- [arguments]]

[options] は以下のコマンドラインのリストで定義されます。

[arguments] は、スクリプトファイルに渡される引数のプレースホルダです。

Logic Builder Shell のコマンドライン引数

コマンドライン引数	詳細	例
-h / --help	IronPython コマンドラインヘルプを表示します。	LogicBuilderShell.exe --help LogicBuilderShell.exe -h
-m <module>	ライブラリーモジュールをスクリプトとして実行します。	LogicBuilderShell.exe -m pdb MyScript.py
-i	スクリプト実行後、対話式でチェックします。	LogicBuilderShell.exe -i MyScript.py

コマンドライン引数	詳細	例
<Script File>	指定した Python スクリプトファイルを実行します。	LogicBuilderShell.exe MyScript.py
--nologo	起動時に Logic Builder の情報テキストの表示をスキップします。 シャットダウン中の …terminated メッセージの表示をスキップします。	LogicBuilderShell.exe --nologo

異なるコマンドライン引数を組み合わせることができます。例えば、-i 引数は、指定された Python スクリプトファイルおよび -nologo と組み合わせることができます。

リストには、IronPython のコマンドライン引数の一部のみが表示されています。完全なリストについては、IronPython Web サイトおよび資料を参照してください。

Logic Builder Shell の使用例

Logic Builder Shell はさまざまな場面で使用できます。以下の使用例は、EcoStruxure Machine Expert と Python スクリプトを使った作業の使いやすさを向上させる方法を示しています。

使用例 1

Subversion (SVN) から EcoStruxure Machine Expert プロジェクトを**チェックアウト**し、プロジェクトをコンパイルしてライブラリーとして保存するなど、CI (Continuous Integration / 継続的インテグレーション) システムでスクリプトを実行する場合は、スクリプト出力をコンソールに出力する必要があります。実際のコンソールアプリケーションとして LogicBuilderShell.exe を使用すると、出力をリダイレクトしたり、出力を取得してビルドログに追加した CI システム (Jenkins など) で呼び出すことができます。

例 :

LogicBuilderShell.exe MyScript.py > output.txt

使用例 2

Visual Studio の Python ツールの対応として LogicBuilderShell.exe を統合することができます (デバッグの章を参照してください (*783* ページ))。この場合、LogicBuilderShell.exe は Visual Studio と EcoStruxure Machine Expert の間の通信をデバッグし、ステートメントを実行するために使用されます。

使用例 3

LogicBuilderShell.exe を使用すると、pdb モジュールと呼ばれる内蔵コマンドラインデバッグ機能を使用できます。

例 :

LogicBuilderShell.exe -m pdb MyScript.py

使用例 4

Test your Python スクリプトを UI なしモードでテストして、機能していることを確認します。コンソールモードでは、コンソールから入力できます。コンソールからの入力の読み取りは LogicBuilderShell.exe でのみ可能であることに注意してください。Logic Builder のユーザーインターフェイスにある**スクリプトイミディエイト**ビューでは、コンソール入力はできません。これは readline() ステートメントが無視されるためです。

例 :

```
import sys
print("Please enter a text: ")
text = sys.stdin.readline()
print("Your entered text: " + text)
```

使用例 5

Python および EcoStruxure Machine Expert API とその資料を調べる。

- 内蔵 help() 機能を使用して、IronPython シェルから Python ヘルプにアクセスします。これを実現するには、インターネット接続が利用可能であることを確認してください。

例 :

```
help()      # start the built-in help module
topics      # list all available topics
##### here you'll get the list of available topics #####
LIST        # dumps the help to work with lists.
##### here you'll get the help working with lists #####
quit        # leaf build-in help feature
```

- 内蔵 dir() 機能で、使用可能な API 機能を表示します。

例 :

```
dir()      # prints the list of defined variables and functions in current script scope
##### here you'll get the list of defined variables or functions in current script scope #####
import sys
dir(sys)    # prints the available functions defined in sys module
##### here you'll get the list of defined variables or functions in sys module #####
```

- inspectapi モジュールを使用して、EcoStruxure Machine Expert Python API を調べます。技術的な制限があるため、Python の内蔵 dir() 機能では、EcoStruxure Machine Expert Python API のヘルプを表示できません。これを実現するには、inspectapi を使用します。

例 :

```
inspectapi.dir(projects)  # prints the available API functions of "projects" which provides access e.g. to open a SoMachine project
##### here you'll get the SoMachine API provided via "projects" variable #####
```

使用例 6

指定されたスクリプトと組み合わせて LogicBuilderShell.exe に渡される引数として -i スイッチを使用して、スクリプト実行の最後にシェルプロンプトを表示しますこれは、スクリプトの実行後に対話形式のチェック、例えば、変数の内容を確認するのに役立ちます。

Logic Builder Shell の利点

LogicBuilder.exe を使用して UI モードなしでスクリプトを実行するのではなく、LogicBuilderShell.exe を使用してスクリプトを実行することをお勧めします。どちらの方法も可能ですが、LogicBuilderShell.exe のコマンドラインの方が使い勝手がよく、Logic Builder Shell シェルの出力がコンソールに表示されます。

LogicBuilder.exe を使用して UI なしモードでスクリプトを実行する例

```
LogicBuilder.exe -noui --runscript="<script file>"
```

LogicBuilderShell.exe を使用する例

```
LogicBuilderShell.exe "<script file>"
```

コマンドラインデバッガーモジュール pdb を使用した EcoStruxure Machine Expert Python スクリプトのデバッグ

LogicBuilderShell.exe は、内蔵コマンドラインデバッグ機能 pdb モジュールを提供します (詳細については、https://docs.python.org/2/library/pdb.html を参照してください)。これはエキスパート向けの機能で、Python スクリプトをデバッグしたり、問題を見つけるために使用されます。IronPython 自体が shell に読み込めるモジュールを提供しています。独自のスクリプトを pdb ファンクション run(…) で起動すると、コンソールで Python コードをデバッグできます。コンソールアプリケーションは、テキストベースの UI の可能性のみを提供します。

注記 : コマンドラインデバッガーの使用は複雑になる可能性があります。ユーザーフレンドリーなデバッグ環境には、Microsoft Visual Studio と PTVS の使用を検討してください (詳細は、Microsoft Visual Studio と PTVS で *Logic Builder Shell を使用する* (*783* ページ) を参照してください)。

モジュール pdb は、Python プログラム用の対話型ソースコードデバッガーを定義します。次の機能が提供されています。

- ソース行レベルで、ブレークポイント (条件付) とシングルステッピングを設定
- スタックフレームのチェック
- ソースコード一覧
- 任意のスタックフレームの文脈における任意の Python コードの評価

pdb で Python スクリプトを直接起動するには、次のコマンドライン構文を使用します。

LogicBuilderShell.exe -m pdb MyScript.py

pdb デバッガーコマンドの選択 (詳細は、インターネットの Python ヘルプを参照してください)。

デバッガーコマンド	コマンドの説明
h(elp) [command]	引数なしでの場合、使用可能なコマンドを表示します。 コマンドを引数とした場合、そのコマンドのヘルプを表示します。 help pdb はドキュメントファイルを表示します。
r(un)	デバッグした Python プログラムを再起動します。
b(reak)	行番号を引数として、ファイル内のこの位置にブレークを設定します。 ファンクションを引数として、そのファンクションの最初の実行ステートメントにブレークを設定します。行番号の前にファイル名とコロンを付けて、別のファイル (おそらくまだロードされていないファイル) のブレークポイントを指定することができます。そのファイルは sys.pat 引数がなければ、すべてのブレークをリストします。リストされている各ブレークポイントについて、そのブレークポイント h で検索されます。各ブレークポイントには、他のブレークポイントコマンドが参照する番号が割り当てられています。 2 番目の引数が存在する場合は、ブレークポイントが適用される前に TRUE と評価される必要がある式です。 がヒットした回数、無視数、および関連する条件があればそれも含まれます。
l(ist) [first[, last]]	ファイルのソースコードを一覧表示します。 引数を指定しないと、現在行の周囲に 11 行をリストするか、前のリストを続行します。 1 つの引数を使用して、その行の周囲に 11 行をリストします。 2 つの引数を使って、与えられた範囲をリストします。2 番目の引数が最初の引数より小さい場合は、カウントと解釈されます。
c(ontinue)	実行を継続します。ブレークポイントに到達した場合にのみ停止します。
n(ext)	現在のファンクションの次の行に到達するか、戻るまで実行を続けます。(next と step の違いは、step は呼び出されたファンクションの中で止まりますが next は呼び出されたファンクションを (ほぼ) フルスピードで実行し、現在のファンクションの中の次の行でのみ停止することです。)
s(tep)	現在の行を実行し、最初に停止できる時点 (呼び出されるファンクション内または現在のファンクション内の次の行) で停止します。
u(ntil)	現在の行番号より大きい行番号の行に達する、または現在のフレームから戻るときまで実行を続けます。
r(eturn)	現在のファンクションが戻るまで実行を続けます。
w(here)	最新のフレームを下にしてスタックトレースを出力します。矢印は現在のフレームを示し、これはほとんどのコマンドのコンテキストを決定します。
q(uit)	デバッガーを終了します。実行されているプログラムは中断されます。

pdb を使用したデバッグの例

Python スクリプトのデバッグ例

```
print("Demonstration how to use Visual Studio + PTVS to debug SoMachine Python scripts")

# close open project
if projects.primary:
    print("Close project")
    projects.primary.close()

print("Open project")
projects.open("c:\\Temp\\MyProject.project")

sercosDevice = projects.primary.find("SERCOSIII", True)[0]

sercosDeviceName = sercosDevice.get_name(False)

print("Sercos device name: " + sercosDeviceName)
```

```
print("Close project")
projects.primary.close()
```

次のリストの（pdb）で始まる各行で、list（l）、help、next（n）、quit（q）などのデバッガーコマンドが実行されます。その間に、実行されたステートメントまたはその出力を見ることができます。

```
LogicBuilderShell.exe -m pdb SoMachinePythonDemonstration.py
SoMachine Logic Builder Shell version 1.53.16.0
Copyright (C) Schneider Electric Automation GmbH 2014-2015

Demonstration how to use Visual Studio + PTVS to debug SoMachine Python scripts
> C:\temp\somachinepythondemonstration.py(5)<module>()
-> if projects.primary:
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv
(Pdb) l
  1
  2     print("Demonstration how to use Visual Studio + PTVS to debug  SoMachine Python scripts")
  3
  4     # close open project
  5  -> if projects.primary:
  6         print("Close project")
  7         projects.primary.close()
  8
  9     print("Open project")
 10     projects.open("c:\\Temp\\MyProject.project")
 11
(Pdb) n
> C:\temp\somachinepythondemonstration.py(9)<module>()
-> print("Open project")
(Pdb) n
Open project
> C:\temp\somachinepythondemonstration.py(10)<module>()
-> projects.open("c:\\Temp\\MyProject.project")
(Pdb) l
  5     if projects.primary:
  6         print("Close project")
  7         projects.primary.close()
  8
  9     print("Open project")
 10  -> projects.open("c:\\Temp\\MyProject.project")
 11
 12     sercosDevice = projects.primary.find("SERCOSIII", True)[0]
 13
 14     sercosDeviceName = sercosDevice.get_name(False)
 15
(Pdb) n
IOError: IOError(...oject'.")
> C:\temp\somachinepythondemonstration.py(10)<module>()
-> projects.open("c:\\Temp\\MyProject.project")
(Pdb) q
SoMachine Logic Builder Shell terminated.
```

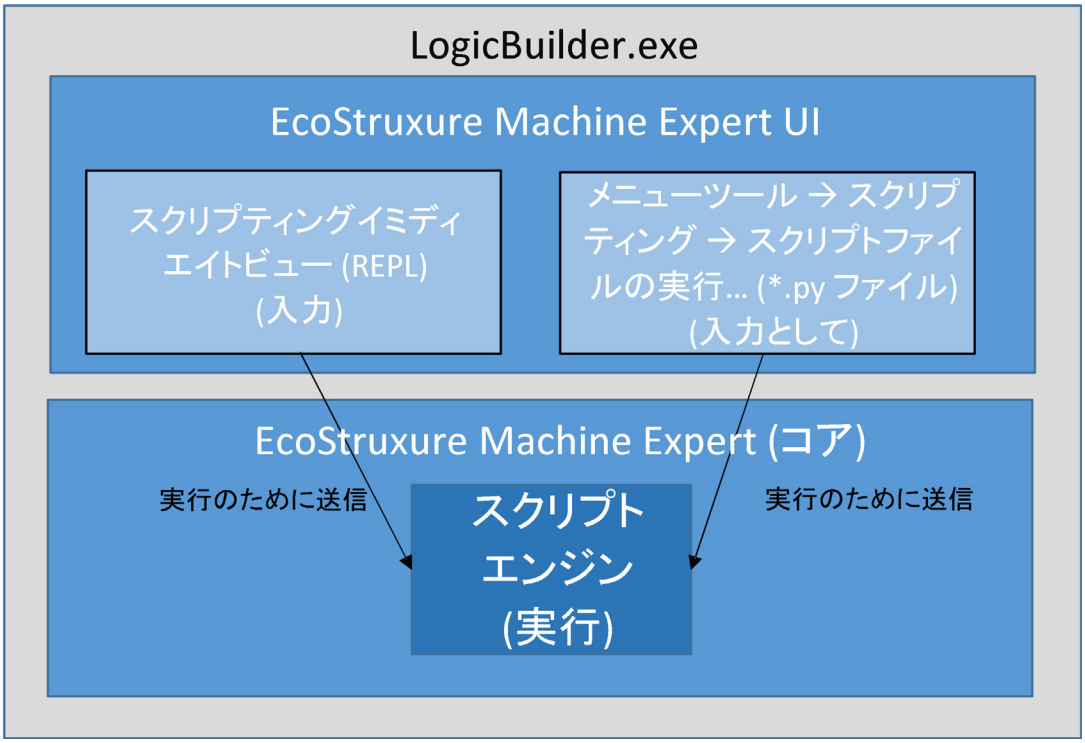
Logic Builder スクリプトイミディエイトビューの使用

概要

LogicBuilder.exe を使用して、ユーザーインターフェイスで Logic Builder を起動すると、次の 2 つのコマンドを使用してスクリプト用の 2 つのビューを開くことができます。

- **ビュー → スクリプトイミディエイト**
- **ビュー → スクリプトウォッチ**

EcoStruxure Machine Expert ユーザーインターフェイスでの **Execute Script File...** コマンド、または**スクリプトイミディエイト**ビューを使用したスクリプトの実行

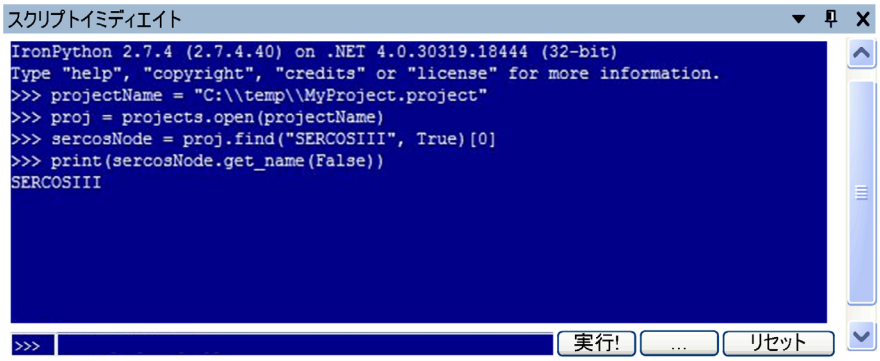


スクリプトイミディエイトビュー

The **スクリプトイミディエイト**ビューは Logic Builder Shell で使用できるものと同じ REPL のホストです。ユーザーインターフェイスと平行して、テキストベースのインターフェイスで EcoStruxure Machine Expert と対話することができます。

Execute the command **ビュー → スクリプトイミディエイト** コマンドを実行して、**スクリプトイミディエイト**ビューを開きます。

この図は **スクリプトイミディエイト**ビューでプロジェクトを開き、デバイスオブジェクトを見つけてその名前を印刷する方法の例を示しています。



スクリプトイミディエイトビューでは、Python ステートメントの入力と実行が可能です (REPL を使用)。

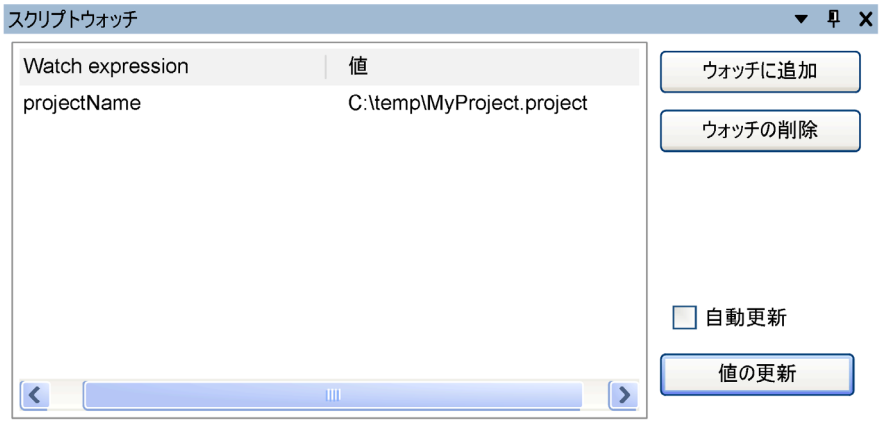
To run a Python スクリプトファイルを実行するには、**...** ボタンをクリックするか、プロンプトテキストボックスにスクリプト名を入力して **実行 !** ボタンをクリックします。

Click the **リセット**ボタンをクリックして button to reset the **スクリプトイミディエイト**ビューをリセットし、カレントスコープをクリアします。

スクリプトウォッチビュー

The **スクリプトウォッチ**ビューは、定義された Python 変数を表示します。

ビュー → スクリプトウォッチ コマンドを実行して、**スクリプトウォッチ**ビューを開きます。



スクリプトウォッチビューの要素

要素	詳細
ウォッチに追加ボタン	ウォッチ式を追加するには、**スクリプトイミディエイト**ビューのプロンプトに式を入力して、**ウォッチに追加**ボタンをクリックします。
ウォッチの削除ボタン	ウォッチ式を削除するには、**スクリプトウォッチ**ビューのエントリーを選択して、**ウォッチの削除**ボタンをクリックします。
自動更新オプション	**自動更新**オプションを選択して、**スクリプトウォッチ**ビューの式の自動更新機能を有効にします。 **注記：**値を更新するプロセスは、追加のパソコンプロセッサーの負荷につながりま ｓ。
値の更新ボタン	**スクリプトウォッチ**ビューリストは、**スクリプトイミディエイト**ビューでステートメントが実行されるたびに自動的に更新されます。このリストを手動で更新するには、**値の更新**ボタンをクリックします。

例

スクリプトイミディエイトビューで、次のステートメントを入力し実行します。

projectName = "C:\\temp\\MyProject.project"

以下の手順に沿って進めます。

手順	手順内容
1	**スクリプトイミディエイト**ビューで projectName を入力します。
2	**スクリプトウォッチ**ビューで**ウォッチに追加**ボタンをクリックします。 **結果** : projectName 変数が**スクリプトウォッチ**リストに表示されます。

projectName 変数に別の値を代入すると、変更された内容が表示されます。

Logic Builder Shell と スクリプトイミディエイト ビューでのキーボードコマンド

概要

この章では、LogicBuilderShell.exe または、Logic Builder **スクリプトイミディエイト**ビューから起動された Logic Builder Shell の使用についての情報を提供します。

タブ補完 / Intellisense

Python ステートメントを入力する場合、キーボードの**タブ**キーを使用して入力したステートメントを完成させるか、オプションを繰り返し入力することができます。

例 :

テキスト ins を入力し、**タブ**キーを押します。**結果** : 式は inspectapi に補完されます。

テキスト inspectapi.d を入力し、**タブ**キーを複数回押します。d で始まる異なる inspectapi ファンクションが表示されます。

現在入力されているプロンプトテキストを消去する

Type a Python ステートメントを入力し、キーボードの **ESC** キーを押します。行がクリアされます。

Python ステートメント履歴

キーボードの **上** または **下**カーソルキーを押して、入力した Python ステートメントの履歴を使用します。

カレントスコープの変数リスト

dir() ファンクションを使用してカレントスコープの変数を表示します。

exit() / exit(…) ファンクション

スクリプトを終了するには exit() / exit(…) ファンクションを使用し、後で使用するために (指定されている場合は) 終了コードを返すようにします。

例 :

Python スクリプト (exit_code_test.py)

```
# …
# Python statements
exit(5)
```

Windows のコマンドプロンプト (またはバッチファイル) でスクリプトを実行、および exit コードを使用。

```
LogicBuilderShell.exe exit_code_test.py
echo %ERRORLEVEL%
```

結果 :

5 がコマンドプロンプトに表示されます。%ERRORLEVEL% を使用してバッチフローを制御できます。

EcoStruxure Machine Expert Python API (dir() および inspectapi 使用)

概要

EcoStruxure Machine Expert は、EcoStruxure Machine Expert スクリプトで使用できる Python API を提供します。Python スクリプトを記述するとき、たいていの場合必要なのは機能名または必要なパラメーターとその名前のリストだけです。呼び出した API ファンクションから返されるオブジェクトの利用可能な機能も必要になることがよくあります。次のファンクションを使用すると、Python スクリプトを開発するための EcoStruxure Machine Expert Python API を試すことができます。

内蔵された Python の dir(…) ファンクション

引数がない場合、このファンクションは現在のローカルスコープ内の名前のリストを返します。引数を指定すると、このファンクションはそのオブジェクトの有効な属性のリストを返そうとします。

Python のデフォルト dir() メカニズムは、完全な情報ではなく最も関連性の高い情報を生成しようとするため、オブジェクトの種類によって動作が異なります。

- オブジェクトがモジュールオブジェクトの場合、リストにはモジュールの属性の名前が含まれます。
- オブジェクトが型またはクラスオブジェクトの場合、リストにはその属性の名前と、その基底の属性が再帰的に含まれます。
- それ以外の場合、リストにはオブジェクトの属性の名前、そのクラスの属性の名前、および基本クラスの属性の再帰が含まれます。

例 (LogicBuilderShell.exe 内):

```
>>> import struct
>>> dir()   # show the names in the module namespace
['__builtins__', '__doc__', '__name__', 'struct']
>>> dir(struct)   # show the names in the struct module
['Struct', '__builtins__', '__doc__', '__file__', '__name__',
 '__package__', '_clearcache', 'calcsize', 'error', 'pack', 'pack_into',
 'unpack', 'unpack_from']
>>> class Shape(object):
        def __dir__(self):
            return ['area', 'perimeter', 'location']
>>> s = Shape()
>>> dir(s)
['area', 'perimeter', 'location']
```

inspectapi ファンクション

dir() ファンクションは、すべての EcoStruxure Machine Expert Python API 要素をリストしないため、追加のファンクションが Python スクリプト開発者用に提供されています。

inspectapi は、LogicBuilderShell.exe または **Scripting Immediate** ビュー起動時に自動的に読み込まれる Python モジュールです。結果として変数 / モジュール inspectapi は現在のスコープで使用可能です。この変数は、dir() ファンクションの実行によってもリストされます。

inspectapi 変数 / モジュールは、次のファンクションを提供します。

機能名	パラメーター	詳細
inspectapi.dir	obj: 詳細情報を得るオブジェクト (例えば、変数) を指定します。(オプション) verbose: デフォルト値は、FALSE です。詳細出力を取得するにはパラメーターを TRUE に設定します。	このファンクションは、呼び出し可能な EcoStruxure Machine Expert Python API メンバーのリストを提供します。フィールド、プロパティ、イベント、およびメソッドがリストされます。
inspectapi.dir_events		このファンクションは、呼び出し可能な EcoStruxure Machine Expert Python API イベントのリストを提供します。
inspectapi.dir_fields		
inspectapi.dir_methods		
inspectapi.dir_properties		

例 :

```
# To see the API of the inspect api itself
inspectapi.dir(inspectapi)
```

出力 :

```
Members of type 'InspectAPI'
Methods:
  dir (obj, verbose = False)
  dir_events (obj, verbose = False)
  dir_fields (obj, verbose = False)
  dir_methods (obj, verbose = False)
  dir_properties (obj, verbose = False)
# To see the API of projects variable
inspectapi.dir(projects)
```

出力 :

```
Members of type 'ScriptProjects'
Properties:
  [get] all
  [get] primary
Methods:
  convert (stPath, stOutputPath, converter, bPrimary)
  convert (stPath, stOutputPath, converterGuid, bPrimary)
  create (stPath, bPrimary)
  get_all ()
  get_by_path (stPath)
  get_primary ()
  open (stPath, stPassword, bPrimary)
  open (stPath, encryption_password, session_user, session_password, bPrimary)
  open_archive (stArchiveFile, stProjectPath, bOverwrite, stPassword)
  open_archive (stArchiveFile, stProjectPath, bOverwrite, encryption_password, session_user, session_p
assword)
# open a project
proj = projects.open("c:\\temp\\MyScript.py ")
# To see API of primary project
inspectapi.dir(projects.primary)
```

出力 (途中切り詰め):

```
Members of type 'ScriptProject'
Properties:
  [set] active_application
  [get] dirty
  [get] handle
  [get] has_library_manager
  [get] has_project_info
  [get] is_root
  [get] library
  [get] path
  [get] primary
  [get] project
  [get] svn
  [get] user_management
Methods:
  add (stName, id, stModuleId)
  add (stName, iType, stId, stVersion, stModuleId)
  check_all_pool_objects ()
  clean_all ()
  close ()
  compare_to (projectFile)
  compare_to (projectFile, ignoreWhiteSpace, ignoreComments, ignoreProperties)
  ...
  logout ()
  save ()
  save_archive (stArchiveFile)
  save_archive (stPath, additional_categories)
  save_archive (stPath, comment, additional_categories)
  save_archive (stPath, additional_files, additional_categories)
  ....
  save_as (stPath, stPassword)
```

```
  save_as_compiled_library (destination_name)
  set_active_application (value)
  update ()
# search objects by name (e.g. a device with name SERCOSIII)
objs = projects.primary.find("SERCOSIII", True)
# extract object at index 0 from returned list (if found).
theObj = obj[0]
# see the available API of the found object SERCOSIII
inspectapi.dir(theObj)
```

出力 (途中切り詰め):

```
Members of type 'ScriptObject'
Properties:
  [get] connectors
  [get] device_parameters
  [get] embedded_object_types
  [get] guid
  [get] handle
  [get] has_textual_declaration
  [get] has_textual_implementation
  [get] index
  [get] is_application
  [get] is_device
  [get] is_explicit_connector
  [get] is_folder
  [get] is_libman
  [get] is_project_info
  [get] is_root
  [get] is_task
  [get] parent
  [get] project
  [get] svn
  [get] type
Methods:
  add (stName, id, stModuleId)
  add (stName, iType, stId, stVersion, stModuleId)
  allowed_interfaces_at (index)
  can_convert (targetDeviceId, targetModuleId = null)
  can_convert (targetType, targetId, targetVersion, targetModuleId = null)
  convert (targetDeviceId, targetModuleId = null)
  convert (targetType, targetId, targetVersion, targetModuleId = null)
  create_folder (stFolderName)
  disable ()
  enable ()
  export_native (destination, includeChildren, profileName, reporter)
  export_xml (reporter, stPath, bRecursive)
  export_xml (reporter, stPath, bRecursive, bExportFolderStructure)
  export_xml (reporter, stPath, bRecursive, bExportFolderStructure, bPlainText)
  export_xml (stPath, bRecursive, bExportFolderStructure, bPlainText)
  find (namePath)
  find (stName, recursive)
  get_address ()
  get_all_parameters ()
  ...
  move (newParent, nNewIndex)
  plug (stName, id, stModuleId)
  plug (stName, iType, stId, stVersion, stModuleId)
  reboot_plc ()
  remove ()
  rename (stNewName)
  reset_diagnosis_messages ()
  set_communication_address (address)
  set_gateway_and_address (stGateway, stAddress)
  set_gateway_and_address (gateway, stAddress)
```

set_parameter (parameter, value)
set_parameter_iec_address (identifier, connectorId, iecAddress)
set_parameter_io_variable_mapping (identifier, connectorId, variable, createVariable = False)
set_parameter_io_variable_mapping (identifier, connectorId, subElementIndex, variable, createVariable = False)
set_simulation_mode (bSimulation)
unplug ()
update (id, stModuleId)
update (iType, stId, stVersion, stModuleId)

Microsoft Visual Studio と PTVS で Logic Builder Shell を使用する

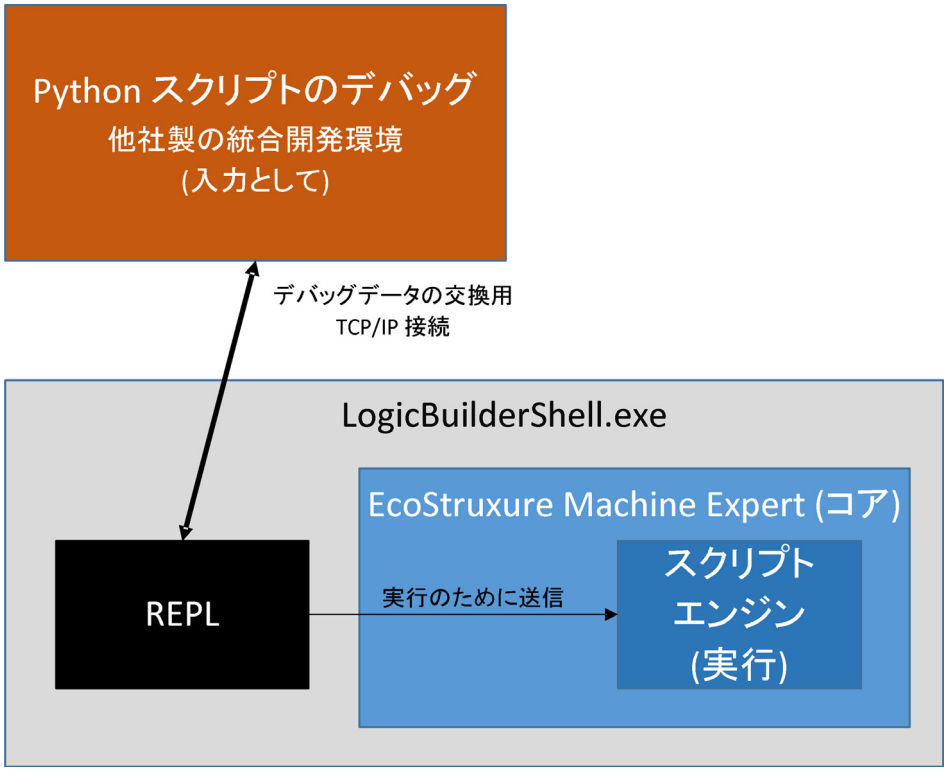
概要

EcoStruxure Machine Expert と LogicBuilderShell.exe を使用すると、Python スクリプトを開発およびデバッグできます。EcoStruxure Machine Expert Python スクリプトを開発するには、*概要*の章 で紹介されているエディターを使用できます (*765* ページ)。

Python Tools for Visual Studio (PTVS) 拡張機能がインストールされた Microsoft Visual Studio などのサードパーティーベンダーの IDE を使用することを推奨します。Visual Studio は Microsoft の製品であり、EcoStruxure Machine Expert には含まれていません。

注記：Visual Studio (PTVS) の Python ツール (*https://pytools.codeplex.com/* を参照) は、無料のオープンソースなのですべてのバージョンが EcoStruxure Machine Expert LogicBuilderShell.exe のバージョンと互換性があるとは限りません。

次の図では、EcoStruxure Machine Expert LogicBuilderShell.exe と連携している 2 つのツールを示しています。



開発とデバッグの手順

ツールのインストールおよび設定後、次の手順に従います。

手順	手順内容
1	Visual Studio ソリューションを作成します。
2	Python プロジェクトを追加します。
3	Python スクリプトを追加します。
4	LogicBuilderShell.exe を インタプリタとして使用するようにプロジェクトを設定します。
5	スクリプトを作成し、Visual Studio で実行します。 **結果** : Visual Studio は次を行います。 ● LogicBuilderShell.exe の実行。 ● Visual Studio (+PTVS) と LogicBuilderShell.exe 間の TCP/IP 接続の確立。 ● Python ステートメントのシェルへの送信 ● 結果のフィードバック取得

Microsoft Visual Studio を使ったスクリプトの開発

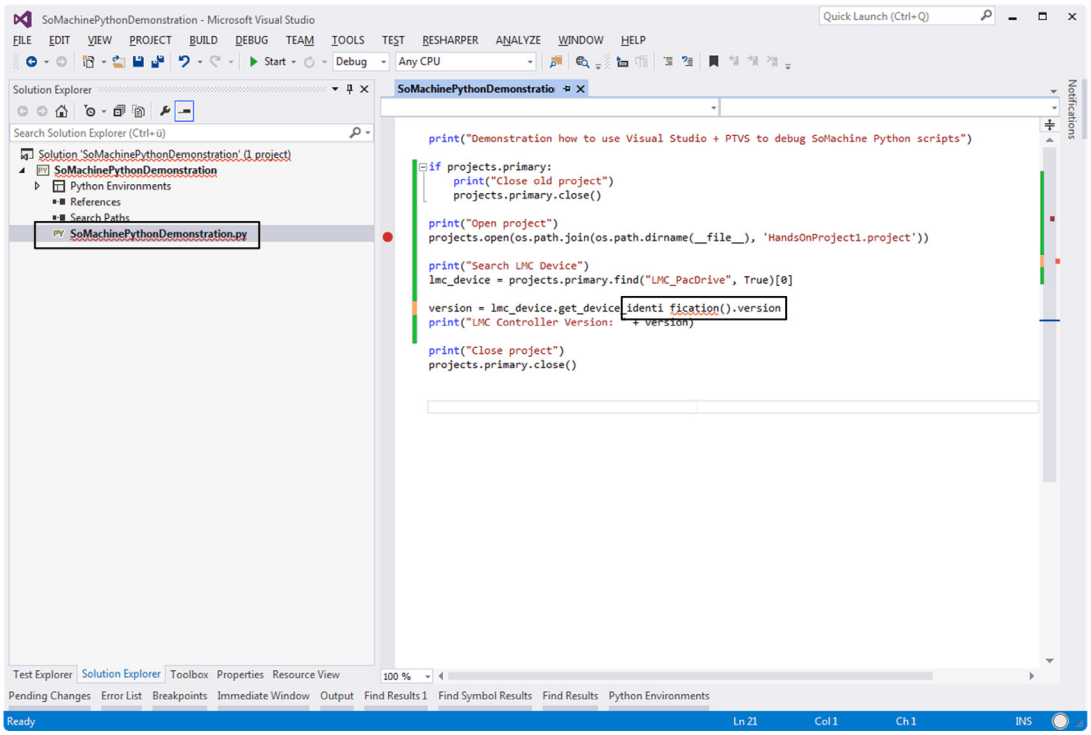
Visual Studio ではスクリプトの中でブレークポイント使用したりデバッガーコマンドの Run、Step Into、Step Over などを使用してデバッグができます。

Visual Studio + PTVS では EcoStruxure Machine Expert Python スクリプトを開発することができます。

Visual Studio は以下に対応しています。

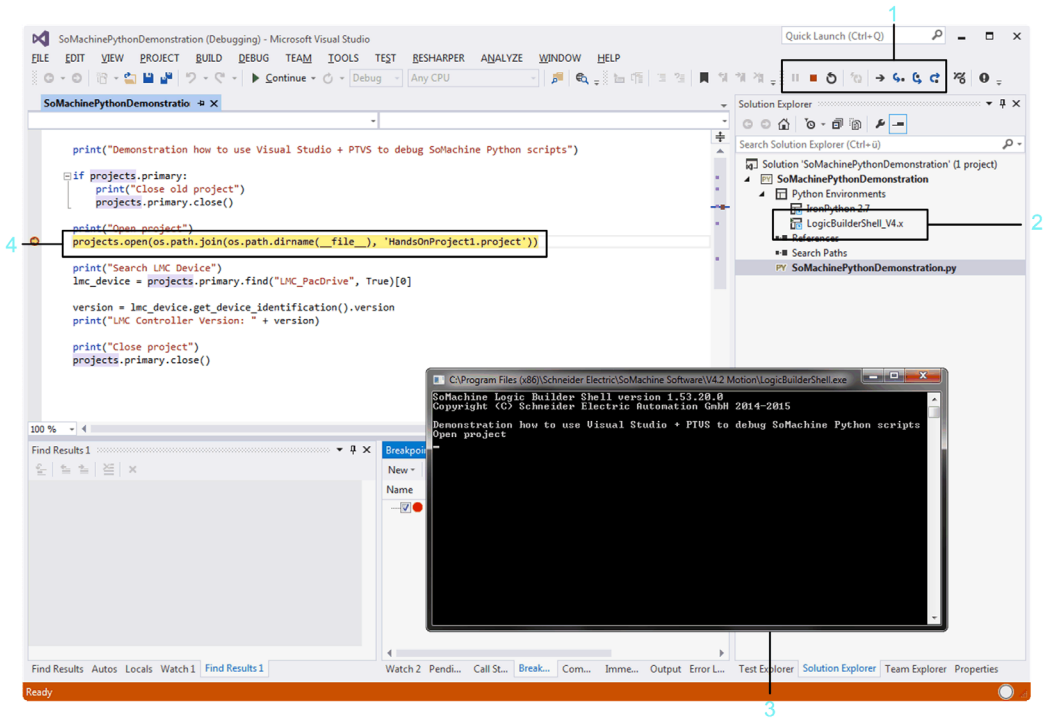
- 記述中の Python シンタックスチェック。
- Intellisense 対応。
- Python スクリプトのシンタックス強調表示。

Visual Studio で、記述中の Python スクリプトのシンタックスチェック。



Start Debugging を Visual Studio でクリックすると、LogicBuilderShell.exe が起動しスクリプトの実行が開始します。ブレークポイントをスクリプトで設定した場合にブレークポイントがヒットすると、Visual Studio によって現在のステートメントがマークされ、デバッガーコマンド (Step-Into、Step-Over、Run、...) を使用することができます。起動された Logic Builder Shell でスクリプトの出力が表示されます。

Visual Studio を使用した Python スクリプトのデバッグ



1 デバッガーコマンド
2 デバッグで使用するために設定された Python 環境
3 Visual Studio で起動された LogicBuilderShell.exe
4 Python スクリプトのブレークポイント

スクリプトデバッグのシステム要件

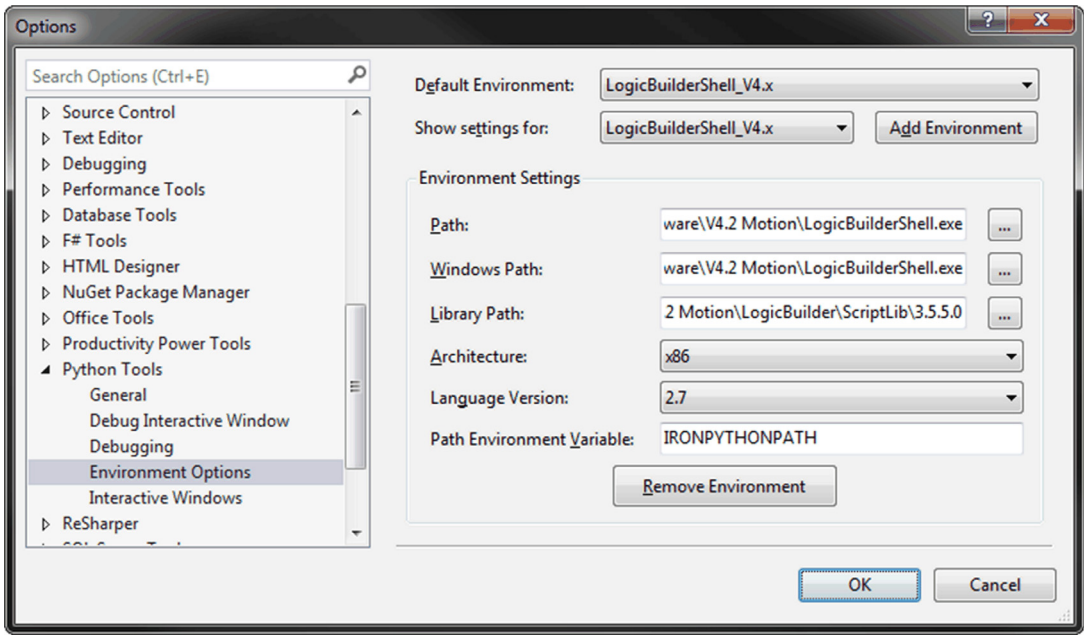
EcoStruxure Machine Expert Python スクリプトをデバッグするには、ご使用のシステムが次の前提条件を満たしている必要があります。

- Logic Builder Shell 機能付きの EcoStruxure Machine Expert インストール済み。
- Visual Studio インストール済み。
- Python Tools for Visual Studio (PTVS) インストール済み
- environment configured in Visual Studio で設定された Logic Builder Shell 環境 (次のパラグラフ参照)。

Logic Builder Shell を Visual Studio で設定する

Logic Builder Shell 機能を使用し、インストールされている各 EcoStruxure Machine Expert バージョンのシステム上で、Visual Studio で Logic Builder Shell 環境を 1 度設定します。

手順	手順内容
1	Visual Studio を起動します。
2	**ツール → オプション → Python Tools → Environment Options** コマンドを実行します。 **結果** : **オプション**ダイアログボックスが表示されます。
3	**オプション** ダイアログボックスで、**Add Environment** ボタンをクリックし例えば、**LogicBuilderShell_V4.x** という名前で環境を作成します。
4	Logic Builder Shell を **Default Environment** として設定します。
5	LogicBuilderShell.exe への**パス** (例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\Vx.x\LogicBuilderShell.exe) を設定します。
6	LogicBuilderShell.exe への **Windows Path** (例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\Vx.x\LogicBuilderShell.exe を設定します。
7	ScriptLib フォルダーへの **Library Path** (例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\Vx.x\LogicBuilder\ScriptLib\3.5.5.0 を設定します。
8	**アーキテクチャー**を **x86** に設定します。
9	**Language Version** を **2.7** に設定します。



Visual Studio 対処方法と Python プロジェクトの設定

次の手順に従って、すべての対処方法または Python プロジェクトに対して、Visual Studio 対処方法と Python プロジェクトを設定します。

手順	手順内容
1	Visual Studio. の起動。
2	新しい対処方法を作成します。
3	新しい IronPython アプリケーションを作成します。
4	対処方法でプロジェクトを右クリックし、コンテキストメニューから**プロパティ**コマンドを実行します。
5	**全般**タブで、設定してある **LogicBuilderShell_V4.x** を **Interpreter** として選択します。
6	**デバッグ**タブで、**IronPython (.NET) launcher** を **Launch mode** として選択します。

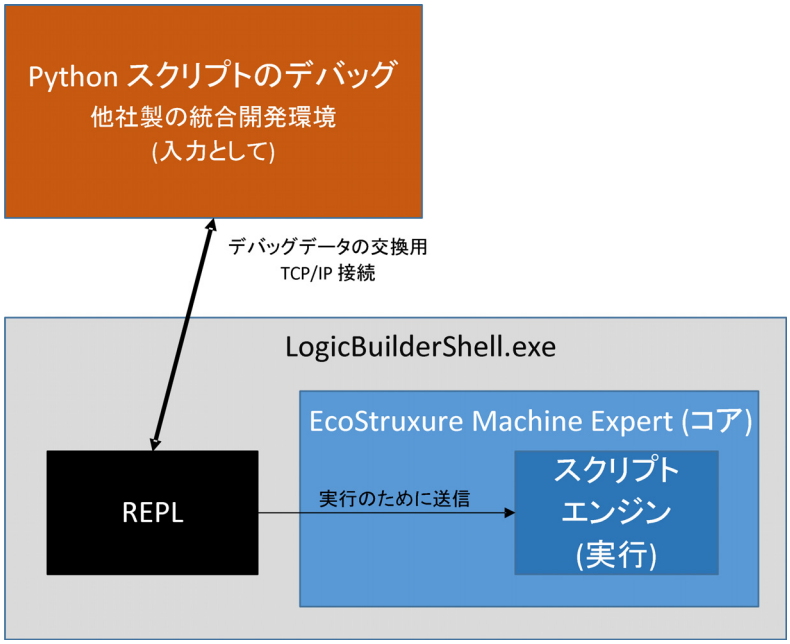
JetBrains PyCharm で Logic Builder Shell を使用する

概要

EcoStruxure Machine Expert と LogicBuilderShell.exe を使用すると、Python スクリプトを開発およびデバッグできます。EcoStruxure Machine Expert Python スクリプトを開発するには、*概要*の章 で紹介されているエディターを使用できます (*765* ページ)。

JetBrains PyCharm などのサードパーティーベンダーの IDE を使用することを推奨します。PyCharm は JetBrains の製品で無料のコミュニティーエディションと購入可能なプロフェッショナルエディションが含まれています。EcoStruxure Machine Expert には付属されていませんが、*https://www.jetbrains.com/pycharm* からダウンロードすることができます。PyCharm はクロスプラットフォームで Windows、macOS、および Linux に対応しています。

次の図では、PyCharm が EcoStruxure Machine Expert LogicBuilderShell.exe とどのように連携しているかを示しています。



開発とデバッグの手順

ツールのインストールおよび設定後、次の手順に従います。

手順	手順内容
1	PyCharm を起動します。
2	PyCharm プロジェクト、または　Python ファイルを作成、または開きます。
3	スクリプトを作成し、PyCharm で実行します。 **結果** : PyCharm は次を行います。 ● LogicBuilderShell.exe の実行 ● PyCharm と LogicBuilderShell.exe 間の TCP/IP 接続の確立 ● Python ステートメントのシェルへの送信 ● 結果のフィードバック取得

JetBrains PyCharm を使ったスクリプトの開発

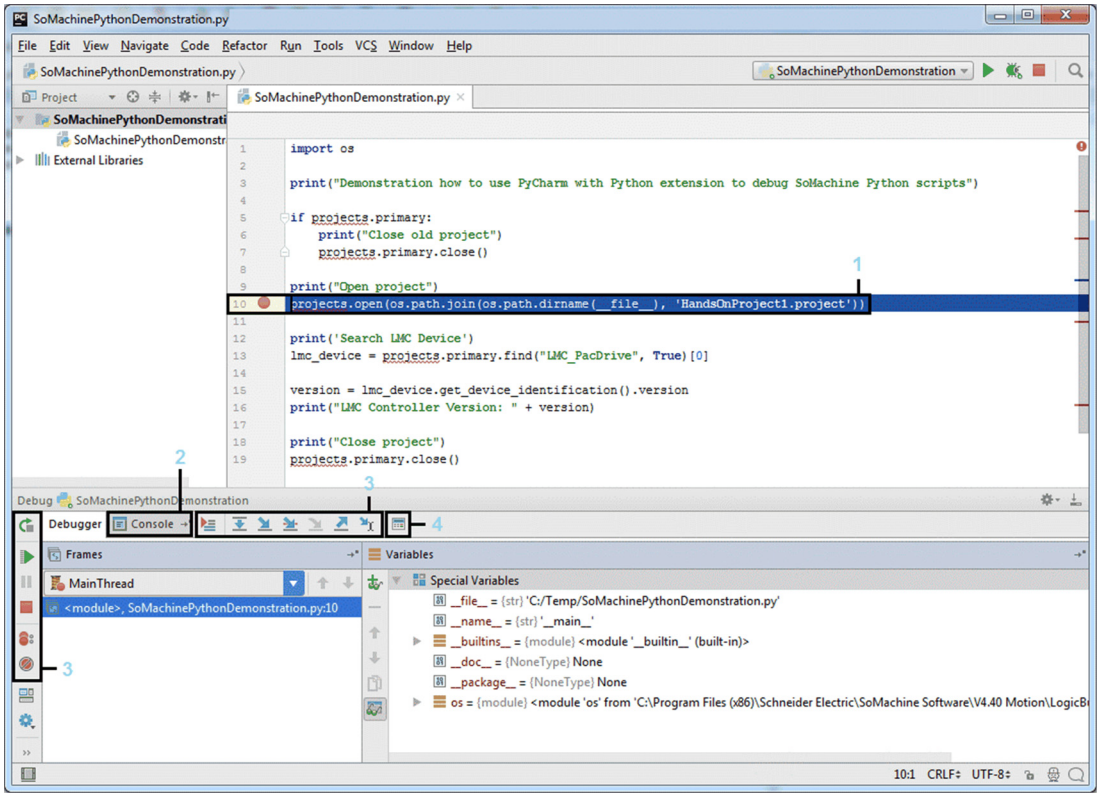
PyCharm で EcoStruxure Machine Expert Python スクリプトを開発することができます。

次の機能がサポートされています。

- 記述中の Python シンタックスチェック。
- ブレークポイントの設定、デバッガーコマンド Step Into、Step Over、Step Out などの使用によるデバッグ。
- Python スクリプトのシンタックス強調表示。
- オートコンプリート対応。(注意 : 外部モジュール内のコードは解決できません。)

Run → Debug... コマンド実行、または **Alt+Shift+F9** を PyCharm で押して Python スクリプトを実行するために選択すると、LogicBuilderShell.exe が起動しスクリプトの実行が開始します。ブレークポイントをスクリプトで設定した場合にブレークポイントがヒットすると、PyCharm よって現在のステートメントがマークされ、デバッガーコマンドを使用することができます。スクリプト出力が **Console** に表示されます。

PyCharm を使用した Python スクリプトのデバッグ



1 Python スクリプトのブレークポイント
2 PyCharm で開始された LogicBuilderShell.exe のデバッグコンソール
3 デバッガーコマンド
4 Python ステートメントを直接発行するための式の評価ダイアログを開きます

For further information on how to use PyCharm デバッグの仕方についての詳細は、
https://www.jetbrains.com/help/pycharm/2017.1/debugging.html を参照してください。

スクリプトデバッグのシステム要件

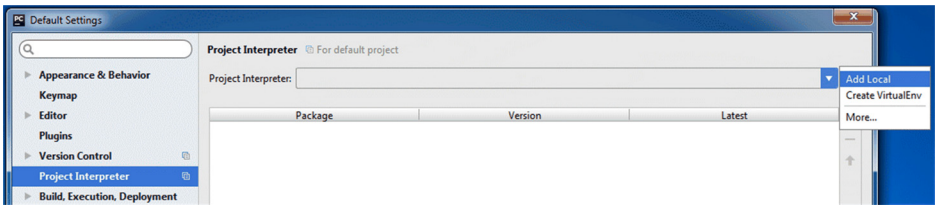
EcoStruxure Machine Expert Python スクリプトをデバッグするには、ご使用のシステムが次の前提条件を満たしている必要があります。

- Logic Builder Shell 機能付きの EcoStruxure Machine Expert インストール済み。
- PyCharm インストール済み。
- PyCharm で設定された Logic Builder Shell の環境 (次の *Logic Builder Shell を PyCharm で設定する* (*788* ページ) を参照してください)。

Logic Builder Shell を PyCharm で設定する

注記：PyCharm を使用できるようにするには、対応する EcoStruxure Machine Expert のインストールに適用する必要のあるアクションがひとつあります。PyCharm は、Python インタプリタ用にオリジナルの実行ファイル名が設定されていることのみを想定、かつ許可しているので、LogicBuilderShell.exe は適切な名前を付ける必要があります。これを実現するためには、以下の手順 1 を実行します。

Logic Builder Shell 機能を使用し、インストールされている各 EcoStruxure Machine Expert バージョンのシステム上で、PyCharm で Logic Builder Shell 環境を 1 度設定します。

手順	手順内容
1	LogicBuilderShell.exe の名前を変更します。 ● Windows Explorer で LogicBuilderShell.exe (例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\V4.40 Motion) に移動します。 ● LogicBuilderShell.exe ファイルと LogicBuilderShell.exe.config ファイルをコピー貼り付けをして同じディレクトリーに複製します。 ● Logic Builder は、IronPython を Python インタプリタとして使用するため、ファイル名を ipy.exe と ipy.exe.config に変更します。
2	PyCharm を起動し、**Configure → Settings** から **Settings** ダイアログボックスを開きます。
3	**Project Interpreter** ビューを開き、**Add Local** コマンドを **Project Interpreter:** ボックスの右側のボタンから実行します。  **結果** : この処理が完了すると (しばらく時間がかかります) **Project Interpreter** が表示されデフォルトとして設定されます。
4	**More...** コマンドを **Project Interpreter:** ボックスの右側のボタンから実行します。
5	新しく作成されたエントリーを選択し、右側のペンシルを選択してプロパティを編集します。
6	名前を LogicBuilderShell_V4.x に変更します。x は、the Logic Builder Shell の特定のバージョンを示します。
7	**OK** をクリックして設定を適用し、ダイアログボックスを閉じます。

PyCharm Python インタプリタの設定方法については、
https://www.jetbrains.com/help/pycharm/2017.1/configuring-available-python-interpreters.html を参照してください。

既存の PyCharm プロジェクト、または Python ファイルの設定

前セクションの *Configuring the Logic Builder Shell を PyCharm で設定する* (*788* ページ) で説明されている手順に従って PyCharm プロジェクトを作成すると、新しく作成された Logic Builder Shell Python インタプリタは事前設定されます。

既存の Python プロジェクト、または 1 つのファイルを設定するには、次の手順で行います。

手順	手順内容
1	PyCharm を起動します。
2	既存のプロジェクト、または Python ファイルを開きます。
3	黄色くハイライトされたヘッダーのメッセージの **Configure Python Interpreter** リンクをクリックします。
4	**Settings** ダイアログボックスで、先に **Project Interpreter** として作成された LogicBuilderShell_V4.x を選択し、**OK** をクリックして確認します。

PyCharm プロジェクトインタプリタの設定方法については、
https://www.jetbrains.com/help/pycharm/2017.1/configuring-python-interpreter-for-a-project.html を参照してください。

Microsoft Visual Studio Code および Python 拡張機能で Logic Builder Shell 使用する

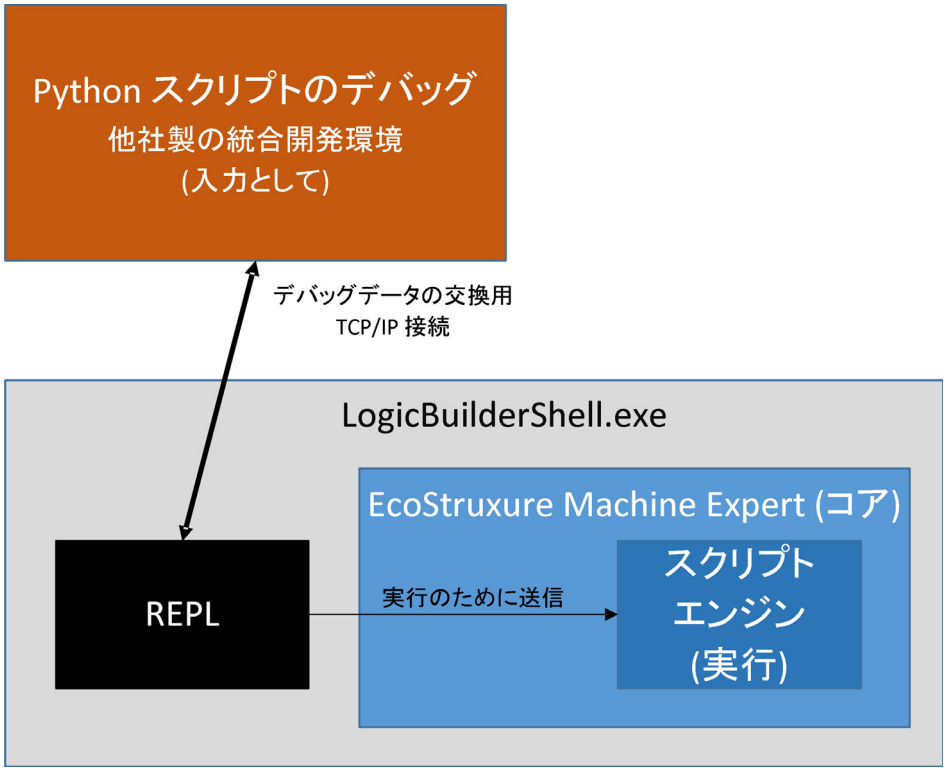
概要

EcoStruxure Machine Expert と LogicBuilderShell.exe を使用すると、Python スクリプトを開発およびデバッグできます。EcoStruxure Machine Expert Python スクリプトを開発するには、*概要*の章 で紹介されているエディターを使用できます (*765* ページ)。

Python 拡張機能がインストールされた Microsoft Visual Studio Code などのサードパーティーベンダーの IDE を使用することを推奨します。Visual Studio Code は、Microsoft のオープンソース製品です。EcoStruxure Machine Expert には付属されていませんが、*https://code.visualstudio.com* からダウンロードすることができます。Visual Studio Code はクロスプラットフォームで Windows、macOS、およびいくつかの Linux ディストリビューションに対応しています。

注記: Python 拡張機能 (*https://marketplace.visualstudio.com/items?itemName=donjayamanne.python* を参照) は無料のオープンソースなのですべてのバージョンが EcoStruxure Machine Expert LogicBuilderShell.exe のバージョンと互換性があるとは限りません。

次の図では、EcoStruxure Machine Expert LogicBuilderShell.exe と連携している 2 つのツールを示しています。



開発とデバッグの手順

ツールのインストールおよび設定後、次の手順に従います。

手順	手順内容
1	Visual Studio Code を起動します。
2	Python ファイルを作成、または開きます。
3	スクリプトを作成し、Visual Studio Code で実行します。 **結果** : Visual Studio Code は次を行います。 ● LogicBuilderShell.exe の実行 ● Visual Studio Code と LogicBuilderShell.exe 間の TCP/IP 接続の確立 ● Python ステートメントのシェルへの送信 ● 結果のフィードバック取得

Microsoft Visual Studio Code を使ったスクリプトの開発

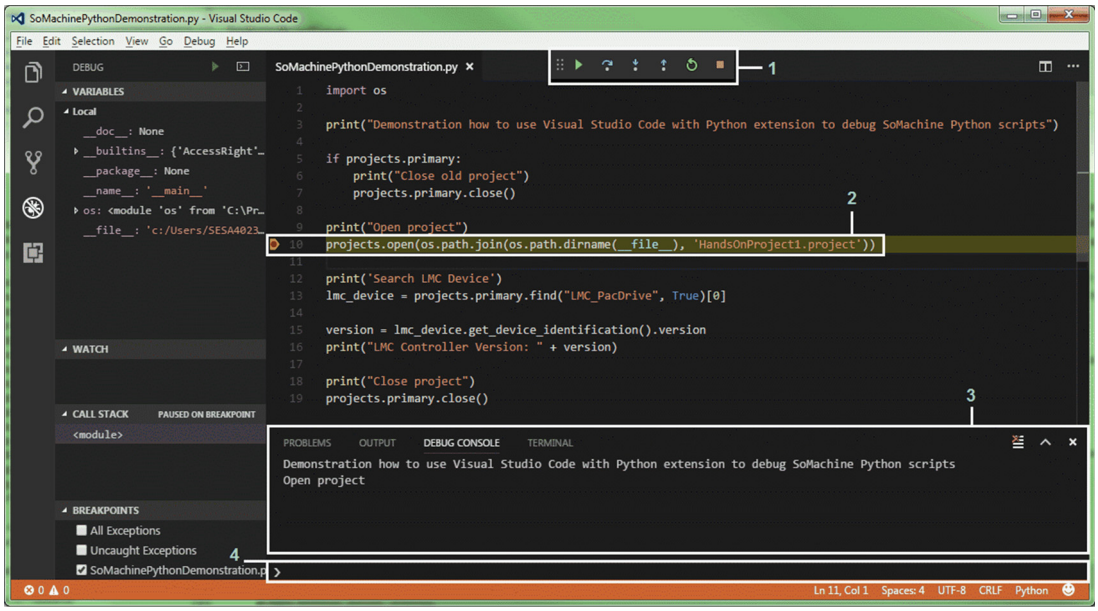
Python 拡張機能付きの Visual Studio Code を使用し、EcoStruxure Machine Expert Python スクリプトを開発することができます。

次の機能がサポートされています。

- ブレークポイントの設定、デバッガーコマンド Step Into、Step Over、Step Out、および Continue の使用によるデバッグ
- Python スクリプトのシンタックス強調表示。
- ローカル要素とコードスニペットのための IntelliSense。(注意 : 外部モジュール内のコードは解決できません。)

Debug → Start Debugging 実行後、または Visual Studio Code で **F5** を押すと、LogicBuilderShell.exe が起動しスクリプトの実行が開始します。ブレークポイントをスクリプトで設定した場合にブレークポイントがヒットすると、Visual Studio Code よって現在のステートメントがマークされ、デバッガーコマンドを使用することができます。スクリプト出力が **Debug Console** に表示されます。

Visual Studio Code を使用した Python スクリプトのデバッグ



1 デバッガーコマンド
2 Python スクリプトのブレークポイント
3 Visual Studio Code で開始された LogicBuilderShell.exe のデバッグコンソール
4 Python ステートメントを直接発行する入力ボックス

For further information on how to use Visual Studio Code デバッグの仕方についての詳細は、*https://code.visualstudio.com/docs/editor/debugging* を参照してください。

スクリプトデバッグのシステム要件

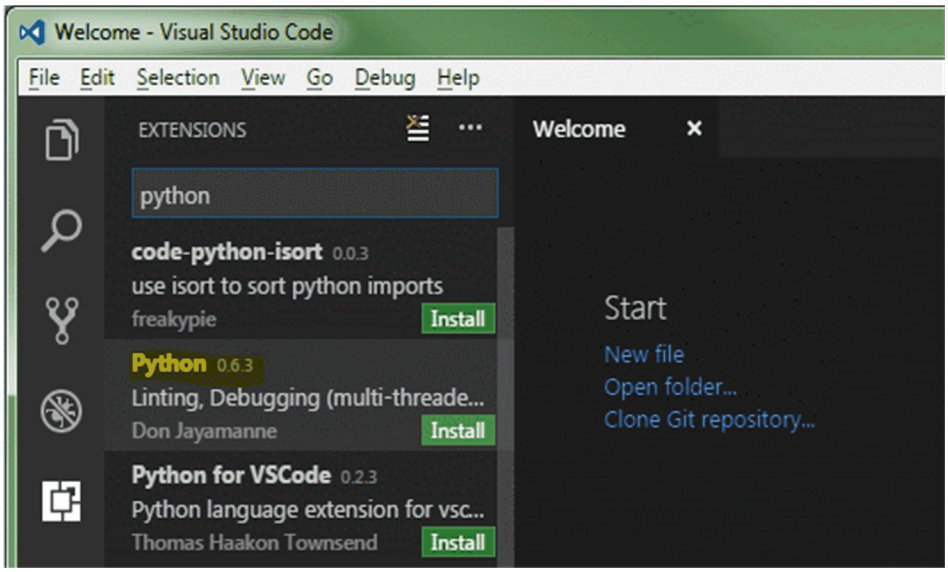
EcoStruxure Machine Expert Python スクリプトをデバッグするには、ご使用のシステムが次の前提条件を満たしている必要があります。

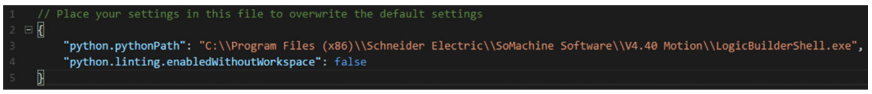
- Logic Builder Shell 機能付きの EcoStruxure Machine Expert インストール済み。
- Visual Studio Code インストール済み。
- Visual Studio Code Python 拡張機能インストール、および設定済み (次のセクション *Python 拡張機能の Visual Studio Code でインストールおよび設定* を参照)。

Python 拡張機能の Visual Studio Code でインストールおよび設定

次の手順 (*https://code.visualstudio.com/docs/editor/extension-gallery* にも記載) に従って、Python 拡張機能を Visual Studio Code で 1 度インストールする必要があります。

手順	手順内容
1	Visual Studio Code を起動します。

手順	手順内容
2	**View → Extensions** コマンドを実行、または **Ctrl+Shift+X** を押して検索ボックスに python を入力します。 **結果** : 実行ビューが開き、対応する拡張機能のリストが表示されます。 
3	対応するエントリーの **Install** ボタンをクリックして Instal Python 拡張機能をインストールします。 **注記**：複数の異なる Python の実装が存在するため、正しい拡張機能をインストールするように確認します。 **注記**：インストールでエラーが発生した場合は、インターネット接続を確認します。接続しているネットワークの既存のプロキシは、http.proxy* 設定を使用して Visual Studio Code で構成する必要があります。 Visual Studio Code についての詳細は、*https://code.visualstudio.com/docs/getstarted/settings* を参照してください。
4	Python 拡張機能のインストールが正常に完了後、Visual Studio Code を手動で再起動、または、**Reload** ボタンを押します。

5	Visual Studio Code の設定で、LogicBuilderShell.exe へのパス (例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\V4.40 Motion\LogicBuilderShell.exe) を設定します。
	File → Preferences → Settings コマンドを実行して settings.json ファイルを開く、または **Ctrl+,** を押して次の行を中括弧の中に追加します (インストールのパスに合わせます)。
	`"python.pythonPath": "C:¥¥Program Files (x86)¥¥Schneider Electric¥¥SoMachine Software¥¥V4.40 Motion¥¥LogicBuilderShell.exe",`
	`"python.linting.enabledWithoutWorkspace": false`
	設定ファイルにすでにエントリーが含まれている場合は、前の行に最後のカンマを追加します。
	その後は、このようになります。
	 

Visual Studio Code についての詳細は、*https://code.visualstudio.com/docs/getstarted/settings* を参照してください。

実行スクリプト

概要

EcoStruxure Machine Expert ユーザーインターフェイスから一連のスクリプトコマンドを含むスクリプトファイル (*filename.py*) を実行できます。

EcoStruxure Machine Expert ユーザーインターフェイスからのスクリプト実行の詳細は、*スクリプト関連コマンド (EcoStruxure Machine Expert, Menu Commands, Online Help 参照)* の章を参照してください。

バッチファイル

よく使用されるコマンド

コマンド	詳細
-　REM または ::	その行はコメントで、無視されます。
cd	別のディレクトーに変更します。
echo off	このコマンドは表示されません。 単一のコマンドが表示されるのを避けるには、コマンドの前に @ 文字を挿入します。
echo	プログラミングコンソールに文字列または変数を表示します。
set	変数を宣言し、この変数に値を代入します。
>	出力をファイルに書き込みます。すでにファイルが存在する場合は、上書きされます。
>>	出力をファイルに追加します。ファイルが存在しない場合は、ファイルが作成されます。

アプリケーション例 :

```
@echo off
REM Go to the directory where EcoStruxure Machine Expert is installed
cd "<Replace this with the path to the LogicBuilder.exe, for example,
C:\Program Files (x86)\Schneider Electric\SoMachine Software\>"
REM Run LogicBuilder.exe with no graphical user interface and the full path to the script
LogicBuilder.exe --noui --runscript="<Replace this with the full file path
where the script is stored, for example, D:\MyScripts\TestScript.py>"
pause
```

C# コンソールアプリケーション

スクリプトがエンジンで実行される前に C# アプリケーションでスクリプトを実行することで、スクリプトを動的に編集できます。また、前のステップを C# アプリケーションで実行することもできます。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace ExecuteScriptExample
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            ProcessStartInfo psi = new ProcessStartInfo();

            // Specify the name and the arguments you want to pass
            psi.FileName = @"<Replace this with the path to the LogicBuilderShell.exe, for example,
C:\Program Files (x86)\Schneider Electric\SoMachine Software\LogicBuilderShell.exe";
            psi.Arguments = "\ "<Replace this with the full file path where the script is stored,
for example, D:\MyScripts\TestScript.py\"";
            // Create new process and set the starting information
```

```
            Process p = new Process();
            p.StartInfo = psi;
            // Set this so that you can tell when the process has completed
            p.EnableRaisingEvents = true;
            p.Start();
            // Wait until the process has completed
            while (!p.HasExited)
            {
                System.Threading.Thread.Sleep(1000);
            }
        }
        catch (Exception ex)
        {
            Console.Write(ex.Message);
        }
        Console.ReadKey();
    }
  }
}
```

推奨方法

インデント (タブとスペース)

Blocks in Python のブロックは、ブロックのコードをインデントすることによって作成されます (例えば、構造化テキストの END_IF ステートメント、または C プログラミング言語の中括弧 {" "} とは異なります)。

例 :

C プログラミング言語	Python プログラミング言語

int factorial(int x)
{
if (x > 1)
return x* factorial(x - 1);
else
return 1;
}
	def factorial(x):
	if x > 1:
	return x * factorial(x - 1)
	else:
	return 1

インデント用にタブとスペースを混在させる場合は、次の点に注意してください。In Python では、タブ文字は内部的に 8 つのスペースに置き換えられます。多数のエディターがデフォルトでスペース 4 つを使用するため、コードエラーの発見が困難になる場合があります。ソースコードのブロックのインデントが同じサイズに見えても、実際のインデントは異なります。

従って、コピー貼り付けコマンドを使用するときは注意が必要です。

これを避けるには、エディターで自動的にタブ文字をスペースに置き換えるように設定します。

大文字と小文字の区別

Python は、大文字と小文字を区別する言語です (構造化テキストとは対照的です)。つまり、MyVar と MYVAR は 2 つの異なる変数を参照します。

パス名のエンコード

Python はバックスラッシュ文字 ¥ を使用して特殊文字をエンコードします。例シーケンス ¥n は改行文字をエンコードします。

バックスラッシュ文字は、Windows ファイルシステムのパス名のエンコードにも一般的に使用されています (例えば、D:\PythonProjects\SetParameter.project)。

これは、Python スクリプトでパス名をエンコードするときに問題を引き起こす可能性があります。

可能な解決策は 2 つあります。

- すべてのバックスラッシュを ¥¥ シーケンスでエンコードします (例えば、project_path = "D:¥¥PythonProjects¥¥SetParameter.project")。
- または、開始引用符の前に r を付けることでデフォルトの動作を無効にします (for example, sequence (例えば、project_path = r"D:¥PythonProjects¥SetParameter.project")。

コロン (:)

ループ宣言とコロンで終わる条件

例 :

```
if len(messages) == 0:
    print("--- Build successful ---")
```

.NET API ドキュメントの読み方

Python プログラマー用の .NET API ドキュメントの読み方

現在のスクリプトインターフェイスドキュメントの暫定版は、基本的な .NET / C# ソースから自動生成されます。そのため、Python プログラマーには一般的ではない用語が含まれています。

次のリストは、それらを Python の考え方に変換するためのヒントです。

- .NET のインターフェイスは、インターフェイスのクラス実装よって供給されるメンバー (メソッド、プロパティ) についてのコントラクトです。IronPython では、メンバーは 1 つまたは複数の .NET インターフェイスを、基本のクラスを使用してそれらを派生させることによって実装できます。宣言にインターフェイスで宣言されたメンバーがない場合、ランタイムで例外が発生します。(*DeviceImportFromSvn.py* の例は、`ImportReporter` インターフェイスのクラス実装です。)
- .NET では、すべてのパラメーター、プロパティ、およびファンクションの戻り値は静的にタイプ分けされます。許可されたタイプは、パラメーター名の前に注釈が付けられます。ファンクションでは、ファンクション名の前に戻り値のタイプが付きます。親クラス (またはインターフェイス) が記述されている場合、サブクラスのインスタンスを使用できます。`void` は戻り値の無いファンクションを表します。
- メソッドはオーバーロードされる場合があり、クラスは同じ名前の複数のメソッドをもつことができるが、パラメーターの数およびタイプは異なります。IronPython は一致するバリアントを自動的に呼び出します。
- INT タイプは -2,147,483,648...2,147,483,647 の間の整数、BOOL は Python の BOOL 型 (TRUE および FALSE) と同じ、STRING 型は Python の `str` 型および unicode (IronPython と同じ) と同じです。IDictionary<Object, Object> は、通常の Python ディクショナリーを意味します。IronPython は、Python と .NET タイプ間を自動的に変換します。
- タイプ T が `IBaseObject<T>` からの派生であれば、そのタイプは他のプラグインによってより多くのメンバーで拡張できます。パラメーターまたは戻り値で T タイプを実際に使用する場合、`IExtended-Object<T>` でマーク付けされます。
- インターフェイス `IEnumerable<T>` は、T タイプ (またはサブクラス) のオブジェクトのみを生成するシーケンス (リスト、配列、ジェネレーター) を記述します。シーケンスが互換性の無いオブジェクトを生成する場合、ランタイムで例外が発生します。
- インターフェイス `IList<T>` は、T タイプ (またはサブクラス) のオブジェクトのみを含むリストを記述します。互換性の無いオブジェクトを追加しようとすると、例外が発生します。
- 構文 `params T [] name` は、変数割り当てリストの Python 構文の `*name` に等しいが、パラメーターは、T タイプ (またはサブクラス) に制限されます。
- 列挙型 (ENUM) は、Python の言語構造としては存在しません。それらは、例えば、1 週間の日数などの固定の定数を定義するために使用します。.NET で定義された列挙型の値は、IronPython で Name.Member 構文 (静的なクラスメンバーに類似) (例、`OnlineChangeOption.Try`) を介してアセクスできます。例えば、*http://pypi.python.org/pypi/enum/* または *http://www.ironpython.info/index.php/Enumerations* などのように、Python での列挙型のエミュレートには複数の方法があります。
- `{ get; set; }` でマーク付けされたプロパティは読み書き可能、`{ get; }` でマーク付けされたプロパティは読み取り専用です。それらは、Python の `@property` デコレーターに類似しています。

スクリプトでは以下のエントリーポイントが使用できます :

- `system`: EcoStruxure Machine Expert システムでの統合用基本ファンクション。このオブジェクトには、EcoStruxure Machine Expert の終了、`ui_present` コマンドにより `--noUI` モードが実行されている場合のメッセージウィンドウまたはクエリへのアクセスのような *ISystem インターフェイス* 下に記述されているすべてのファンクションがあります。
- `projects`: プロジェクト管理用基本機能。このオブジェクトには、プロジェクトおよびプロジェクトアーカイブの読み込みのような *IScriptProjects インターフェイス* 下に記述されているすべてのファンクションがあります。さらに、それが個別のプロジェクトへのエントリーポイントです。
- `online`: デバイスへのオンライン接続用基本機能 `create_online_application` メソッドを使用して、アプリケーションオブジェクトのオンラインオブジェクトを作成できます。このオンラインプロジェクトによってコントローラーへログインし、アプリケーションの起動および変数値の取得ができます。

EcoStruxure Machine Expert Python API

エントリーポイントの詳細

ドライバー	名前 (タイプ)	詳細
システム	system (ISystem)	EcoStruxure Machine Expert システムへの統合用基本ファンクション
	Severity	ニュースの優先度用 ENUM
	Guid	**G**lobally **u**nique **id**entifier (グローバルで固有な識別子) 用データ型
	PromptResult	ユーザーリクエストの戻り値用 ENUM
	MultipleChoiceSelector	複数選択肢のプロンプト用デリゲートタイプ
	PromptChoiceFilter	
プロジェクト	projects (IScriptProjects)	プロジェクト管理の基本ファンクション
	ExportReporter	エクスポート中のイベント処理用インターフェイス
	ImportReporter	インポート中のイベント処理用インターフェイス
	ConflictResolve	インポート中の競合解決用 ENUM
オンライン	online (IScriptOnline)	デバイスへのオンライン接続用基本ファンクション
	OnlineChangeOption	デバイスへのログイン中のダウンロード用 ENUM
	ApplicationState	アプリケーションの状態用 ENUM
	OperatingState	オペレーションの状態用 ENUM
	ValuesFailedException	オンラインの式でエラーが検出されたことによる例外
	TimeoutException	オンライン処理中のタイムアウトによる例外
DeviceObject	DeviceID	デバイス識別用のカプセル化タイプ

詳細は、本書の *Schneider Electric Script Engine 例* の章 (*803* ページ)、および EcoStruxure Machine Expert で提供されている ScriptEngine.chm ファイル参照してください。

EcoStruxure Machine Expert スクリプト - Python API

概要

EcoStruxure Machine Expert は、EcoStruxure Machine Expert スクリプトで使用できる Python API を提供します。

Python API は 2 つのカテゴリーで構成されています。

- 標準 Python API および Python モジュール / パッケージ
- EcoStruxure Machine Expert Python API

標準 Python API (文字列、配列、ファイルなどを扱う) は、EcoStruxure Machine Expert のインストールの一部です。詳細なヘルプは、インターネット (例えば、*https://docs.python.org/3/*) を参照してください。インターネットで利用可能な Python モジュールおよびパッケージを使用して、EcoStruxure Machine Expert のインストールを拡張することができます。

EcoStruxure Machine Expert を自動化するために EcoStruxure Machine Expert Python API は EcoStruxure Machine Expert のインストールに含まれています。例えば、プロジェクトを開く、閉じる、プロジェクトの内容を変更する、プロジェクトをコンパイルしてコントローラーにダウンロードするなどの目的で使用されます。

この章では、EcoStruxure Machine Expert Python API のコンセプトを説明します。スクリプトエンジンの例 (*803* ページ) も参照してください。EcoStruxure Machine Expert Python API を調べるには、EcoStruxure Machine Expert の API を調べる (*779* ページ) の章を参照してください。

メソッド / ファンクションの説明、またはパラメーターの説明については、EcoStruxure Machine Expert のオンラインヘルプの *ソフトウェア*部分の*スクリプトエンジン*の章を参照してください。

オブジェクト指向プログラミングに基づく EcoStruxure Machine Expert Python API コンセプト

ご使用の Python スクリプトに適した API メソッドを見つけるには、EcoStruxure Machine Expert Python API のコンセプトと API がどのようにスクリプトエンジンに統合されているか (スクリプトの実行) に慣れておく必要があります。EcoStruxure Machine Expert Python API の背後にある主なコンセプトはオブジェクト指向の API アプローチです。オブジェクト指向プログラミング (OOP) は、オブジェクトとコードという 2 つの概念に基づくプログラミングパラダイムです。オブジェクトは、属性とも呼ばれるフィールドの形式でデータを含むデータ構造です。コードは、メソッドとも呼ばれるプロシージャの形で利用できます。オブジェクトのプロシージャは、それらが関連付けられているオブジェクトのデータフィールドにアクセスして変更することができます。OOP (オブジェクト指向プログラミング) に従って設計されているコンピュータープログラムは、互いに対話するオブジェクトで構成されています。

スクリプトを書く際の課題は、詳細な EcoStruxure Machine Expert Python API ドキュメントを正しいオブジェクトにマップしてプロシージャを呼び出すことです。

定義済み変数およびタイプ

(LogicBuilderShell.exe または **スクリプトイミディエイト** ビューで) スクリプトを実行、または REPL を使用する場合、定義済みの変数 (およびタイプ) を含むメインスクリプトスコープができます。それらは、スクリプトで EcoStruxure Machine Expert Python API へのエントリーとして使用できます。メインスクリプトスコープで使用可能な定義済みの変数をリストするには、LogicBuilderShell.exe を起動し `dir()` を実行します。

`dir()` の実行による定義済み変数のリスト :

```
>>> dir()
['AccessRight', 'ApplicationState', 'ArchiveCategories', 'ChannelType', 'Compile
rMessage', 'ConflictResolve', 'ConnectorRole', 'CredentialSourceKind', 'DeviceID
', 'DeviceUserManagementFlags', 'DiagType', 'ExportReporter', 'Guid', 'Implement
ationLanguage', 'ImportReporter', 'MultipleChoiceSelector', 'NativeExportReporte
r', 'NativeImportFilter', 'NativeImportHandler', 'NativeImportResolve', 'Nativel
mportResult', 'ObjectPermissionKind', 'OnlineChangeOption', 'OperatingState', 'P
ermissionState', 'ProjectType', 'PromptChoice', 'PromptChoiceFilter', 'PromptHan
dling', 'PromptResult', 'ResetOption', 'SV_DEV', 'SV_POU', 'Severity', 'TimeoutE
xception', 'ValuesFailedException', 'Version', '__SoMachine__', '__builtins__',
'__doc__', '__file__', '__name__', 'communication_settings', 'compiler_settings'
, 'etest_test_provider', 'feature_settings_manager', 'librarymanager', 'libraryp
ackage_service', 'new_project', 'online', 'projects', 'system', 'visualization_s
ettings']
```

小文字で書かれたこのリストの定義済みエントリーは、メソッドまたはフィールドを提供するオブジェクトを参照する変数です。これは、読み込まれた EcoStruxure Machine Expert プロジェクトの有無にかかわらず機能する一種のグローバル API です。

例えば、system、projects、または online。

- system: EcoStruxure Machine Expert に統合するための機能。このオブジェクトは、EcoStruxure Machine Expert から **メッセージ**ビューへのアクセス、またはプログラムが --noUI モードで実行されているかどうかを確認するための ui_present の使用など、ISystemInterface に記載されている機能を提供します。
- projects: プロジェクト管理用機能。このオブジェクトは、プロジェクトやプロジェクトアーカイブを読み込むなど、IScriptProjects Interface に記載されている機能を提供します。また、これは個々のプロジェクトへのエントリーポイントとなります。
- online: コントローラーへオンラインアクセスするための機能。create_online_application メソッドを使用してアプリケーションオブジェクトに適切なオンラインオブジェクト (IScriptOnlineApplication に記載) を取得できます。このオブジェクトを使用すると、コントローラーにログインしてアプリケーションを起動し、変数値を読み取ることができます。

詳細については、詳細は EcoStruxure Machine Expert のオンラインヘルプの *スクリプトエンジンプラグイン API リファレンス* 部分を参照してください。

大文字で始まるこのリストの定義済みエントリーは、Python スクリプトで使用、またはインスタンス化できる列挙型、またはタイプ / クラスです。例えば、DeviceID (クラス)、Guid (クラス)、または PromptChoice (列挙)。

EcoStruxure Machine Expert ツリー構造での検索およびナビゲーション

EcoStruxure Machine Expert プロジェクトは、デバイス、POU、DUT、GVL、などから構成され、それらはオブジェクトのツリーに編成されています (例えば、Logic Builder の **デバイスツリー**)。このオブジェクトのツリー (プロジェクトツリー) は、projects.primary 添付されています。projects.primary では、API find(…) メソッドを使用してオブジェクトを検索することができます。例えば、get_children() メソッドを使用して直接の子オブジェクト (例えば、コントローラーデバイス) を取得することによって、プロジェクトツリー内を移動できます。それぞれの子オブジェクトで、get_children() をもう一度呼び出してそれらの直接の子オブジェクトを取得することができます。

注記：プロジェクトツリーオブジェクトで使用できるメソッドは、その種類 (POU、DUT、GVL、デバイス) によってオブジェクトごとに異なります。inspectapi.dir(…) を使用して、オブジェクトで使用可能な API メソッドのリストを確認することを推奨します。

例

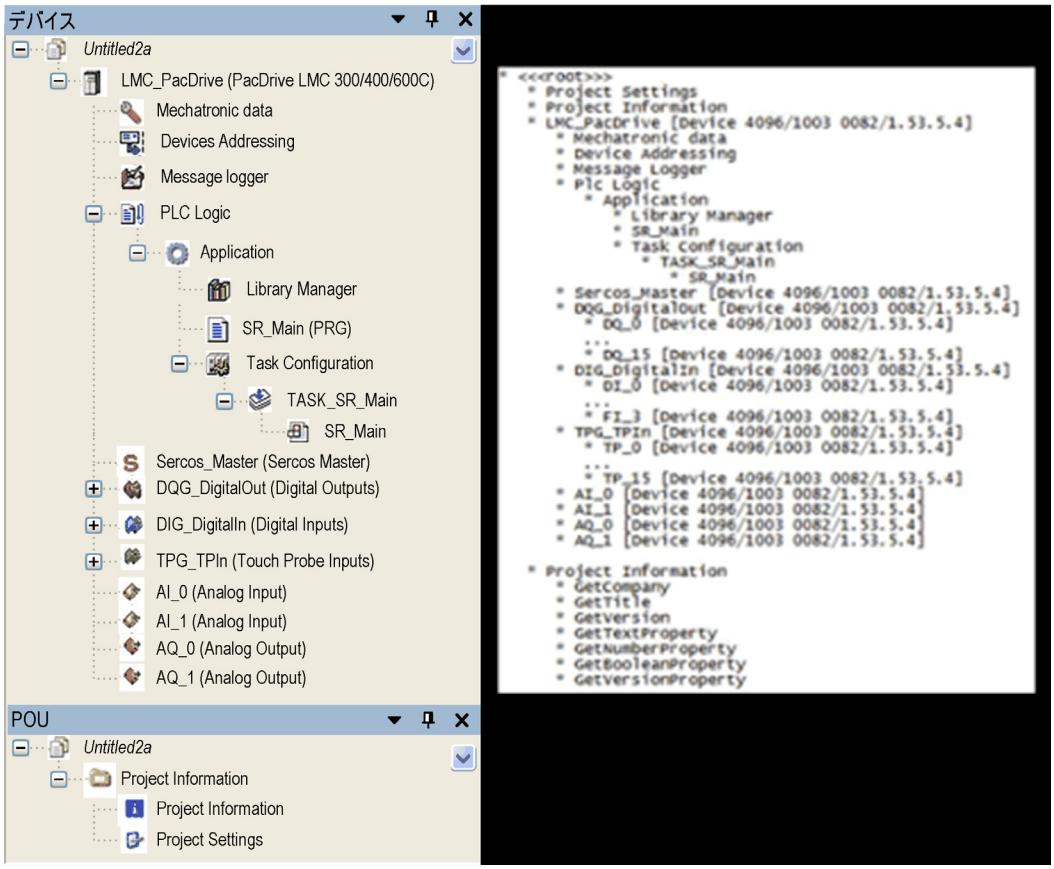
グローバル API 機能を使用してプロジェクトを開く例

```
projects.open("MyProject.project")
```

プロジェクトツリーオブジェクトを検索し名前の変更をする例 (プロジェクト読込み後):

```
myObject = projects.primary.find("SERCOSIII")[0]
myObject.rename("New_SERCOSIII_Name")
```

この図は、ユーザーインターフェイスのプロジェクトツリーオブジェクトと、スクリプトで詳説を表示されたプロジェクトツリーを示しています。



プロジェクトツリーをスクリプトで詳説した例 (上記表示)

```
def print_tree_of_obj(treeobj, depth=0, verbose=False):
    name = treeobj.get_name(False)
    if treeobj.is_device:
        deviceid = treeobj.get_device_identification()
        details = ""
        if verbose == True:
            details = " [Device {0}/{1}/{2}]".format(deviceid.type, deviceid.id, deviceid.version)
        print("{0} * {1}{2}".format("    "*depth, name, details))
    else:
        print("{0} * {1}".format("    "*depth, name))

    for child in treeobj.get_children(False):
        print_tree_of_obj(child, depth+1, verbose)

def print_tree_of_project(pro, verbose=False):
    if pro == None:
        print("No project open.")
    else:
        for obj in pro.get_children():
            print_tree_of_obj(obj, 0, verbose)

def print_tree(verbose=False):
    if projects.primary != None:
        print_tree_of_project(projects.primary, verbose)
    else:
        print("No project open.")

print_tree(True)
```

ツールバーアイコンからスクリプトを呼び出す

概要

ツールバーに表示されるアイコンを設定して、1 回のクリックで最大 32 の異なるスクリプトファイルにアクセスできます。

これを実現するために、それぞれ最大 16 個のシンボルエントリを含む最大 2 つの config.json ファイルを作成します。

以下の場所に config.json ファイルを保存します。

- C:\ProgramData\SoMachine Software\Script Commands
- <SoMM_Installation_Directory>\LogicBuilder\Script Commands、例えば、C:\Program Files (x86)\Schneider Electric\SoMachine Software\V4.40 Motion\LogicBuilder\Script Commands

EcoStruxure Machine Expert を起動すると、スクリプト呼び出しとそれに関連するアイコンが **Command Icons** タブ、**ScriptEngine Commands** カテゴリーの **ツール → カスタマイズ**ダイアログボックスに追加されます。**ツール → カスタマイズ** ダイアログボックスの**ツールバー**タブで、新規または既存のツールバーにそれらを追加できます。

設定ファイル config.json の要素

各設定ファイル config.json には次の必須またはオプションの要素が含まれています。

要素	必須	詳細
"Name"	はい	ツールバーのアイコンのツールチップとしてテキストが表示されます。
"Desc"	いいえ	この要素は EcoStruxure Machine Expert で使用されません。
"Icon"	はい	イメージファイルが config.json ファイルと同じフォルダー (Script Commands) にある場合は、この要素には、イメージファイル (.ico 形式) の名前が含まれます。 イメージファイルがフォルダー (Script Commands) にない場合は、ここにパス情報を追加します。
"Path"	あり	スクリプトファイルが config.json ファイルと同じフォルダー (Script Commands) にある場合は、この要素には、.py 形式の Python スクリプトファイルの名前が含まれます。 スクリプトファイルがフォルダー (Script Commands) にない場合は、ここにパス情報を追加します。

config.json ファイルの例

config.json ファイルには次の内容を含めることががができます。

```
[
  {
    "Name": "Start Process1",
    "Desc": "Processing1",
    "Icon": "process1.ico",
    "Path": "proc1.py",
  },
  {
    "Name": "Start Process2",
    "Desc": "Processing2",
    "Icon": "process2.ico",
    "Path": "proc2.py"
  }
]
```

よって、Script Commands フォルダーには次のファイルが含まれます。

- config.json
- process1.ico
- proc1.py
- process2.ico
- proc2.py

ツール → カスタマイズダイアログボックスの **ScriptEngine Commands** リストには、ツールバーに追加可能な次のエントリーが含まれます。

- Start Process1
- Start Process2

ツールバーのアイコンをクリックして、それぞれのスクリプトファイルを実行します。

B.2
Schneider Electric Script Engine の例

このセクションについて

このセクションには次の項目が含まれています。

新規プロジェクト

概要

この章では、Schneider Electric Script Engine で新規プロジェクトを作成する方法を説明します。

4 種類の異なるプロジェクトタイプがありますが、プロジェクトを作成するコードは同じです。この 4 種類の場合、グローバルオブジェクト new_project の create_project メソッドが呼び出します。

これにより、次の要素で構成される 2 つのデータコンテナがインスタンス化されます。

- 共通プロジェクト設定
- コントローラー設定

共通プロジェクト設定

共通プロジェクト設定は次の要素で構成されます。

- 機械名
- 作成者
- 顧客
- 詳細
- 画像 (画像へのパス)
- プロジェクト名 (ファイル拡張子 *.project* を含むプロジェクトの名前)
- プロジェクトタイプ

プロジェクトタイププロパティによって、4 種類のプロジェクトタイプから 1 つを選択できます。

プロパティ値	詳細
StandardProject	標準プロジェクト
EDESIGN_Project	EDESIGN プロジェクト
Library	ライブラリー
EmptyProject	空のプロジェクト

コントローラー設定

コントローラー設定は次の要素で構成されます。

要素	詳細
type	コントローラーのタイプ。コントローラーのデバイス識別から取得できます。
id	コントローラーの ID。コントローラーのデバイス識別から取得できます。
version	ファームウェアバージョン
device_name	割り当て可能なデバイスの名前
implementation_language	6 つの記述言語が使用可能

標準プロジェクトの作成

標準プロジェクトを作成するためには、create_project メソッドを呼び出します。これは、データコンテナー CommonProjectSettings および ControllerSettings で構成されています。

```
# Clean up any open project
if projects.primary:
    projects.primary.close()

# Create a new instance of the common project settings object
common_settings_DTO = new_project.create_common_project_settings()
common_settings_DTO.machine_name = "Machine Name"
common_settings_DTO.author = "Author"
common_settings_DTO.customer = "Customer"
common_settings_DTO.description = "Description"
common_settings_DTO.picture = None
common_settings_DTO.project_name = "Example.project"
common_settings_DTO.project_path = r"D:\PythonProjects"
common_settings_DTO.project_type = ProjectType.StandardProject
```

```
# Create a new instance of the controller settings object
controller_settings_DTO = new_project.create_controller_settings()
controller_settings_DTO.type = 4096
controller_settings_DTO.id = "1003 0082"
controller_settings_DTO.version = "1.53.5.4"
controller_settings_DTO.device_name = "LMC_PacDrive"
controller_settings_DTO.implementation_language = ImplementationLanguage.structured_text

# Create new standard project
new_project.create_project(common_settings_DTO, controller_settings_DTO)

# Save the project
projects.primary.save()
```

デバイスパラメーター

概要

パラメーターを変更するには、パラメーター ID および ParameterSet が必要です。

必要なデバイスを探し各パラメーターをリスト表示するには、プロジェクトの指定した名前およびパスでオブジェクトを探す検索メソッドを使用します。

Script Engine の例

```
from __future__ import print_function

# Example-Device Lexium62
moduleName = 'DRV_Lexium62'

# find module/device in Project (you also can do this generic e.g. with get_children()...)
module = projects.primary.find(moduleName, True)[0]

# get all possible connectors for this device
connectors = module.connectors
print("Number of Connectors: ", connectors.Count)
print(" - ")

cnt = 1
for c in connectors:
    print("Connector " +str(cnt) +" for "+moduleName +": ")
    params = c.host_parameters
    count = params.Count
    print("Number of available Parameters for this connector: ", count)

    if count > 0:
       for p in params:
            print(p.visible_name + " - ID: %s " % p.id + " - Value: %s" % p.value)

       # example to get the value of a parameter by id:
       print(" - ")
       print(" - Example for a single Parameter (ObjectType) - ")
       objectType = params.by_id(268435457)
       print("ObjectType: ", objectType.value)

    cnt = cnt+1

    print("------------------------------------------------------------------")
```

コンパイラーバージョン

概要

コンパイラーバージョン拡張機能を使用して、マップされたコンパイラーバージョンを表示し、スクリプトを実行して新規コンパイラーバージョンを設定できます。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

# The path to the project
project_path = r"D:\PythonProjects\GetCompilerVersion.project"

# Clean up any open Project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path);

print("All compiler versions")
# Get all compiler versions (filtered)
compiler_versions = compiler_settings.get_all_compiler_versions()

# Print all compiler versions (filtered)
for version in compiler_versions:
    print (" - OEM mapped version: " + version)

# Get active compiler version
compiler_version = compiler_settings.active_compiler_version
print("Current compiler version:" + compiler_version)

# Set new compiler version
compiler_settings.active_compiler_version = "4.2.0.0"
print("New compiler version: " +
compiler_settings.active_compiler_version)

# Save project
projects.primary.save()
```

ビジュアライゼーションプロファイル

概要

ビジュアライゼーションプロファイル拡張機能を使用して、ビジュアライゼーションプロファイルの表示およびプロジェクトの有効なビジュアライゼーションプロファイルの設定ができます。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

#The path to the project
project_path = r"D:\PythonProjects\GetVisualizationProfile.project"

# Clean up any open Project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path);

# Set the project as primary project
proj = projects.primary

# Print the active profile
print("Current visual profile: " +
visualization_settings.active_profile_name)

# Get all available visualization profiles
profile_names = visualization_settings.get_all_visual_profile_names()

# Print the profiles
for visual_profile in profile_names:
    print (" - " + visual_profile)

# Set the profile to "SoMachine Motion V4.2"
visualization_settings.active_profile_name = "SoMachine Motion V4.2"

# Get the active profile
profile_name = visualization_settings.active_profile_name

# Print the active profile
print("New visual profile: " + profile_name)

# Save project
projects.primary.save()
```

プロジェクトの更新

概要

更新プロジェクト拡張機能を使うとプロジェクトを更新できます。これは、EcoStruxure Machine Expert のグラフィカルユーザーインターフェイスでプロジェクトを開き **OK** をクリックしたときに表示される **Project Update** ダイアログボックスで提供される機能と同じものです。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

# The path to the project
project_path = r"D:\PythonProjects\Example.project"

# Clean up any open project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path)
# Set the project as primary project
proj = projects.primary

# Update the project (devices, libraries, compiler version, visualization style)
proj.update_project()
```

ライブラリーの更新

概要

ライブラリーの更新拡張機能を使用して、プロジェクトのライブラリーを自動的に更新できます。これは、EcoStruxure Machine Expert グラフィックユーザーインターフェイスの**ライブラリー → 自動バージョンマップ (すべてのライブラリー)** コマンドの実行によるファンクションと同じです。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

# The path to the project
project_path = r"D:\PythonProjects\Example.project"

# Clean up any open project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path);

# Search for die library manager objects in the project
lib_managers = [i for i in proj.get_children(True) if i.is_libman]

# Make the auto mapping for each library manager found
for lib_manager in lib_managers:
    lib_manager.make_auto_mapping()
```

アプリケーションのクリーンとビルド

概要

アプリケーションのクリーンおよびビルド拡張機能を使用して、プロジェクトのクリーンまたは新規プロジェクトのビルドができます。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

# The path to the project
project_path = r"D:\PythonProjects\Example.project"

# Clean up any open project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path);

# Fetch the active application.
app = proj.active_application

# Clean application
new_project.clean_application(app)

# Compile application and store compiler messages in a list
messages = new_project.compile_application(app)

# If messages == None the build was successful
if len(messages) == 0:
    print("--- Build successful ---")

# Otherwise print results
else:
    for i in messages:
        print(i.Severity, i.Text)
```

通信設定

概要

この例では、スクリプトを実行することによって選択したコントローラーの IP アドレスの読み込み、または設定する方法を示します。

Script Engine の例

```
from __future__ import print_function

def main():
    if not projects.primary:
        system.ui.error("No active project.")
        return
    project = projects.primary

    #find the controller by the object name where address should be set and read.
    controller = project.find('LMC_PacDrive', True)[0]

    #reboot the controller
    controller.set_communication_address('192.168.2.25')

    #read address back and show it.
    print('get_communication_address := ' + controller.get_communication_address())


main()
```

ETEST の起動

概要

ソフトウェアプログラミングでは、個々のメソッドまたはプログラムのアルゴリズムの機能の正確さを確認するためにモジュール ETEST を使用します。ETEST は、テストケースを設計し、予想される結果を決めるときに検証するソースコードが分かっていることを意味するホワイトボックステストです。

処理は典型的に 3 つの手順で構成されます。

- 初期状態の初期化
- テストする処理の実行
- 予想される結果と実際の結果の比較

ETEST スクリプトドライバーは次のタスクを実行します

- テスト要素の表示
- 特定のテストの検索
- 特定のテストの実行

次の例を参照してください。

Script Engine の例

```
# Enable the new python 3 print syntax
from __future__ import print_function

# The path to the project
project_path = r"D:\PythonProjects\Example.project"

# Clean up any open project:
if projects.primary:
    projects.primary.close()

# Load the project
proj = projects.open(project_path)

# Fetch the active application.
app = proj.active_application

# Create the online application for it.
onlineapp = online.create_online_application(app)

# Log in to the device.
onlineapp.login(OnlineChangeOption.Try, True)

# Start the application, if necessary.
if not onlineapp.application_state == ApplicationState.run:
    onlineapp.start()

# Let the app do its work for some time...
system.delay(1000)

# This function runs the specified test/test series
def func_run_test(test_name):
    print(test_name)
    etest_test_provider.run_test(test_name)
    while etest_test_provider.is_test_running:
        system.delay(1000)
    return

# Get all test series from the project
test_series = etest_test_provider.get_all_testseries()

# Run all test series
for test_object in test_series:
    func_run_test(test_object)

# Get all test elements from test series "TS_MySeries"
```

```
test_elements = etest_test_provider.get_all_testelements("TS_MySeries")

if test_elements == None:
    print ("No test series 'TS_MySeries' found")
else:
    # Print all testelements
    for test_element in test_elements:
        print ("Found test element in Series TS_MySeries: " + test_element.Name)

        # Run only "TS_Crank" test object
        func_run_test(test_element.Name)

onlineapp.logout()

projects.primary.close()
```

診断メッセージのリセット

概要

アプリケーションにログインすると、コントローラーの診断メッセージをリセットできます。これは、コントローラーオブジェクトの拡張メソッドです。

次の例では、診断メッセージのリセット方法を説明します。別の例 (アプリケーションのビルド (*811* ページ)) で示されているように、主要オブジェクトを取得し、アプリケーションにログインしてください。

Script Engine の例

```
# get the project instance and log in to the application

# find the controller which messages shall be reset
controller = project.find("LMC", True)[0]

# "reset diagnosis messages on controller"
controller.reset_diagnosis_messages()
```

コントローラーの再起動

概要

スクリプトにコントローラーオブジェクトのインスタンスがあれば、そのオブジェクトでメソッドを使用することでコントローラーを再起動できます。

Script Engine の例

```
from __future__ import print_function

def perform_application_login(project):
    app = project.active_application
    onlineapp = online.create_online_application(app)
    onlineapp.login(OnlineChangeOption.Try, True)

def main():
    if not projects.primary:
        system.ui.error("No active project.")
        return

    perform_application_login(projects.primary)

    #find the controller named 'LMC_PacDrive' which shall be rebooted
    controller = projects.primary.find('LMC_PacDrive', True)[0]

    #reboot the controller
    controller.reboot_plc()

    system.ui.info("Download or OnlineChange complete")


main()
```

デバイスの変換

概要

プロジェクト内でのデバイス変換は、手順が複雑になる場合があります。この API によって変換処理が簡素化し、エラーを回避できます。

DeviceID オブジェクトの使用

変換処理 API では、指定のバージョンでデバイスまたはデバイスモジュールを識別する DeviceID オブジェクトを使用します。次のように DeviceID が作成されます。

< デバイスタイプ > < デバイスモデル > < デバイスバージョン > < モジュール名 >

要素	例	詳細
デバイスタイプ	4096	コントローラーを識別します。
デバイスモデル	1003 0082 または 1003 009D	それぞれ LMCx00C または LMCx01C
デバイスバージョン	1.50.0.4	コントローラーのファームウェアバージョン
モジュール名	LXM52	表示器モジュール

変換処理 API は、DeviceID をオブジェクトインスタンスまたは単一のパラメーターとして受け入れます。これにより、上記の表に記載されているすべての要素を含む、または各要素を単一のパラメーターとして渡す DeviceID を使用できます。

次の例は、ファームウェアバージョン 1.50.0.4 の LMCx00C コントローラー用に DeviceID を作成する方法を示しています。

```
Lmcx00c = DeviceID(4096, "1003 0082", "1.50.0.4")
```

デバイスが変換可能であるかのテスト

次のスクリプトを使用して、デバイスを変換する前に指定したバージョンへの変換が可能であるかを確認できます。

```
from __future__ import print_function

def main():
    # Set the project as primary project
    proj = projects.primary
    controller = proj.find('LMC_PacDrive', True)[0]
    drives = controller.find('DRV_Lexium62', True)

    if len(drives) == 0:
        print("Expected drive object not found")
        return

    drive = drives[0]

    # test if controller can be converted using DeviceID
    x01c = DeviceID(4096, "1003 009D", "1.53.5.4")
    if controller.can_convert(x01c):
        print("Conversion to LMCx01C possible")

    # test if drive can be converted using Parameters and module id
    if drive.can_convert(4096, "1003 0082", "1.53.5.4", "LXM52"):
        print("Conversion to LXM52 possible")


if not projects.primary:
    print("No project open.")
else:
    main()
```

代替変換用デバイスの取得

API は、特定のデバイス用に変換可能な対象デバイスを取得する呼び出しができます。各対象デバイスの DeviceID を返します。

```
from __future__ import print_function

#help function to print the delivered device ids
def deviceid_to_string(devId):
    mystr = "ID: {0.id} Type: {0.type} Version: {0.version}".format(devId)

    if hasattr(devId, 'module_id') and devId.module_id is not None:
        mystr += " ModuleID: {0.module_id}".format(devId)
        return mystr


def main():

    # Set the project as primary project
    proj = projects.primary

    controller = proj.find('LMC_PacDrive', True)[0]
    alternativeControllers = controller.get_alternative_devices()

    print("ALTERNATIVE DEVICES FOR LMC")
    for id in alternativeControllers:
        print(deviceid_to_string(id))

    drive = proj.find('DRV_Lexium62', True)[0]
    alternativeDrives = drive.get_alternative_devices()

    print("ALTERNATIVE DEVICES FOR DRIVE")
    for id in alternativeDrives:
        print(deviceid_to_string(id))

    print("Test complete. Please check the script output window")



if not projects.primary:
    print("No project open.")
else:
    main()
```

デバイスの変換

このデバイス変換の手順は、変換メソッドの呼び出しのみのため明快です。

```
from __future__ import print_function

def main():
    proj = projects.primary
    controller = proj.find('LMC_PacDrive', True)[0]
    drive = proj.find('DRV_Lexium62', True)[0]

    # converting the controller
    controller.convert(4096, "1003 009D", "1.53.5.4")
    # converting the drive
    drive.convert(DeviceID(4096, "1003 0082", "1.53.5.4"), "LXM52")


if not projects.primary:
    print("No project open.")
else:
    main()
```

プロジェクトの比較

概要

場合によっては、2 つのプロジェクトの内容を自動的に比較するスクリプトが便利です。Python プロジェクト比較ファンクションにより、2 つのプロジェクトを比較できます。その結果、プロジェクトが異なる場合はその情報、およびプロジェクトツリーを反映し各オブジェクトの違いを示す詳細な XML ツリーが提供されます。

Script Engine の例

```
from __future__ import print_function

def main():
    proj = projects.primary

    # compare the Primary Project to another Project on disk
    diff = proj.compare_to(r"d:\PythonProjects\CompTest_Right.project")
    write_diff(diff, "Diff1.xml")

    # compare, but ignore whitespaces, comments and properties
    diff = proj.compare_to(r"d:\PythonProjects\CompTest_Right.project", True, True, True)
    write_diff(diff, "Diff2.xml")


def write_diff(differences, filename):
    if differences.difference_found:
        f = open(filename, 'wb')
        f.writelines(differences.result_tree)


if not projects.primary:
    print("No project open.")
else:
    main()
```

アドバンストライブラリー管理ファンクション

概要

EcoStruxure Machine Expert Logic Builder では、前方互換ライブラリー *(EcoStruxure Machine Expert, ファンクションおよびライブラリーユーザーガイド 参照)* と呼ばれるライブラリー管理用アドバンストファンクションが使用できます。ライブラリー間の参照と依存関係を管理するために便利です。

この機能はスクリプトからも使用でき、プロジェクト全体の**ライブラリーマネージャー**またはプロジェクトのアプリケーション単体で使用できます。次のスクリプトは、ライブラリーの前方互換性および有効なリファレンスを確認する方法を示しています。リファレンスを自動的にマップし、ライブラリーのバージョンを明示的に設定します。

Script Engine の例

```
proj = projects.primary
app = proj.active_application

libmgr = app.get_library_manager()
print("# Checking all libraries:")
for lib in libmgr.get_libraries():
    print("- " + lib + " Is Forward Compatible Library? " + str(libmgr.is_library_forward_compatible(lib)))

if not libmgr.is_current_mapping_valid():
    for lib in libmgr.get_invalid_library_mappings():
        print("Library reference cannot be satisfied for: " + lib)
    print("Trying to auto-map libraries to valid versions")
    libmgr.make_auto_mapping()
else:
    print("All mappings valid")

# set version using individual parameters
libmgr.set_new_library_version("PD_GlobalDiagnostics", "Schneider Electric", "1.0.1.0")

# set version using the library full name
libmgr.set_new_library_version("PD_AxisModule, 1.1.6.0 (Schneider Electric)", "1.2.4.0")

# set version to Legacy
libmgr.set_new_library_version("PD_Template", "Schneider Electric", None)
```

POU へのアクセス

概要

次の例では、POU のコードの印刷および操作方法を示しています。これらは、テキストプログラミング言語でのみ使用できます。

Script Engine の例

```
if not projects.primary:
    print('No primary project set')

proj = projects.primary
pou = proj.find('SR_Main', True)[0]

# read and print the declaration of the program
decl = pou.textual_declaration
print(decl.text)

# read and print the implementation of the program
code = pou.textual_implementation
print(code.text)

decl_text = "PROGRAM SR_Main\n" + \
            "VAR\n" + \
            "  iTest: INT;\n" + \
            "END_VAR";

code_text = "iTest := iTest + 1;"

# write new code to the declaration and implementation
decl.replace(decl_text)
code.replace(code_text)
```

廃止された set_interface_text() API 用　Script Engine の例

```
if not projects.primary:
    print("No primary project set")

proj = projects.primary
pou = proj.find('SR_Main', True)[0]

# read and print the declaration of the program
decl = pou.get_interface_text()
print(decl)

# read and print the implementation of the program
code = pou.get_implementation_text()
print(code)

decl = "PROGRAM SR_Main\n" + \
       "VAR\n" + \
       "  iTest: INT;\n" + \
       "END_VAR";
code = "iTest := iTest + 1;"

# write new code to the declaration and implementation
pou.set_interface_text(decl)
pou.set_implementation_text(code)
```

B.3
CoDeSys Script Engine の例

概要

この章では、頻繁に使用される CoDeSys Script Engine メンバーの例を説明します。各名前空間のメンバーの完全な詳細は、CoDeSys API 概要を参照してください。

このセクションについて

このセクションには次の項目が含まれています。

プロジェクト

概要

この名前空間の例は比較的短く明白なため、詳細は説明していません。例の完成形は、適当箇所で説明しています。

新規プロジェクト

このメソッドは、新規プロジェクトを作成します。

2 つのパラメーターで構成されています。

- プロジェクトを保存する場所を指定する文字列
- ブール型パラメーター : TRUE の場合、プロジェクトは新規の主要プロジェクトになります。このパラメーターはオプションです。初期値は TRUE です。

メソッドは、後のステップで使用できる `IProject` インスタンス (*オートメーションプラットフォーム SDK* ドキュメントの仕様を参照) を返します。

```
import os

try:
    # Clean up any open project
    if projects.primary:
        projects.primary.close()

    # Define the new file name for the project
    project_name = "Example.project"

    # Define the new path where the new project should be stored
    project_path = r"C:\Python"

    # Create the new project
    proj = projects.create(os.path.join(project_path, project_name), True)

    # Save the project to the specified path
    proj.save()
except Exception as exception:
    print("Error: " + str(exception))
    if not system.trace:
        print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

プロジェクトの読み込み

このメソッドは、プロジェクトを読み込みます。開いてるプロジェクトは、閉じません。

最初のパラメーターは、読み込むプロジェクトのパスを指定します。

```
import os

try:
    # Clean up any open project
    if projects.primary:
        projects.primary.close()

    # Define the file name for the project
    project_name = "Example.project"

    # Define the path where the project is stored
    project_path = r"C:\Python"

    # Load the existing project
    proj = projects.open(os.path.join(project_path, project_name))
except Exception as exception:
    print("Error: " + str(exception))
    if not system.trace:
        print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

プロジェクトの保存

このメソッドは、プロジェクトを物理的な場所に保存します。

```
# Save project
projects.primary.save()
```

アーカイブの保存

このメソッドは、プロジェクトをアーカイブとして保存します。デフォルトで選択されている追加のカテゴリーは含まれますが、追加ファイルは含みません。

最初のパラメーターは、アーカイブを保存する場所のパスを指定します。

```
import os

# Define the new file name for the archive
archive_name = "Example.archive"

# Define the new path where the archive should be stored
archive_path = r"C:\Python"

# Save archive with the default values
projects.primary.save_archive(os.path.join(archive_path, archive_name))
```

プロジェクトの終了

このメソッドは、プロジェクトを終了させます。プロジェクトに保存されていない変更がある場合、その変更は破棄されます。

```
# Clean up any open project:
if projects.primary:
   projects.primary.close()
```

オブジェクトの検索

このメソッドは、指定した名前に一致するオブジェクトを検索します。

2 つのパラメーターで構成されています。

- 最初のパラメーターは、検索するオブジェクト名です。
- 2 番目のパラメーターは、再帰検索を実行するかを指定します。このパラメーターはオプションです。デフォルト値は FALSE です。このメソッドは、オブジェクトのコレクションを返します。

```
# Search for
result_list = projects.primary.find('MyController', True)

for result in result_list:
   print("Object " + result.get_name() + " found with Guid " + str(result.guid))
```

名前はこのツリー内で固有ではありません。複数のオブジェクトを検索させる場合があります。ローカライズされていない名前に対して検索します。

ネイティブインポート

このメソッドは、指定したネイティブ XML 形式のファイルをプロジェクトの最上位にインポートします。

```
import os

# Specify the project file name
project_name = "NativeImport.project"

# Define the path where the project should be/is stored
project_path = r"C:\Python"

# Define the path where the exported objects are be stored
object_path = os.path.join(project_path, "Objects")

# Create the import reporter
class Handler(NativeImportHandler):
   def conflict(self, name, obj, guid):
      print("Object already exists: " + name)
```

```
            return NativeImportResolve.skip

        def progress(self, name, obj, exception):
            print("in progess: " + name)

        def skipped(self, list):
            return

    def import_filter(name, guid, type, path):
        # Workaround, skip the project settings object because we cant import it
        if(type == "_3S.CoDeSys.Engine.WorkspaceObject"):
            return False
        return True

    try:
        # Clean up any open project
        if projects.primary:
            projects.primary.close()

        # Create the new project
        project_reference = projects.create(os.path.join(project_path, project_name), True)

        files = os.listdir(object_path)

        # Create the importer instance.
        handler = Handler()

        for file in files:
            file_path = os.path.join(object_path, file)
            project_reference.import_native(file_path, import_filter, handler)

        project_reference.save()

    except Exception as exception:
        print("Error: " + str(exception))
        if not system.trace:
            print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
    n.")
```

PLCOpenXML インポート

このメソッドは、指定した PLCopenXML ファイルの内容をプロジェクトの最上位にインポートします。

```
import os

# Specify the project file name
project_name = "PLCOpenXMLImport.project"

# Define the path where the project should be/is storedproject_path = r"C:\Python"

# Define the file where the exported object is stored
file_name = os.path.join(project_path, r"Objects\MyController.xml")

# Create the import reporter
class Reporter(ImportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
    def resolve_conflict(self, obj):
        return ConflictResolve.Copy
    def added(self, obj):
        print("added: ", obj)
    def replaced(self, obj):
        print("replaced: ", obj)
    def skipped(self, obj):
```

```
        print("skipped: ", obj)
    @property
    def aborting(self):
        return False

try:
    # Clean up any open project
    if projects.primary:
        projects.primary.close()

    # Create the reporter instance
    reporter = Reporter()

    # Create the new project
    project_reference = projects.create(os.path.join(project_path, project_name), True)

    # Import the data into the project
    project_reference.import_xml(reporter, file_name)

    # Save the project to the specified path
    project_reference.save()
except Exception as exception:
    print("Error: " + str(exception))
    if not system.trace:
        print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

ネイティブエクスポート

このメソッドは、指定したネイティブ形式のオブジェクトを文字列、または指定したパスのファイルにエクスポートします。エクスポートできないオブジェクトはエラーとして検出されますが、エクスポートは続行します。

```
import sys,io,os

# Specify the project file name
project_name = "Example.project"

# Define the path where the project should be/is stored
project_path = r"C:\Python"

# Define the path where the exported objects should be stored
object_path = os.path.join(project_path, "Objects")

def collect_objects(project_reference):
    # List that stores all POU nodes
    project_objects = []

    # Collect all the leaf nodes.
    for node in project_reference.get_children(True):
        project_objects.append(node)

    for i in project_objects:
        print("Found: ", i.type, i.guid, i.get_name())
    return project_objects

def export_objects(collected_objects, project_reference):
    if not os.path.exists(object_path):
        os.makedirs(object_path)

    # Export the files.
    for candidate in collected_objects:
        # Create a list of objects to export:
        # The object itself
        objects = [candidate]
```

```
        # And sub-objects (POUs can have actions, properties, ...)
        objects.extend(candidate.get_children(True))

        # And the parent folders.
        parent = candidate.parent
        while ((not parent.is_root) and parent.is_folder):
            objects.append(parent)
            parent = parent.parent

        # Create a unique file name
        filename = os.path.join(object_path, "%s__%s.export" % (candidate.get_name(), candidate.guid))

        # Print some user information
        print("Exporting " + str(len(objects)) + " objects to: " + filename)

        # And actually export the project
        project_reference.export_native(objects, filename)

try:
    # Clean up any open project
    if projects.primary:
        projects.primary.close()

    # Open a project first
    project_reference = projects.open(os.path.join(project_path, project_name))

    # Collect the objects
    collected_objects = collect_objects(project_reference)

    export_objects(collected_objects, project_reference)
except Exception as exception:
    print("Error: " + str(exception))
    if not system.trace:
        print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

PLCOpenXML エクスポート

このメソッドは、指定した PLCopenXML 形式のオブジェクトを文字列、または指定したパスのファイルにエクスポートします。エクスポートできないオブジェクトはエラーとして検出されますが、エクスポートは続行します。

```
import os

# Specify the object which should be exported
object_name = "MyController"

# Define the path where the exported objects should be stored
object_path = r"C:\Python\Objects"

# Define the printing function
def print_tree(treeobj, depth=0):
    name = treeobj.get_name(False)
    if treeobj.is_device:
        deviceid = treeobj.get_device_identification()
        print("{0} - {1} {2}".format(" "*depth, name, deviceid))

    for child in treeobj.get_children(False):
        print_tree(child, depth+1)

# Create the export reporter
class Reporter(ExportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
```

```
    def nonexportable(self, message):
        print(message)
    @property
    def aborting(self):
        return False

try:
    # Get the project reference of the currently opened project
    project_reference = projects.primary

    # Get a reporter instance
    reporter = Reporter()

    # Print all devices in the project
    for obj in project_reference.get_children():
        print_tree(obj)

    # Finds the object in the project, and return the first result
    device = project_reference.find(object_name, True)

    if device != None:
        filename = os.path.join(object_path, device[0].get_name() + ".xml")

        # Exports the object to the hard drive
        project_reference.export_xml(reporter, device, filename, True, True)

except Exception as exception:
    print("Error: " + str(exception))
    if not system.trace:
       print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

オンラインアプリケーション

概要

注記：一部のオンラインアプリケーションコマンドは、有効なアプリケーションを一時的に変更します。

このインターフェイスは Python にエクスポートされるため、Python の命名規格に準拠しています。

オンラインアプリケーションの作成

このメソッドは、オンラインアプリケーションを作成します。

```
# Create online application
online_application = online.create_online_application()
```

設定済みの値

この例では、設定済みの値の記述方法を説明します。

```
try:
   # Trying to create online application
   online_application = online.create_online_application()

   print("app:" + str(online_application.application_state) + "op:" + online_application.operation_state.To
String("f"))

   for expression in online_application.get_prepared_expressions():
      online_application.set_prepared_value(expression, '')
      print "%s: '%s' '%s'" % (expression, online_application.read_value(expression), online_application.g
et_prepared_value(expression))

   for expression in online_application.get_forced_expressions():
      online_application.set_unforce_value(expression)
      print "%s: '%s' '%s'" % (expression, online_application.read_value(expression), online_application.g
et_prepared_value(expression))

   assert len(online_application.get_prepared_expressions()) == 0, "Still some prepared values remain."
   assert len(online_application.get_forced_expressions()) == 0, "Still some prepared values remain."

   # Preparing a value and forcing it
   online_application.set_prepared_value("POU.testoutput", "123")
   online_application.force_prepared_values()

   # Preparing a value and writing it    online_application.set_prepared_value("POU.testint", "INT#1147")
   online_application.write_prepared_values()

except Exception as exception:
   print("Error: " + str(exception))
   if not system.trace:
      print("Please turn on the 'Script Tracing' function to get detailed information about the script executio
n.")
```

アプリケーションログインの実行

このメソッドは、アプリケーションのログインを実行します。アプリケーションが前にログインされている場合は、ログアウトしてから再度新しくログインします。

2 つのパラメーターで構成されています。

- 最初のパラメーターは変更オプションです。
- 2 番目のパラメーターが True に設定されている場合、以前のアプリケーションが削除されます。

```
# Get the project reference
project_reference = projects.primary

# Fetch the active application
active_application = project_reference.active_application

# Create the online application for it
online_application = online.create_online_application(active_application)

# Log in to the device.
online_application.login(OnlineChangeOption.Try, True)
```

ログアウトアプリケーション

このメソッドは、アプリケーションからログアウトします。アプリケーションにログインしていない場合は、何も起こりません。

```
# Get the project reference
project_reference = projects.primary

# Fetch the active application
active_application = project_reference.active_application

# Create the online application for it
online_application = online.create_online_application(active_application)

# Log in to the device.
online_application.logout()
```

アプリケーションの起動

このメソッドは、アプリケーションを起動します。

```
# Get the project reference
project_reference = projects.primary

# Fetch the active application
active_application = project_reference.active_application

# Create the online application for it
online_application = online.create_online_application(active_application)

# Log in to the device.
online_application.login(OnlineChangeOption.Try, True)

# Start the application, if necessary
if not online_application.application_state == ApplicationState.run:
    online_application.start()

    # Let the app do its work for some time
    system.delay(1000)
```

アプリケーションの停止

このメソッドは、アプリケーションを停止します。

```
# Get the project reference
project_reference = projects.primary

# Fetch the active application
active_application = project_reference.active_application

# Create the online application for it
online_application = online.create_online_application(active_application)

# Log in to the device.
online_application.login(OnlineChangeOption.Try, True)

# Stop the application, if necessary
if online_application.application_state == ApplicationState.run:
    online_application.stop()
```

オブジェクト

検索

このメソッドは、指定した名前に一致するオブジェクトを検索します。

2 つのパラメーターで構成されています。

- 最初のパラメーターは、検索するオブジェクト名です。
- 2 番目のパラメーターは、再帰検索を実行するかを指定します。このパラメーターはオプションです。デフォルト値は FALSE です。このメソッドは、オブジェクトのコレクションを返します。

```
# Search for the specified object
results = projects.primary.find('MyController', True)

for result in results:
    print("Object " + result.get_name() + " found with Guid " + str(result.guid))
```

削除

このメソッドは、オブジェクトを削除します。

```
# Finds the MyController object in the project
results = projects.primary.find("MyController", True)

for result in results:
    # Removes all objects with the name MyController
    result.remove()
```

名前の変更

このメソッドは、オブジェクトの名前を変更します。

```
# Finds the MyController object in the project
results = projects.primary.find("MyController", True)

for result in results:
    # Rename all MyController objects to MyController_2
    result.rename("MyController_2")
```

インポート / エクスポート

プロジェクトのインポート / エクスポート (*825* ページ) の概要を参照してください。違いは、インポート / エクスポートにプロジェクトではなくオブジェクトが呼び出されることのみです。

デバイス

概要

この章では、デバイスオブジェクトを操作するメソッドについて説明します。

追加

このメソッドは、指定したデバイスを追加します。

3 つのパラメーターで構成されています。

- デバイスの名前を指定する文字列
- デバイスの ID を指定する DeviceID DeviceID の値については、追加したいデバイスのデバイス説明を参照してください。
- モジュール ID を指定する文字列

```
# We want to add a TM5C12D6T6L Module to the TM5_Manager
# Search for the TM5_Manager. The new Module will be inserted below this object.
parent_device = projects.primary.find('TM5_Manager', True)[0]

if not(parent_device == None):
    parent_device.add("Module_TM5C12D6T6L", DeviceID(51063, "101a 0363", "3.1.2.2"))
```

無効

このメソッドによって、ダウンロード中にデバイスが無効であるという印が付けられます。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    device.disable()
```

有効

このメソッドによって、ダウンロード中にデバイスが有効であるという印が付けられます。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    device.enable()
```

アドレスの取得

このメソッドは、デバイスのアドレスを取得します。文字列を返します。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    print("Address: " + device.get_address())
```

デバイス識別の取得

このメソッドは、デバイス識別を取得します。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    print(str(device.get_device_identification()))
```

ゲートウェイの取得

このメソッドは、ゲートウェイの GUID を返します。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    print("Gateway: " + str(device.get_gateway()))
```

挿入

このメソッドは、指定したインデックスに指定したデバイスを挿入します。

4 つのパラメーターで構成されています。

- デバイスの名前を指定する文字列
- デバイスを挿入するインデックスを指定する Int32
- デバイスの ID を指定する DeviceID DeviceID の値については、追加したいデバイスのデバイス説明を参照してください。
- モジュール ID を指定する文字列

```
# Finds the MyController object in the project
parent_device = projects.primary.find("TM5_Manager", True)[0]

if not(parent_device == None):
    # We need the amount of children to add the new device at the end
    child_count = len(parent_device.get_children())

    # Use a unique name for the inserted device
    parent_device.insert("Module_TM5C12D6T6L_" + str(child_count), child_count, DeviceID(51063, "101
a 0363", "3.1.2.2"))
```

ゲートウェイおよびアドレスの設定

このメソッドは、ゲートウェイとアドレスを設定します。空の GUID および空のアドレスを渡した場合、ゲートウェイアドレスはクリアされます。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    device.set_gateway_and_address(gateway, address)
```

シミュレーションモードの設定

このメソッドは、シミュレーションモードを設定します。True に設定すると、シミュレーションが有効になります。

```
# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    device.set_simulation_mode (True)
```

更新

このメソッドは、指定したデバイスを更新します。

```
# This is a more generic approach to find and update a device by its device description
import os

def find_device_repository(head_dir, sub_dir):
    for root, dirs, files in os.walk(head_dir):
        for d in dirs:
            current_dir = os.path.join(root, d)
            if current_dir.endswith(sub_dir):
                return current_dir

# This function is used to find the newest version
def version_tuple(v):
```

```
    return tuple(map(int, (v.split("."))))

def get_newest_version(directory, current_version):
    newest_version = None
    available_versions = os.listdir(directory)
    for version in available_versions:
        if version_tuple(version ) > version_tuple(current_version):
            newest_version = version

    return newest_version

# Finds the MyController object in the project
device = projects.primary.find("MyController", True)[0]

if not(device == None):
    device_identification = device.get_device_identification()

    # Device Repository subdirectory
    sub_dir = os.path.join(str(device_identification.type), device_identification.id)

    # Find the path where the device description is stored
    result_directory= find_device_repository(r"c:/", sub_dir)

    if not result_directory == None:
        newest_version = get_newest_version(result_directory, device_identification.version)

        if not newest_version == None:
            device.update(device_identification.type, device_identification.id, newest_version)
```

システム / ユーザーインターフェイス

ディレクトリーの参照用ダイアロボックス

ディレクトリーを参照するためのダイアログボックスを開きます。--noUI モードでは、ここにパスを入力するのみです。

```
# Import the .NET class System.Environment from mscorlib.dll first
# Otherwise the Environment.SpecialFolder Enumeration is not available
from System import Environment
import os

# Set the necessary fields
dialog_message = "Select Python Example Directory"
preselected_path =  r"c:\Python"

# Store the selected path
selected_path = system.ui.browse_directory_dialog(dialog_message, preselected_path, Environment.SpecialFolder.Desktop, True)

# Remember to check if the user canceled the dialog
if(selected_path != None):
    # List files in the specified directory
    print(os.listdir(selected_path))
```

4 つのパラメーターで構成されています。

- メッセージを含む文字列
- ダイアログボックスが開いたときに、あらかじめ選択されているパスを含む文字列
- Environment.SpecialFolder: 参照ダイアログボックスのルートフォルダーを含む
- ブール型パラメーター : True の場合、ダイアログボックスに新しいフォルダーを作成するためのボタンが表示されます。

このメソッドは、選択したパスを返します。ダイアログボックスをキャンセルすると、何も返しません。

選択

このメソッドによって、リストにある複数の項目の 1 つを選択できます。

```
# Set the necessary fields
option_list = ["TM221C16R", "TM241C24R", "TM258LF66DT4L"]

dialog_message = "Select the controller which should be added to the project:"

# A python tuple will be returned with the index of the selected item or -1 if canceled
selected_item = system.ui.choose(dialog_message, option_list, True)

# Remember to check if the user canceled the dialog
if(selected_item[0] != -1):
    print(option_list[selected_item[0]] + " will be added to the project.")
```

3 つのパラメーターで構成されています。

- メッセージを含む文字列
- 表示されるオプションのリスト : オブジェクトは、文字列に変換されて表示されます。
- ブール型パラメーター : True の場合、ダイアログボックスに新しいフォルダーを作成するためのボタンが表示されます。

このメソッドは、2 つの項目を含む Python タプルを返します。

- 選択した項目のインデックス、または
  キャンセル可能が True に設定され、ダイアログボックスをキャンセルした場合は -1。
- 選択した項目または None

エラー検出

このメソッドは、エラー検出メッセージを示します。メッセージが確認されるまで、これ以上のアクションは禁止されます。

```
system.ui.error("Error")
```

情報

このメソッドは、情報メッセージを示します。メッセージが確認されるまで、これ以上のアクションは禁止されます。

```
# Set the necessary fields
dialog_message = "Project update has been completed successfully"

# This method is used to display a simple message to the user
system.ui.info(dialog_message)
```

ファイルを開くダイアログボックス

このメソッドは、**ファイルを開く**ダイアログボックスを表示します。--noUI モードでは、ここにパスを入力するのみです。

```
# Set the necessary fields
dialog_title = "Select a Project"
dialog_filename = None
initial_directory = None
file_filter = "(*.project)|*.project"

# Open a select file dialog which only accepts *.project files
selected_file = system.ui.open_file_dialog(dialog_title, dialog_filename, initial_directory, file_filter)

# Remember to check if the user canceled the dialog
if(selected_file != None):
    # Open the specified project
    if projects.primary:
        projects.primary.close()
    projects.open(selected_file)
```

クエリー文字列

このメソッドは、テキスト文字列の入力または編集を問い合わせます。

```
# Set the necessary fields
dialog_message = "Please enter a filename for your project:"
dialog_prefilled_text = "c:\Python\QueryStringExample.project"

# This functions queries a string from the user
project_name = system.ui.query_string(dialog_message, dialog_prefilled_text)

# If a project is opened, the project will be saved under the new nameif(projects.primary):
    projects.primary.save_as(project_name)
```

入力されたテキストを含む文字列を返します。

ファイルの保存ダイアログ

このメソッドは、**ファイルの保存**ダイアログボックスを表示します。--noUI モードでは、ここにパスを入力するのみです。

```
import os

# Set the necessary fields
dialog_title = "Select a Filename"
initial_directory, dialog_filename = os.path.split(projects.primary.path)
file_filter = "(*.project)|*.project"

# Open a select file dialog which only accepts *.project files
selected_file = system.ui.save_file_dialog(dialog_title, dialog_filename, initial_directory, file_filter)

# Remember to check if the user canceled the dialog
if(selected_file != None):
    # File was saved under the specified name, lets close the project
    projects.primary.close()
```

警告

このメソッドは、警告メッセージを示します。メッセージが確認されるまで、これ以上のアクションは禁止されます。

```
# Set the necessary fields
dialog_message = "Project update failed. Please check the log for detailed errors."

# This method is used to display a simple warning to the user
system.ui.warning(dialog_message)
```

値の読み込み

概要

スクリプトは EcoStruxure Machine Expert のアプリケーションを開き、デバイスにログインします。コントローラーが RUNNING 状態ではない場合、RUNNING に設定されます。その後、変数 iVar1 が読み込まれ、**メッセージ**ビューまたはコマンドラインに表示されます。最後にアプリケーションが閉じます。

スクリプト例 *ReadVariable.py*

```
import os

# Define the necessary fields for this example
project_name = "Ohne Titel.project"
project_path = r"C:\Python"
value_name = "POU.testoutput"

try:
    # Clean up any open project
    if projects.primary:
        projects.primary.close()

    # Create the new project
    proj = projects.open(os.path.join(project_path, project_name))

    # Set the active application
    app = proj.active_application

    # Create the online application
    onlineapp = online.create_online_application(app)

    # Login to device
    onlineapp.login(OnlineChangeOption.Try, True)

    # Set status of application to "run" if not in "run"
    if not onlineapp.application_state == ApplicationState.run:
        onlineapp.start()

    # Wait 1 second
    system.delay(1000)

    # Read value of iVar1
    value = onlineapp.read_value(value_name)

    # Display value in message view or command line
    print (value_name + ": " + str(value))

    # Log out from device and close the project
    onlineapp.logout()
    #proj.close()
except Exception as exception:
    print("Error: " + str(exception))
    print("Please turn on the 'Script Tracing' function to get detailed information about the script execution.
")
```

レシピから値を読み込み、メールで送信

概要

スクリプトは EcoStruxure Machine Expert のアプリケーションを開き、デバイスにログインします。コントローラーが RUNNING モードではない場合、RUNNING に設定されます。その後、変数 iVar1 が読み込まれ、**メッセージ**ビューまたはコマンドラインに表示されます。最後にアプリケーションが閉じます。

スクリプト例 *ScriptEmail.py*

```
# Get the primary project reference
project = projects.primary
 # Retrieve active application
active_application = project.active_application

# Create online application
online_application = online.create_online_application(active_application)

# Login to application
online_application.login(OnlineChangeOption.Try, True)

# Start PLC if necessary
if not online_application.application_state == ApplicationState.run:
    online_application.start()

# Wait 2 seconds
system.delay(2000)

# Define read values
watch_expressions = ["POU.value_1", "POU.value_2", "POU.value_3"]

# Read values from the controller
watch_values = online_application.read_values(watch_expressions)

# Open output file to write values
recipe_output_file = open(r"C:\Python\Objects\Output.txt", "w")

for i in range(len(watch_expressions)):
    recipe_output_file.write(watch_expressions[i])
recipe_output_file.write(" = ")
recipe_output_file.write(watch_values[i])
recipe_output_file.write("\n")

# Close files
recipe_output_file.close()

# Send Email
# Import respective libraries
import smtplib
from email.mime.text import MIMEText
#open output file
recipe_output_file = open("C:\Python\Objects\Output.txt", "r")
mail = MIMEText(recipe_output_file.read())
recipe_output_file.close()

# Email address sender and recipient
sender = "sender@sender.com"
to = "to@to.com"

# Set sender and recipient
mail["Subject"] = "Attention value has changed"
mail["From"] = sender
mail["To"] = to
```

```
# Send email
smtp = smtplib.SMTP("Name of the SMTP Server")
smtp.sendmail(sender, [to], mail.as_string())
smtp.quit()

# Logout and close application
online_application.logout()
project.close()
```

開いているプロジェクトのデバイスツリーの決定

概要

この例では、開いているプロジェクトの**デバイスツリー**のオブジェクトを決定し、コマンドラインまたは**メッセージ**ビューに表示します。EcoStruxure Machine Expert ユーザーインターフェイスまたはコマンドラインから実行できます。

スクリプト例 *DevicePrintTree.py*

```
# We enable the new python 3 print syntax
import sys

# Define the printing function
def printtree(treeobj, depth=0):
    if treeobj.is_root:
        name = treeobj.path
        deviceid = ""
    else:
        name = treeobj.get_name(False)
        if treeobj.is_device:
            deviceid = treeobj.get_device_identification()
        else:
            deviceid = ""
    print("{0} - {1} {2}".format(" "*depth, name, deviceid))
    for child in treeobj.get_children(False):
        printtree(child, depth+1)

# Now see whether a primary project is open.
if not projects.primary:
    print("Error: Please open a project file first!")
    sys.exit()

# And the actual output
print("Print the current tree of: ")
printtree(projects.primary)
```

スクリプト例 4: PLCOpenXML のデバイスを Subversion からインポートする

概要

この例では、コマンドライン `svn client` によりデバイスを Subversion から PLCOpenXML にインポートします。EcoStruxure Machine Expert ユーザーインターフェイスまたはコマンドラインから実行できます。

スクリプト例 *DeviceImportFromSvn.py*

```
# Imports a Device in PLCOpenXML from Subversion via command line svn client
import os

# Some variable definitions
svn_executable = r"C:\Program Files\TortoiseSVN\bin\svn.exe"
project_file = r"C:\Python\test.project"

# !!! Important: File must be under version control !!!
export_file = r"C:\Python\SVN\MyController.xml"

# Clean up any open project:
if projects.primary:
    projects.primary.close()

# Fetch the PLCOpenXML data from subversion
# The with construct automatically closes the open pipe for us.
with os.popen('"' + svn_executable + '" cat ' + export_file, "r") as pipe:
    xmldata = pipe.read()

# Create a new project
project_reference = projects.create(project_file)

# Create the import reporter
class Reporter(ImportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
    def resolve_conflict(self, obj):
        return ConflictResolve.Copy
    def added(self, obj):
        print("added: ", obj)
    def replaced(self, obj):
        print("replaced: ", obj)
    def skipped(self, obj):
        print("skipped: ", obj)
    @property
    def aborting(self):
        return False

# Create the importer instance
reporter = Reporter()

# Import the data into the project
project_reference.import_xml(reporter, xmldata)

# And finally save
project_reference.save()
```

スクリプト例 5: POU の作成と編集

概要

スクリプト *CreateDut.py* は、オブジェクト MyStruct、MyAlias、および MyUnion を EcoStruxure Machine Expert プロジェクトに作成します。フォルダー *DataTypes* が必要です。

CreateDut.py

```
# encoding:utf-8
from __future__ import print_function
STRUCT_CONTENT = """\
   a : BOOL;
   b : BIT;
   c : BIT;
"""
UNION_WHOLE = """\
TYPE MyUnion :
UNION
   Zahl : INT;
   Prozent : MyAlias;
   Bits : MyStruct;
END_UNION
END_TYPE
"""
proj = projects.primary
folder = proj.find('DataTypes', recursive = True)[0]
# Create a struct DUT and insert the list of variables just into the right
# place in line two, row 0 (line numbering starts with line 0)
struktur = folder.create_dut('MyStruct') # DutType.Structure is the default
struktur.textual_declaration.insert(2, 0, STRUCT_CONTENT)
# Alias types get their "content" via the base type, which will just end up
# as one line in the declaration part:
# TYPE MyAlias : INT (0..100); END_TYPE
bereich = folder.create_dut('MyAlias', DutType.Alias, "INT (0..100)")
# Instead of injecting the variables into the existing declaration,
# one can also just replace the complete declaration part, including the
# boilerplate code.
union = folder.create_dut('MyUnion', DutType.Union)
union.textual_declaration.replace(UNION_WHOLE)
```

スクリプト例 6: ユーザーインターフェイス / ユーザーとの対話

概要

場合によっては、スクリプトはユーザーと対話する必要があります。一般的な対話のためにシンプルな API が提供されています。サンプルスクリプト *System_UI_Test.py* にはそれらの機能が示されています。

System_UI_Test.py

```
# encoding:utf-8
from __future__ import print_function

"""Performs some tests on the messagestore and UI."""

print("Some Error, Warning and Information popups:")
system.ui.error("Fatal error: Everything is OK. :-)")
system.ui.warning("Your bank account is surprisingly low")
system.ui.info("Just for your information: 42")

print("Now, we ask the user something.")
res = system.ui.prompt("Do you like this?", PromptChoice.YesNo, PromptResult.Yes);
print("The user selected '%s'" % res)

print("Now, the user can choose between custom options:")
res = system.ui.choose("Please choose:", ("First", 2, 7.5, "Something else"))
print("The user selected option '%s'" % str(res)) # res is a tuple

print("Now, the user can choose several options:")
res = system.ui.select_many("Please select one or more options", PromptChoice.OKCancel,
PromptResult.OK, ("La Premiere", "The Second", "Das Dritte"))
print("The returned result is: '%s'" % str(res)) # res is a tuple

print("Now, the user can select files and directories")
res = system.ui.open_file_dialog("Choose multiple files:", filter="Text files (*.txt)|*.txt|Image
Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*", filter_index = 0, multiselect=True)
print("The user did choose: '%s'" % str(res)) # res is a tuple as multiselect is true.

res = system.ui.save_file_dialog("Choose a file to save:", filter="Text files (*.txt)|*.txt|Image
Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*", filter_index = 0)
print("The user did choose: '%s'" % res)

res = system.ui.browse_directory_dialog("Choose a directory", path="C:\\")
print("The user did choose: '%s'" % res)

print("Now we query a single line string")
res = system.ui.query_string("What's your name?")
print("Nice to meet you, dear %s." % res)

print("Now we query a multi line string")
res = system.ui.query_string("Please tell me a nice story about your life!", multi_line=True)
if (res):
   print("Huh, that has been a long text, at least %s characters!" % len(res))
else:
   print("Hey, don't be lazy!")

print("Username and passwort prompts...")
res = system.ui.query_password("Please enter your favourite password!", cancellable=True)
if res:
```

```
    print("Huh, it's very careless to tell me your favourite password '%s'!" % res)
else:
    print("Ok, if you don't want...")

res = system.ui.query_credentials("Now, for real...")
if res:
    print("Username '%s' and password '%s'" % res) # res is a 2-tuple
else:
    print("Sigh...")
```

スクリプト例 7: プロジェクト情報オブジェクトの操作

概要

スクリプト *ProjectInfoExample.py* は、**プロジェクト情報**オブジェクトの情報を提供します。**タイトル**や**バージョン**などの重要な情報項目には詳細なプロパティがあります。ただし、辞書の構文を使用してほかの情報フィールドを読み書きできます。例えば、ライブラリープロジェクトの特性として推奨されるもの。

以下の例は非現実的に見えるかもしれませんが、自動ライブラリープロジェクトや他のプロジェクトを作成、テスト、そしておそらく解放するビルドサーバーでも同様のコードが使用されています。ScriptEngine は、CI (Continuous Integration / 継続的インテグレーション) および CD (Continuous Delivery / 継続的デリバリー) システムを作成するための重要な要素の 1 つです。

ProjectInfoExample.py

```
# encoding:utf-8
from __future__ import print_function

proj = projects.load("D:\Some.library")

info = proj.get_project_info()

# Set some values
info.company = "Test Library Ltd"
info.title = "Script Test Project"
info.version = (0, 8, 15, 4711)
info.default_namespace = "testlibrary"
info.author = "Python von Scriptinger"

# some values recommended in the library toolchain
info.values["DefaultNamespace"] = "testlibrary"
info.values["Placeholder"] = "testlibrary"
info.values["DocFormat"] = "reStructuredText"

# now we set a custom / vendor specific value.
info.values["SpecialDeviceId"] = "PLC0815_4711"

# Enable generation of Accessor functions, so the IEC
# application can display the version in an info screen.
info.change_accessor_generation(True)

# And set the library to released
info.released = True;

proj.save()
```

詳細な例 : SVN からライブラリーをチェックアウトし EcoStruxure Machine Expert にインストールする

概要

次のサンプルスクリプトは、CT (Continuous Testing / 継続的テスト) 環境の一部としてライブラリーのチェックアウトとインストールを実行して、ライブラリーをテストできるようにします。SVN アドオンをパソコンにインストールし、SVN アドオンライセンスを有効にする必要があります。

サンプルスクリプト

```
import tempfile

if projects.primary:
    projects.primary.close()

tempdir = tempfile.mkdtemp()
URL = "svn://localhost/testrepo/trunk/SvnTestLibrary/"

proj = svn.checkout(URL, tempdir, "testlibrary", as_library=True)
proj.save()

repo = librarymanager.repositories[0]
librarymanager.install_library(proj.path, repo, True)

proj.close()
```

付録 C
移行用コントローラー機能セット

移行用コントローラー機能セット

Twido コントローラー

コントローラー	デジタル入力	デジタル出力	MOD	FC	HSC	PWM	シリアル	ETH
TWDLCAA10DRF	6	4	なし	3	1	0	1	なし
TWDLCDA10DRF	6	4	なし	3	1	0	1	なし
TWDLCAA16DRF	9	7	なし	3	1	0	1+1	なし
TWDLCDA16DRF	9	7	なし	3	1	0	1+1	なし
TWDLCAA24DRF	14	10	なし	3	1	0	1+1	なし
TWDLCDA24DRF	14	10	なし	3	1	0	1+1	なし
TWDLCAA40DRF	24	16	なし	4	2	2	1+1	なし
TWDLCDA40DRF	24	16	なし	4	2	2	1+1	なし
TWDLCAE40DRF	24	16	なし	4	2	2	1+1	あり
TWDLCDE40DRF	24	16	なし	4	2	2	1+1	あり
TWDLMDA20DTK	12	8	あり	2	2	2	1+1	なし
TWDLMDA20DUK	12	8	あり	2	2	2	1+1	なし
TWDLMDA20DRT	12	8	あり	2	2	2	1+1	なし
TWDLMDA40DTK	24	16	あり	2	2	2	1+1	なし
TWDLMDA40DUK	24	16	あり	2	2	2	1+1	なし

デジタル入力 = デジタル入力の数
デジタル出力 = デジタル出力の数
MOD = 拡張モジュール
FC = 高速カウンター (FC) の数
HSC = 高速カウンター (HSC) の数
PWM = パルス出力の数
シリアル = シリアルポートの数
ETH = Ethernet ポート

M221 コントローラー

コントローラー	デジタル入力	デジタル出力	アナログ入力	MOD	FC	HSC	PWM	シリアル	ETH	CART
TM221C16R	9	7	2	TM2/TM3	4	2	0	1	なし	1
TM221C16T	9	7	2	TM2/TM3	4	2	2	1	なし	1
TM221C24R	14	10	2	TM2/TM3	4	2	0	1	なし	1

デジタル入力 = デジタル入力の数
デジタル出力 = デジタル出力の数
アナログ入力 = アナログ入力の数
MOD = 拡張モジュール
FC = 高速カウンター (FC) の数
HSC = 高速カウンター (HSC) の数
PWM = パルス出力の数
シリアル = シリアルポートの数
ETH = Ethernet ポート
CART = カートリッジの数

コントローラー	デジタル入力	デジタル出力	アナログ入力	MOD	FC	HSC	PWM	シリアル	ETH	CART
TM221C24T	14	10	2	TM2/TM3	4	2	2	1	なし	1
TM221C40R	24	16	2	TM2/TM3	4	2	0	1	なし	2
TM221C40T	24	16	2	TM2/TM3	4	2	2	1	なし	2
TM221CE16R	9	7	2	TM2/TM3	4	2	0	1	あり	1
TM221CE16T	9	7	2	TM2/TM3	4	2	2	1	あり	1
TM221CE24R	14	10	2	TM2/TM3	4	2	0	1	あり	1
TM221CE24T	14	10	2	TM2/TM3	4	2	2	1	あり	1
TM221CE40R	24	16	2	TM2/TM3	4	2	0	1	あり	2
TM221CE40T	24	16	2	TM2/TM3	4	2	2	1	あり	2
TM221M16R/G	8	8	2	TM2/TM3	4	2	0	2	なし	0
TM221M16T/G	8	8	2	TM2/TM3	4	2	2	2	なし	0
TM221M32TK	16	16	2	TM2/TM3	4	2	2	2	なし	0
TM221ME16R/G	8	8	2	TM2/TM3	4	2	0	1	あり	0
TM221ME16T/G	8	8	2	TM2/TM3	4	2	2	1	あり	0
TM221ME32TK	16	16	2	TM2/TM3	4	2	2	1	あり	0

デジタル入力 = デジタル入力の数
デジタル出力 = デジタル出力の数
アナログ入力 = アナログ入力の数
MOD = 拡張モジュール
FC = 高速カウンター (FC) の数
HSC = 高速カウンター (HSC) の数
PWM = パルス出力の数
シリアル = シリアルポートの数
ETH = Ethernet ポート
CART = カートリッジの数

EcoStruxure Machine Expert コントローラー

コントローラー	デジタル入力	デジタル出力	アナログ入力	MOD	FC	HSC	PWM	シリアル	ETH	CART
TM241C24R	14	10	0	TM2/TM3	8	2	4	2	なし	1

デジタル入力 = デジタル入力の数
デジタル出力 = デジタル出力の数
アナログ入力 = アナログ入力の数
MOD = 拡張モジュール
FC = 高速カウンター (FC) の数
HSC = 高速カウンター (HSC) の数
PWM = パルス出力の数
シリアル = シリアルポートの数
ETH = Ethernet ポート
CART = カートリッジの数

コントローラー	デジタル入力	デジタル出力	アナログ入力	MOD	FC	HSC	PWM	シリアル	ETH	CART
TM241C24T/U	14	10	0	TM2/TM3	8	2	4	2	なし	1
TM241C40R	24	16	0	TM2/TM3	8	2	4	2	なし	2
TM241C40T/U	24	16	0	TM2/TM3	8	2	4	2	なし	2
TM241CE24R	14	10	0	TM2/TM3	8	2	4	2	あり	1
TM241CE24T/U	14	10	0	TM2/TM3	8	2	4	2	あり	1
TM241CE40R	24	16	0	TM2/TM3	8	2	4	2	あり	2
TM241CE40T/U	24	16	0	TM2/TM3	8	2	4	2	あり	2
TM241CEC24R	14	10	0	TM2/TM3	8	2	4	2	あり	1
TM241CEC24T/U	14	10	0	TM2/TM3	8	2	4	2	あり	1
HMISCU•A5	16	10	0	なし	2	1	2	1	あり	0
HMISCU•B5	8	8	2	なし	2	1	2	1	あり	0

デジタル入力 = デジタル入力の数
デジタル出力 = デジタル出力の数
アナログ入力 = アナログ入力の数
MOD = 拡張モジュール
FC = 高速カウンター (FC) の数
HSC = 高速カウンター (HSC) の数
PWM = パルス出力の数
シリアル = シリアルポートの数
ETH = Ethernet ポート
CART = カートリッジの数

付録 D
技術情報保護

プロジェクトおよびライブラリーの技術情報保護

技術情報保護のメカニズム

EcoStruxure Machine Expert は、次の技術情報保護のメカニズムを取り入れています。

- ユーザー管理は、保護されたオブジェクトへの意図しない変更からエンドユーザーを保護します。
- ユーザー管理は、機能ブロックの技術情報保護としては機能しません。EcoStruxure Machine Expert 自体だけでなく、プラグインやプロジェクトファイル形式を知っている人も、ファンクションブロックを表示または変更できます。
- ライブラリーがコンパイル済みライブラリーとして配布されている場合は、ライブラリーの技術情報保護を実現できます。これらはソースコードを含まず、暗号化されたコンパイル情報のみを含みます。コンパイラーだけがこれらのデータを解釈できます。他の EcoStruxure Machine Expert コンポーネントはこの情報を拒否されます。
  コンパイルされたライブラリーは、その名前が示すものとは対照的に、ターゲットシステムから独立しています。
- プロジェクトファイルを暗号化することでプロジェクトの技術情報保護を実現できます。これを実現するには、プロジェクトパスワードを割り当てます。
  注記：プロジェクトのパスワードを紛失すると、プロジェクトを読み込めなくなります。

用語集

シェル

シェルは、テキストベースのインターフェイスのプログラムでユーザーが対話できる (コンソール) プログラムです。

要素

ARRAY 要素の省略名

CFC

(*continuous function chart*、*コンティニュアスファンクションチャート*) フローチャートのように機能するファンクション・ブロック・ダイアグラム言語 (FBD 言語) に基づくグラフィカルプログラミング言語 (標準規格 IEC 61131-3 の拡張版)。ネットワークは使用せず、グラフィック要素の自由な位置決めが可能なためフィードバックループが利用できます。各ブロックの入力は左側にあり、出力は右側にあります。ブロック出力を他のブロックの入力にリンクして、複雑な式を作成することができます。

DTM

(*device type manager*、*デバイスタイプマネージャー*) 2 つのカテゴリーに分類されます。

- デバイス DTM は、フィールドデバイス設定コンポーネントに接続します。
- CommDTM は、ソフトウェア通信コンポーネントに接続します。

DTM は、デバイスパラメーターにアクセスし、デバイスの設定、操作、および診断するための統一された構造体です。DTM は、デバイスパラメーターを設定するための単純なグラフィックユーザーインターフェイスから、診断および保守用のリアルタイムで複雑な計算ができる高度なアプリケーションにまで及びます。

DUT

(*data unit type*、*データユニットタイプ*) 標準データタイプに加え、DUT エディターでデータタイプ単位として独自のデータタイプ構造、列挙型、およびリファレンスを定義できます。

FBD

(*function block diagram*、*ファンクションブロックダイアグラム*) 制御システム用 標準規格 IEC 61131-3 でサポートされているロジックまたは制御用の 5 つの言語の 1 つ。ファンクションブロックダイアグラムは、グラフィカルなプログラミング言語です。ネットワークのリストで動作し、各ネットワークは、論理式、算術式、ファンクションブロックの呼び出し、ジャンプ、またはリターン命令のいずれかを表すボックスと接続線のグラフィックで構成されます。

GRAFCET

一連のオペレーションがどのように機能しているかを、構造化されたグラフィック形式で表したもの。

シーケンス制御システムを、アクション、移行、および条件を含む一連のステップに分割する分析的手法。

GVL

(*global variable list*、*グローバル変数リスト*) EcoStruxure Machine Expert プロジェクト内でグローバル変数を管理します。

IL

(*instruction list*、*インストラクションリスト*) コントローラーにより順に実行される一連のテキストベースの命令で書かれたプログラム。各命令は、ライン番号、命令コードおよびオペランドを含む。
(IEC 61131-3 を参照してください。)

LD

(*ladder diagram*、*ラダーダイアグラム*) コントローラープログラムの命令を表す図。コントローラーで順次実行される一連のラングにある接点、コイル、およびブロックのシンボルを含む。(IEC 61131-3 参照)

MAC アドレス

(*media access control address*、*メディアアクセス制御アドレス*) 特定のハードウェアに関連付けられた固有の 48 ビット番号。MAC アドレスは、製造時に各ネットワークカードまたはデバイスにプログラムされます。

OPC UA

OPC 統合アーキテクチャー。

POU

(*program organization unit*、*プログラムオーガニゼーションユニット*) ソースコード内の変数宣言、および対応する命令セット。POUs はソフトウェアプログラム、ファンクション、およびファンクションブロックのモジュラー化した再利用を容易にします。一度宣言すると POU 同士で利用可能となります。

Python インタプリタ

入力された命令を直接実行するプログラム。

REPL

(*read-eval-print-loop*、*読み込み - 評価 - プリント - ループ*) シンプルな対話型コンピュータープログラミング環境。単一のユーザー入力をとり、評価して、結果を返します。

RTS

(*request to send*、*送信要求*) データの送信信号、および通信先からの RTS を認識する CTS 信号。

Sercos

(*serial real-time communications system*、*シリアルリアルタイム通信システム*) 数値的に制御される機械およびシステムの、モーション制御、ドライブ、I/O、センサー、およびアクチュエーターを相互接続するデジタル制御バス。これは、標準化されたオープンなコントローラーから高度なデジタルデバイスへのインターフェイスであり、標準化された閉ループリアルタイムデータの高速シリアル通信用に設計されています。

SFC

(*シーケンシャル ファンクションチャート*) アクション付きステップ、ロジック条件付き遷移、およびステップと遷移間のリンクで構成される言語。(SFC は、IEC 848 で定義されている。IEC 61131-3 準拠。)

TLS

(*transport layer security*、*トランスポート層セキュリティ*) 暗号化されたデータ転送用プロトコル。Successor of *SSL*.

UDP

(*User Datagram Protocol*、*ユーザーデータグラムプロトコル*) コネクションレスモードプロトコル (IETF RFC 768 により定義)。メッセージはデータグラム (データテレグラム) で IP ネットワーク上の送信先コンピューターに送信されます。通常、UDP プロトコルはインターネットプロトコルに付属しています。UDP/IP メッセージは応答を待つことがないため、損失したパケットの再送信の必要がないアプリケーション (リアルタイムパフォーマンスが必要なストリーミングビデオやネットワークなど) に最適です。

UTC

(*universal time coordinated*、*協定世界時*) 世界で時計と時間を調整した主要な基準時刻。

索引