

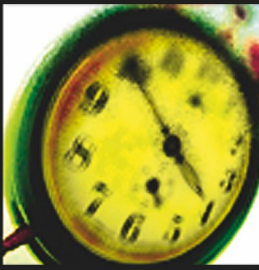
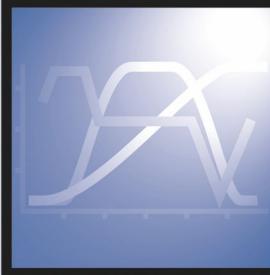
マニュアル | JA

TwinCAT 2

クイックスタート



TwinCAT 2 | System



目次

1 序文	5
1.1 取扱説明書に関する注記	5
1.2 安全にご使用いただくために	5
1.3 情報セキュリティに関する注記	6
2 TwinCATの使い方とシステム要件	8
3 インストール	9
3.1 インストールプログラムの起動	9
3.2 インストールの終了.....	17
4 概要	20
4.1 PLC規格IEC 61131-3.....	21
5 概要	24
6 概要	25
7 サンプルプログラム	26
7.1 サンプルMaschine.pro	26
7.2 プログラムフローの追跡	34
7.3 サンプルプログラムの変換	39
7.3.1 変数の宣言	39
7.3.2 Lightbus - バスターミナルのセットアップ	40
7.3.3 イーサネット - バスターミナルのセットアップ	50
7.3.4 EtherCAT - バスターミナルのセットアップ	60
7.4 「Machine」を使用したサンプル	72
7.4.1 Microsoft Visual C#による「Machine」サンプル	72
7.4.2 Microsoft Visual Basic .NETによる「Machine」サンプル	78
7.4.3 Microsoft Visual Basic 6.0を使用した機械サンプル	83
7.4.4 Microsoft Visual C++ による「Machine」サンプル.....	86
7.4.5 Microsoft Expression Blend による「Machine」サンプル.....	95
7.4.6 Microsoft Windows Vista Media Center による「Machine」サンプル.....	100
7.4.7 Microsoft SilverlightとJavaScript による「Machine」サンプル.....	107
7.4.8 Microsoft Silverlight for Windows Embedded による「Machine」サンプル.....	113

1 序文

1.1 取扱説明書に関する注記

この説明書は対応する国内規格を熟知した、トレーニングを受けた制御、オートメーションエンジニアリングの有資格者のみの使用を対象としています。

本製品の設置およびコミッショニングの際は、必ず以下の注意事項と説明に従ってください。

有資格者は、常に最新版のドキュメントを参照する管理義務があります。

本製品を使用する上での責任者は、本製品の用途および使用方法が、関連するすべての法律、法規、ガイドラインおよび規格を含む、安全に関するすべての要件を満たしていることを確認してください。

免責事項

この取扱説明書の記載内容は、一般的な製品説明および性能を記載したものであり、場合により記載通りに動作しないことがあります。

製品の情報・仕様は予告なく変更されます。

この説明書に記載されているデータ、図および説明に基づいて、既に納品されている製品の変更を要求することはできません。掲載されている写真やイラストと、実際の製品は異なる場合があります。この説明書は最新でない可能性があります。必ず最新バージョンの説明書を参照してください。

商標

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS®, XPlanar® は、Beckhoff Automation GmbH の登録商標です。

この取扱説明書で使用されているその他の名称は商標である可能性があり、第三者が独自の目的のために使用すると所有者の権利を侵害する可能性があります。

特許

EtherCAT Technologyについては、欧州特許

EP1590927、EP1789857、EP1456722およびEP2137893、ドイツ特許DE102015105702

に記載されていますが、これらに限定されるものではありません。

EtherCAT®

EtherCAT®は、Beckhoff Automation GmbHの登録商標および特許技術です。

著作権

© Beckhoff Automation GmbH & Co.KG, Germany.

明示的な許可なく、本書の複製、配布、使用、および他への内容の転載は禁止されています。

これに違反した者は損害賠償の責任を負います。ベッコフは、特許、実用新案、意匠の付与に関するすべての権利を留保しています。

1.2 安全にご使用いただくために

安全に関する注意事項

本書の各項目に記載されている製品の安全に関する指示をよくお読みになり、それに従ってください。

免責事項

すべての製品は、用途に適した特定のハードウェア構成およびソフトウェア構成を有する状態で供給されます。ハードウェアまたはソフトウェアに取扱説明書に記載されている以外の変更を加えることは許可されていません。許可されていない変更を加えると、Beckhoff Automation GmbH & Co. KGの保証の対象外となります。

使用者の資格

この説明は対応する国内規格を熟知した、トレーニングを受けた制御、自動化、およびドライブ技術の有資格者のみを対象としています。

安全記号の説明

この取扱説明書では、以下の安全記号を使用します。人的傷害および物的損害を避けるため、安全に関する注意事項はよくお読みになり、必ず指示に従ってください。人的傷害および物的損害を避けるため、安全に関する注意事項はよくお読みになり、必ず指示に従ってください。

人的傷害に関する警告:

⚠ 危険

この記号がついた注意事項に従わない場合、死亡または重傷を負います。

⚠ 警告

この記号がついた注意事項に従わない場合、死亡や重傷を負う可能性があります。

⚠ 注意

この記号がついた注意事項に従わない場合、軽傷を負う可能性があります。

物的損害に関する警告

注記

環境汚染、物的損害またはデータ消失の可能性があります。

製品の取り扱いに関する情報



ここには
製品に関する推奨措置、援助、または追加情報などが記載されます。

1.3 情報セキュリティに関する注記

Beckhoff Automation GmbH & Co. KG (ベッコフ) の製品は、オンラインアクセスが可能であれば、プラント、システム、機械、ネットワークの安全な運用をサポートするセキュリティ機能を備えています。セキュリティ機能にもかかわらず、プラント、システム、機械、ネットワークをサイバー脅威から守るためには、運用のための全体的なセキュリティ コンセプトの作成、実施、継続的な更新が必要です。ベッコフが販売する製品は、全体的なセキュリティ コンセプトの一部に過ぎません。お客様は、プラント、システム、機械、ネットワークへの第三者による不正アクセスを防止する責任を負います。ネットワークは、適切な保護措置が講じられている場合にのみ、社内ネットワークまたはインターネットに接続すべきです。

また、ベッコフが推奨する適切な保護対策も遵守してください。情報セキュリティと産業セキュリティに関する詳細は、<https://www.beckhoff.com/secguide> を参照してください。

ベッコフの製品とソリューションは常に進化し続けています。これはセキュリティ機能にも当てはまります。継続的な開発により、ベッコフでは、製品を常に最新の状態に保ち、アップデートが提供され次第、製品にインストールすることを明示的に推奨しています。古いバージョンやサポートが終了した製品の使用は、サイバー脅威のリスクを高めるおそれがあります。

ベッコフ製品の情報セキュリティ情報については、RSSフィードをご購読ください <https://www.beckhoff.com/secinfo>。

2 TwinCATの使い方とシステム要件

本項の目的は、TwinCATの概要を紹介することであり、詳細については立ち入りません。各章のサンプルアプリケーションで補足説明しています。

詳細情報については、各プログラムの説明を参照してください。

システム要件

x86 プロセッサ

TwinCAT2の動作には、x86プロセッサを搭載したPCが必要です。

オペレーティングシステム Windows10/11

TwinCAT 2.11は、Windows10 OSで動作します。

ランタイムにはWindows10の32ビット版が必要です。

エンジニアリングは、Windows 10/11の64ビット版に対応しています。

サンプルプログラム

サンプルプログラムを使用するには、TwinCATバージョン2.11以降が必要です。

このサンプルはフィールドバス(Lightbus、イーサネットまたはEtherCAT)に対して有用です。実装の要件は以下の通りです：

- - I/O-Lightbus (FC2001)またはイーサネット(FC9001)用のPCインターフェースカード
- - バスカプラ
- - バスターミナル x 2 (KL2032, 1枚につき2チャンネルのデジタル出力を搭載)
- - バスエンドターミナル (KL9010)
- - 配線部材(ファイバケーブル、より線など)
- - 24 V電源ユニット

これらの要件に従えば、他のフィールドバスも使用可能です。

デモキット:

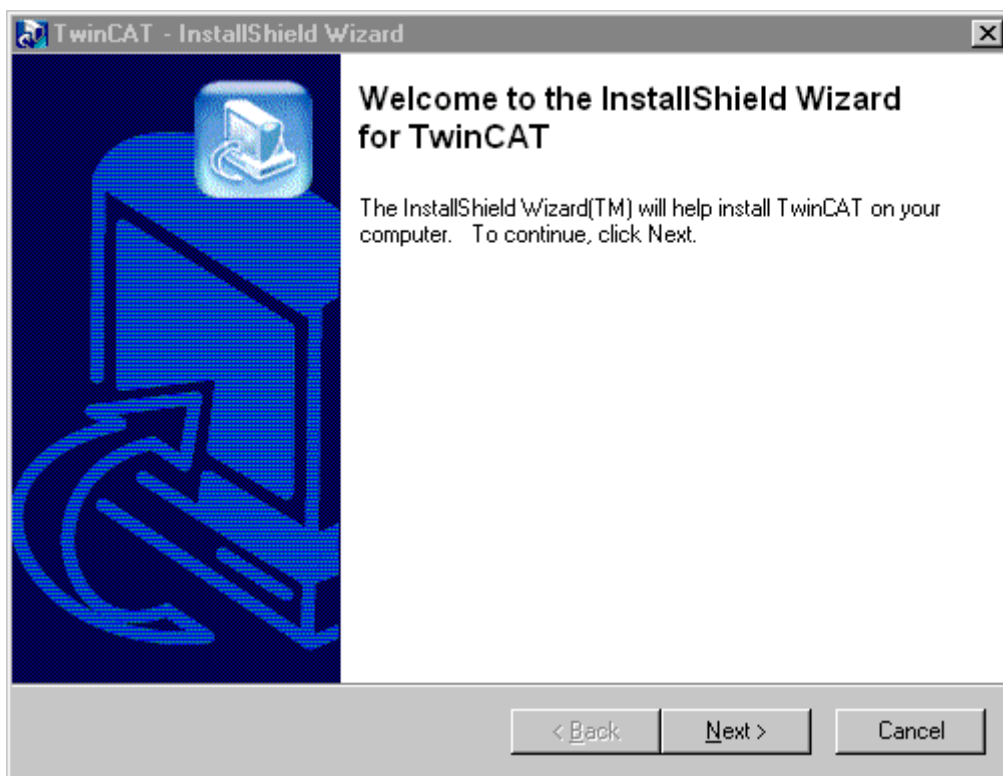
サンプルプログラムに必要なハードウェアがベッコフデモキットに含まれています。

3 インストール

3.1 インストールプログラムの起動

CDのSETUP.EXEプログラムを起動します。起動するには、エクスプローラでCD ROM内を開き、SETUP.EXEプログラムをダブルクリックします。

次のダイアログボックスが開きます。[Next]をクリックして続行します。



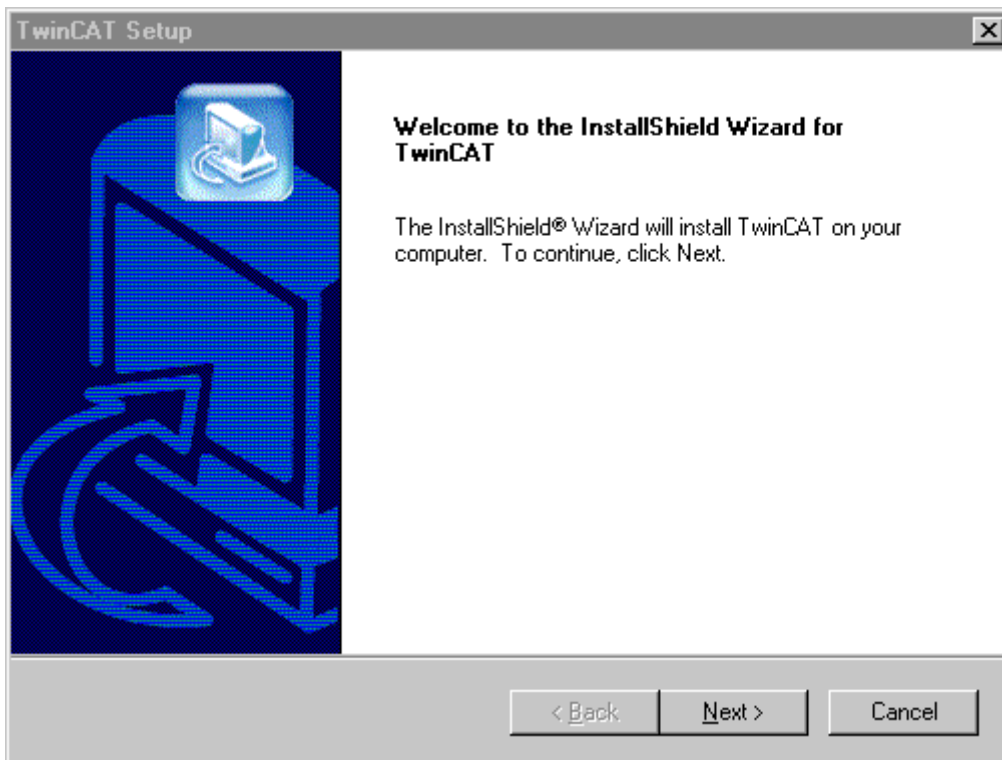
言語の選択

TwinCATをどの言語でインストールするかを選択します。例えば英語でインストールするには、エントリ[English]を選択し、[OK]をクリックして入力を確定します。インストールが完全にメニューに従って進められます。

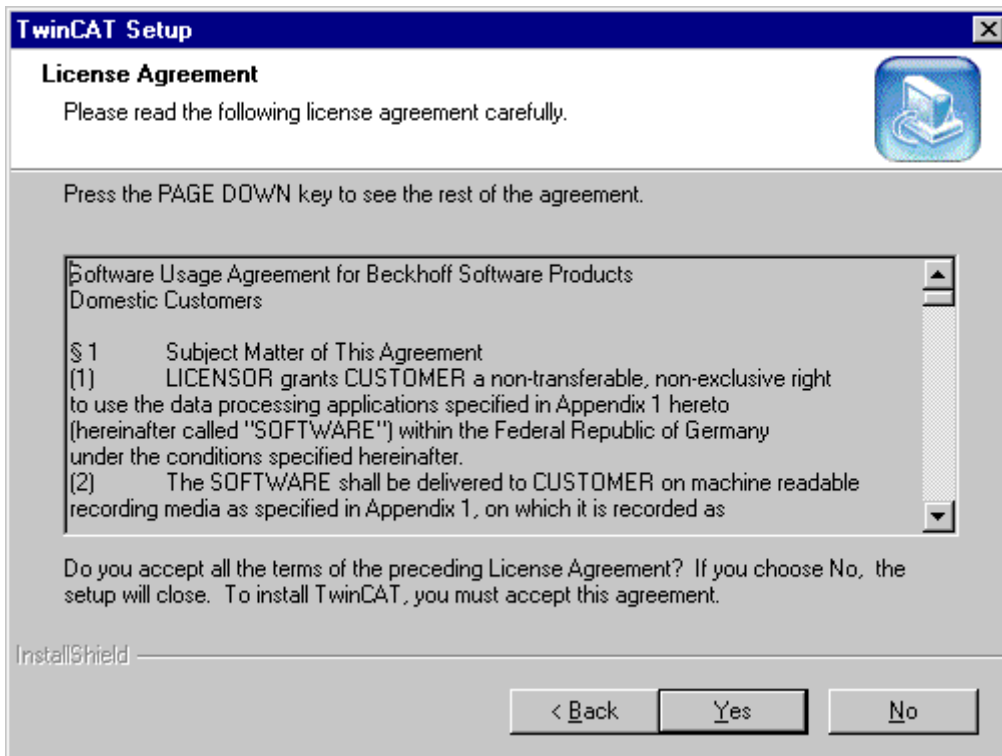


プログラムの終了

このセットアッププログラムを実行する前に他の実行中のすべてのWindowsプログラム（アプリケーション）を終了して行うことを推奨します。

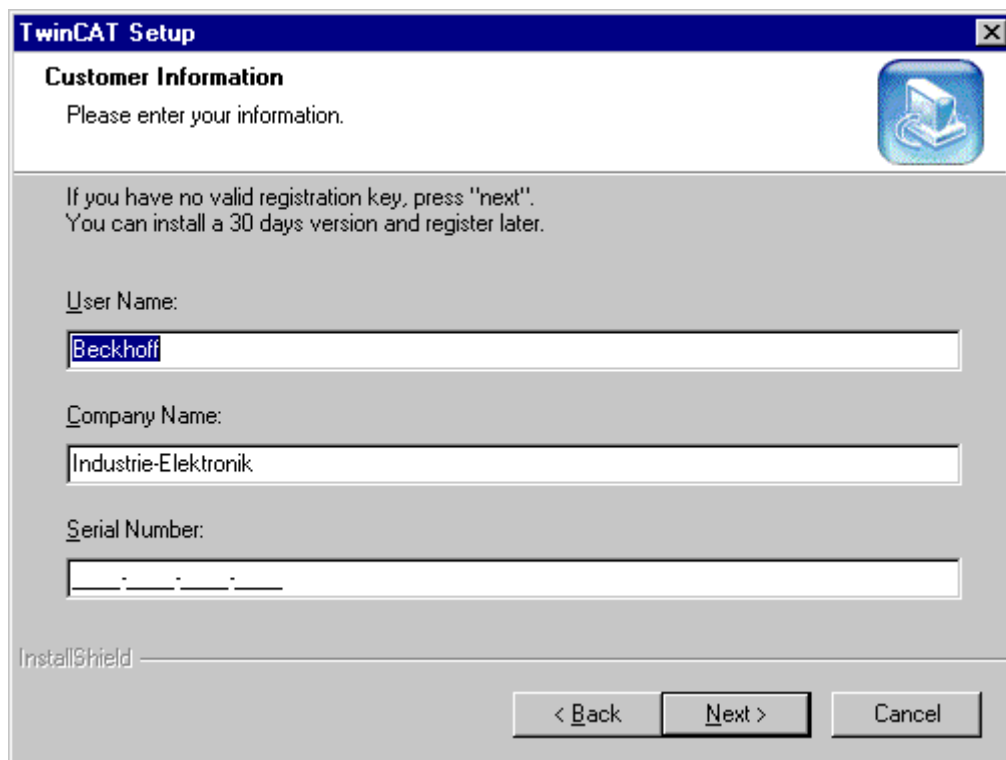


ライセンス契約



ユーザ情報の入力

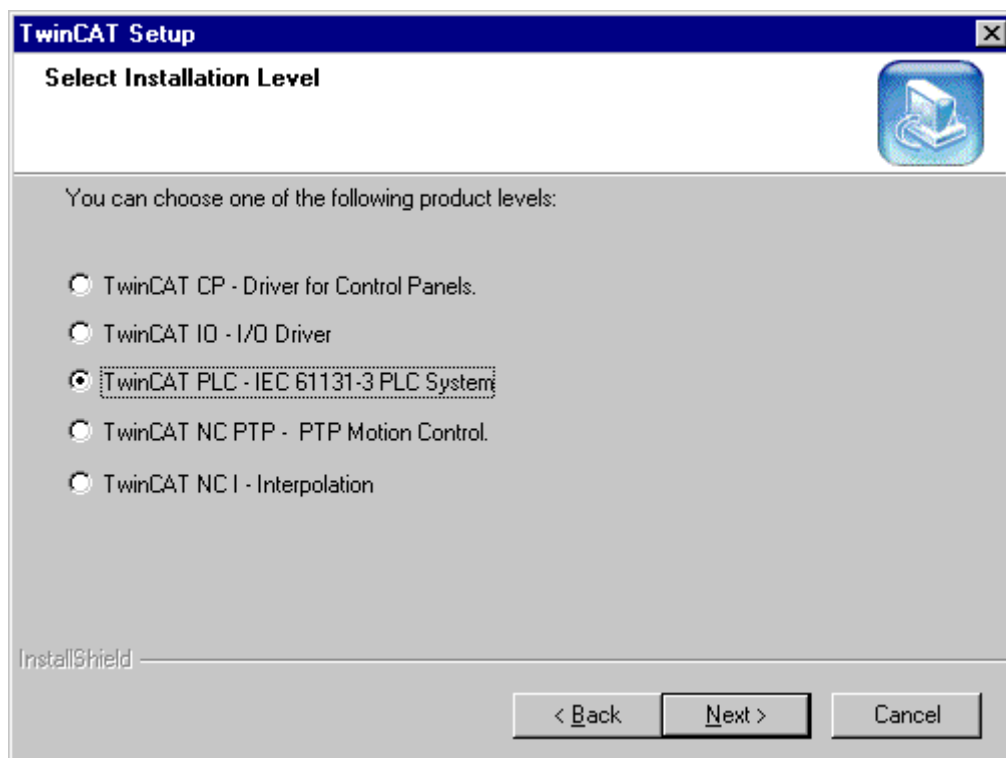
このダイアログボックスにシリアル番号を入力する必要があります。シリアル番号はTwinCAT購入契約に記載されています。TwinCATのデモバージョンをインストールする場合は、このボックスを空白にしておいてください。



The screenshot shows the 'TwinCAT Setup' window with the 'Customer Information' tab selected. The window title is 'TwinCAT Setup'. The main heading is 'Customer Information' with a sub-heading 'Please enter your information.' and a small computer icon. Below this, there is a note: 'If you have no valid registration key, press "next". You can install a 30 days version and register later.' The form contains three input fields: 'User Name:' with the text 'Beckhoff', 'Company Name:' with the text 'Industrie-Elektronik', and 'Serial Number:' which is empty. At the bottom, there is an 'InstallShield' logo and three buttons: '< Back', 'Next >', and 'Cancel'.

インストールレベルの選択

次のインストールレベルのいずれかを選択する必要があります。



The screenshot shows the 'TwinCAT Setup' window with the 'Select Installation Level' tab selected. The window title is 'TwinCAT Setup'. The main heading is 'Select Installation Level' with a small computer icon. Below this, there is a note: 'You can choose one of the following product levels:'. There are five radio button options: 'TwinCAT CP - Driver for Control Panels.', 'TwinCAT IO - I/O Driver', 'TwinCAT PLC - IEC 61131-3 PLC System' (which is selected), 'TwinCAT NC PTP - PTP Motion Control.', and 'TwinCAT NC I - Interpolation'. At the bottom, there is an 'InstallShield' logo and three buttons: '< Back', 'Next >', and 'Cancel'.

製品レベルの説明:

TwinCAT CP

ベッコフ操作パネルの特殊機能(UPS、S-Keysなど)に必要なコンポーネントが含まれています。

TwinCAT IO

(ユーザモード)プログラムがIOデバイスに直接アクセスします。このレベルにはPLCが含まれていません。

TwinCAT PLC

TwinCAT PLCには、IEC61131-3ソフトウェア開発キットが含まれています。

TwinCAT NC PTP

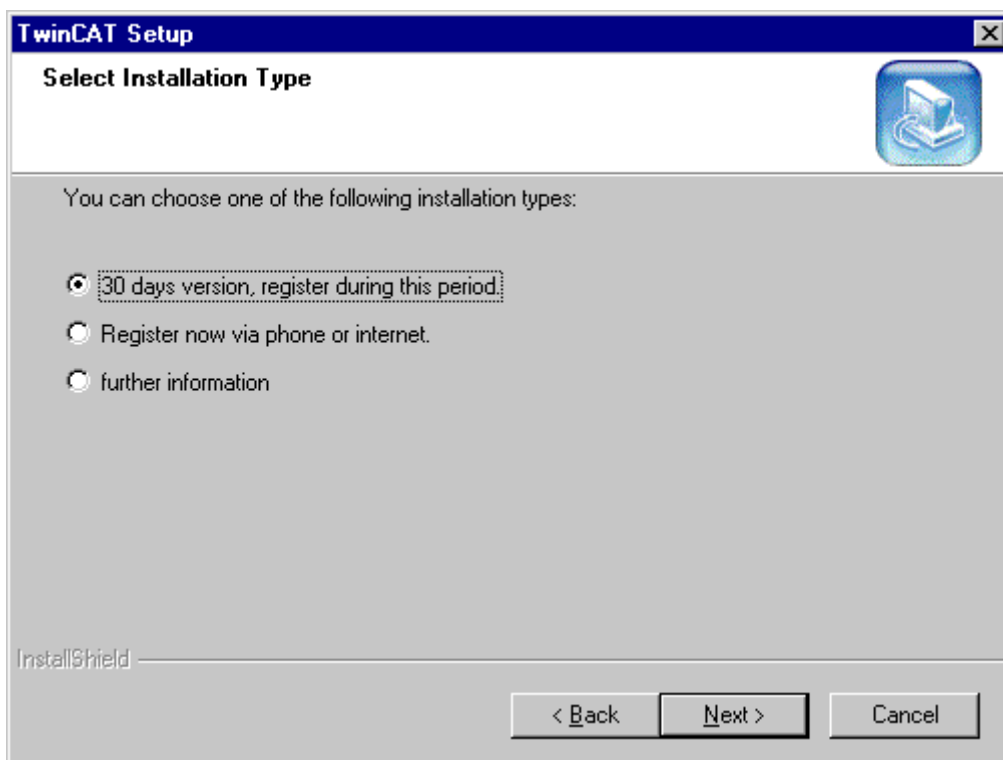
PLCに加えて、このモジュールには、PTP軸を制御するためのNC/CNC機能が含まれています。

TwinCAT NC I

PLCに加えて、このモジュールには、3Dでドライブを補間するNC機能が含まれています。

インストールタイプ

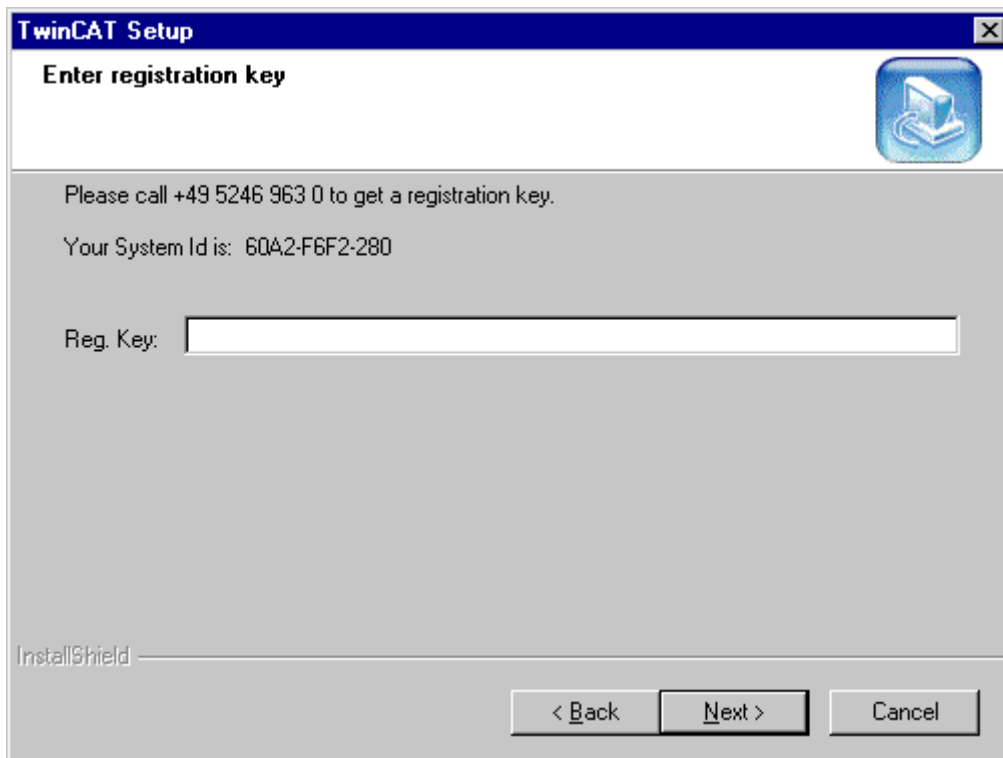
30日テストバージョンとしてTwinCATのインストールの登録、または詳細情報の要求を選択します。登録番号がない場合は、[30 days Version register during this period]を選択してください。



インストールタイプ	制限事項
30日試用バージョン	TwinCATを30日間、無制限に使用できます。この期間中にライセンス番号を申請する必要があります。これを行わないと、30日が経過した後にプログラムを起動できなくなります。
電話またはインターネットによる登録	インストールを完了したら、ライセンス番号の入力を求められます。下記を参照してください。

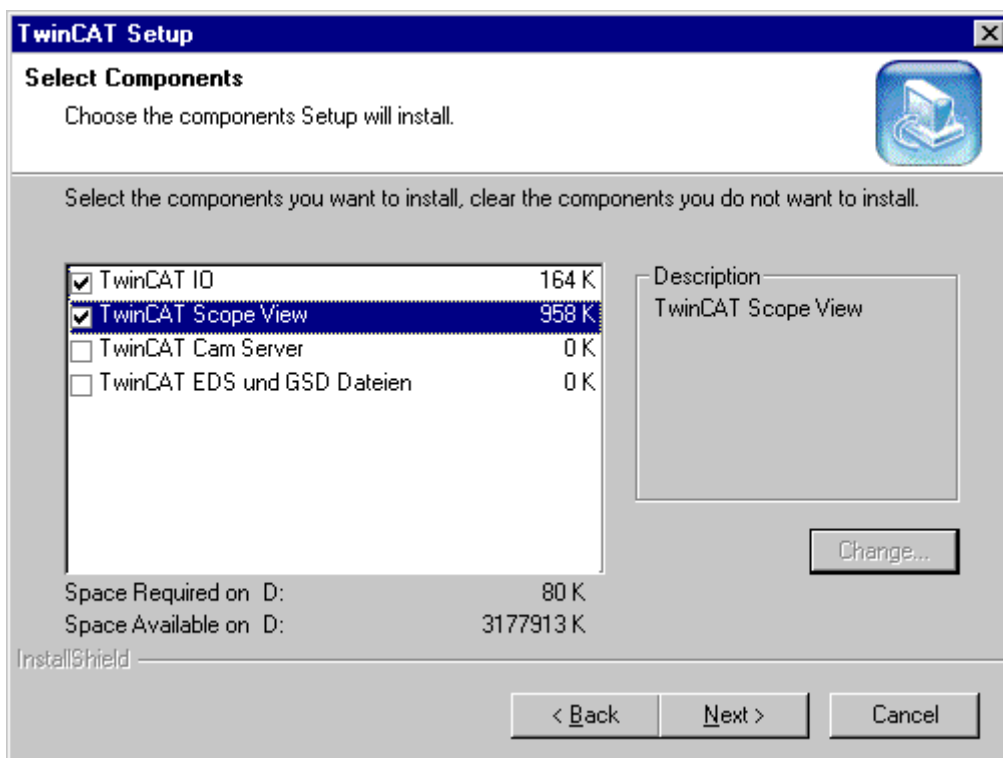
レジストレーションキー

TwinCATのライセンス登録をする場合は、レジストレーションキーを入力する必要があります。このレジストレーションキーをベッコフオートメーションから直接取得する必要があります。電話番号がダイアログボックスに表示されます。レジストレーションキー発行のためにシステムIDを提供する必要があります。このシステムIDはダイアログボックスに表示されます。



コンポーネントの選択

標準設定では、TwinCATのすべてのコンポーネントがインストールされるわけではありません。

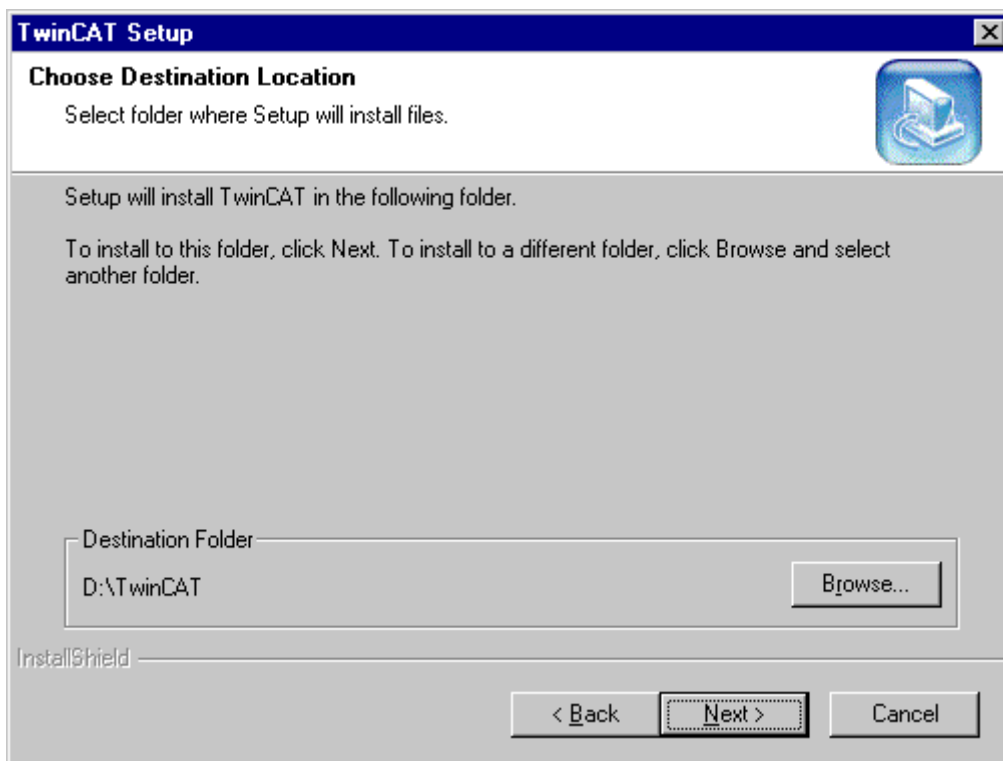


要件

コンポーネント	説明
TwinCAT IO	DLL経由でのIOへのダイレクトアクセスを許可します。TwinCAT PLCまたはTwinCAT NC PTPとともにインストールできます。
TwinCATスコープビュー	TwinCATプロセス変数をグラフィックに可視化するためのプログラムです。
TwinCAT Cam Server	高速カムサーバです。
TwinCAT EDSファイルおよびGSDファイル	EDS (DeviceNet)およびGSD (特性マスタデバイスファイル、PROFIBUS)は、システムの構成のためにユーザがすべての設定を使用できるようにします。

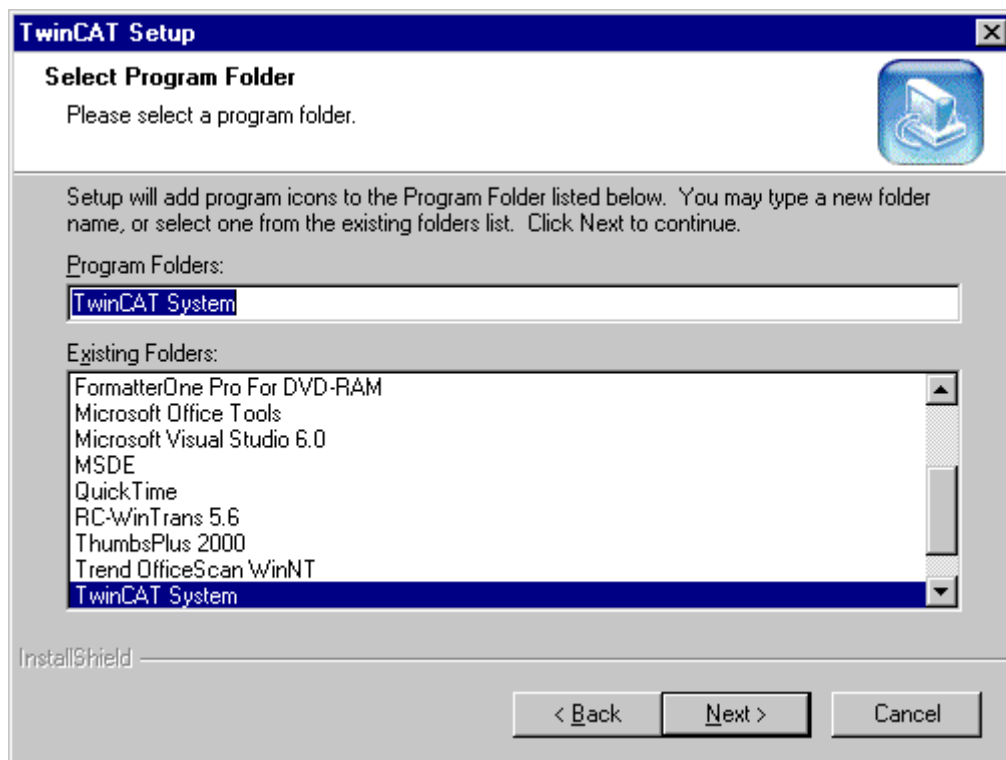
保存先パスとプログラムフォルダの選択

ここで任意のディレクトリを選択することも、プログラムフォルダを選択することもできます。通常は既定の保存先が指定されています。



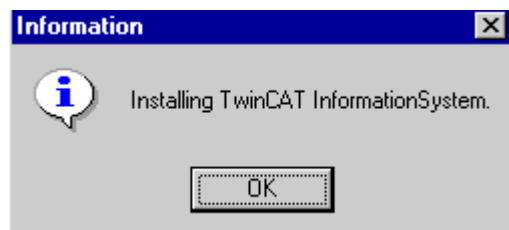
プログラムフォルダの選択

ここで任意のディレクトリを選択することも、プログラムフォルダを選択することもできます。通常は既定の保存先が指定されています。



TwinCATのインストール後

Beckhoff Information Systemのインストールが自動的に開始されます。Beckhoff Information Systemには、TwinCATのドキュメンテーションが含まれています。[OK]をクリックして、このインストールを開始します。

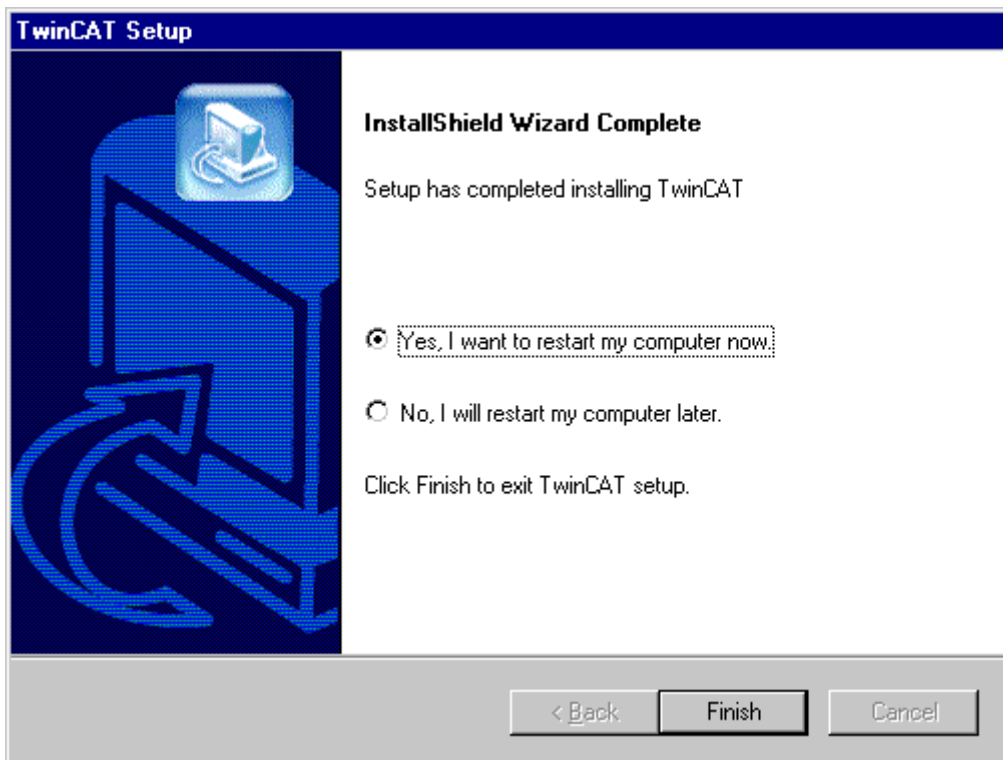


Beckhoff Information Systemのインストールの終了

[Finish]をクリックして、Beckhoff Information Systemのインストールを終了します。その後、インストールもTwinCATによって終了されます。

コンピュータの再起動

インストールが完了した後、コンピュータを再起動する必要があります。



この時点で、セットアップによりTwinCATのインストールが完了します。

サイレントインストール

InstallShieldに応答ファイルを作成させるというオプションがあります。Setup.exe-rコマンドラインパラメータを使用してセットアップを実行するだけで済みます。InstallShieldがセットアップの選択をすべてSetup.issに記録し、このファイルをWindowsフォルダに配置します。

InstallShieldの内蔵型/Sdダイアログボックス機能は、InstallShieldが記録モード(Setup -r)で実行するときに、値をSetup.issファイルに書き込むように設計されています。カスタムダイアログボックスを作成する場合、SdMakeNameとSilentWriteDataを呼び出して、セットアップが記録モードで実行しているときに、セクションとダイアログボックスデータを応答ファイルに追加する必要があります。これらの機能を使用してSetup.issに書き込む例については、<InstallShield location>¥IncludeフォルダのSdダイアログのソースコードを参照してください。SdMakeNameとSilentWriteDataの呼び出し時にSetup.issに追加する必要があるデータに関する詳細については、次のセクションをお読みください。

セットアップと応答ファイルを作成した後、以下を行ってください。

1. 対応ファイル(Windowsフォルダにある)を[Setup Files]枠内のAdvanced¥Disk 1フォルダに配置します。
2. メディアをビルドします。セットアップ用の自己解凍型実行ファイルを作成する場合、メディアウィザードの[Self-Extracting Package]パネルまたはメディアプロパティシートの[Packaging]ページの[Setup Command Line Parameters]編集ボックスに-sを入力します。

これで、InstallShield Silentを使用してサイレントモードでセットアップを実行する準備が整います。サイレントモードでセットアップを実行する際は、メッセージが表示されないので注意してください。メッセージの代わりに、Setup.logという名前のログファイルにセットアップ情報(セットアップの成否を含む)が保存されます。このログファイルを調べて、セットアップの結果を判断することができます。(一部のセットアップ初期化エラーの場合、代わりにSetupexe.logという名前のログファイルがSUPPORTDIR (セットアップがインターネットから実行される場合)またはSRCDIR (それ以外の場合)に作成されます。

InstallShield Silentを開始するには、-sオプションを使用してSetup.exeを実行してください。

InstallShieldには-f1スイッチと-f2スイッチもあるため、応答ファイルの名前と場所およびログファイルの場所を指定することができます。

サイレントセットアップが成功したことを確認するには、Setup.logの[ResponseResult]セクションのResultCode値を調べてください。InstallShieldがResultCodeキー名の後に適切な戻り値を書き込みます。

Setup.logはサイレントセットアップログファイルのデフォルト名で、その既定の場所はDisk1 (Setup.inxと同じフォルダの)です。Setup.exeで-f1スイッチと-f2スイッチを使用して、Setup.logと異なる名称と場所を指定することができます。

2番目のセクション[Application]に、インストールされたアプリケーションの名前とバージョンおよび会社名が示されます。

3番目のセクション[ResponseResult]には、サイレントセットアップの成否を示す結果コードが含まれます。[ResponseResult]セクションのResultCodeキー名に整数値が割り当てられます。InstallShieldがResultCodeキー名の後に以下の戻り値のいずれかを配置します。

- 0 成功。
- 1 一般的なエラー。
- 2 無効なモード。
- 3 必要なデータがSetup.issファイルにない。
- 4 使用可能なメモリが不足している。
- 5 ファイルが存在しない。
- 6 応答ファイルに書き込むことができない。
- 7 ログファイルに書き込むことができない。
- 8 InstallShield Silent応答ファイルへのパスが無効である。
- 9 有効なリストタイプ(文字列または数)でない。
- 10 データタイプが無効である。
- 11 セットアップ中の不明なエラー。
- 12 ダイアログボックスが正常でない。
- 51 指定されたフォルダを作成できない。
- 52 指定されたファイルまたはフォルダにアクセスできない。
- 53 無効なオプションが選択された。

3.2 インストールの終了

新しいプログラムアイコン

インストール後、Windows NT/2000のスタートアップメニューに、5つのプログラムシンボルを含む新しいフォルダと、2つの追加プログラムフォルダが含まれるようになります。



スタートアップ



[Auto-Boot]ファンクションによって有効にされている場合、TwinCATが自動的にStartupフォルダのすべてのプログラムを起動します。その目的は、TwinCATからのプロセス値を使用するユーザプログラムがTwinCATの後に起動されることを保証するためです。

TwinCATシステムマネージャ



このプログラムを利用して、物理的I/Oアドレス(Fieldbus)を論理的プロセス変数(PLCプログラム)に割り当てます。この割り当てはマッピングと呼ばれます。ここでリアルタイムプロパティが定義されます。

TwinCAT PLC Control



これは、IEC61131-3に対応するソフトウェア開発キットです。ここでPLCプログラムの書き込みとテストを行います。

TwinCAT System Control



可視的なプログラムの他に、バックグラウンドで実行される隠れたタスクとドライバも存在します。TwinCAT System Controlはこれらのプログラムを管理します。

TwinCATスコープビュー



TwinCAT Scope Viewを利用して、プロセス値をリアルタイムでグラフィック表示することができます。動的な軸値を完全に調べることができます。

Windows NT/2000でのTwinCAT

システムの起動時に、TwinCATリアルタイムサーバアイコンがタスクバーの右に表示されます。



システムの一般的な動作状態が色によって示されます。色は、「起動済み」(緑)、「起動中」(黄)、および「停止」(赤)のいずれかです。このアイコンをクリックすると、ポップアップメニューが開きます。このメニューで、他のシステム設定を定義できます。これらの指示の範囲内で、初期設定を受け入れることができます。このメニューでTwinCATサーバの停止と起動ができます。

Beckhoff Information System



Beckhoff Information SystemはTwinCAT製品に関する情報源であり、継続的な改良を続けています。Beckhoff Information Systemには、技術情報、マニュアル、サンプルコード、TwinCAT知識ベースなどが含まれています。文書が階層的に配列されているため、必要な情報を容易に見つけることができます。

完全版

ベッコフ製品CDのTwinCATをインストールした場合、Beckhoff Information System全体がコンピュータにインストールされています。

基本バージョン

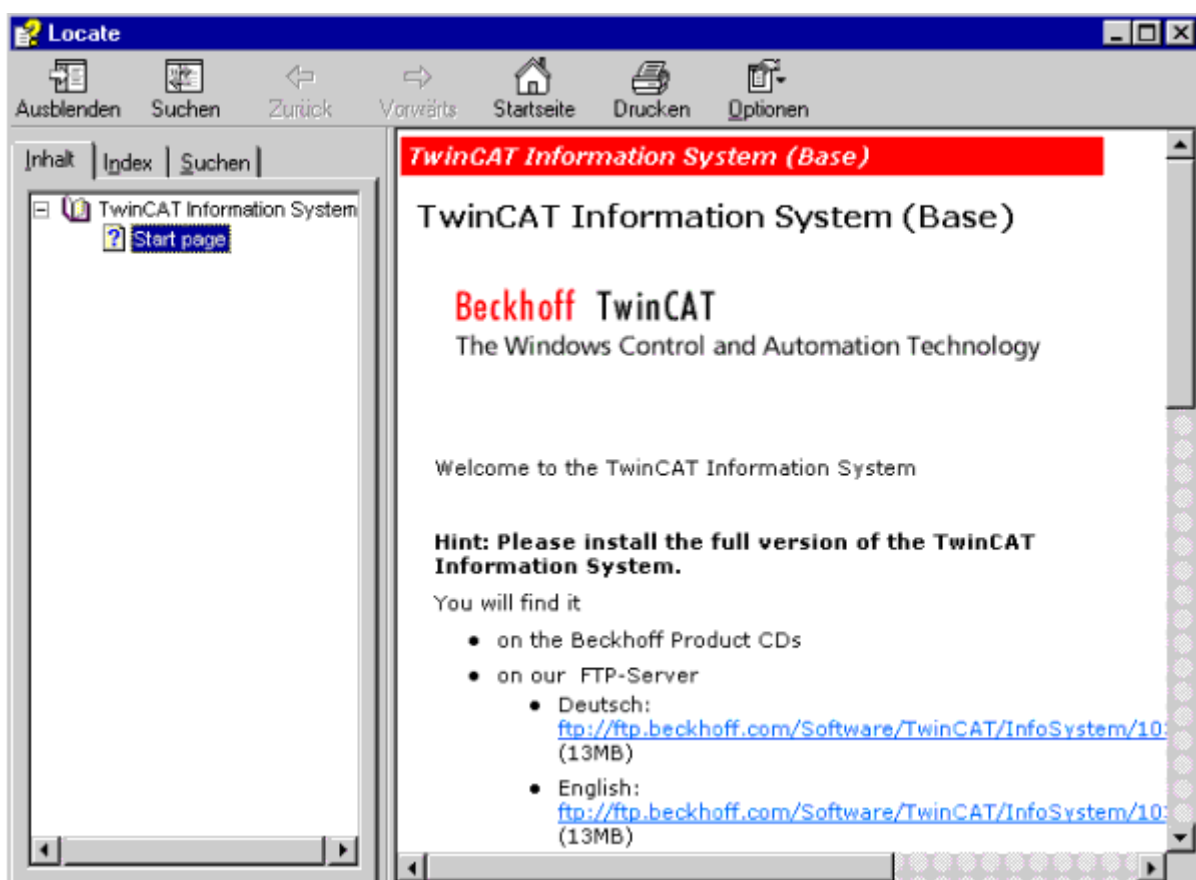
インターネットからTwinCATのインストールをロードした場合、Beckhoff Information Systemの基本バージョンのみがコンピュータにインストールされます。

完全版(サイズ200 MB)を入手するにはいくつか方法があります。

完全版が存在する場所

- ベックホフ製品CD
- 当社のFTPサーバ
 - <http://download.beckhoff.com/download/Software/TwinCAT/TwinCAT2/InfoSystem/1033/Install/InfoSys.exe>
- 当社のWebサーバ
 - <http://www.beckhoff.com>

スタートページからインストールに関する情報を得ることもできます。スタートページは、上記のようにスタートメニューから開くことができます。



4 概要



TwinCAT PLC Controlとは?

TwinCAT PLC ControlはPLCの完全な開発環境です。エディタとデバッグ機能の使用は、高度なプログラミング言語の実績のある開発プログラム環境に基づいています。

IEC 61131-3

TwinCAT PLCを使用すると、PLCプログラマがIEC 61131-3の強力な言語資源に容易にアクセスできるようになります。TwinCAT PLC開発の過程で、以下の機能が実装されました。

ファンクションブロック

TwinCAT PLCはさまざまなプログラミング言語をサポートしています。インストラクションリスト(IL)、ストラクチャードテキスト(ST)、シーケンシャルファンクションチャート(SFC)、ファンクションブロックダイアグラム(FBD)、連続ファンクションチャートエディタ(CFC)、ラダーダイアグラム(LD)です。

PLCを使用しないテスト

内蔵ソフトウェアPLCを使用すると、外部ハードウェアなしでPLCプログラムをテストすることができます。

動作中の変更

プログラムは、PLCで「オンライン」で変更されます。

再利用性

既存のPLCプログラムブロックの再利用可能性

標準化されたインターフェイス

標準化されたオープンインターフェイス(OCX、DLLなど)のおかげで、他のプログラムとコンピュータへのリンク(ネットワーク経由であっても)が可能です。

異種環境

システムに依存しない広く普及したネットワークプロトコルを使用しているため、TwinCATを異種ネットワーク環境に統合することが可能です。例えば、UNIXの下で実行されているOracleデータベースがTCP/IP経由でTwinCATとデータを交換して、そのデータをさらにPDAまたはPPSシステムで処理したり、製造プロセスに対応するためにTwinCAT PLC内でパラメータを指定したりすることができます。

高級言語ライブラリ

複雑なアルゴリズムをC/C++またはアセンブラで開発することができます(例えば、TwinCAT PLCからアルゴリズムに対処するために)。TwinCAT PLCからそれら进行处理するために使用されます。そこには、特定の作業領域ライブラリーを扱うサードパーティのベンダーが多数存在します。

SCADAシステム

SCADAシステム(Fix32、InTouch、Citect、Genesis、Wizconなど)の製造メーカーの一部は、TwinCATにリンクするための直接ドライバサポートを提供しています。

リモートアクセス

プログラミング環境とランタイム環境が分離している事により、ネットワーク(ISDNを含む)を通じた分散制御システムの中央プログラミングが可能です。

直観的な開発環境

TwinCAT PLCを使用すると、最新の開発環境と同様に、技術的に成熟した高級言語開発環境(Visual C++など)、ブレイクポイント、シングルステップモード、変数のトレースなどの例によるシミュレーションが可能です。

4.1 PLC規格IEC 61131-3

IEC 61131-3に準拠する5つの異なる言語を、TwinCAT PLCを用いたPLCプログラムの作成に使用できます。

インストラクションリスト(IL)

インストラクションリストはSTEP5プログラミング言語によく似ています。各インストラクションが新しい行で始まり、1つの演算子と1つまたは複数のオペランドが含まれています。インストラクションでラベルが先行し、それにコロンが続きます。コメントは、1行の最後の項目でなければなりません。

例:

ラベル	演算子	オペランド	コメント
Start:	LD	Basin_level	(* 負荷レベル *)
	GE	13	(* 制限に達した? *)
	JMPC	Pump_on	
	R	Pump_control	(* ポンプオフ *)
	JMP	End	
Pump_on:	S	S Pump_control	(* ポンプオン *):
End:			

ストラクチャードテキスト(ST)

このプログラミング言語の場合、使用されるのは「機械指向」のコマンドではないため、より高いレベルのプログラミング言語とも言えます。その代わりに、抽象的なコマンドを使用してパワーコマンド文字列を作成できます。PC分野における同等の高級プログラミング言語として、Basic、PASCAL、Cが挙げられます。

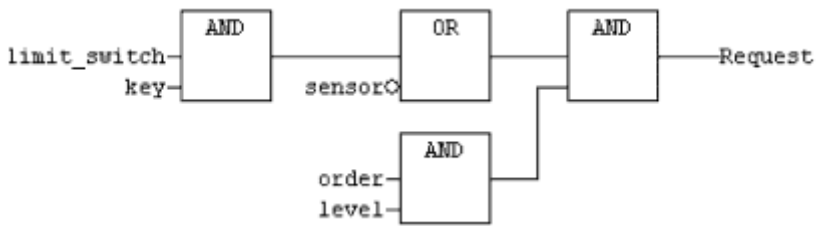
例:

演算子	オペランド	コメント
CASE Temperatur_furnace OF		(* heating出力の制御 *)
	60..99: Heating := 80;	(* 80% *)
	100..149: Heating := 60;	(* 60% *)
	150..199: Heating := 35;	(* 35% *)
	200..250: Heating := 10;	(* 10% *)
ELSE Alarm := TRUE;		(* アラームの設定 *)
END_CASE;		

ファンクションブロックダイアグラム(FBD)

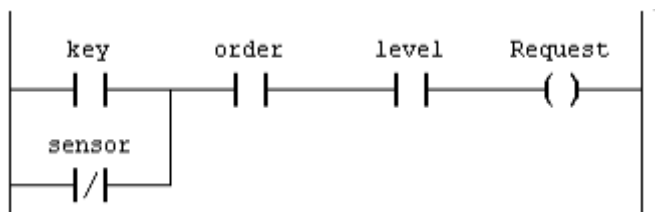
ファンクションブロックダイアグラムを用いたPLCプログラミングの背景にある基本的な考えは、ファンクションを重視した論理的な一連のカスケード(ネットワーク)にプログラムを構造化するというものです。1つのネットワーク内で、実行される方向は常に「左から右へ」です。ファンクションブロックの実行の前に、すべての入力値が常に生成されていなければなりません。すべての要素の出力値が計算されて初めて、ネットワークの評価が下されます。

例:



ラダーダイアグラム(LD)

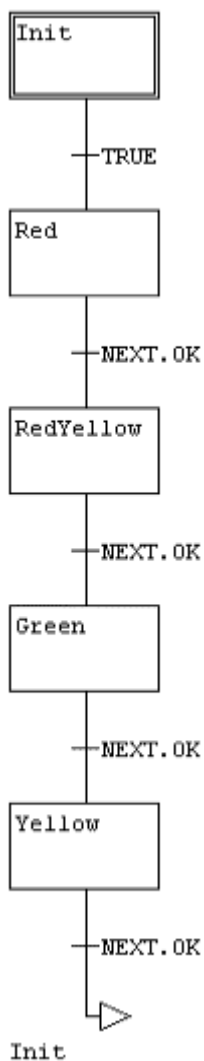
ラダーダイアグラムの論理シーケンスの表現は、電子技術プラントエンジニアリングの分野に由来します。この表現方法は、PLCプログラムでリレー切り替え処理を実装するのに特に適しています。処理はブール信号1および2に制限されています。



シーケンシャルファンクションチャート(SFC)

シーケンシャルファンクションチャートは、シーケンスのプログラミングが必要な場合に適しています。複雑なタスクが、プログラムの明確に配列された部分(ステップ)に分割されます。これらのステップ間の順序が図式的に定義されます。ステップ自体は、異なるプログラミング言語(ST、ILなど)で作成されるか、SFCで再び表現することもできます。

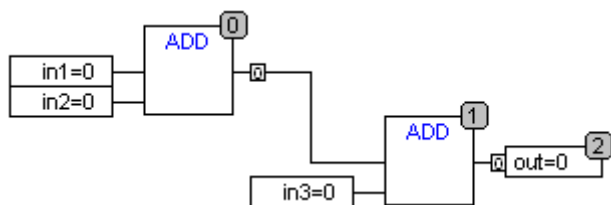
例:



SFCプログラムは基本的に、ステップ、トランジション、両者のリンクで構成されます。各ステップが一連のコマンドに割り当てられます。これらのコマンドは、ステップが有効なときに実行されます。次のステップが確実に実行されるためには、トランジションが実行する必要があります。ステップとトランジションは、任意の選ばれた言語で記述できます。

連続ファンクションチャートエディタ(CFC)

連続ファンクションチャートエディタは、ネットワークに関してファンクションブロックダイアグラム(FBD)の様には動作しませんが、自由に配置可能な要素を使用して動作します。これにより、例えばフィードバックが可能になります。連続ファンクションチャートエディタでのネットワークの典型的な例は以下のようになります。



5 概要



TwinCAT System Managerとは?

TwinCAT System Managerは、TwinCATシステムの構成のための中心的なツールです。関与するソフトウェアタスクの入出力と、接続されたフィールドバスの物理的入出力がTwinCAT System Managerによって管理されます。また、有効な構成のオンライン値を監視することもできます。

ソフトウェアタスクの変数とフィールドバスの変数を論理的にリンクすることによって、論理の入出力が物理の入出力に割り当てられます。

TwinCAT System Managerの構成モジュール

次に、TwinCAT System Managerの主なコンポーネントを挙げています。これらのコンポーネントの存在は、インストールされたTwinCATシステムのレベルによって左右されます。

Realtime Configuration

Realtime Configurationとユーザ定義タスクの作成

PLC Configuration

このエントリの下に、ローカルシステムで実行されているすべてのPLCプロジェクトがリストされます(現状では最大4つのプロジェクト)。

Cam Configuration

電子カムサーバとその構成

I/O Configuration

コントローラをプロセスレベルにリンクするために、対応するフィールドバスインターフェイスカードが必要です。どのカードを使用するかをこのエントリの下で定義する必要があります。

6 概要



Scope Viewとは?

TwinCAT Scope Viewは、さまざまなPLCタスクに関連する変数をグラフィック表示するための分析ツールです。

時間と対比して曲線を表示できますが、XY表示を選択することもできます。

各Scope Viewで複数のチャンネルを利用できます。チャンネル数はメモリサイズと処理能力によってのみ制限されます。時間表示のために、1つの変数が各チャンネルに割り当てられます。

Scope View分析

Scope View内の分析にいくつかのツールを利用できます。

データバックアップ

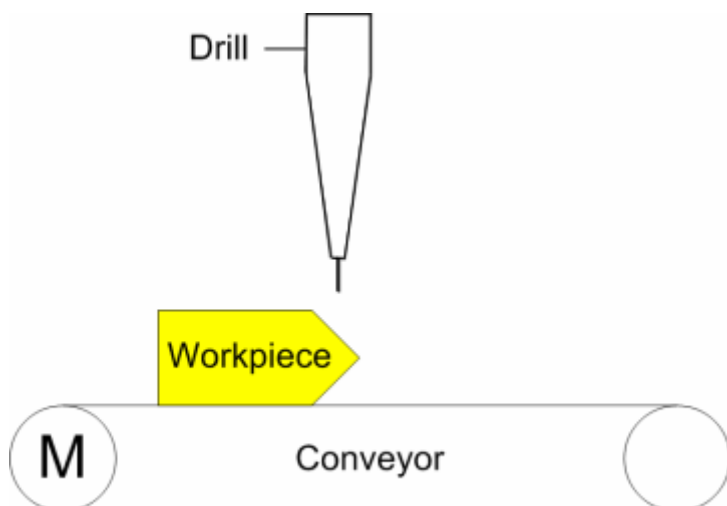
TwinCAT Scope Viewでは、複数のデータ形式(Excel表など)でデータを保存できます。

7 サンプルプログラム

7.1 サンプルMaschine.pro

TwinCATを用いてアプリケーションを作成する方法について、サンプルプログラムを参照しながら説明します。このプログラムは、任意の選ばれたワークピース向けの工作機械を表しています。TwinCATをインストールすると、このプログラムが「¥TwinCAT¥Samples¥First Steps」ディレクトリに配置されます。ファイル名は「Maschine.pro」です。

略図:

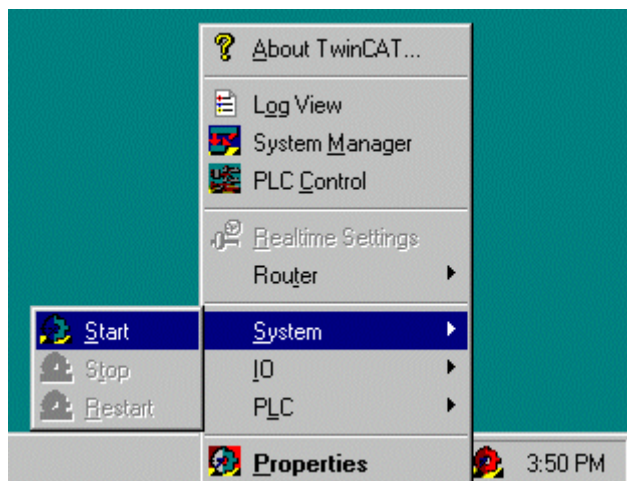


説明

- 1.) コンベヤベルトが25段階移動します。
- 2.) ドリルが2秒間下に移動します。
- 3.) ドリルが2秒間上に移動します。
- 4.) コンベヤベルトの25段階の移動が再び始まります。

TwinCATの起動:

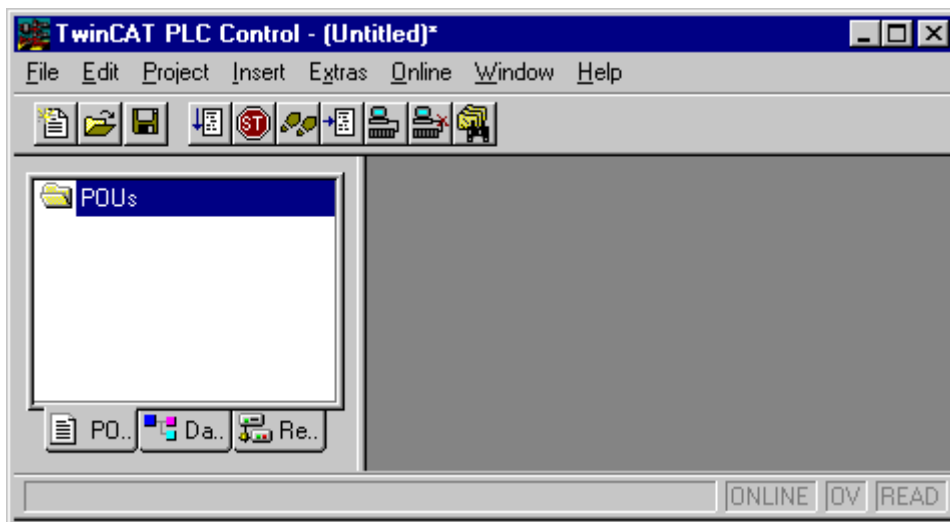
プログラムを実行する前に、TwinCATリアルタイムサーバを有効にする必要があります。



有効にするには、TwinCATリアルタイムサーバのアイコンをクリックし、[System]メニューから[Start]コマンドを実行します。アイコンの色が黄色から緑色に変わります。これは、TwinCATリアルタイムカーネルが有効であることを意味します。

TwinCAT PLC Control の起動

TwinCAT PLCからプログラミングを開始します。次に、[Start|Programs|TwinCAT system|TwinCAT PLC control]をクリックします。



プロジェクトを開く:



PLCプロジェクトは、ハードディスクまたはディスク上の、プロジェクトの名前が付けられたファイルに保存されています。プロジェクトを開くには、[File]メニュー項目を選択し、[Open]コマンドを選択します。

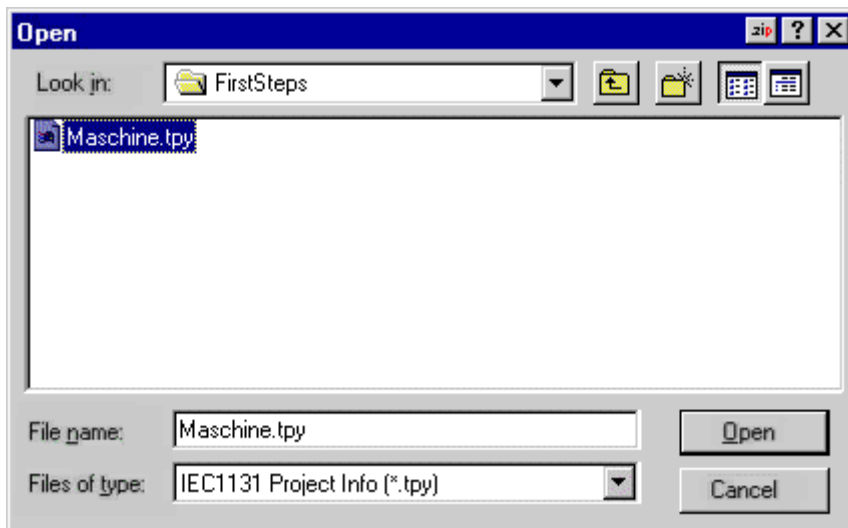
ディレクトリの選択:



ダイアログボックスの左に表示されているシンボルをクリックして、上記で指定したディレクトリに切り替えます。[Samples]エントリをダブルクリックします。次に、「First steps」を選択します。

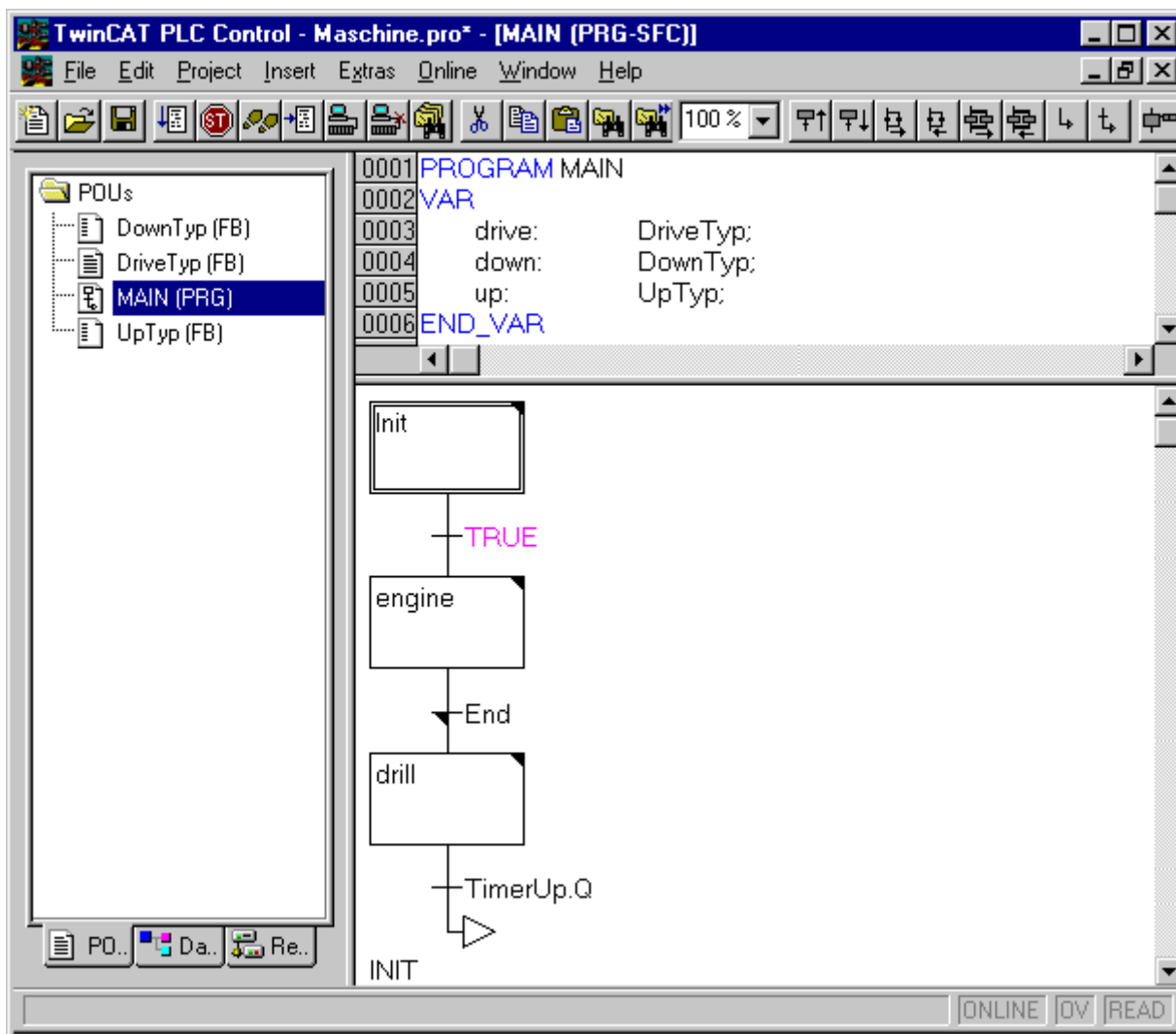
プロジェクトの選択:

「Machine.tpy」プロジェクトを選択します。選択するには、ダイアログボックスの該当するエントリをクリックし、[Open]コマンドを実行します。



PLC制御の項目:

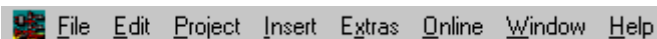
Maschine.proを開いた後、POU (プログラムオブジェクト) MAINを選択してダブルクリックします。次のダイアログボックスが開きます。



ダイアログボックスの一番上に、Maschine.proというプロジェクト名が青色で示されます。



その下に、コマンドメニューとツールバーがあります。

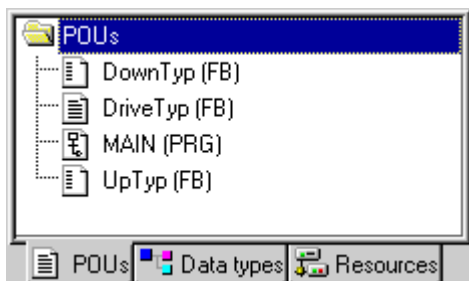


一番下部の灰色ライン部にステータスラインがあります。



このダイアログウィンドウは3つの個別ウィンドウに分かれています。オブジェクトリスト、変数宣言、プログラム表現です。

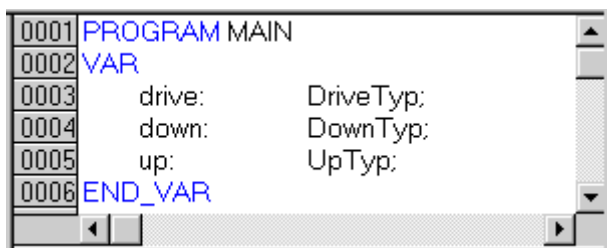
オブジェクトリスト:



TwinCATでは、1つのプロジェクトで3種類の基本オブジェクトが区別されます。

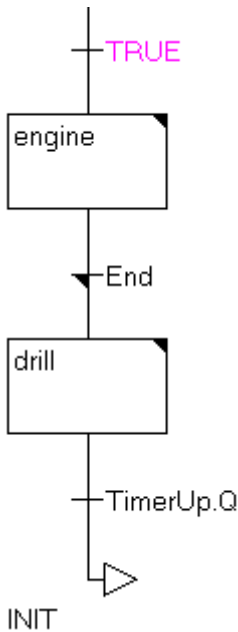
- ・ (プログラム)ブロック
- ・ データタイプ
- ・ リソース

変数宣言:



PLCプログラムでは、そのデータが変数で保存されます。変数はフラグワード（0か1かなど）またはデータワードに相当します。変数を使用する前に、変数を宣言する必要があります。つまり、変数が特定のデータタイプ(BYTEやREALなど)に属していることを明示する必要があります。宣言には、バッテリーによるバッファリング、初期値、物理アドレスへの所属などの特定の属性を定義することも含まれます。変数が入力イメージまたは出力イメージにおいて必要ない場合(つまり、PLCプログラム内でのみ必要な場合)、PLCプログラムがデータの格納先について心配する必要はありません。格納先はTwinCATによって確保されます。これにより、従来のシステムで起こり得たような、意図的でない見出し語/データワードの重複(副次的影響)が避けられます。変数と同様にファンクションブロックも宣言する必要があります(インスタンス)。このサンプルでは、3つのファンクション(「DriveType」、「DownType」、「UpType」)のそれぞれのインスタンス(drive、down、up)が1つ作成されます。インスタンスの作成後、インスタンスの使用と有効化が可能になります。

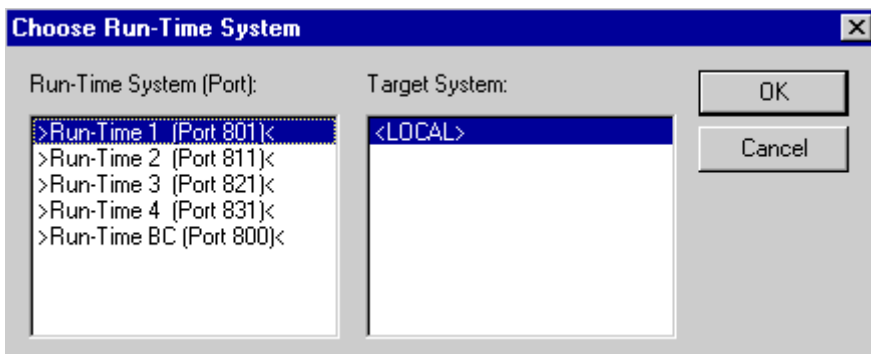
プログラム表現:



実際のPLCプログラムはTwinCAT PLC Controlの領域で入力され表示されます。

ターゲットシステムの選択

TwinCATは最大4つのランタイムシステムを提供します。これらの各ランタイムシステムは、他のランタイムシステムとは無関係にIEC 61131-3準拠のPLCプログラムを実行できます。[Online]メニューの[Choose run time system...]コマンドを使用して、どのランタイムシステムでプログラムを実行するかを定義します。



TwinCATのインストール後、1つのランタイムシステムのみがリリースされるため、その最初のランタイムシステム(Run Time 1)のみがこのダイアログに表示されます。[OK]を選択して、選択を確定します。

PLCランタイムがローカルコンピュータにない場合、最初に望ましいターゲットシステムへの接続を確立する必要があります(TwinCATドキュメンテーションを参照)。

ログイン:



PLCプログラムをTwinCAT PLC controlにロードしました。これで、PLCプログラムを実行することができます。TwinCATリアルタイムサーバが有効であることを確認してください。有効であれば、TwinCATリアルタイムサーバのアイコンが画面の右下に緑色で表示されます。PLCプログラムを起動する前にTwinCAT PLC controlをランタイムシステムにリンクする為、制御システムに「ログイン」する必要があります。

[Online]メニューの[ログイン]コマンドを実行します。ランタイムシステムにPLCプログラムがまだないため、「No program on the controller! Rebuild All?」というメッセージが表示されますので[OK]をクリックしてください。

接続の現在の状態がステータス行に表示されます。



PLCプログラムの起動:



TwinCATリアルタイムサーバでPLCプログラムを起動するには、[Online]メニューから[RUN]コマンドを選択します。ステータス行に表示される「RUN」という表記に色がかかります。シーケンシャルファンクションチャート(SFC)内の個々のステップが一時的に青色で表示されることも確認してください。

青色で表示されたステップは現在実行されています。つまり、有効なステップです。

PLCプログラムシーケンスのトレース:

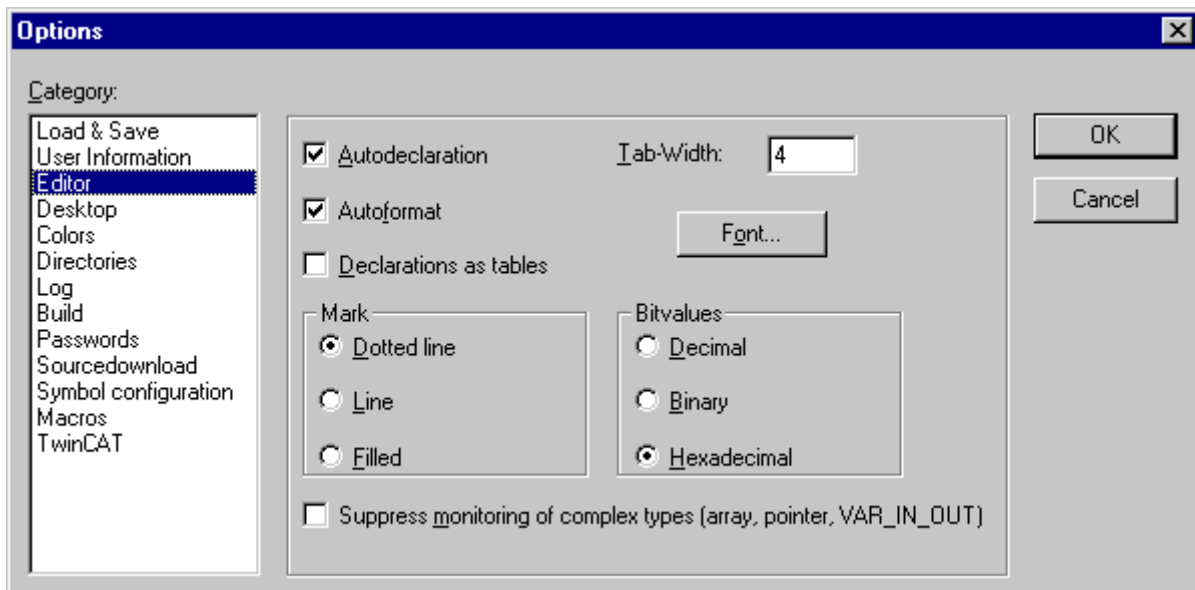
[Object List]ウィンドウの下部の[Resources]タブをクリックし、次に[Global variables]オブジェクトをダブルクリックすると、[Global variables]ウィンドウが有効になります。このウィンドウに、グローバル宣言された変数がすべて表示されます。グローバル変数は、すべてのプログラムオブジェクト(POU)で使用できます。

0001	engine(%QX0.0) = FALSE
0002	deviceUp(%QX0.1) = FALSE
0003	deviceDown(%QX0.2) = FALSE
0004	timerUp
0005	timerDown
0006	steps = 16#0D
0007	count = 16#004A
0008	devSpeed = T#10ms
0009	devTimer
0010	.M = TRUE
0011	.StartTime = T#6m50s754ms
0012	.IN = TRUE
0013	.PT = T#10ms
0014	.Q = FALSE
0015	.ET = T#0ms
0016	switch = FALSE

これらの変数の他に、ファンクションブロック(「timerUp」、「timerDown」、「devTimer」)もここに表示されます。ファンクション名の前に四角(□)が表示されます。四角(□)をダブルクリックすると、ツリー状のディスプレイが開き、そこにファンクションの変数がすべて表示されます。

数値表示(進数)の変更

変数の内容をさまざまな記数法で表示できます。10進数、16進数、2進数から選択できます。表示を変更する場合は、[Project]メニューの[Options]コマンドを選択する必要があります。現在の設定が、対応する項目の前にチェックマークで示されます。



プログラムの終了:



PLCプログラムをTwinCAT PLC control(IEC 61131-3プログラミング環境)にロードし、TwinCAT PLCサーバ(ランタイムシステム)上で実行しました。この時点でPLCプログラムを終了します。終了するには、[Online]メニューの[Stop]コマンドを選択します。

ログアウト:



次のセクションで、PLCプログラムへの追加を行います。そのために、TwinCAT PLCサーバからログアウトする必要があります。ログアウトするには、[Online]メニューの[Log out]コマンドを実行します。

プログラムテキストの表示:

このサンプルは、さまざまなIEC 61131-3プログラミング言語でプログラムされています。プログラムの主要部分は、シーケンシャルファンクションチャート(SFC)で作成されています。主要部分には、以下のステップとトランジションが含まれています。ステップ:

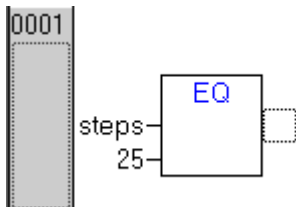
- ・ Init
- ・ Engine
- ・ Drill

トランジション (変遷) :

- ・ TRUE
- ・ End
- ・ TimerUp.Q

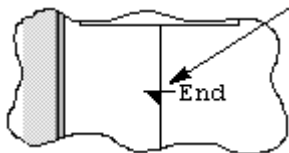
トランジションの表示:

「TRUE」トランジションは継続的に「Yes」としてプログラムは以後のステップに進みます。「TRUE」キーワードがシステム全体の定数であり常に実行されるため。よって「Engine」ステップは「INIT」ステップの後に無条件で実行されます。「TimerUp.Q」トランジションは、Upファンクションの変数QがTRUE (または1)でなければ次のステップに進めないことを意味します。「End」は、他のプログラムテキストを含むトランジションです。このトランジションをクリックすると、対応するプログラムテキストが表示されるウィンドウが開きます。



「End」トランジションで、モータのステップが25に到達しているか比較が行われます。到達している場合、プログラムが次のサイクルで「engine」ステップから「drill」ステップに変わります。

ステップまたはトランジションに他のプログラムテキストが含まれている場合、その旨が小さい黒色の三角形によって示されます。



PLCプログラムの変更

[MAIN]ウィンドウに戻ります。変数を用いてモータのサイクル速度を2段階(速い/遅い)で切り替えることができるように、この時点でPLCプログラムを変更する必要があります。オブジェクトリストの[Drive Type]ファンクションブロックをダブルクリックして開きます。入力カーソルを最初の行に移動し、以下のテキストを入力します。

```
IF switch = TRUE THEN
```

Enter (Return)キーを押すと、ダイアログボックスが表示されます。そのダイアログボックスで以下のように入力する必要があります。

The screenshot shows a dialog box titled 'Declare Variable'. It has several fields and buttons:

- Class:** VAR_GLOBAL (dropdown)
- Name:** switch (text input)
- Type:** BOOL (dropdown)
- Symbol list:** Globale_Variablen (dropdown)
- Initial Value:** (empty text input)
- Address:** (empty text input)
- Comment:** (empty text input)
- Buttons:** OK, Cancel, CONSTANT (checkbox), RETAIN (checkbox)

[OK]を選択すると、「switch」変数が[MAIN]の変数リストに追加されます。このダイアログボックスが開かない場合、自動宣言が無効になっています。(有効にするには、[Project]メニューの[Options|Editor]で[自動宣言]を選択します)。メニューの[Edit]にて[Auto declare]を開きます。

以下のプログラムも入力します。

```
devSpeed := T#10ms;
ELSE
devSpeed := T#25ms;
END_IF
```

ウィンドウの内容が以下のようであればなりません。

```
0001 IF switch = TRUE THEN
0002     devSpeed:=T#10ms;
0003 ELSE
0004     devSpeed:=T#25ms;
0005 END_IF
0006
0007 IF devTimer.Q THEN
0008     devTimer ( IN := FALSE, PT := devSpeed);
0009     engine := NOT engine;
0010     IF engine = FALSE THEN
0011         steps := steps + 1;
0012     END_IF
0013 ELSE
0014     devTimer ( IN := TRUE, PT := devSpeed);
0015 END_IF
0016
```

「switch」変数が設定されていると、「devSpeed」変数が25 msにセットされます(そうでない場合は、10 msにセットされます)。この結果、以下のプログラム行のクロックパルスジェネレータのパルス/ポーズ持続期間が合計で25 msまたは10 msになります。

プログラムの保存:



[File]メニューの[Save]コマンドを選択して、プログラムを保存します。

プログラムのコンパイル:

プログラムをTwinCAT PLCサーバに転送する前に、プログラムをコンパイルする必要があります。つまり、プログラムをテキスト/グラフィック表現から、制御システムが認識できる形式に変換する必要があります。コンパイルするには、[Project]メニュー項目から[Rebuild all]コマンドを選択します。

プログラムの起動:

制御システムにログインし、PLCプログラムを起動します。起動時に「switch」変数が「FALSE」に設定されていることが分かります。

変数値の変更:

PLCプログラムの実行中に変数の値を変更することが可能です。[Global variables]ウィンドウを開き、[switch]エントリをダブルクリックします。表示がFALSEからTRUEに変わり、文字が赤色になります。ただし、この時点までTwinCAT PLCサーバにおける値は変わっていません。変更するには、[Online]メニュー項目の[Write values]コマンドを実行する必要があります。文字が再び黒色になり、「devSpeed」変数が10 msに変わります。

プログラムシーケンスのトレース:

TwinCATスコープビューを使用してプログラムフローを追跡します。

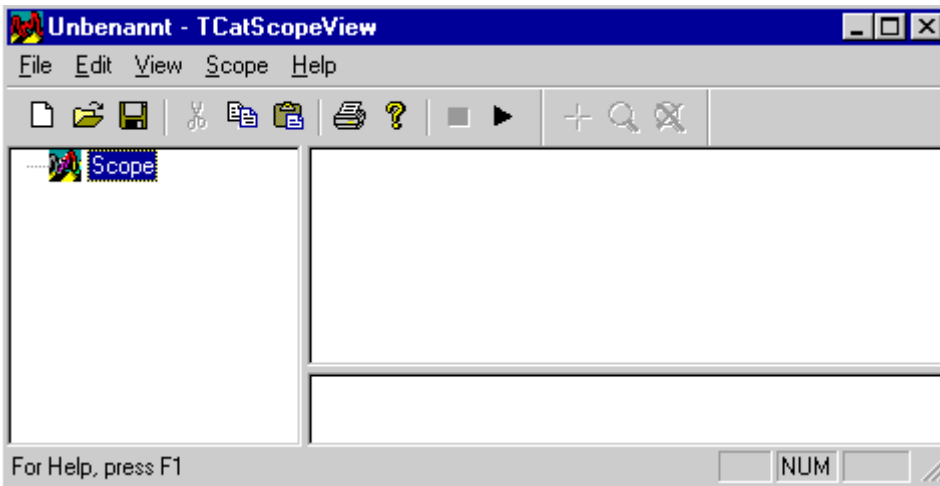
7.2 プログラムフローの追跡

TwinCATスコープビューは、プログラムの記録と分析に使用します。

TwinCATスコープビューを開く:

スコープビューは、スタートメニューからのみ開くことができます。

[Start|Programs|TwinCAT System|TwinCAT Scope View]を選択します。



TwinCATスコープビューの要素:

TwinCATスコープビューのウィンドウはTwinCAT PLCのウィンドウに似ています。最初の行がプロジェクトの名称で、その下にコマンドラインとツールバーがあります。3つの大きなウィンドウは空です。左のウィンドウでスコープを設定できます。

TwinCATスコープビューの起動:

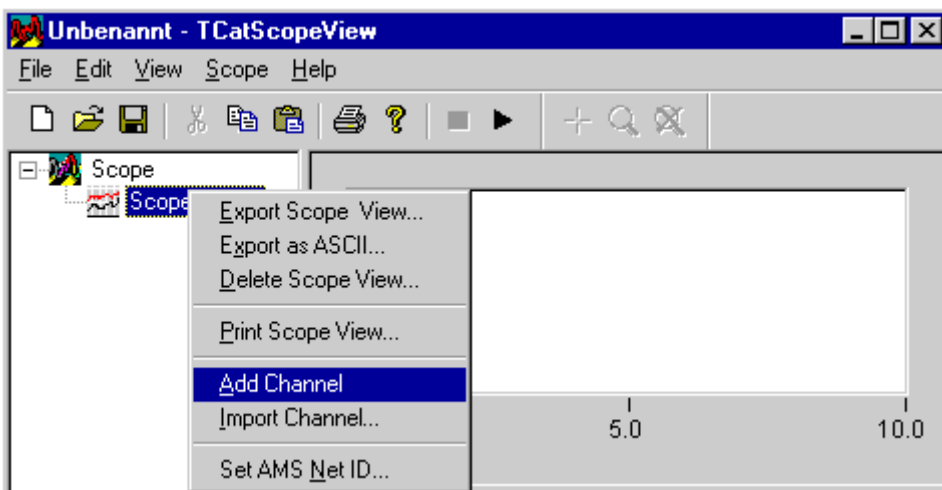
サンプルプログラムMaschine.proを起動するために、まずスコープビュー(つまりプロジェクト)を追加する必要があります。追加するには、[Scope]を右クリックし、[Add Scope View]を選択します。[OK]で選択を確定します。



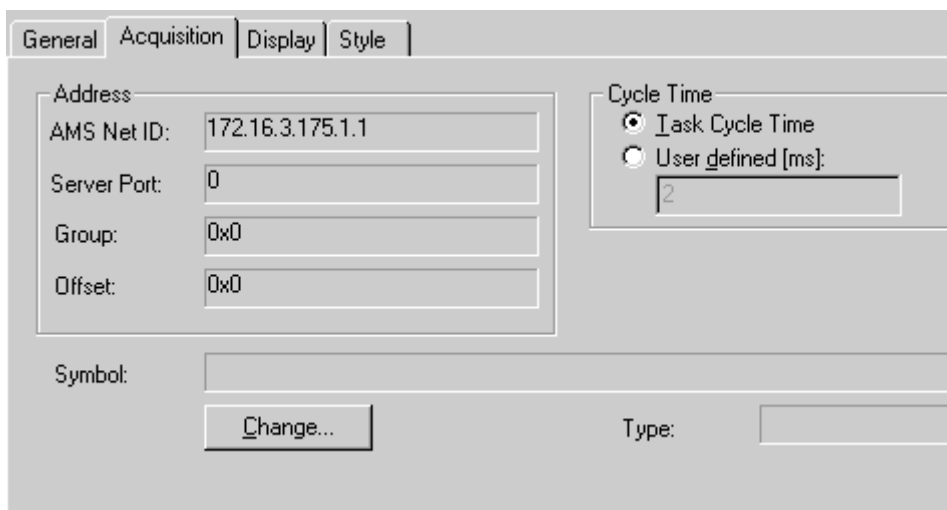
チャンネルの挿入:

個々の信号を表示するために、関連するチャンネルを作成する必要があります。

作成するには、[Scope View 1]を右クリックし、[Add channel]を選択します。[OK]で選択を確定します。



値が記録される変数が記述されている以下のタブが表示されます。



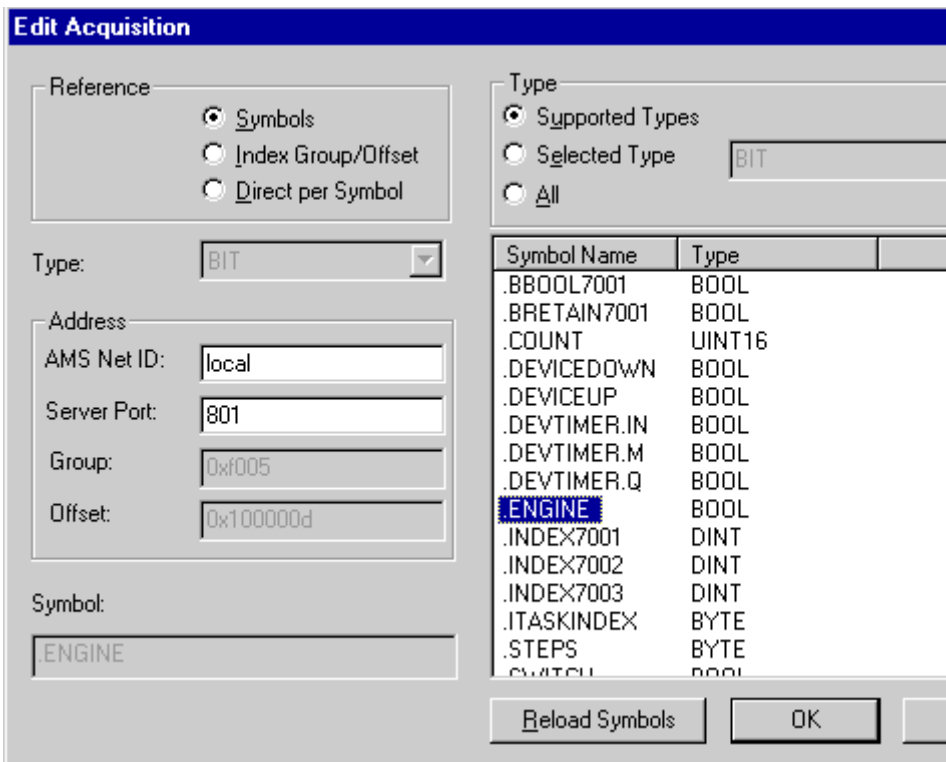
ユーザ定義のサンプリング時間を10 msに設定すれば、表示が明確になります。

サーバポートのセットアップ:

次に、サーバポートをセットアップします。Acquisitionタブ下の[Change]をクリックし、番号を入力します。[OK]で入力を確認します。(PLC制御からサーバポートを検索できます。「ターゲットシステムの選択」[▶ 30]を参照してください。)

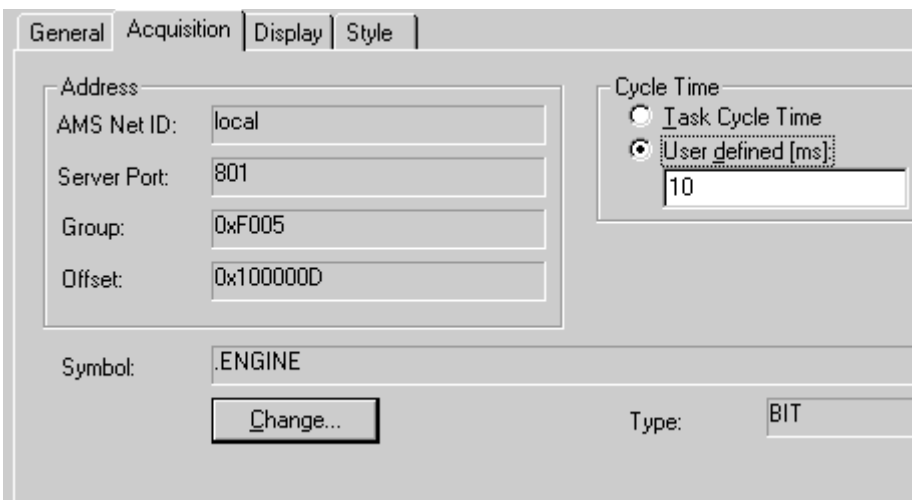
チャンネルの割り当て:

信号.ENGINEをChannel 1に割り当てます。割り当てるには、[Change]を再度クリックし、[OK]で確定します。



設定

タブに以下の設定が表示されます。



チャンネル名の変更:

[Channel1]をゆっくりダブルクリックすると、チャンネル名をENGINEに変更できます。

チャンネルの追加:

同じ方法で他のチャンネルを割り当てることができます。

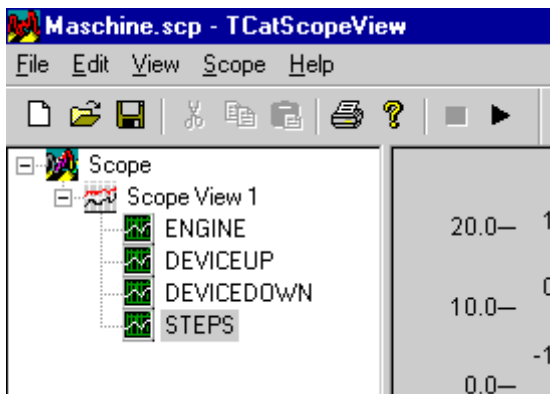
Channel2=.DEVICEUP

Channel3=.DEVICEDOWN

Channel4=.STEPS

サーバポートとサンプリング時間は、すべてのチャンネルで同じままです。

4つのチャンネルを作成し、それらの名前を変更したら、スコープビューを保存します。保存するには、[File]メニューで[Save as]を選択し、名前としてMaschine.scpを入力します。



さまざまな曲線を互いに区別するために、各チャンネルの色/形/軸を異なるものにすることができます。これは、[Style]タブまたは[Display]タブを使用して行います。

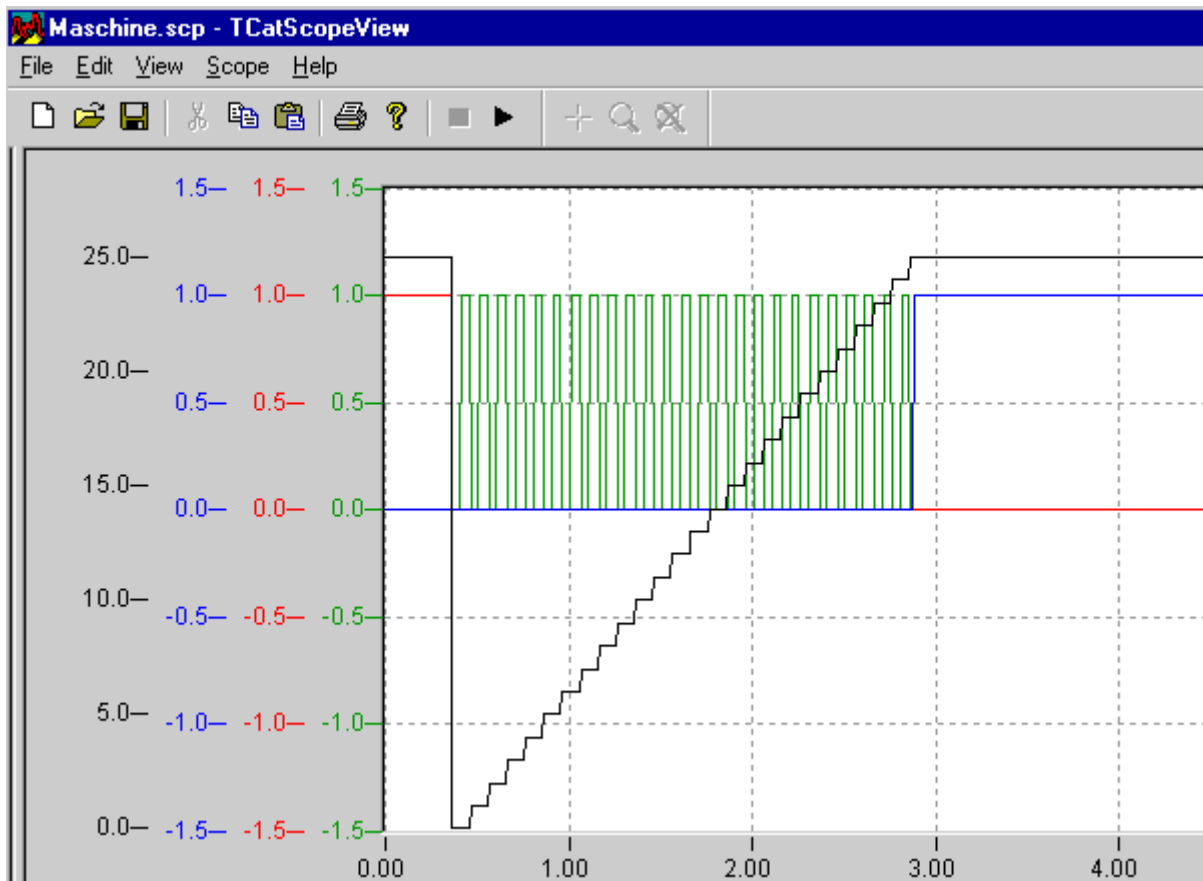
記録の開始:



記録を開始するには、メニュー[Scope]から[Start Scope]を選択します。

Maschine.pro

サンプルプログラムが次のようになります。



7.3 サンプルプログラムの変換

7.3.1 変数の宣言

この章では、前述のPLCプログラムをベッコフバスターミナルに接続します。これは、TwinCATシステムマネージャを使用して行います。システムマネージャを使用して、すべての入出力インターフェイス接続を管理し、アドレス指定し、I/Oデータに割り当てます。各I/Oチャンネルを論理名によってアドレス指定することができます。TwinCATシステムマネージャは、1つの共通プロセスイメージで複数のフィールドバスを管理します。

必要なハードウェア

必要ハードウェアを用意する必要があります。Lightbusのデモキットに、II/O-Lightbus (FC2001)用のPCインターフェイスカード、バスカプラBK 2000、バスターミナル、2つの光ファイバ、いくつかのドキュメンテーションが含まれています。

ハードウェアがなければ、次の章：アプリケーションサンプルに進みます。

変数宣言:

変数の格納先(アドレス)はシステムによって内部で管理されます。プログラムはメモリ管理についての配慮は不要です。PLCプログラムはシンボリックな変数名で動作し、変数使用時に副作用(重複)が発生するのを防ぎます。入出力レベルにアクセスするには、プログラムが個々の変数に固定アドレスを割り当てる必要があります。これは、変数を宣言するときに常に指定されなければならないキーワード 'AT'によって実現されます。キーワード「AT」の後には、データの場所(入力/出力またはフラグエリア)とデータの幅(BIT、BYTE、WORDまたはDWORD)に関する情報を提供するいくつかのパラメータが続きます。上記の例の変数宣言には以下の構造があります。

```
VAR_Global
engine      AT %QX0.0:    BOOL;
deviceup    AT %QX0.1:    BOOL;
devicedown  AT %QX0.1:    BOOL;
timerup:    TON;
timerdown:  TON;
steps:      BYTE;
count:      UINT := 0;
devspeed:   TIME := t#10ms;
devtimer:   TP;
timerup:    TON;
switch:     BOOL;
END_VAR
```

変数アドレスの記載

	データの場所	データ幅	意味
%			I/O定義の開始
	I		入力
	Q		出力
	M		フラグ
		X	ビット(1ビット)
		B	バイト(8ビット)
		W	ワード(16ビット)
		D	ダブルワード(32ビット)

Datawidth (データ幅) 後の数字で変数のアドレスを指定します。ビット変数の場合、アドレスをx.yという形式で指定する必要があります。バイト/ワード/ダブルワード変数の場合は単にxという形式で指定します。そのため、異なるストレージ領域にある入出力に、同じアドレスが含まれることがあります。

7.3.2 Lightbus - バスターミナルのセットアップ

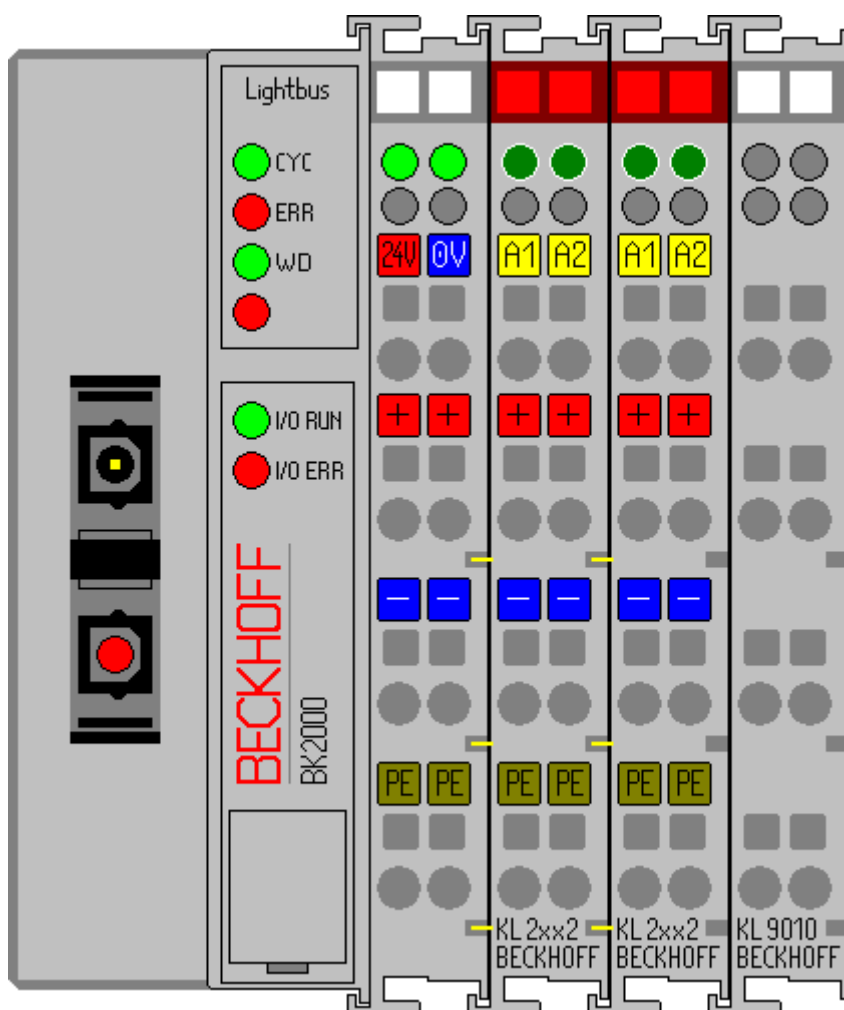
ハードウェア要件:

- PCインターフェイスカード: FC2001
- バスカプラ: BK2000
- 3つのデジタル出力: 24V:2x KL2032
- バスエンドターミナル: KL9010

この例は、ベッコフLightbusデモキット(TC9910-B200-0100)の内容に最適な要件です。ただし、ハードウェアは交換可能です。交換する場合は、I/Oデバイスの構成を変更する必要があります。

ターミナルのセットアップ

次の図で示したように、バスカプラとバスターミナルをセットアップしてください。



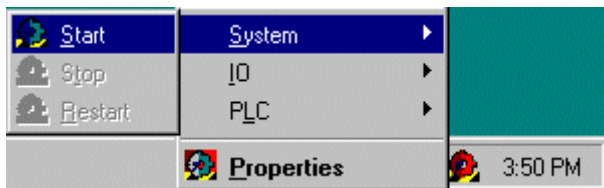
バスカプラをPCインターフェイスに接続し、バスカプラに24 V DCの電源を投入してください。

ハードウェアドキュメンテーション

ハードウェア接続に関する詳細情報がデモキットのハードウェアドキュメンテーションに記載されています。

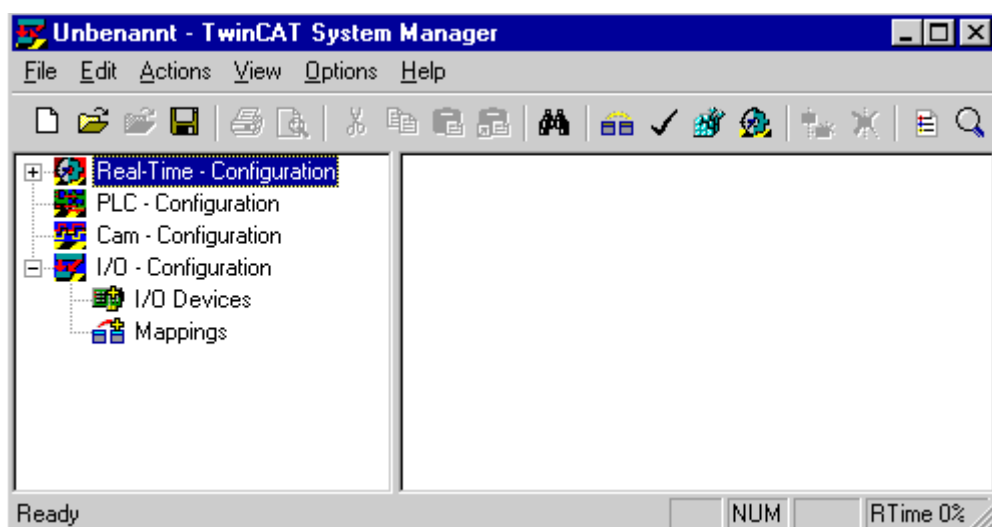
TwinCATリアルタイムサーバの起動:

この時点でまだ起動していなければTwinCATリアルタイムサーバを起動してください。その後、TwinCATメッセージルータが有効になります。



TwinCATシステムマネージャの起動

システムが起動していれば、アイコンの色が赤から緑に変わります。この時点でTwinCATシステムマネージャを起動してください。起動するには、[Start|Programs|TwinCAT System|TwinCAT System Manager]を選択します。



TwinCAT System Managerの項目:

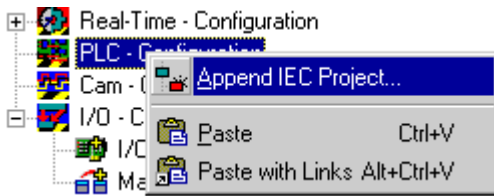
最初の行はプロジェクトの名前(ここでは「unbenannt」)で、その下にコマンドライン(メニュー)とツールバーがあります。最後の行にはシステムの状態が表示されます。この例では、システムが実行中(RTime)です。中央の2つのウィンドウにシステムの構成が表示されます。次の手順でシステムを構成します。

システム構成がシステムマネージャの左側にツリー構造として表示されます。ツリー構造は以下の4つの主要項目から成ります。

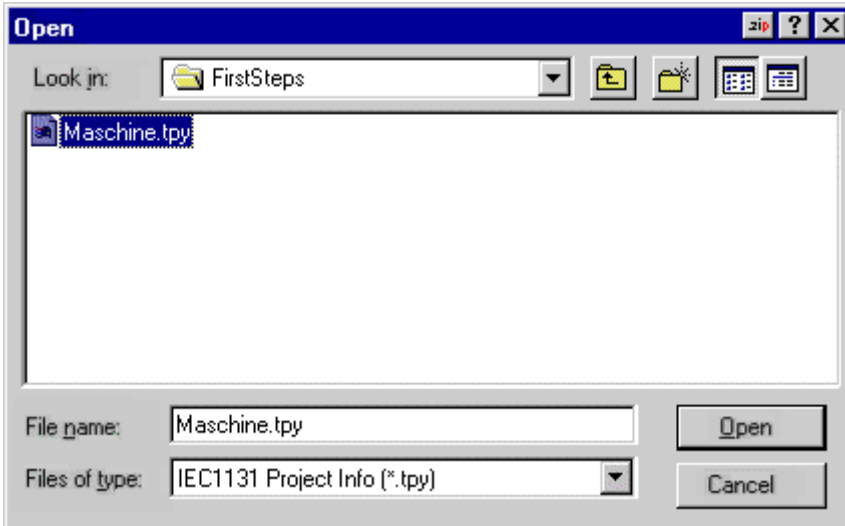
構成	意味
Realtime	リアルタイムパラメータを設定します。
PLC	構成する必要があるすべてのPLCプロジェクト
Cam	カムサーバを追加します。
I/O	コントローラをプロセスレベルにリンクするために、システムにインターフェイスが必要です。このエントリにてすべてのインターフェイスのリストが示されます。

PLC Configuration:

個々のPLCプロジェクトをシステムマネージャに認識させる必要があります。そうすれば、TwinCATがPLCプログラムの変数にアクセスできます。認識させるには、マウスポインタを[PLC configuration]の上に置いて右クリックします。

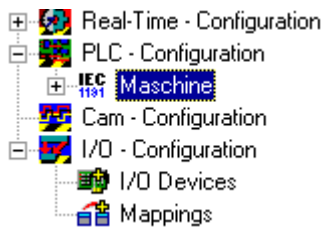


コンテキストメニューが開きます。そのメニューで[Append IEC project...]を選択します。

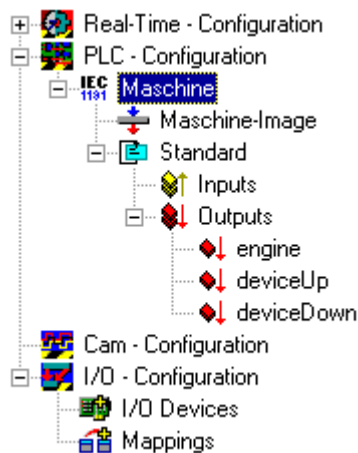


「¥TwinCAT¥Samples¥FirstSteps¥」ディレクトリに切り替えて「maschine.tpy」ファイルを選択します。

PLCプロジェクトの名前が付けられたエントリが[PLC configuration]の下に追加されています。

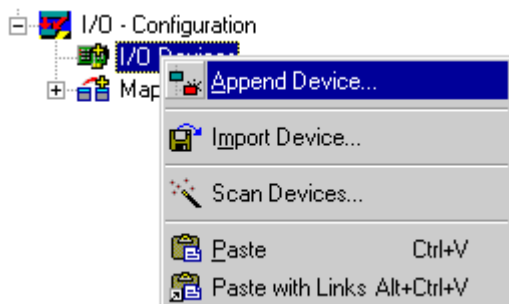


+記号と-記号は、エントリに他の下位エントリが含まれているかどうかを示します。これらの記号をクリックすると、その下のエントリを開いたり閉じたりすることができます。できるだけ深くツリーを開くと、以下の構造が表示されます。



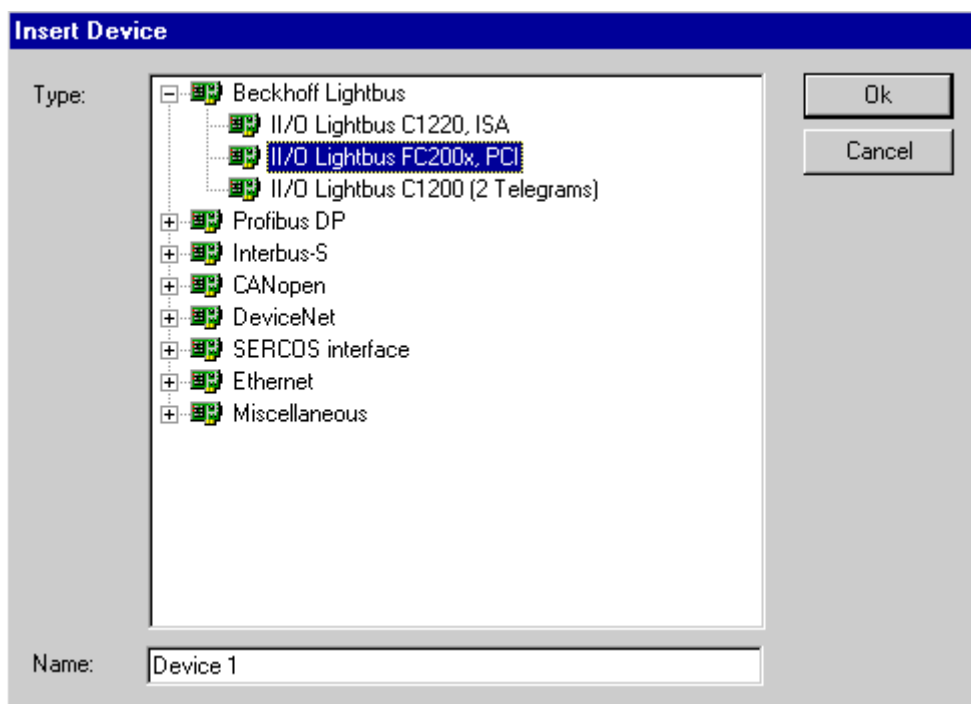
I/O Configuration

PLCプロジェクトがPLC構成に追加され、プロセスイメージの変数がすべて認識されている場合、I/O構成を指定する必要があります。[I/O devices]エントリを右クリックして選択します。コンテキストメニューが開きます。そのメニューで[Add device]エントリを選択する必要があります。



I/Oデバイスの選択:

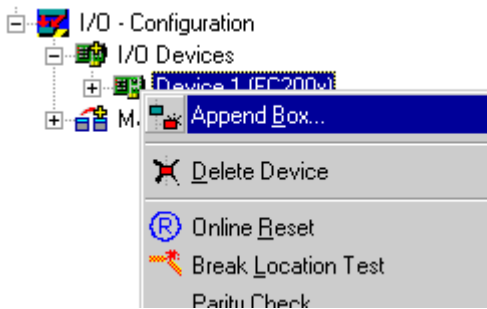
次のウィンドウが開きます。デバイスタイプを選択します(この場合は[I/O lightbus FC200x, PCI])。デバイス名は自由に選ぶことができます。



右側にインターフェイスカードの設定を指定できるダイアログボックスが開きます。例えば、'FC2001'スライダの下で重要な設定の1つはライトバスカードのI/Oアドレスです。カードのデフォルト設定を変更していない場合は指定したエントリを使用できます。

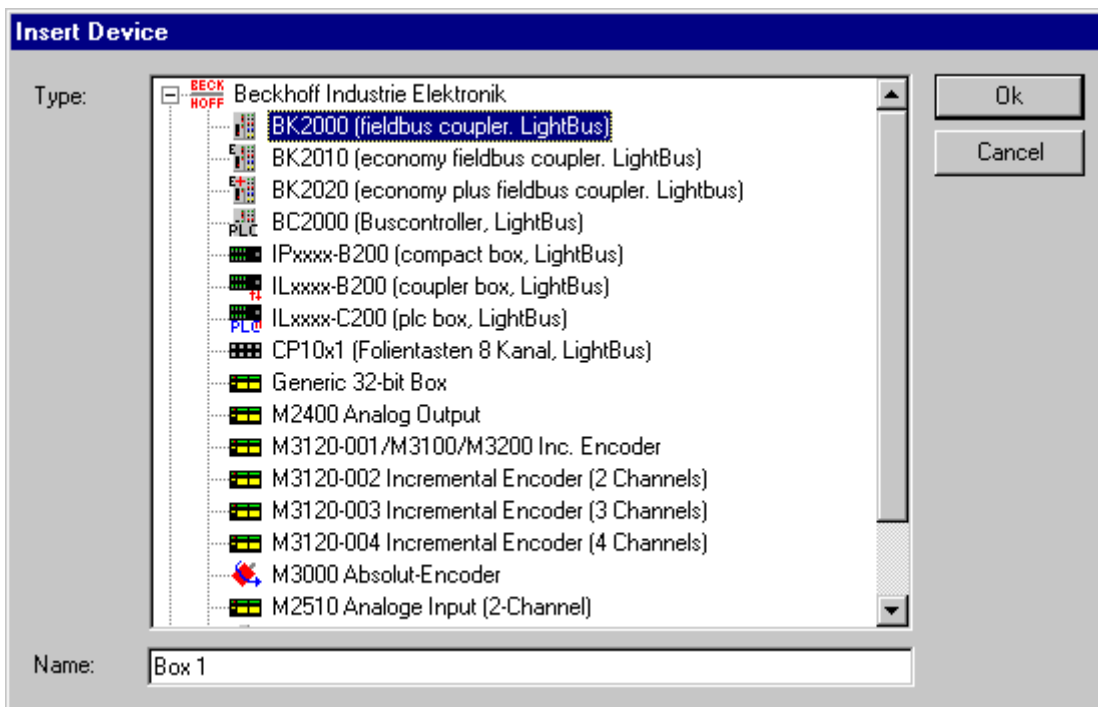
バスカプラ追加

FC2001カード(Device 1)のコンテキストメニューを開き、[Add box...]コマンドを選択します。



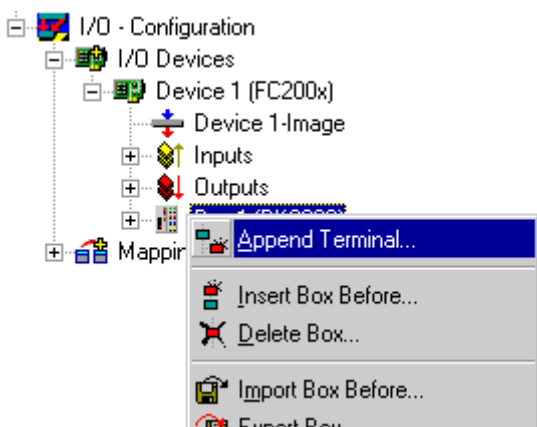
バスカプラの選択:

フィールドバスモジュールタイプを選択します(この場合は[BK2000])。フィールドバスモジュール名は自由に選ぶことができます。



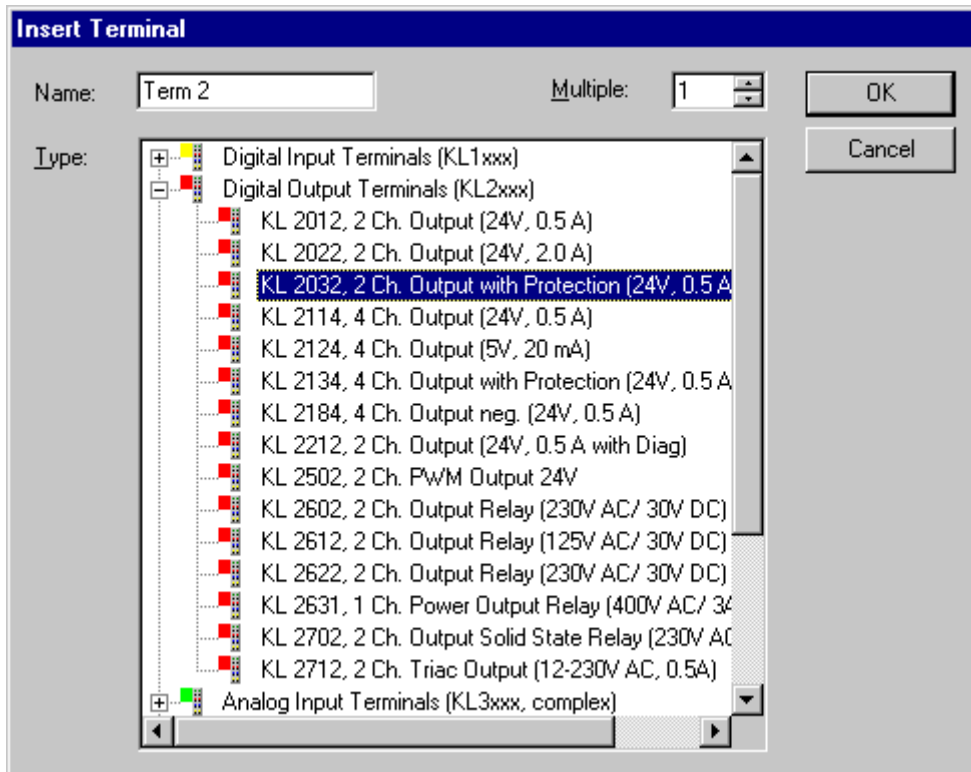
バスターミナルの追加:

BK2000 (Box 1)のコンテキストメニューを開き(右クリック)、[Add terminal...]コマンドを選択します。



バスターミナルの選択:

KL2032という名前のターミナルを選択します。+記号と-記号をクリックすると、詳細なターミナル選択ができます。



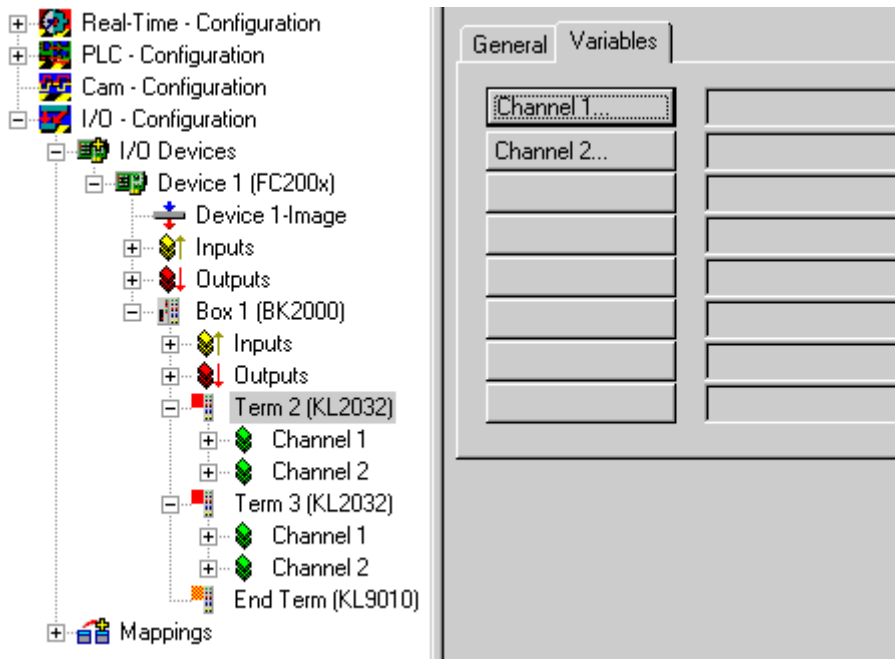
バスターミナル名は自由に選ぶことができます。

サンプルプログラムでは3つのデジタル出力が必要です。2番目のバスターミナルを挿入するには、上記手順を繰り返します。

バスエンドターミナルKL9010がシステムマネージャによって自動的に挿入されます。

構成の終了:

構成の概要は以下のとおりです。



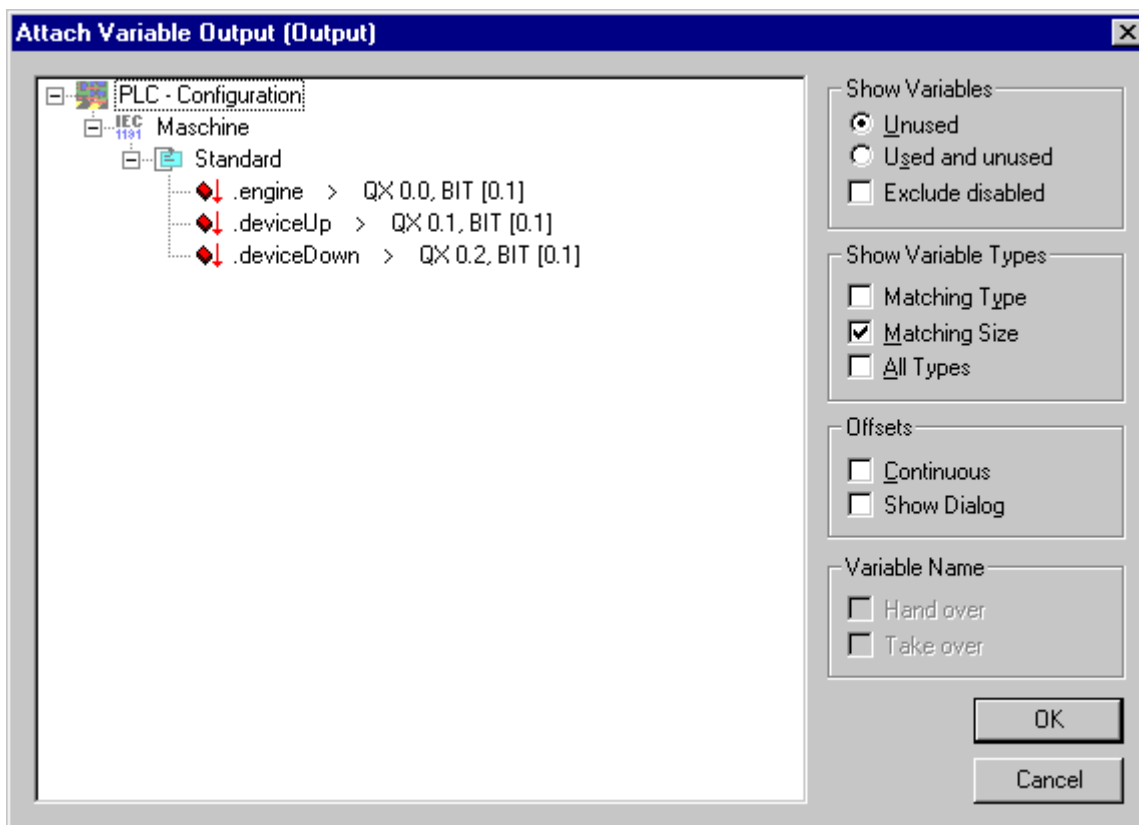
補足として、デフォルトで付く標準名称(device 1、box 1、terminal 1など)は変更できます。変更は対応する名前をゆっくりとダブルクリックし、新しい名前を入力します。

入出力チャンネルへの変数の割り当て:

この時点までに、上記のサンプルプログラムに完全必要なハードウェアが構成されています。次に、PLCプロジェクトからの個々の変数を個々の入出力チャンネルに割り当てる必要があります。

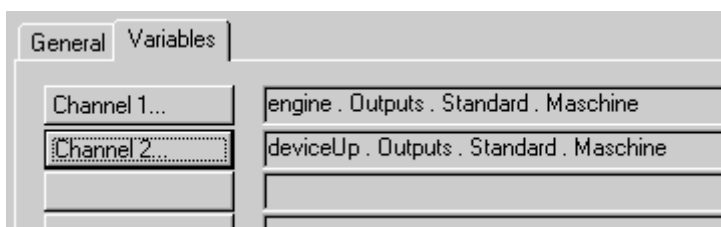
割り当てるには、構成するターミナルをマークします。terminal 1 (4つのデジタル出力)の場合、[General]と[Variables]というタブを含むダイアログボックスが右側に開きます。[Variables]タブを選択します。

リスト上に2つの出力チャンネルの表示されますが、2つのチャンネルはフリーです。channel 1を設定するには、対応する([channel 1...])ボタンを選択して次のダイアログボックスが開きます。次のダイアログボックスが開きます。



すべての出力変数がこのダイアログボックスにリストされています。1番目の変数(engine)を選択し、[OK]をクリックしてエントリを確定します。2番目の出力変数を同様に処理します。

1番目のバスターミナルが接続されました。



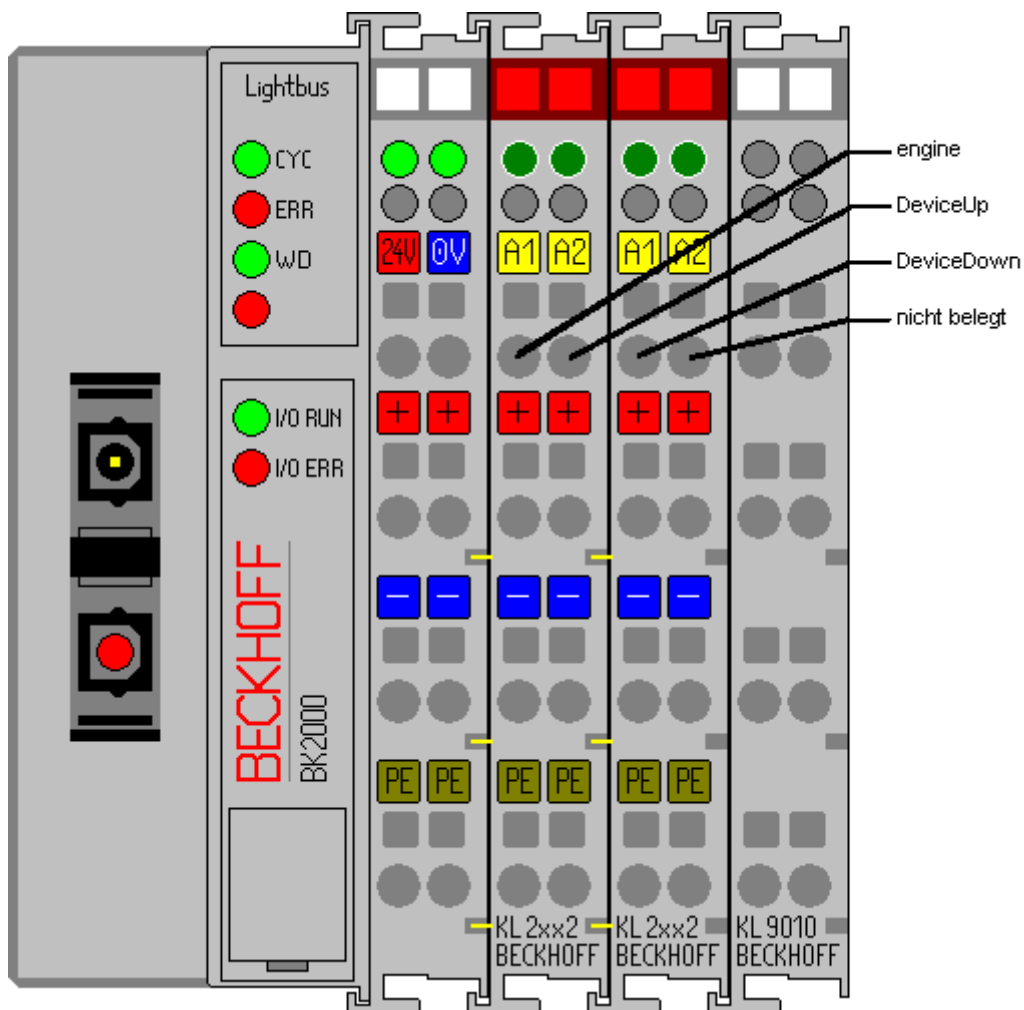
他のバスターミナルを同様に処理します。

変数の割り当て

ターミナル2	PLC変数	意味
チャンネル1 (=出力1)	engine	ステッピングモータ制御
チャンネル2 (=出力2)	device.Up	ドリルアップ制御

ターミナル3	PLC変数	意味
チャンネル1 (=出力3)	device.Down	ドリルダウン制御
チャンネル2 (=出力4)	-	空きチャンネル

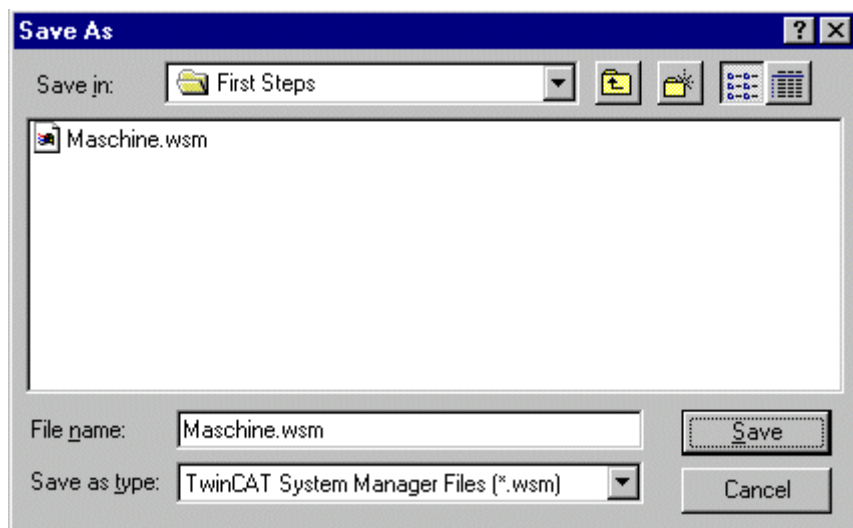
バスターミナルの割り当て:



プロジェクトの保存:



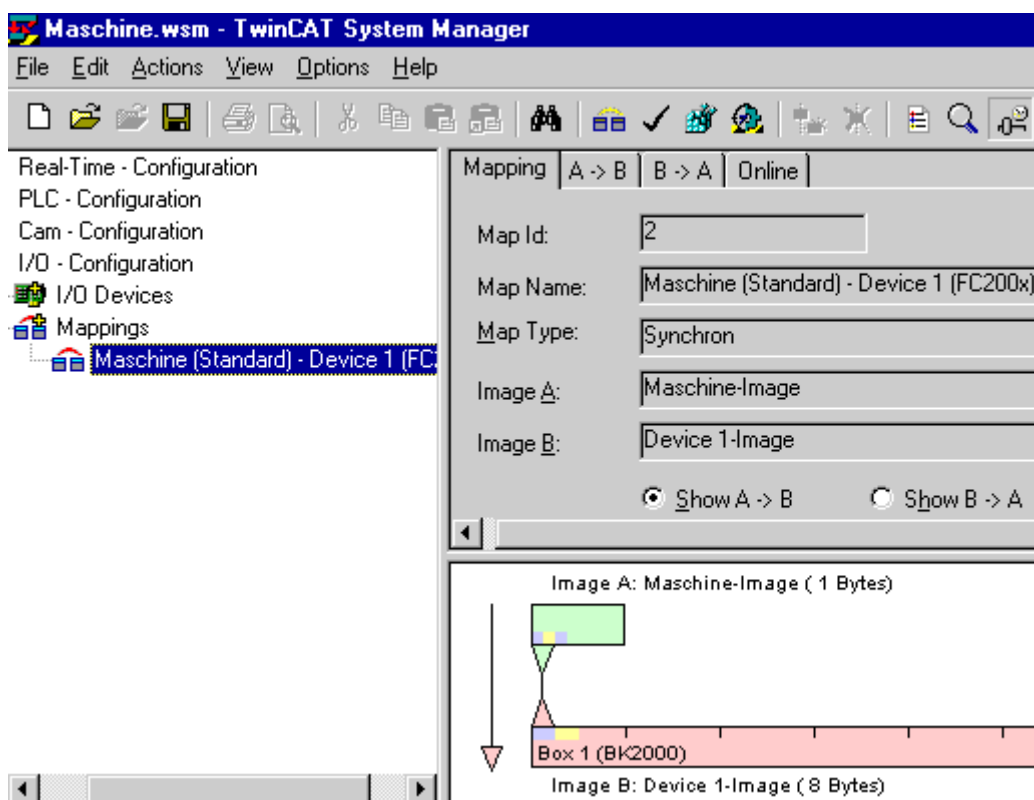
この時点で構成を保存しておくこと、後で保存した構成に確実にアクセスできます。保存するには、[File]メニューから[Save as...]コマンドを実行します。



変数のマッピング:



上記のサンプルプログラム用に完全なシステムを構成しました。この時点で割り当てを作成する必要があります。作成するには、[Actions]メニューの[Create allocation]コマンドを実行します。[Allocations]ツリーエントリの下に、[Standard device 1(FC2001)]が表示されます。このエントリをクリックします。次のウィンドウが開きます。



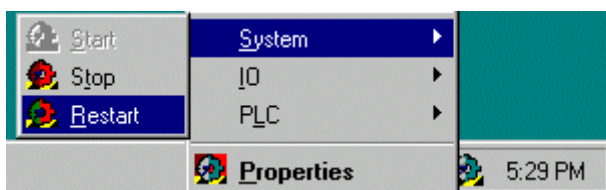
ダイアログボックスで、「AからBへ」または「BからAへ」のどちらのデータフローを表示するのかを定義できます。このサンプルでは、イメージAがPLC変数(入出力変数)のプロセスイメージに対応しています。イメージBがI/Oデバイス(バスカプラBK2000)のプロセスイメージに対応しています。それぞれの変数またはバスターミナルは、色付きプロセスイメージで表示されています。いずれかのエリア上でマウスを停止すると、小さな表示ボックスが開き、正確な名前が表示されます。

レジストリへの構成の書き込み:

最後の手順として、構成をWindows NTレジストリに保存する必要があります(理由として、保存された情報がTwinCATの起動時に評価されるため)。[Actions]メニューから[Save in registry...]コマンドを実行します。古い構成がレジストリに既に保存されている場合、安全確認表示されますので確定する必要があります。

TwinCATの再起動:

システムを再起動する必要があります。これを行うと、TwinCATが変更を受け入れることができます。



個々のPLC変数がバスターミナルKL 2032で出力されます。バスターミナルで、LEDによって信号状態が示されます。

7.3.3 イーサネット - バスターミナルのセットアップ

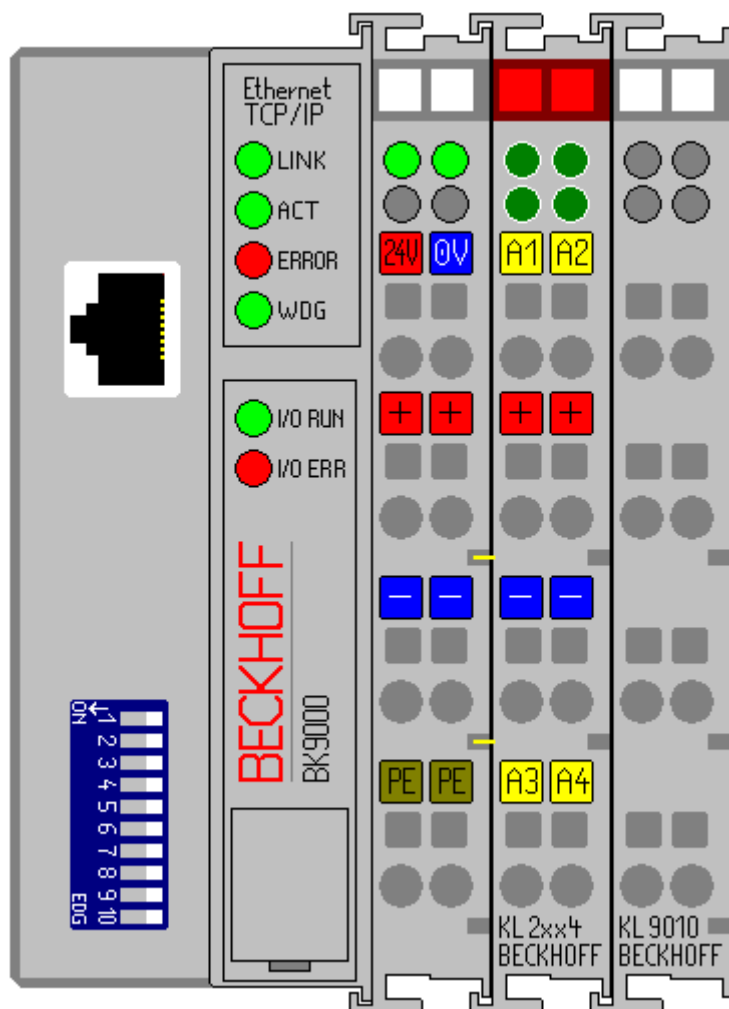
ハードウェア要件:

- イーサネット - バスカプラ(BK9000)
- 4つのデジタル出力を備える1つのバスターミナル(KL2114)
- バスエンドターミナル(KL9010)

この例は、このイーサネットバスステーションの内容に最適な要件です。ただし、ハードウェアは交換可能です。交換する場合は、I/Oデバイスの構成を変更する必要があります。

バスカプラとターミナルのセットアップ

次の図で示したように、バスカプラとバスターミナルをセットアップしてください。



ハードウェアドキュメンテーション

ハードウェア接続に関する詳細情報がバスカプラのハードウェアドキュメンテーションに記載されていません。

TwinCAT PLC Configuration

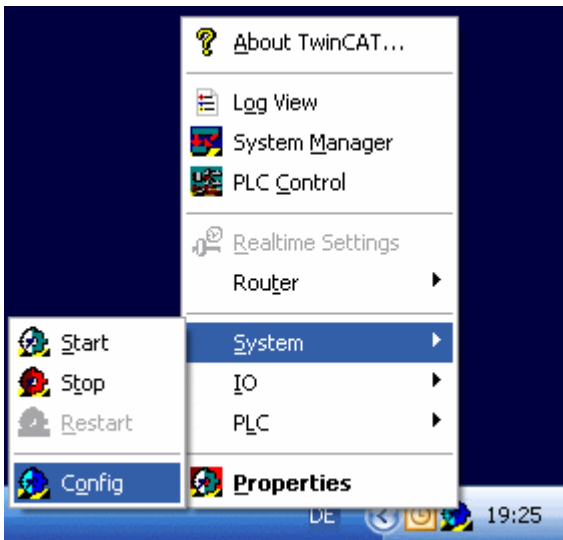
ローカルPLCが最初に実行した時に、'Machine'プロジェクトの拡張バージョンが必要です。

「Machine_Final」は、この章 [▶ 26]の変更に対応しています。

PLCサンプルをダウンロード: <https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281688587/.zip>

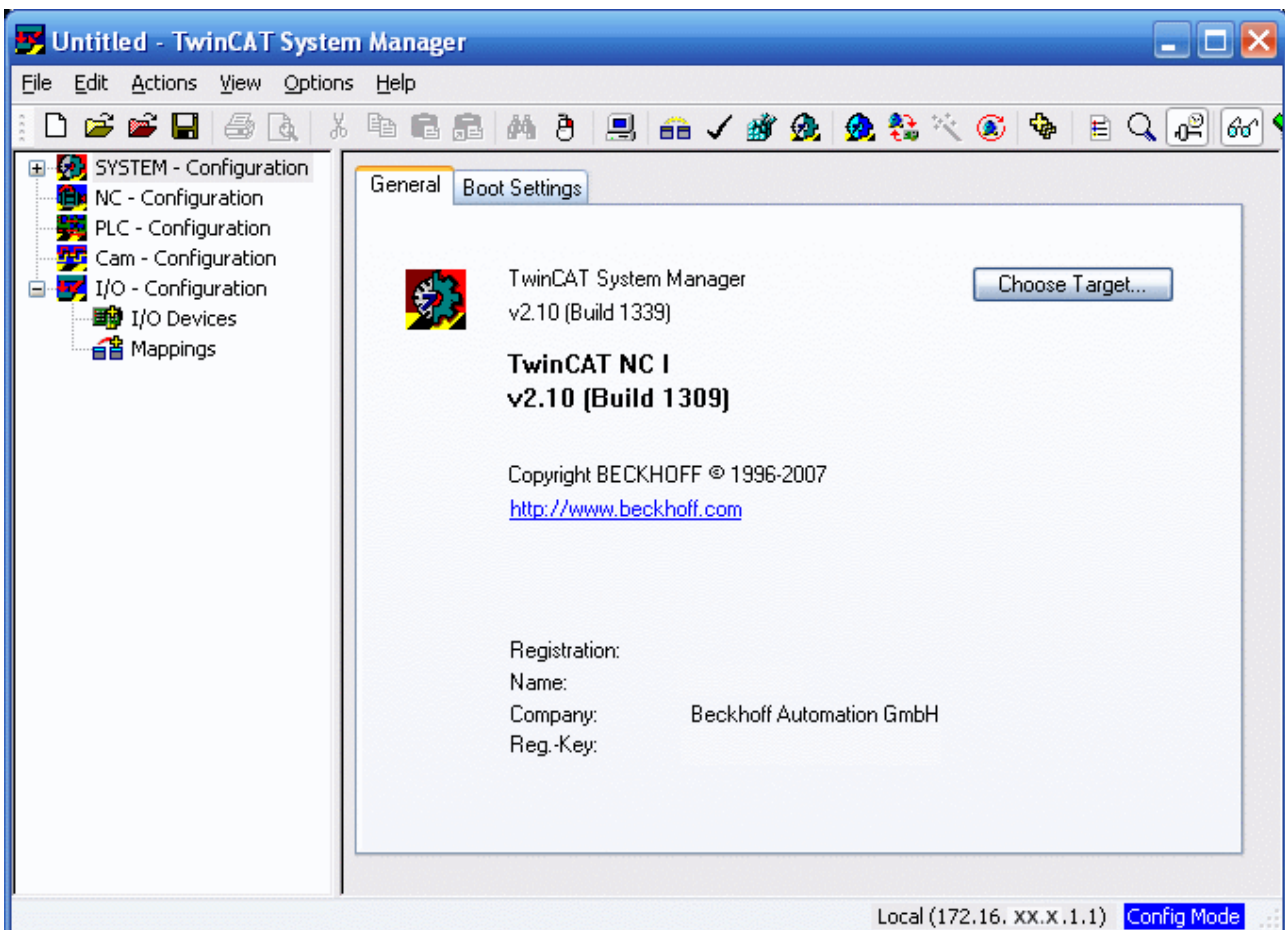
コンフィグモードでのTwinCATの起動

タスクバーでTwinCATのシンボルをクリックし、System/Configに移動します。これでTwinCATがコンフィグモードで起動され、ボックスのスキャンが可能になります。



TwinCATシステムマネージャの起動

システムが起動していれば、アイコンの色が赤から青に変わります。この時点でTwinCATシステムマネージャを起動してください。起動するには、[Start|Programs|TwinCAT System|TwinCAT System Manager]を選択します。



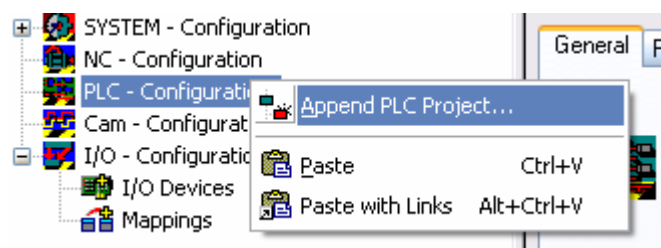
TwinCATシステムマネージャの項目

最初の行はプロジェクトの名前(ここでは「unbenannt」)で、その下にコマンドライン(メニュー)とツールバーがあります。最後の行にはシステムの状態が表示されます。この例では、システムが実行中(RTime)です。中央の2つのウィンドウにシステムの構成が表示されます。次の手順でシステムを構成します。システム構成がシステムマネージャの左側にツリー構造として表示されます。ツリー構造は以下の4つの主要項目から成ります。

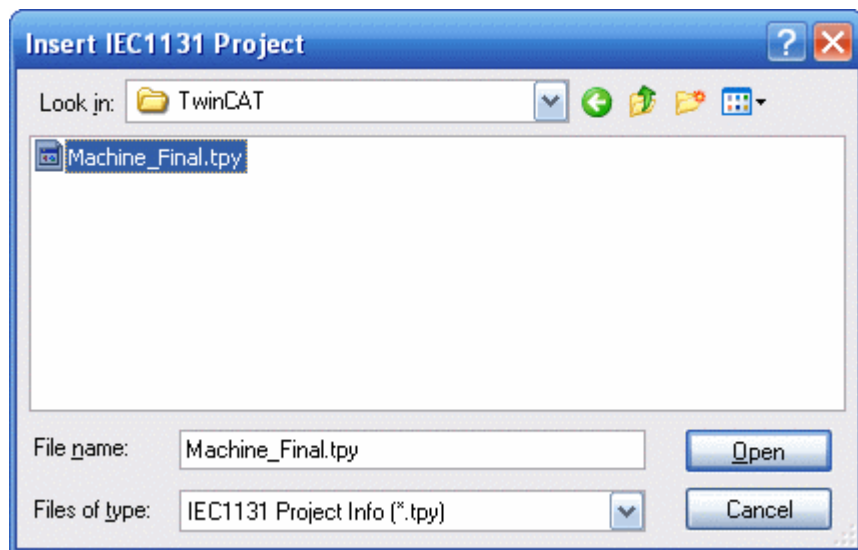
構成	意味
システム	システムの構成とリアルタイムパラメータ
NC	(オプション)NCタスクを追加し、それらを構成します。
PLC	構成する必要があるすべてのPLCプロジェクト
Cam	(オプション)カムグループを追加します。
I/O	コントローラをプロセスレベルにリンクするために、システムにインターフェイスが必要です。このエントリにてすべてのインターフェイスのリストが示されます。

PLC Configuration:

個々のPLCプロジェクトをシステムマネージャに認識させる必要があります。そうすれば、TwinCATがPLCプログラムの変数にアクセスできます。認識させるには、マウスポインタを[PLC configuration]の上に置いて右クリックします。

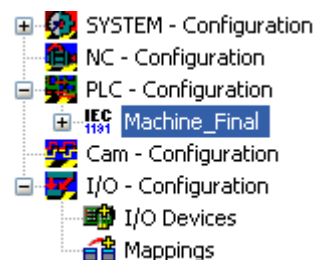


コンテキストメニューが開きます。そのメニューで[Append PLC project...]エントリを選択します。

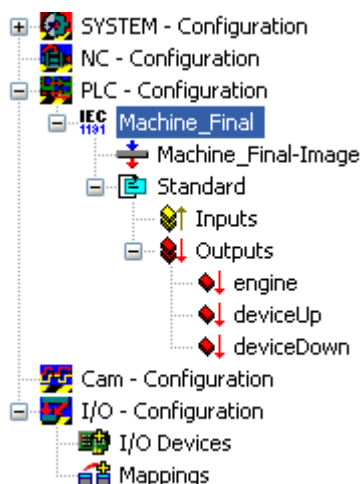


「¥TwinCAT¥Samples¥FirstSteps¥TwinCAT」ディレクトリに切り替えて「Machine_Final.tpy」ファイルを選択します。

PLCプロジェクトの名前が付けられたエントリが[PLC configuration]の下に追加されています。

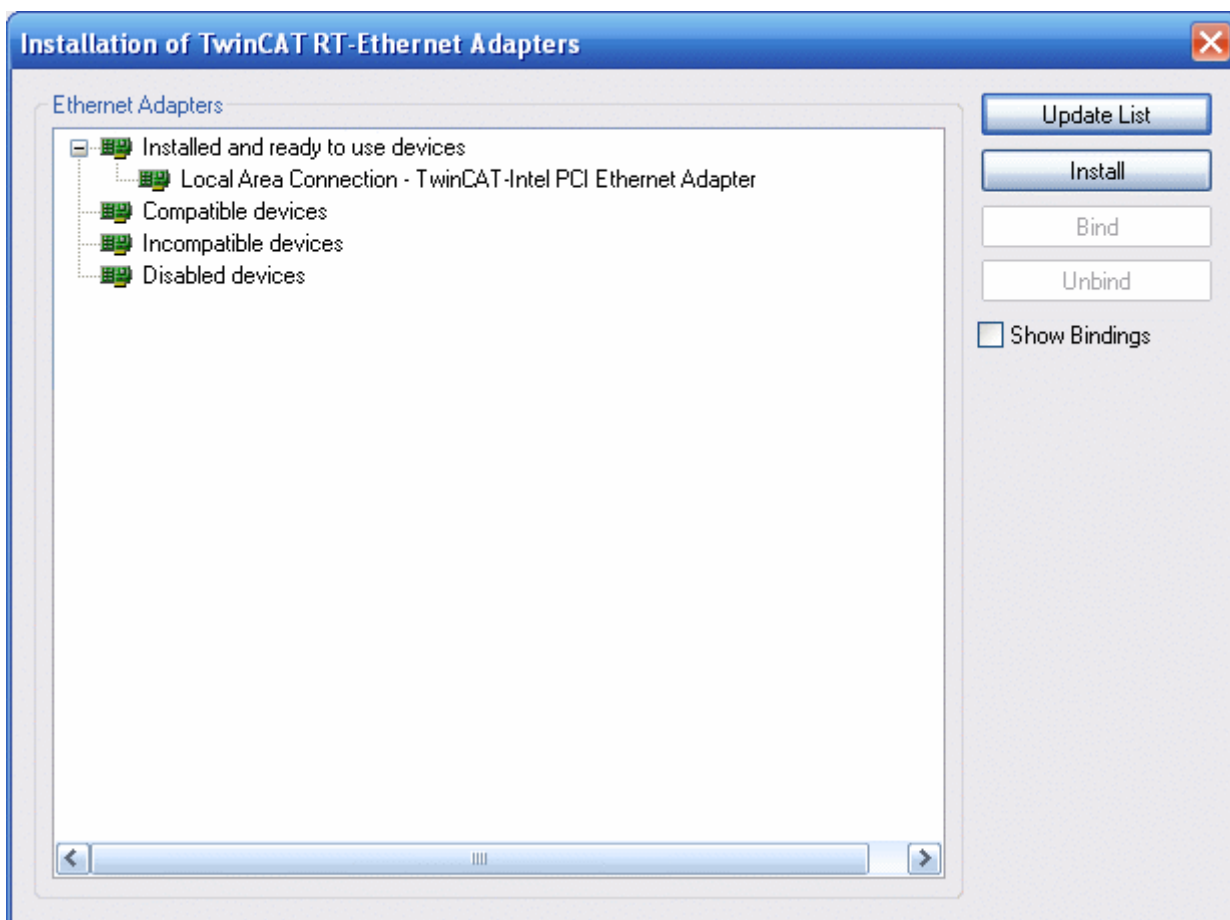


+記号と-記号は、エントリに他の下位エントリが含まれているかどうかを示します。これらの記号をクリックすると、その下のエントリを開いたり閉じたりすることができます。できるだけ深くツリーを開くと、以下の構造が表示されます。



リアルタイムイーサネットのセットアップ

リアルタイムイーサネット対応デバイス(BK9000)を使用するには、ネットワークカードに関連ドライバをインストールする必要があります。
メニューバーで[Options|List of real-time Ethernet compatible devices...]を選択します。



次のダイアログに4つの下位項目があります。

インストールされすぐに使用できるデバイス	ここにリストされるネットワーク接続はリアルタイムイーサネットに対応しており、使用できます。
適合するデバイス	ここにリストされるネットワーク接続は原則としてリアルタイムイーサネットに対応していますが、関連ドライバを必要とします。[Install]をクリックしてください。インストールが成功すると、ネットワーク接続が[Installed and ready to use devices]の下に表示されます。
適合しないデバイス	ここにリストされるネットワーク接続はリアルタイムイーサネットに対応していません。
無効なデバイス	ここにリストされるネットワーク接続は無効であり、現在使用できません。

続行するには、少なくとも1つのネットワーク接続が[Installed and ready to use devices]の下に表示されていないとなりません。

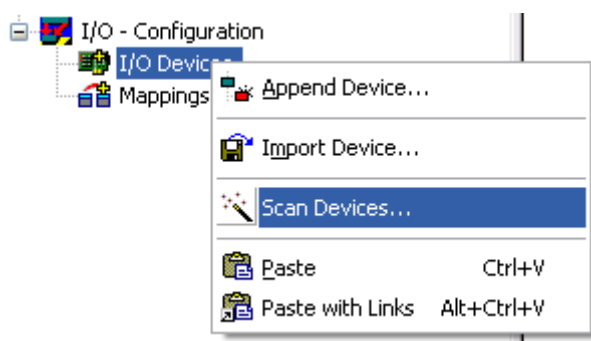
I/O Configuration

デバイスの追加

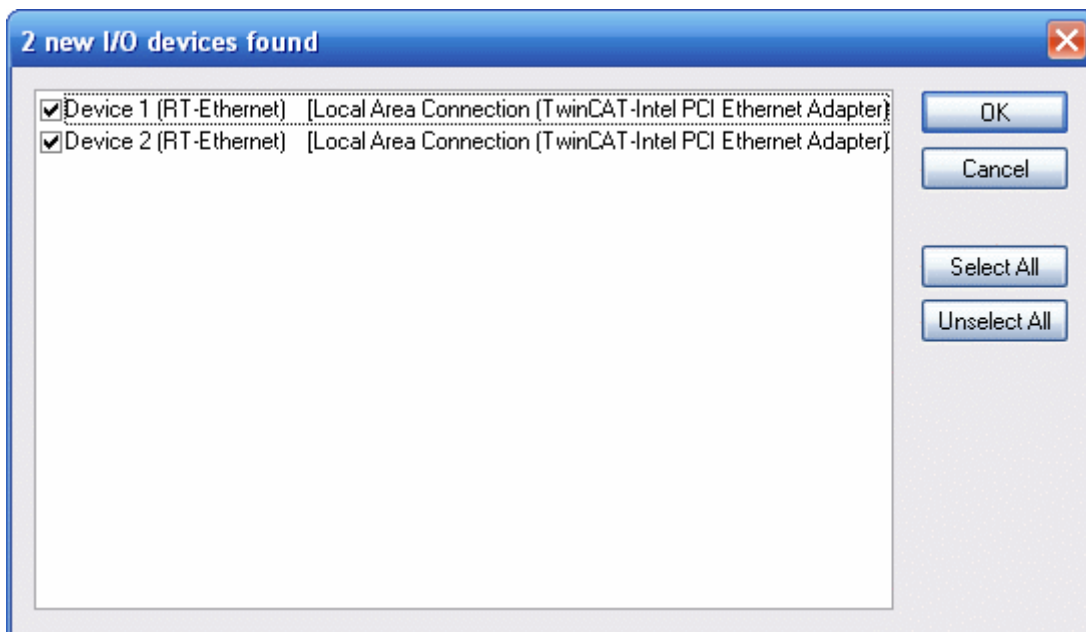
PLC構成と同じ方法でI/Oデバイスを挿入します。

TwinCAT 2.9ではデバイスを自動的にスキャンすることができます([Scan devices])。スキャン後、検出されたデバイスがツリー表示のI/Oデバイスの下にリストされます。この機能は、ターゲットシステムのコンフィグモードで行われます。

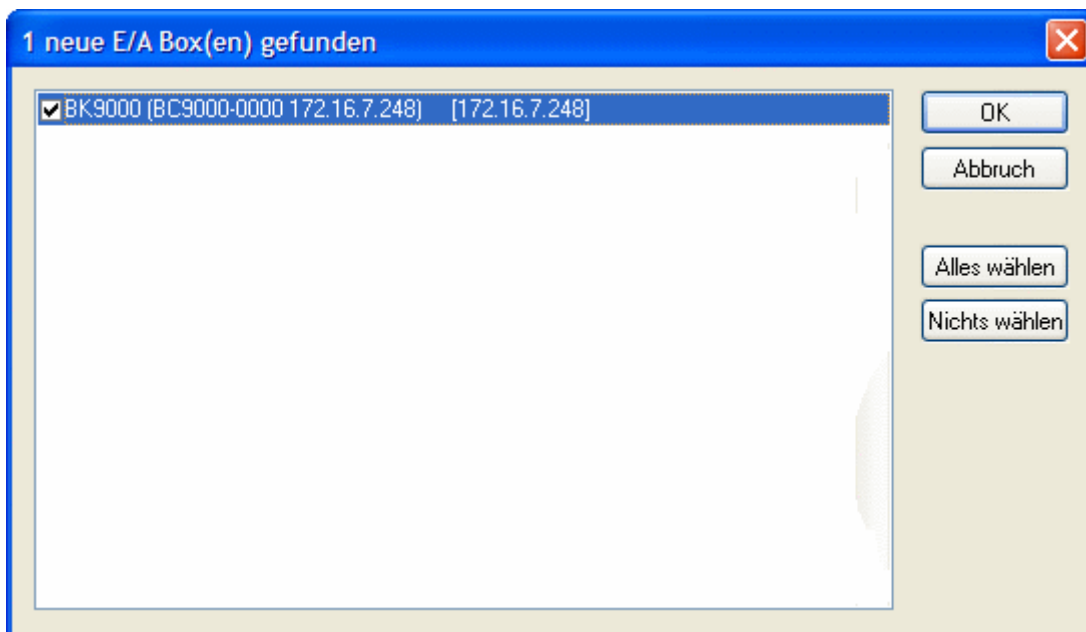
[I/O devices]エントリを右クリックして選択します。コンテキストメニューが開きます。そのメニューで[Scan devices]エントリを選択します。



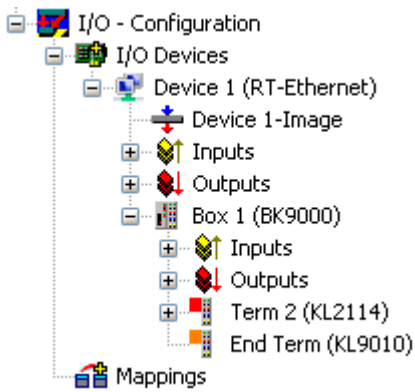
見つかったすべてのデバイスのリストを含む次のウィンドウが開きます。[RT-Ethernet]デバイスを選択します。このタイプのデバイスが複数リストされている場合は、BK9000に接続するデバイスを選択してください。



TwinCATにより、接続された周辺機器がスキャンされ、検出されたデバイスがリストされます。[OK]で確定します。



BK9000を選択し、[OK]で確定します。[Free Run]は有効にしないでください。
これで、見つかったI/O構成がシステムマネージャで表示されます。



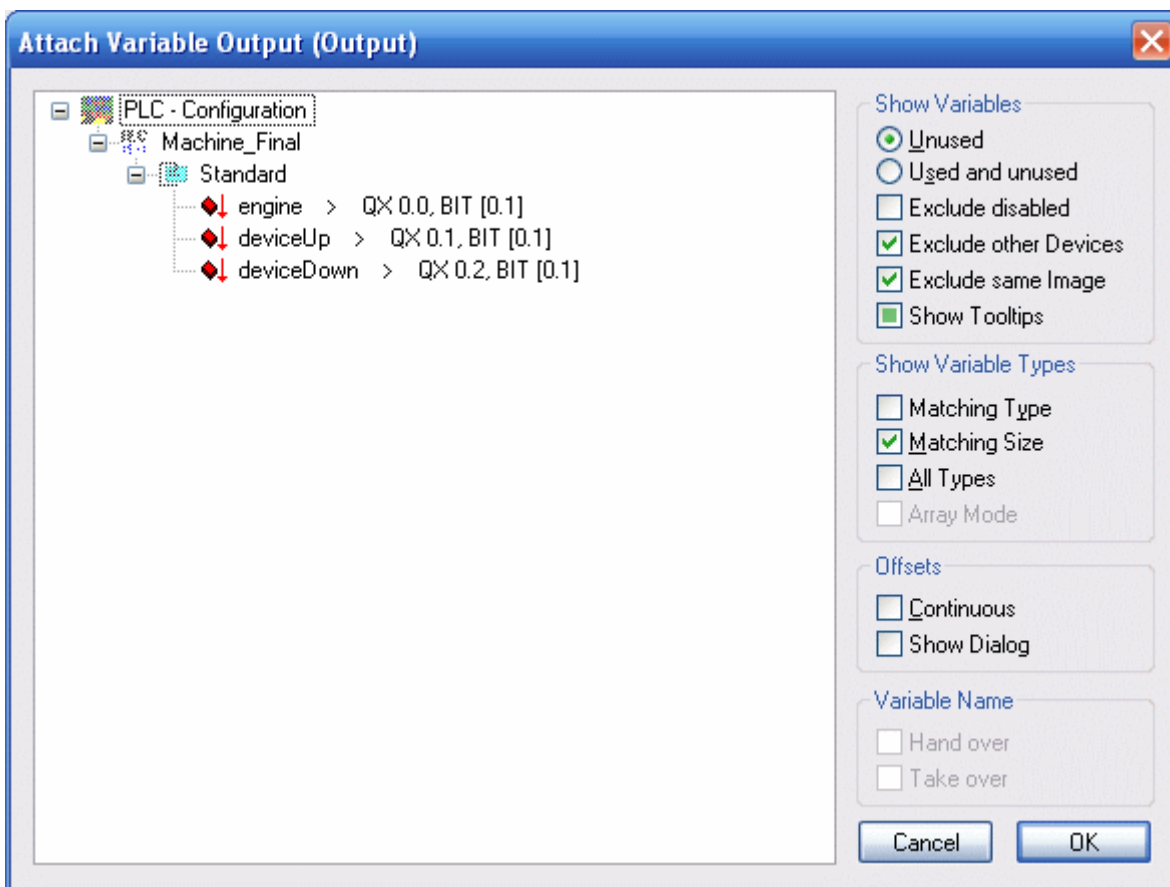
補足として、デフォルトで付く標準名称(device 1、box 1、terminal 1など)は変更できます。変更は対応する名前をゆっくりとダブルクリックし、新しい名前を入力します。

入出力チャンネルへの変数の割り当て:

この時点までに、上記のサンプルプログラムに完全必要なハードウェアが構成されています。次に、PLCプロジェクトからの個々の変数を個々の入出力チャンネルに割り当てる必要があります。

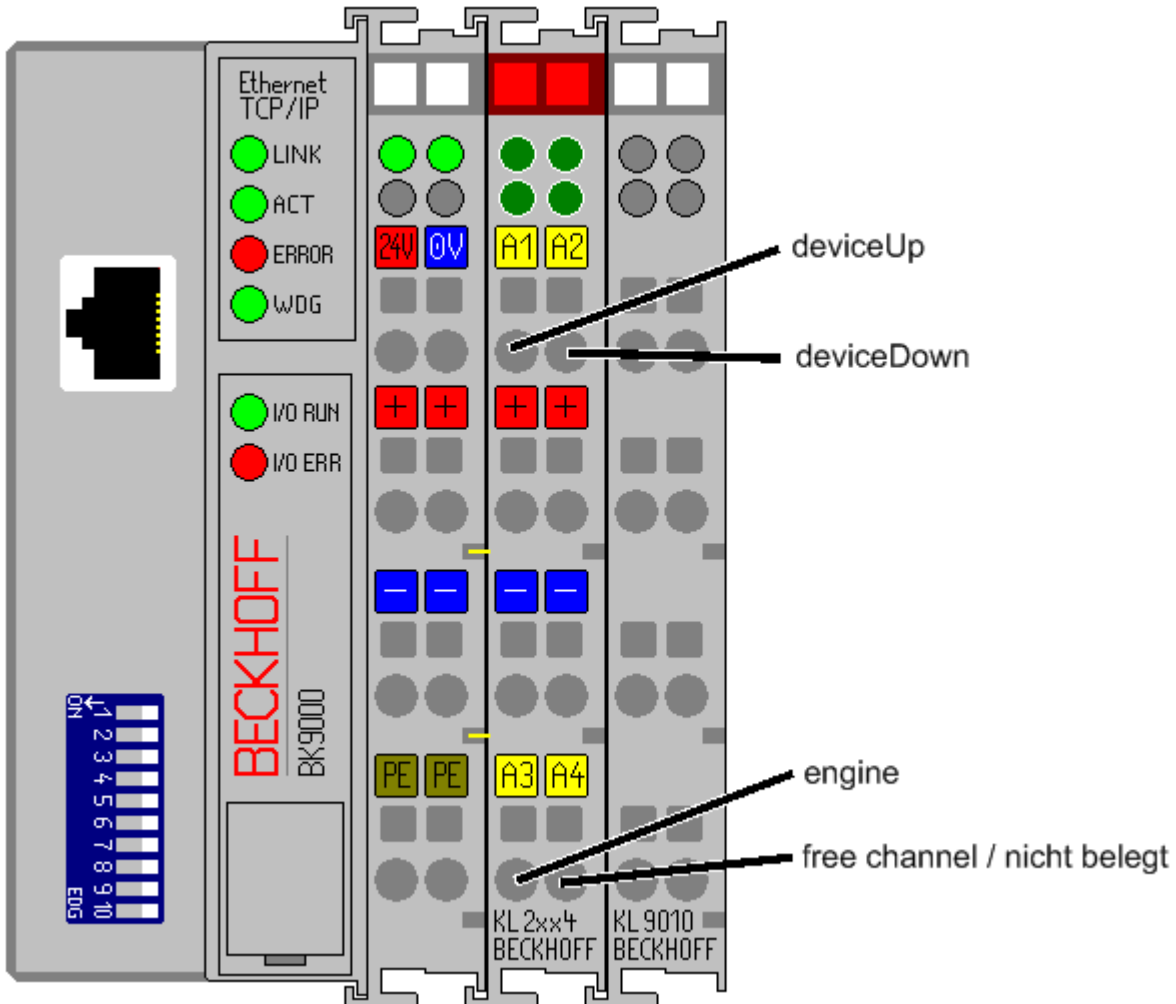
割り当てるには、構成するターミナルをマークします。接続された周辺機器の入力チャンネルと出力チャンネルを含むダイアログボックスが開きます。[Variables]タブを選択します。

リスト上に4つの出力チャンネルが表示されますが、4つのチャンネルはフリーです。必要な出力チャンネルを選択します。望ましいデジタル出力チャンネルの変数をすべて選択します。



ターミナル2	PLC変数	意味
ターミナル1 (=出力1)	DeviceUp	ドリルアップ制御
ターミナル2 (=出力2)	DeviceDown	ドリルダウン制御
ターミナル3 (=出力3)	Engine	ステッピングモータ制御

バスターミナルの割り当て:

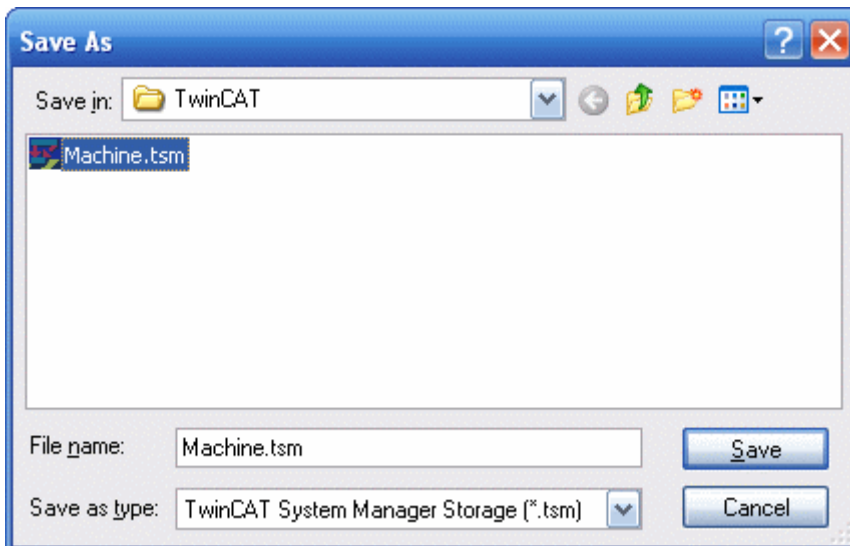


プロジェクトの有効化

プロジェクトの保存



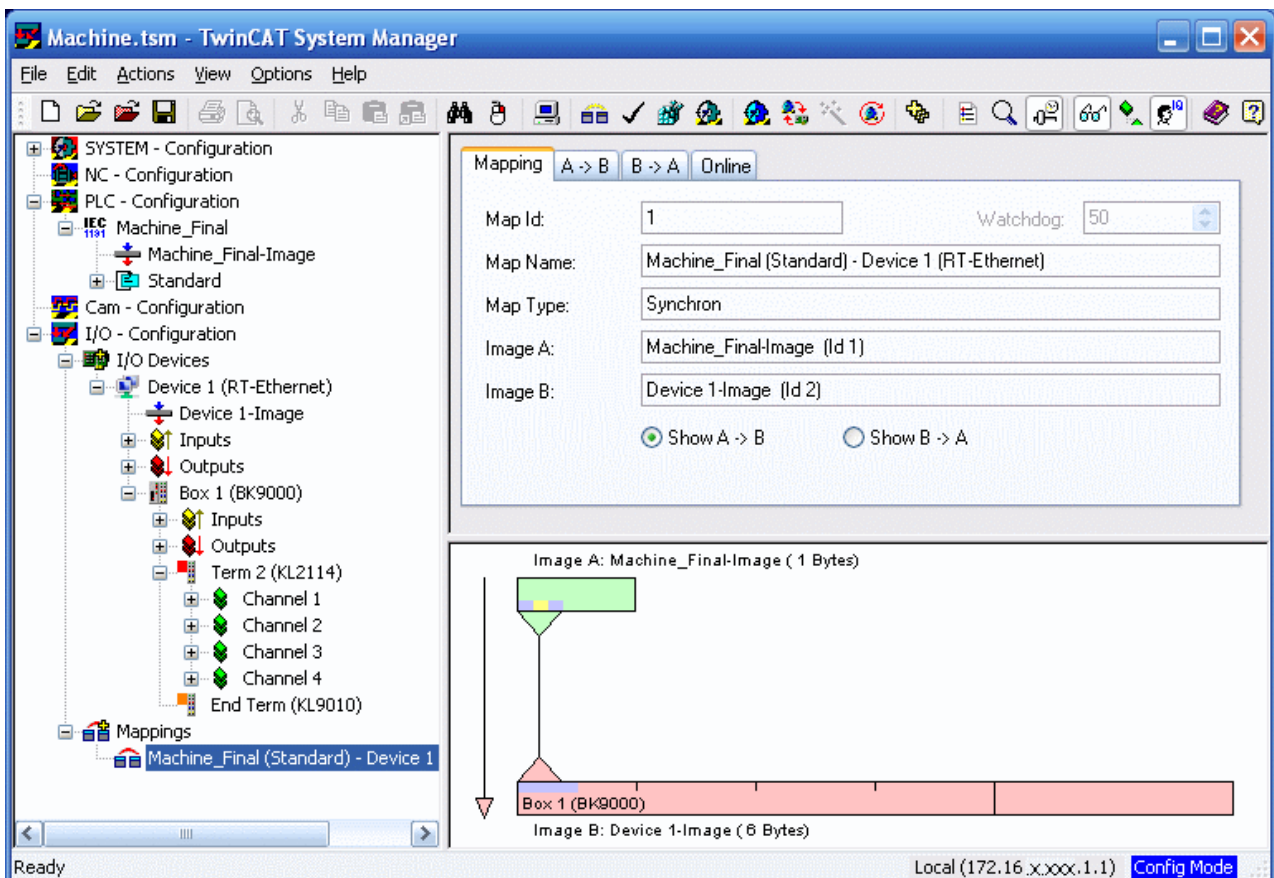
この時点で構成を保存しておくこと、後で保存した構成に確実にアクセスできます。保存するには、[File]メニューから[Save as...]コマンドを実行します。



変数のマッピング:



上記のサンプルプログラム用に完全なシステムを構成しました。この時点でレジストリの割り当てを作成する必要があります。作成するには、[Actions]メニューの[Create mapping]コマンドを実行します。[Mappings]ツリーエントリの下に[Machine (Standard) - Device 1 (RT-Ethernet)]と表示されています。このエントリをクリックします。次のウィンドウが右側に開きます。



ダイアログボックスで、「AからBへ」または「BからAへ」のどちらのデータフローを表示するのかを定義できます。このサンプルでは、イメージAがPLC変数(入出力変数)のプロセスイメージに対応しています。イメージBがI/Oデバイス(バスカプラBK9000)のプロセスイメージに対応しています。それぞれの変数またはバスターミナルは、色付きプロセスイメージで表示されています。いずれかのエリア上でマウスを停止すると、小さな表示ボックスが開き、正確な名前が表示されます。右クリックすると拡大できます。

レジストリへの構成の書き込み



最後の手順として、構成をレジストリに保存する必要があります(理由として、保存された情報がTwinCATの起動時に評価されるため)。[Actions]メニューから[Save in registry...]コマンドを実行します。古い構成がレジストリに既に保存されている場合、安全確認表示されますので確認する必要があります。

TwinCATの再起動:

変更を有効にするには、TwinCATを[Run Mode]で再起動する必要があります。[OK]でシステムマネージャのダイアログを確定します。

PLCプロジェクト「Machine_Final.pro」を再ロードしなければならないことがあります。

個々のPLC変数がバスターミナルKL2404で出力されます。バスターミナルで、LEDによって信号状態が示されます。

TwinCATシステムマネージャに関する詳細情報がBeckhoff Information Systemに含まれています。

7.3.4 EtherCAT - バスターミナルのセットアップ

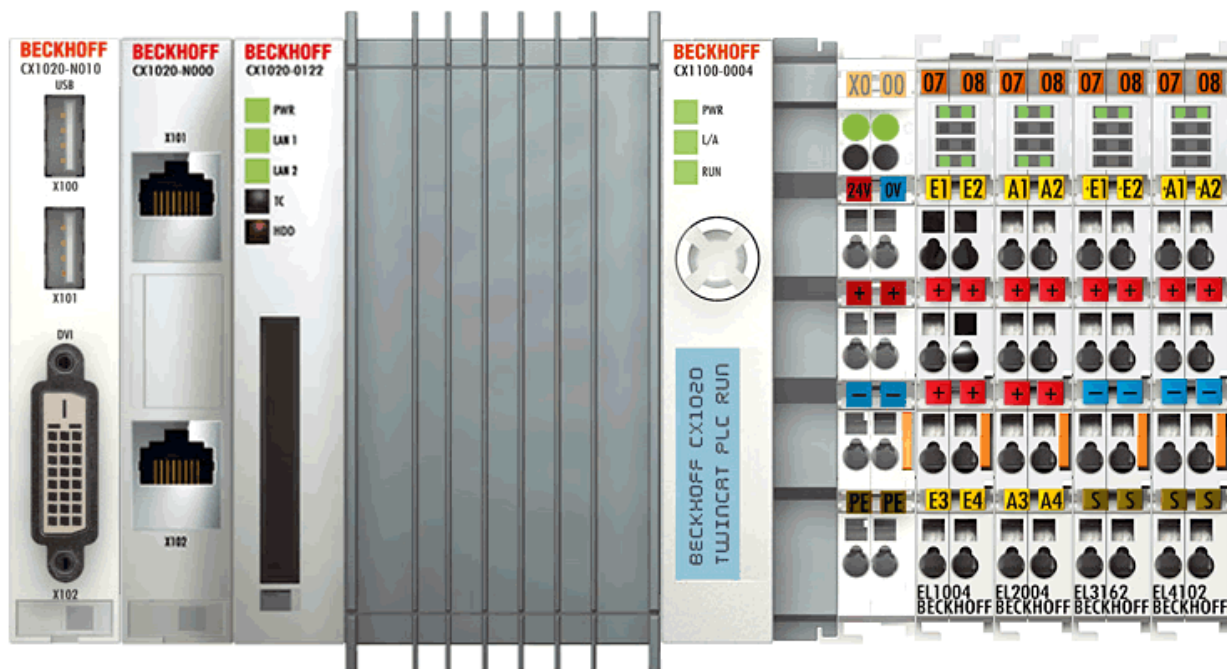
ハードウェア要件:

- 組込み型PC CX1020
 - CPUベースモジュールC1020-0123:512 MBフラッシュメモリ、WindowsXP Embedded
 - システムインターフェイスCX1020-N010:1つのDVI-Iインターフェイスと2つのUSBインターフェイス。
 - 電源ユニットCX1100-0004(EtherCATインターフェイスを装備)
- または従来型のIPC(Windows 2000、XP、またはVistaと未使用イーサネットポートを少なくとも1つ装備)。
- EtherCAT I/O-ターミナル:
 - EL2004:4xデジタル出力、24 V DC、0.5 A、典型的なサイクルタイム90 µs
 - その他のI/Oターミナルが可能

ここでの説明は組込み型PC CX1020に関するものですが、他のすべてのWindows PCにも当てはまりません。

ターミナルのセットアップ:

次の図で示したようにCX1020をセットアップしてください。(構成例参照)



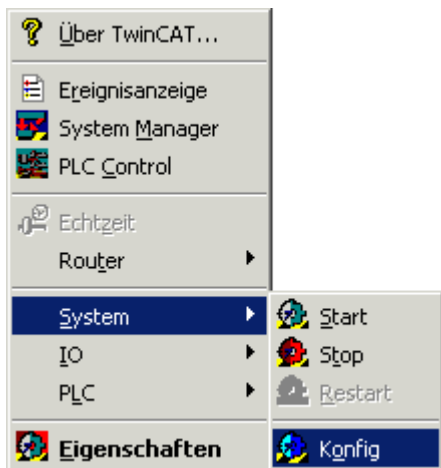
CX1020に24 V DCの電源を投入します。

ハードウェアドキュメンテーション

ハードウェア接続に関する詳細情報が、対応するハードウェアドキュメンテーションに記載されています。

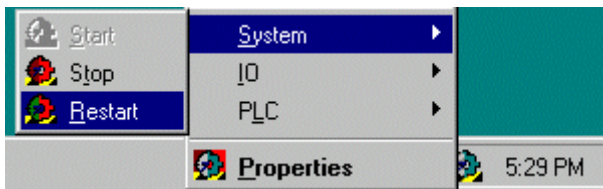
TwinCATリアルタイムサーバの起動

この時点で、まだ起動していなければTwinCATリアルタイムサーバを「Config Mode」で起動してください。これを行うと、TwinCATメッセージルータが有効になります。



TwinCAT System Managerの起動

システムが起動していれば、アイコンの色が赤から青に変わります。この時点でTwinCATシステムマネージャを起動してください。起動するには、[Start|Programs|TwinCAT System|TwinCAT System Manager]を選択します。



TwinCAT System Managerの項目:



最初の行はプロジェクトの名前(ここでは「unbenannt」)で、その下にコマンドライン(メニュー)とツールバーがあります。最後の行にはシステムの状態が表示されます。この例では、システムが実行中(RTime)です。中央の2つのウィンドウにシステムの構成が表示されます。次の手順でシステムを構成します。

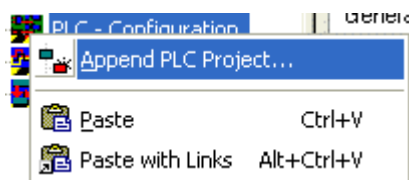
システム構成がシステムマネージャの左側にツリー構造として表示されます。ツリー構造は以下の3つの主要項目から成ります。

構成	意味
System configuration	システムとリアルタイムパラメータを設定します。
PLC configuration	構成する必要があるすべてのPLCプロジェクト
I/O configuration	コントローラをプロセスレベルにリンクするために、システムにインターフェイスが必要です。このエントリにてすべてのインターフェイスのリストが表示されます。

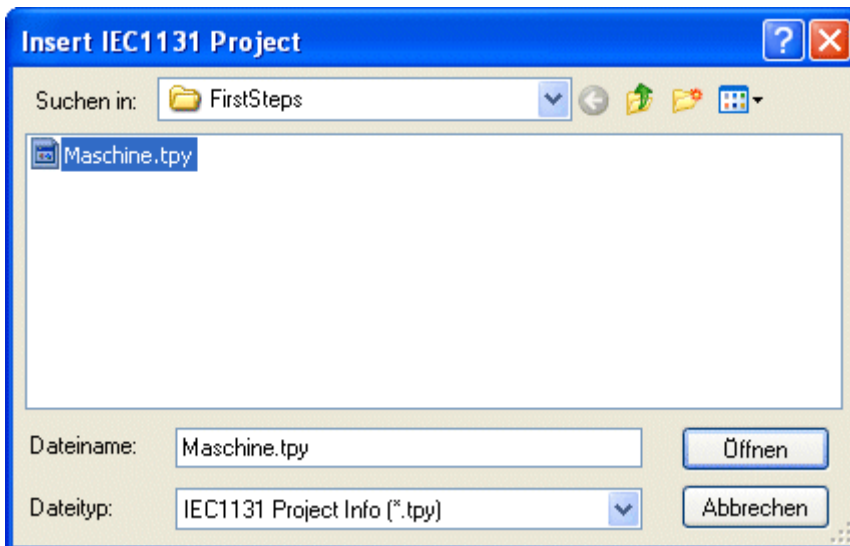
その他のシステムコンポーネントがリストされることもあります(例えば、カムサーバやNC構成)。

PLC Configuration:

個々のPLCプロジェクトをシステムマネージャに認識させる必要があります。そうすれば、TwinCATがPLCプログラムの変数にアクセスできます。認識させるには、マウスポインタを[PLC configuration]の上に置いて右クリックします。

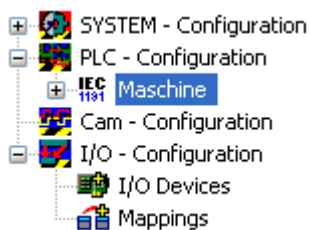


コンテキストメニューが開きます。そのメニューで[Append IEC project...]を選択します。

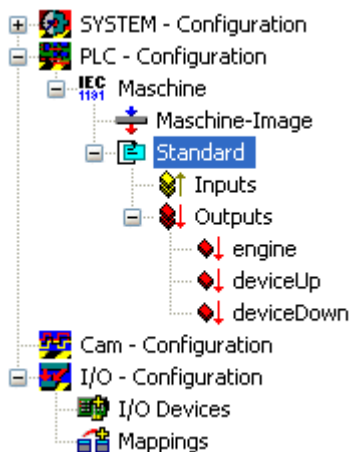


「¥TwinCAT¥Samples¥FirstSteps¥」ディレクトリに切り替えて「maschine.tpy」ファイルを選択します。

PLCプロジェクトの名前が付けられたエントリが[PLC configuration]の下に追加されています。



+記号と-記号は、エントリに他の下位エントリが含まれているかどうかを示します。これらの記号をクリックすると、その下のエントリを開いたり閉じたりすることができます。できるだけ深くツリーを開くと、以下の構造が表示されます。



上記の図で、「standard」という名前のタスクがサンプルプログラムに含まれていることが分かります。

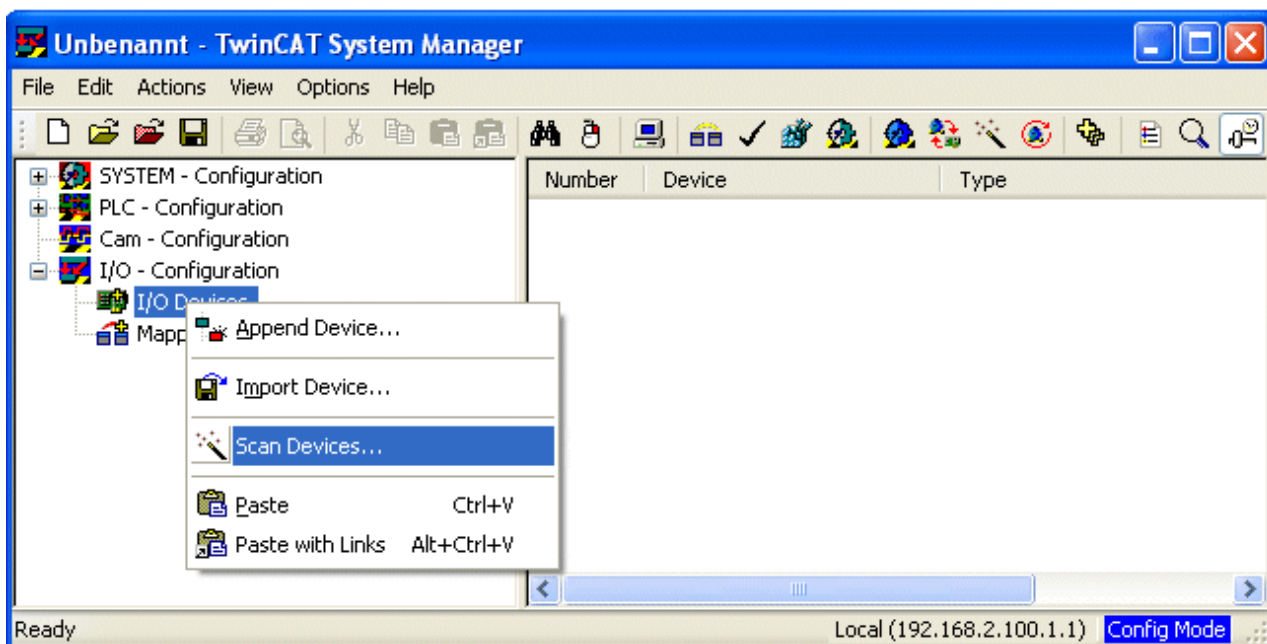
I/OConfiguration

デバイスの追加

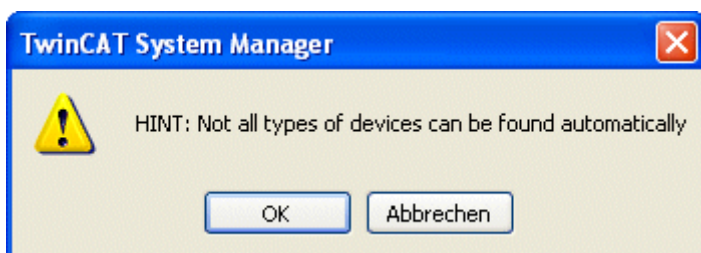
PLC構成と同じ方法でI/Oデバイスを挿入します。

TwinCAT 2.9ではデバイスを自動的にスキャンすることができます([Scan devices])。スキャン後、検出されたデバイスがツリー表示のI/Oデバイスの下にリストされます。
この機能は、ターゲットシステムのコンフィグモードで行われます。

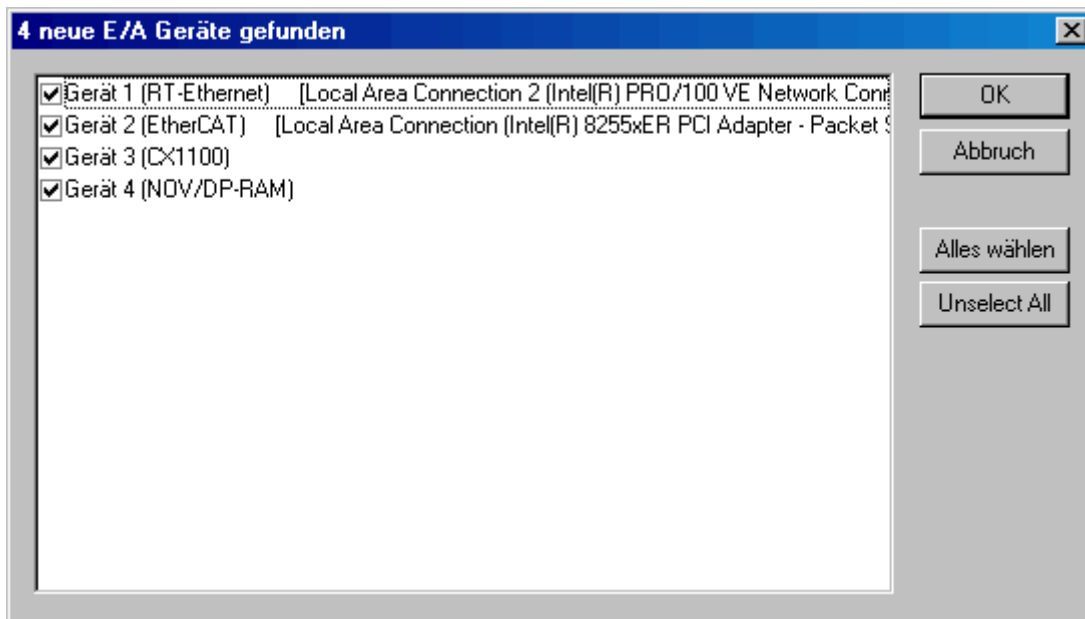
[I/O devices]エントリを右クリックして選択します。コンテキストメニューが開きます。そのメニューで[Scan devices]エントリを選択します。



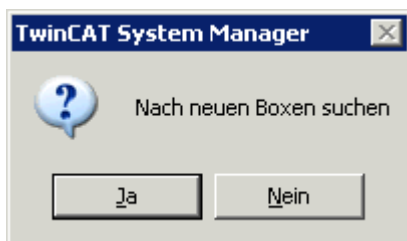
次のダイアログが開きます。



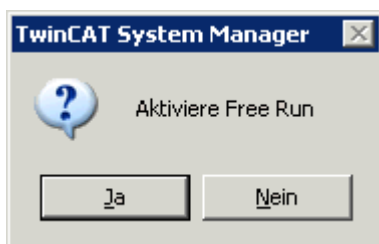
TwinCATにより、接続された周辺機器がスキャンされ、見つかったデバイスがリストされます。[OK]をクリックしてください。



システムマネージャはCX1020のすべてのチャンネルにおけるボックス(バスカプラ、バスターミナル、バスターミナルコントローラ、またはフィールドバスボックスモジュール)を検索することを求めます。[Yes]をクリックしてください。

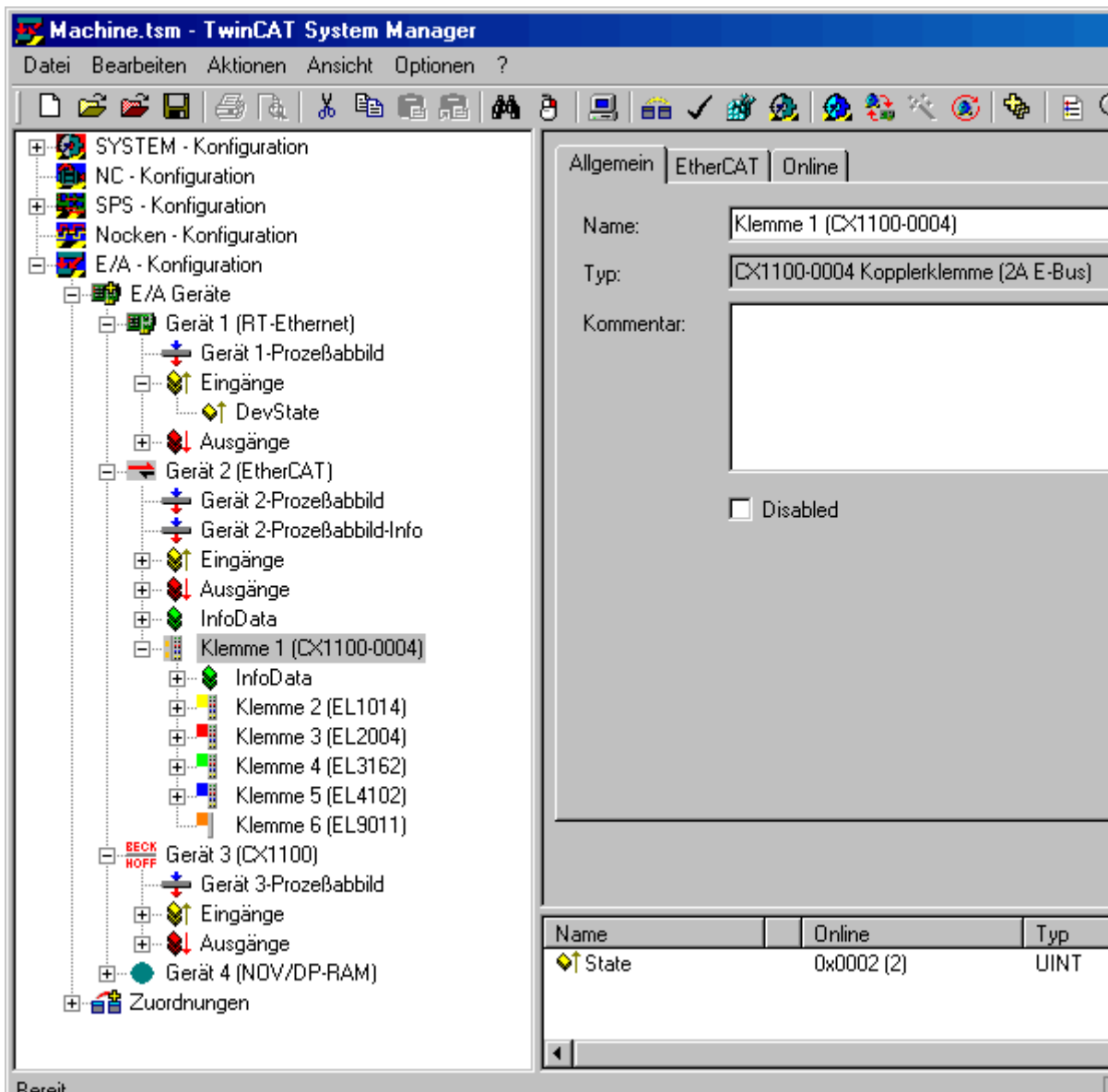


フリーランモードを有効にするには、[Yes]をクリックします。



見つかったI/Oデバイスをフリーランモードに設定できます。つまり、PLCプロジェクトまたはその他のトリガタスクを有効にすることなくバスターミナルのI/Oチャンネルを特定の状態に設定(書き込む)ことができます。

これで、見つかったI/O構成がシステムマネージャで表示されます。



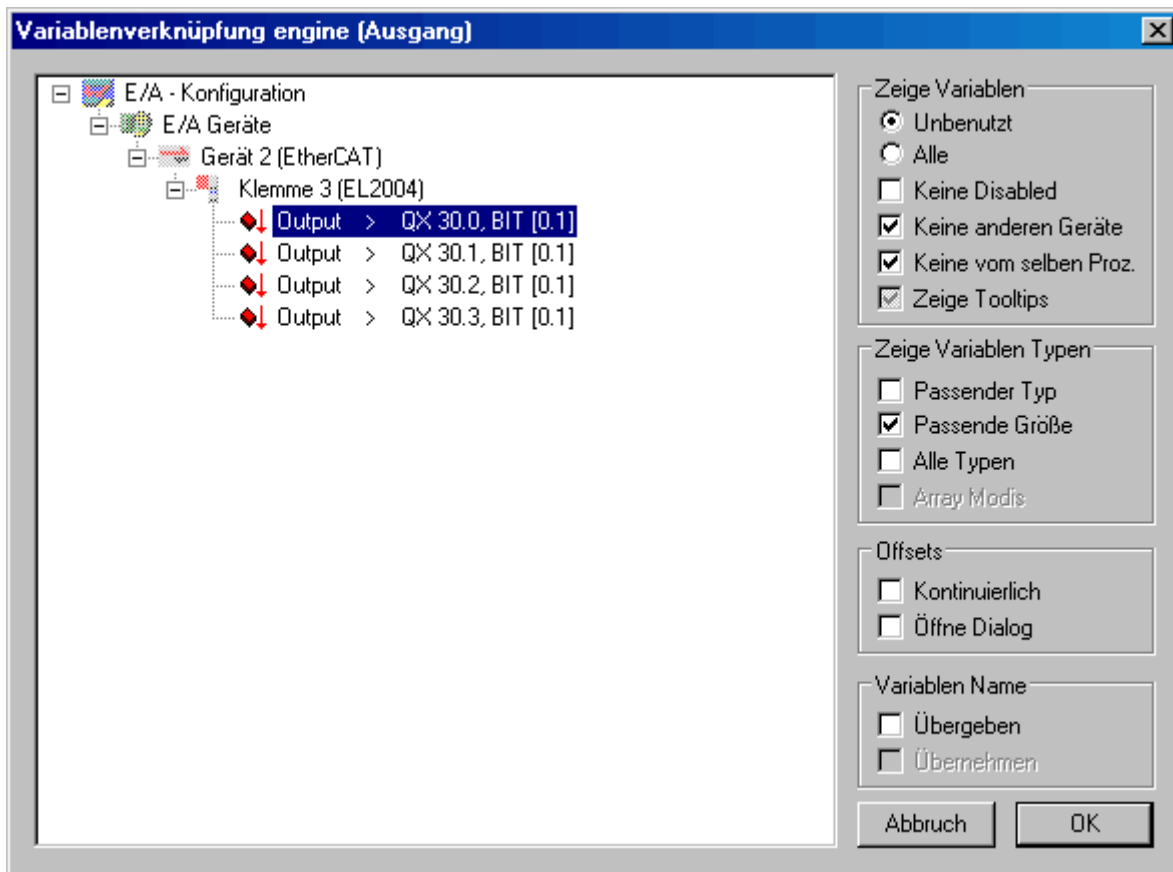
補足として、デフォルトで付く標準名称(device 1、box 1、terminal 1など)は変更できます。変更は対応する名前をゆっくりとダブルクリックし、新しい名前を入力します。

入出力チャンネルへの変数の割り当て:

この時点までに、上記のサンプルプログラムに完全必要なハードウェアが構成されています。次に、PLCプロジェクトからの個々の変数を個々の入出力チャンネルに割り当てる必要があります。

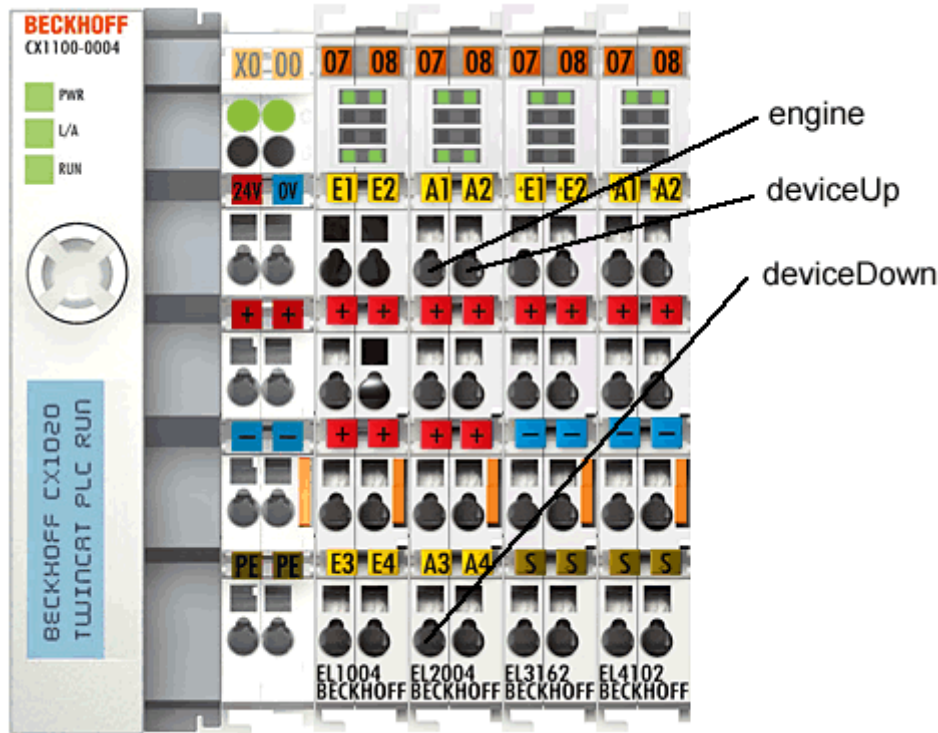
割り当てるには、構成するターミナルをマークします。接続された周辺機器の入力チャンネルと出力チャンネルを含むダイアログボックスが開きます。[Variables]タブを選択します。

リスト上に4つの出力チャンネルが表示されますが、4つのチャンネルはフリーです。必要な出力チャンネルを選択します。必要な出力チャンネルを選択します。



ターミナルEL1014	PLC変数	意味
チャンネル1 (=出力1)	engine	ステッピングモータ制御
チャンネル2 (=出力2)	deviceUp	ドリルアップ制御
チャンネル3 (=出力3)	deviceDown	ドリルダウン制御

バスターミナルの割り当て(構成例)



プロジェクトの保存:

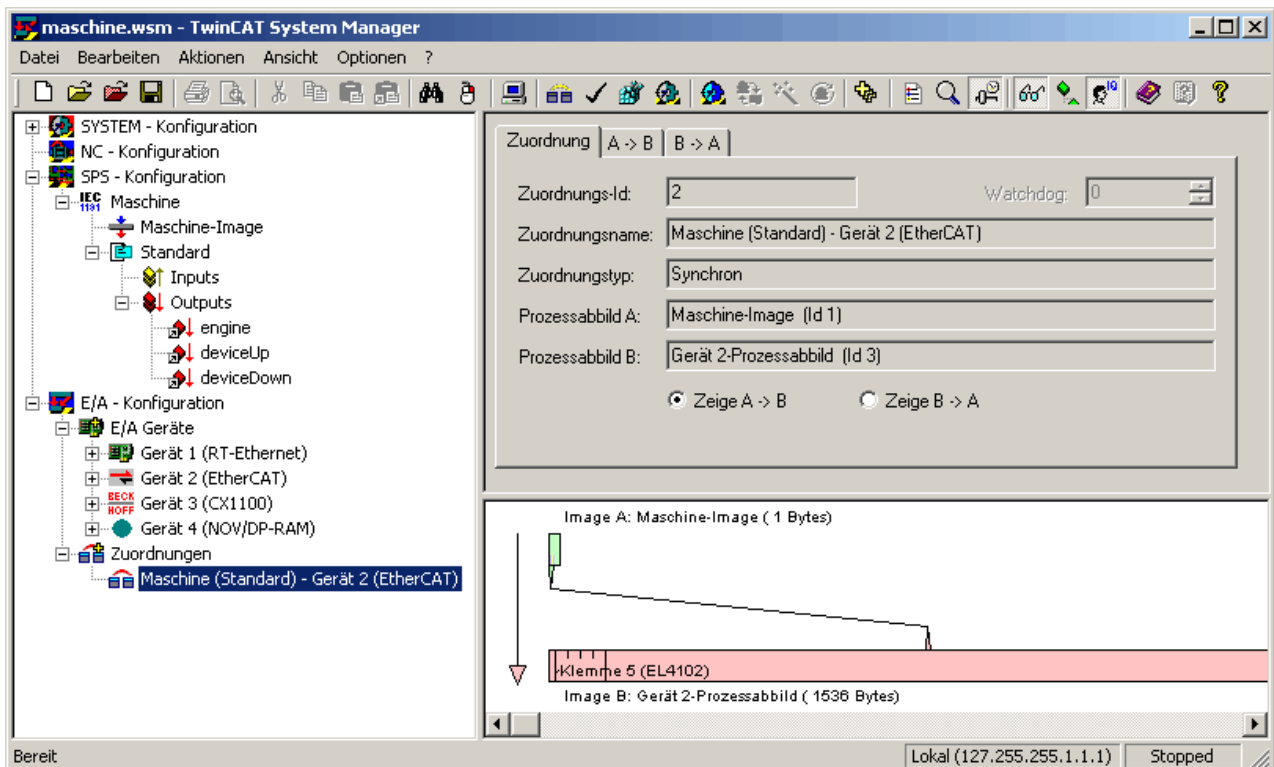


この時点で構成を保存しておく、後で保存した構成に確実にアクセスできます。保存するには、[File]メニューから[Save as...]コマンドを実行します。

変数のマッピング:



上記のサンプルプログラム用に完全なシステムを構成しました。この時点でレジストリの割り当てを作成する必要があります。作成するには、[Actions]メニューの[Create mapping]コマンドを実行します。[Mappings]ツリーエントリの下に[Maschine (Standard) - Device 2 (EtherCAT)]と表示されています。このエントリをクリックします。次のウィンドウが右側に開きます。



ダイアログボックスで、「AからBへ」または「BからAへ」のどちらのデータフローを表示するのかを定義できます。このサンプルでは、イメージAがPLC変数(入出力変数)のプロセスイメージに対応しています。イメージBがI/Oデバイス(EtherCATバスカプラ)のプロセスイメージに対応しています。それぞれの変数またはバスターミナルは、色付きプロセスイメージで表示されています。いずれかのエリア上でマウスを停止すると、小さな表示ボックスが開き、正確な名前が表示されます。右クリックすると拡大できます。

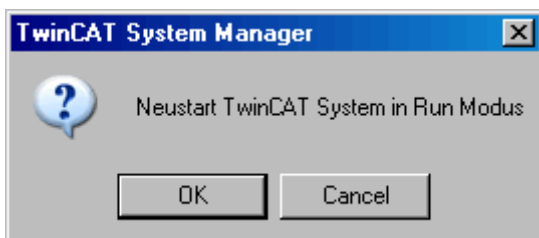
レジストリへの構成の書き込み:



最後の手順として、構成をレジストリに保存する必要があります(理由として、保存された情報がTwinCATの起動時に評価されるため)。[Actions]メニューから[Save in registry...]コマンドを実行します。古い構成がレジストリに既に保存されている場合、安全確認表示されますので確認する必要があります。

TwinCATの再起動:

変更を有効にするには、TwinCATを[Run Mode]で再起動する必要があります。



新しい構成により、PLC制御上にロードされたプログラムは存在なくなります。バスターミナルで、LEDによって信号状態が表示されます。

プログラム[PLC Control]を開き、[Online]メニューの[Login]コマンドを選択します。[No program on the control!]というメッセージが表示されます。[OK]で確定すると、プログラムがPLCにロードされます。

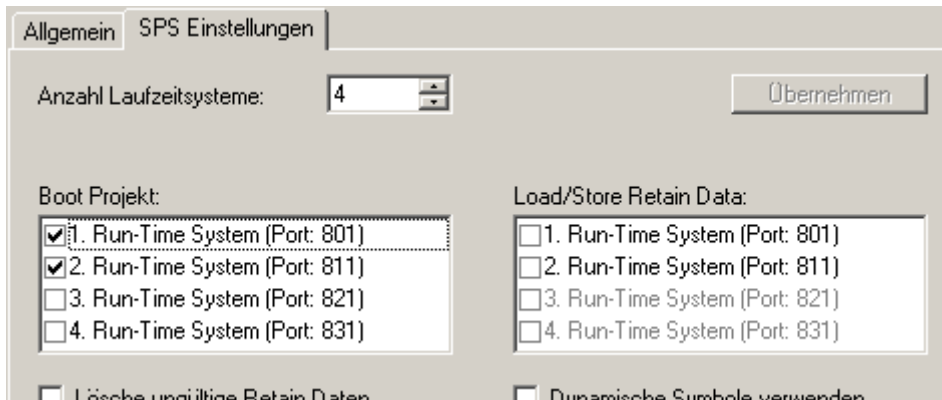
[Online|Start]を選択すると、プログラムが新しい構成で起動します。

個々のPLC変数がバスターミナルEL2004で出力されます。バスターミナルで、LEDによって信号状態が示されます。

ブート設定:

PLCを再起動するとすぐにプロジェクトを再ロードする必要があります。別の方法として、ブートプロジェクトを作成することもできます。このプロジェクトはレジストリに保存されていて、TwinCATリアルタイムサーバの起動時に起動します(ただし、構成が変更されていないことが条件です)。

まず、システムマネージャの[SPS Configuration]メニューの[SPS Settings]タブで、ブートプロジェクトをどのランタイムシステムに含めるのかを指定します。ここで、使用可能なランタイムシステムの数を指定することもできます。

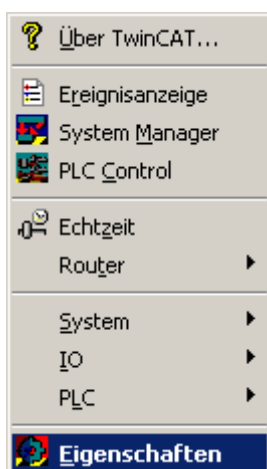


システムマネージャの[Append SPS project]で関連するプログラムをロードし、I/Oをリンクし構成を有効にします。ログイン([Online|Login])の後で、関連するPLCプログラムごとにTwinCAT PLCコントロールを使用してコマンド[Online|Create a boot project]を実行します。関連するコマンドを使用してプロジェクトを再び削除することができます。関連するランタイムシステムが一致していることを確認してください。ログアウトしてPLC Controlを閉じます。システムマネージャプロジェクトを保存します。

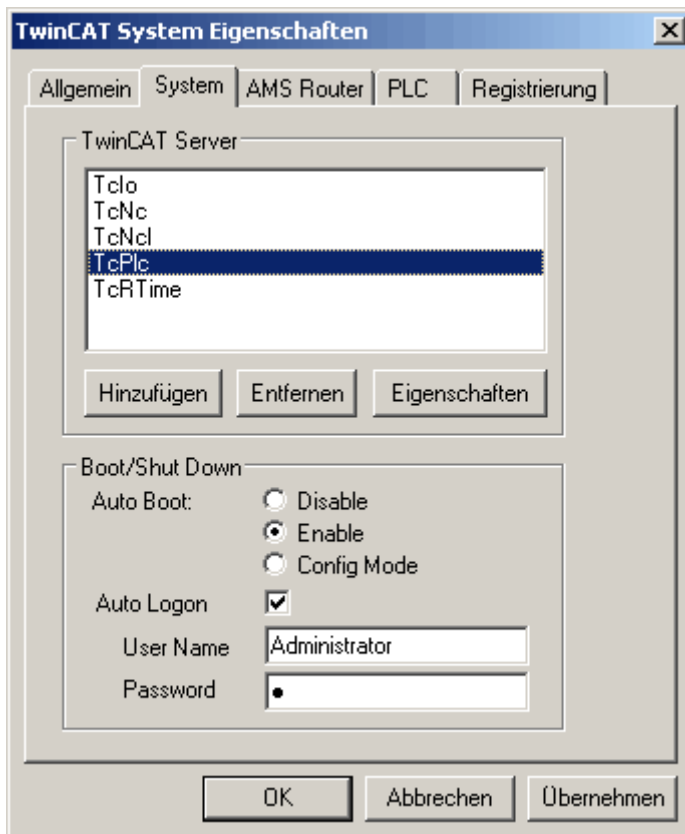
注記:

ブートプロジェクトの上書きまたは削除を行わないと、このランタイムシステムの新しいブートプロジェクトを作成するか削除するまで、古いブートプロジェクトがPLCにロードされたままになります。

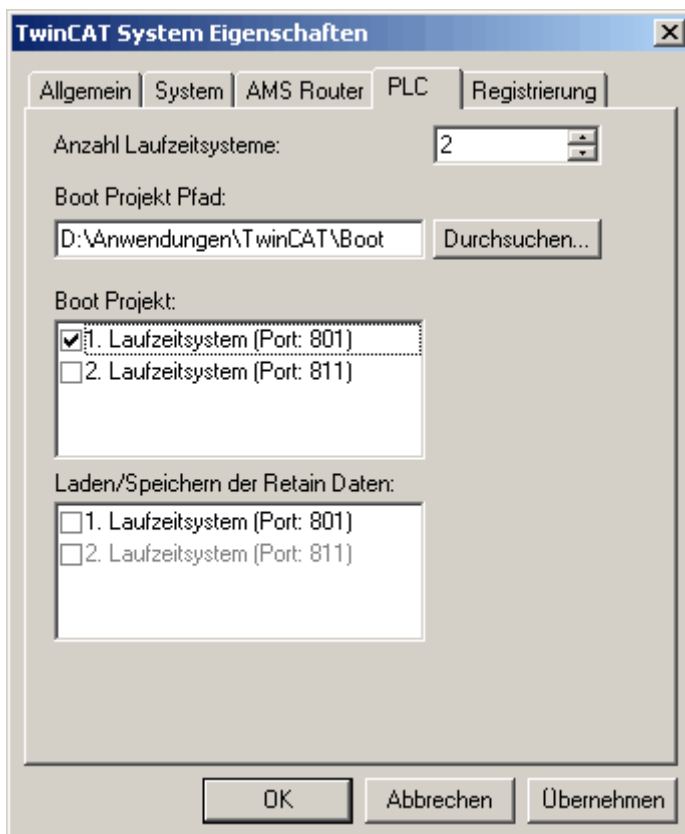
システムの起動時にTwinCATで1つまたは複数のPLCプログラムを自動的に処理する必要がある場合、ランタイムシステムの数を指定し、ブートプロジェクトをロードする必要があります(上記を参照)。ツールバーのTwinCATボタンを使用して[Properties]メニューを開きます。



[System]タブの[Auto Boot]の下の[Enable]オプションを有効にします。PLCプログラムが自動的に起動する場合は、[Auto Logon]でログインデータを挿入します。このエントリが無効である場合、システムの起動後にユーザログオンが必要です。



[Auto-Boot]の設定は、システムマネージャでの設定[System configuration|Boot settings]と同じです。これは、システムマネージャの[PLC configuration|PLC settings]の下の[PLC]タブの設定にも当てはまります。



システムの再起動の後にTwinCATが[Run]モードで起動します(自動ログオンが前提)。ロードされ有効になったブートプロジェクトの処理が始まります。

7.4 「Machine」を使用したサンプル

7.4.1 Microsoft Visual C#による「Machine」サンプル

Microsoft Visual Studio 2005は、C#プロジェクトを作成するための開発環境です。プログラミング言語C#を用いたTwinCAT ADS .NETコンポーネントの統合について、machineサンプルを参照しながら説明します。

必要なソフトウェア:

- Microsoft .NET Frameworkバージョン2.0、詳細については[ここをクリックしてください](#)。
- Microsoft Visual Studio 2005
- TwinCAT 2.10

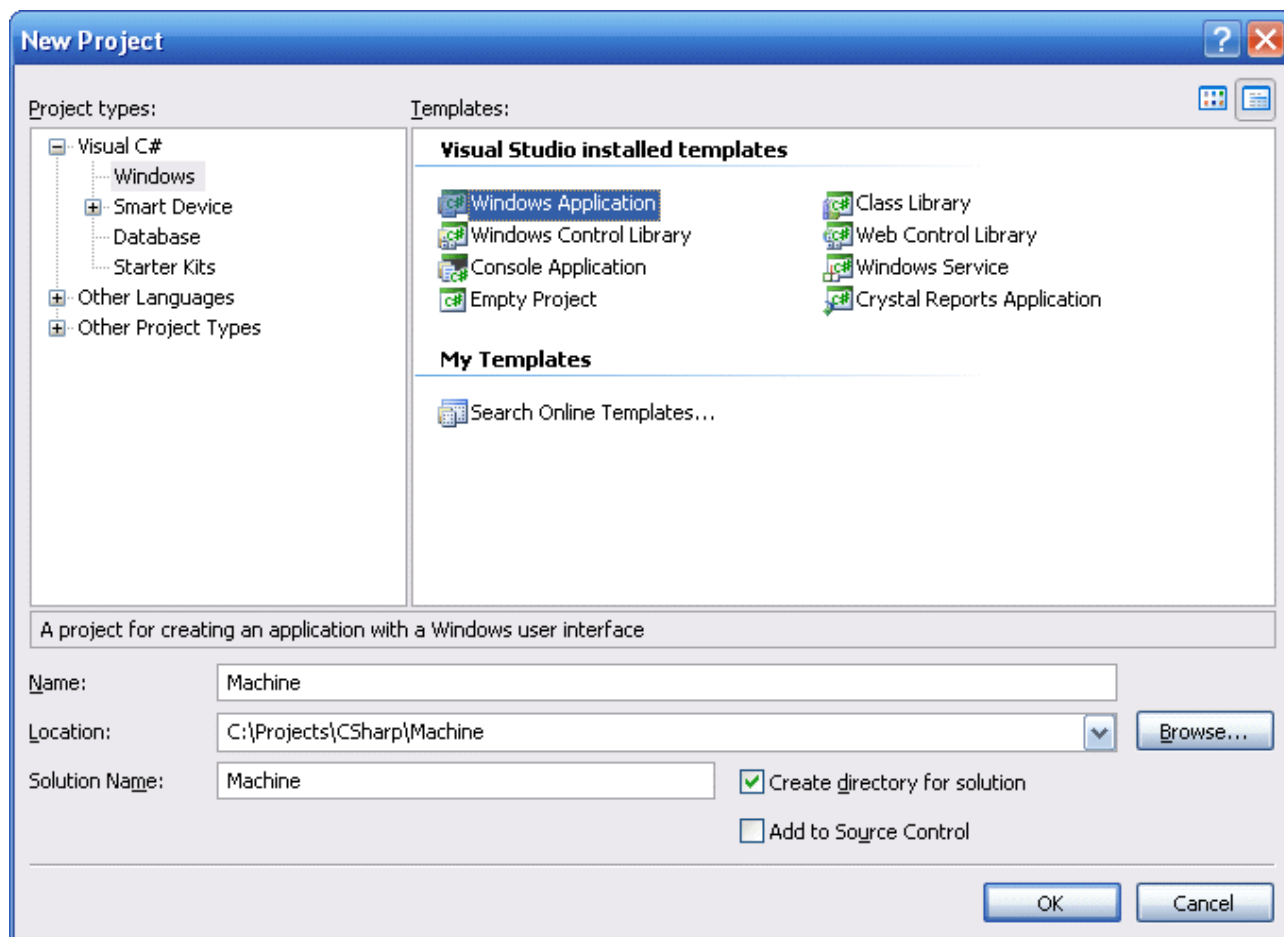
C#プログラムを起動する前に、TwinCATとPLCプログラムが有効でなければなりません。Microsoft Visual Studio 2005がコンピュータにインストールされていない場合は、Microsoft .NET Frameworkバージョン2.0をインストールしてください。これにより、システム上の必要なDLLがセットアップされます。

最初の手順...

C#プログラムの開発とTwinCAT ADS .NETコンポーネントの統合について、サンプルを参照しながら段階的に説明します。

1. 新しいプロジェクトの作成

Visual Studio 2005を起動します。メニューの[File|New|Project]をクリックして、新しいプロジェクトを作成します。[New Project]ダイアログボックスを使用して、さまざまなテンプレートに基づく新しいプロジェクトを作成できます。[Project Types]の下でプログラミング言語Visual C#を選択し、テンプレートの下で[Windows Application]を選択します。プロジェクトの新しい名前を入力します。今回は「Machine」と入力してください。次に、[Location]の下でプロジェクトのディレクトリパスを選択します。



2. ユーザーインターフェイスの作成

まず、ツールボックスを使用して、フォーム「frmMachine」のデザインモードでインターフェイスを作成します。さまざまなコントロール(10個のラベル、2個のラジオボタン、1個のプログレスバー、1個のピクチャボックス、グループボックスなど)の設定を[Visual Studio Properties]ウィンドウで確認し設定することができます。

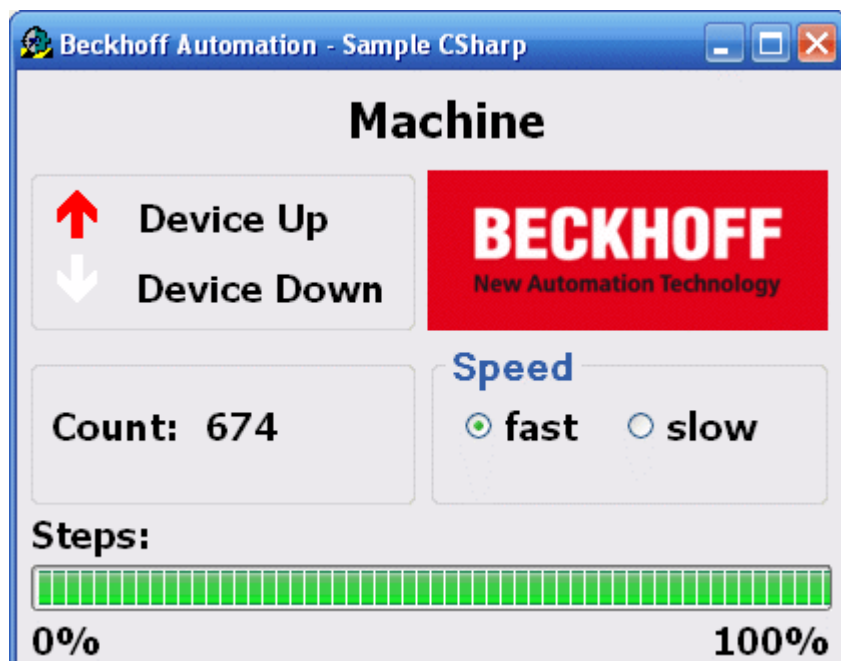
```
// Group fields (group box)
grpDevice.Text = "";
grpCount.Text = "";
grpSpeed.Text = "Speed";

// Labels
lblMachine.Text = "Machine";
lblDeviceDown.Text = "Device Down";
lblDeviceUP.Text = "Device Up";
lblCount.Text = "0";
lblCountLabel.Text = "Count:";
lblSteps.Text = "Steps:";
lbl100Procent.Text = "100%";
lbl0Procent.Text = "0%";

// These are the DeviceDown and DeviceUp arrows with a different font and size (label)
DeviceDown.Text = "ê";
DeviceDown.Font = new System.Drawing.Font("Wingdings", 20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)2));
DeviceUp.Text = "é";
DeviceUp.Font = new System.Drawing.Font("Wingdings", 20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)2));

// Progress bar
prgSteps.Name = "prgSteps";

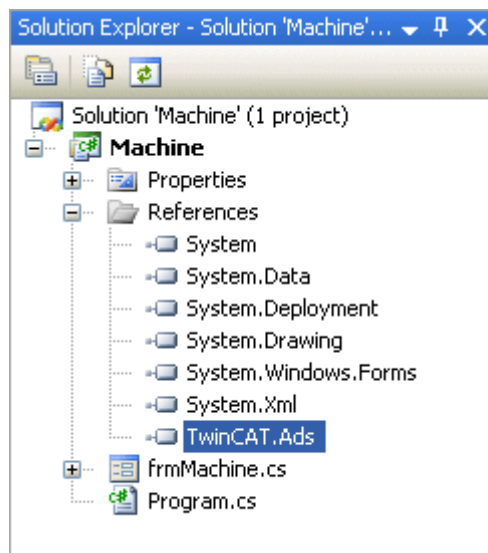
// Option fields (radio button)
optSpeedSlow.Text = "slow";
optSpeedFast.Text = "fast";
```



バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右の[Speed]フィールドを使用して、モータのサイクル速度を変更できます。[Steps]表示には、出力1で出力されるサイクルの数が示されます。

3. リファレンスの追加

まず、TwinCAT.Ads.dllというリファレンスを追加する必要があります。メニューの[Project|Add Reference]を選択すると、TwinCAT ADS .NETコンポーネントが統合されます。リファレンスが実際に統合されたかどうかをチェックするために、ソリューションエクスプローラを利用することができます(**CTRL + W + S**)。



4. ソーステキストの編集

インターフェイスを作成し、TwinCAT ADSコンポーネントを統合したら、C#ソーステキストを変更できます。必要な名前空間「System.IO」および「TwinCAT.Ads」がソーステキストの一番上の行に追加されます。

```
using System.IO;
using TwinCAT.Ads;
```

この後にfrmMachineクラス内の宣言が続きます。

```
privateTcAdsClient tcClient;
privateAdsStream dataStream;
privateBinaryReader binReader;
private int hEngine;
private int hDeviceUp;
private int hDeviceDown;
private int hSteps;
private int hCount;
private int hSwitchNotify;
private int hSwitchWrite;
```

メソッド「frmMachine_Load」はWindowsアプリケーションの呼び出し時に開始されます。このメソッドは、フォームの「Load」イベントを[Properties]ウィンドウのメソッドとリンクすることによってリンクされます。これは、異なるクラスのインスタンスを生成し、ポート801を介してTwinCAT.Adsコンポーネントのランタイムシステム1とのリンクを作成するために使用されます。

```
//-----//
This is called first when the program is started//-----//
---private void frmMachine_Load(object sender, EventArgs e)
{
    try
    {
        // Create a new instance of the AdsStream class
        dataStream = newAdsStream(7);

        // Create a new instance of the BinaryReader class
        binReader = newBinaryReader(dataStream);

        // Create a new instance of the TcAdsClient class
        tcClient = newTcAdsClient();

        // Linking with local PLC - runtime 1 - port 801
        tcClient.Connect(801);
    }
    catch
    {
        MessageBox.Show("Fehler beim Laden");
    }

    //...
}
```

PLC変数のリンク

フォームのfrmMachine_Loadイベントで、PLCの各変数への接続がTcAdsClient.AddDeviceNotification()メソッドによって作成されます。この接続のハンドルは配列に保存されます。TransModeパラメータでデータ交換タイプを指定します。この例では、AdsTransMode.OnChangeを使用して、PLCでの値が変更されている場合のみPLC変数が転送されることを指定します(AdsTransModeを参照)。パラメータcycleTimeで、対応する変数が変更されているかどうかをPLCによってチェックする頻度を決めます。次に、メソッド「tcClient_OnNotification」(すでに書き込まれる必要があります)を使用して変数がリンクされます。このメソッドは、変数が変化したときに呼び出されます。

```
try
{
    // Initialising of PLC variable monitoring
    hEngine = tcClient.AddDeviceNotification(".engine", dataStream, 0, 1, AdsTransMode.OnChange, 10, null);
    hDeviceUp = tcClient.AddDeviceNotification(".deviceUp", dataStream, 1, 1, AdsTransMode.OnChange, 10, 0, null);
    hDeviceDown = tcClient.AddDeviceNotification(".deviceDown", dataStream, 2, 1, AdsTransMode.OnChange, 10, 0, null);
    hSteps = tcClient.AddDeviceNotification(".steps", dataStream, 3, 1, AdsTransMode.OnChange, 10, 0, null);
    hCount = tcClient.AddDeviceNotification(".count", dataStream, 4, 2, AdsTransMode.OnChange, 10, 0, null);
    hSwitchNotify = tcClient.AddDeviceNotification(".switch", dataStream, 6, 1, AdsTransMode.OnChange, 10, 0, null);

    // Retrieving of the "switch" handle. This is required for writing the value
    hSwitchWrite = tcClient.CreateVariableHandle(".switch");

    // Creating an event for changes in the PLC variable values
    tcClient.AdsNotification += newAdsNotificationEventHandler(tcClient_OnNotification);
}
catch (Exception ex)
{
}
```

```

    MessageBox.Show(ex.Message);
}
}

```

定義:

変数をリンクするためにメソッド「AddDeviceNotification()」を使用しました。

```

public int AddDeviceNotification(string variableName, AdsStream dataStream, int offset, int length,
    AdsTransMode transMode, int cycleTime, int maxDelay, object userData);

```

- **variableName:** PLC変数の名前。
- **dataStream:** データを受信するデータストリーム。
- **offset:** ストリームにおけるデータ間隔。
- **length:** ストリームにおけるデータ長。
- **transMode:** 変数が増加する場合のイベント。
- **cycleTime:** 変数が増加したかどうかをPLCサーバがチェックするまでの経過時間(ms)。
- **maxDelay:** イベントが完了するまでの最遅完了時間(ms)。
- **userData:** 特定のデータを保存するために使用できるオブジェクト。

変数「hSwitchWrite」をリンクするためにメソッド「CreateVariableHandle」を使用しました。

```

int TcAdsClient.CreateVariableHandle(string variableName);

```

- **variableName:** PLC変数の名前。

メソッドの書き込み:

上で、まだ存在していないメソッドが参照されました。このメソッド(「tcClient_OnNotification」)が次に書き込まれます。このメソッドは、PLC変数のいずれかが変化した場合に呼び出されます。

```

//-----//
This is called if one of the PLC variables changes//-----
private void tcClient_OnNotification(object sender, AdsNotificationEventArgs e)
{
    try
    {
        // Setting the position of e.DataStream to the position of the current required value
        e.DataStream.Position = e.Offset;

        // Determining which variable has changed if(e.NotificationHandle == hDeviceUp)
        {
            //
            Adapting the colours of the diagrams to match the variables if (binReader.ReadBoolean() == true)
            {
                DeviceUp_LED.ForeColor = Color.Red;
            }
            else
            {
                DeviceUp_LED.ForeColor = Color.White;
            }
        }
        else if(e.NotificationHandle == hDeviceDown)
        {
            if (binReader.ReadBoolean() == true)
            {
                DeviceDown_LED.ForeColor = Color.Red;
            }
            else
            {
                DeviceDown_LED.ForeColor = Color.White;
            }
        }
        else if(e.NotificationHandle == hSteps)
        {
            // Setting the progress bar to the current step
            prgSteps.Value = binReader.ReadByte();
        }
        else if(e.NotificationHandle == hCount)
        {
            // Displaying the "count" value
            lblCount.Text = binReader.ReadUInt16().ToString();
        }
    }
}

```

```

else if(e.NotificationHandle == hSwitchNotify)
{
    // Selecting the correct radio button if (binReader.ReadBoolean() == true)
    {
        optSpeedFast.Checked = true;
    }
    else
    {
        optSpeedSlow.Checked = true;
    }
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
}

```

最後に、機械を「速い(fast)」または「遅い(slow)」に設定するための2つのメソッドが必要です。これらのメソッドを使用してPLC変数「switch」に値を書き込むことで、仮想スイッチを切り替えます。

```

//-----//
This is called if the 'slow' field is selected//-----//
--private void optSpeedFast_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, true);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//-----//
This is called if the 'fast' field is selected//-----//
--private void optSpeedSlow_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
}

```

最後に、正しいイベントに対してメソッド「optSpeedFast_Click」および「optSpeedSlow_Click」が呼び出される必要があります。そのために、[Design]ビューに切り替え、ラジオボタン[optSpeedFast]をクリックし、[Properties]ウィンドウの[Events]の下の[Click]をクリックし、メソッド[optSpeedFast_Click]を選択してください。ラジオボタン[optSpeedSlow]とメソッド[optSpeedSlow_Click]も同様に処理してください。

通知とハンドルの削除:

フォームのfrmMachine_FormClosingイベントで、メソッドDeleteDeviceNotification()を使用してリンクを再び有効にします。

```

//-----//
is called when the program is terminated//-----//
private void frmMachine_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        // Delete notifications and handles
        tcClient.DeleteDeviceNotification(hEngine);
        tcClient.DeleteDeviceNotification(hDeviceUp);
        tcClient.DeleteDeviceNotification(hDeviceDown);
        tcClient.DeleteDeviceNotification(hSteps);
        tcClient.DeleteDeviceNotification(hCount);
        tcClient.DeleteDeviceNotification(hSwitchNotify);

        tcClient.DeleteVariableHandle(hSwitchWrite);
    }
    catch (Exception ex)

```

```
{  
    MessageBox.Show(ex.Message);  
}  
tcClient.Dispose();  
}
```

PLCプログラムMachine_Final.proがランタイムシステム1で起動。また、作成されたC#プログラムMachine.exeをテスト。

C#サンプルプログラムのダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281689995/.zip>

7.4.2 Microsoft Visual Basic .NETによる「Machine」サンプル

Microsoft Visual Studio 2005は、Visual Basicプロジェクトを作成するための開発環境です。プログラミング言語Visual Basicを用いたTwinCAT ADS .NETコンポーネントの統合について、machineサンプルを参照しながら説明します。

必要なソフトウェア:

- Microsoft .NET Frameworkバージョン2.0 (詳細は<http://msdn2.microsoft.com>を参照)
- Microsoft Visual Studio 2005
- TwinCAT 2.10

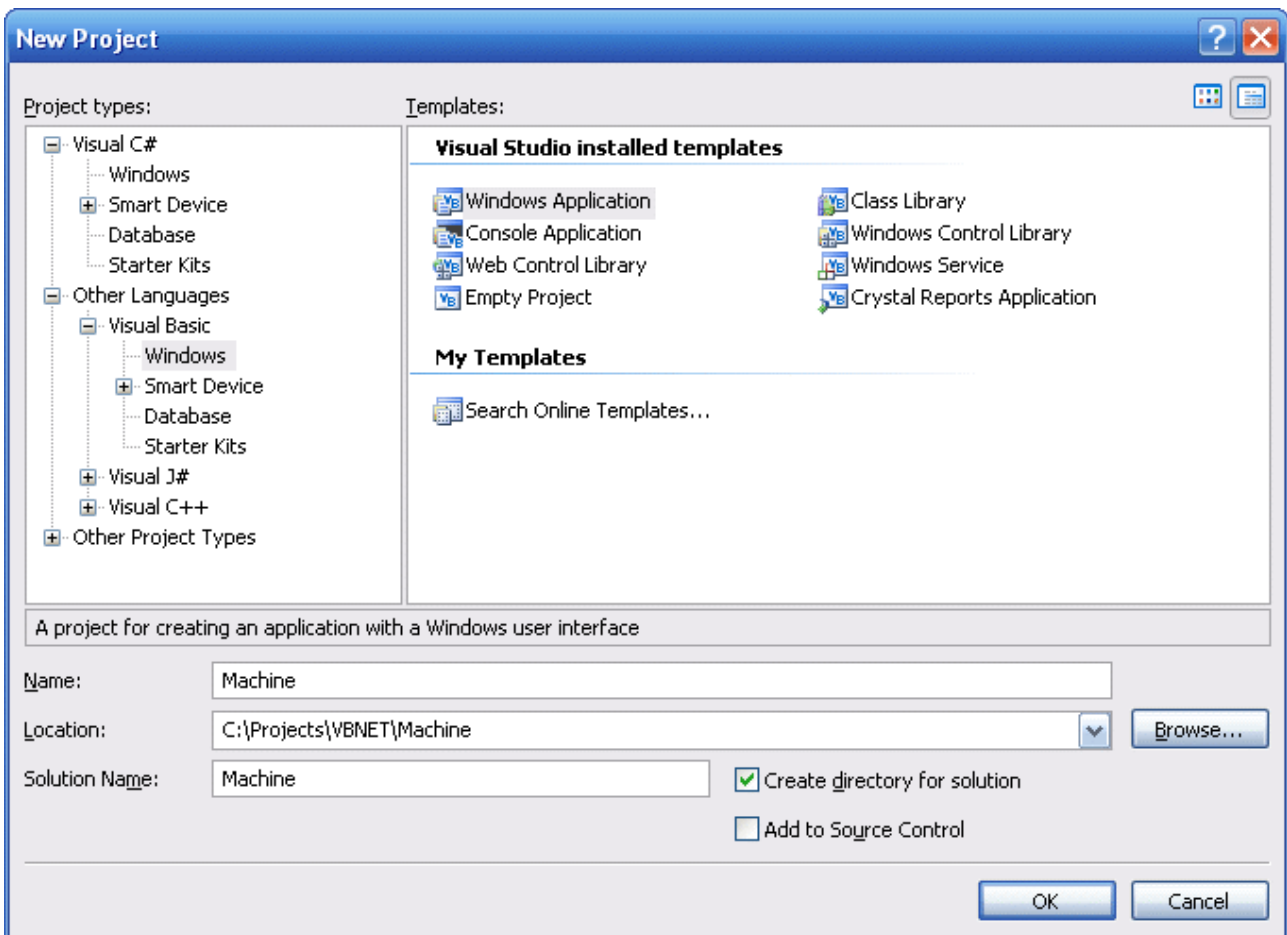
Visual Basicプログラムを起動する前に、TwinCATとPLCプログラムが有効でなければなりません。Microsoft Visual Studio 2005がコンピュータにインストールされていない場合は、Microsoft .NET Frameworkバージョン2.0をインストールしてください。これにより、システム上の必要なDLLがセットアップされます。

最初の手順...

次の手順で、Visual Basicプログラムの開発とTwinCAT ADS .NETコンポーネントの統合について説明します。

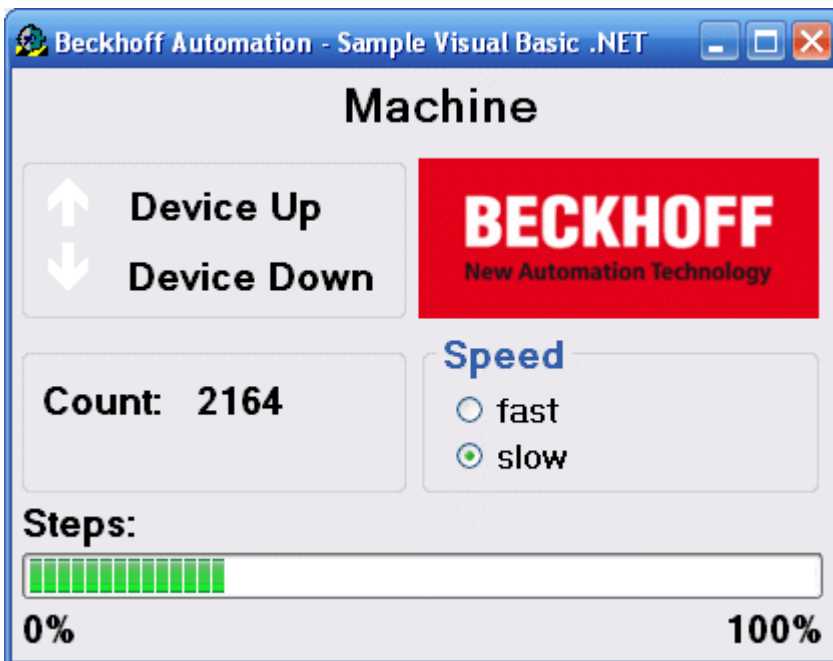
1. 新しいプロジェクトの作成

Visual Studio 2005を起動します。メニューの[File|New|Project]をクリックして、新しいプロジェクトを作成します。[New Project]ダイアログボックスを使用して、さまざまなテンプレートに基づく新しいプロジェクトを作成できます。[Project Types]の下でプログラミング言語[Visual Basic]を選択し、テンプレートの下で[Windows Application]を選択します。プロジェクトの新しい名前を入力します。今回は「Machine」と入力してください。次に、[Location]の下でプロジェクトのディレクトリパスを選択します。



2. ユーザーインターフェイスの作成

まず、ツールボックスを使用して、フォーム「frmMachine」のデザインモードでインターフェイスを作成します。さまざまなコントロール(10個のラベル、2個のラジオボタン、1個のプログレスバー、1個のピクチャボックス、グループボックスなど)の設定を[Visual Studio Properties]ウィンドウで確認し設定することができます。



バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右の[Speed]フィールドを使用して、モータのサイクル速度を変更できます。[Steps]表示には、出力1で出力されるサイクルの数が示されます。

3. リファレンスの追加

まず、TwinCAT.Ads.dllというリファレンスを追加する必要があります。メニューの[Project|Add Reference]を選択すると、TwinCAT ADS .NETコンポーネントが統合されます。

Microsoft Visual Studio .NETでのTwinCAT ADS .NETコンポーネントの統合に関する詳細情報。

4. ソーステキストの編集

インターフェイスを作成し、TwinCAT ADSコンポーネントを統合したら、Visual Basicソーステキストを変更できます。必要な名前空間「System.IO」および「TwinCAT.Ads」がソーステキストの一番上の行に追加されます。

```
Imports System.IO
Imports TwinCAT.Ads
```

この後にfrmMachineクラス内の宣言が続きます。

```
Private tcClient As TwinCAT.Ads.TcAdsClient
Private dataStream As TwinCAT.Ads.AdsStream
Private binReader As System.IO.BinaryReader
Private hEngine As IntegerPrivate hDeviceUp As IntegerPrivate hDeviceDown As IntegerPrivate hSteps As IntegerPrivate hCount As IntegerPrivate hSwitchNotify As IntegerPrivate hSwitchWrite As Integer
```

メソッド「frmMachine_Load」はWindowsアプリケーションの呼び出し時に開始されます。このメソッドは、フォームの「Load」イベントを[Properties]ウィンドウのメソッドとリンクすることによってリンクされます。これは、異なるクラスのインスタンスを生成し、ポート801を介してTwinCAT.Adsコンポーネントのランタイムシステム1とのリンクを作成するために使用されます。

```
'-----
'Wird als erstes beim Starten des Programms aufgerufen
'Is activated first when the program is started
'-----
Private Sub frmMachine_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Try
        ' Create a new instance of the AdsStream class
        dataStream = New AdsStream(7)

        ' Create a new instance of the BinaryReader class
        binReader = New BinaryReader(dataStream)

        ' Create a new instance of the TcAdsClient class
        tcClient = New TwinCAT.Ads.TcAdsClient()

        ' Linking with local PLC - runtime 1 - port 801
        tcClient.Connect(801)

    Catch ex As Exception
        MessageBox.Show("Fehler beim Laden")
    End Try
' ...
```

PLC変数のリンク

フォームのfrmMachine_Loadイベントで、PLCの各変数への接続がTcAdsClient.AddDeviceNotification()メソッドによって作成されます。この接続のハンドルは配列に保存されます。TransModeパラメータでデータ交換タイプを指定します。この例では、AdsTransMode.OnChangeを使用して、PLCでの値が変更されている場合のみPLC変数が転送されることを指定します(AdsTransModeを参照)。パラメータcycleTimeで、対応する変数が変更されているかどうかをPLCによってチェックする頻度を決めます。次に、メソッド「tcClient_OnNotification」(「メソッド「tcClient_OnNotifiacion」の書き込み」を参照)を使用して変数がリンクされます。このメソッドは、変数が変化したときに呼び出されます。

```
Try
    ' Initialisieren der Überwachung der SPS-Variablen
    ' Initializing the monitoring of the PLC variables
    hEngine = tcClient.AddDeviceNotification("engine", dataStream, 0, 1, AdsTransMode.OnChange,
```



```

10, 0, DBNull.Value)
    hDeviceUp = tcClient.AddDeviceNotification(".deviceUp", dataStream, 1, 1, AdsTransMode.OnChange, 10, 0, DBNull.Value)
    hDeviceDown = tcClient.AddDeviceNotification(".deviceDown", dataStream, 2, 1, AdsTransMode.OnChange, 10, 0, DBNull.Value)
    hSteps = tcClient.AddDeviceNotification(".steps", dataStream, 3, 1, AdsTransMode.OnChange, 10, 0, DBNull.Value)
    hCount = tcClient.AddDeviceNotification(".count", dataStream, 4, 2, AdsTransMode.OnChange, 10, 0, DBNull.Value)
    hSwitchNotify = tcClient.AddDeviceNotification(".switch", dataStream, 6, 1, AdsTransMode.OnChange, 10, 0, DBNull.Value)

    ' Holen des Handles von "switch" - wird für das Schreiben des Wertes benötigt
    ' Getting the handle for "switch" - needed for writing the value
    hSwitchWrite = tcClient.CreateVariableHandle(".switch")

    ' Erstellen eines Events für Änderungen an den SPS-Variablen-Werten
    ' Creating an event for changes of the PLC variable values
    AddHandler tcClient.AdsNotification, AddressOf tcClient_OnNotification
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
End Sub

```

変数をリンクするためにメソッド「AddDeviceNotification()」を使用しました。

```

Public Function AddDeviceNotification(ByVal variableName As String, ByVal dataStream As TwinCAT.Ads.AdsStream,
ByVal offset As Integer, ByVal length As Integer, ByVal transMode As TwinCAT.Ads.AdsTransMode,
ByVal cycleTime As Integer, ByVal maxDelay As Integer, ByVal userData As Object)
As Integer

```

- **variableName:** PLC変数の名前。
- **dataStream:** データを受信するデータストリーム。
- **offset:** データストリームにおける間隔。
- **length:** データストリームにおける長さ。
- **transMode:** 変数が変化する場合のイベント。
- **cycletime:** 変数が変化したかどうかをPLCサーバがチェックするまでの経過時間(ms)。
- **maxDelay:** イベントが完了するまでの最遅完了時間(ms)。
- **userData:** 特定のデータを保存するために使用できるオブジェクト。

変数「hSwitchWrite」をリンクするためにメソッド「CreateVariableHandle」を使用しました。

```

Public Function CreateVariableHandle(ByVal variableName As String) As Integer

```

- **variableName:** PLC変数の名前。

メソッド「tcClient_OnNotification」の書き込み:

このメソッドは、PLC変数のいずれかが変化した場合に呼び出されます。

```

Private Sub tcClient_OnNotification(ByVal sender As Object, ByVal e As AdsNotificationEventArgs)
    Try
        ' Setzen der Position von e.DataStream auf die des aktuellen benötigten Wertes
        ' Setting the position of e.DataStream to the position of the current needed value
        e.DataStream.Position = e.Offset

        ' Ermittlung welche Variable sich geändert hat
        ' Detecting which variable has changed
        If e.NotificationHandle = hDeviceUp Then
            'Die Farben der Grafiken entsprechen der Variablen anpassen
            'Adapt colors of graphics according to the variables
            If binReader.ReadBoolean() = True Then
                DeviceUp_LED.ForeColor = Color.Red
            Else
                DeviceUp_LED.ForeColor = Color.White
            End If
        ElseIf e.NotificationHandle = hDeviceDown Then
            If binReader.ReadBoolean() = True Then
                DeviceDown_LED.ForeColor = Color.Red
            Else
                DeviceDown_LED.ForeColor = Color.White
            End If
        ElseIf e.NotificationHandle = hSteps Then

```

```

' Einstellen der ProgressBar auf den aktuellen Schritt
' Setting the ProgressBar to the current step
prgSteps.Value = binReader.ReadByte()
ElseIf e.NotificationHandle = hCount Then
' Anzeigen des "count"-Werts
' Displaying the "count" value
lblCount.Text = binReader.ReadUInt16().ToString()
ElseIf e.NotificationHandle = hSwitchNotify Then
' Markieren des korrekten RadioButtons
' Checking the correct RadioButton
If binReader.ReadBoolean() = True Then
optSpeedFast.Checked = True
Else
optSpeedSlow.Checked = True
End If

End If
Catch ex As Exception
MessageBox.Show(ex.Message)
End Try
End Sub

```

最後に、機械を速くまたは遅くするための2つのメソッドが必要です。これらのメソッドを使用してPLC変数「switch」に値を書き込むことで、仮想スイッチを切り替えます。

```

' Schreiben des Wertes von "switch"
' Writing the value of "switch"

'-----
'wird aufgerufen, wenn das Feld 'fast' markiert wird
'is activated when the 'fast' field is marked
'-----
Private Sub optSpeedFast_Click(ByVal sender As Object, ByVal e As EventArgs) Handles optSpeedFast.Click
Try
' Schreiben des Wertes von "switch"
' Writing the value of "switch"
tcClient.WriteAny(hSwitchWrite, True)
Catch ex As Exception
MessageBox.Show(ex.Message)
End Try
End Sub

'-----
'wird aufgerufen, wenn das Feld 'slow' markiert wird
'is activated when the 'slow' field is marked
'-----
Private Sub optSpeedSlow_Click(ByVal sender As Object, ByVal e As EventArgs) Handles optSpeedSlow.Click
Try
tcClient.WriteAny(hSwitchWrite, False)

Catch ex As Exception
MessageBox.Show(ex.Message)
End Try
End Sub

```

正しいイベントに対してメソッド「optSpeedFast_Click」および「optSpeedSlow_Click」が呼び出される必要があります。そのために、[Design]ビューに切り替え、ラジオボタン[optSpeedFast]をクリックし、[Properties]ウィンドウの[Events]の下の[Click]をクリックし、メソッド[optSpeedFast_Click]を選択してください。ラジオボタン[optSpeedSlow]とメソッド[optSpeedSlow_Click]も同様に処理してください。

通知とハンドルの削除:

フォームのfrmMachine_FormClosingイベントで、メソッドDeleteDeviceNotification()を使用してリンクを再び有効にします。

```

'-----
'wird beim Beenden des Programms aufgerufen
'is activated when ending the program
'-----
Private Sub frmMachine_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
Try
' Löschen der Notifications und Handles
' Deleting of the notifications and handles

```

```

tcClient.DeleteDeviceNotification(hEngine)
tcClient.DeleteDeviceNotification(hDeviceUp)
tcClient.DeleteDeviceNotification(hDeviceDown)
tcClient.DeleteDeviceNotification(hSteps)
tcClient.DeleteDeviceNotification(hCount)
tcClient.DeleteDeviceNotification(hSwitchNotify)

tcClient.DeleteVariableHandle(hSwitchWrite)
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
tcClient.Dispose()
End Sub

```

この時点で、ランタイムシステム1でPLCプログラムMachine_Final.proを起動し、作成したVisual BasicプログラムMachine.exeを起動することができます。

Visual Basicサンプルのダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281691403/.zip>

7.4.3 Microsoft Visual Basic 6.0を使用した機械サンプル

TwinCAT ADS OCX

TwinCATには、アプリケーション固有のプログラムをシステムに統合するための複数のプログラミングインターフェイスがあります。Microsoft Visual Basicは、サポートされているプログラミング言語の1つです。このプログラミング言語の長所は、グラフィカルユーザインターフェイスが作成されることと、そのインターフェイスをデータベースにリンクすることが可能であるという事実です。Visual Basicは数年で広く普及し、Microsoftによって絶えず強化されています。多くのサードパーティベンダが、さまざまな分野向けのアドオンモジュールを提供しています。このようなモジュールの総称がOCXです。ベッコフは、AdsOCXという名前のTwinCAT向けOCXを提供しています。AdsOCXは、他のADSデバイス(PLC、NC/CNCなど)とのTwinCATメッセージルータ経由の通信のための方法です。含まれているサンプルプログラムは、Visual BasicからTwinCAT PLCサーバの個々の変数にアクセスする方法を示しています。

アクセスメソッド:

AdsOCXには、他のADSデバイスから値を読み取るためのさまざまなメソッドが含まれています。使用するべきメソッドの決定は、プログラムを実行する環境によって左右されます。その他の注意事項と個々のファンクションに関する詳細情報が、特別なAdsOCX取扱説明書に記載されています。ここでは、個々のアクセスメソッドの概要だけを示しています。

アクセス	意味
connect	PLC変数とVisual Basic変数の間の通信が必要になるとすぐに、これら2つの変数間の接続はメソッドの軌道によって確立されます。プログラムの進行中に、TwinCATはVisual Basic変数をPLC変数に適応させます。このタイプのデータ交換は、Visual BasicプログラムでPLC変数が変化する場合にイベントファンクション(イベントによって制御されるデータ転送)を有効にするためにも使用できます。
synchronous	read/writeメソッドを有効にした後、要求されたデータが到着するまでVisual Basicプログラムの実行が中断されます。到着すると、プログラムが新しいデータで動作を続けることができます。
asynchronous	asynchronousアクセスを使用すると、Visual Basicプログラムは中断されず、次のコマンドの実行が自動的に続けられます。要求されたデータがAdsOCXで到着すると、イベントファンクションがVisual Basicプログラムで実行され、値がパラメータとして渡されます。

サンプルプログラム

TwinCATとPLCプログラムの起動:

Visual Basicプログラムを起動する前に、TwinCATとPLCプログラムが有効でなければなりません。

Visual Basicプログラムの起動:

「maschine.exe」プログラムを起動します。(TwinCAT¥Samples¥First Steps¥)



バスターミナルにも出力される2つの出力が左上に表示されます。ワークピースをカウントする変数が左下に表示されます。モータのサイクル速度を[Speed]ボックスで変更できます。[position]表示には、出力1に出力されるサイクルの数が示されます。

ソースコード:

```
Option Explicit

Dim deviceUp As Boolean
Dim deviceDown As Boolean
Dim steps As Integer
Dim counter As Long
Dim hDeviceUp As Long
Dim hDeviceDown As Long
Dim hSteps As Long
Dim hSwitch As Long
Dim hCounter As Long
'-----'Is activated first when the program is started'-----
Private Sub Form_Load()

    'load language dependent words from the resource-file
    lblMachine.Caption = LoadResString(0 + GetLanguageId)
    lplDeviceUp.Caption = LoadResString(1 + GetLanguageId)
    lplDeviceDown.Caption = LoadResString(2 + GetLanguageId)
    lblCountLabel.Caption = LoadResString(3 + GetLanguageId)
    lplSteps.Caption = LoadResString(4 + GetLanguageId)
    fraSpeed.Caption = LoadResString(5 + GetLanguageId)
    optSpeedFast.Caption = LoadResString(6 + GetLanguageId)
    optSpeedSlow.Caption = LoadResString(7 + GetLanguageId)

    'Connect PLC variables with VB variables
    Call AdsOcx1.AdsReadIntegerVarConnect(".steps", 2&, 4, 55, steps)
    Call AdsOcx1.AdsReadBoolVarConnect(".deviceUp", 2&, 4, 55, deviceUp)
    Call AdsOcx1.AdsReadBoolVarConnect(".deviceDown", 2&, 4, 55, deviceDown)
    Call AdsOcx1.AdsReadLongVarConnect(".count", 4&, 4, 55, counter)

    'Determine handle of the variables
    Call AdsOcx1.AdsCreateVarHandle(".steps", hSteps)
    Call AdsOcx1.AdsCreateVarHandle(".deviceUp", hDeviceUp)
    Call AdsOcx1.AdsCreateVarHandle(".deviceDown", hDeviceDown)
    Call AdsOcx1.AdsCreateVarHandle(".count", hCounter)
    Call AdsOcx1.AdsCreateVarHandle(".switch", hSwitch)
End Sub

'-----'is activated when ending the program'-----
```

```

-----
Private Sub Form_Unload(Cancel As Integer)
    'Separate Visual Basic variables from PLC variables
    Call AdsOcx1.AdsReadIntegerDisconnect(steps)
    Call AdsOcx1.AdsReadBoolDisconnect(deviceUp)
    Call AdsOcx1.AdsReadBoolDisconnect(deviceDown)
    Call AdsOcx1.AdsReadLongDisconnect(counter)

    'Release handle of the variables
    Call AdsOcx1.AdsDeleteVarHandle(hSteps)
    Call AdsOcx1.AdsDeleteVarHandle(hDeviceUp)
    Call AdsOcx1.AdsDeleteVarHandle(hDeviceDown)
    Call AdsOcx1.AdsDeleteVarHandle(hCounter)
    Call AdsOcx1.AdsDeleteVarHandle(hSwitch)
End Sub

'-----'is activated when the 'fast' field is marked
-----
Private Sub optSpeedFast_Click()
    Dim switch As Boolean
    'set PLC variable switch to TRUE
    switch = True
    Call AdsOcx1.AdsSyncWriteBoolVarReq(hSwitch, 2&, switch)
End Sub

'-----'is activated when the 'slow' field is marked
-----
Private Sub optSpeedSlow_Click()
    Dim switch As Boolean
    'set PLC variable switch to FALSE
    switch = False
    Call AdsOcx1.AdsSyncWriteBoolVarReq(hSwitch, 2&, switch)
End Sub

'-----'is activated when a PLC variable changes'-----
-----
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
    Select Case nIndexOffset
        Case hCounter:
            'Display quantity in form
            lblCount.Caption = counter
        Case hDeviceUp
            'Adapt colors of graphics according to the variables
            DeviceUp_LED.ForeColor = IIf(deviceUp = True, vbRed, vbWhite)
        Case hDeviceDown
            'Adapt colors of graphics according to the variables
            DeviceDown_LED.ForeColor = IIf(deviceDown = True, vbRed, vbWhite)
        Case hSteps
            'Display position of workpiece
            prgSteps.Width = steps * 240
    End Select
End Sub

```

動作原理:

Form1_Load()では言語依存テキストがリソースとしてロードされます。このテキストは、Windows NTで事前に選択された言語に従って表示されます。メソッド「AdsReadVarConnect」によって、適切なVB変数がPLCシステムに接続されます。メソッドのパラメータは以下のとおりです。

パラメータ	意味
adsVarName	PLC変数の名前
cbLenght	データの長さ(バイト)
nRefreshType	データ転送のタイプ
nCycleTime	リフレッシュサイクル(ms)
pData	データが書き込まれるVisual Basic変数

変数「count」、「deviceUp」、「deviceDown」、または「steps」が変化するたびに、AdsOcx1_AdsReadConnectUpdate()ファンクションが有効になります。オブジェクトはPLCの変数とともにフォームで有効になります。

ユーザが[Fast]または[Slow]マーキングボックスをクリックするたびに、optFast_Click()ファンクションとoptSlow_Click()ファンクションが有効になります。これらのファンクションで、「switch」変数がTRUEまたはFALSEに設定されます。TwinCAT PLCサーバがこの変数を読み出し、フラッシングの動作を適切に変更します(PLCプログラムの「DriveType」ファンクションブロック)。

パラメータ	意味
hVar	PLC変数のハンドル
cbLenght	データの長さ(バイト)
pData	データを含むVisual Basic変数

フォームが閉じられると、不要になった変数がすべて、PLC変数から切り離されるはずですが、これはForm1_Unload()ファンクションで行われます。

サンプルのダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281692811/.zip>

7.4.4 Microsoft Visual C++ による「Machine」サンプル

Microsoft Visual Studio 2005は、C++プロジェクトを作成するための開発環境です。プログラミング言語C++を用いたTwinCAT ADS DLL .LIBコンポーネントの統合について、machineサンプルを参照しながら説明します。

必要なソフトウェア:

- Microsoft .NET Frameworkバージョン2.0、詳細については[ここをクリックしてください](#)。
- Microsoft Visual Studio 2005
- TwinCAT 2.10

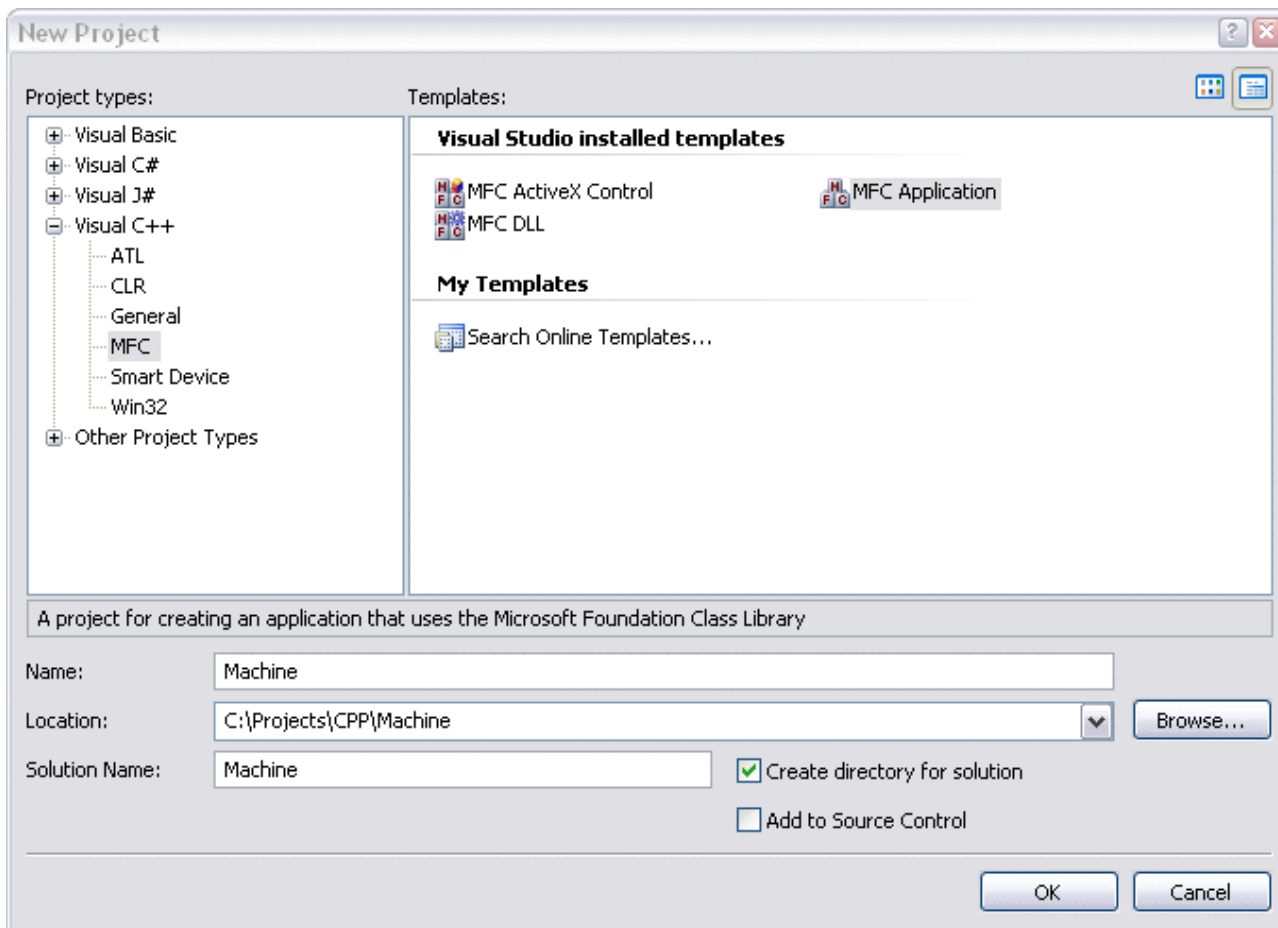
C++プログラムを起動する前に、TwinCATとPLCプログラムが有効でなければなりません。Microsoft Visual Studio 2005がコンピュータにインストールされていない場合は、Microsoft .NET Frameworkバージョン2.0をインストールしてください。これにより、システム上の必要なDLLがセットアップされます。

最初の手順...

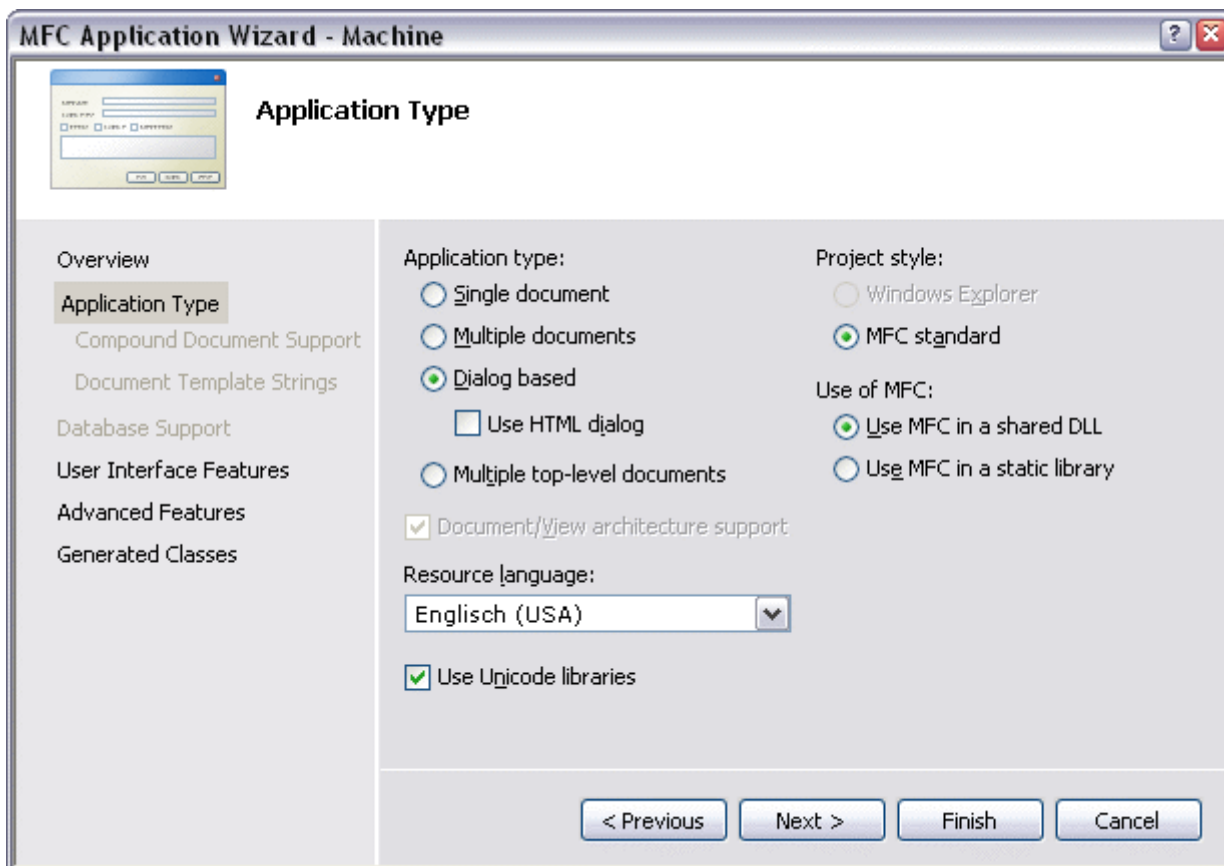
C++プログラムの開発とTwinCAT ADS DLL .LIBコンポーネントの統合について、サンプルを参照しながら段階的に説明します。

1. 新しいプロジェクトの作成

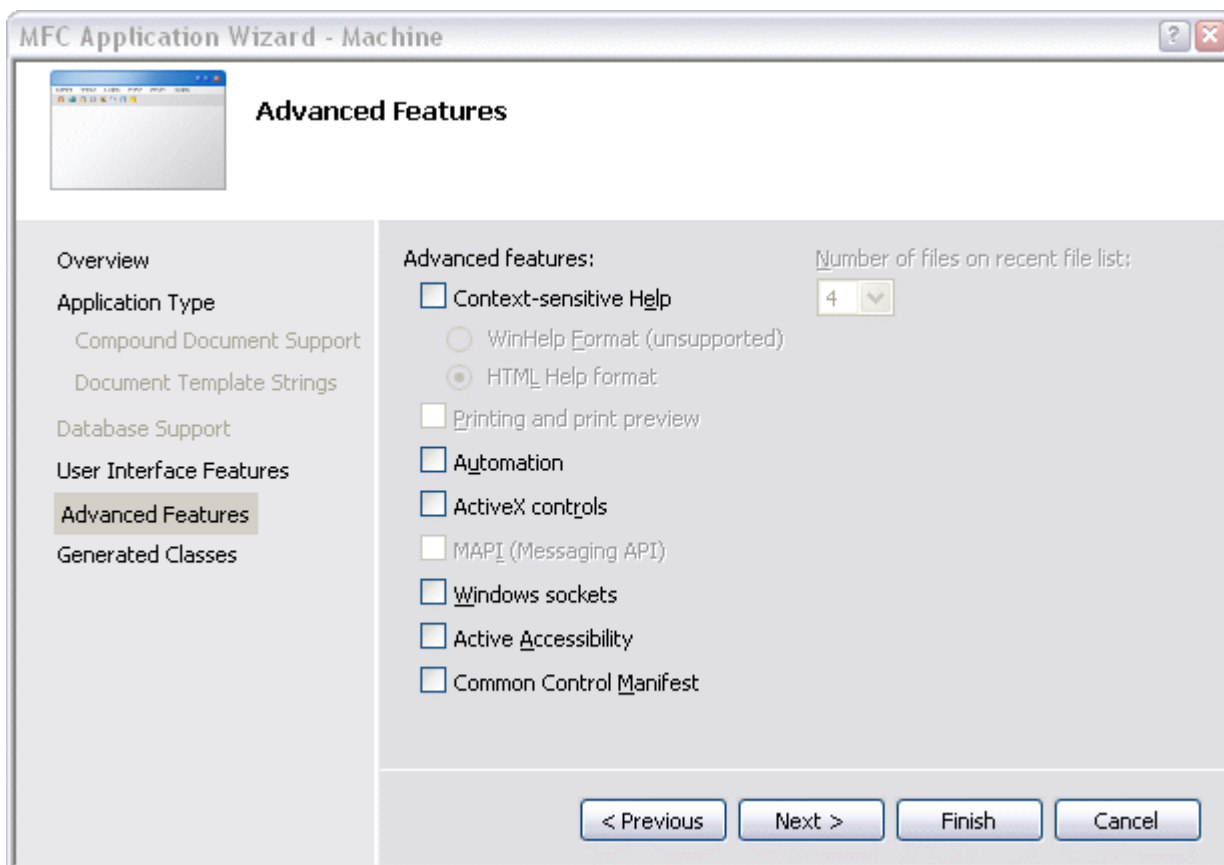
メニューの[File|New|Project...]をクリックして、新しいプロジェクトを作成します。[New Project]ダイアログボックスを使用して、さまざまなテンプレートに基づく新しいプロジェクトを作成できます。[Project Types]の下でプログラミング言語[Visual C++]、[MFC]を選択し、テンプレートの下で[MFC Application]を選択します。プロジェクトの新しい名前を入力します。今回は「Machine」と入力してください。次に、[Location]の下でプロジェクトのディレクトリパスを選択します。



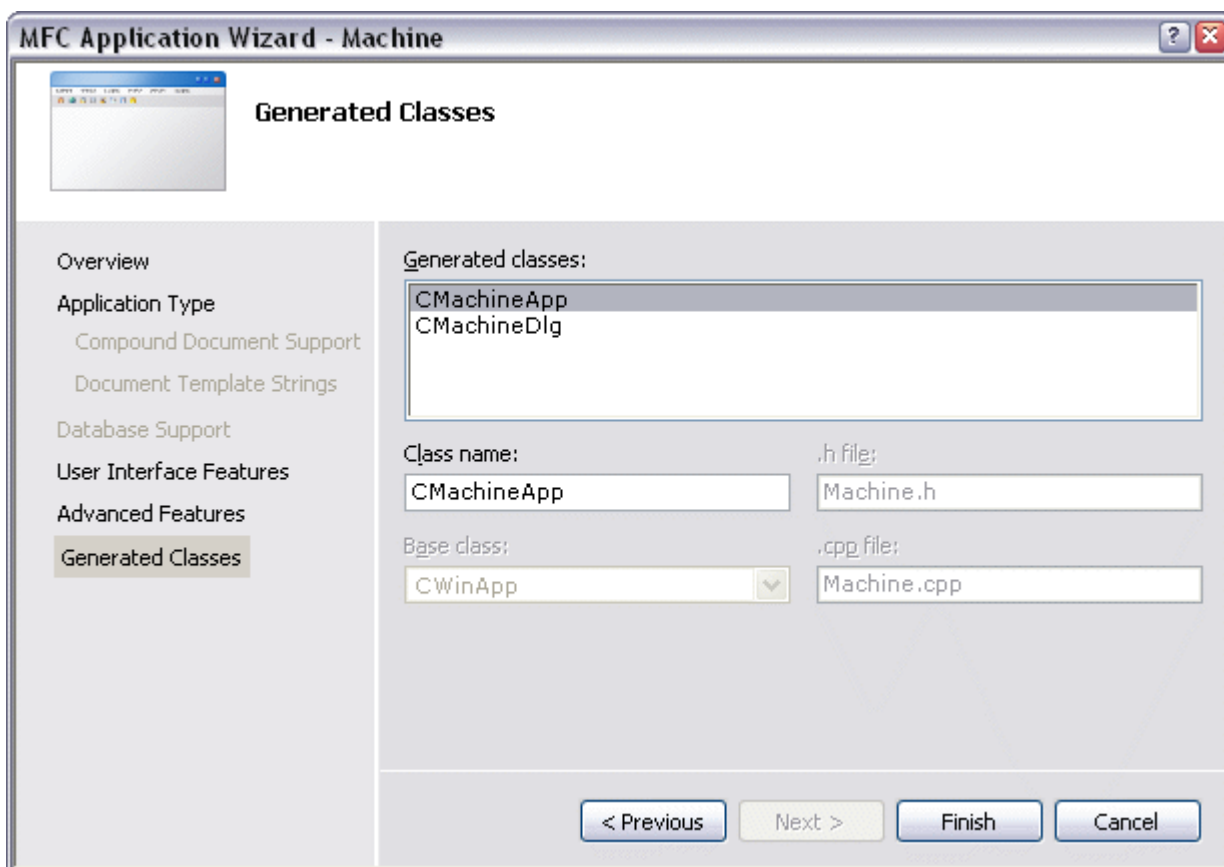
[Wizard]ウィンドウが開いたら、次の設定を受け入れてください。



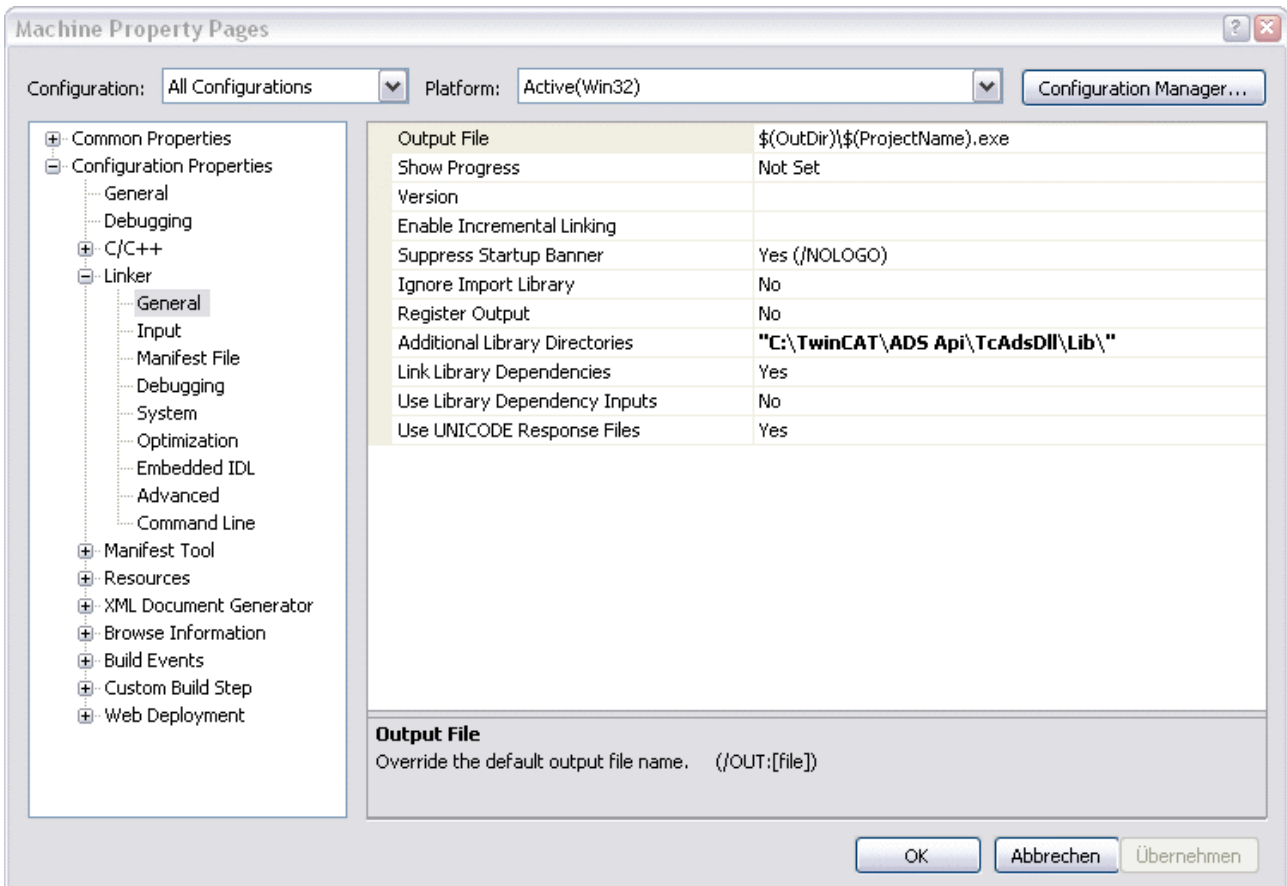
その他の機能は必要ありません。



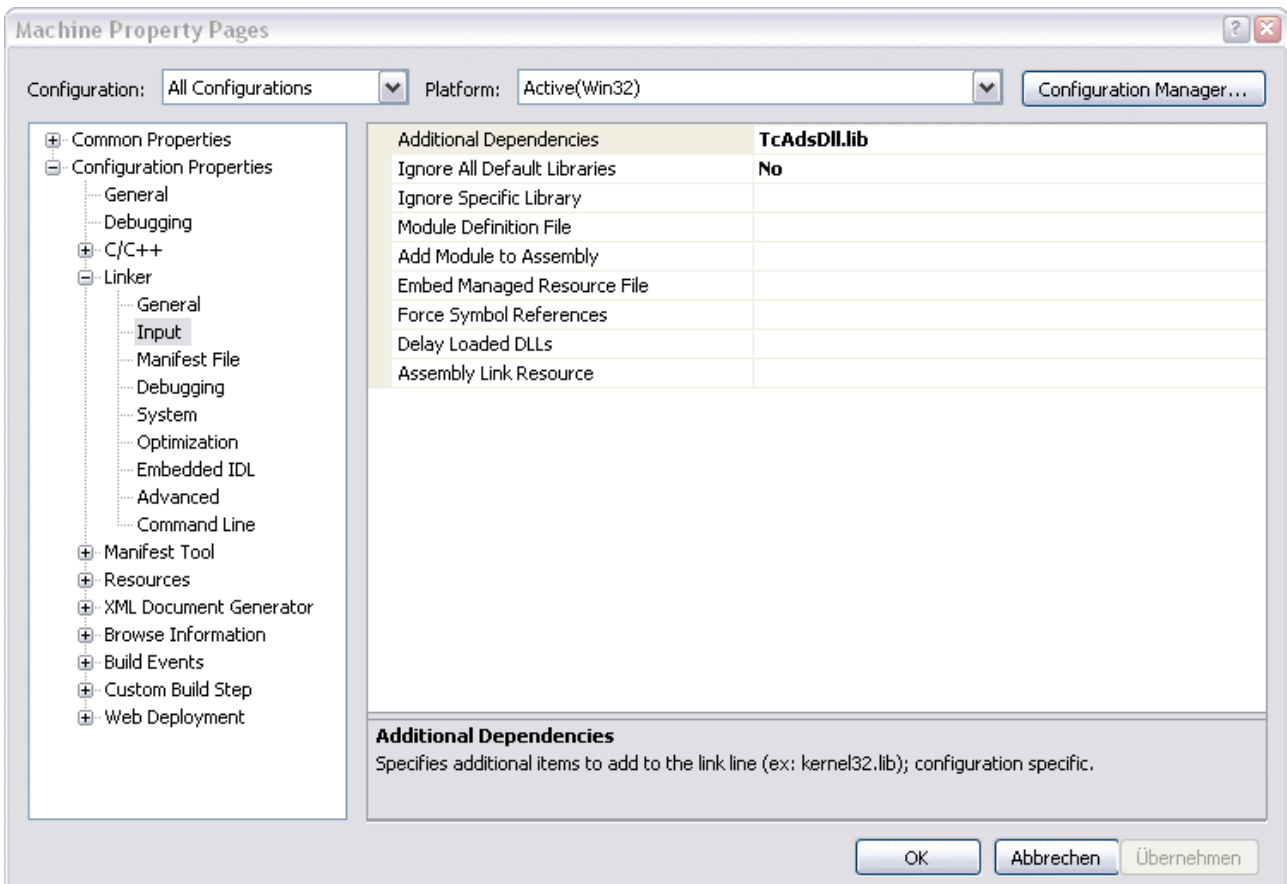
ウィザードでの最後の手順。ここで変更は必要ありません。



最後に、必要なTcAdsDll.libをリンクします。[Project|Machine Properties...|Configuration Properties|Linker|General|Additional Library Directories]の下でパスを入力します。



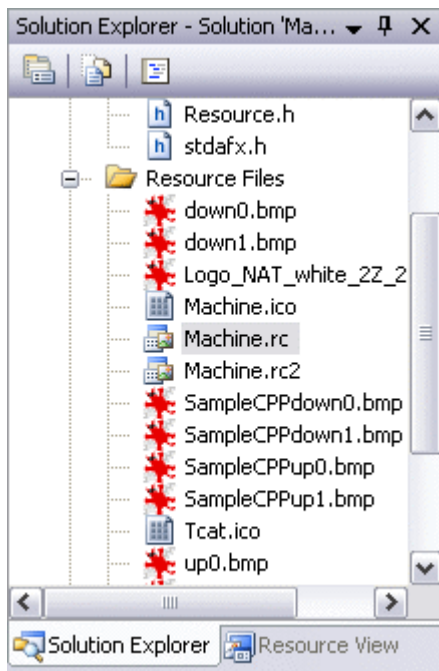
追加エントリが[Project|Machine Properties...|Configuration Properties|Linker|Input|Additional Dependencies]の下で必要です。



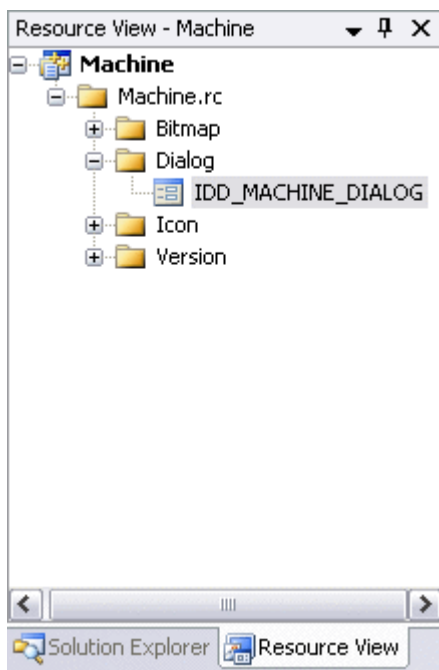
左上の[Configuration:]で[All Configurations]が選択されていることを確認してください。

2. ユーザーインターフェイスの作成

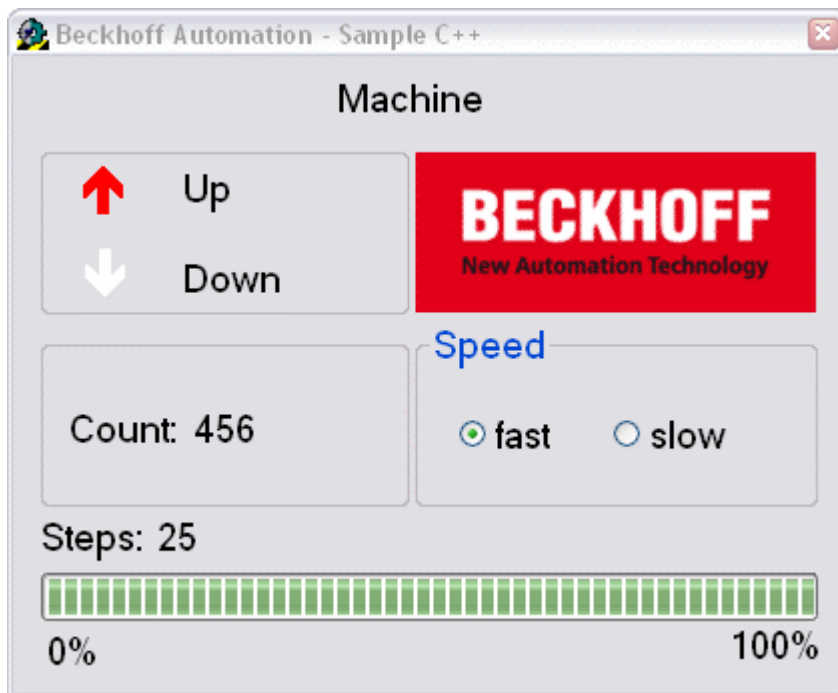
まず、ツールボックスを使用して、フォーム「Machine.rc」のデザインモードでインターフェイスを作成します。[Solution Explorer]で[Machine.rc]をダブルクリックして開きます。



[Resource View]が表示されます。[IDD_MACHINE_DIALOG]をダブルクリックすると、すべての必要なコントロール(9個のスタティックテキスト、3個のピクチャボックス、2個のラジオボタン、1個のプログレスバー、グループボックスなど)を配置するためのフォームが表示されます。



結果はこのようになるはずですが。



バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右の[Speed]フィールドを使用して、モータのサイクル速度を変更できます。[Steps]表示には、出力1で出力されるサイクルの数が示されます。

3. ソーステキストの編集

インターフェイスを作成し、TwinCAT DLL.LIBコンポーネントを統合したら、C++ソーステキストを変更できます。「MachineDlg.h」のソーステキストの下部で宣言を実装します。

```
[...]

// Variablen des Programms// Variables of the programprivate:
// Variablen der Bitmaps// Variables of the bitmaps
CBitmap CbmUp0, CbmUp1, CbmDown0, CbmDown1;
private:
// Timerfunktion und Variablen// Timerfunction and variablesvoid OnTimer(UINT nIDEvent);
UINT_PTR m_nTimer;
UINT nIDEvent;
private:
// Membervariable der Statusanzeige// Member variable of the progressbar
CProgressCtrl m_ctrlBar;
private:
// Membervariablen der Controls// Member variables of the controls
CStatic m_ctrlCount;
CStatic m_ctrlSteps;
CStatic m_ctrlEngine;
CStatic m_Up;
CStatic m_Down;
CStatic m_rdoFast;
private:
// Implementierung der Radio Buttons// Implementation of the radio buttons
afx_msg void OnBnClickedRadio1();
afx_msg void OnBnClickedRadio2();
private:
// Allgemeine Variablendeklarationen// General declarations of variables
CButton *pRB;
short sCount;
byte bySteps;
CString sOutput;
AmsAddr Addr;
PamsAddr pAddr;
long nErr, nSwitch, nNotify, nPort;
bool bEngine, bUp, bDown, bSwitch, bNotify;
ULONG hEngine, hDeviceUp, hDeviceDown, hSteps, hCount, hSwitch, hNotify;
};
```

MFCアプリケーションの起動時にメソッド「CMachineDlg::OnInitDialog()」が呼び出されます。このメソッドの用途は、さまざまなクラスのインスタンスの生成、ポート801経由でのコンポーネントTcAdsDll.libのランタイムシステム1との接続の確立、必要なビットマップのロード、PLC変数用のハンドルの作成です。

```

BOOL CMachineDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    [...]

    // Kommunikationsport für lokale SPS (Laufzeitsystem 1) öffnen// Open the communication port for
the SPS (Runtime system 1)
    pAddr = &Addr;
    nPort = AdsPortOpen();
    nErr = AdsGetLocalAddress(pAddr);
    pAddr->port = AMSPORT_R0_PLC_RTS1;

    // Timer initialisieren// Timer initialization
    m_nTimer = CDialog::SetTimer( 1, 40, NULL );
    m_ctrlBar.SetRange( 0, 25 );

    // Laden der Bitmaps// Loading of the bitmaps
    CbmUp0.LoadBitmapW( IDB_UP0 );
    CbmUp1.LoadBitmapW( IDB_UP1 );
    CbmDown0.LoadBitmapW( IDB_DOWN0 );
    CbmDown1.LoadBitmapW( IDB_DOWN1 );

    // Einen Pointer auf ein CButton Objekt holen// Get a pointer to CButton Object
    pRB = (CButton*)GetDlgItem( IDC_rdoFast );

    // Setzt den Radio Button auf "checked"// Set the radio button on checked state
    pRB->SetCheck(1);

    // Abfrage vom SPS-Switch-Status// Inquiry of the SPS switch status
    nNotify = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hNotify, 8, ".switch" );
    nNotify = AdsSyncReadReq ( pAddr, ADSIGRP_SYM_VALBYHND, hNotify, 8, &bNotify );
    if ( bNotify )
    {
        pRB = ( CButton* )GetDlgItem( IDC_rdoFast ); pRB->SetCheck(1);
        pRB = ( CButton* )GetDlgItem( IDC_rdoSlow ); pRB->SetCheck(0);
    }
    else
    {
        pRB = ( CButton* )GetDlgItem( IDC_rdoFast ); pRB->SetCheck(0);
        pRB = ( CButton* )GetDlgItem( IDC_rdoSlow ); pRB->SetCheck(1);
    }
    nNotify = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hNotify );

    // Handles für die SPS-Variablen holen// Get handles for the SPS variables
    nErr = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hDeviceUp, 10, ".deviceUp" );
    nErr = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hDeviceDown, 12, ".deviceDown" );
    nErr = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hSteps, 7, ".steps" );
    nErr = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hCount, 7, ".count" );
    nNotify = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hNotify, 8, ".switch" );

    return TRUE; // return TRUE unless you set the focus to a control
}

```

PLC変数の出力:

変数出力は「CMachineDlg::OnTimer()」で準備されます。メソッド「SetTimer()」のパラメータ「nElapse」（「CMachineDlg::OnInitDialog()」で既に指定されている）によって、タイマが再び実行されるまでに経過する必要がある時間(ms)が決まります。

```

void CMachineDlg::OnTimer(UINT nIDEvent)
{
    //--> Ausgabe der SPS-Variablen <--/////--
    > Output of the SPS variables <--//////////////////////////////////////
    //////////////////////////////////////// Auslesen der SPS-Variablen// Reading the SPS variables
    nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, hDeviceUp, 10, &bUp );
    // Bitmap für den "Up"-Zustand setzen// Sets the Bitmap of the "Up" positionif( bUp == 0 )

```

```

{ m_Up.SetBitmap(CbmUp0); }
else{ m_Up.SetBitmap(CbmUp1); }
//-----//
nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, hDeviceDown, 12, &bDown );
// Bitmap für den "Down"-
Zustand setzen// Sets the Bitmap of the "Down" positionif( bDown == 0 )
{ m_Down.SetBitmap(CbmDown0); }
else{ m_Down.SetBitmap(CbmDown1); }
//-----//
nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, hSteps, 7, &bySteps );
// Anzahl der Schritte in das Control schreiben// Writes the number of steps in the control
sOutput.Format( _T("%d"), bySteps );
m_ctrlSteps.SetWindowText( sOutput );
m_ctrlBar.SetPos( bySteps );
//-----//
nErr = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, hCount, 7, &sCount );
// Zählerstand in das Control schreiben// Writes the count in the control
sOutput.Format( _T("%i"), sCount );
m_ctrlCount.SetWindowText( sOutput );
//-----// A
npassen der "Switch"-
Variable, bei Veränderung in der SPS// Fits the switch variable if it is changing
nNotify = AdsSyncReadReq( pAddr, ADSIGRP_SYM_VALBYHND, hNotify, 8, &bNotify );
if ( bNotify )
{ pRB = (CButton*)GetDlgItem( IDC_rdoFast ); pRB->SetCheck(1);
  pRB = (CButton*)GetDlgItem( IDC_rdoSlow ); pRB->SetCheck(0);
}
else
{ pRB = (CButton*)GetDlgItem( IDC_rdoFast ); pRB->SetCheck(0);
  pRB = (CButton*)GetDlgItem( IDC_rdoSlow ); pRB->SetCheck(1);
}
nNotify = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hNotify );
////////////////////////////////////
CDialog::OnTimer( nIDEvent );
};

```

変数をリンクするためにメソッド「AdsSyncReadWriteReq()」を使用しました。

```

long AdsSyncReadWriteReq(
PAmAddr pAddr, ULONG nIndexGroup, ULONG nIndexOffset, ULONG nReadLength, PVOID pReadData, ULONG nWriteLength,
PVOID pWriteData );

```

- **pAddr:** ADSサーバのNetIdとポート番号を含む構造体。
- **nIndexGroup:** インデックスグループ。
- **nIndexOffset:** インデックスオフセット。
- **nReadLength:** ADSデバイスによって返されるデータの長さ(バイト)。
- **pReadData:** ADSデバイスによって返されるデータを含むバッファ。
- **nWriteLength:** ADSデバイスに書き込まれるデータの長さ(バイト)。
- **pWriteData:** ADSデバイスに書き込まれるデータを含むバッファ。

変数「hSwitch」をリンクするためにもメソッド「AdsSyncReadWriteReq()」を使用しました。唯一の違いは、「AdsSyncReadReq()」(変数の読み取り)が「AdsSyncWriteReq()」(変数の書き込み)の代わりに使用されることです。

```

long AdsSyncReadReq( PAmAddr pAddr, ULONG nIndexGroup, ULONG nIndexOffset, ULONG nLength, PVOID pData );

```

- **pAddr:** ADSサーバのNetIdとポート番号を含む構造体。
- **nIndexGroup:** インデックスグループ。
- **nIndexOffset:** インデックスオフセット。
- **nLength:** ADSサーバに書き込まれるデータの長さ(バイト)。
- **pData:** ADSサーバに書き込まれるデータへのポインタ。

最後に、機械の速度を制御するための2つのメソッドが必要です。制御するには、[Design]ビューに切り替え、[Fast]と[Slow]のラジオボタンをダブルクリックし、関連するコードを入力するだけで済みます。

[Fast]ラジオボタンのコード。

```
void CMachineDlg::OnBnClickedRadio1 ()
{
    // Anpassen der "Switch"-
    Variable, bei klick auf Radio Button "fast"// Fits the switch variable if the radio button "fast" was
    s clicked
    bSwitch = true;
    nSwitch = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hSwitch, 8, ".
    switch" );
    nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_VALBYHND, hSwitch, 0x1, &bSwitch );
    nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hSwitch );
}

```

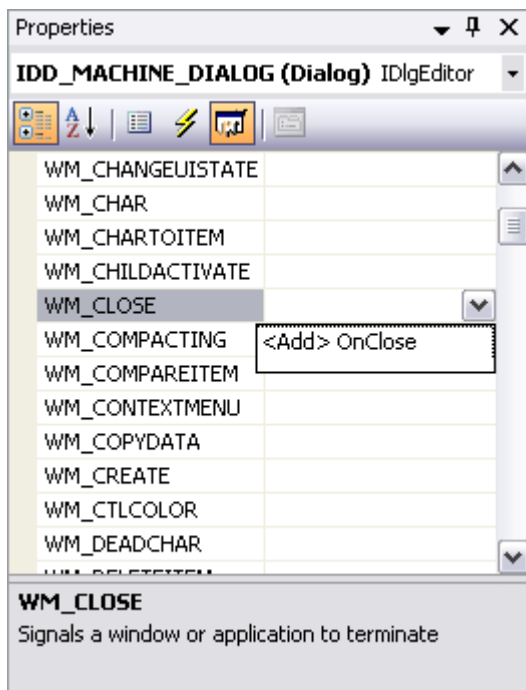
[Slow]ラジオボタンのコード。

```
void CMachineDlg::OnBnClickedRadio2 ()
{
    // Anpassen der "Switch"-
    Variable, bei klick auf Radio Button "slow"// Fits the switch variable if the radio button "slow" was
    s clicked
    bSwitch = false;
    nSwitch = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hSwitch, 8, ".
    switch" );
    nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_VALBYHND, hSwitch, 0x1, &bSwitch );
    nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hSwitch );
}

```

ハンドルの削除:

最後に、プロセス中に作成されたハンドルを削除します。削除するには、フォーム[Properties]を開き、上部の小さいアイコンで[Properties]から[Messages]に切り替えます。次に、下部のウィンドウで[WM_CLOSE]メッセージを選択し、ドロップダウンメニューで[<Add> OnClose]を選択します。



これにより、以下のコードを追加する必要があるコード位置に直接移動します。

```
void CMachineDlg::OnClose ()
{
    // Handles der SPS-
    Variablen freigeben und Port schließen// Release the handles of the SPS variables and close the port
    nErr = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(ULONG), &hDeviceUp );
    nErr = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(ULONG), &hDeviceDown );
    nErr = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(ULONG), &hSteps );
    nErr = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(ULONG), &hCount );
    nErr = AdsPortClose();
}

```

PLCプログラムMachine_Final.proがランタイムシステム1で起動するはずですが、また、作成されたC++プログラムMachine.exeをテストすることができます。

C++サンプルプログラムのダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281694219/.zip>

7.4.5 Microsoft Expression Blend による「Machine」サンプル

Microsoft Expression Blendは、C#とVisual Basicのプログラムインターフェイスを作成するためのプログラムです。このサンプルでは、プログラムによって作成されたインターフェイスが、機械サンプルを用いてリンクされます。プログラミング言語C#を使用しました。

必要なソフトウェア:

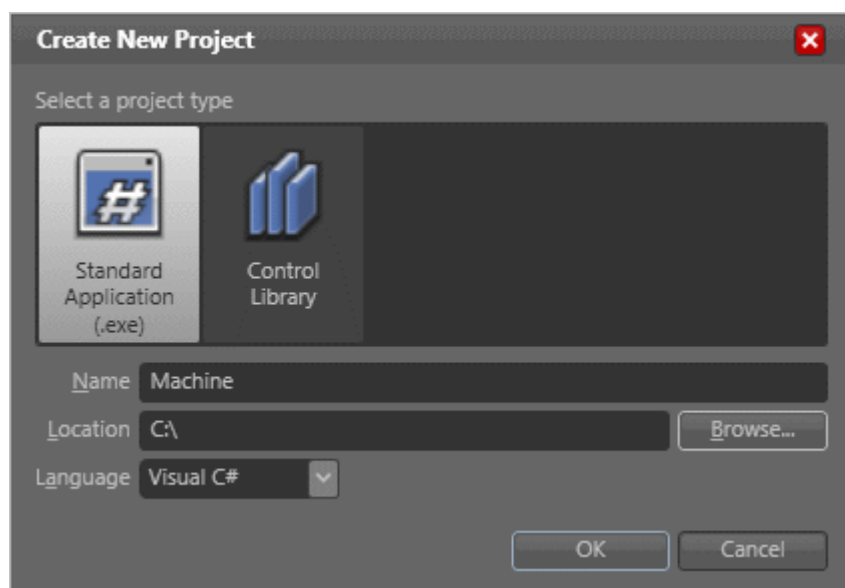
- Microsoft .NET Frameworkバージョン3.0(詳細はwww.microsoft.com を参照)
- Microsoft Expression Blend(詳細はwww.microsoft.com を参照)
- Microsoft Visual Studio 2005
- TwinCAT 2.10

最初の手順...

Microsoft Expression Blendを使用したプログラム開発とTwinCAT ADS .NETコンポーネントの統合について、サンプルを参照しながら説明します。

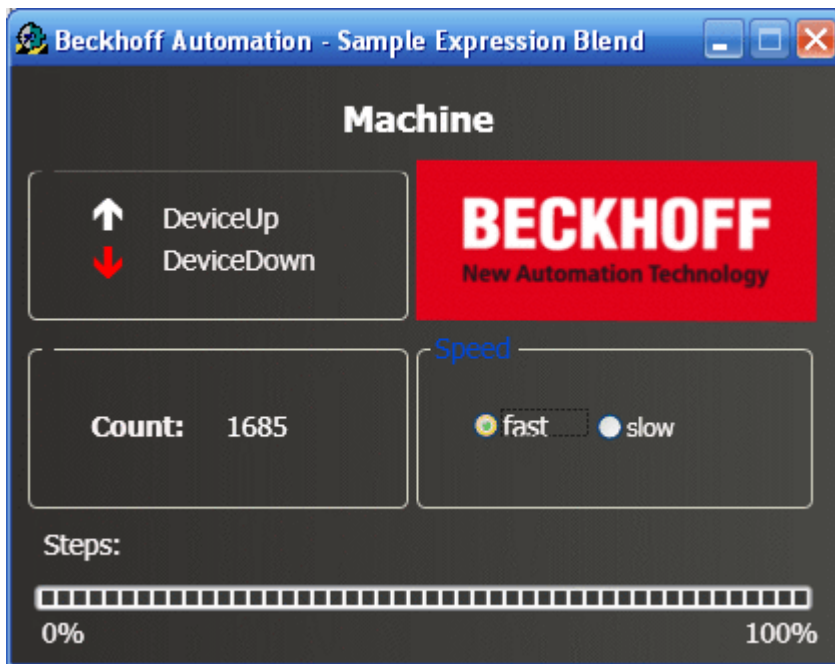
1. 新しいプロジェクトの作成

Expression Blendを起動し、メニューの[File|New Project...]を選択して新しいインターフェイスを作成します。[Create New Application]ダイアログボックスが開きます。ここで、タイプ、名前、場所、プログラミング言語を選択できます。このサンプルでは、タイプとして[Standard Application]、名前として[Machine]、プログラミング言語として[C#]をそれぞれ選択してください。



2. ユーザインターフェイスの作成

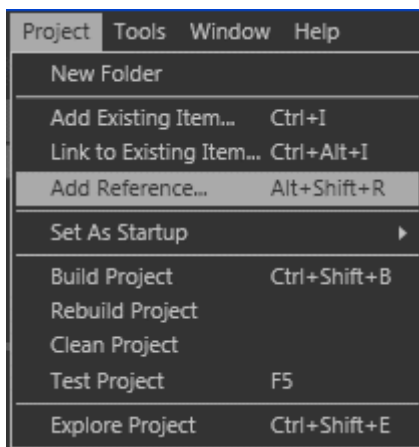
インターフェイス設定がファイルWindow1.xamlに保存されています。



バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右の[Speed]フィールドを使用して、モータのサイクル速度を変更できます。[Steps]表示には、出力1で出力されるサイクルの数が示されます。

3. リファレンスの追加

インターフェイスを作成したら、TwinCAT.Ads.dllというリファレンスを追加します。追加するには、メニューの[Project|Add Reference]を選択します。



4. ソーステキストの編集

インターフェイスを作成し、TwinCAT ADSコンポーネントを統合したら、C#ソーステキストを変更できます。ソーステキストはVisual Studioで表示できます。必要な名前空間「System.IO」および「TwinCAT.Ads」がソーステキストの一番上の行に追加されます。

```
using System.IO;
using TwinCAT.Ads;
```

この後に宣言が続きます。

```
private TcAdsClient tcClient;
private AdsStream dataStream;
private BinaryReader binReader;
private int hEngine;
private int hDeviceUp;
private int hDeviceDown;
private int hSteps;
```



```
private int hCount;
private int hSwitchNotify;
private int hSwitchWrite;
```

最初のメソッドはLoadメソッドです。このメソッドを使用して、さまざまなクラスのインスタンスを生成し、ポート801へのリンクを作成します。

```
//-----//
This is called first when the program is started//-----//
---private void Load(object sender, EventArgs e)
{
    try
    {
        // Create a new instance of the AdsStream class
        dataStream = new AdsStream(7);

        // Create a new instance of the BinaryReader class
        binReader = new BinaryReader(dataStream);

        // Create a new instance of the TcAdsClient class
        tcClient = new TcAdsClient();

        // Linking with local PLC - runtime 1 - port 801
        tcClient.Connect(801);
    }
    catch
    {
        MessageBox.Show("Fehler beim Laden");
    }

    //...
}
```

Loadメソッド内の変数は、リンクされてメソッドとリンクされます。(「メソッド「tcClient_OnNotification」の書き込み」を参照)tcClient_OnNotificationは、変数が変化したときに呼び出されます。

```
try
{
    // Initialising of PLC variable monitoring
    hEngine = tcClient.AddDeviceNotification(".engine", dataStream, 0, 1, AdsTransMode.OnChange, 10, 0, null);
    hDeviceUp = tcClient.AddDeviceNotification(".deviceUp", dataStream, 1, 1, AdsTransMode.OnChange, 10, 0, null);
    hDeviceDown = tcClient.AddDeviceNotification(".deviceDown", dataStream, 2, 1, AdsTransMode.OnChange, 10, 0, null);
    hSteps = tcClient.AddDeviceNotification(".steps", dataStream, 3, 1, AdsTransMode.OnChange, 10, 0, null);
    hCount = tcClient.AddDeviceNotification(".count", dataStream, 4, 2, AdsTransMode.OnChange, 10, 0, null);
    hSwitchNotify = tcClient.AddDeviceNotification(".switch", dataStream, 6, 1, AdsTransMode.OnChange, 10, 0, null);

    // Retrieving of the "switch" handle. This is required for writing the value
    hSwitchWrite = tcClient.CreateVariableHandle(".switch");

    // Creating an event for changes in the PLC variable values
    tcClient.AdsNotification += new AdsNotificationEventHandler(tcClient_OnNotification);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
```

PLC変数のリンク

メソッドAddDeviceNotificationは、変数をリンクするために使用されました。

```
public int AddDeviceNotification(string variableName, AdsStream dataStream, int offset, int length, AdsTransMode transMode, int cycleTime, int maxDelay, object userData);
```

- **variableName:** PLC変数の名前。
- **dataStream:** データを受信するデータストリーム。
- **offset:** データストリームにおける間隔。
- **length:** データストリームにおける長さ。

- **transMode:** 変数が増える場合のイベント。
- **cycletime:** 変数が増えたかどうかをPLCサーバがチェックするまでの経過時間(ms)。
- **maxDelay:** イベントが完了するまでの最遅完了時間(ms)。
- **userData:** 特定のデータを保存するために使用できるオブジェクト。

メソッドCreateVariableHandleは変数hSwitchWriteをリンクするために使用されました。

```
int TcAdsClient.CreateVariableHandle(string variableName);
```

- **variableName:** PLC変数の名前。

メソッド「tcClient_OnNotification」の書き込み:

このメソッドは、PLC変数のいずれかが変化した場合に呼び出されます。

```
//-----//
This is called if one of the PLC variables changes//-----
private void tcClient_OnNotification(object sender, AdsNotificationEventArgs e)
{
    try
    {
        // Setting the position of e.DataStream to the position of the current required value
        e.DataStream.Position = e.Offset;

        // Determining which variable has changed if (e.NotificationHandle == hDeviceUp)
        {
            //
            Adapting the colours of the diagrams to match the variables if (binReader.ReadBoolean() == true)
            {
                DeviceUp_LED.Foreground = new SolidColorBrush(Colors.Red);
            }
            else
            {
                DeviceUp_LED.Foreground = new SolidColorBrush(Colors.White);
            }
        }
        else if (e.NotificationHandle == hDeviceDown)
        {
            if (binReader.ReadBoolean() == true)
            {
                DeviceDown_LED.Foreground = new SolidColorBrush(Colors.Red);
            }
            else
            {
                DeviceDown_LED.Foreground = new SolidColorBrush(Colors.White);
            }
        }
        else if (e.NotificationHandle == hSteps)
        {
            // Setting the progress bar to the current step
            prgSteps.Value = binReader.ReadByte();
        }
        else if (e.NotificationHandle == hCount)
        {
            // Displaying the "count" value
            lblCount.Text = binReader.ReadUInt16().ToString();
        }
        else if (e.NotificationHandle == hSwitchNotify)
        {
            // Selecting the correct radio button if (binReader.ReadBoolean() == true)
            {
                optSpeedFast.IsChecked = true;
            }
            else
            {
                optSpeedSlow.IsChecked = true;
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

最後に、機械を速くまたは遅くするための2つのメソッドが必要です。これらのメソッドを使用してPLC変数「switch」に値を書き込むことで、仮想スイッチを切り替えます。

```
//-----//
This is called if the 'slow' field is selected//-----
--private void optSpeedFast_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, true);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//-----//
This is called if the 'fast' field is selected//-----
--private void optSpeedSlow_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

正しいイベントに対してメソッドが確実に呼び出されるために、[Design]ビューの[DocumentRoot]を選択し、[Events]の下の[Add]をクリックし、[Load method]を選択します。

2つのラジオボタンにも同じことが当てはまりますが、このサンプルではイベント[Click]とメソッド「optSpeedFast_Click」または「optSpeedSlow_Click_Click」を選択します。

通知とハンドルの削除:

ウィンドウの[Close]イベントで、メソッドDeleteDeviceNotification()を使用してリンクを再び有効にしません。

```
private void Close(object sender, EventArgs e)
{
    try
    {
        // Delete notifications and handles
        tcClient.DeleteDeviceNotification(hEngine);
        tcClient.DeleteDeviceNotification(hDeviceUp);
        tcClient.DeleteDeviceNotification(hDeviceDown);
        tcClient.DeleteDeviceNotification(hSteps);
        tcClient.DeleteDeviceNotification(hCount);
        tcClient.DeleteDeviceNotification(hSwitchNotify);

        tcClient.DeleteVariableHandle(hSwitchWrite);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    tcClient.Dispose();
}
```

PLCプログラムMachine_Final.proがランタイムシステム1で起動するはずですが、また、作成されたC#プログラムMachine.exeをテストすることができます。

ダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281695627/.zip>

7.4.6 Microsoft Windows Vista Media Center による 「Machine」 サンプル

Microsoft Expression Blendは、C#とVisual Basicのプログラムインターフェイスを作成するためのプログラムです。このサンプルでは、このプログラムによって作成されたインターフェイスが、機械サンプルを用いてリンクされ、その後、Microsoft Windows Vista Media Centerに統合されます。プログラミング言語C#を使用しました。

ターゲットプラットフォーム:

-Windows Vista

実装

-Visual C#

必要なソフトウェア:

- Microsoft .NET Frameworkバージョン3.0
- Microsoft Expression Blend
- Microsoft Visual Studio 2005
- Microsoft Windows Vista Media Center
- Microsoft Windows SDK for .Net Framework 3.0
- Microsoft Visual Studio 2005 extensions for .Net Framework 3.0 (2006年11月CTP)
- TwinCAT 2.10
- Notepadまたはその他のテキストエディタ

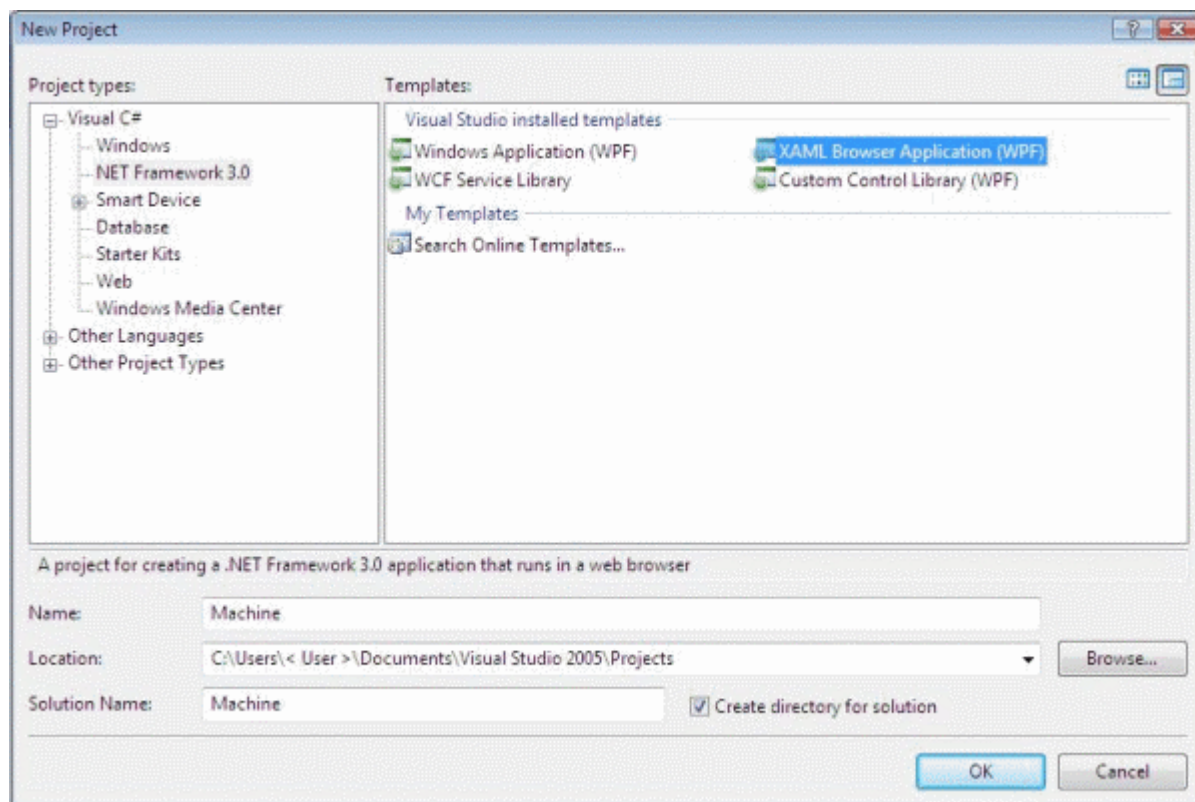
最初の手順...

Microsoft Visual StudioおよびMicrosoft Expression Blendを使用したプログラムの開発、TwinCAT ADS .NETコンポーネントの統合、およびMicrosoft Windows Vista Media Centerへの統合について、サンプルを参照しながら段階的に説明します。

1. 新しいプロジェクトの作成

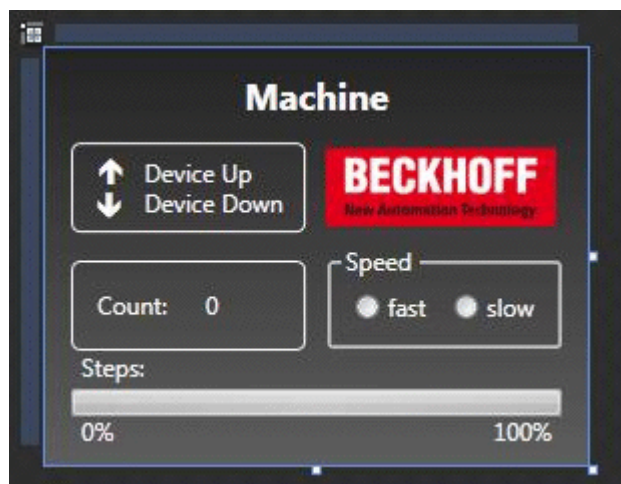
Microsoft Visual Studioを起動し、新しいXAMLブラウザアプリケーションを作成します。メニューの[File|New|Project...]を選択して続行します。[New Project]ダイアログボックスが開きます。

まず、プロジェクトタイプ[Project types|Visual C#|Net Framework 3.0]を選択します。プロジェクトタイプテンプレートが右に表示されます。[XAML Browser Application]を選択します。プロジェクトの名前(この場合は「Machine」)を入力し、場所を指定します。



2. ユーザインターフェイスの作成

ユーザインターフェイスを作成するために、Microsoft Expression Blendに切り替え、作成したばかりのプロジェクトを開きます。

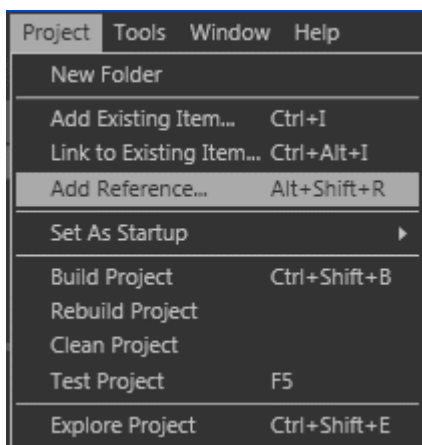


バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右の[Speed]フィールドを使用して、モータのサイクル速度を変更できます。[Steps]表示には、出力1で出力されるサイクルの数が示されます。

ユーザインターフェイスを常にそのサイズに合わせるために、上部グリッドをコピーし、そのグリッドの代わりに表示ボックスを挿入します。グリッドを表示ボックスに挿入します。ページ/表示ボックス/グリッドのサイズを[Auto]に設定します。これにより要素が移動することがあります。その場合は、要素を再配置する必要があります。ページ/表示ボックス/グリッドのサイズが不注意で[Fixed]にリセットされないようにしてください。

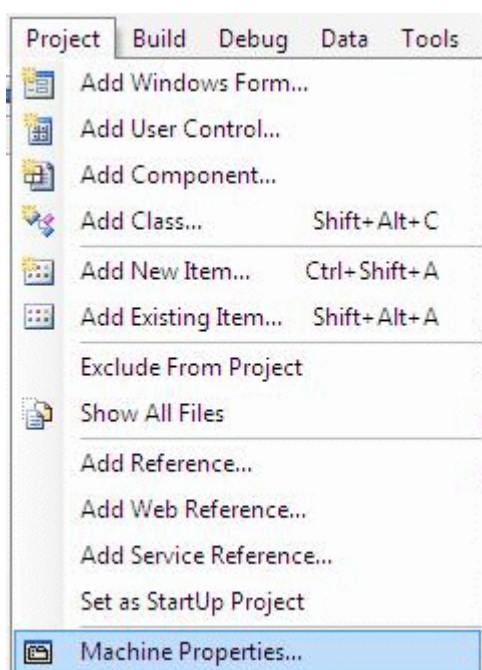
3. リファレンスの追加

インターフェイスを作成したら、「TwinCAT.Ads.dll」というリファレンスを追加します。Visual Studio または Expression Blend で追加できます。いずれの場合も、メニューから[Project|Add Reference]を選択してください。

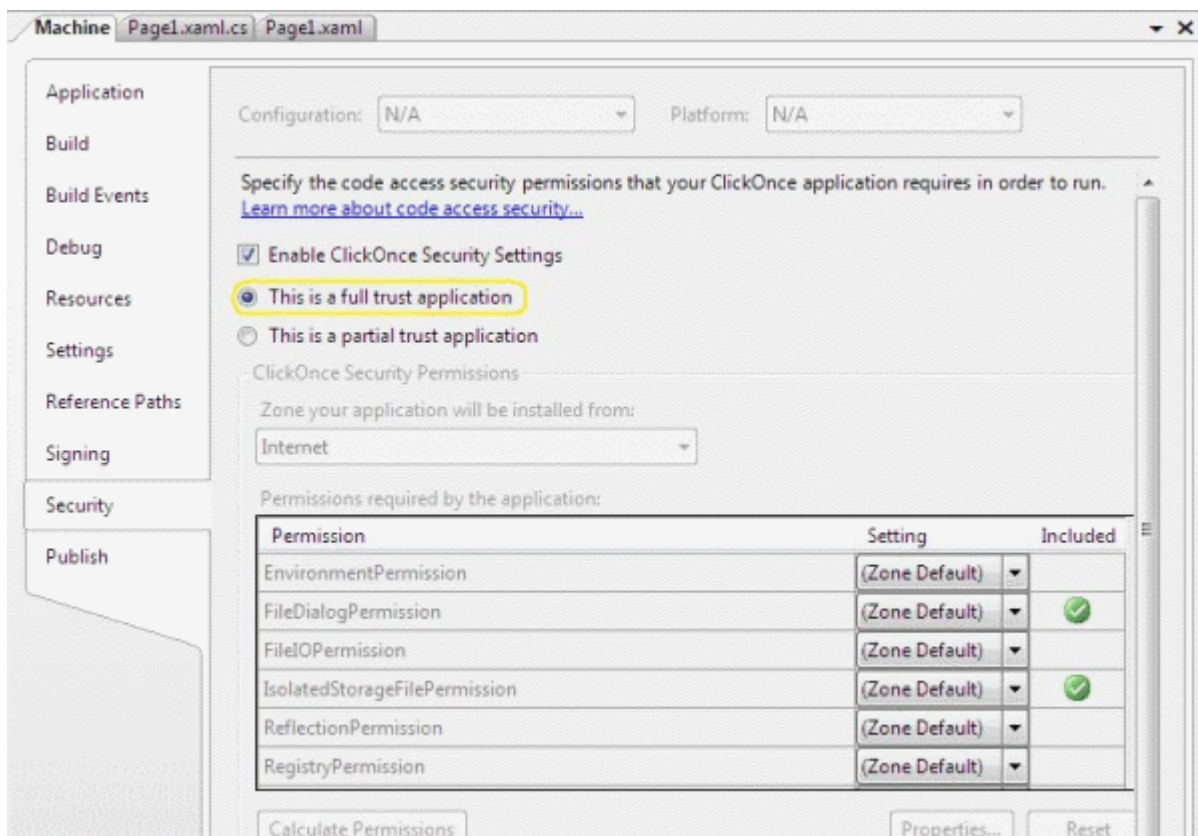


4. セキュリティの有効化

メニューから[Project]<Project Name> Properties...]を選択します。



タブが開きます。そのタブでプロジェクトのプロパティを指定できます。[Security]を選択し、[This is a full trust application]を選択します。



5. ソーステキストの編集

この時点で、C#でのソーステキストの作成を始めることができます。
必要な名前空間「System.IO」および「TwinCAT.Ads」がソーステキストの一番上の行に追加されます。

```
using System.IO;
using TwinCAT.Ads;
```

次の手順は宣言です。

```
private TcAdsClient tcClient;
private AdsStream dataStream;
private BinaryReader binReader;
private int hEngine;
private int hDeviceUp;
private int hDeviceDown;
private int hSteps;
private int hCount;
private int hSwitchNotify;
private int hSwitchWrite;
```

最初のメソッドは「Load」メソッドです。このメソッドを使用して、さまざまなクラスのインスタンスを生成し、ポート801へのリンクを作成します。

```
//-----// Wird als erstes beim Starten des Programms
aufgerufen// Is activated first when the program is started//-----
-----private void Load(object sender, EventArgs e)
{
    try
    {
        // Eine neue Instanz der Klasse AdsStream erzeugen// Create an new instance of the AdsStream class
        dataStream = new AdsStream(7);

        // Eine neue Instanz der Klasse BinaryReader erzeugen// Create a new instance of the BinaryReader class
        binReader = new BinaryReader(dataStream);

        // Eine neue Instanz der Klasse TcAdsClient erzeugen// Create an new instance of the TcAdsClient class
        tcClient = new TcAdsClient();

        // Verbinden mit lokaler SPS - Laufzeit 1 - Port 801// Connecting to local PLC - Runtime 1 -
```

```

Port 801
    tcClient.Connect(801);
}
catch
{
    MessageBox.Show("Fehler beim Laden");
}

//...

```

Loadメソッド内の変数は、リンクされてメソッドとリンクされます。(すでに書かれていなければならぬ)tcClient_OnNotificationは、変数が変化したときに呼び出されます。

```

try
{
    // Initialisieren der Überwachung der SPS-
    Variablen// Initializing the monitoring of the PLC variables
    hEngine = tcClient.AddDeviceNotification(".engine", dataStream, 0, 1, AdsTransMode.OnChange, 10,
    0, null);
    hDeviceUp = tcClient.AddDeviceNotification(".deviceUp", dataStream, 1, 1, AdsTransMode.OnChange,
    10, 0, null);
    hDeviceDown = tcClient.AddDeviceNotification(".deviceDown", dataStream, 2, 1, AdsTransMode.OnCha
    nge, 10, 0, null);
    hSteps = tcClient.AddDeviceNotification(".steps", dataStream, 3, 1, AdsTransMode.OnChange, 10, 0
    , null);
    hCount = tcClient.AddDeviceNotification(".count", dataStream, 4, 2, AdsTransMode.OnChange, 10, 0
    , null);
    hSwitchNotify = tcClient.AddDeviceNotification(".switch", dataStream, 6, 1, AdsTransMode.OnChan
    ge, 10, 0, null);

    // Holen des Handles von 'switch' -
    wird für das Schreiben des Wertes benötigt// Getting the handle for 'switch' -
    needed for writing the value
    hSwitchWrite = tcClient.CreateVariableHandle(".switch");

    // Erstellen eines Events für Änderungen an den SPS-Variablen-
    Werten // Creating an event for changes of the PLC variable value
    tcClient.AdsNotification += newAdsNotificationEventHandler(tcClient_OnNotification);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

6. 定義

PLC変数の定義:

メソッドAddDeviceNotificationは変数をリンクするために使われました。

```

public int AddDeviceNotification(string variableName, AdsStream dataStream, int offset, int length,
    AdsTransMode transMode, int cycleTime, int maxDelay, object userData);

```

- **variableName:** PLC変数の名前。
- **dataStream:** データを受信するデータストリーム。
- **offset:** データストリームにおける間隔。
- **length:** データストリームにおける長さ。
- **transMode:** 変数が変化する場合のイベント。
- **cycletime:** 変数が変化したかどうかをPLCサーバがチェックするまでの経過時間(ms)。
- **maxDelay:** イベントが完了するまでの最遅完了時間(ms)。
- **userData:** 特定のデータを保存するために使用できるオブジェクト。

メソッドCreateVariableHandleは変数「hSwitchWrite」をリンクするために使われました。

```

int TcAdsClient.CreateVariableHandle(string variableName);

```

- **variableName:** PLC変数の名前。

7. メソッドの書き込み:

上で、まだ存在していないメソッドが参照されました。このメソッド(「tcClient_OnNotification」)が次に書き込まれます。このメソッドは、PLC変数のいずれかが変化した場合に呼び出されます。


```
//-----// wird bei Änderung einer SPS-
Variablen aufgerufen// is activated when a PLC variable changes//-----
-----private void tcClient_OnNotification(object sender, AdsNotificationEventArgs e)
{
    try
    {
        // Setzen der Position von e.DataStream auf die des aktuellen benötigten Wertes// Setting the po
        sition of e.DataStream to the position of the current needed value
        e.DataStream.Position = e.Offset;

        // Ermittlung welche Variable sich geändert hat// Detecting which variable has changedif(e.Notif
        icationHandle == hDeviceUp)
        {
            // Die Farben der Grafiken entsprechened der Variablen anpassen// Adapt colors of graphice a
            ccording to the variablesif (binReader.ReadBoolean() == true)
            {
                DeviceUp_LED.Foreground = newSolidColorBrush(Colors.Red);
            }
            else
            {
                DeviceUp_LED.Foreground = newSolidColorBrush(Colors.White);
            }
        }
        else if(e.NotificationHandle == hDeviceDown)
        {
            if (binReader.ReadBoolean() == true)
            {
                DeviceDown_LED.Foreground = newSolidColorBrush(Colors.Red);
            }
            else
            {
                DeviceDown_LED.Foreground = newSolidColorBrush(Colors.White);
            }
        }
        else if(e.NotificationHandle == hSteps)
        {
            // Einstellen der ProgressBar auf den aktuellen Schritt// Setting the ProgressBar to the cur
            rent step
            prgSteps.Value = binReader.ReadByte();
        }
        else if(e.NotificationHandle == hCount)
        {
            // Anzeigen des 'Zähler'-Wertes// Displaying the 'count' value
            lblCount.Text = binReader.ReadUInt16().ToString();
        }
        else if(e.NotificationHandle == hSwitchNotify)
        {
            // Markieren des korrekten RadioButtons// Checking the correct RadioButtonif (binReader.Read
            Boolean() == true)
            {
                optSpeedFast.IsChecked = true;
            }
            else
            {
                optSpeedSlow.IsChecked = true;
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
}
```

最後に、機械を「速い(fast)」または「遅い(slow)」に設定するための2つのメソッドが必要です。これらのメソッドを使用してPLC変数「switch」に値を書き込むことで、仮想スイッチを切り替えます。

```
//-----// wird aufgerufen, wenn das Feld 'schnell'
markiert wird// is activated when the 'fast' field is marked//-----
-----private void optSpeedFast_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, true);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
}
```

```
//-----// wird aufgerufen, wenn das Feld 'langsam'
markiert wird// is activated when the 'slow' field is marked//-----
-----private void optSpeedSlow_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

8. 通知とハンドルの削除:

ウィンドウの[Close]イベントで、メソッドDeleteDeviceNotification()を使用してリンクを再び有効にします。

```
//-----// wird beim Beenden des Programms aufgerufe
n// is activated when ending the program//-----
private void Close(object sender, EventArgs e)
{
    try
    {
        // Löschen der Notifications und Handles// Deleting of the notification and handles
        tcClient.DeleteDeviceNotification(hEngine);
        tcClient.DeleteDeviceNotification(hDeviceUp);
        tcClient.DeleteDeviceNotification(hDeviceDown);
        tcClient.DeleteDeviceNotification(hSteps);
        tcClient.DeleteDeviceNotification(hCount);
        tcClient.DeleteDeviceNotification(hSwitchNotify);

        tcClient.DeleteVariableHandle(hSwitchWrite);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    tcClient.Dispose();
}
```

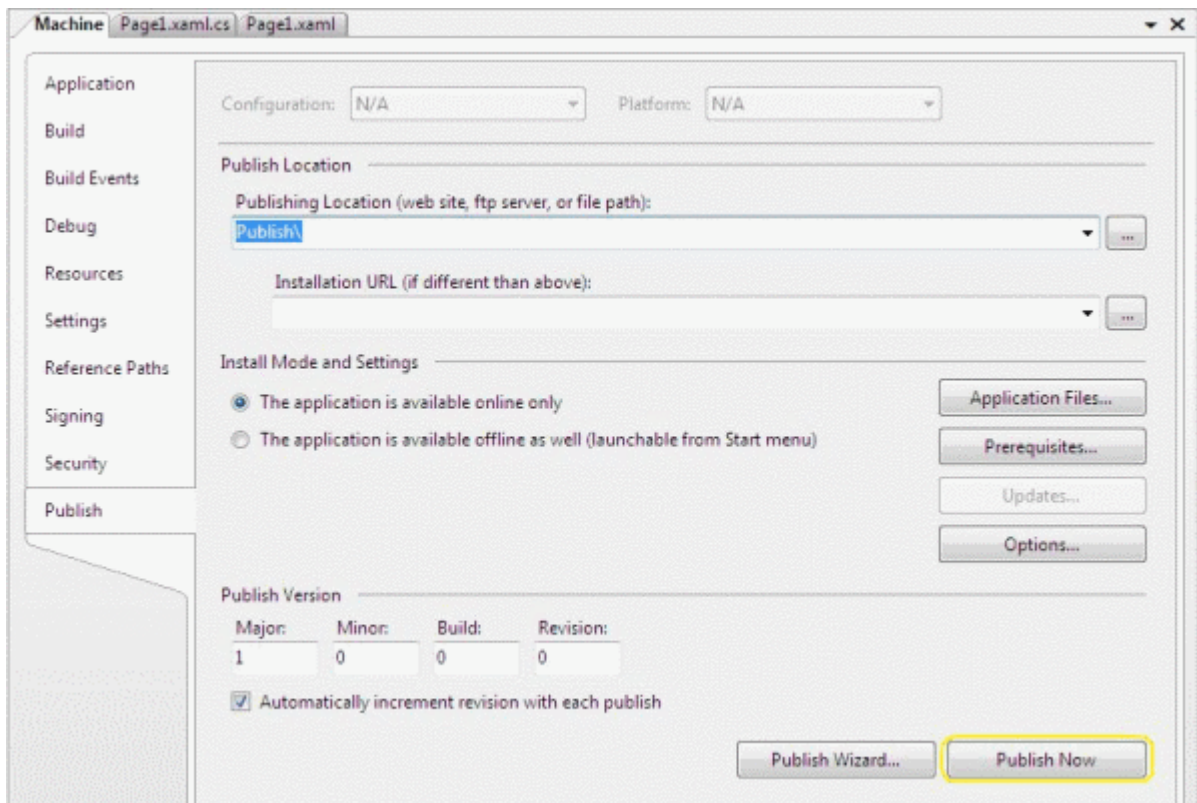
最後に、正しいイベントに対してメソッドが呼び出される必要があります。そのために、Expression Blendを開き、[Page]を選択し、[Properties]の[Events]に切り替えます。次に、[Loaded]の下に[Load]、[Unloaded]の下に[Close]をそれぞれ入力します。2つのラジオボタンとメソッド「optSpeedFast_Click」または「optSpeedSlow_Click」も同様に処理してください(ただし、[Click]イベントを選択する必要があります)。

PLC機械プログラム「Machine_Final.pro」がランタイムシステム1で実行されるはずですが、また、このプログラムをInternet Explorer 7でテストすることができます。

9. Vista Media Centerへの統合

プロジェクトを十分にテストし、エラーが検出されなかったら、プロジェクトをMicrosoft Windows Vista Media Centerに統合することができます。

Visual Studioで、プロジェクトのプロパティをもう一度呼び出しますが、今度は[Publish]を選択します。[Publish Now]をクリックします。これでxbapファイルが作成されます。後で、このファイルをMedia Centerで呼び出します。この手順は、プログラムが変更されるたびに実行する必要があります。変更をMedia Centerに転送する必要があります。



ここでテキストエディタ(Notepadなど)を開き、以下を入力します。

```
<application
  URL = "C:
\Users\

```

これを「C:¥Users¥<User>¥AppData¥Roaming¥Media Center Programs¥Machine.mcl」に保存します。この時点でMedia Centerを起動すると、[Online Media|Program library|Programs by name|Machine]でプログラムが表示されます。

これは、Microsoft Windows Vista Media Centerへの統合の最も簡単な方法です。Media Centerへの統合に関する詳細情報は、[ここをクリックしてください](#)。

10. Expression Blendサンプルのダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281697035/.zip>

7.4.7 Microsoft SilverlightとJavaScript による「Machine」サンプル

Microsoft SilverlightはWebプレゼンテーションテクノロジーです。

ターゲットプラットフォーム

- Windows XP、XPE、WES
- Windows Vista
- Windows 7

実装

- JavaScript

必要なソフトウェア:**- ランタイム:**

- Microsoft Silverlight 1.0 / 1.1

- 開発者ツール:

- Microsoft Visual Studio 2008 Beta 2

- Microsoft Silverlight Tools Alpha Refresh for Visual Studio (2007年7月)

または

- Microsoft Visual Studio 2005

- Microsoft Silverlight 1.0ソフトウェア開発キット

このサンプルではMicrosoft Visual Studio 2005を使用しました。

- 設計者ツール:

- Expression Blend 2

- その他:

- TwinCAT 2.10

- Browser (例: Internet Explorer 7またはMozilla Firefox 2)

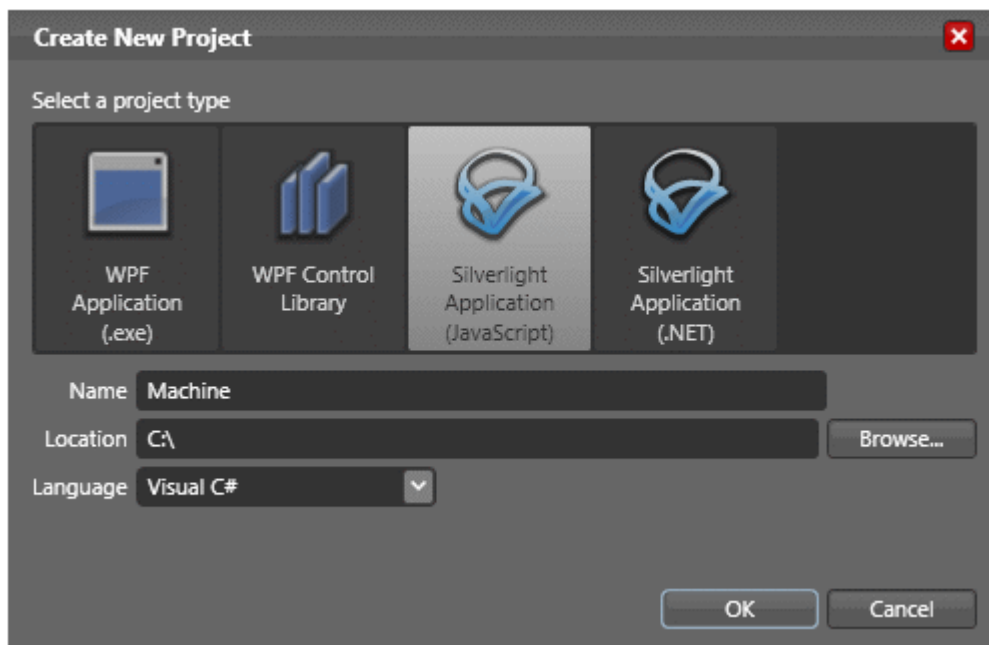
- Microsoft .NET Frameworkバージョン3.0

最初の手順...

Silverlightアプリケーションの開発とTwinCAT ADS Webサービスの統合について、サンプルを参照しながら段階的に説明します。

1. 新しいプロジェクトの作成:

Microsoft Expression Blend 2を起動し、メニューの[File|New Project...]を選択して新しいユーザインターフェイスを作成します。ダイアログ[Create New Project]が開きます。ここで、タイプ、名前、場所、言語を編集できます。このサンプルでは、タイプ「Silverlight Application (JavaScript)」と名前「Machine」を選択します。

**2. ユーザインターフェイスの作成:**

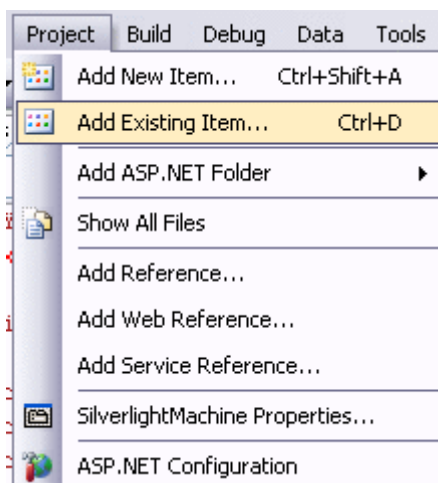
Silverlightアプリケーションでのユーザインターフェイスの開発にいくつかのコントロールを利用できます。1つのテキストボックスと2つの楕円を含むラジオボタンを作成できます。これらはキャンバスにまとめられています。プログレスバーは、別々のキャンバスにまとめられた3つの四角形で作成できます。インターフェイスのプロパティはPage.xamlファイルに記述されています。オブジェクトサイズを「Auto」に設定することはできません。このように設定すると、後でエラーが生じることがあります。



2つの出力が左上隅に表示されます。ワークピースをカウントする変数が下に表示されます。右側の2つのラジオボタンを使用して、エンジン速度を変更できます。表示されるステップ数は、出力1に返されるクロックの数に相当します。

3. XMLHTTP.JSの追加

Visual Studioで[Project|Add Existing Item...]を選択して、ファイル「XMLHTTP.JS」を追加します。このファイルには、PLC変数の読み取り/書き込みとデータタイプの変換の一般的な方法が含まれています。



4. ソースコードの編集

Visual Studioでファイル「Default.html」を開き、「XMLHTTP.JS」ファイルをヘッダーに含めます。

```
<script type="text/javascript" src="xmlhttp.js"></script>
```

JavaScript RangeをHTMLページのHEADフィールドに追加します。そこに次のソースコードが表示されています。

最初にグローバル変数を宣言する必要があります。

```
<script type="text/javascript">
//enter URL to webservice here:
var url = "http://localhost/TcAdsWebService/TcAdsWebService.dll";
//enter netId here:
var netId = "172.16.2.63.1.1";
//enter the port here:
var port = 811;
//send soap request every x seconds:
var refresh = 1000;

var inuse = false;
var b64s, success, errors, req;

var vUp, vDown, vProgressbar, vCount, vFast, vSlow;
...

```

TcAdsWebServiceのURL、netID、およびポートを適合させる必要があります。

GUIオブジェクトに直接アクセスすることはできません。直接アクセスするために、アプリケーションの起動後に、既に宣言されている変数にオブジェクトが割り当てられます。

```
function Load(sender, EventArgs)
{
    vUp = sender.findName("pathUp");
    vDown = sender.findName("pathDown");
    vProgressbar = sender.findName("recProgressbar");
    vCount = sender.findName("txbCount");
    vFast = sender.findName("ellPointFast");
    vSlow = sender.findName("ellPointSlow");
}
```

Loadメソッドには、変数の割り当てが含まれています。Loadメソッドを実行するために、Page.xamlの一番上のキャンバスにLoaded="Load"を追加する必要があります。

```
<Canvasxmlns="http://schemas.microsoft.com/client/2007"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="368" Height="256"
x:Name="Page"
Loaded="Load"
>
...

```

```
function loop(x)
{
    Read(netId, port, '16416', '0', '86'); //send soap read request via xmlhttprequest
    window.setTimeout("loop("+x+")", x);
}
```

PLC変数の読み取り

```
Read(netId, nPort, indexGroup, indexOffset, cbLen)
```

- **netId:** PLCのAMS-Net-Idを示す文字列
- **nPort:** ランタイムシステムのポート番号
- **indexGroup:** 読み取られる変数のインデックスグループ
- **indexOffset:** 読み取られる最初のバイト
- **ncbLen:** 読み取られるバイトの数

```
function init()
{
    b64s = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-=";
    success = 0;
    errors = 0;
    loop(refresh); //send request every x seconds
}
```

onload="init()"を<body>に追加します。これを行うと、ロード中にメソッドが実行されます。

```
<body onload="init()" >
```

次のファンクションが値を読み取り、表示します。Machine.proでの変数のアドレスが正しいことが重要です。

```
function processReqChange()
{
    /*
    readyStates:
    0 = uninitialized
    1 = loading
    2 = loaded
    3 = interactive
    4 = complete
    */ if (req.readyState == 4)
    {
        // only if "OK"if (req.status == 200)
        {
            response = req.responseXML.documentElement;

            inuse = false;

            try//check if there was an error in the request
            {
                errortext = response.getElementsByTagName('faultstring')[0].firstChild.data;
                try
                {

```

```

        errorcode = response.getElementsByTagName('errorcode')[0].firstChild.data;
    }
    catch (e){errorcode="-";}
    alert(errortext + " (" +errorcode+"");
    return;
}
catch (e)
{
    errorcode=0;
}

var data;
try//
if the server returns a <ppData> element decode it, otherwise (write request) do nothing
{
    data = response.getElementsByTagName('ppData')[0].firstChild.data;
    mode = "read";
}
catch (e)
{
    data = "";
    mode = "";
}

if (mode=="read")
{
    try
    {
        data = b64t2d(data); //decode result string

        steps = toInt(data.substr(1, 2));

        bool = toInt(data.substr(5,2));
        bool2 = toInt(data.substr(4,2));

        count = toInt(data.substr(3, 2));

        speed = toInt(data.substr(6, 2));
    }
    catch (e)
    {
        alert("Parsing Failed:" + e);
        return;
    }

    vProgressbar.Width = 306.321/100*steps*4;

    if (bool2 != "1")
        { vCount.Text = count.toString();}

    if (bool == "1")
        { vUp.Opacity=1.0;
          vDown.Opacity=0.0; }
    else if (bool2 == "1")
        { vUp.Opacity=0.0;
          vDown.Opacity=1.0; }
    else
        { vUp.Opacity=0.0;
          vDown.Opacity=0.0; }

    if (speed == "0")
        { vFast.Opacity=1.0;
          vSlow.Opacity=0.0; }
    else
        { vSlow.Opacity=1.0;
          vFast.Opacity=0.0; }

}
}
else alert(req.statusText+" "+req.status); //cannot retrieve xml data
}
}

```

最後の2つのメソッドで、機械の速度を制御するPLC変数が0または1に設定されます。

```

function Fast_MouseLeftButtonDown(sender, EventArgs)
{
    Write(netId, port, '16416', '6', '2', '0', 'int');
}
function Slow_MouseLeftButtonDown(sender, EventArgs)

```

```
{
    Write(netId, port, '16416', '6', '2', '1', 'int');
}
```

SPS変数の書き込み

```
Write(netId, port, indexGroup, indexOffset, cbLen, pwrData, type)
```

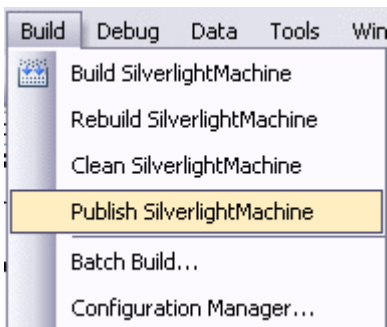
- **netId:** PLCのAMS.Net.Idを示す文字列
- **nPort:** ランタイムシステムのポート番号
- **indexGroup:** 変数のインデックスグループ
- **indexOffset:** 書き込まれる最初のバイト
- **ncbLen:** 書き込まれるバイトの数
- **pwrData:** 書き込まれるデータを含む配列
- **type:** 「bool」、「int」、または「string」

両方のボタンで対応する行を「Click」イベントとして宣言するため「ロード」機能で行われていたように、「Expression Blend 2」に変更する必要があります。

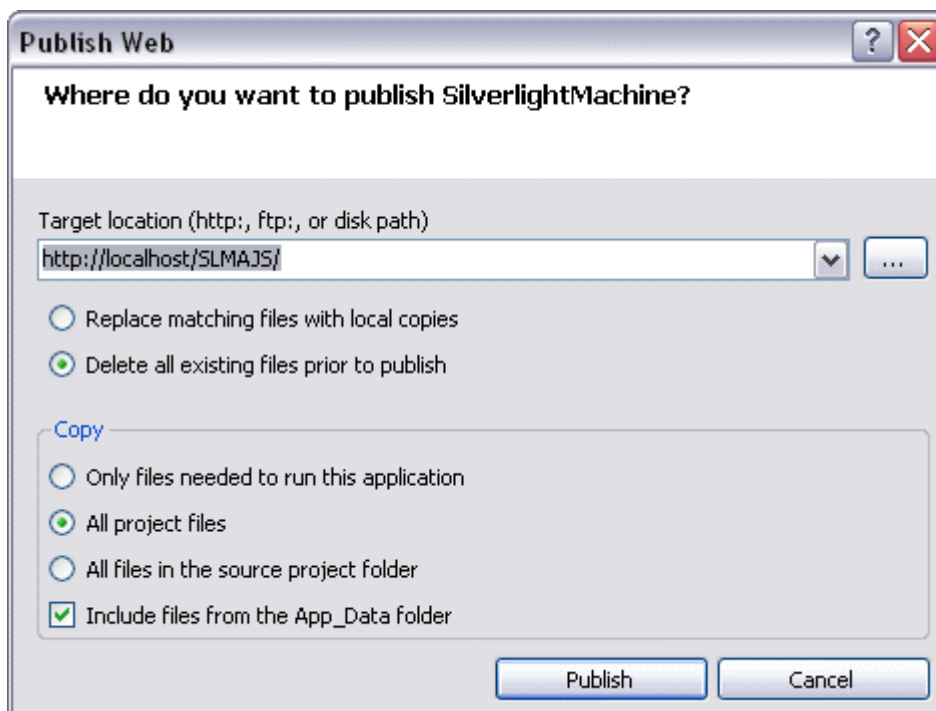
```
<Canvas x:Name="canvasFast" MouseLeftButtonDown="Fast_MouseLeftButtonDown" ...
<Canvas x:Name="canvasSlow" MouseLeftButtonDown="Slow_MouseLeftButtonDown" ...
```

5. テスト:

デバッグすることで、アプリケーションが期待どおりに動作しないことを認識します。これを防止するには、[Build|Publish]を選択します。



ダイアログウィンドウで[Target location]を選択し、[Copy]の下の[All project files]を選択します。



ステータスバーで[Publish succeeded]と表示されたら、ブラウザでアプリケーションを実行しテストすることができます。

ダウンロード:

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281698443/.zip>

7.4.8 Microsoft Silverlight for Windows Embedded による「Machine」サンプル

Windows Embedded CE 6.0 R3の新しいツールがSilverlight for Windows Embeddedです。この新しいテクノロジーを使用すると、CEデバイスのユーザインターフェイスをXAMLで書き込み、Microsoft Expression Blendなどのツールを用いてユーザインターフェイスを設計することができます。Silverlight for Windows Embeddedアプリケーションの作成とADSコンポーネントの統合について、機械サンプルを参照しながら説明します。

必要なソフトウェア

- Microsoft Visual Studio 2008
- Microsoft Expression Blend 2 SP1
- TwinCAT 2.11

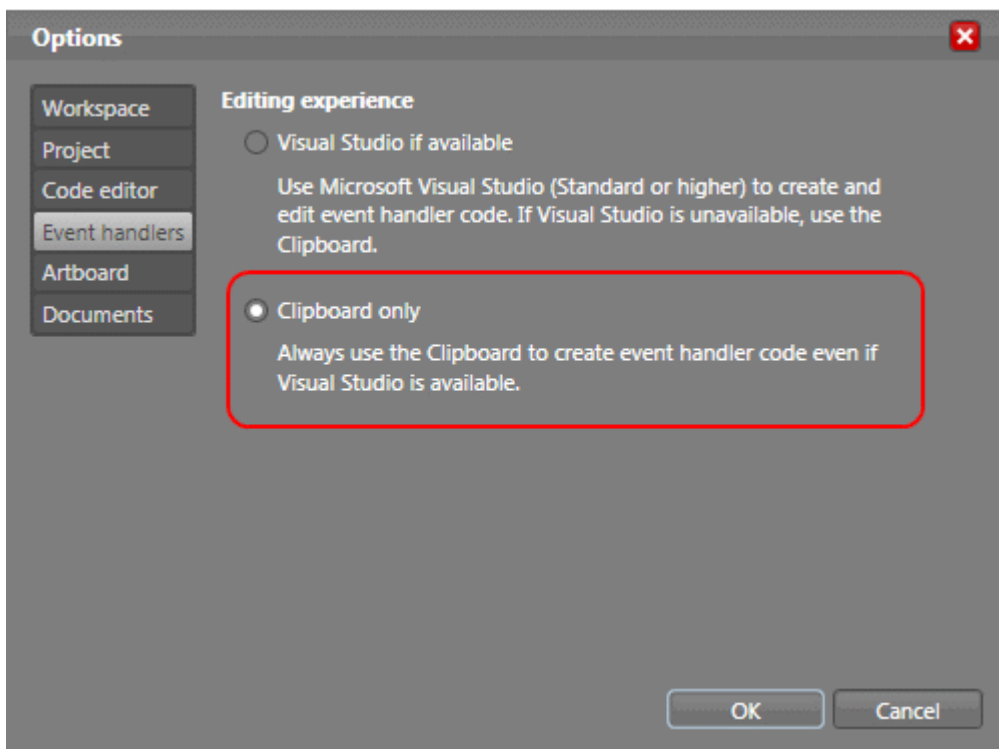
必要なハードウェア

- Windows CE 6.0 R3デバイス (CX1020など)

最初の手順...

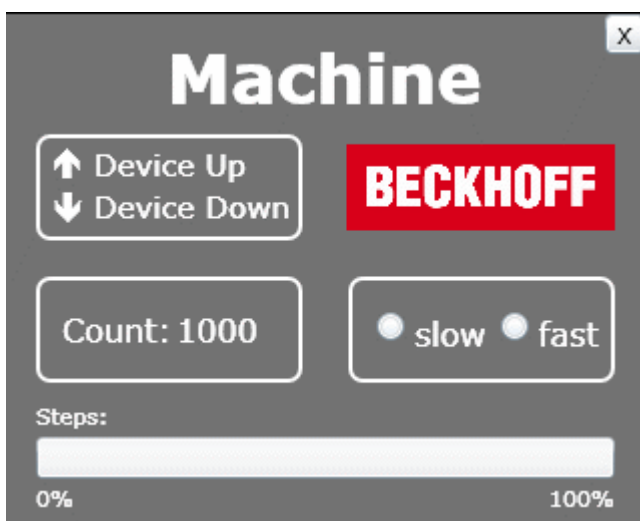
1. 新しいSilverlight 2プロジェクトの作成:

Silverlight for Windows Embeddedアプリケーションの設計をXAMLで記述します。そのために、Microsoft Expression Blend 2 SP1を使用してSilverlight 2プロジェクトを作成します。作成するには、*[File|New Project]*を選択します。よってVisual Studio Solutionをこのサンプルで作成する必要はありません。また、プログラミング言語(Visual C#またはVisual Basic)の選択を無視できます。Silverlight for Windows EmbeddedはExpression Blendには統合されていないVisual C++のみをサポートします。そのため、このツールによって生成されるソースコードを使用することもできません。Visual C#およびVisual Basicのコードが無駄に自動生成されることを避けるために、Expression BlendへのVisual Studioの統合を無効にすることが最善策です。無効にするには、*[Options|Event handlers]*の下の*[Clipboard only]*を選択します。



2. ユーザーインターフェイスの作成

この時点でユーザーインターフェイスをExpression Blendで作成できます。



バスターミナルにも出力される2つの出力が左上に表示されます。左下には、ワークピースをカウントするための変数が表示されます。右側で速度を設定できます。[Steps]表示はサイクル数に対応します。また、プログラムを終了するボタンが右上に作成されます。

```
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="319" Height="255"><!-- Timelines --><UserControl.Resources><!-- Timeline Device Down --
><Storyboardx:Name="timelineDeviceDown"><ColorAnimationUsingKeyFramesBeginTime="00:00:00"
  Storyboard.TargetName="txtDeviceDown"
  Storyboard.TargetProperty="(TextBlock.Foreground) .
  (SolidColorBrush.Color)"><SplineColorKeyFrameKeyTime="00:00:00.4000000" Value="#FFFF0000"/></
ColorAnimationUsingKeyFrames></Storyboard><!-- Timeline Device Up --
><Storyboardx:Name="timelineDeviceUp"><ColorAnimationUsingKeyFramesBeginTime="00:00:00"
  Storyboard.TargetName="txtDeviceUp"
  Storyboard.TargetProperty="(TextBlock.Foreground) .
  (SolidColorBrush.Color)"><SplineColorKeyFrameKeyTime="00:00:00.4000000" Value="#FFFF0000"/></
ColorAnimationUsingKeyFrames></Storyboard><!-- Timeline Engine --
><Storyboardx:Name="timelineEngine"/></UserControl.Resources><!-- Beginn der Layout Beschreibung --
><Gridx:Name="LayoutRoot" Background="#FF595959"><!-- Title Machine --
><TextBlockText="Machine" Margin="80,8.438,80,0" VerticalAlignment="Top"
  FontWeight="Bold" Foreground="FFFFFFFF" FontSize="34"/><!-- Device Up / Device Down --
><GridMargin="15,60,0,0" HorizontalAlignment="Left" VerticalAlignment="Top"
```

```

Height="53" Width="132.532"><RectangleFill="{x:Null}" Stroke="#FFFFFFFF" StrokeThickness="2"
    RadiusX="6" RadiusY="6"/><TextBlockText="Device Up" Margin="28.823,5.396,-8.823,0"
    VerticalAlignment="Top" Foreground="#FFFFFFFF" FontSize="16"/
><TextBlockText="Device Down" Margin="29.002,0,-9.002,4.994"
    VerticalAlignment="Bottom" Foreground="#FFFFFFFF" FontSize="16"/
><TextBlockx:Name="txtDeviceDown" Text="ê" Margin="7.517,0,0,4.998"
    HorizontalAlignment="Left" VerticalAlignment="Bottom" FontFamily="Wingdings"
    FontWeight="Bold" Foreground="#FFFFFFFF" FontSize="16"/
><TextBlockx:Name="txtDeviceUp" Text="é" Margin="7.517,5.497,0,0"
    HorizontalAlignment="Left" VerticalAlignment="Top" FontFamily="Wingdings"
    FontWeight="Bold" Foreground="#FFFFFFFF" FontSize="16"/></Grid><!-- Counter --
><GridMargin="15,0,0,71" HorizontalAlignment="Left" VerticalAlignment="Bottom"
    Height="53" Width="133"><RectangleFill="{x:Null}" Stroke="#FFFFFFFF" StrokeThickness="2"
    RadiusX="6" RadiusY="6"/><StackPanelMargin="12.991,16,0,16" HorizontalAlignment="Left"
    Orientation="Horizontal" Width="113"><TextBlockText="Count:" Width="54.824" Foreground="#
    #FFFFFFFF" FontSize="16"/><TextBlockx:Name="txtCount" Margin="2,0,0,0" Foreground="#FFFFFFFF"
    FontSize="16"/></StackPanel></Grid><!-- Speed --
><GridMargin="0,0,15,71" Height="53" HorizontalAlignment="Right"
    VerticalAlignment="Bottom" Width="132.532"><RectangleFill="{x:Null}" Stroke="#FFFFFFFF" Stro
    keThickness="2"
    RadiusX="6" RadiusY="6"/
><StackPanelMargin="12.988,0,3.012,0" Orientation="Horizontal"><RadioButtonx:Name="radSpeedSlow" Con
    tent="slow" Margin="0,0,6,0"
    Foreground="#FFFFFFFF" FontSize="16" Height="19.496"/
><RadioButtonx:Name="radSpeedFast" Content="fast" Foreground="#FFFFFFFF"
    FontSize="16" Height="19.496"/></StackPanel></Grid><!-- Steps --
><GridMargin="15,0,15,5" VerticalAlignment="Bottom"
    Height="57"><ProgressBarx:Name="prgSteps" Margin="0,18,0,18"
    Maximum="25"/><TextBlockText="Steps:" HorizontalAlignment="Left"
    VerticalAlignment="Top" Foreground="#FFFFFFFF"/
><TextBlockText="0%" HorizontalAlignment="Left"
    VerticalAlignment="Bottom" Foreground="#FFFFFFFF"/
><TextBlockText="100%" HorizontalAlignment="Right"
    VerticalAlignment="Bottom" Foreground="#FFFFFFFF"/></Grid><!-- Close Button --
><Buttonx:Name="butClose" Content="X"
    HorizontalAlignment="Right" VerticalAlignment="Top"
    Height="20" Width="20"/><!-- Beckhoff Logo --
><ImageSource="beckhoff_logo_white.jpg" Margin="0,64.502,15,0" HorizontalAlignment="Right"
    VerticalAlignment="Top" Height="43.151" Width="133.745"
    Stretch="Fill"/></Grid></UserControl>

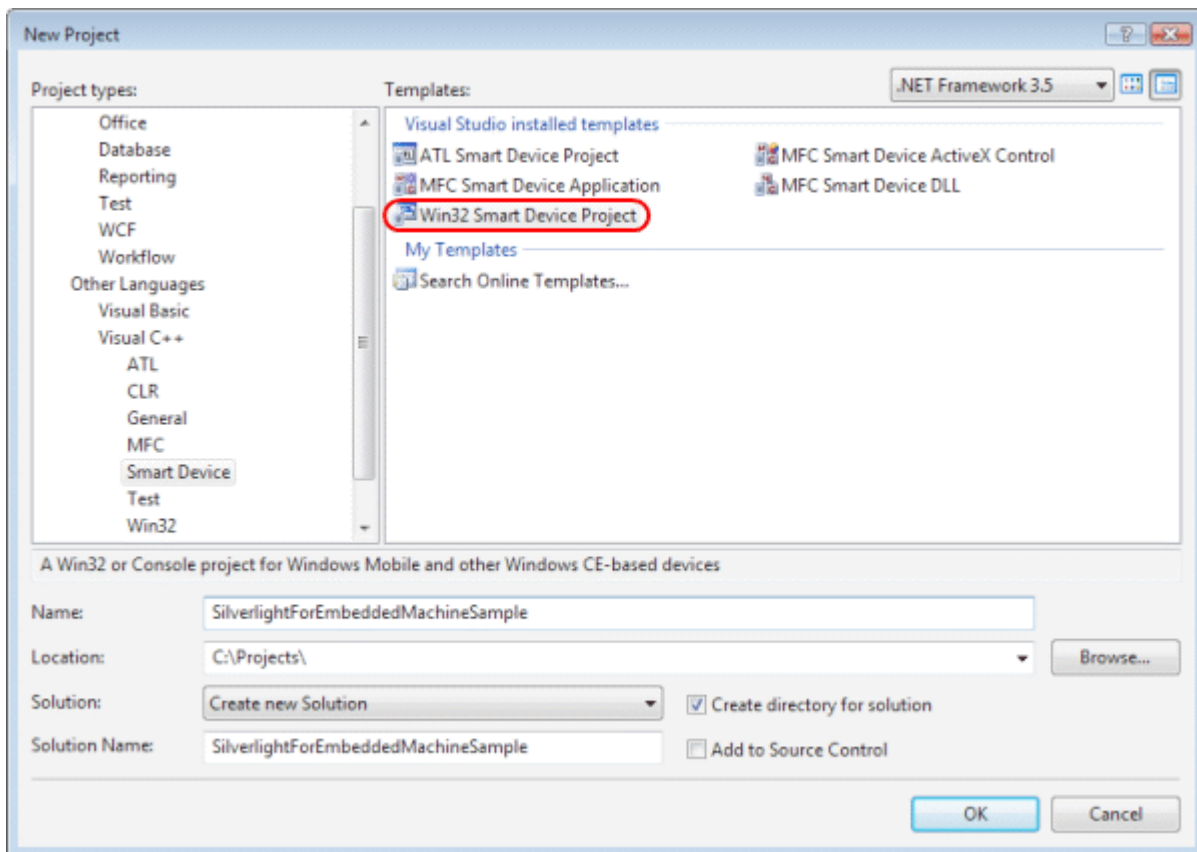
```

3. 新しいWin32 Smart Deviceプロジェクトの作成

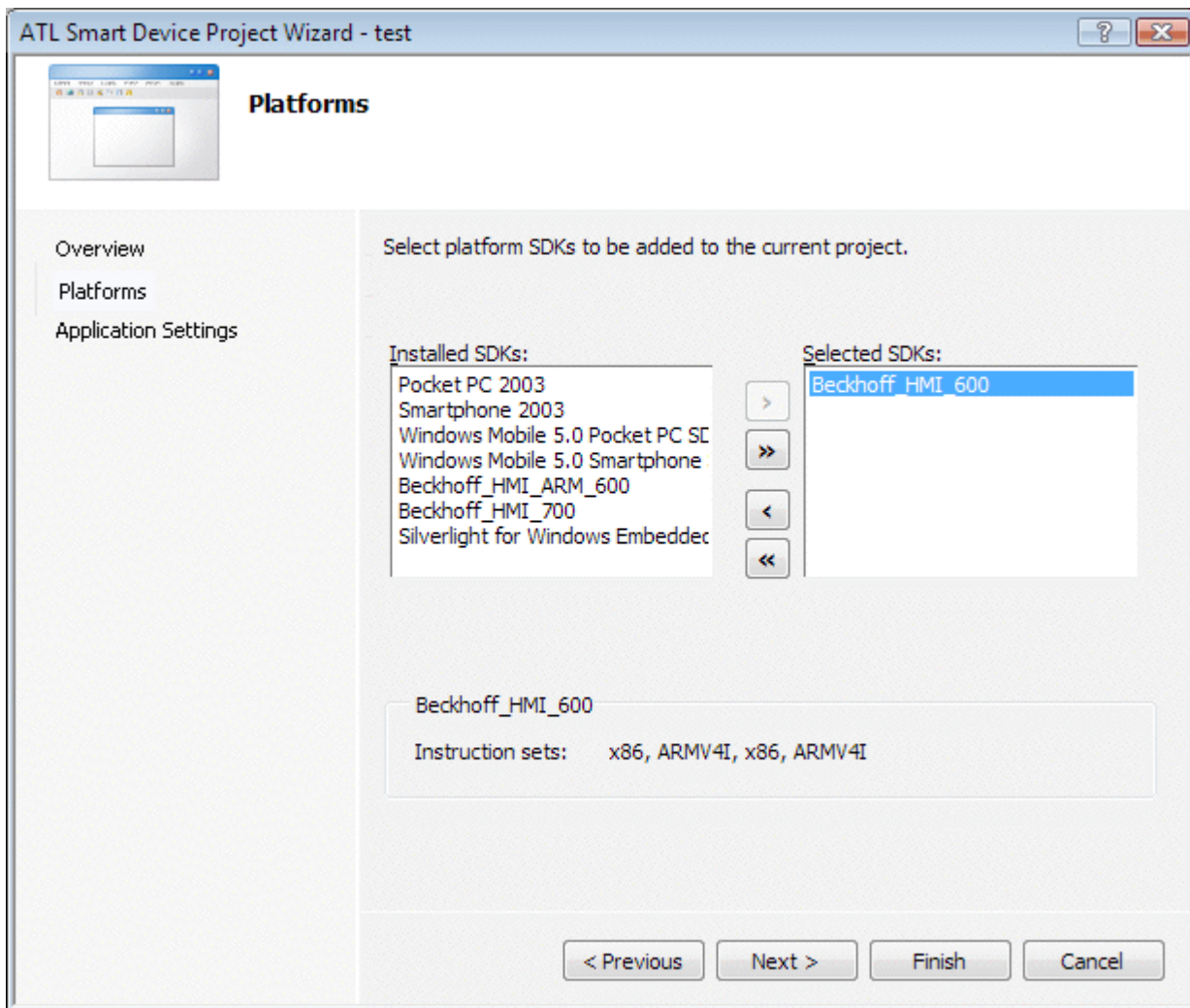
新しいWin32 Smart DeviceプロジェクトをVisual Studio 2008で作成する必要があります。

注記:

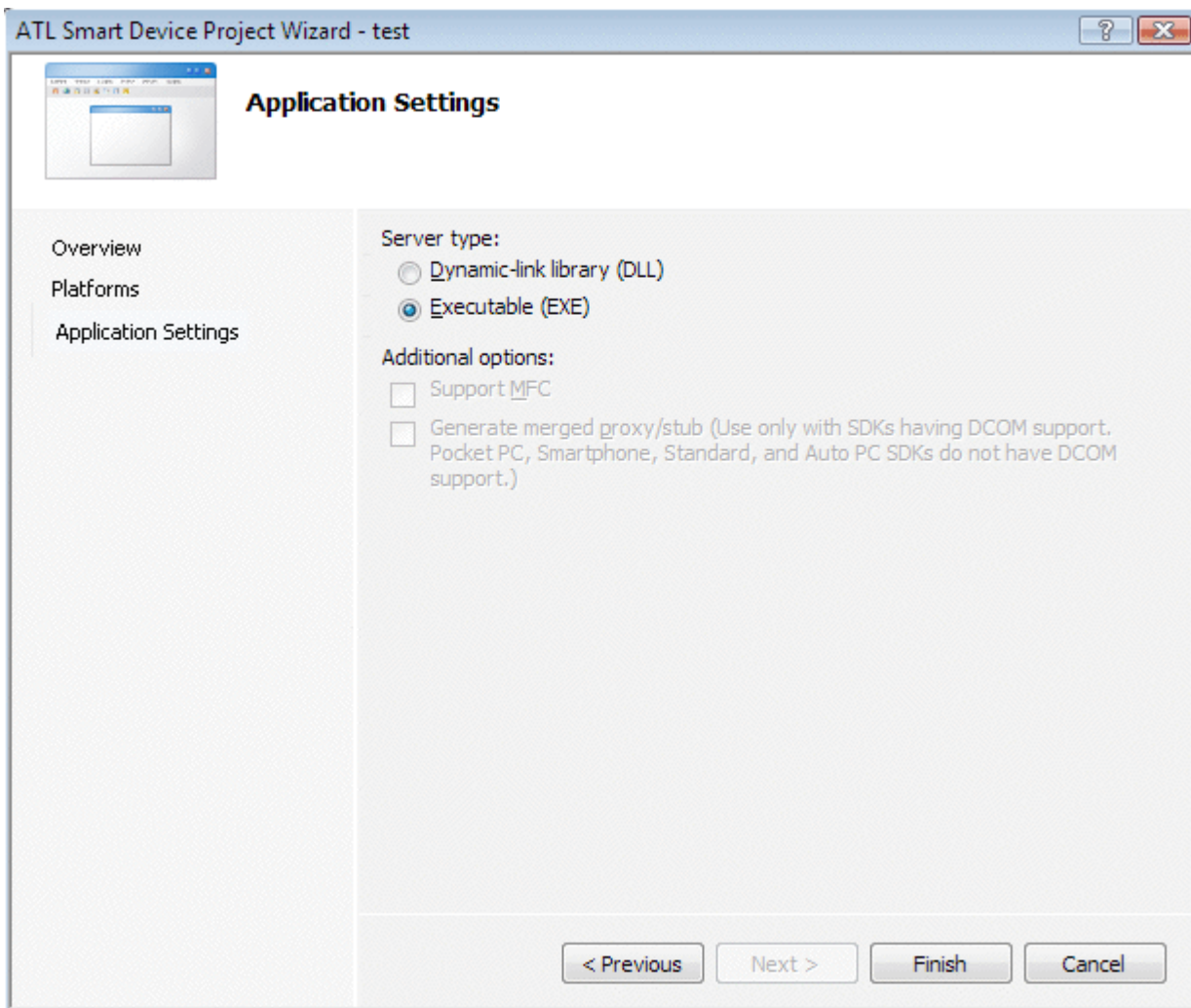
ベッコフHMI 600 SDKがコンピュータにインストールされていない場合は、新しいVisual Studioプロジェクトを作成する前にインストールしてください。



このプロジェクト用のプラットフォームはベッコフHMI 600 SDKです。

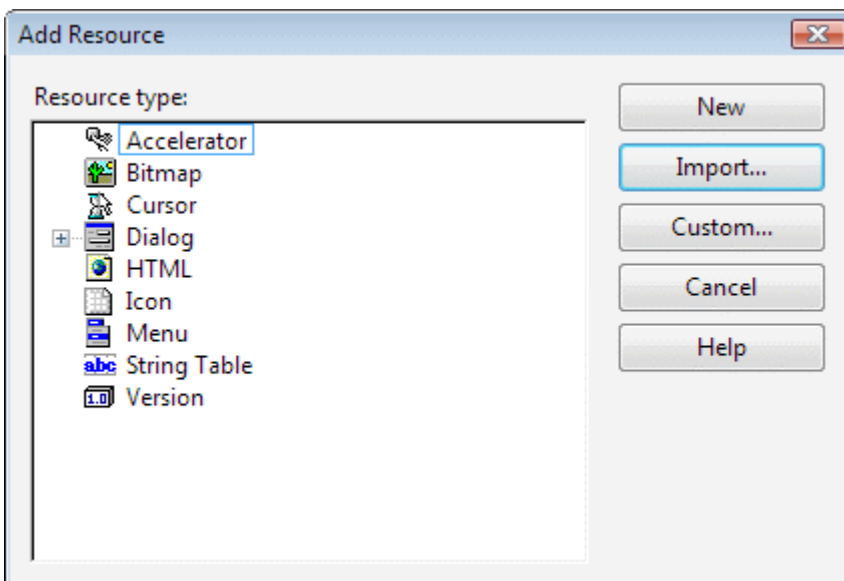


[Server type]で[Executable (EXE)]を選択します。

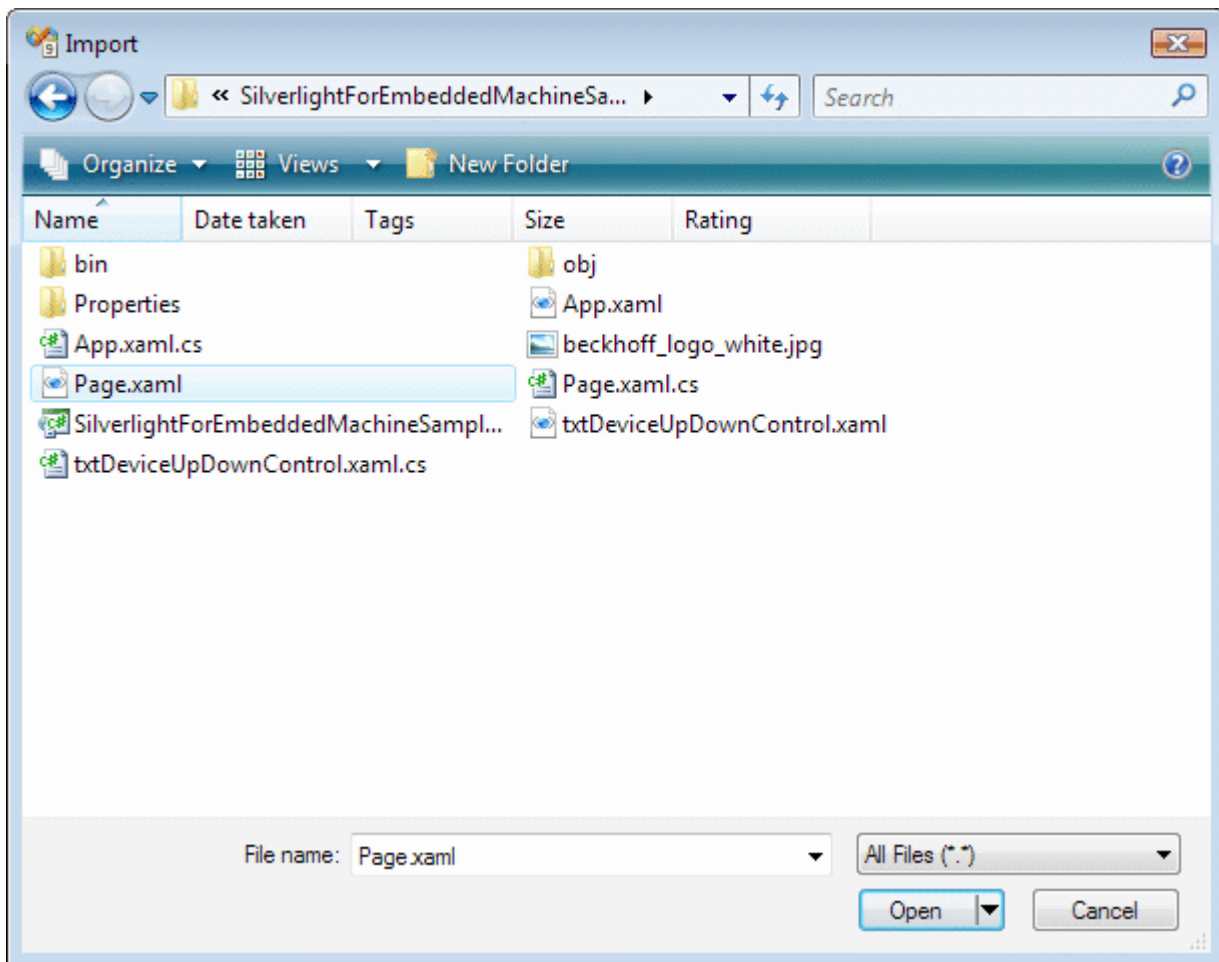


4. XAMLファイルをリソースとして統合する

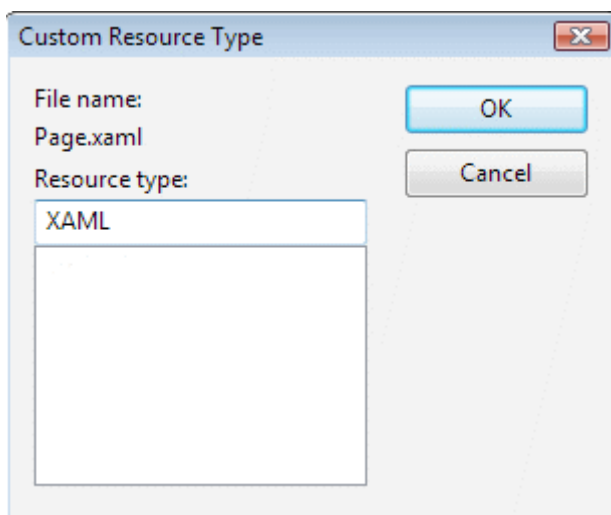
Expression Blendを使用して設計したユーザインターフェイスは、新しいプロジェクトに統合できます。これを行うには、リソースファイル(.rc)を開きます。[Resource View]タブでリソースを右クリックして [Add]-> [Resource...]を選択すると、ダイアログボックスが開き、XAMLファイルを統合できます。



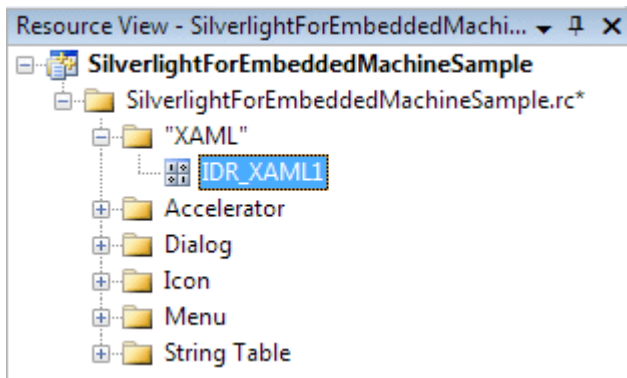
このダイアログを使用して、XAMLファイルをプロジェクトにインポートできます。



リソースタイプとしてXAMLを指定します。



標準的なリソースID (IDR_XAML1)をこのサンプルで保持できます。ただし、実際のプロジェクトではリソースIDを変更することをお勧めします。



5. AdsHelperクラスを作成する

Ads通信の大部分は、AdsHelper here.AdsHelper.hという別のクラスに格納できます。

AdsHelper.h

TcAdsヘッダーをAdsHelperヘッダーに組み込みます。

```
#include "..\AdditionalFiles\TcpAdsApiCe\include\TcAdsDef.h"#include "..\AdditionalFiles\TcpAdsApiCe\include\TcAdsAPI.h"
```

ここで宣言も行います。

```
typedef enum E_NOTIFICATION_IDENT
{
    engine = 0,
    deviceUp = 1,
    deviceDown = 2,
    steps,
    count,
    switchSpeed
};

long AdsGetVarHandle(AmsAddr* pServerAddr, const char* szVarname, long* pHandle);
long AdsFreeVarHandle(AmsAddr* pServerAddr, long handle);
long AdsGerVarHandleEx(long port, AmsAddr* pServerAddr, const char* szVarname, long* pHandle);
long AdsFreeVarHandleEx(long port, AmsAddr* pServerAddr, long handle);

long SpeedSlowEx(long port, AmsAddr* pServerAddr);
long SpeedFastEx(long port, AmsAddr* pServerAddr);

long connect(long port, AmsAddr* pServerAddr);
long disconnect(long port, AmsAddr* pServerAddr);
```

AdsHelper.cpp

ヘッダーStdAfx.hおよびAdsHelper.hをAdsHelper.cppに組み込む必要があります。

```
#include "StdAfx.h"#include "AdsHelper.h"
```

その後でグローバル変数を定義する必要があります。

```
long hEngine, hDeviceUp, hDeviceDown, hSteps, hCount, hSwitch;
unsigned long hEngineNotification, hDeviceUpNotification, hDeviceDownNotification,
    hStepsNotification, hCountNotification, hSwitchNotification;
The methods AdsGetVarHandle and AdsGetVarHandleEx serve to create handles for PLC variables

long AdsGetVarHandle(AmsAddr* pServerAddr, const char* szVarname, long* pHandle)
{
    if (pHandle == NULL || pServerAddr == NULL)
        return E_POINTER;

    unsigned long read = 0;
    long nErr =
        AdsSyncReadWriteReqEx(pServerAddr ADSIGRP_SYM_HNDBYNAME, 0x0,
            sizeof(long), pHandle, strlen(szVarname), (char*)szVarname, &read);

    return nErr;
}

long AdsGetVarHandleEx(long port, AmsAddr* pServerAddr, const char* szVarname, long* pHandle)
{
    if (pHandle == NULL || pServerAddr == NULL)
        return E_POINTER;
```



```

unsigned long read = 0;

long nErr =
AdsSyncReadWriteReqEx2(port, pServerAddr, ADSIGRP_SYM_HNDBYNAME, 0x0,
sizeof(long), pHandle, strlen(szVarname), (char*)szVarname, &read);

return nErr;
}

```

PLC変数のハンドルがAdsFreeVarHandleおよびAdsFreeVarHandleExによって再び解除されます。

```

long AdsFreeVarHandle(AmsAddr* pServerAddr, long handle)
{
return AdsSyncWriteReq(pServerAddr, ADSIGRP_SYM_RELEASEHND, 0,
sizeof(handle), &handle);
}

long AdsFreeVarHandleEx(long port, AmsAddr* pServerAddr, long* pHandle)
{
return AdsSyncWriteReqEx(port, pServerAddr, ADSIGRP_SYM_REALEASEHND, 0,
sizeof(handle), &handle);
}

```

次の2つのメソッドがPLC変数「.switch」を書き込み、その際に速度を「遅い」または「速い」に設定します。

```

long SpeedSlowEx(long port, AmsAddr* pServerAddr)
{
// Handle der SPS-Variable ".switch" erstellen.long handleSpeedSlow = 0;
long adserror = AdsGetVarHandleEx(port, pServerAddr, ".switch", &handleSpeedSlow);

// Die SPS-Variable ".switch" auf FALSE setzenbool datafalse = false;
adserror = AdsSyncWriteReqEx(port, pServerAddr, ADSIGRP_SYM_VALBYHND,
handleSpeedSlow, 0x1, &datafalse);

// Handle der SPS-Variable ".switch" freigeben
adserror = AdsFreeVarHandleEx(port, pServerAddr, handleSpeedSlow);
return adserror;
}

long SpeedFastEx(long port, AmsAddr* pServerAddr)
{
// Handle der SPS-Variable ".switch" erstellen.long handleSpeedFast = 0;
long adserror = AdsGetVarHandleEx(port, pServerAddr, ".switch", &handleSpeedFast);

// Die SPS-Variable ".switch" auf TRUE setzenbool datatrue = true;
adserror = AdsSyncWriteReqEx(port, pServerAddr, ADSIGRP_SYM_VALBYHND,
handleSpeedFast, 0x1, &datatrue);

// Handle der SPS-Variable ".switch" freigeben
adserror = AdsFreeVarHandleEx(port, pServerAddr, handleSpeedFast);
return adserror;
}

```

connectメソッドで、変数への接続がPLCで作成されます。

```

long connect (long port, AmsAddr* pServerAddr, PAdsNotificationFuncEx Callback)
{
// Attribute der Notification festlegen
AdsNotificationAttrib attr;
attr.cbLength = 2;
attr.nTransMode = ADSTRANS_SERVERCYCLE;
attr.nMaxDelay = 100000000; // = 1 sec
attr.nCycleTime = 100000; // = 0,5 sec// Handles der SPS-
Variablen holenlong adserr = AdsGetVarHandleEx(port, addr, ".engine", &hEngine);

if(adserr == 0)
adserr = AdsGetVarHandleEx(port, addr, ".deviceUp", &hDeviceUp);

if(adserr == 0)
adserr = AdsGetVarHandleEx(port, addr, ".deviceDown", &hDeviceDown);

if(adserr == 0)
adserr = AdsGetVarHandleEx(port, addr, ".steps", &hSteps);

if(adserr == 0)
adserr = AdsGetVarHandleEx(port, addr, ".count", &hCount);

if(adserr == 0)

```

```

adserr = AdsGetVarHandleEx(port, addr, ".switch", &hSwitch);

// Überwachung der SPS-Variablen initialisierenif(adserr == 0)
{
  attr.cbLength = 1;
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hEngine,
      &attr, Callback, engine, &hEngineNotification);
}
if(adserr == 0)
{
  attr.cbLength = 1;
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hDeviceUp,
      &attr, Callback, deviceUp, &hDeviceUpNotification);
}
if(adserr == 0)
{
  attr.cbLength = 1;
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hDeviceDown,
      &attr, Callback, deviceDown, &hDeviceDownNotification);
}
if(adserr == 0)
{
  attr.cbLength = 1
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hSteps,
      &attr, Callback, steps, &hStepsNotification);
}
if(adserr == 0)
{
  attr.cbLength = 2;
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hCount,
      &attr, Callback, count, &hCountNotification);
}
if(adserr == 0)
{
  attr.cbLength = 1;
  adserr = AdsSyncAddDeviceNotificationReqEx(port, addr, ADSIGRP_SYM_VALBYHND, hSwitch,
      &attr, Callback, switchSpeed, &hSwitchNotification);
}

return adserr;
}

```

Ads接続を切断する際に、PLC変数のハンドルを解除してポートを閉じる必要があります。

```

long disconnect(long port, AmsAddr* addr)
{
  // Handles der SPS-Variablen freigeben
  AdsFreeVarHandleEx(port, addr, hEngine);
  AdsFreeVarHandleEx(port, addr, hDeviceUp);
  AdsFreeVarHandleEx(port, addr, hDeviceDown);
  AdsFreeVarHandleEx(port, addr, hSteps);
  AdsFreeVarHandleEx(port, addr, hCount);
  AdsFreeVarHandleEx(port, addr, hSwitch);

  // Notifications löschen
  AdsSyncDelDeviceNotificationReqEx(port, addr, hEngineNotification);
  AdsSyncDelDeviceNotificationReqEx(port, addr, hDeviceUpNotification);
  AdsSyncDelDeviceNotificationReqEx(port, addr, hDeviceDownNotification);
  AdsSyncDelDeviceNotificationReqEx(port, addr, hStepsNotification);
  AdsSyncDelDeviceNotificationReqEx(port, addr, hCountNotification);
  AdsSyncDelDeviceNotificationReqEx(port, addr, hSwitchNotification);

  // Kommunikationsport schließen
  AdsPortCloseEx(port);

  return 0;
}

```

6. ソースコードの処理

ヘッダーはSilverlightForEmbeddedMachineSample.cppファイルの最初に組み込まれています。

```
#include "pwinuser.h" #include "xamlruntime.h" #include "xrdelegate.h" #include "xrptr.h" #include "resource.h"
```

次に変数の宣言が続きます。

```
IXRDelegate<XRMouseButtonEventArgs>* clickdelegate;
UINT exitcode;
```

```

IXRVisualHostPtr    vhost;
IXRButtonBasePtr   butClose;
IXRRadioButtonPtr  radSpeedSlow;
IXRRadioButtonPtr  radSpeedFast;
IXRTextBlockPtr    txtDeviceDown;
IXRTextBlockPtr    txtDeviceUp;
IXRTextBlockPtr    txtCount;
IXRProgressBarPtr  prgSteps;

IXRStoryboardPtr   timelineDeviceDown;
IXRStoryboardPtr   timelineDeviceUp;
IXRStoryboardPtr   timelineEngine;

long port;
AmsAddr addr;

unsigned long TimerID;
DWORD EventID;
CRITICAL_SECTION cs;
long event_cnt;
long event_cntold;

void __stdcall Callback(AmsAddr* addr, AdsNotificationHeader* handler, unsigned long User);

```

コールバック関数のタイマーがAds接続をチェックし、必要に応じて接続を復元します。

```

VOID CALLBACK MyTimerProc(
    HWND hwnd, // handle to window for timer messages
    UINT message, // WM_TIMER message
    UINT idTimer, // timer identifier
    DWORD dwTimer) // current system time
{
    if (event_cnt == event_cntold)
    {
        // Handels werden freigegeben und der Port geschlossen
        disconnect(port, &addr);

        // Kommunikationsport auf dem ADS Router öffnen
        port = AdsPortOpenEx();

        addr.port = 0x321;
        long adserror = -1;

        // Neue Verbindung zur SPS herstellen.while(adserror != 0)
        {
            adserror = connect(port, &addr, Callback);
            Sleep(1000);
        }
    }

    event_cntold = event_cnt;
}

```

リンクが存在するPLC変数が変化すると、次のAdsイベントハンドラが呼び出されます。

```

// ADS-State Callback-
Functionvoid __stdcall Callback(AmsAddr* addr, AdsNotificationHeader* handler, unsigned long User)
{
    event_cnt++;
}

```

対応する矢印が赤色で表示されるかどうかは、deviceUp、deviceDown、またはengineがTRUEに設定されているかどうかによって決まります。

タイムラインを使用すれば、この効果をさらに改良することができます。

```

if (User == deviceUp)
{
    if (*(bool*)handler->data == true)
    {
        timelineDeviceDown->Stop();
        timelineEngine->Stop();
        timelineDeviceUp->Begin();
    }
}
else if (User == deviceDown)
{
    if (*(bool*)handler->data == true)
    {
        timelineDeviceUp->Stop();
    }
}

```

```

        timelineEngine->Stop();
        timelineDeviceDown->Begin();
    }
}
else if (User == engine)
{
    if (*(bool*)handler->data == true)
    {
        timelineDeviceDown->Stop();
        timelineDeviceUp->Stop();
        timelineEngine->Begin();
    }
}
}
}

```

*steps*はサイクル数を示します。この値はプログレスバー*prgSteps*によって出力されます。これには、関連するPLC変数がバイト型であるため、データをまずバイトに変換する必要があります。プログレスバーはfloat型のデータしか転送できないため、その後float型への変換が行われます。

```

else if (User == steps)
{
    prgSteps->SetValue((float)*((byte*)handler->data));
}
}

```

*count*の場合、*steps*と同様に、データをまず元のデータタイプに変換する必要があります。その後で、データをテキストに変換してテキストブロック*txtCount*に転送することができます。

```

else if (User == count)
{
    WCHAR text[6];
    wprintf(text, L"%d", *(unsigned short*)handler->data);
    txtCount->SetText(text);
}
}

```

*speed*タイプがラジオボタンによって出力されます。適切なラジオボタンが速度に応じてマークされます。

```

else if (User == switchSpeed)
{
    if (*(bool*)handler->data == true)
    {
        radSpeedFast->SetIsChecked(XRThreeState_Checked);
    }
    else
    {
        radSpeedSlow->SetIsChecked(XRThreeState_Checked);
    }
}
}
}

```

OnClickイベントがさまざまなインスタンスによってトリガされ、開始インスタンスを名前から区別することができます。

```

class BtnEventHandler
{
public:

    HRESULT OnClick(IXRDependencyObject* source, XRMouseButtonEventArgs* args)
    {
        BSTR name;
        HRESULT hr = NULL;
        source->GetName(&name);

        short state = 0;

        long adserror = 0;

        if wcsncmp(name, L"butClose") == 0)
        {
            // Machine Dialog schließen
            vhost->EndDialog(exitcode);
        }
        if wcsncmp(name, L"radSpeedSlow") == 0)
        {
            // Aufruf der Methode SpeedSlowEx um die Geschwindigkeit auf langsam zu setzen.
            adserror = SpeedSlowEx(port, &addr);
        }
        if wcsncmp(name, L"radSpeedFast") == 0)
        {
            // Aufruf der Methode SpeedFastEx um die Geschwindigkeit auf schnell zu setzen.
            adserror = SpeedFastEx(port, &addr);
        }
    }
}

```

```

}

if (adserver != NULL)
{
    // Die Handels werden freigegeben und der Port geschlossen
    disconnect(port, &addr);

    // Der Kommunikationsport auf dem ADS-Router wird geöffnet
    port = AdsPortOpenEx();

    addr.port = 0x321;

    // Neu Verbindung zur SPS herstellen.
    adserver = connect(port, &addr, Callback);

    Sleep(1000);
}

SysFreeString(name);
return S_OK;
}
};

```

WinMainメソッドで、XAMLランタイムをまず初期化する必要があります。XamlRuntimeInitializeが成功すると、Silverlight for Windows Embeddedランタイムがアプリケーションで起動されます。

```

int WINAPI WinMain(HINSTANCE hInstance
    HINSTANCE hPrevInstance,
    LPCTSTR lpCmdLine,
    int nCmdShow)
{
    // Initialisierung der XAML Runtimeif (!XamlRuntimeInitialize())
    return -1;
}

```

それぞれのSilverlight for Windows Embeddedアプリケーションが、固有の「applications」オブジェクトを持ちます。それを經由してグローバル特性へのアクセスが可能です。このオブジェクトにアクセスするためにGetXrApplicationInstance APIを使用します。

```

HRESULT retcode;

// Load an dinit XAML resource
IXRApplicationPtr app;

if (FAILED (retcode=GetXrApplicationInstance(&app)))
return -1;

if (FAILED (retcode=app->AddResourceModule(hInstance)))
return -1;

```

アプリケーションオブジェクトの初期化の後、メインウィンドウを作成し、オブジェクトの管理をSilverlight for Windows Embeddedに引き渡すことができます。

```

XRWindowCreateParams wp;

ZeroMemory(&wp, sizeof(XRWindowCreateParams));

// Set window styles
wp.Style = WS_BORDER;
wp.pTitle = L"Silverlight for Windows Embedded Machine Sample";
wp.Left = 0;
wp.Top = 0;
wp.AllowsMultipleThreadAccess = true;

XRXamlSource xamlsrc;

xamlsrc.SetResource(hInstance, TEXT("XAML"), MAKEINTRESOURCE(IDR_XAML1));

if (FAILED(retcode=app->CreateHostFromXaml(&xamlsrc, &wp, &vhost)))
return -1;

```

Silverlight for Windows Embeddedアプリケーション内のオブジェクトがオブジェクトツリーに配置されます。このオブジェクトにアクセスするには、ルート要素のポインタが必要です。

```

IXRFrameworkElementPtr root;

if (FAILED (retcode=app->CreateHostFromXaml(&xamlsrc, &wp, &vhost)))
return -1;

```

コントロールとタイムラインのインスタンスを作成します。

```
// Get controls by name
if (FAILED(retcode=root->FindName(TEXT("butClose"), &butClose)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("radSpeedSlow", &radSpeedSlow)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("radSpeedFast", &radSpeedFast)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("txtDeviceDown", &txtDeviceDown)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("txtDeviceUp", &txtDeviceUp)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("txtCount", &txtCount)))
    return -1;

if (FAILED(retcode=root->FindName(TEXT("prgSteps", &prgSteps)))
    return -1;

// Get timelines by name
if (FAILED (retcode=root->FindName(TEXT("timelineDeviceDown"), &timelineDeviceDown)))
    return -1;

if (FAILED (retcode=root->FindName(TEXT("timelineDeviceUp"), &timelineDeviceUp)))
    return -1;

if (FAILED (retcode=root->FindName(TEXT("timelineEngine"), &timelineEngine)))
    return -1;
```

「RadioButtonGroup」を作成し、2つのラジオボタンをこのグループから割り当てます。

```
WCHAR groupName[17];
wsprintf(groupName, L"RadioButtonGroup");
radSpeedFast->SetGroupName (groupName);
radSpeedSlow->SetGroupName (groupName);
```

リダイレクトオブジェクトがボタンを使用してEventHandlerをリンクする必要があります。

```
BtnEventHandler handler;

// Set the event handler for the buttons
if (FAILED(retcode=CreateDelegate(&handler, &BtnEventHandler::OnClick, &clickdelegate)))
    return -1;

if (FAILED(retcode=butClose->btnAddClickEventHandler (clickdelegate)))
    return -1;

if (FAILED(retcode=radSpeedSlow->AddClickEventHandler (clickdelegate)))
    return -1;

if (FAILED(retcode=radSpeedFast->AddClickEventHandler (clickdelegate)))
    return -1;
```

Adsコンポーネントを統合します。

```
long adsererror = -1;
port = AdsPortOpenEx();

AdsGetLocalAddressEx(port, &addr);

// connect to the PLC and register callbacks
addr.port = 0x321;
adsererror = connect(port, &addr, Callback);

event_cnt = 0;
event_cntold = -1;

// init timer for reconnect
SetTimer(NULL, NULL, 5000, MyTimerProc);

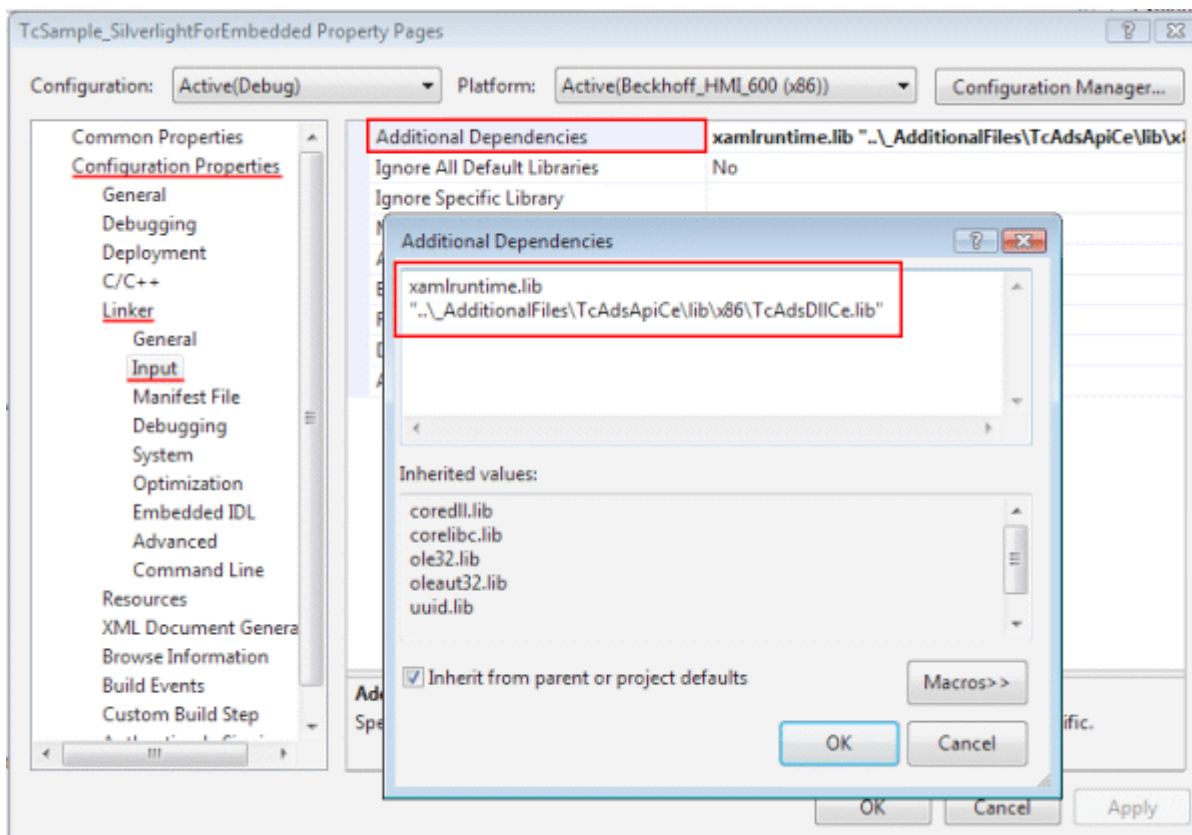
if (FAILED(retcode=vhost->StartDialoge (&exitcode)))
    return -1;

// cleanup
disconnect(port, &addr);
```

```
clickdelegate->Release();  
  
return 0;  
}
```

7. 機能

*xamlruntime.lib*および*TcAdsDllCe.lib*への接続をプロジェクト機能で行う必要があります。



Silverlight for Windows Embeddedサンプルのダウンロード

<https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281699851/.zip>

注記:

このサンプルでは、TcAdsDllCe.libのX86バージョンを使用します。このサンプルをARMデバイスで使用するには、このライブラリを対応するARMバージョンと取り替える必要があります。

詳細はこちら:

www.beckhoff.com/automation

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
+49 5246 9630
info@beckhoff.com
www.beckhoff.com

