BECKHOFF New Automation Technology

Manual | EN

TwinCAT 2

Quick Start

2024-01-08 | Version: 1.3

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Introduction to working with TwinCAT and System requirements

The aim of this introduction is to provide the reader with a swift overview of the possibilities offered by TwinCAT without going into details. The explanations are supplemented by an example application that is added to step by step in the individual chapters.

Refer to the individual instructions of the programs for detailed information.

System requirements

x86 processor

To operate TwinCAT 2, you need a PC with an x86 processor.

Operating system Windows 10 / 11

TwinCAT 2.11 runs under the Windows 10 operating system.

The runtime requires a 32-bit version of Windows 10.

The engineering is also provided for 64-bit versions of Windows 10 and 11.

Sample program

To be able to follow the sample program, you need TwinCAT version 2.11 or higher.

This example refers to the Lightbus, Ethernet or EtherCAT fieldbuses. The following are required for implementation:

- PC interface card for I/O Lightbus (FC2001) or Ethernet card
- Bus coupler
- 2 Bus Terminals with 2 digital outputs each (KL2032)
- Bus end terminal (KL9010)
- Wiring material (fiber optic cable, stranded wire, etc...)
- 24V power supply unit

Other field buses can also be used according to these instructions.

Demokit:

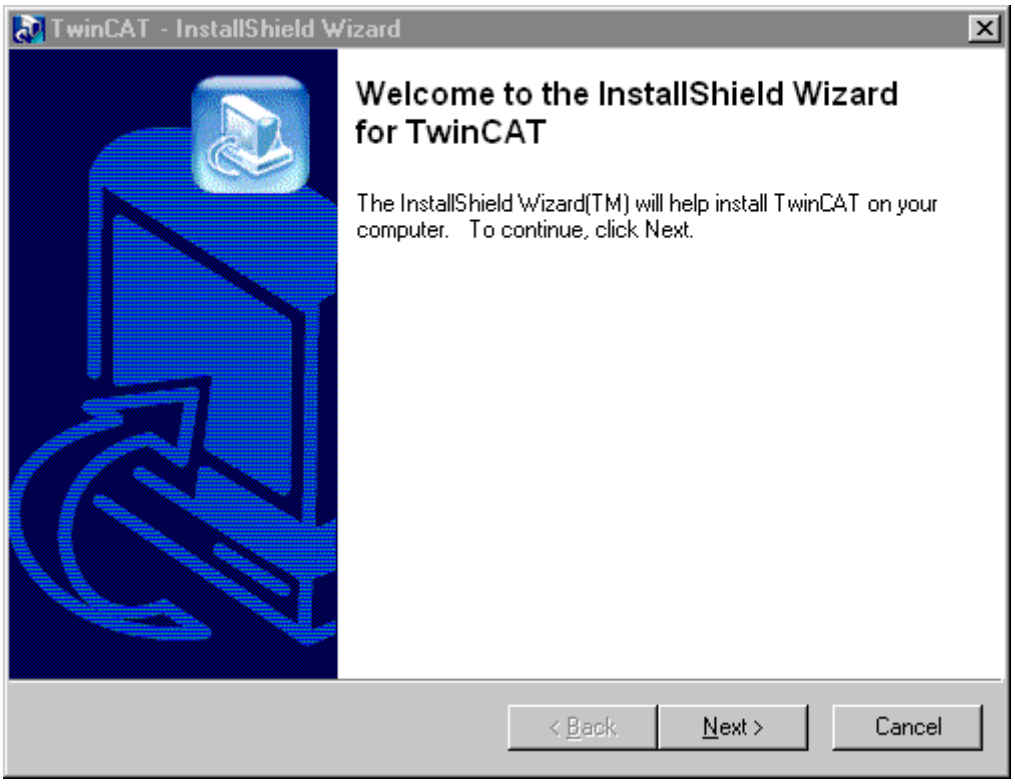
The required Hardware for the sample program is contained in the Beckhoff Demokits.

3 Installation

3.1 Starting the installation program

Start the SETUP.EXE program on the CD. To do this, open the Explorer, change to the CD ROM and double click the SETUP.EXE program.

The following dialog box opens. Click 'Next ' to continue.

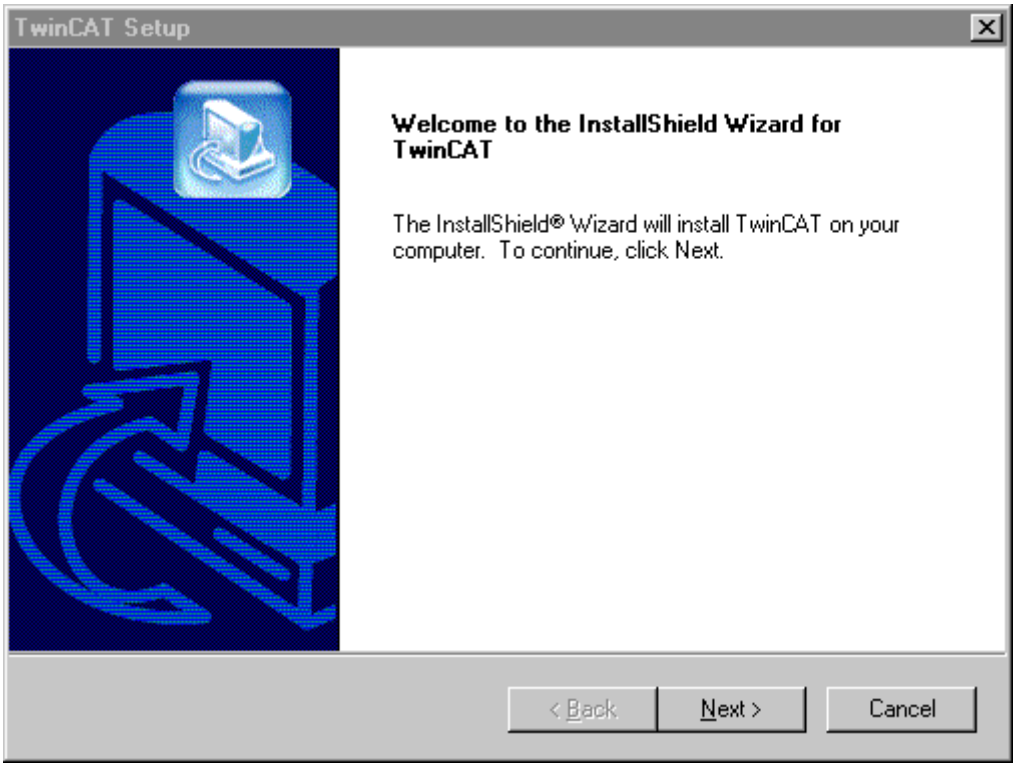


Selecting the language

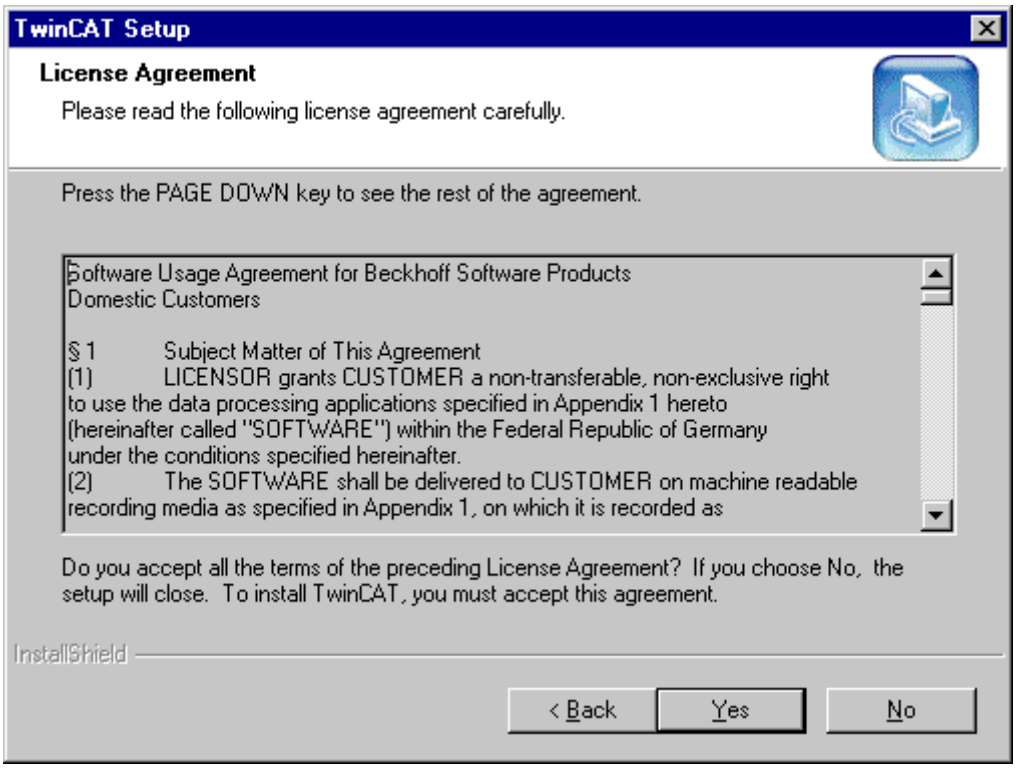
Select the national language in which you wish to install TwinCAT. To install it in English, for example, select the entry 'English' and click 'OK' to confirm your input. Installation is completely menu-prompted.

Exit programs

The installation program recommends, to exit all Windows programs, before running this Setup program.

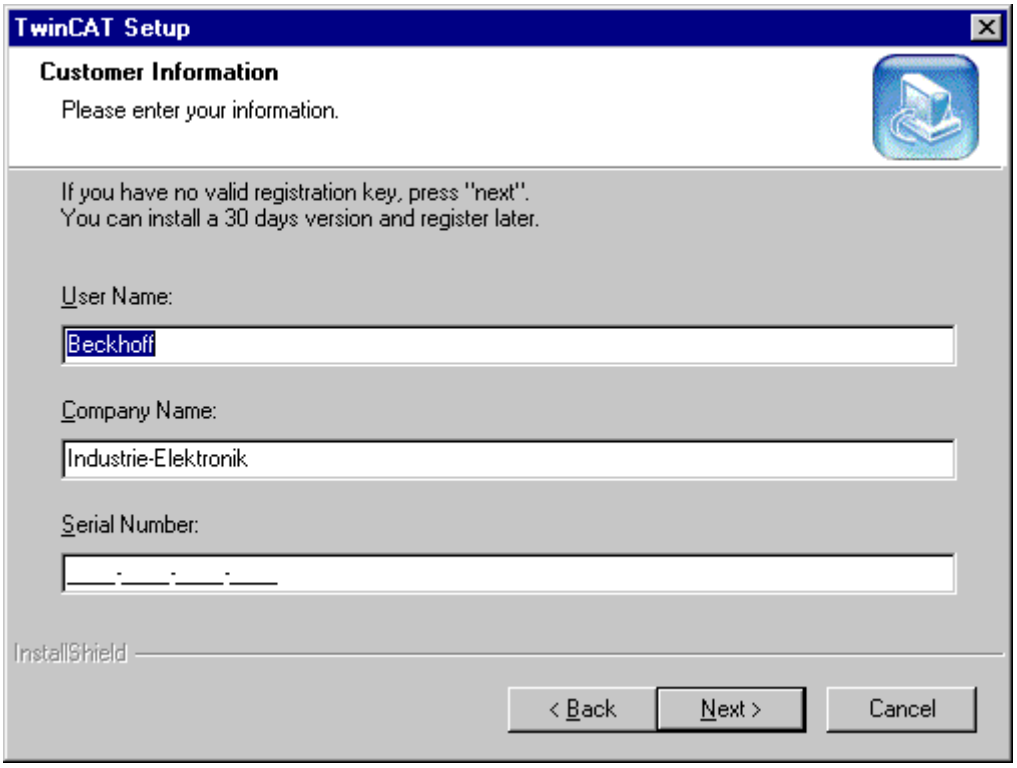


License Agreement



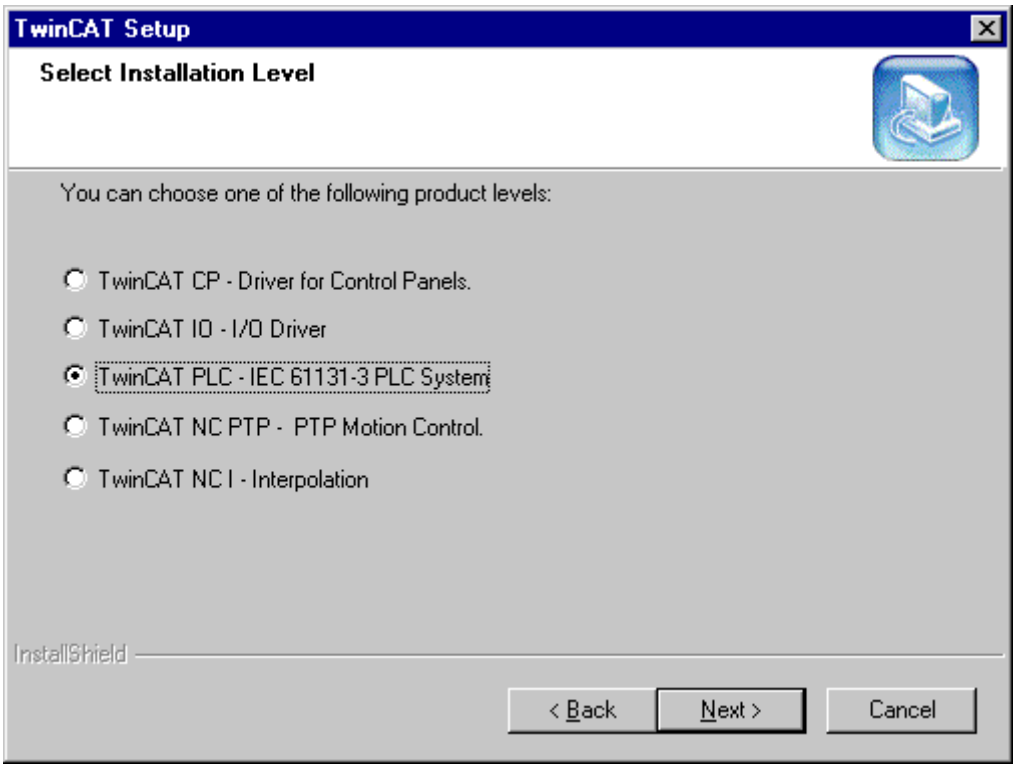
Entering the User Information

You must enter the serial number in this dialog box. You will find the serial number in your TwinCAT purchase agreement. Leave this box blank if you wish to install the demo version of TwinCAT.



Select installation level

One of the following Installation levels must be selected:



Description of the product levels:

TwinCAT CP

Contains the necessary components for the Special functions (UPS, S-Keys, ...) of the Beckhoff Control Panels

TwinCAT IO

The (User Mode-) Program can directly access the IO Devices. This Level does not include the PLC.

TwinCAT PLC

In TwinCAT PLC includes the IEC61131-3 Software Development kit.

TwinCAT NC PTP

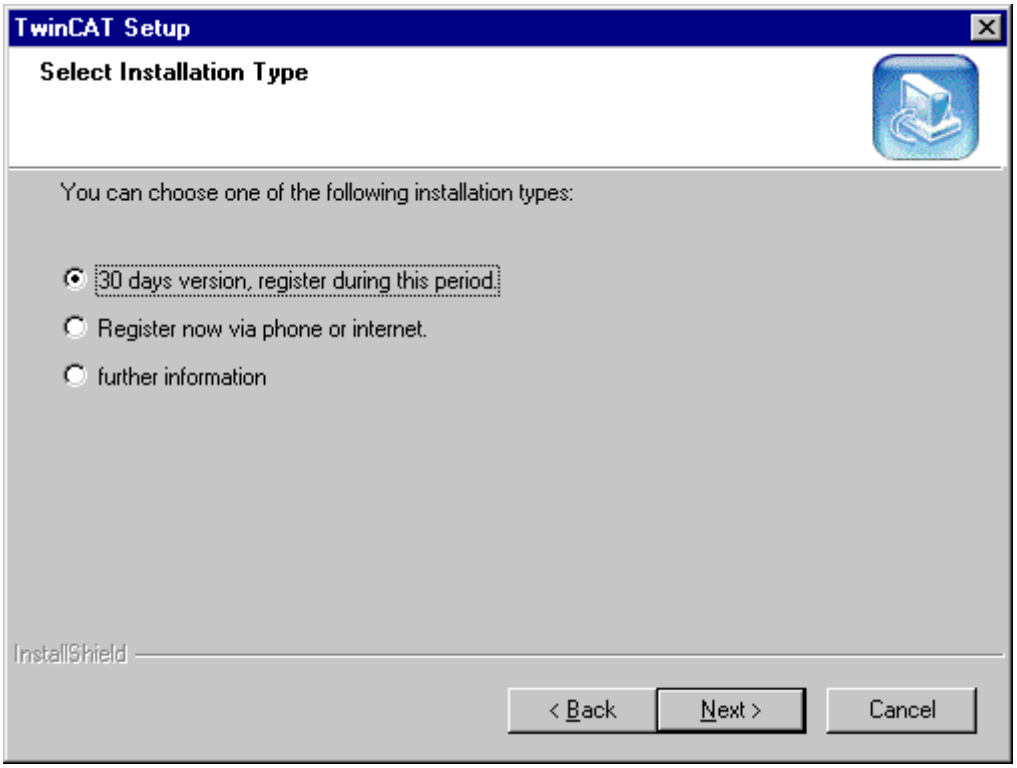
Further to PLC this module contains the NC/CNC functionality to control the PTP-Axis .

TwinCAT NC I

Further to PLC this module contains the NC functionality interpolates drives in 3D.

Installation type

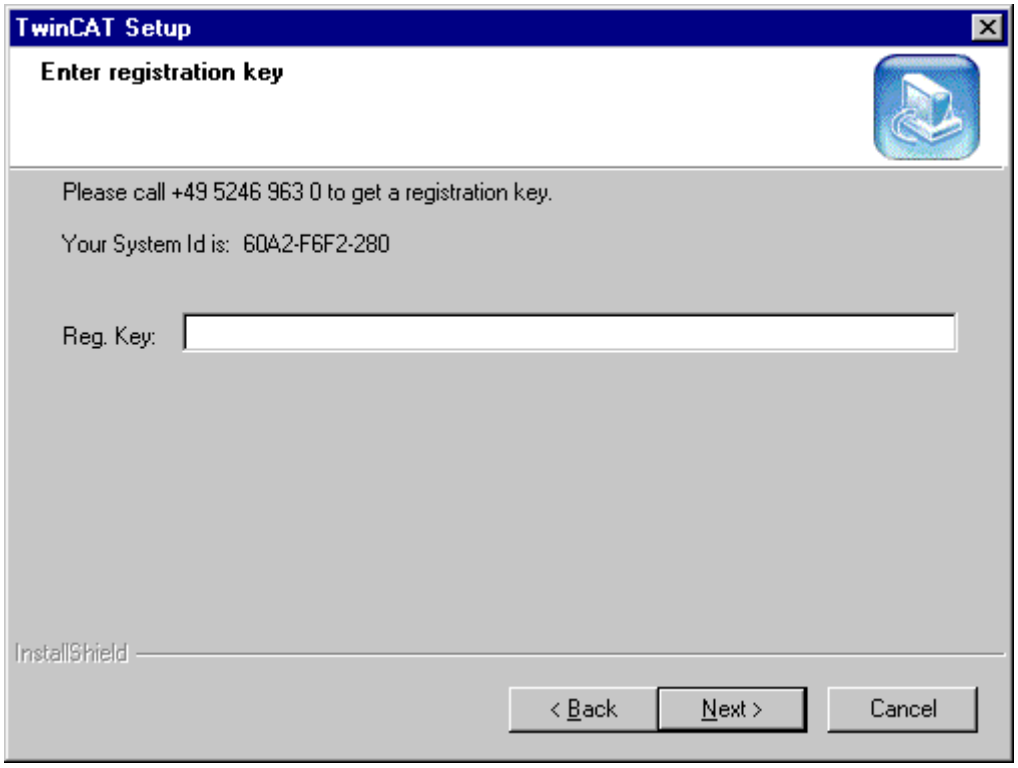
Select TwinCAT as 30-Day-Test-Version, Register an installation or a request for further Information. If you have no Registration Number 'Install the 30 day Version' and register during the 30 day period.



installation type	Restrictions
30 Day Version	TwinCAT is usable for 30 Days without any restictions . During this time a license number must be appled for otherwise after the 30 days has expired the program can no longer be started.
Registration via Telephone or Internet	When you have completed the installion you will be requested to enter the license number. See below

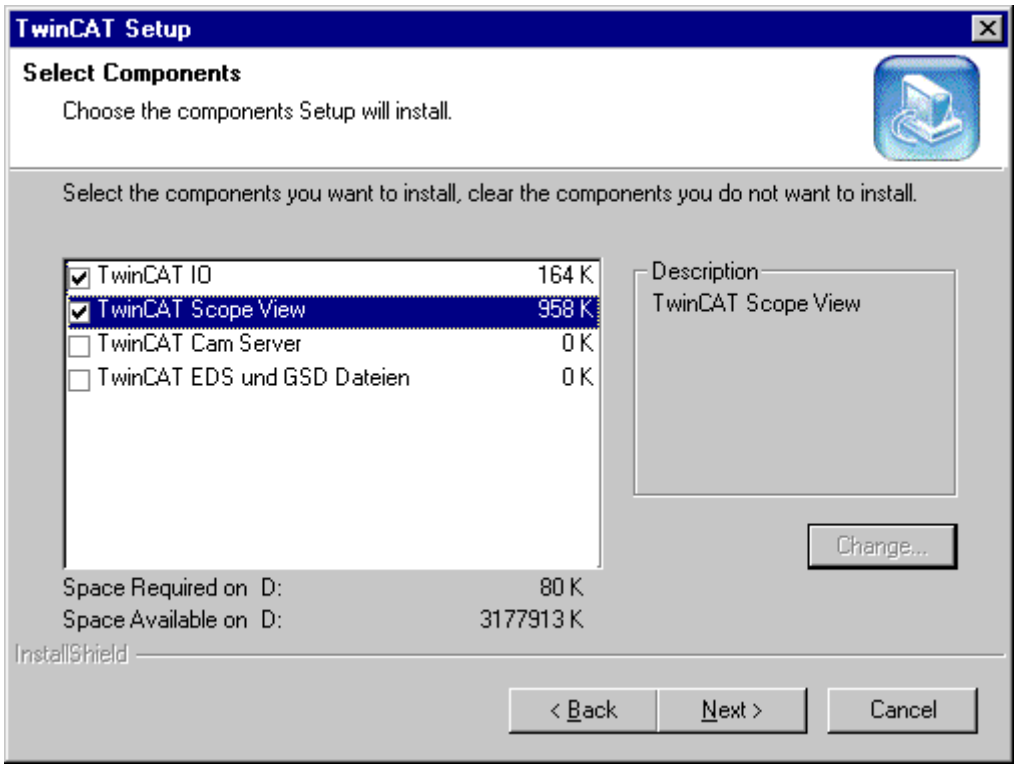
Registration key

If you decide to register TwinCAT, you must now enter the Key. You must obtain this register key directly from Beckhoff Automation. The Telephone number is in the dialog box. In order to compute the registration number, the System ID must be given. The System ID is shown in the dialog box.



Component selection

By Default, not all the components in TwinCAT are installed.

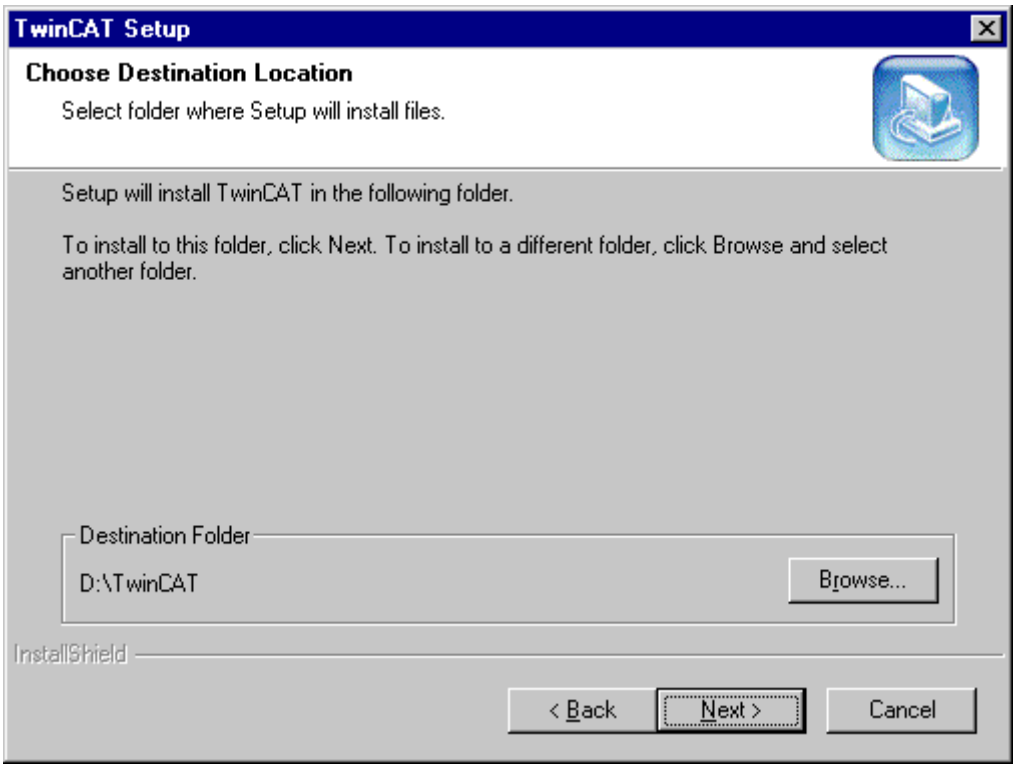


Requirements

Component	Description
TwinCAT IO	Allows the direct access to IO via a DLL. Can be installed with TwinCAT PLC or TwinCAT NC PTP.
TwinCAT Scope View	Program for the graphical visualisation of the TwinCAT process variables.
TwinCAT Cam Server	Fast Cam Server.
TwinCAT EDS and GSD Files	The EDS (DeviceNet) and GSD (characteristic master device file, Profibus) makes all the settings available to the user for the configuration of his system.

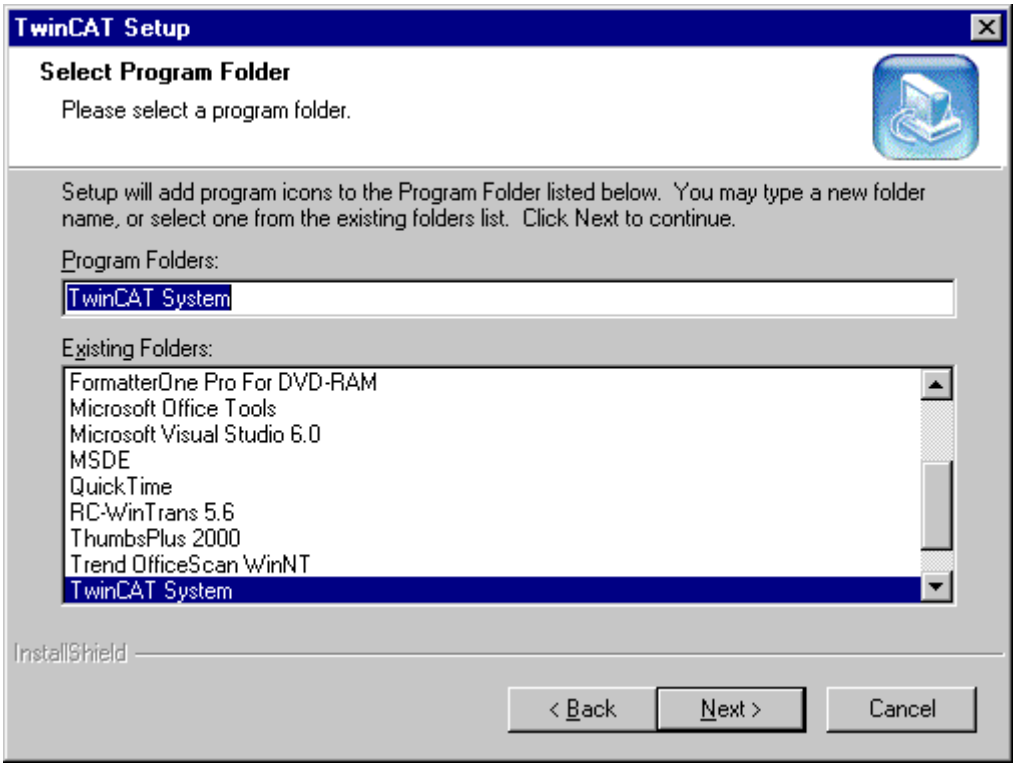
Selecting the destination path and the program folder

You can select any directoy here as well as selecting the Program folder. Usually the default values would be accepted.



Select program folder

You can select any directoy here as well as selecting the Program folder. Usually the default values would be accepted.



After the installation of TwinCAT

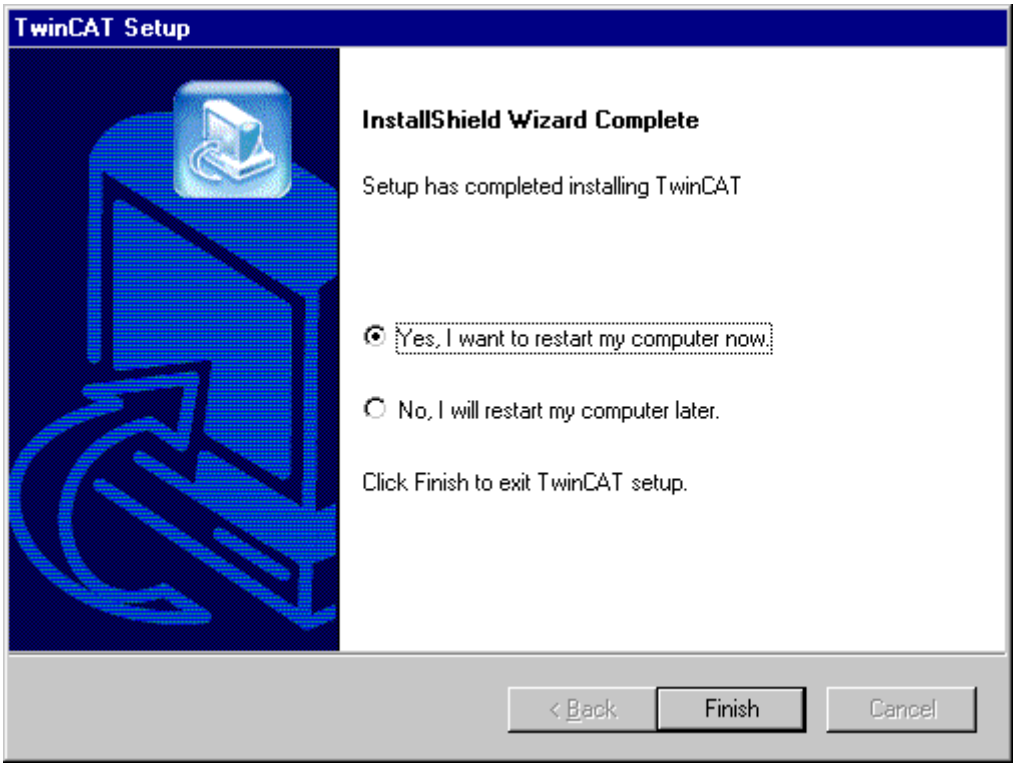
The installation of the Beckhoff Information System is automatically started. The Beckhoff Information System contains the documentation of TwinCAT. Click 'OK' to start this installation.

End of the installation of the Beckhoff Information System

Click 'Finish' to finish the installation of the Information System. Afterwards also the installation is terminated by TwinCAT.

Restart the computer

After the installation has been completed, the Computer must be restarted.



Now, the setup has completed the installation of TwinCAT.

Silent installation

You have the option of letting InstallShield create the response file for you. Simply run your setup with theSetup.exe-r command line parameter. InstallShield will record all your setup choices in Setup.iss and place the file in the Windows folder.

All InstallShield built-in and Sd dialog box functions are designed to write values into the Setup.iss file when InstallShield runs in record mode (Setup -r). If you are creating custom dialog boxes, you will need to call SdMakeName and SilentWriteData to add sections and dialog box data to the response file when setup runs in record mode. Refer to the Sd dialogs' source code in the <InstallShield location>\Include folder for examples of using these functions to write to Setup.iss. Please read the following section for more information about what data to add to Setup.iss when calling SdMakeName and SilentWriteData.

After you have created the setup and the response file, do the following:

1. Place the response file (which is located in your Windows folder) in the Setup Files pane's Advanced\Disk 1 folder.
2. Build your media. If you are creating a self-extracting executable for your setup, enter -s in the Setup Command Line Parameters edit box of the Media Wizard's Self-Extracting Package panel or the media property sheet's Packaging page.

Now you are ready to run the setup in silent mode using InstallShield Silent. When running an setup in silent mode, be aware that no messages are displayed. Instead, a log file namedSetup.logcaptures setup information, including whether the setup was successful. You can review the log file and determine the result of the setup. (Note that for certain setup initialization errors, the log file may instead be named Setupexe.log and be created in SUPPORTDIR if the setup is run from the Internet or in SRCDIR otherwise.)

To launch InstallShield Silent, runSetup.exewith the -s option.

InstallShield also provides the -f1 and -f2 switches so you can specify the name and location of the response file and the location of the log file.

To verify if a silent setup succeeded, look at the ResultCode value in the [ResponseResult] section of Setup.log. InstallShield writes an appropriate return value after the ResultCode keyname.

Setup.log is the default name for the silent setup log file, and its default location is Disk1 (in the same folder as Setup.inx). You can specify a different name and location for Setup.log using the -f1 and -f2 switches with Setup.exe

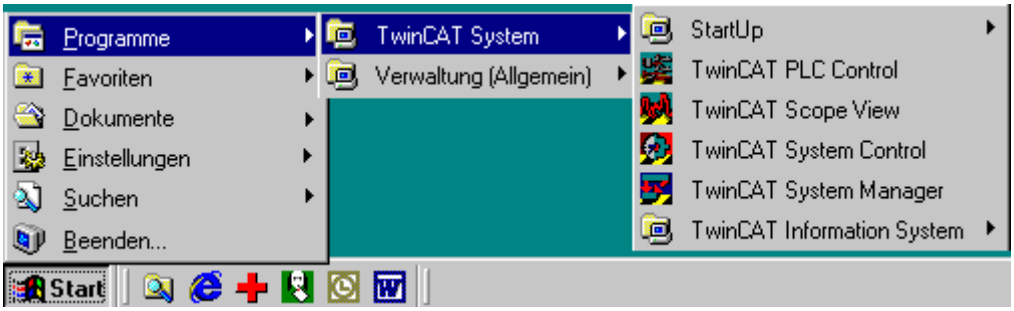
The second section, [Application], identifies the installed application's name and version, and the company name.
The third section, [ResponseResult], contains the result code indicating whether or not the silent setup succeeded. An integer value is assigned to the ResultCode keyname in the [ResponseResult] section. InstallShield places one of the following return values after the ResultCode keyname:

0 Success.
-1 General error.
-2 Invalid mode.
-3 Required data not found in the Setup.iss file.
-4 Not enough memory available.
-5 File does not exist.
-6 Cannot write to the response file.
-7 Unable to write to the log file.
-8 Invalid path to the InstallShield Silent response file.
-9 Not a valid list type (string or number).
-10 Data type is invalid.
-11 Unknown error during setup.
-12 Dialog boxes are out of order.
-51 Cannot create the specified folder.
-52 Cannot access the specified file or folder.
-53 Invalid option selected.

3.2 End of installation

New Program icons

After the installation the Windows NT/2000 Startup menu will contain a new folder containing five program symbols and two further program folders.



StartUp

TwinCAT automatically starts all programs in the Startup folder if activated via the Auto-Boot Function. This is to ensure that user programs that use process values from TwinCAT are started after TwinCAT.

TwinCAT System-Manager

With the help of this program, assign physical I/O adresses (Fieldbus) to the logical process variables (PLC program). This assignment is called mapping. The realtime properties are defined here.

TwinCAT PLC Control

This is the software development kit for IEC61131-3. The PLCPrograms are written and tested here.

TwinCAT System Control

Apart from the visible programs, there are also hidden tasks and drivers that run in the background. The TwinCAT System Control administrates these programs.

TwinCAT Scope View

With the help of the TwinCAT Scope View process values can be displayed graphical in real time. The dynamic Axis values can be thoroughly examined.

TwinCAT in Windows NT/2000

When the System is started the TwinCAT Real time server icon is displayed on the right of the Taskbar.

The color indicates the general operating state of the system. It can have one of the states 'started' (green), 'starting' (yellow)' and 'stopped' (red). If you click the icon, a pop up menu opens in which you can define further system settings. Within the scope of these instructions, you can accept the default setting. The TwinCAT server can be stopped and started in this menu.

Beckhoff Information System

The Beckhoff Information System is a constantly growing reference source for TwinCAT products. It contains technical information, manuals, example code, the TwinCAT Knowledge Base and much more. The hierarchical arrangement of the documents makes it easy to find the required information.

Full version:

If you installed TwinCAT of the Beckhoff product CD, now the entire Beckhoff Information System is installed on your computer.

Base Version

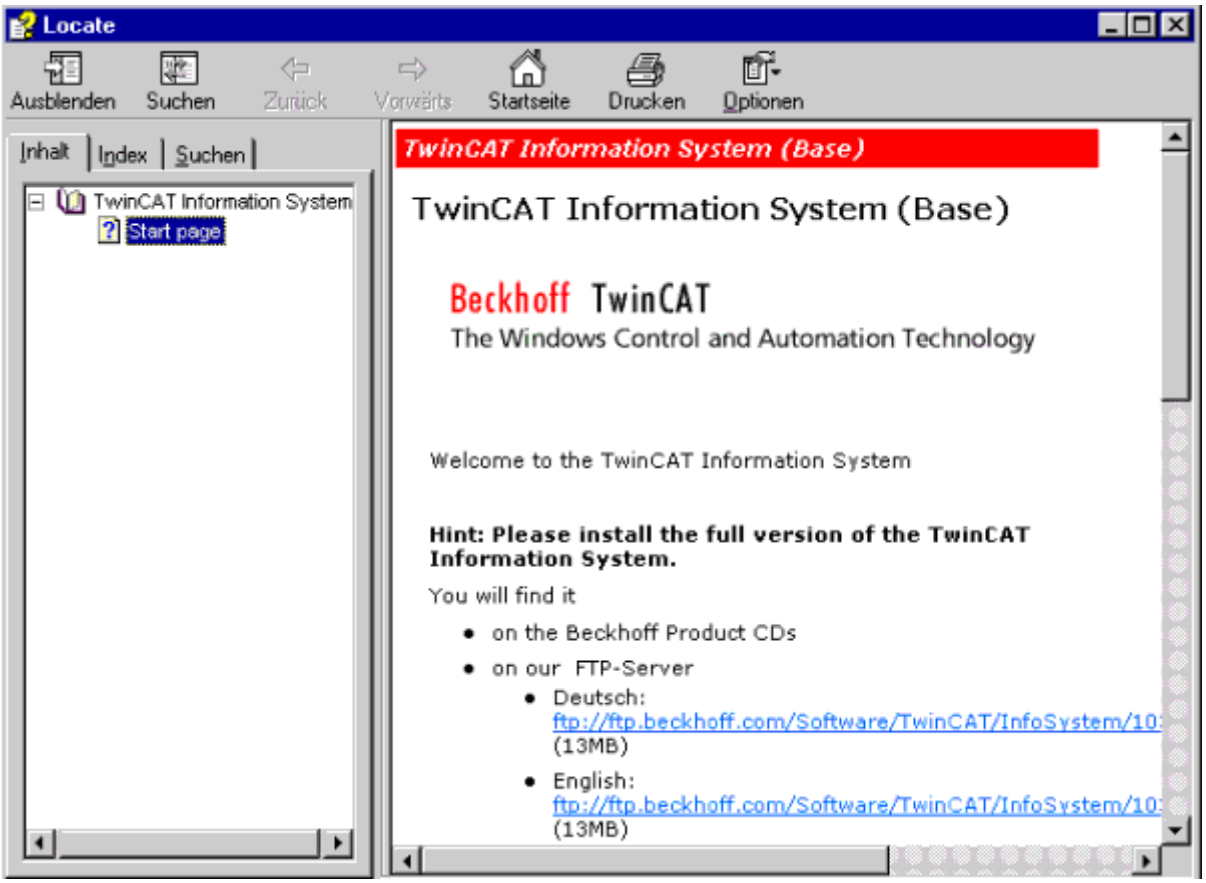
If you loaded yourselves the installation from TwinCAT from the Internet, only a basic version of the Beckhoff Information System on your computer is installed.
In order to receive a complete version (size 200 MB), you have several possibilities:

You will find it

- on the Beckhoff Product CDs
- on our FTP-Server

- http://download.beckhoff.com/download/Software/TwinCAT/TwinCAT2/InfoSystem/1033/Install/InfoSys.exe
- and on our Web-Server
  - http://www.beckhoff.com

You can infer the statements for the installation also from the start page, which you can open with the start menu, as represented above.



4 TwinCAT PLC Control

What is TwinCAT PLC Control?

TwinCAT PLC Control is a complete development environment for your PLC. Use of the editors and debugging functions is based upon the proven development program environments of advanced programming languages.

IEC 61131-3

TwinCAT PLC allows the PLC programmer to obtain easy access to the powerful language resources of IEC 61131-3. The following features were implemented during the course of TwinCAT PLC development:

Function blocks

TwinCAT PLC supports different programming languages: instruction list (IL), structured text (ST), sequential function chart (SFC), function block diagram (FBD) continuous function chart editor (CFC) and ladder diagram (LD).

Test without PLC

The integrated software PLC allows you to test the PLC program without external hardware.

Changes during operation

Programs are modified "online" in the PLC.

Reusability

Reuseability of existing PLC program blocks

Standardized interfaces

A link to other programs and computers, even through a network, is possible thanks to standardized open interfaces (OCX, DLL, etc).

Heterogeneous environment

Thanks to the use of system-independent and widespread network protocols, it is possible to integrate TwinCAT into a heterogeneous network environment. For example, an Oracle database running under UNIX can exchange data with TwinCAT via TCP/IP and can process this data further in a PDA or PPS system or can specify parameters within TwinCAT PLC in order to influence the production process.

High-level language libraries

Complex algorithms can be developed in C/C++ or Assembler, for example, in order to address them from TwinCAT PLC. There are many third-party vendors of such libraries who deal with specific task areas.

SCADA systems

Some manufacturers of SCADA systems (Fix32, InTouch, Citect, Genesis, Wizcon, etc) offer direct driver support for linking to TwinCAT

Remote access

Central programming of distributed control systems through a network (including ISDN) is possible thanks to the fact that the programming and run time environments are separate.

Intuitive development environment

Simulation according to the example of technically matured high-level language development environments (e.g. visual C++), breakpoints, single step mode and tracing of variables etc. are possible using TwinCAT PLC, as is the case with modern development environments.

4.1 The PLC standard IEC 61131-3

Five different languages conforming to IEC 61131-3 are available for creating a PLC program using TwinCAT PLC.

Instruction list (IL)

The instruction list is very similar to the STEP5 programming language. Every instruction begins in a new line and contains one operator and one or several operands. An instruction may be preceded by a label, followed by a colon. A comment must be the last element in one line.

Example:

label	operator	operand	comment
Start:	LD	Basin_level	(* Load level *)
	GE	13	(* Limit reached? *)
	JMPC	Pump_on	
	R	Pump_control	(* Pump off *)
	JMP	End	
Pump_on:	S	S Pump_control	(* Pump on *):
End:			

Structured text (ST)

In the case of this programming language, we also speak of a higher-level programming language because it is not "machine-oriented" commands that are used. Instead, power command strings can be created by way of abstract commands. From the area of the PC, Basic, PASCAL and C are comparable higher-level programming languages.

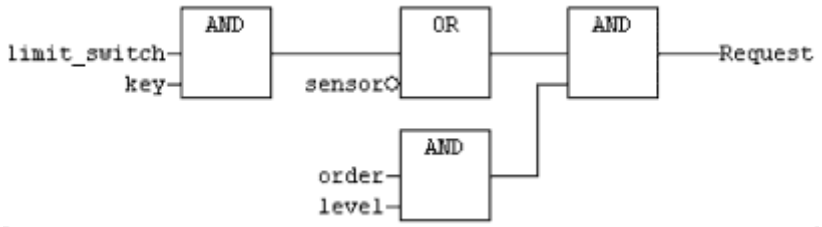
Example:

operator	operand	comment
CASE Temperatur_furnace OF		(* control heating output *)
	60..99: Heating := 80;	(* 80% *)
	100..149: Heating := 60;	(* 60% *)
	150..199: Heating := 35;	(* 35% *)
	200..250: Heating := 10;	(* 10% *)
ELSE Alarm := TRUE;		(* Set alarm *)
END_CASE;		

Function block diagram (FBD)

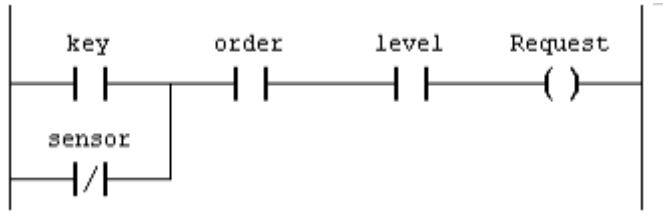
The basic idea behind PLC programming with the function block diagram is that the program is structured in function-oriented logical sequence cascades (networks). Within one network, the executed direction is always from left to right. All input values must always have been generated before execution of a function block. Evaluation of a network is not concluded until the output values of all elements have been calculated.

Example:



Ladder diagram (LD)

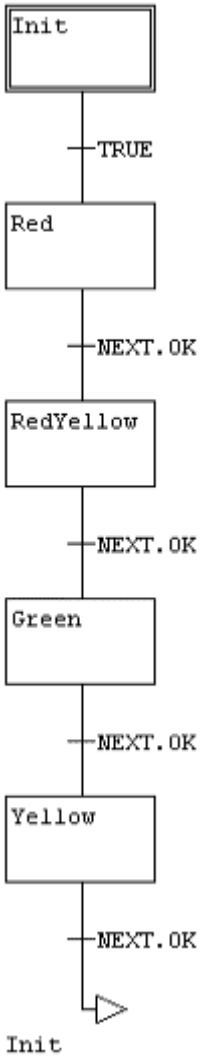
The representation of logical sequences in the form of the ladder diagram originates from the area of electrotechnical plant engineering. This mode of representation is particularly suitable for implementing relay switching operations in PLC programs. Processing is limited to the Boolean signals 1 and 2.



Sequential function chart (SFC)

The sequential function chart is expedient wherever sequencer program-ming is necessary. Complex tasks are split into clearly arranged portions of program (steps). The sequence between these steps is then defined graphically. The steps themselves are created in a different programming language (ST, IL,…) or can also be represented in SFC again.

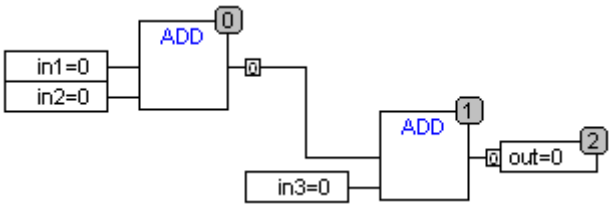
Example:



SFC programs essentially consist of steps, transitions and their links. Each step is assigned a set of commands. These commands are executed when the step is active. A transition must be fulfilled to ensure that the next step is executed. The steps and the transitions can be formulated in any chosen language.

Continuous function chart Editor (CFC)

The continuous function chart editor does not operate like the function block diagram FBD with networks, but rather with freely placeable elements. This allows feedback, for example. An example of a network in the continuous function chart editor, as it would typically appear:



5 TwinCAT System Manager

What is TwinCAT System Manager?

The TwinCAT System Manager is the central tool for the configuration of the TwinCAT System. The inputs and outputs of the participating software tasks and the physical inputs and outputs of the connected field busses are managed by the TwinCAT System Manager. Additionally the online values of the active configurations can be regarded.

The logical inputs and outputs are assigned to the physical ones by logically linking variables of the software tasks and variables of the field buses.

Configuration modules of the TwinCAT System Manager

In the following, the main components of the TwinCAT System Manager are listed. The existence of these components depends on the level of the installed TwinCAT System.

Realtime Configuration

Realtime configurations and creation of user defined tasks.

PLC Configuration

Under this entry, all PLC projects that are running on the local system are listed (currently up to four projects).

Cam Configuration

Electronical Cam Server and its configuration.

I/O Configuration

Corresponding field bus interface cards are necessary to link the control to the process level. Which cards are used must be defined under this entry.

6 TwinCAT Scope View

What is Scope View?

TwinCAT Scope View is an analysis tool providing graphical display of the variables related to various PLC tasks.

Curves can be drawn versus time, or you can choose to see XY displays.

Each Scope View can utilise several channels, the number being limited only by the memory size and computing capacity. For time displays, one variable is assigned to each channel.

Scope View analysis

Tools are available for analysis within Scope View.

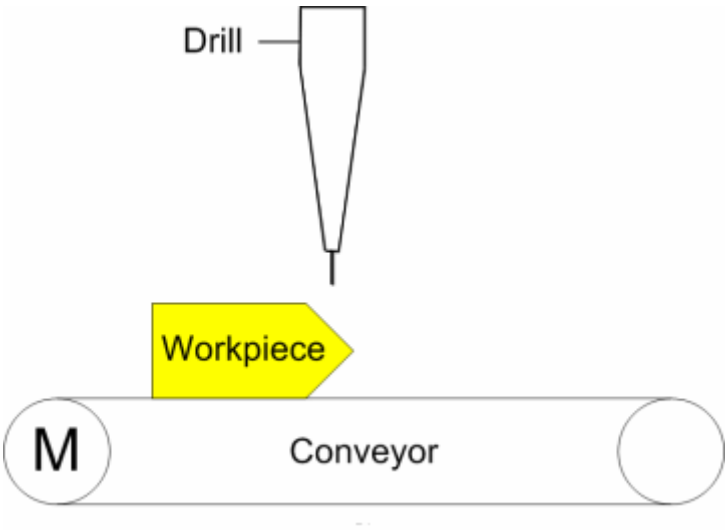
Data backup

TwinCAT Scope View permits data to be saved in a number of data formats, such as in an Excel table.

7 Example Program

7.1 Sample Maschine.pro

Creation of applications using TwinCAT will be explained with reference to a sample program. This program represents a machine tool for any chosen workpieces. After you have installed TwinCAT, you will find it in the '\TwinCAT\Samples\First Steps' directory. It has the name 'Maschine.pro'.
Sketch:



Description

1) The conveyor belt is moved by 25 steps.

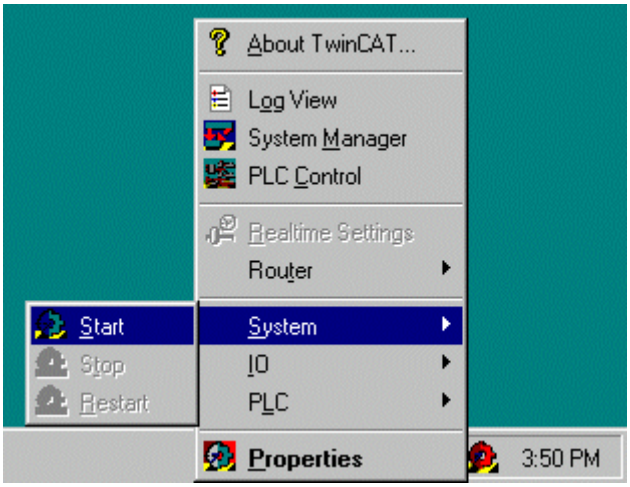
2.) The drill is moved down for 2 sec.

3.) The drill is moved up for 2 sec.

4.) Begin at step 1 again.

Start TwinCAT:

Before you are able to execute the program, you must activate the TwinCAT real time server.



To do this, click on the TwinCAT real time server icon and activate the "Start" command from the 'System' menu. The color of the icon changes through yellow to green, which means that the TwinCAT real time kernel is active.

Starting TwinCAT PLC Control :

Start now the programming surface from the TwinCAT PLC. Then click with the mouse 'start' -> 'programs' -> 'TwinCAT system' -> 'TwinCAT PLC control'.

Open Project:

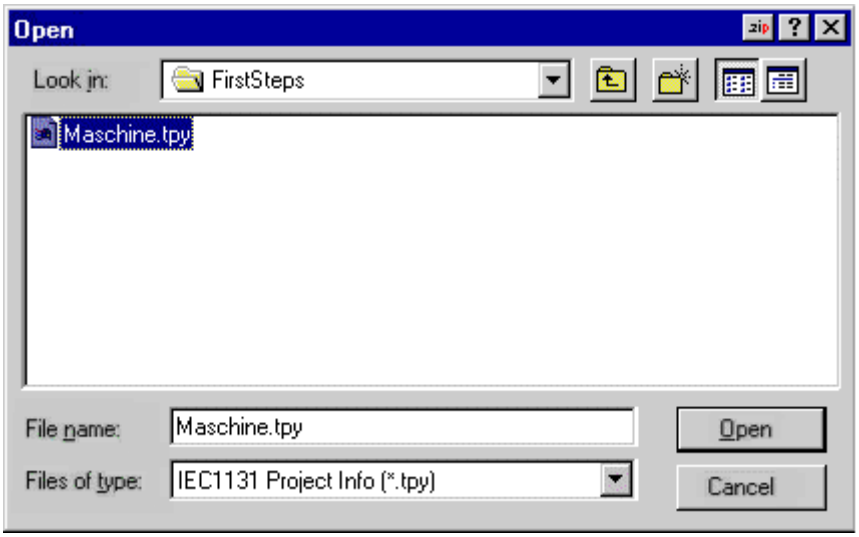
A PLC project is stored in a file on the hard disk or on a diskette that bears the name of the project. To open a project, select the 'File' menu item and then the 'Open' command.

Select directory:

Switch to the directory specified above by clicking the symbol shown on the left in the dialog box. Double click the 'Samples' entry. Then proceed in exactly the same way with the 'First steps' entry.

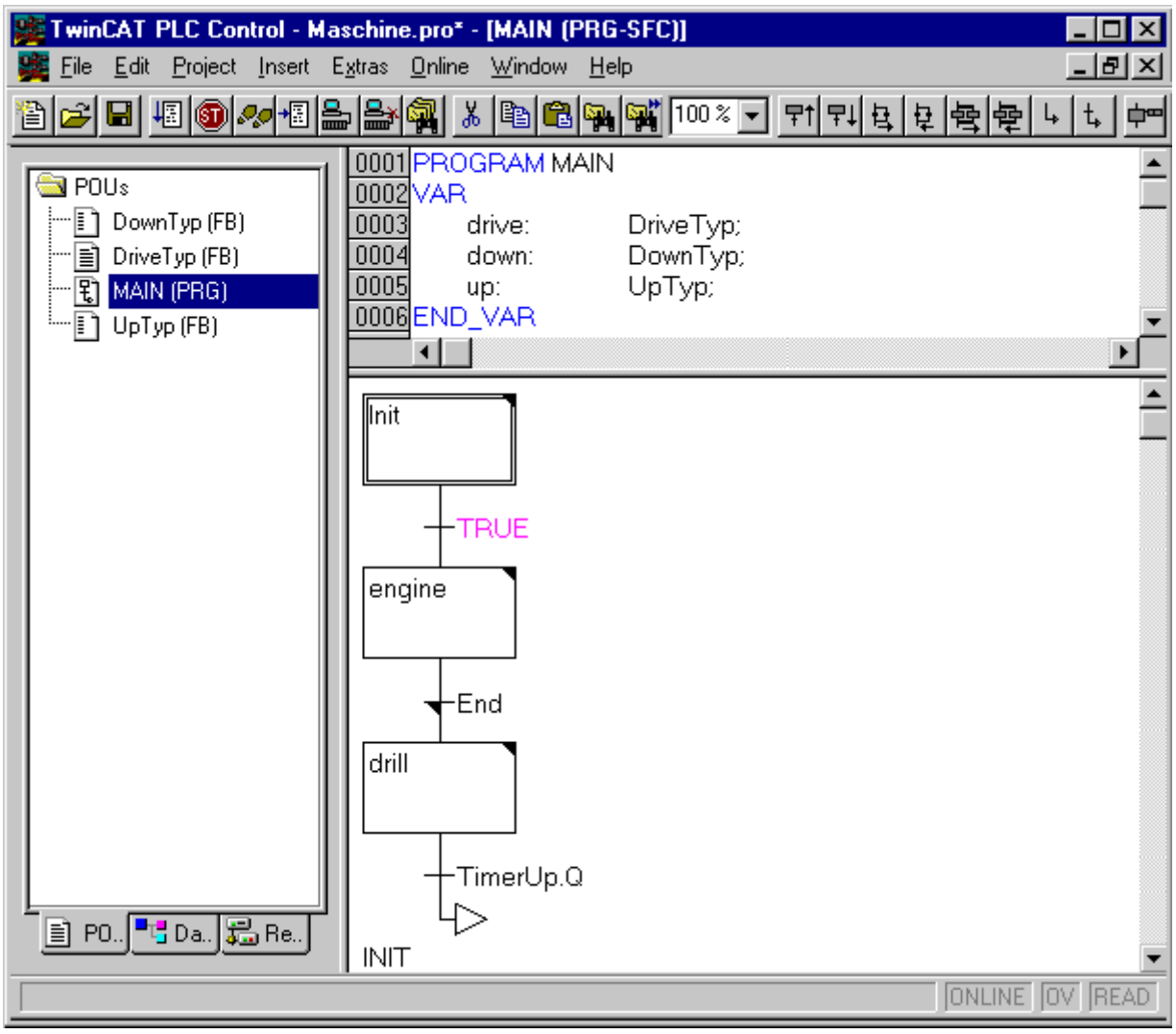
Select project:

Select the 'Maschine.tpy' project by clicking the entry in the dialog box with the mouse and by then executing the 'Open' command.



Items of the PLC Control:

After opening Maschine.pro select the POU (programm object) MAIN and make a double click with the left mouse button. The following dialog box opens:



In the upper blue beam the project name Maschine.pro is located. 

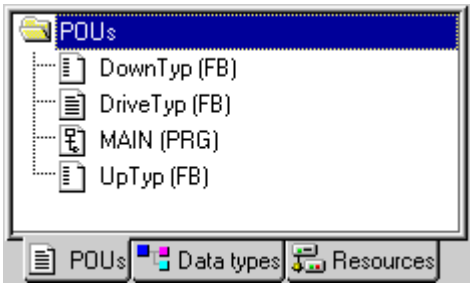
Among them there´s the command menu 

and the toolbar.

The lowest grey beam contains the status line. 

The dialog window divides into three individual windows, it contains the object list, the variable declaration and the program representation.

Object list:



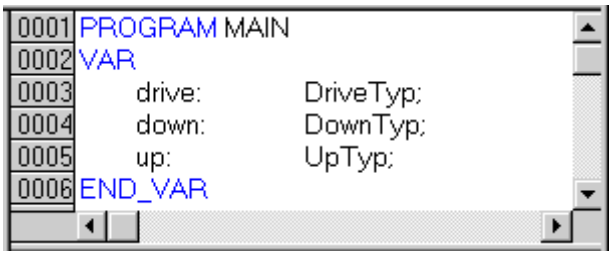
TwinCAT distinguishes three kinds of basic objects in a project:

· (Program) Blocks

· Data Types

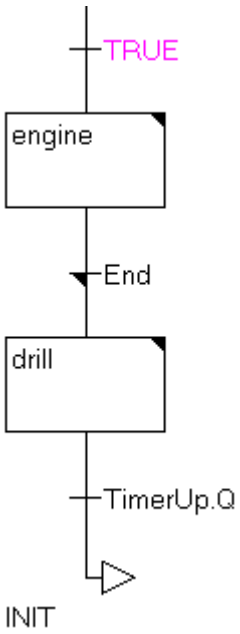
· Resources

Variable declaration:



A PLC program stores its data in variables. Variables are comparable to flag words or data words. Before a variable can be used, it must be declared, i.e. its affiliation to a specific data type (e.g. BYTE or REAL) must be made known. Declaration also involves defining specific attributes such as battery buffering, initial values or affiliation to physical addresses. If a variable is not needed in the input or output image, i.e. only within the PLC program, the PLC programmer need not worry about the storage location of the data. This is taken care of by TwinCAT. This avoids unintentional overlapping of flag words/data words as was possible in previous systems (side effects). As in the case of variables, function blocks also have to be declared (instance). In the example, one instance each (drive, down and up) of the three functions 'DriveType', 'DownType' and 'UpType' are created. After instancing, the instances can be used and can be activated.

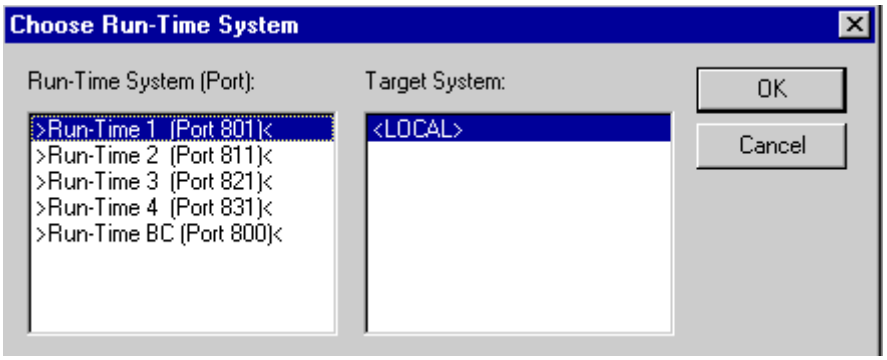
Program representation:



The actual PLC program is entered and represented in this area of TwinCAT PLC control.

Select destination system

TwinCAT provides up to 4 run time systems. Each of these run time systems is capable of executing PLC programs in IEC 61131-3 independently of the other run time systems. To use the 'Choose run time system...' command in the 'Online' menu to define which run time system you wish to run your programs on.



After installation of TwinCAT, only one run time system is released and so only the first run time system (Run Time 1) is displayed in this dialog. Select OK to confirm the selection.

If the PLC runtime is not on the local computer, you must first establish a connection to the desired target system (see TwinCAT documentation).

Logging in:

You have now loaded the PLC program in TwinCAT PLC control and you are able to execute it. Make sure the TwinCAT real time server is active. This is recognizable by the fact that the TwinCAT real time server icon is displayed in green on the bottom right of the screen. Before you start a PLC program, you must link TwinCAT PLC control to the run time system, i.e. you must 'log in' with the control system. Execute the 'Log in' command in the 'Online' menu. As there is still no PLC program in the run time system, you get the message: No program on the controller! Rebuild All? Acknowledge the question with ' OK'.

The current status of the connection is displayed in the status line:

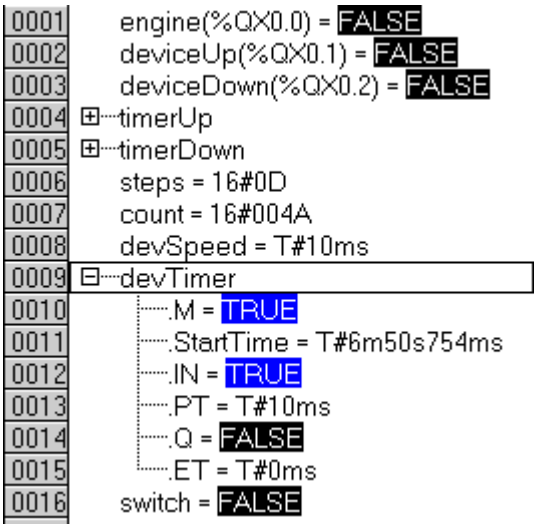
Starting PLC program:

You start the PLC program in the TwinCAT real time server by selecting the 'Start' command from the 'Online' menu. The word "RUN" displayed in the status line darkens. You should also see that individual steps within the sequential function chart are temporarily displayed in blue.

A step shown in blue is currently being executed, i.e. it is the active step.

Tracing the PLC program sequence:

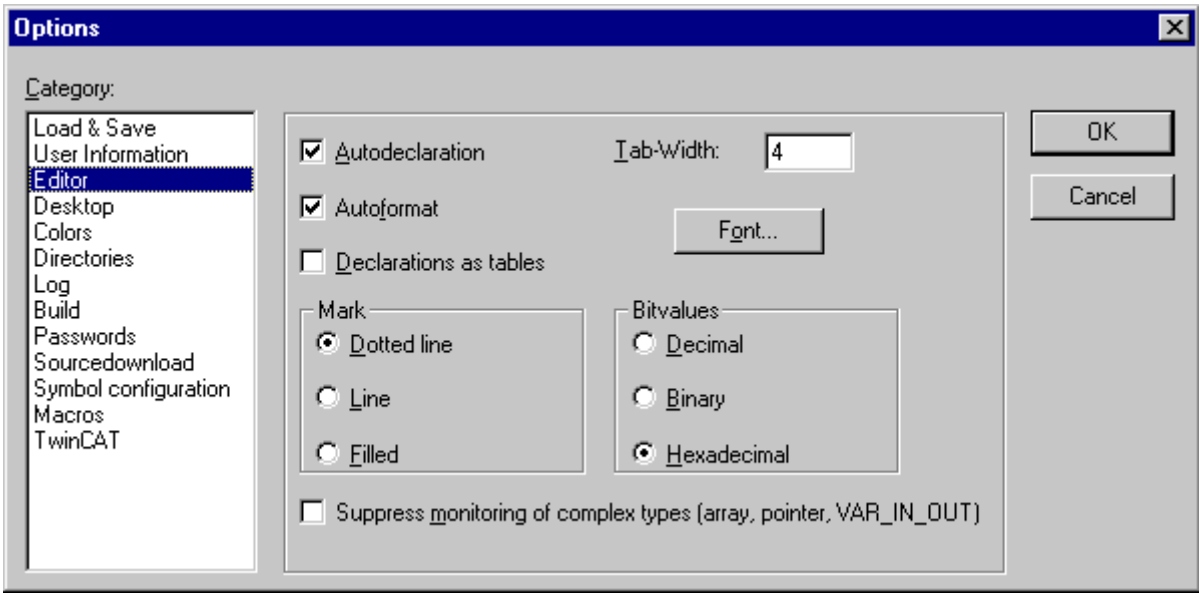
When you activate the 'Global variables' window by clicking the Resources Tab at the bottom of the 'Object List' window, and then double clicking the 'Global Variables' object, you will see all globally declared variables. Global variables can be used jointly by all program objects (POU).



Besides the variables, the 'timerUp', 'timerDown' and 'devTimer' function blocks are also displayed there. A lozenge is visible before the function names. When you double click the lozenge, a tree-like display opens in which all variables of the function are displayed.

Changing the numeric representation

You can display the contents of the variables in various numeric systems. You have a choice between decimal, hexadecimal and binary. If you wish to change the display, you must select the 'Options' command in the 'Project' menu. The current setting is indicated by a tick mark before the corresponding entry.



Ending the program:

You have now loaded a PLC program into TwinCAT PLC control (IEC 61131-3 programming environment) and you have executed it on the TwinCAT PLC server (run time system). Now end the PLC program. To do this, select the 'Stop' command in the 'Online' menu.

Logging out:

In the next sections, we will add to the PLC program. To do this, you must log out of the TwinCAT PLC server. To do this, execute the 'Log out' command in the 'Online' menu.

Viewing program text:

This example was programmed in the various IEC 61131-3 programming languages. The main part of the program was created in the sequential function chart (SFC). It contains the steps

· Init

· Engine

· Drill

And the transitions:

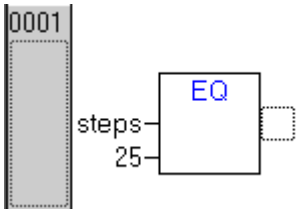
· TRUE

· End

· TimerUp.Q

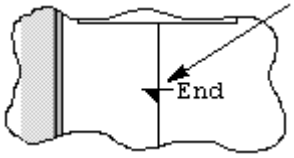
Viewing transitions:

The 'TRUE' transition is constantly fulfilled because the 'TRUE' key word is a system-wide constant and is permanently fulfilled. The 'Engine' step is executed unconditionally after the 'INIT' step. TimerUp.Q means that the variable Q in the Up function must be TRUE (or also 1) for this transition to be fulfilled. 'End' is a transition that contains further program text. When you double click the transition, a further window opens in which the corresponding program text is displayed.



In the 'End' transition, a comparison is made as to whether the 25 steps of the motor have already been reached. If this is the case, the program changes in the next cycle from the 'engine' step to the 'drill' step.

If a step or a transition contains further program text, this is indicated by a small black triangle.

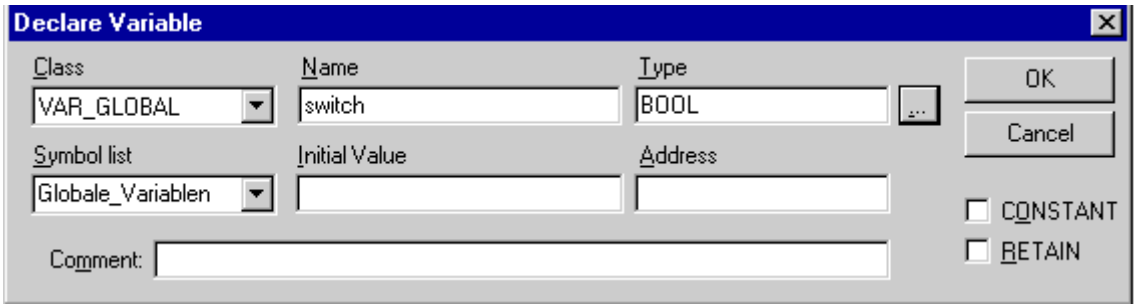


Modifying the PLC program

Switch back to the 'MAIN' window. At this point, you should modify the PLC program in such a way that the motor's cycle speed can be modified in two stages (fast/slow) by way of a variable. Open the 'Drive Type' function block in the object list by double clicking it. Move the input cursor to the first line and enter the following text:

```
IF switch = TRUE THEN
```

When the Enter (Return) key is pressed, a dialog box appears which you must fill out as shown below.



When you select 'OK', the 'switch' variable is added to the variable list of MAIN. If this dialog box not opens, your auto declare is deactivated. (You can activate it by selecting autodeclaration under 'Options' 'Editor' in the menu 'Project'.) Open 'Auto declare' in the menu 'Edit'.

Then also enter the following program lines:

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

The window must then have the following contents:

```
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
```

If the 'switch' variable is set, the 'devSpeed' variable is set to 25 ms, or otherwise to 10 ms. The result of this is that the pulse and pause duration of the clock pulse generator in the following program lines either amount to 25 ms or 10 ms.

Saving the program:

Save the program by selecting the 'Save ' command in the 'File' menu.

Compiling the program:

Before a program can be transferred to the TwinCAT PLC server, it must be compiled, i.e. it must be converted from its textual or graphical representations to a form that is understandable to the control system. To do this, select the 'Rebuild all' command from the 'Project' menu item.

Starting the program:

Log in with the control system and start the PLC program. You see that the 'switch' variable is set to 'FALSE' when starting.

Modifying variable values:

You have the possibility of modifying values of variables while the PLC program is running. Open the 'Global variables' window and double click the 'switch' entry. The display changes from FALSE to TRUE and the lettering turns red. Up to this time, however, the value in the TwinCAT PLC server has not yet changed. To do this, you must execute the 'Write values' command in the 'Online' menu item. The lettering turns black again and the 'devSpeed' variable changes to 10 ms.

Tracing the program sequence:

Follow the program flow with TwinCAT Scope View.

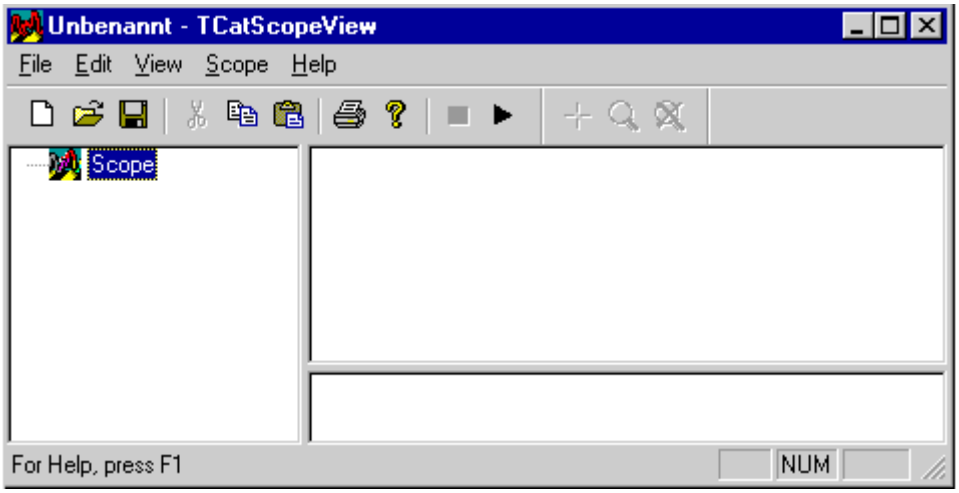
7.2 Following Program Flow

TwinCAT Scope View is used for recording and analysis of the program.

Open TwinCAT Scope View:

Scope View can only be opened through the start menu.

Use the mouse to select 'Start' -> 'Programs' -> 'TwinCAT System' -> 'TwinCAT Scope View'.

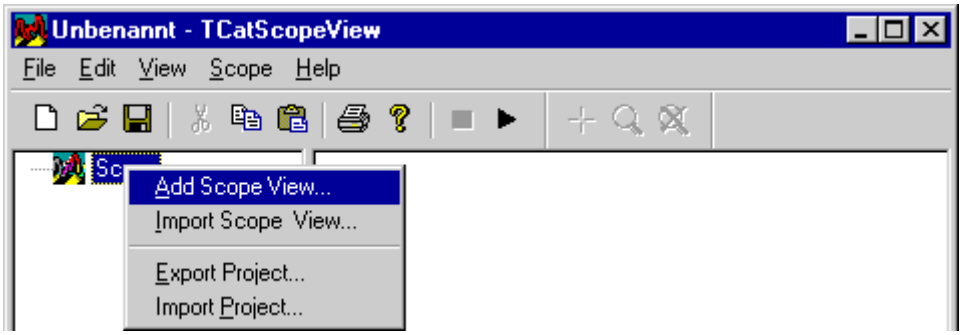


The elements of TwinCAT Scope View:

The window of TwinCAT Scope View is similar to that of TwinCAT PLC Control. In the first line is the name of the project, among them is the commend line and the toolbar. The three large windows are empty. In the left window you can configure the Scope.

Start TwinCAT Scope View:

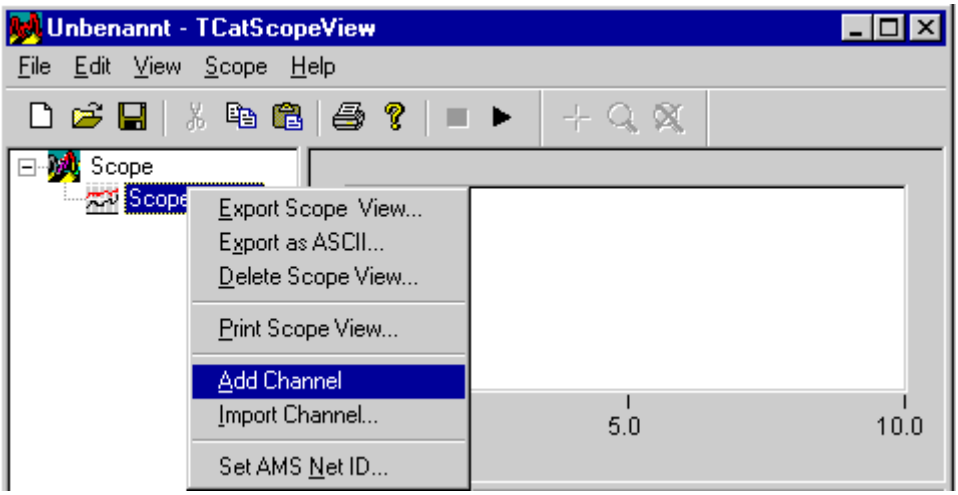
You must first add a Scope View, or in other words a project, in order to be able to start the example program Maschine.pro. To do this, click with the right mouse button on Scope, choose 'Add Scope View' there, and confirm with 'OK'.



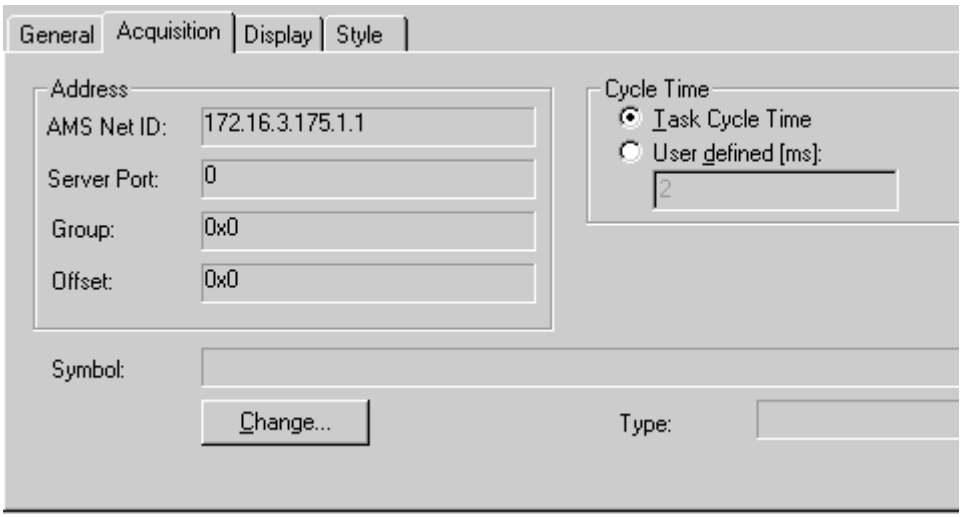
Insert channel:

In order to display the individual signals it is necessary to create the associated channels.

To do this, click with the right mouse button on 'Scope View 1', choose 'Add channel', and confirm with 'OK'.



The following tabs then appear, with which the variables whose values are to be recorded are described.



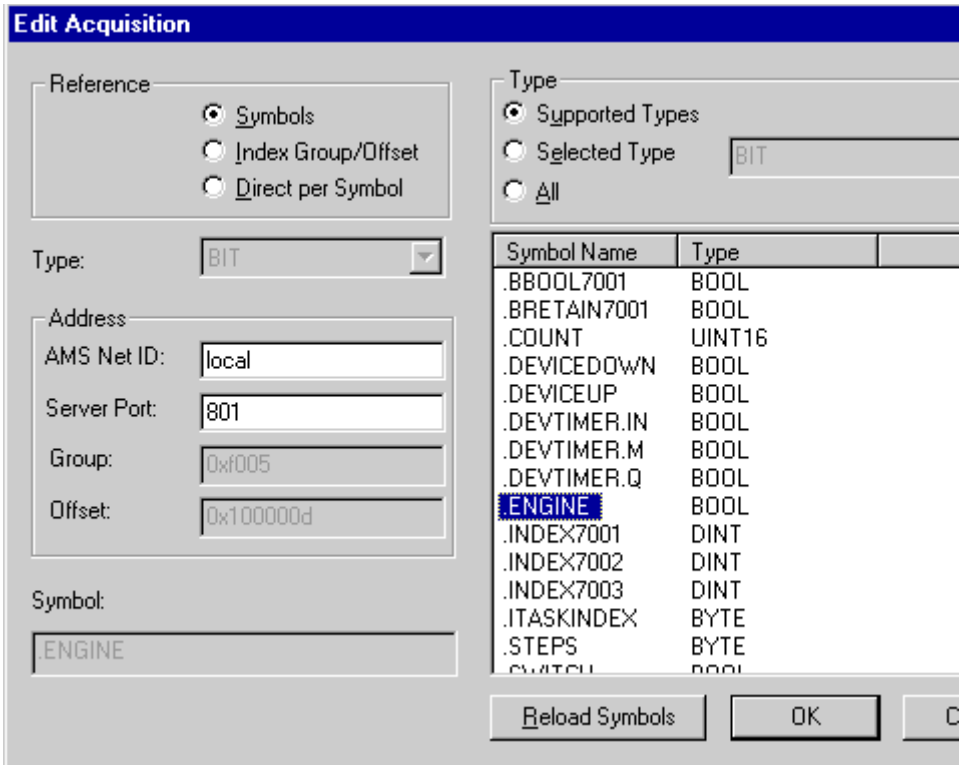
A clear display is obtained by setting the user-defined sampling time to 10 ms.

Set up server port:

The next step is to set up the server port. Under Acquisition click on 'Change', enter the number, and confirm with 'OK'. (The server port can be found from PLC Control, see under Selecting the target system [▶ 30].)

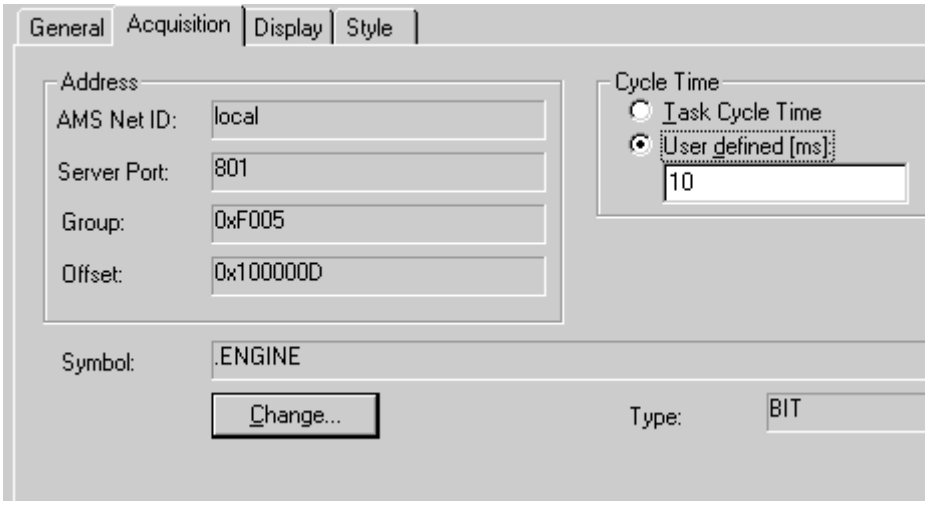
Assign channel:

Assign the signal .ENGINE to Channel 1 by clicking on 'Change' again, and confirm this action with 'OK'.



Settings

The tab then shows the following settings:



Renaming a channel:

A slow double click on 'Channel1' allows it to be renamed to ENGINE.

Add more channels:

You can assign the other channels in the same way.

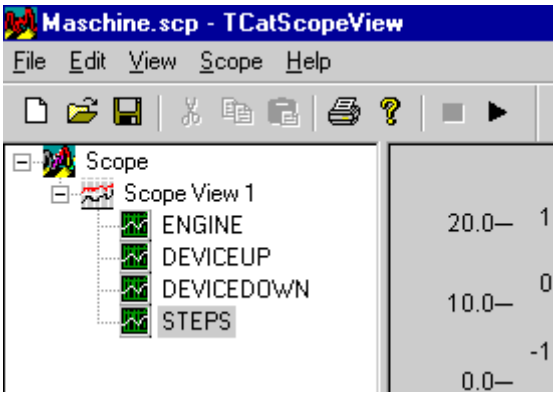
Channel2=.DEVICEUP

Channel3=.DEVICEDOWN

Channel4=.STEPS

The server port and sampling time remain the same for all the channels.

When you have created and renamed the four channels, save the Scope View via the 'File' menu, selecting 'Save as' and giving Maschine.scp as the name.



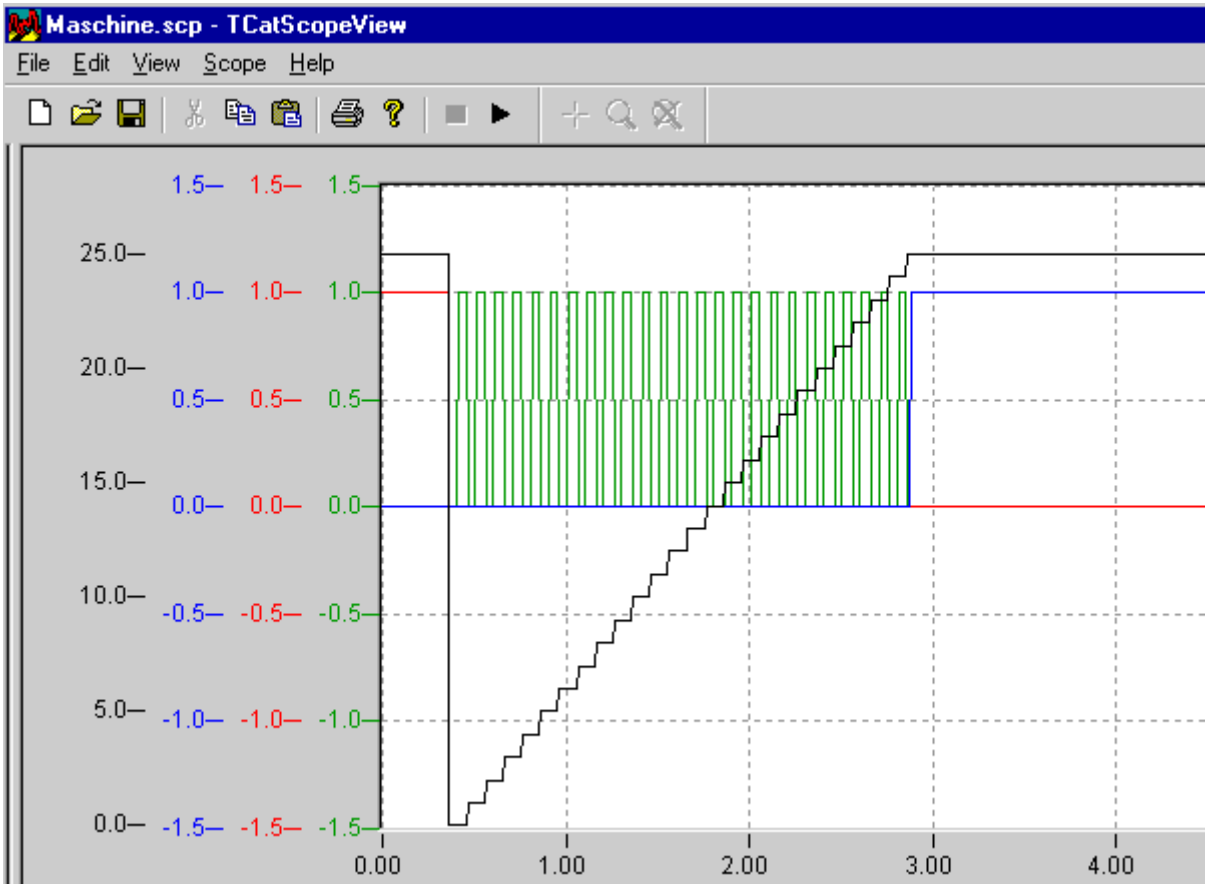
In order to be able to distinguish the various curves from one another, each channel can be given a different colour, form or axis. This is done with the aid of the Style or Display tabs.

Start recording:

Recording is started by selecting 'Start Scope' from the 'Scope' menu.

Maschine.pro

The sample program then looks like this:



7.3 Conversion of the sample program

7.3.1 Declaration of the variables

In this chapter, you will connect the previous PLC program to the Beckhoff bus terminals. This happens with the TwinCAT System Manager. By means of the System Manager, all input/output interface connections are managed and addressed and are assigned to the I/O data. Each I/O channel can be addressed by a logical name. The TwinCAT System Manager manages several field buses on one common process image

Hardare required

It´s neccessary to have the required hardware. The Demokit for the Lightbus contains the PC interface card for the II/O-Lightbus (FC2001), the bus Coupler BK 2000, Bus Terminals, 2 fibre optic and some documentation.

Without hardware go to the next chapter: Application Samples .

Variable declaration:

The storage locations (addresses) of variables are managed internally by the system. The programmer does not need to bother about memory management. The PLC programs operate with symbolic variable names, thus preventing the occurrence of side effects (overlaps) when using variables. To access the input/output level, it is necessary for the programmer to be able to assign a fixed address to individual variables. This is achieved by means of the key word 'AT', which must always be specified when declaring a variable. The key word 'AT' is followed by several parameters which provide information about the data location (input/output or flag area) and the width of the data (BIT, BYTE, WORD or DWORD). The variable declaration for the above example has the following structure:

```
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
```

Where:

	Data location	Datawidth	Meaning
%			Start of I/O definition
	I		Input
	Q		Output
	M		Flag
		X	Bit (1bit)
		B	Byte (8 bits)
		W	Word (16 bits)
		D	Double word (32 Bit)

The digits after the data width specify the address of the variable. For bit variables, the address must be specified in the format x.y, or simply x for byte, word and double word. The input and outputsare in different Storage areas , therefor may contain the same address.

7.3.2 Lightbus - Setting up the Bus Terminals

Hardware Requirements:

- PC Interface card: FC2001

- Bus coupler: BK2000

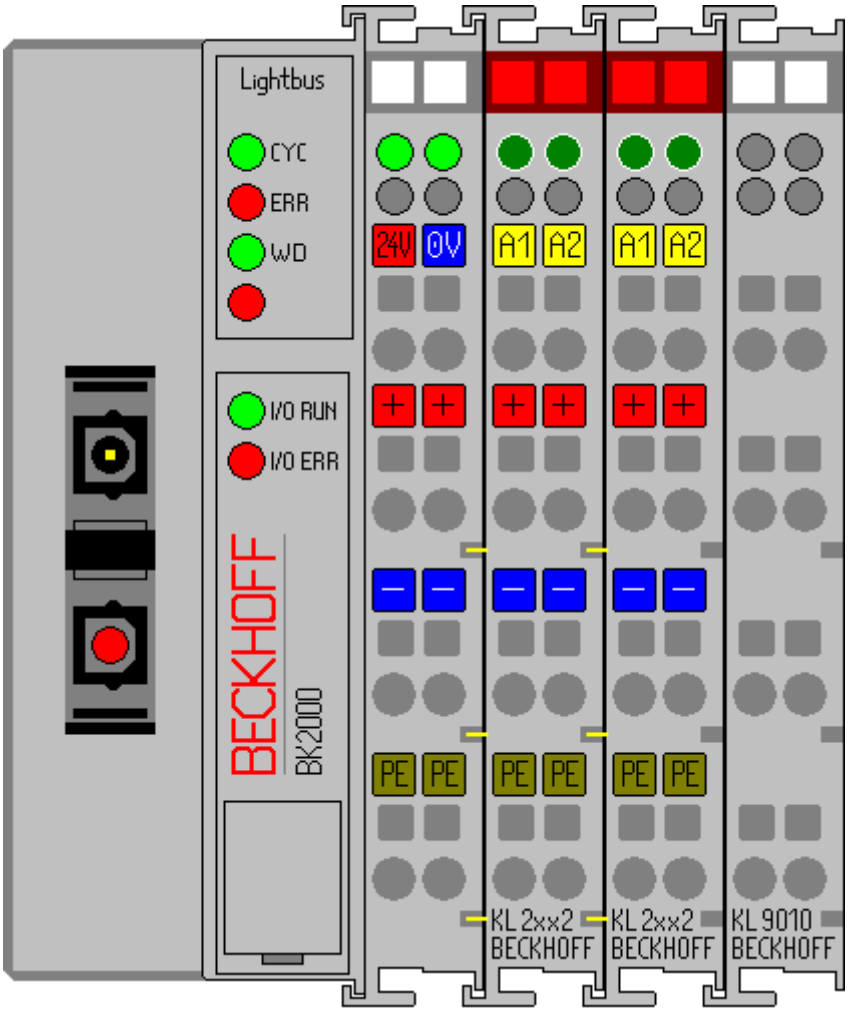
- 3 digital outputs, 24V: 2x KL2032

- Bus end terminal: KL9010

This example is exactly appropriate for the contents of the Beckhoff Lightbus demokit (TC9910-B200-0100). The hardware is however exchangeable. In this case, the configuration of the I/O devices has to be changed

Setting up the terminals

Set up the bus coupler and the bus terminals as shown in the drawing below:



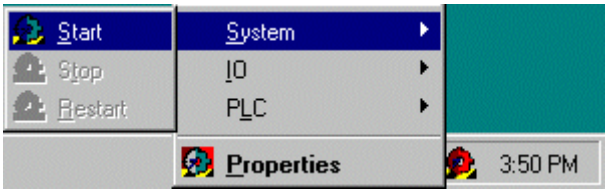
Connect the bus coupler to the PC interface card and power the bus coupler with 24 V DC.

Hardware Documentation

Further information about the hardware connection contains the hardware documentation of the demokit.

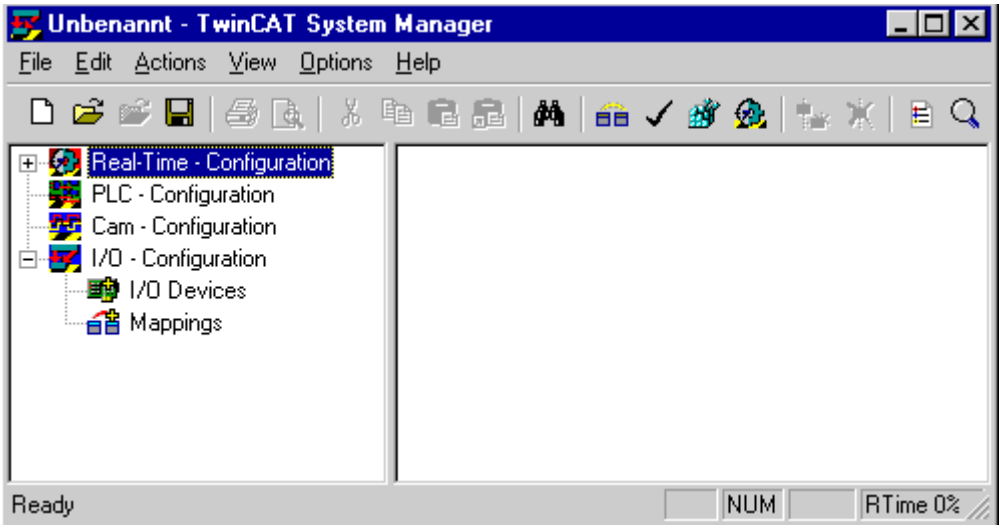
Starting the TwinCAT real time server:

Now, start the TwinCAT real time server, unless you have already done so, so that the TwinCAT message router is active.



Starting the TwinCAT System Manager

Once the system has been started, the color of the icon changes from red to green. Now start the TwinCAT System Manager by selecting 'Start'-> 'Programs' -> 'TwinCAT System' -> 'TwinCAT System Manager'.



The items of the TwinCAT System Manager:

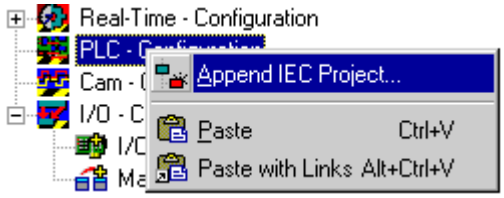
In the first line is the name of the project, (here 'unbenannt'), amount there´s the command line (menu) and the toolbar. In the last line you can see the status of the system, in this picture, the system is running (RTime). The two windows in the middle contains the configuration of the system. In the following steps you ´ll configure the system.

The system configuration is shown as a tree structure on the left side of the System Manager. It consists of the following four main points:

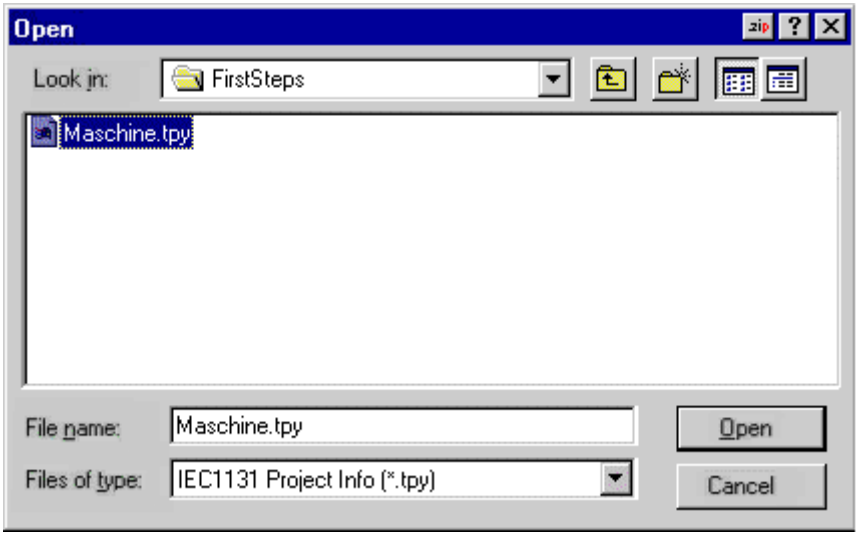
Configuration	Meaning
Realtime	Set the Real Time Parameters
PLC	All PLC Projects that are required to be configured
Cam	Add the cam server
I/O	In order to link the controller to the process level, the system needs interfaces. This entry offers a list of all interfaces.

PLC Configuration:

The individual PLC projects must be made known to the System Manager so that TwinCAT can access the variables of the PLC programs. To do this, press the right mouse button while the mouse pointer is over 'PLC configuration'

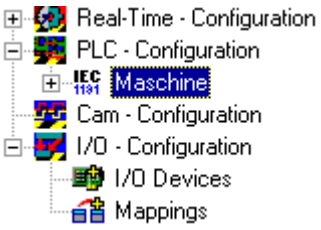


A context menu opens, in which you must select the 'Append IEC project…' entry.

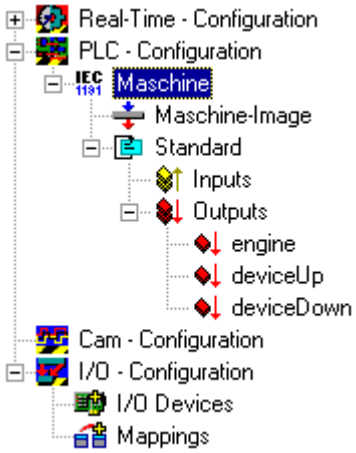


Switch to the '\TwinCAT\Samples\FirstSteps\' directory and select the 'maschine.tpy' file.

A further point has been added under 'PLC configuration' which bears the name of the PLC project.

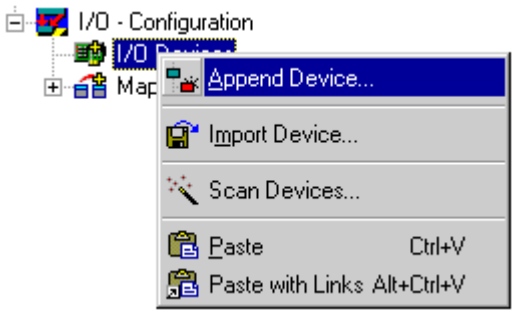


The + and the - symbols indicate whether the entry contains a further subpoints. By clicking these symbols, you open or close the entries below them. The following structure appears if you open the tree as far as possible:



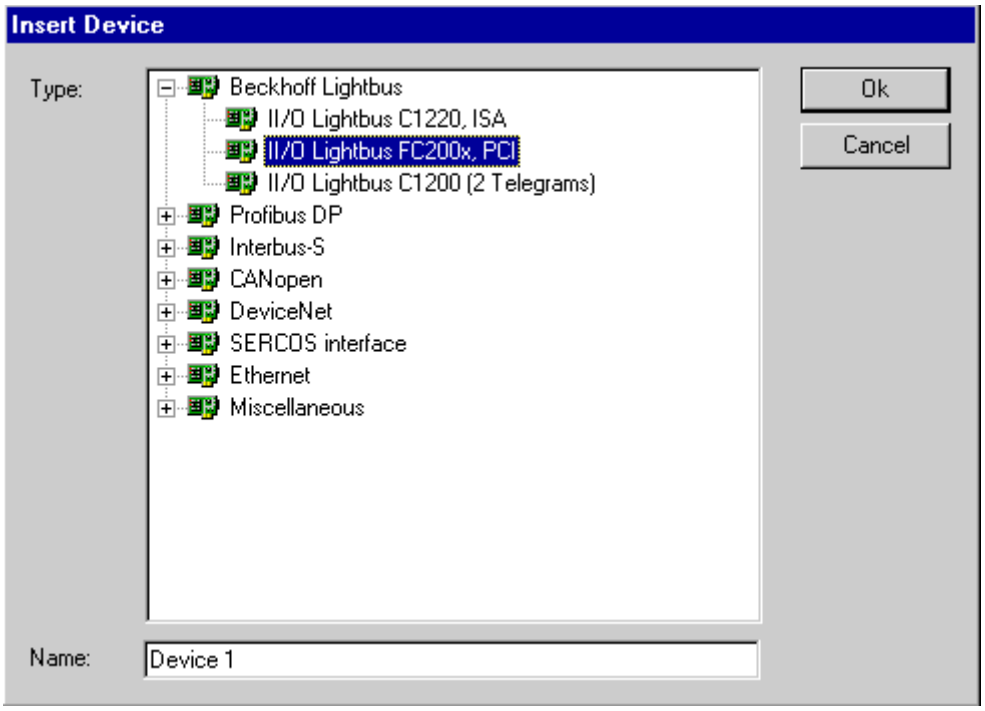
I/O Configuration:

Once the PLC project has been added to the PLC configuration, and thus all variables that are in the process image are known, it is necessary to specify the I/O configuration. Select the 'I/O devices' entry with the right mouse button. A context menu opens, in which you must select the 'Add device' entry.



Selecting an I/O device:

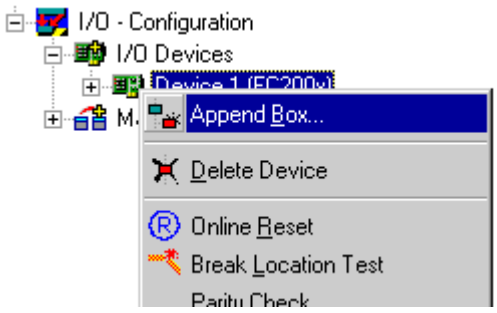
The following window opens. Select the device type, in this case 'I/O lightbus FC200x, PCI '. You can choose the device name freely.



On the right hand side, a dialog box now opens in which you can specify the configuration of the interface card. One important setting under the 'FC2001' slider, for example, is the I/O address of the lightbus card. You can use the specified entries if you have not made any changes to the default settings of the card.

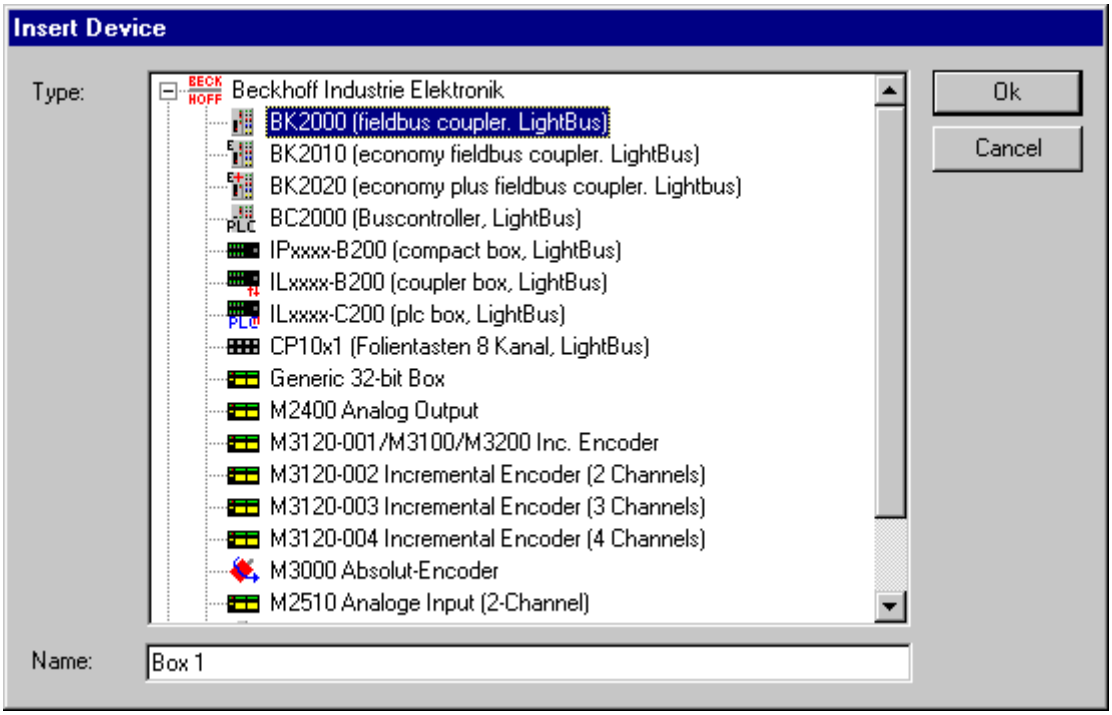
Adding a bus coupler:

Open the context menu of the FC2001 card (device 1) and select the ‘Add box…’ command.



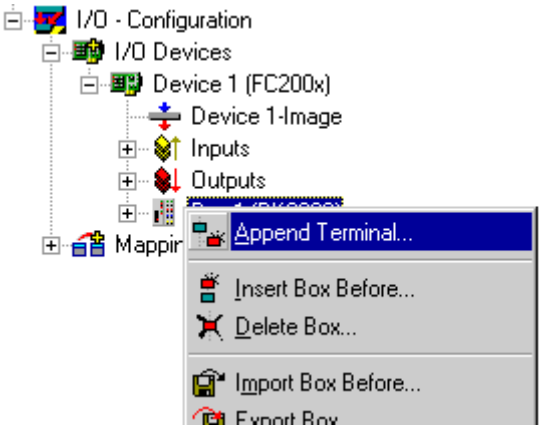
Selecting a bus coupler:

Select the field bus module type, in this case 'BK2000'. You can choose the field bus module name freely.



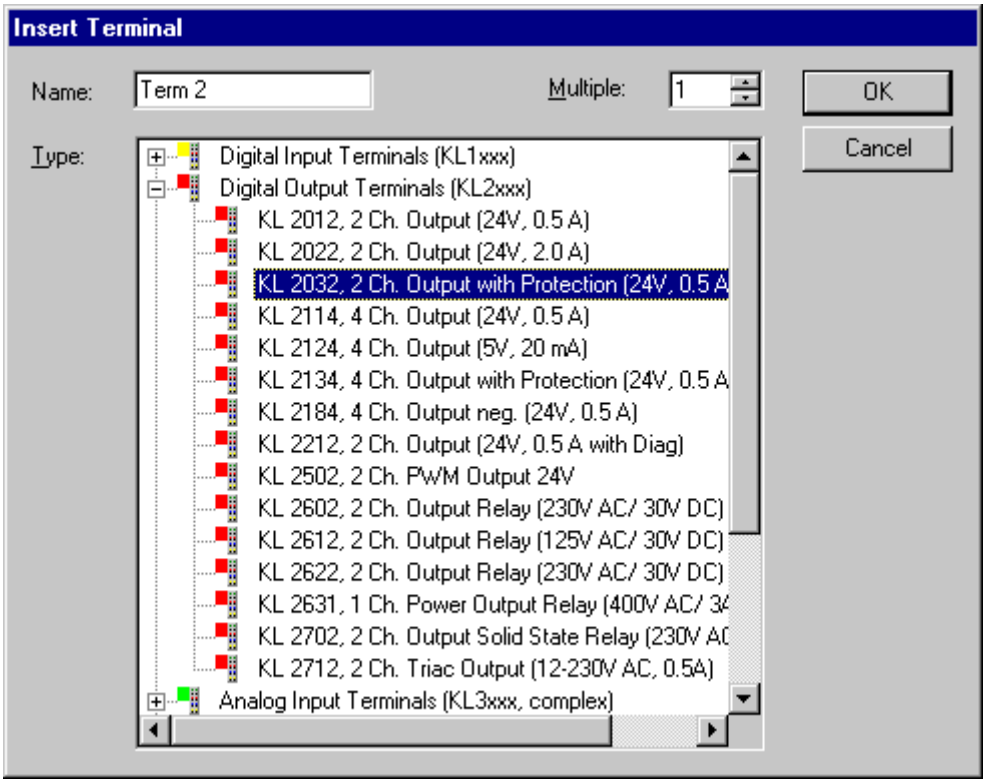
Adding a bus terminal:

Open the context menu (press the right mouse button) of the BK2000 (box 1) and select the 'Add terminal…' command.



Selecting a bus terminal:

Select the 2032 terminal. To do this, move the mouse pointer to the - sign and click to close the selection of digital input terminals. Click on the '+' sign for the digital output terminals to open the new selection and click on the desired KL2032 terminal to select it. Confirm your selection with 'OK'.



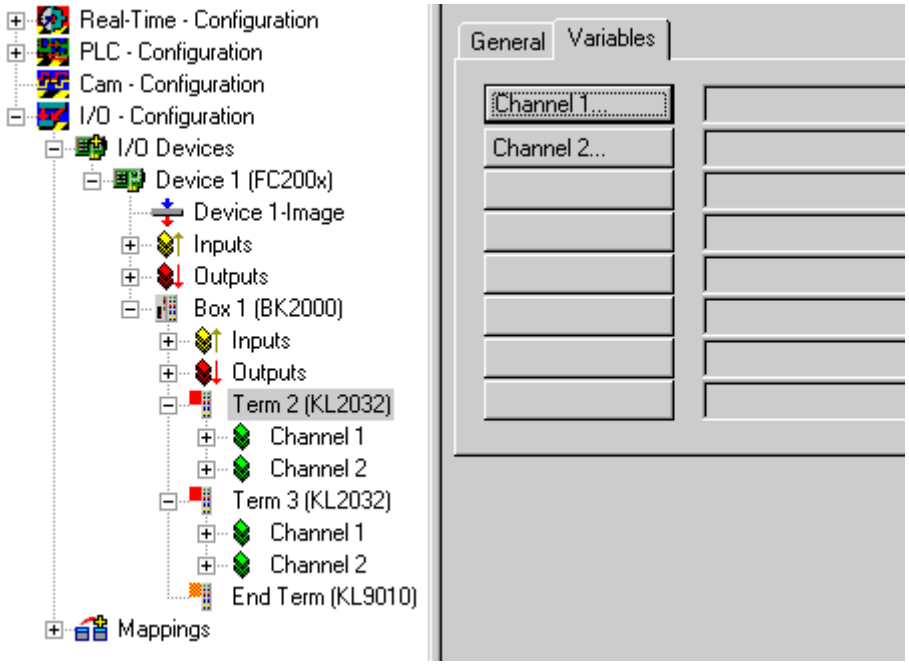
You can choose the bus terminal name freely.

The sample program requires 3 digital outputs. To insert the second bus terminal, repeat the steps.

The Bus end Terminal KL9010 is inserted automatically by the System Manager.

End of configuration:

The configuration then has the following breakdown:



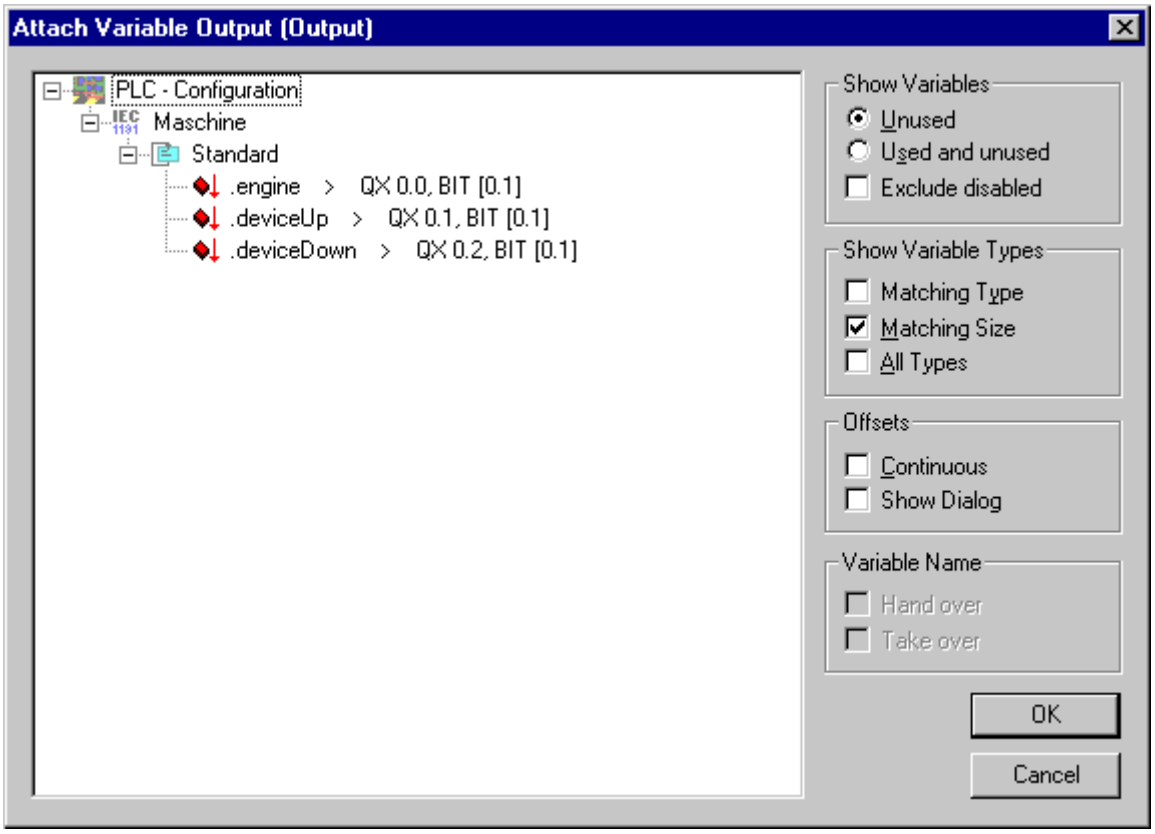
It goes without saying that you can rename the standard designations (device 1, box 1, terminal 1 etc.) To do this, slowly double click the corresponding name and enter the new designation.

Assigning variables to the input/output channels:

Up to this point, the complete hardware needed for the above example program has been configured. Next, the individual variables from the PLC project must be assigned to the individual input/output channels.

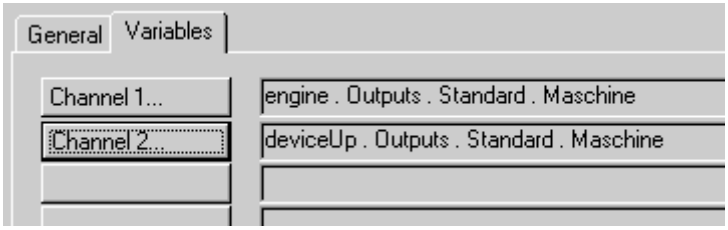
To do this, mark the terminal you wish to configure. In the case of terminal 1 (four digital outputs), a dialog box containing the 'General' and 'Variables' sliders opens on the right. Select the 'Variables' tab.

Obove you see a list of the two output channels, but they are still all free. To configure channel 1, select the corresponding button ('channel 1…'). The following dialog box opens:



All output variables are now listed in this dialog box. Select the first variable (engine) and confirm your entry by clicking 'OK'. Proceed analogously with the second output variable.

The first bus terminal is attached:



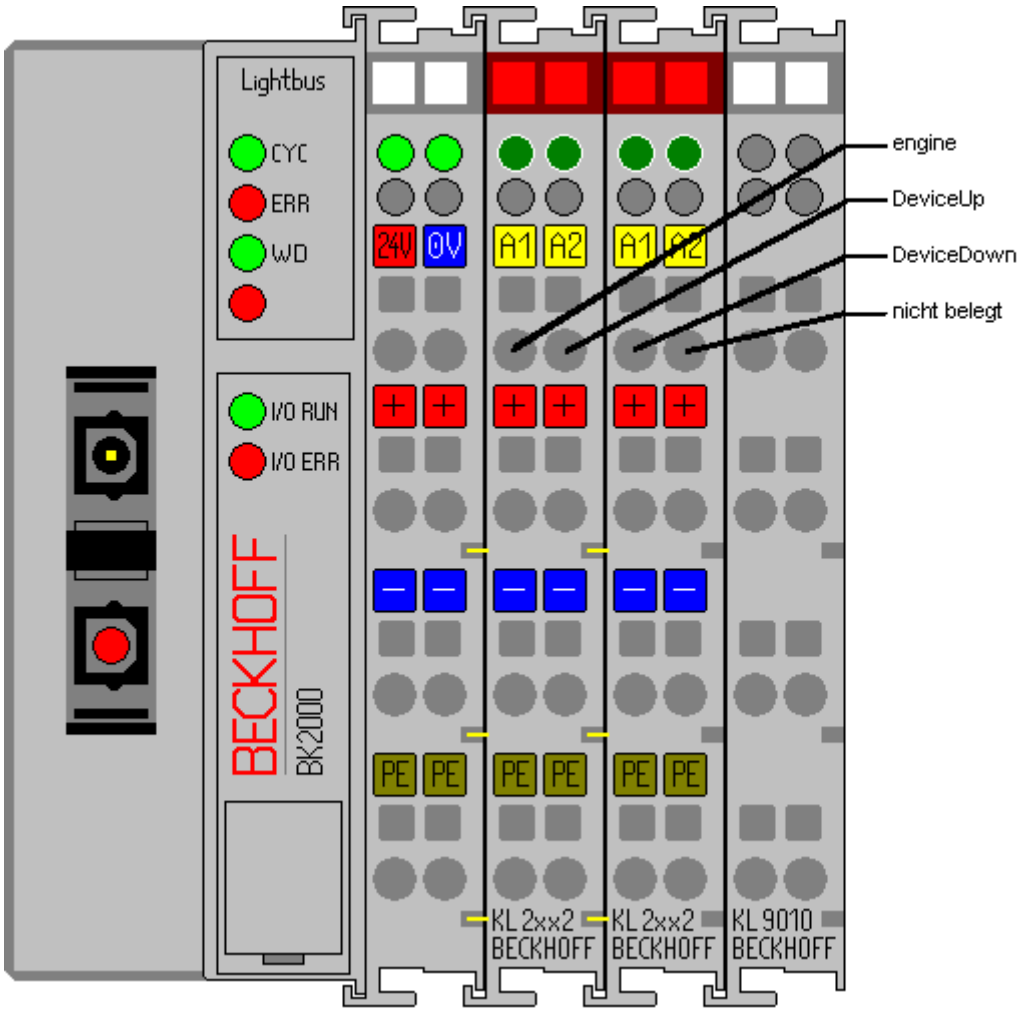
The second bus terminal is also assigned, channel 1 is assigned to device.down. Channel 2 of this terminal remains unassigned.

Assigning variables

Terminal 2	PLC variables	Meaning
Channel1(=output 1)	engine	Stepper motor control
Channel2 (=output 2)	device.Up	Drill up control

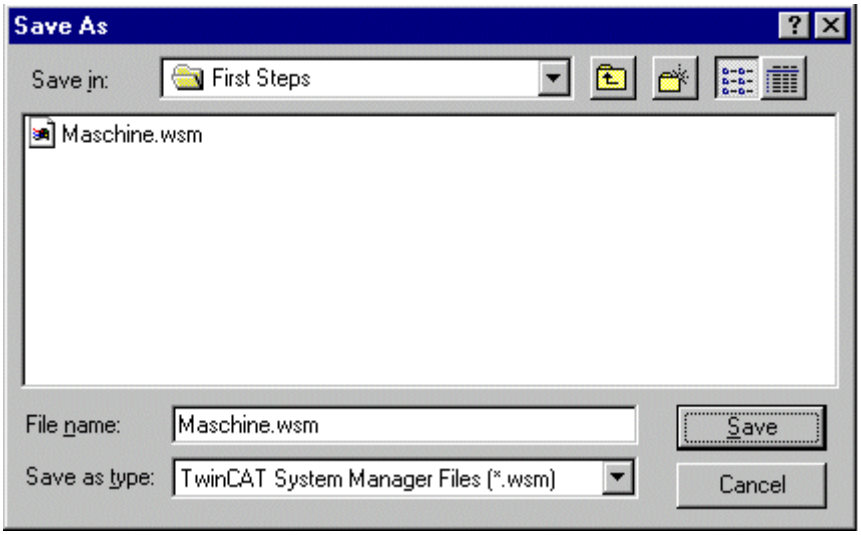
Terminal 3	PLC variables	Meaning
Channel1 (=output 3)	device.Down	Drill down control
Channel2 (=output 4)	-	free channel

Assignments of the bus terminals:



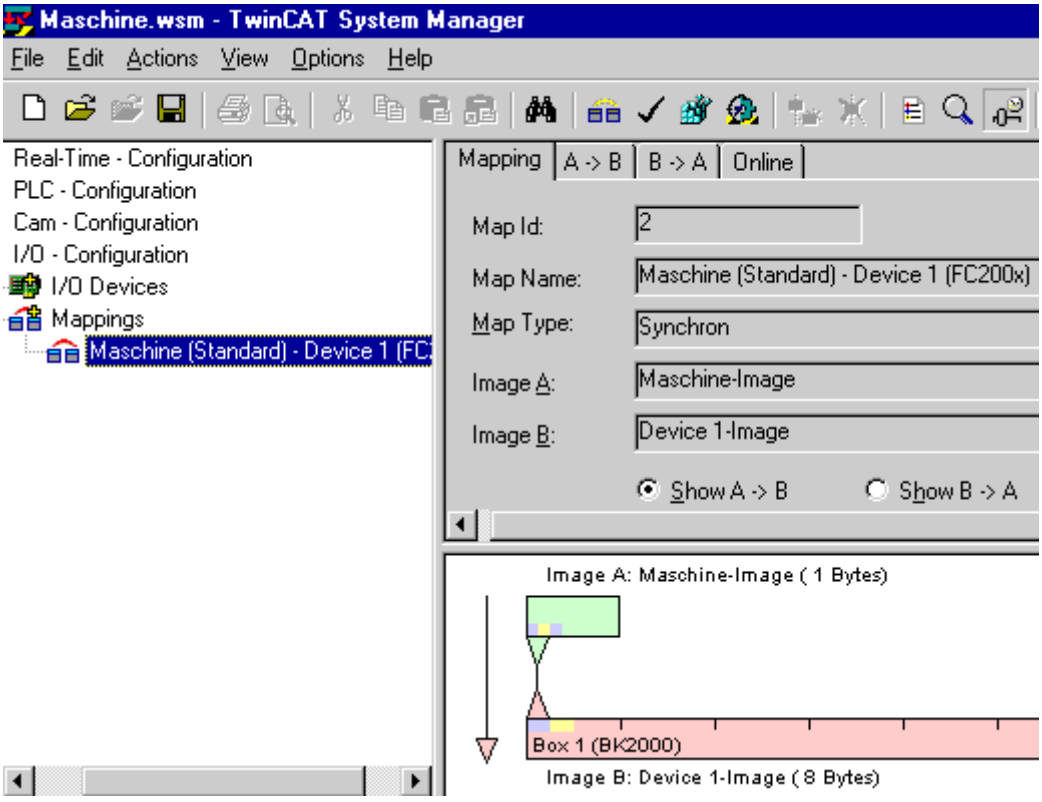
Saving the project:

You should save the configuration at this point to make sure you can access it later on. To do this, run the 'Save as…' command from the 'File' menu.



Mapping Variables:

You have now configured the complete system for the above example pro-gram. You must now create the allocation. To do this, go to the 'Create allocation' command in the 'Actions' menu. Under the 'Allocations' tree entry, you now see 'Standard device 1(FC2001)'. Click this entry. The following window opens:



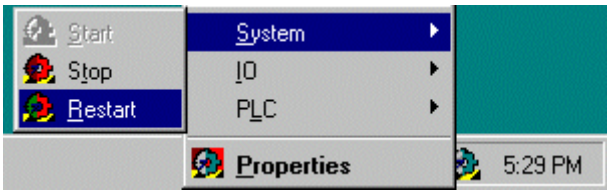
In the dialog box, you can define whether the data flow from A to B or from B to A is to be displayed. In this case, Image A corresponds to the process image of the PLC variables, i.e. the input/output variables. Image B corresponds to the process image of the I/O devices, in this case of the bus coupler BK2000. Each variable or bus terminal is color highlighted in the process image. If you stop on one of these areas with the mouse, a small display box appears in which the precise designation is shown.

Writing the configuration to the registry:

As the last step, you must save the configuration to Windows NT registry because the information stored there is evaluated when you start Twin-CAT. Run the 'Save in registry…' command from the 'Actions' menu. If an older configuration is already stored there, a safety prompt will appear, which you must confirm.

Restarting TwinCAT:

You must restart the system so that TwinCAT will accept the change.



The individual PLC variables are now output on the bus terminal KL 2032. The Bus Terminals indicate their signal state by means of light emitting diodes.

7.3.3 Ethernet - Setting up the bus terminals

Hardware Requirements:

- Ethernet - Bus coupler (BK9000)

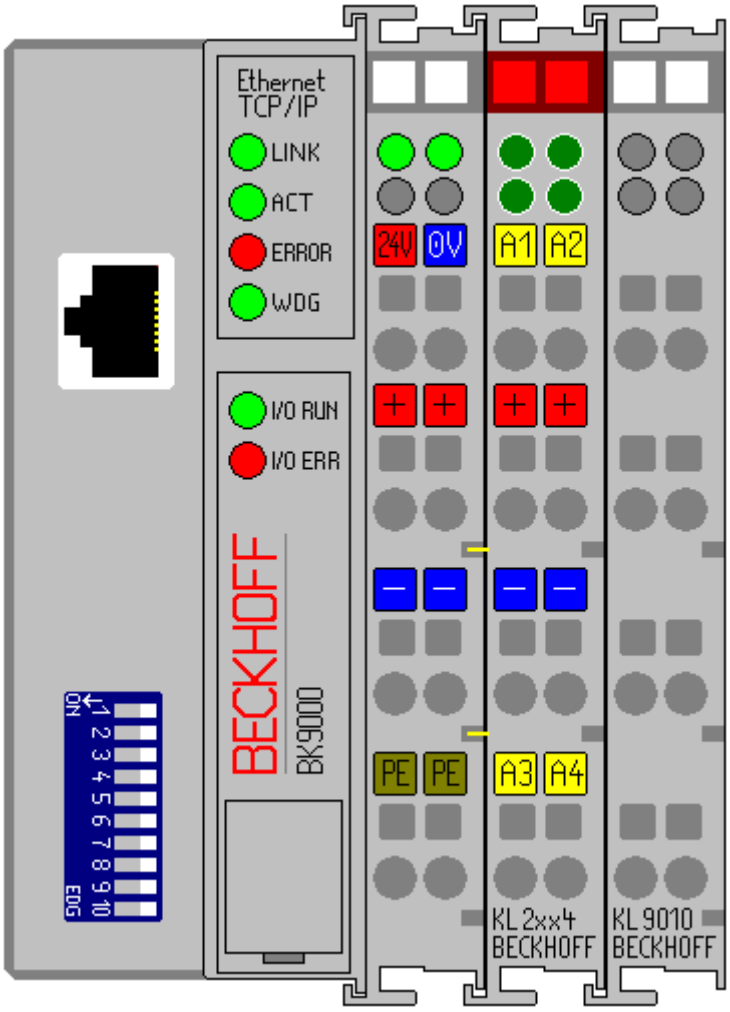
- 1 bus terminal with 4 digital outputs (KL2114)

- Bus end terminal (KL9010)

This example is exactly appropriate for the contents of this Ethernet bus station. The hardware is however exchangeable. In this case, the configuration of the I/O devices has to be changed.

Setting up the bus coupler and the terminals

Set up the bus coupler and the bus terminals as shown in the drawing below:



Hardware Documentation

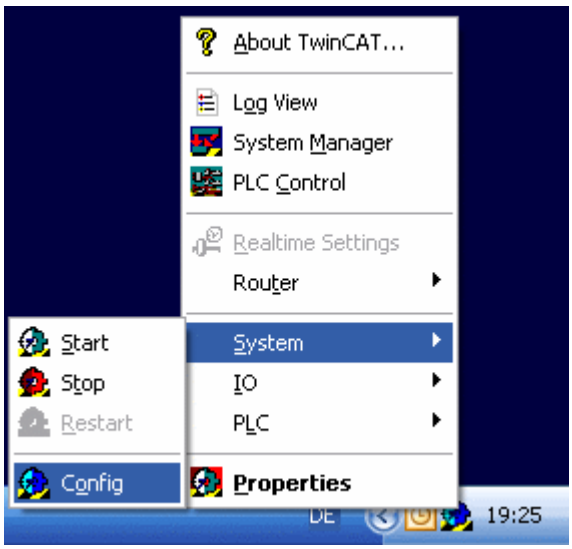
Further information about the hardware connection contains the hardware documentation of the bus coupler.

TwinCAT PLC Configuration

It must the extended version of the 'Machine'-project on the first run time the local PLC arranged its.
'Machine_Final' corresponds to the changes in this chapter [▶ 26] was made.
Download PLC samples: https://infosys.beckhoff.com/content/1033/tcquickstart/Resources/5281688587/.zip

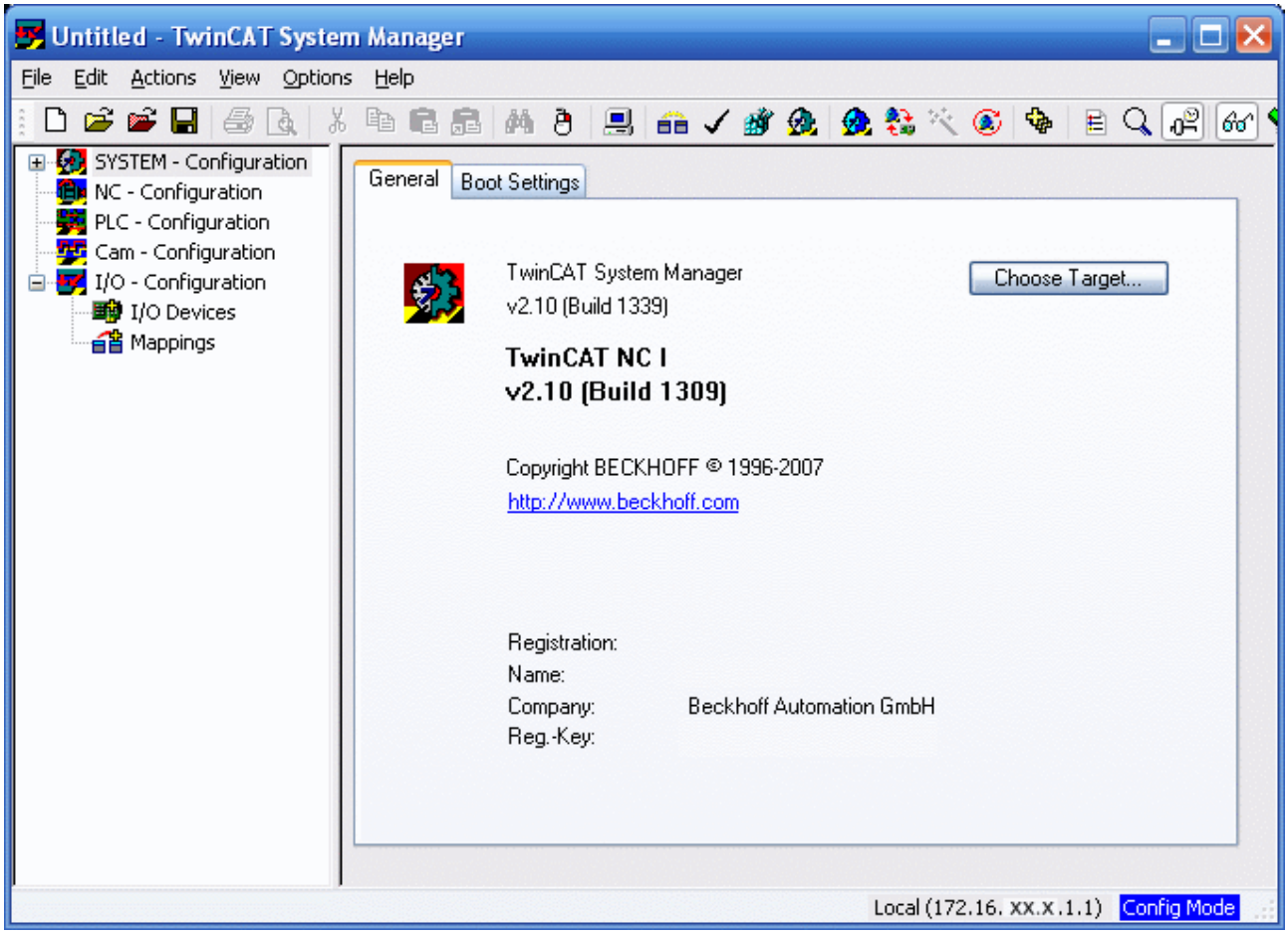
Start the TwinCAT in Config-Mode

Click in the taskbar right down on the symbol of TwinCAT, then goto System/Config. This TwinCAT is started in config-mode , which makes the scanning possible of the boxes.



Starting the TwinCAT System Manager

Once the system has been started, the color of the icon changes from red to blue. Now start the TwinCAT System Manager by selecting 'Start'-> 'Programs' -> 'TwinCAT System' -> 'TwinCAT System Manager'.



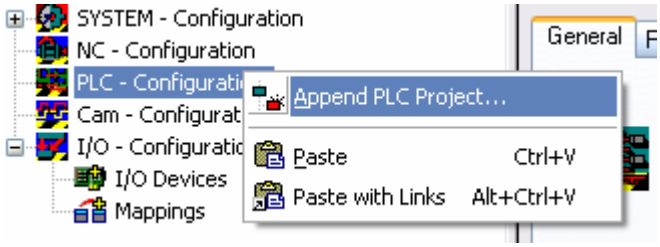
The items of the TwinCAT System Manager

In the first line is the name of the project, (here 'unbenannt'), amount there´s the command line (menu) and the toolbar. In the last line you can see the status of the system, in this picture, the system is running (RTime). The two windows in the middle contains the configuration of the system. In the following steps you ´ll configure the system.
The system configuration is shown as a tree structure on the left side of the System Manager. It consists of the following four main points:

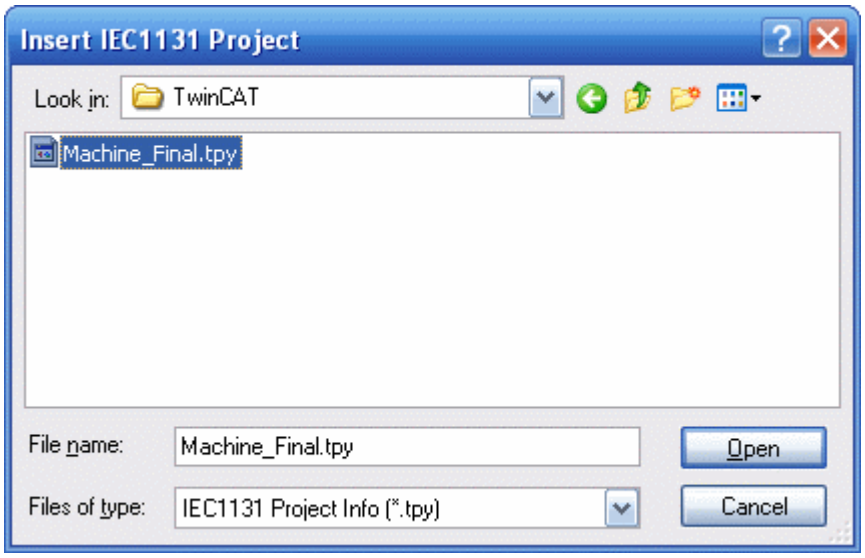
Konfiguration	Meaning
System	Configuration of the system and real time parameters
NC	(Optional) Append the NC-tasks and configure them.
PLC	All PLC Projects that are required to be configured
Cam	(Optional) Append Cam Group
I/O	In order to link the controller to the process level, the system needs interfaces. This entry offers a list of all interfaces.

PLC Configuration:

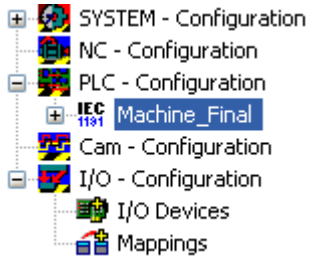
The individual PLC projects must be made known to the System Manager so that TwinCAT can access the variables of the PLC programs. To do this, press the right mouse button while the mouse pointer is over 'PLC configuration'.



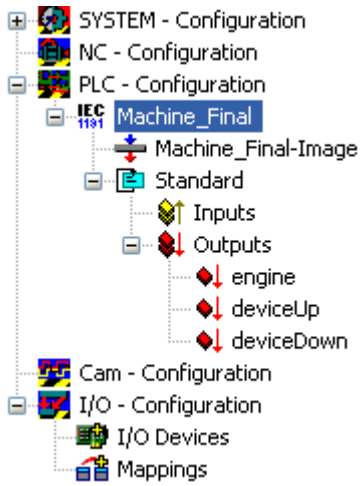
A context menu opens, in which you must select the 'Append PLC project…' entry.



Switch to the your '\TwinCAT\Samples\FirstSteps\TwinCAT' directory and select the 'Machine_Final.tpy' file. A further point has been added under 'PLC configuration' which bears the name of the PLC project.

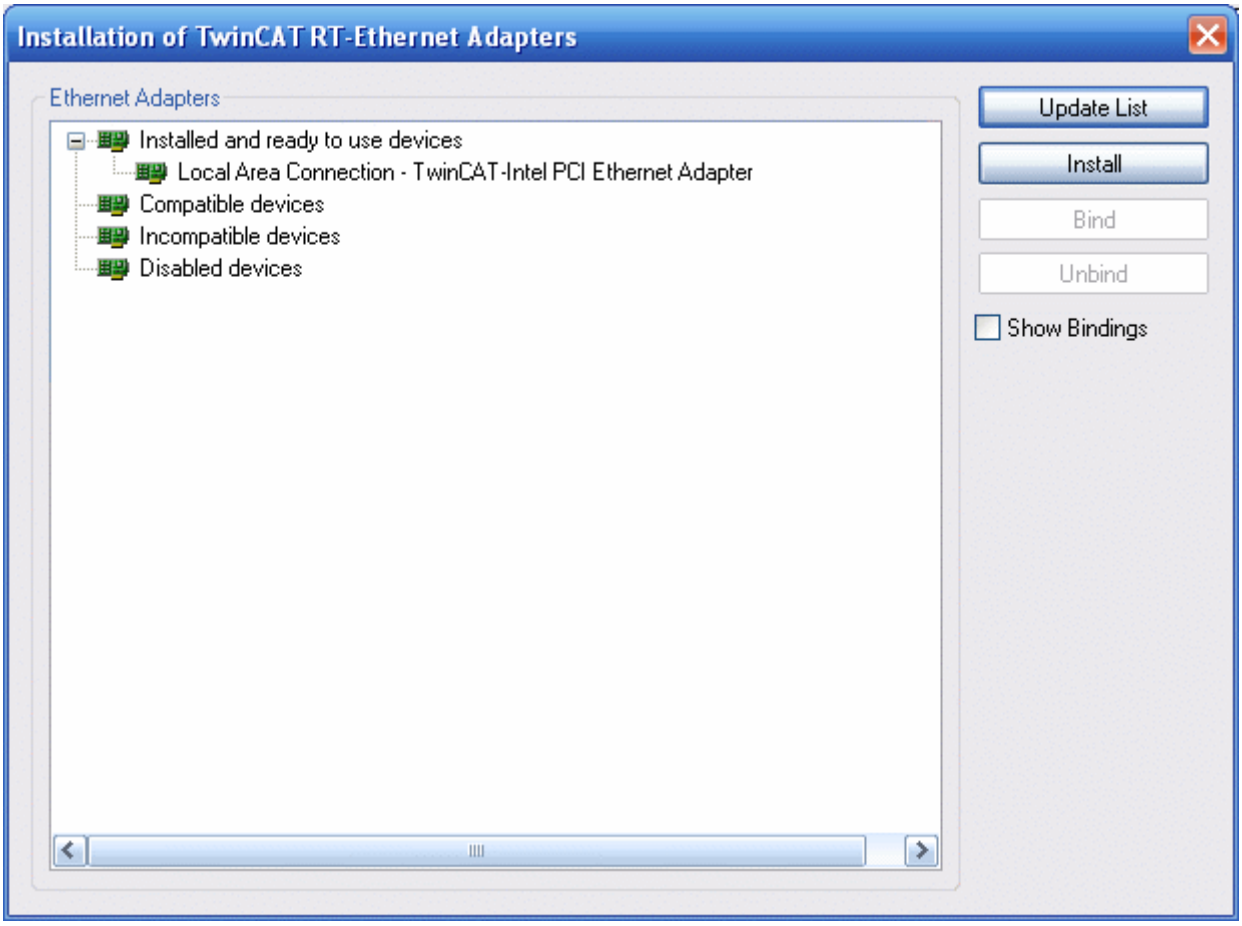


The + and the - symbols indicate whether the entry contains a further subpoints. By clicking these symbols, you open or close the entries below them. The following structure appears if you open the tree as far as possible:



Setting up real-time Ethernet

Before you can use real-time Ethernet-capable devices (BK9000), your network card must be furnished with associated drivers.
In the menu bar select 'Options/List of real-time Ethernet compatible devices...'



The following dialog has four subitems:

Installed and ready to use devices	The network connections listed here are real-time Ethernet-capable and can be used
Compatible devices	Network connections listed here are real-time Ethernet-capable in principle, although they still requires the associated drivers. Click on 'Install'. If the installation was successful the network connections will be shown under 'Installed and ready to use devices'
Incompatible devices	The network connections listed here are not real-time Ethernet-capable.
Disabled devices	Network connections listed here are deactivated and are currently not available.

To continue, at least one network connection must be available under 'Installed and ready to use devices'.

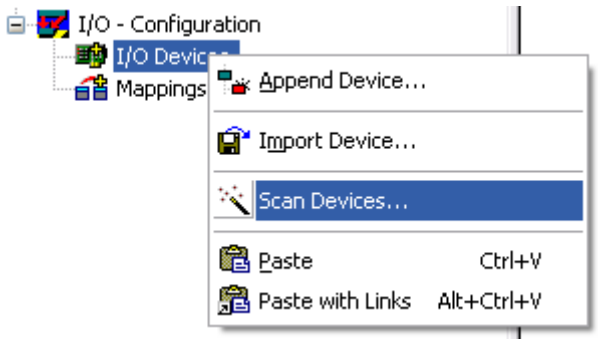
I/O Configuration

Add a device

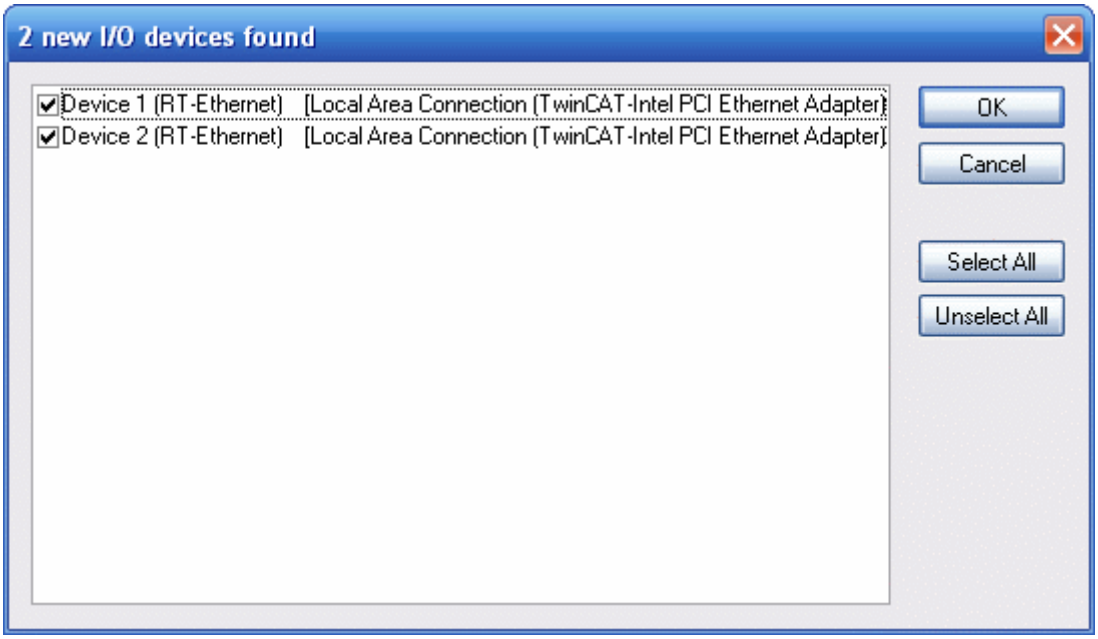
Insert the I/O devices in the same way like the PLC configuration.

With TwinCAT 2.9 the devices can be scanned automatically ('Scan devices'). Found devices are listed afterwards under I/O devices in the tree view.
The the target system has to be in Config Mode for this function.

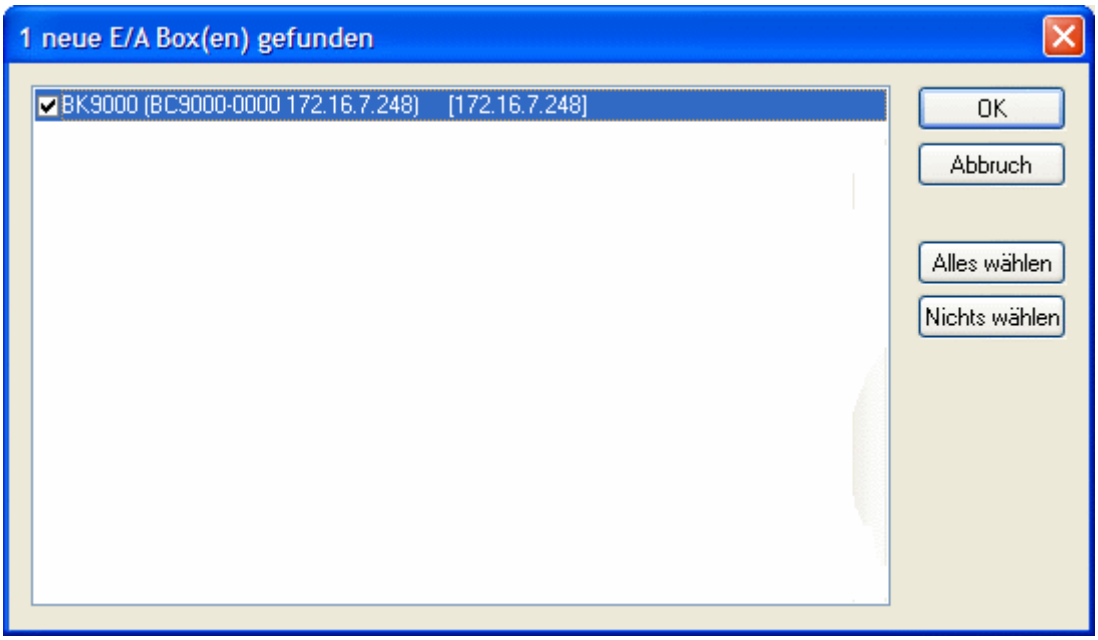
Select the 'I/O devices' entry with the right mouse button. A context menu opens, in which you must select the 'Scan devices' entry.



The following window containing a list of all found devices opens. Select the "RT-Ethernet" device. If several devices of this type are listed, select the device that accords to the network connection of the BK9000.



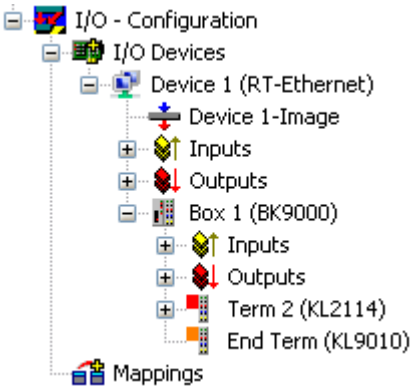
TwinCAT scans the connected peripheral and lists the found devices ->Confirm with 'OK'.



Select your BK9000 and confirm with 'OK'. Don't activate the 'Free Run'.

Expand the tree view by clicking on the '+' symbol to the left of it. As can be seen, the terminals connected to the Bus Coupler have been automatically detected and added.

Now, the System Manager shows the found I/O configuration:



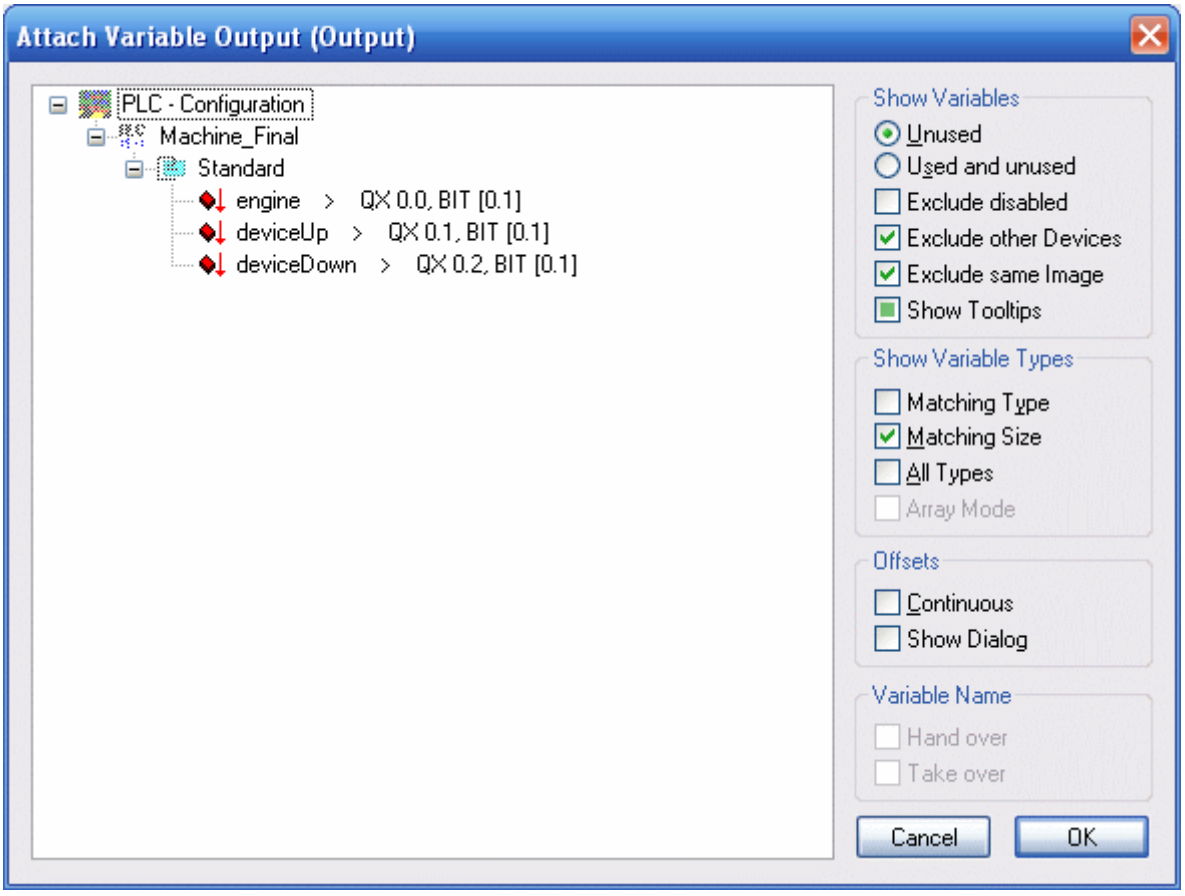
It goes without saying that you can rename the standard designations (device 1, box 1, terminal 1 etc.) To do this, slowly double click the corresponding name and enter the new designation.

Assigning variables to the input/output channels:

Up to this point, the complete hardware needed for the above example program has been configured. Next, the individual variables from the PLC project must be assigned to the individual input/output channels.

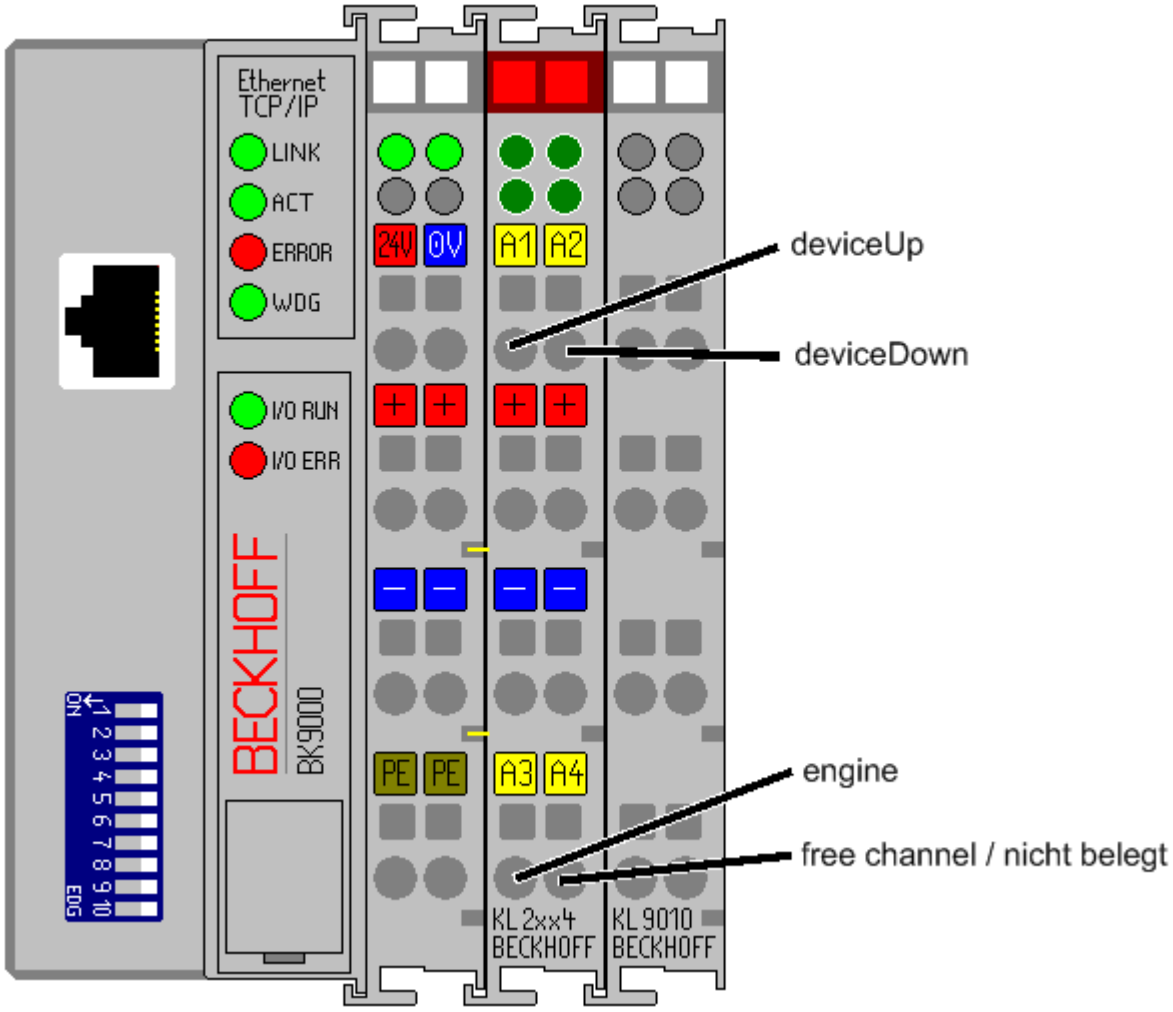
To do this, mark the terminal you wish to configure. A dialog box containing the input and output channels of the connected peripheral opens. Select the 'Variables' tab.

You see a list of the four output channels, but they are still all free. Select for all variables of the wanted digital output channel.



Terminal 2	PLC variables	Meaning
Terminal 1 (=output 1)	DeviceUp	Drill up control
Terminal 2 (=output 2)	DeviceDown	Drill down control
Terminal 3(=output 3)	Engine	Stepper motor control

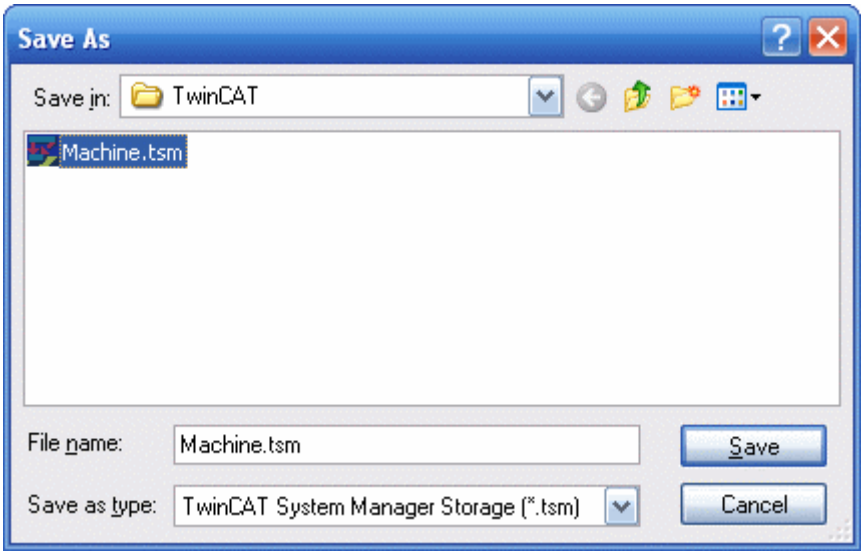
Assignments of the bus terminals:



Project activation

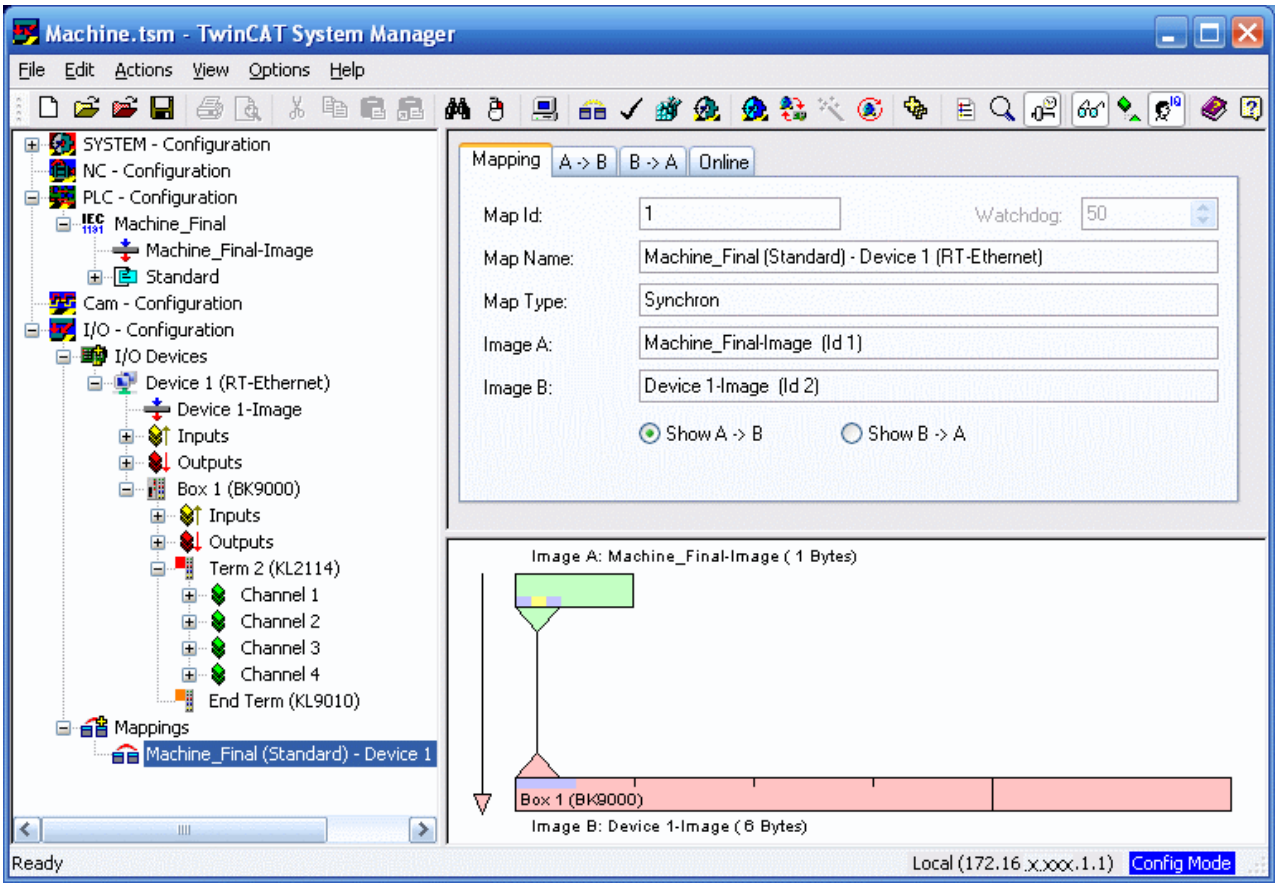
Saving the project

You should save the configuration at this point to make sure you can access it later on. To do this, run the 'Save as…' command from the 'File' menu.



Mapping variables:

You have now configured the complete system for the above example program. You must now create the allocation for the registry. To do this, go to the 'Create mapping' command in the 'Actions' menu. Under the 'Mappings' tree entry, you now see 'Maschine (Standard) - Device 1 (RT-Ethernet)'. Click this entry. The following window opens on the right side:



You can define whether the data flow from A to B or from B to A is to be displayed. In this case, Image A corresponds to the process image of the PLC variables, i.e. the input/output variables. Image B corresponds to the process image of the I/O devices, in this case of the BK9000bus coupler. Each variable or bus

terminal is color highlighted in the process image. If you stop on one of these areas with the mouse, a small display box appears in which the precise designation is shown. With a right mouse click you can zoom the picture.

Writing the configuration to the registry

As the last step, you must save the configuration to registry because the information stored there is evaluated when you start TwinCAT. Run the 'Save in registry…' command from the 'Actions' menu. If an older configuration is already stored there, a safety prompt will appear, which you must confirm.

Restarting TwinCAT:

TwinCAT wants to restart in the 'Run Mode' to take the changes. Confirm the dialog of the System Manager with 'OK'.

Maybe the PLC project Machine_Final.pro' has to be reloaded

The individual PLC variables are now output on the bus terminal KL2404. The Bus Terminals indicate their signal state by means of light emitting diodes.

Further information about the TwinCAT System Manager can be found in the Beckhoff Information System.

7.3.4 EtherCAT - Setting up the Bus Terminals

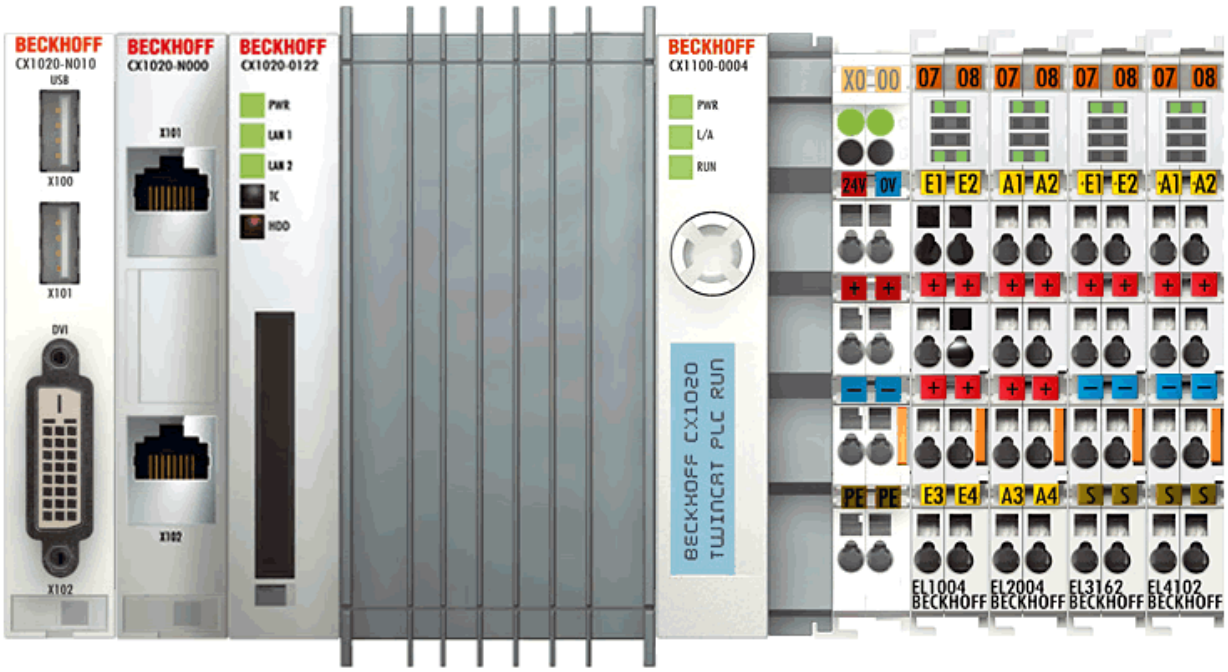
Hardware Requirements:

- embedded PC CX1020
  - CPU base module C1020-0123: 512 MB Flash memory, WindowsXP embedded
  - System interface CX1020-N010: one DVI-I and two USB interfaces.
  - Power supply unit CX1100-0004 with EtherCAT interface
- or a conventional IPC with Windows 2000, XP or Vista and at least one unused Ethernet port.
- EtherCAT I/O-Terminals:
  - EL2004: 4x Digital-OUT, 24V DC, 0.5A, typical cycle time 90 µs
  - Further I/O terminals are possible

This instruction is written for an embedded PC CX1020. But it is applicable to every other Windows PC, too.

Setting up the terminals:

Set up the CX1020 as shown in the drawing below. (The drawing shows an example configuration.)



Power the CX1020 with 24 V DC.

Hardware Documentation

Further infomration about the hardware connection can be found in the correspondent hardware documentation.

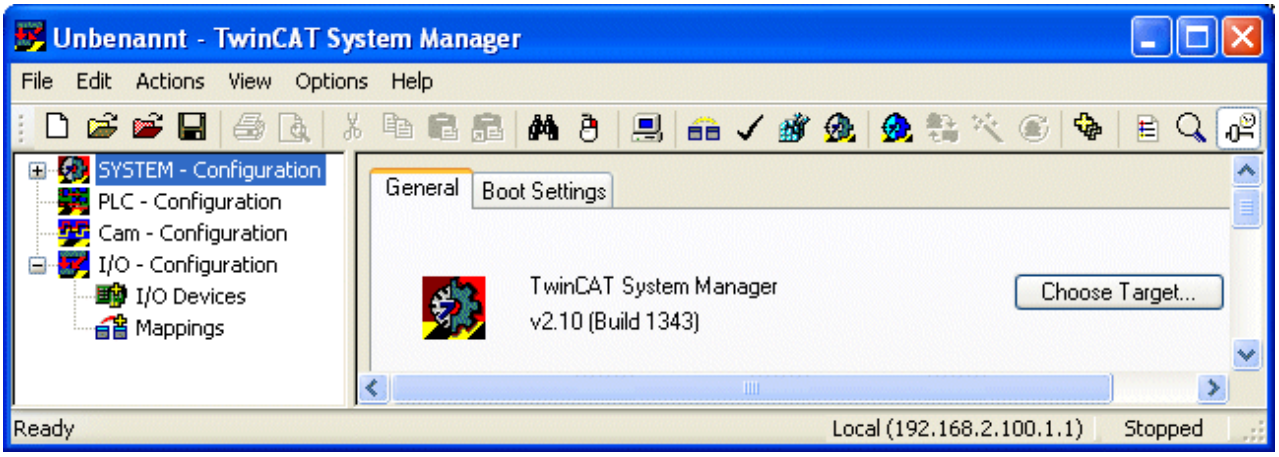
Starting the TwinCAT real time server

Now, start the TwinCAT real time server in the 'Config Mode', unless you have already done so, so that the TwinCAT message router is active.

Starting the TwinCAT System Manager

Once the system has been started, the color of the icon changes from red to blue. Now start the TwinCAT System Manager by selecting 'Start'-> 'Programs' -> 'TwinCAT System' -> 'TwinCAT System Manager'.

The items of the TwinCAT System Manager:



In the first line is the name of the project, (here 'unbenannt'), amount there´s the command line (menu) and the toolbar. In the last line you can see the status of the system, in this picture, the system is running (RTime). The two windows in the middle contains the configuration of the system. In the following steps you ´ll configure the system.

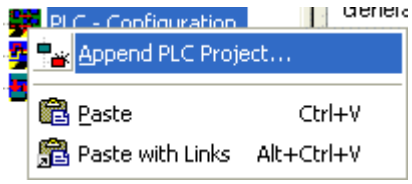
The system configuration is shown as a tree structure on the left side of the System Manager. It consists of the following three main points:

Configuration	Meaning
System configuration	Set the system and the Real Time Parameters
PLC configuration	All PLC Projects that are required to be configured
I/O configuration	In order to link the controller to the process level, the system needs interfaces. This entry offers a list of all interfaces.

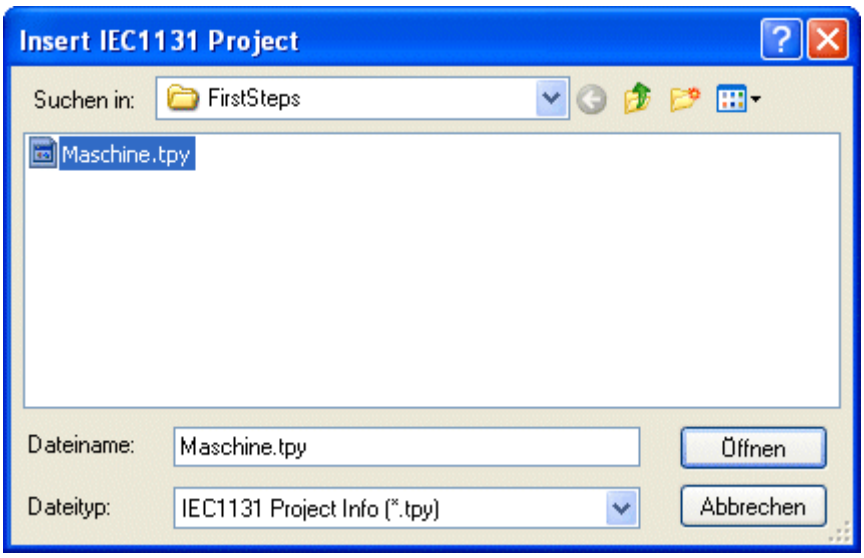
Further System components are possible listed (e.g. Cam server or NC configuration).

PLC Configuration:

The individual PLC projects must be made known to the System Manager so that TwinCAT can access the variables of the PLC programs. To do this, press the right mouse button while the mouse pointer is over 'PLC configuration'

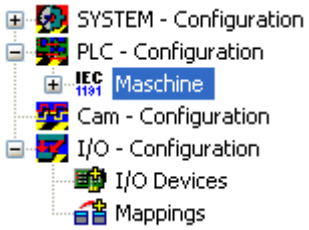


A context menu opens, in which you must select the 'Append IEC project…' entry.

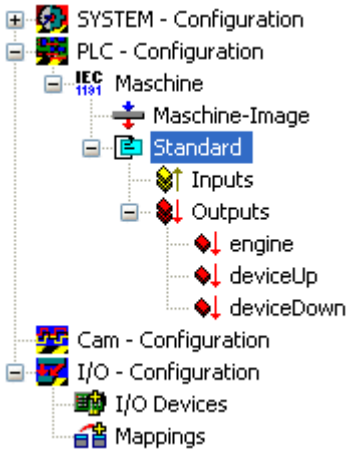


Switch to the '\TwinCAT\Samples\FirstSteps\' directory and select the 'maschine.tpy' file.

A further point has been added under 'PLC configuration' which bears the name of the PLC project.



The + and the - symbols indicate whether the entry contains a further subpoints. By clicking these symbols, you open or close the entries below them. The following structure appears if you open the tree as far as possible:



In the picture above you can see that your example program contains a task named 'standard'.

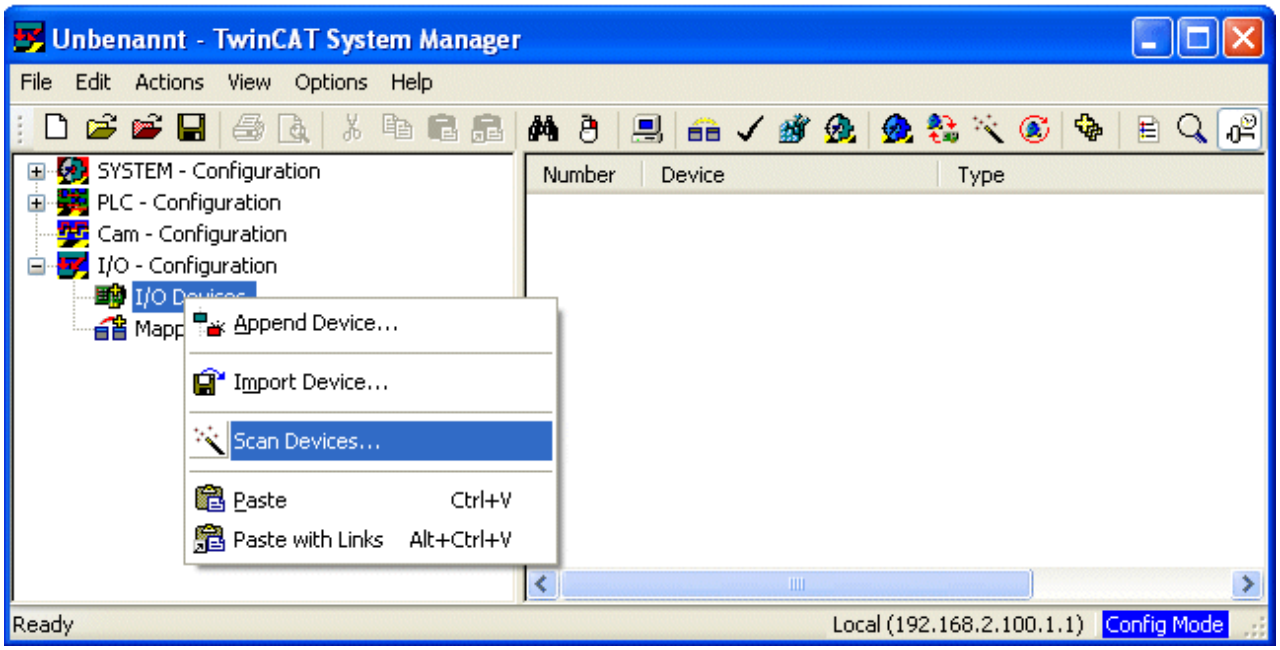
I/O Configuration

Add a device

Insert the I/O devices in the same way like the PLC configuration.

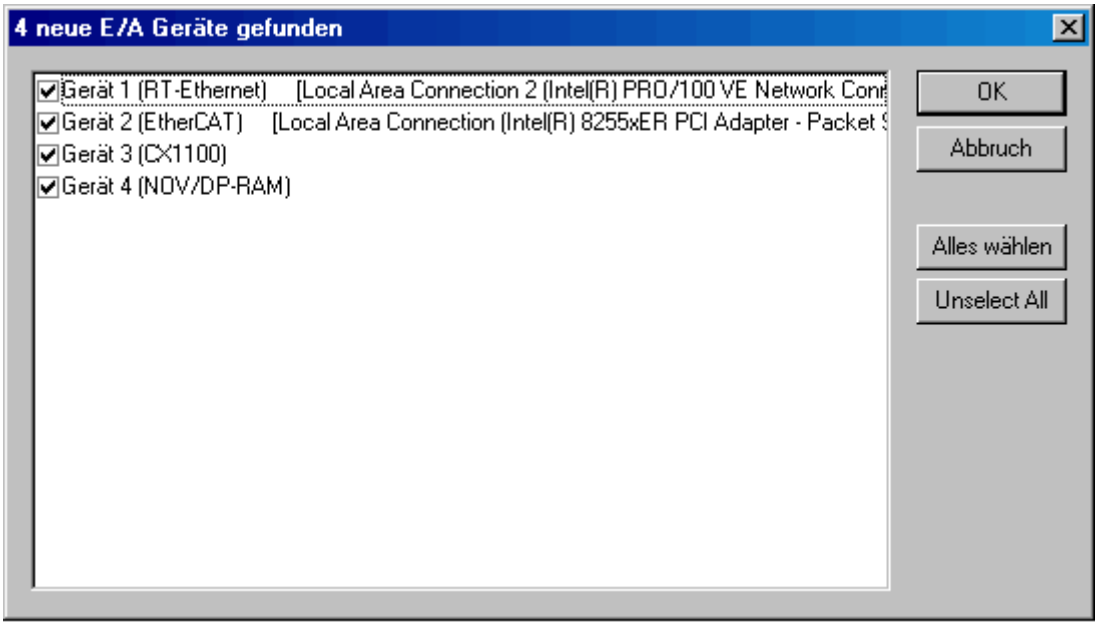
With TwinCAT 2.9 the devices can be scanned automatically ('Scan devices'). Found devices are listed afterwards under I/O devices in the tree view.
The the target system has to be in Config Mode for this function.

Select the 'I/O devices' entry with the right mouse button. A context menu opens, in which you must select the 'Scan devices' entry.

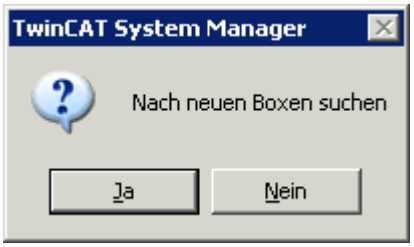


The following dialog opens:

TwinCAT scans the connected peripheral and lists the found devices -> 'OK'.



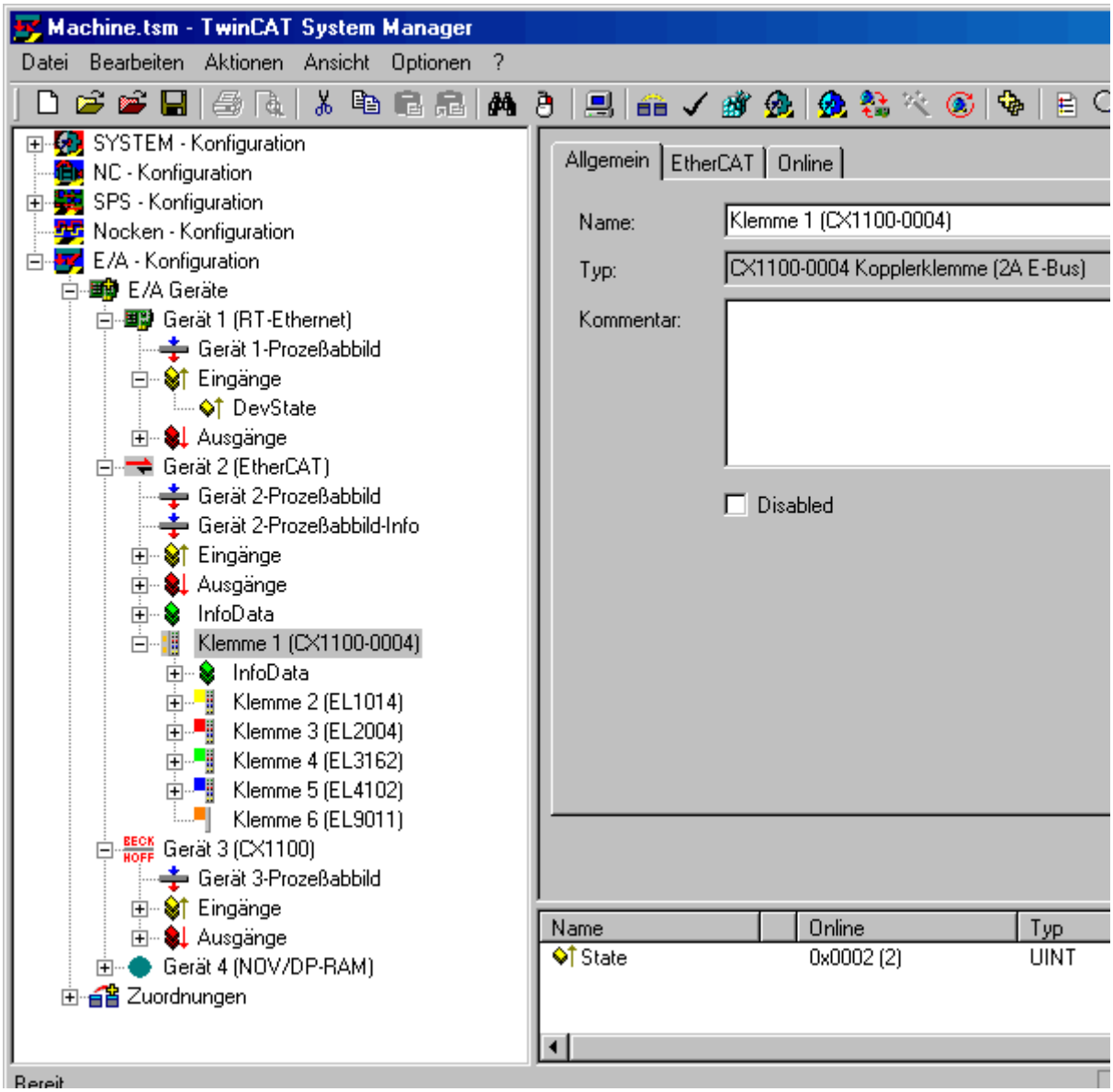
The System-Manager wants to search for boxes (Bus Couplers, Bus Terminals, Busterminal Controllers or Fieldbus Box Modules) at all channels of the CX1020 -> 'Yes'.



To activate the Free Run Modus confirm with ->'Yes'.

(Found I/O devices can be set to Free-Run mode, that means, e.g. I/O channels of Bus Terminals can be set (written) to a certain status without having any PLC project or other triggering task active.)

Now, the System Manager shows the found I/O configuration:



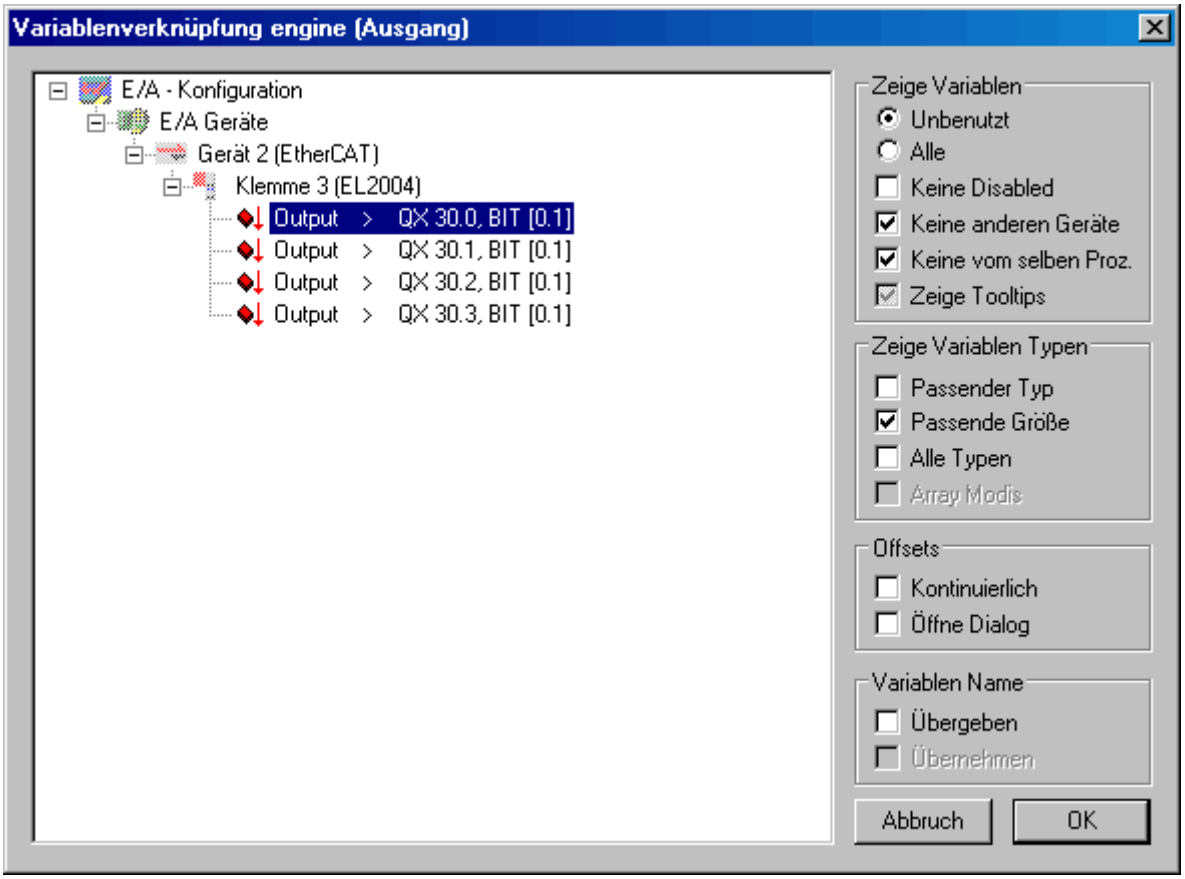
It goes without saying that you can rename the standard designations (device 1, box 1, terminal 1 etc.) To do this, slowly double click the corresponding name and enter the new designation.

Assigning variables to the input/output channels:

Up to this point, the complete hardware needed for the above example program has been configured. Next, the individual variables from the PLC project must be assigned to the individual input/output channels.

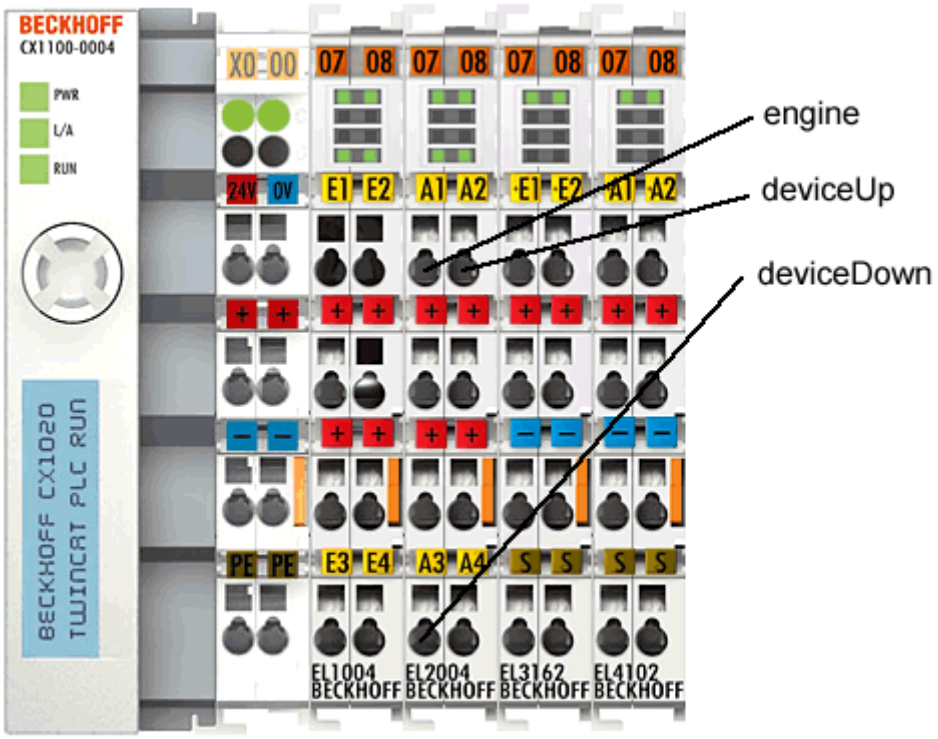
To do this, mark the terminal you wish to configure. A dialog box containing the input and output channels of the connected peripheral opens. Select the 'Variables' tab.

You see a list of the four output channels, but they are still all free. Select for all variables the wanted digital output channel.



Terminal EL1014	PLC variables	Meaning
Channel 1 (= output 1)	engine	Stepper motor control
Channel 2 (= output 2)	deviceUp	Drill up control
Channel 3 (= output 3)	deviceDown	Drill down control

Assignments of the bus terminals (sample configuration)

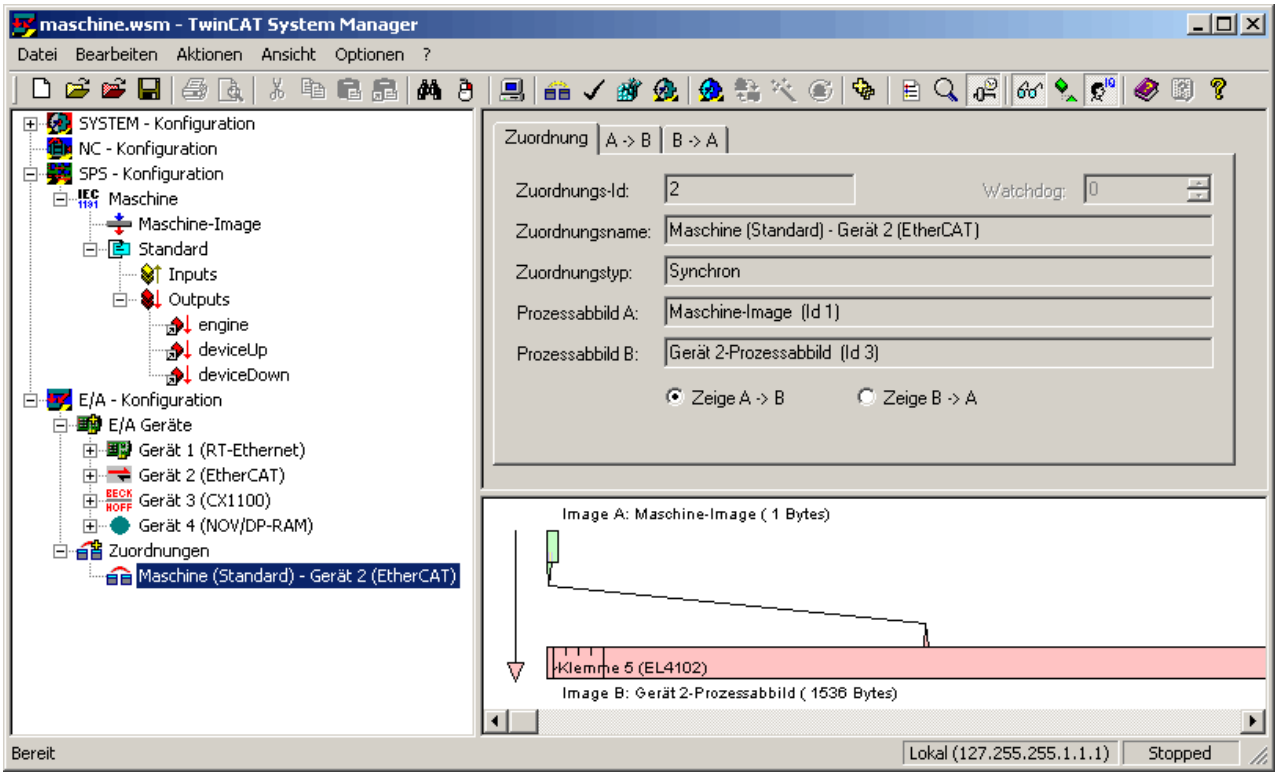


Saving the project:

You should save the configuration at this point to make sure you can access it later on. To do this, run the 'Save as…' command from the 'File' menu.

Mapping variables:

You have now configured the complete system for the above example program. You must now create the allocation for the registry. To do this, go to the 'Create mapping' command in the 'Actions' menu. Under the 'Mappings' tree entry, you now see 'Maschine (Standard) - Device 2 (EtherCAT)'. Click this entry. The following window opens on the right side:



You can define whether the data flow from A to B or from B to A is to be displayed. In this case, Image A corresponds to the process image of the PLC variables, i.e. the input/output variables. Image B corresponds to the process image of the I/O devices, in this case of the EtherCAT bus coupler. Each variable or bus terminal is color highlighted in the process image. If you stop on one of these areas with the mouse, a small display box appears in which the precise designation is shown. With a right mouse click you can zoom the picture.

Writing the configuration to the registry:

As the last step, you must save the configuration to registry because the information stored there is evaluated when you start TwinCAT. Run the 'Save in registry…' command from the 'Actions' menu. If an older configuration is already stored there, a safety prompt will appear, which you must confirm.

Restarting TwinCAT:

TwinCAT wants to restart in the 'Run Mode' to take the changes.

By the new configuration the loaded program on the Control doesn´t exisit anymore. Bus Terminals indicate their signal state by means of light emitting diodes.

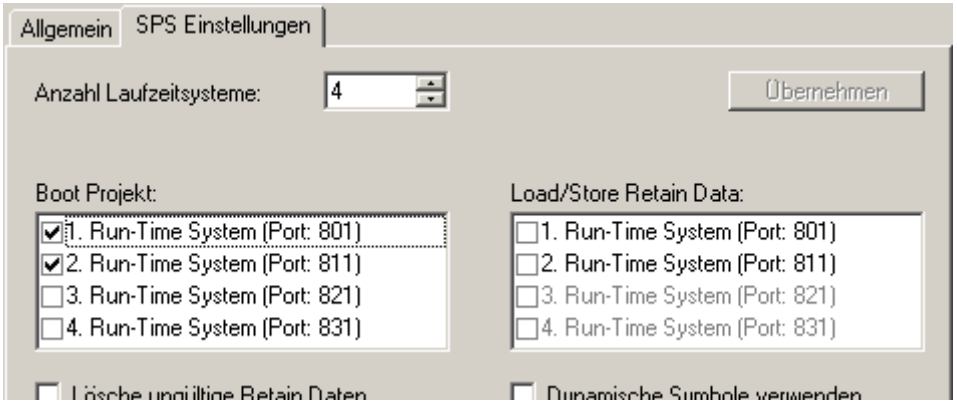
Open the program 'PLC Control' and select the command 'Login' in the menue 'online'. The message appears 'No program on the control!' Confirm with 'OK', and the program is loaded to the PLC.

With 'Online' -> 'Start' the programm starts with the new configuration.

The individual PLC variables are now output on the bus terminal EL2004. The Bus Terminals indicate their signal state by means of light emitting diodes.

Boot settings

The project has to be reloaded as soon as the PLC is restarted. Alternatively, you may create a boot project. This remains stored in the registry and starts as soon as the TwinCAT RealTime Server is started, provided that the configuration has not changed.
First, specify which runtime system is to include a boot project in the System Manager menu 'PLC Configuration' under the 'PLC Settings' tab, where the number of available runtime systems can also be specified.

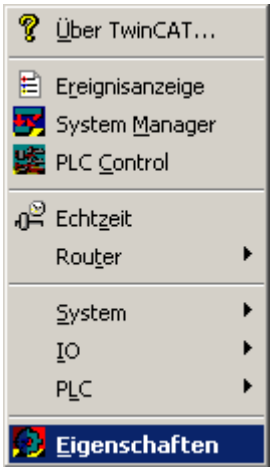


Load the associated programs via 'Add PLC project' in the System Manager, then link the I/Os and activate the configuration. Now execute the command 'Online' -> Creating a boot project' with TwinCAT PLC Control for each associated PLC program after login ('Online' -> 'Login'). The associated command can be used to delete it again. Please ensure that the associated runtime systems match. Log out again and close PLC Control.
Save your System Manager project.

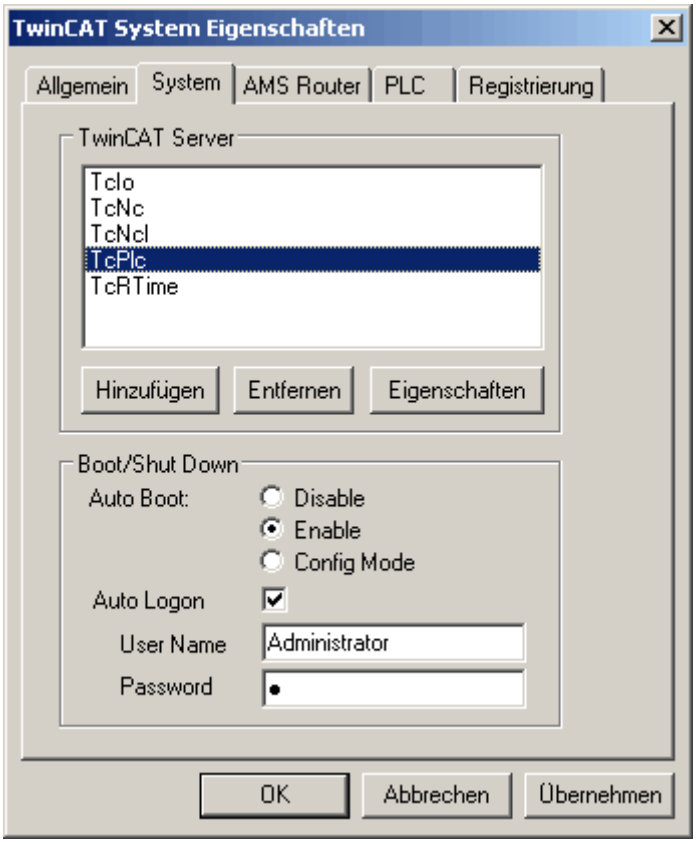
Boot project is retained

If the boot project is not overwritten or deleted, it remains loaded in the PLC until you create a new boot project for this runtime system or delete it.

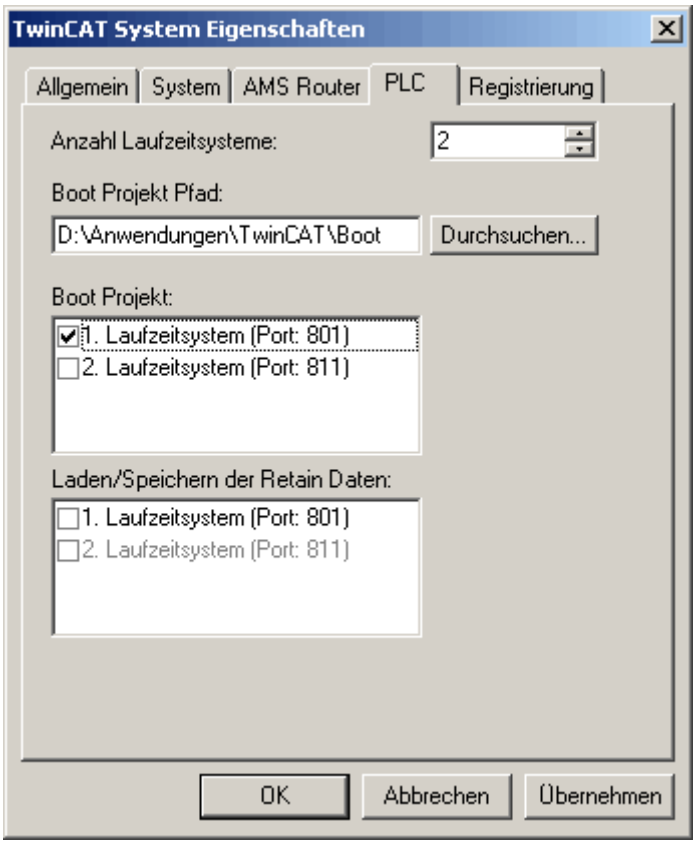
If TwinCAT is to automatically process one or several PLC programs on system startup, the number of runtime systems must be specified and the boot projects loaded (see above).
Now open the 'Properties' menu via the TwinCAT button in the taskbar:



In the 'System' tab, you must select the 'Enable' option under 'Auto-Boot'. If you want the PLC program to start completely without your intervention, you must also enter your login data for 'Auto Logon'. If this option is not checked, a user login is required first after the system start:



The settings for 'Auto-Boot' are identical with the setting 'System configuration' -> 'Boot settings' in the System Manager.
Likewise, the settings of the 'PLC' tab are identical to the settings under 'PLC Configuration' -> 'PLC Settings' in the System Manager:



After a restart of the system TwinCAT will start in 'Run' mode (assuming auto logon) and start to process the loaded and released boot projects independently.

7.4 Example "Machine" with ...

7.4.1 Example: machine with Microsoft Visual C#

Microsoft Visual Studio 2005 is a development environment for creating C# projects. The machine example is used for familiarisation with the integration of the TwinCAT ADS .NET component with the programming language C#.

Required software:

- Microsoft .NET Framework Version 2.0, for more information click here
- Microsoft Visual Studio 2005
- TwinCAT 2.10

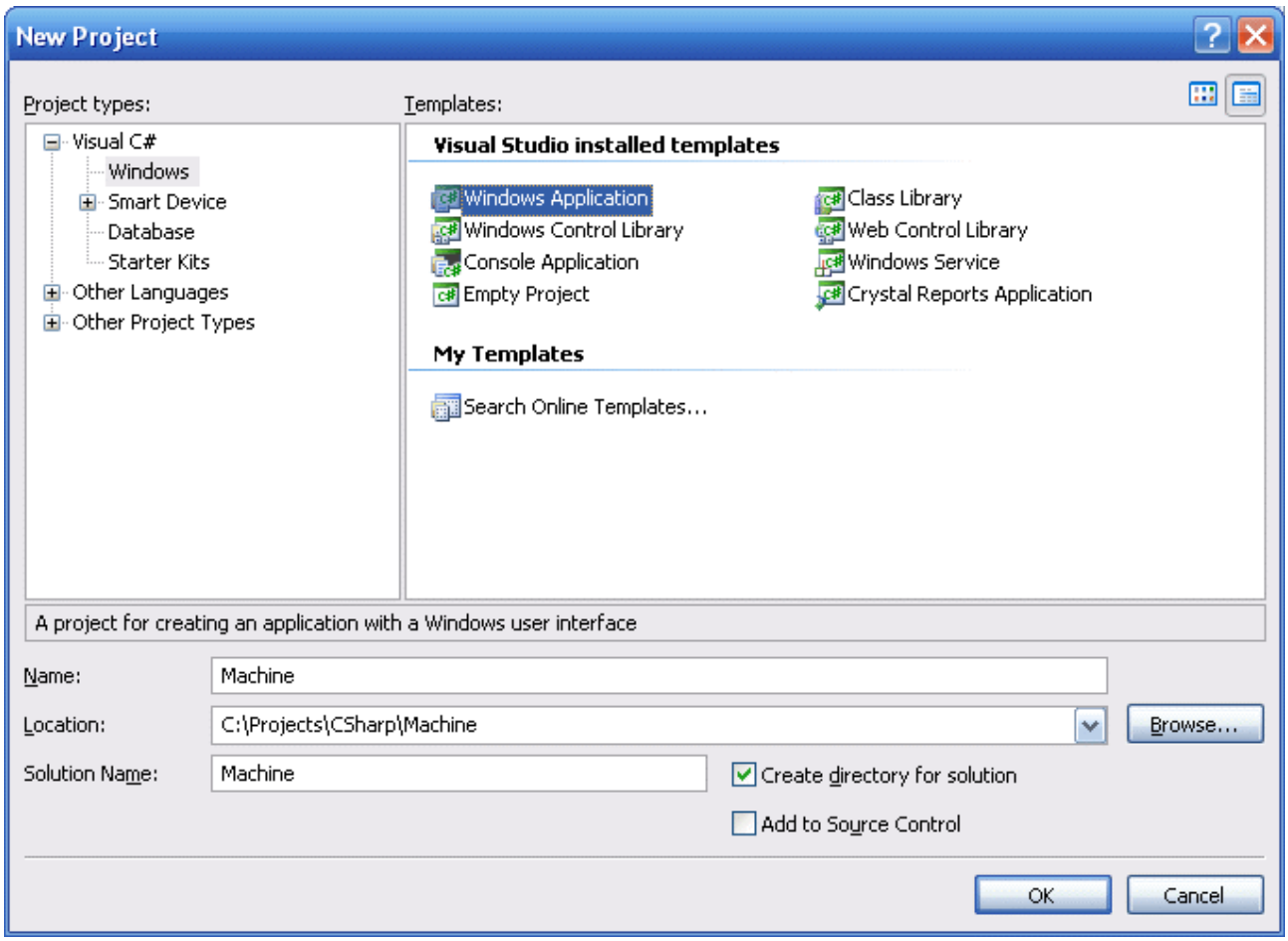
Before you can start the C# program, TwinCAT and the PLC program must be active. If Microsoft Visual Studio 2005 is not installed on your computer, please install Microsoft .NET Framework Version 2.0. This sets up the required DLLs on your system.

First steps...

Step by step familiarisation with the development of a C# program and integration of the TwinCAT ADS .NET component based on an example.

1. Create new project:

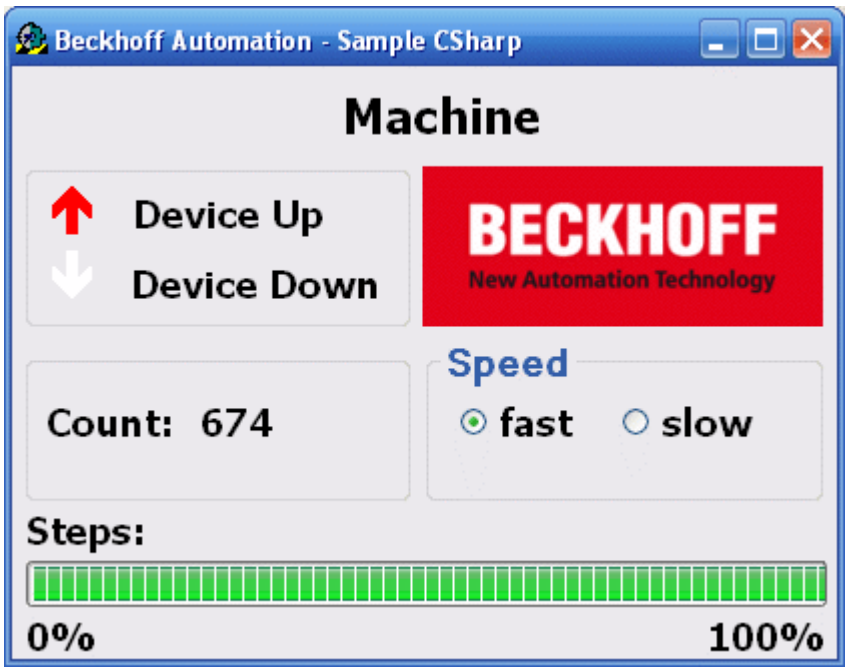
Start Visual Studio 2005. Create a new project by clicking File -> New -> Project in the menu. A new project based on different templates can be created via the dialog box *'New Project'*. Under *'Project Types'* select the programming language Visual C#, and under the templates select *'Windows Application'*. Enter a new name for your project, in this case please enter *'Machine'*. Then select the directory path for your project under *'Location'*.



2. Creating a user interface

First create an interface in the design mode of the form 'frmMachine' with the *Toolbox*. The settings for the different controls (such as 10 labels, 2 radio buttons, 1 progress bar, 1 picture box, group box...) can be viewed and set in the Visual Studio Properties window.

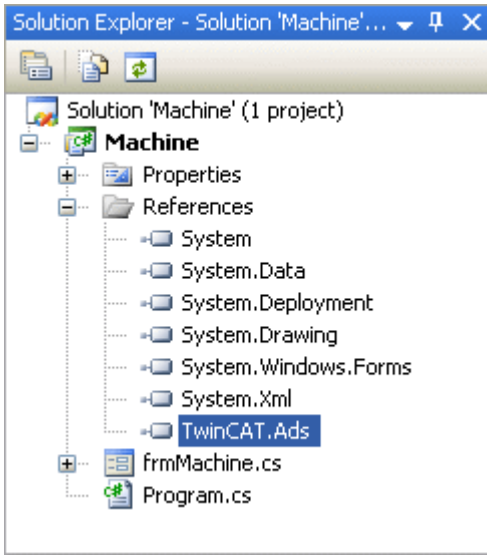
```
// Group fields (group box)
grpDevice.Text = "";
grpCount.Text = "";
grpSpeed.Text = "Speed";

// 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.D
rawing.GraphicsUnit.Point, ((byte)(2)));

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

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



In the upper left you see the two outputs that are also output to the Bus Terminals. The bottom left shows the variable for counting the workpieces. The cycle speed of the motor can be changed via the 'Speed' field on the right. The *'Steps'* display shows the number of cycles that are output on output 1.

3. Adding a reference

First a reference called TwinCAT.Ads.dll must be added. The TwinCAT ADS .NET component is integrated via the menu -> Project -> Add Reference. To check whether the reference was actually integrated, you can call up the Solution Explorer (*CTRL + W + S*).



4. Edit source text

Once the interface has been created and the TwinCAT ADS component has been integrated, you can change to the C# source text. The required namespaces 'System.IO' and *'TwinCAT.Ads'* are added in the top row of the source text.

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

This is followed by declarations within the frmMachine class.

```
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;
```

The method 'frmMachine_Load' is started when the Windows application is called. This method is linked by linking the 'Load' event of the form with the method in the Properties window. It is used to generate instances of different classes and create a link with runtime system 1 of the TwinCAT.Ads component via port 801.

```
//-----------------------------------------------------//
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");
    }

    //...
```

Linking PLC variables:

In the frmMachine_Load event of the form, a connection to each variable in the PLC is created via the TcAdsClient.AddDeviceNotification()method. The handle for this connection is stored in an array. The TransMode parameter specifies the data exchange type. In this case AdsTransMode.OnChange is used to specify that the PLC variable is only transferred if the value in the PLC has changed (see AdsTransMode). The parameter *cycleTime* determines how often the PLC should check whether the corresponding variable has changed. The variables are then linked with a method *'tcClient_OnNotification'* (that still has to be written), which is called when a variable changes.

```
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.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);

    // 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);
    }
}
```

Definition:

The method 'AddDeviceNotification()' was used for linking the variables.

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

- **variableName:** Name of the PLC variable.
- **dataStream:** Data stream receiving the data.
- **offset:** Data interval in the stream.
- **length:** Data length in the stream.
- **transMode:** Event if the variable changes.
- **cycletime:** Time (in ms) after which the PLC server checks whether the variable has changed.
- **maxDelay:** Latest time (in ms) after which the event has finished.
- **userData:** Object that can be used for storing certain data.

The method 'CreateVariableHandle' was used for linking the variable *'hSwitchWrite'*.

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

- **variableName:** Name of the PLC variable.

Write method:

A method that does not exist yet was referred to above. This method ('tcClient_OnNotification') is written next. The method is called if one of the PLC variables has changed.

```
//----------------------------------------------//
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 changedif(e.NotificationHandle == hDeviceUp)
    {
        //
Adapting the colours of the diagrams to match the variablesif (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 buttonif (binReader.ReadBoolean() == true)
        {
        optSpeedFast.Checked = true;
```

```
        }
        else
        {
        optSpeedSlow.Checked = true;
        }
    }
    }
    catch (Exception ex)
    {
    MessageBox.Show(ex.Message);
    }
}
```

Lastly, we require two methods for setting the machine to fast or slow. They are used to switch a virtual switch by writing a value to the PLC variable 'switch'.

```
//-----------------------------------------------------//
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);
    }
}
```

Last but not least we need to ensure that the methods 'optSpeedFast_Click' and *'optSpeedSlow_Click'* are called for the right event. To this end switch to Design view, select the radio button *'optSpeedFast'*, click on *'Click'* under Events in the Properties window and select the method *'optSpeedFast_Click'*. Proceed accordingly for the radio button *'optSpeedSlow'* and the method *'optSpeedSlow_Click'*.

Delete notifications and handles:

In the frmMachine_FormClosing event of the form the links are activated again with the method DeleteDeviceNotification().

```
//---------------------------------------//
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();
}
```

The PLC programMachine_Final.proshould start on runtime system 1, and the created C# programMachine.execan be tested.

Download C# program example:

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

7.4.2 Example: machine with Microsoft Visual Basic .NET

Microsoft Visual Studio 2005 is a development environment for creating Visual Basic projects. The machine example is used for familiarisation with the integration of the TwinCAT ADS .NET component with the programming language Visual Basic.

Required software:

- Microsoft .NET Framework Version 2.0, for further information please refer to http://msdn2.microsoft.com

- Microsoft Visual Studio 2005
- TwinCAT 2.10

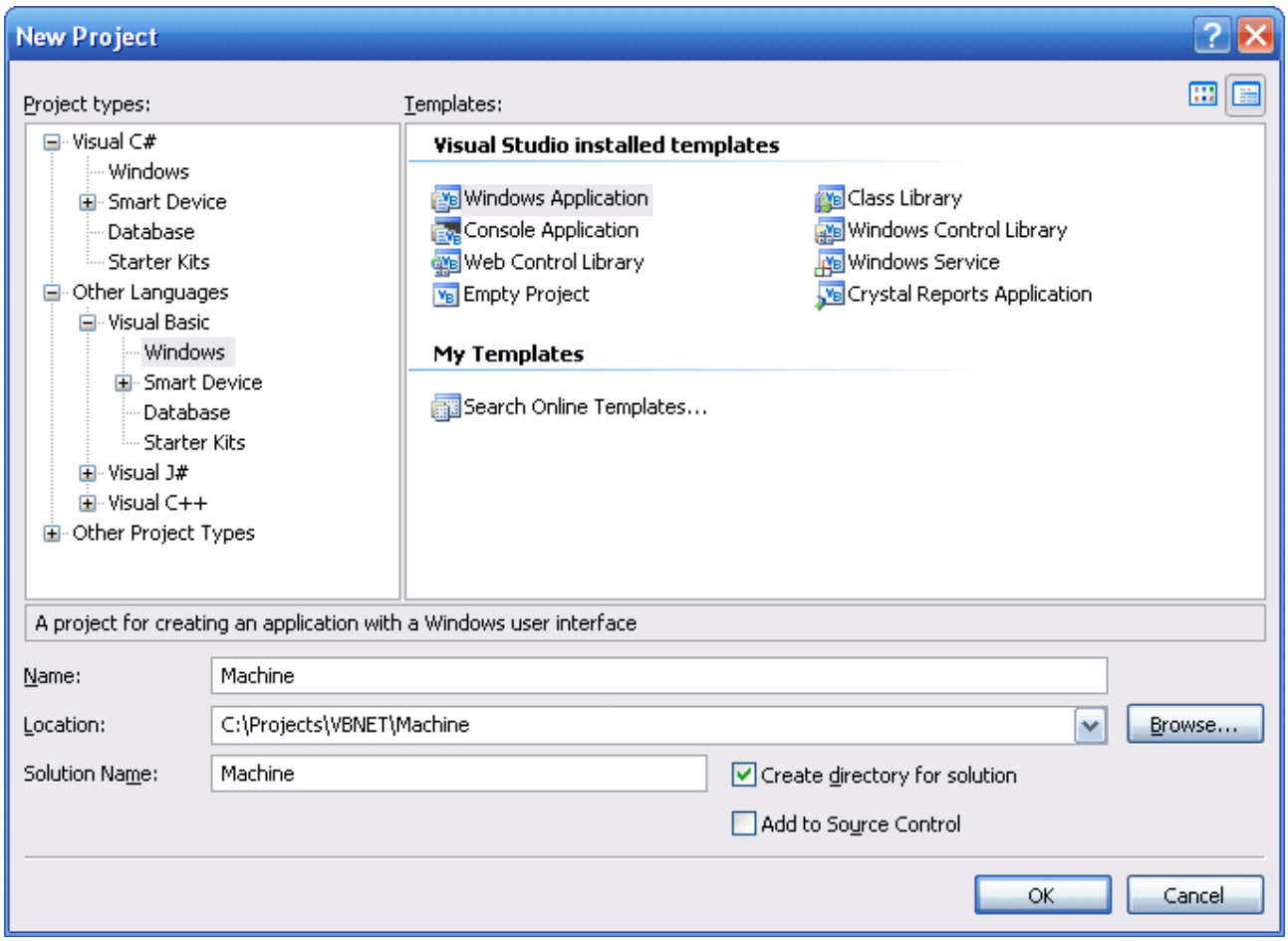
Before you can start the Visual Basic program, TwinCAT and the PLC program must be active. If Microsoft Visual Studio 2005 is not installed on your computer, please install Microsoft .NET Framework Version 2.0. This sets up the required DLLs on your system.

First steps...

The following steps describe the development of a Visual Basic program and integration of the TwinCAT ADS .NET component.

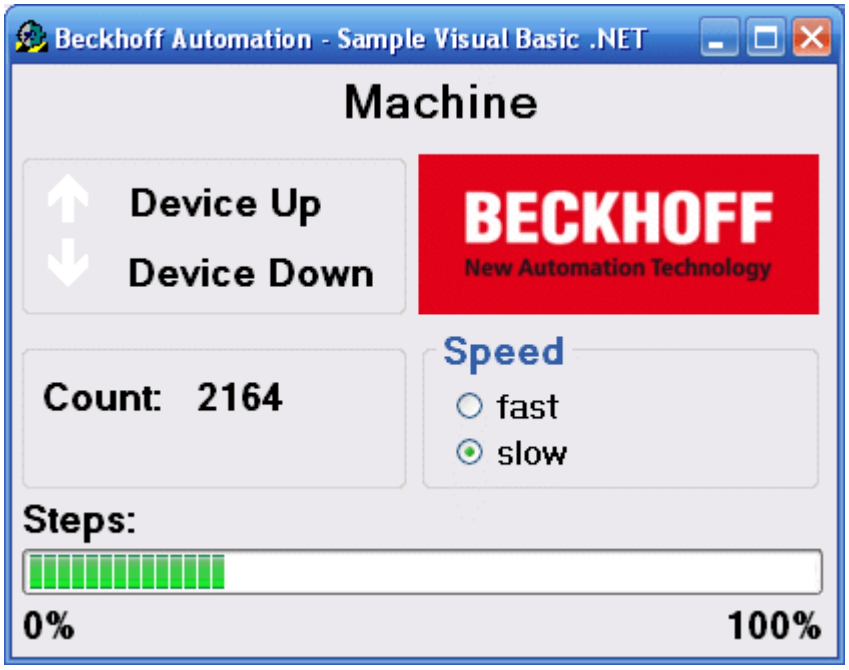
1. Create new project:

Starting Visual Studio 2005 and creating a new project by selecting -> File -> New -> Project from the menu. A new project based on different templates can be created via the dialog box *'New Project'*. Under *'Project Types'* select the programming language 'Visual Basic', and under the templates select *'Windows Application'*. Enter a new name for your project, in this case please enter *'Machine'*. Then select the directory path for your project under *'Location'*.



2. Creating a user interface

First create an interface in the design mode of the form 'frmMachine' with the *Toolbox*. The settings for the different controls (such as 10 labels, 2 radio buttons, 1 progress bar, 1 picture box, group box...) can be viewed and set in the Visual Studio Properties window.



In the upper left you see the two outputs that are also output to the Bus Terminals. The bottom left shows the variable for counting the workpieces. The cycle speed of the motor can be changed via the 'Speed' field on the right. The *'Steps'* display shows the number of cycles that are output on output 1.

3. Adding a reference

First a reference called TwinCAT.Ads.dll must be added. The TwinCAT ADS .NET component is integrated via the menu -> Project -> Add Reference.

Further information about integration of the TwinCAT ADS .NET component in Microsoft Visual Studio .NET .

4. Edit source text

Once the interface has been created and the TwinCAT ADS component has been integrated, you can change to the Visual Basic source text. The required namespaces 'System.IO' and *'TwinCAT.Ads'* are added in the top row of the source text.

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

This is followed by declarations within the frmMachine class.

```
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 A
s IntegerPrivate hCount As IntegerPrivate hSwitchNotify As IntegerPrivate hSwitchWrite As Integer
```

The method 'frmMachine_Load' is started when the *'Windows Application'* is called. This method is linked with the method in the Properties window by linking the 'Load' event of the form. It is used to generate instances of different classes and create a link with runtime system 1 of the TwinCAT.Ads component via port 801.

```
'-----------------------------------------------------
    '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

'...
```

Linking PLC variables:

In the frmMachine_Load event of the form, a connection to each variable in the PLC is created via the TcAdsClient.AddDeviceNotification() method. The handle for this connection is stored in an array. The TransMode parameter specifies the data exchange type. In this case AdsTransMode.OnChange is used to specify that the PLC variable is only transferred if the value in the PLC has changed (see AdsTransMode). The parameter *cycleTime* determines how often the PLC should check whether the corresponding variable has changed. The variables are then linked with the method *'tcClient_OnNotification'* (see "Write method 'tcClient_OnNotifiaction'"), which is called when a variable changes.

```
    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.OnCha
nge, 10, 0, DBNull.Value)
        hDeviceDown = tcClient.AddDeviceNotification(".deviceDown", dataStream, 2, 1, AdsTransMode.O
nChange, 10, 0, DBNull.Value)
        hSteps = tcClient.AddDeviceNotification(".steps", dataStream, 3, 1, AdsTransMode.OnChange, 1
0, 0, DBNull.Value)
```

```
        hCount = tcClient.AddDeviceNotification(".count", dataStream, 4, 2, AdsTransMode.OnChange, 1
0, 0, DBNull.Value)
        hSwitchNotify = tcClient.AddDeviceNotification(".switch", dataStream, 6, 1, AdsTransMode.OnC
hange, 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
```

The method 'AddDeviceNotification()' was used for linking the variables.

```
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:** Name of the PLC variable.
- **dataStream:** Data stream receiving the data.
- **offset:** Interval in the data stream.
- **length:** Length in the data stream.
- **transMode:** Event if the variable changes.
- **cycletime:** Time (in ms) after which the PLC server checks whether the variable has changed.
- **maxDelay:** Latest time (in ms) after which the event has finished.
- **userData:** Object that can be used for storing certain data.

The method 'CreateVariableHandle' was used for linking the variable *'hSwitchWrite'*.

```
Public Function CreateVariableHandle(ByVal variableName As String) As Integer
```

- **variableName:** Name of the PLC variable.

Write method 'tcClient_OnNotification':

The method is called if one of the PLC variables has changed.

```
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 entsprechened 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
```

Lastly, we require two methods for making the machine faster or slower. They are used to switch a virtual switch by writing a value to the PLC variable 'switch'.

```
' 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 optSpeedFas
t.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 optSpeedSlo
w.Click
    Try
        tcClient.WriteAny(hSwitchWrite, False)

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

To ensure that the methods 'optSpeedFast_Click' and *'optSpeedSlow_Click'* are called at the right event, select the radio button *'optSpeedFast'* in Design view, click on *'Click'* under Events in the Properties window, and select the method *'optSpeedFast_Click'*. Proceed accordingly for the radio button *'optSpeedSlow'* and the method *'optSpeedSlow_Click'*.

Delete notifications and handles:

In the frmMachine_FormClosing event of the form the links are activated again with the method DeleteDeviceNotification().

```
 '---------------------------------------
    '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.FormC
losingEventArgs) 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
```

You can now start the PLC program Machine_Final.pro in runtime system 1 and then start the VisualBasic program Machine.exe you created.

Download Visual Basic example:

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

7.4.3 Sample Machine With Microsoft Visual Basic 6.0

TwinCAT ADS OCX

TwinCAT has several programming interfaces in order to integrate applica-tion-specific programs in the system. Visual Basic from Microsoft is one programming language that is sup-ported. The strength of this programming language is the fact that graphi-cal user interfaces are created and it is possible to link them to databases. Visual Basic has been widely spread for some years now and is constantly enhanced by Microsoft. Many third-party vendors offer add-on modules for various areas. The generic term for such modules is OCX.Beckhoff supplies an OCX for TwinCAT that bears the designation Ad-sOCX. AdsOCX provides methods for communicating via the TwinCAT message router with other ADS devices (PLC, NC/ CNC, ...).The included example program shows how to access individual variables of the TwinCAT PLC server from Visual Basic.

Access methods:

AdsOCX contains various methods for reading the values out of other ADS devices. The decision as to which method to use depends on the environment in which the program is to be run. You will find further notes and detailed references to the individual functions in the special AdsOCX instructions. Only a brief overview of the individual access methods will be given here:

Access	Meaning
connect	As soon as communication between a PLC variable and a Visual Basic variable is needed, a connection between these two variables is established by activation of a method. During the further course of the program, TwinCAT adapts the Visual Basic variable to the PLC variable. This type of data exchange can also be used so as to activate an event function (event-controlled data transfer) in the event of a change in a PLC variable in the Visual Basic program.
synchronous	After activation of a read/write method, execution of the Visual Basic program is interrupted until the requested data has arrived. The Program can then continue working with the new data.
asynchronous	When using asynchronous access, execution of the Visual Basic program is not interrupted and instead, execution of the next command is continued automatically. Once the requested data arrives in AdsOCX, an event function is triggered in the Visual Basic program in which the value is passed on as the parameter

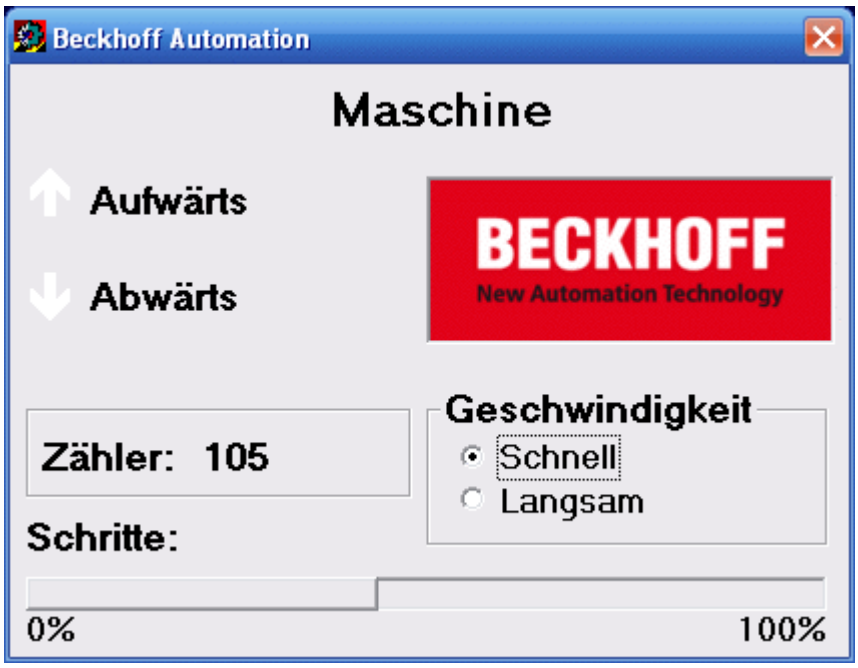
Sample program

Starting TwinCAT and the PLC program:

TwinCAT and the PLC program must be active before you can start the Visual Basic program.

Starting the Visual Basic program:

Start the 'maschine.exe' program. (TwinCAT\ Samples\First Steps\)



In the left area, you see the two outputs, which are also output to the bus terminals. The variable that counts the workpieces is shown on the bottom left. You can modify the cycle speed of the motor in the 'speed' box. The 'position' display corresponds to the number of cycles that are output to the output 1.

Sourcecode:

```
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 starte
d'-------------------------------------------------------
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
```

Operating principle:

In Form1_Load() the language dependent text are loaded as resources.The Text will displayed according to the preselected language in Windows NT. The Method ‚AdsReadVarConnect' connects the appropiate VB variables to the PLC System. The parameter of the methods are::

Parameter	Meaning
adsVarName	Name of the PLC variable
cbLenght	Length of the data in bytes
nRefreshType	Type of data transfer
nCycleTime	Refresh cycle in ms
pData	Visual Basic variable in which the data is written

The AdsOcx1_AdsReadConnectUpdate() function is activated whenever the variable 'count', 'deviceUp', 'deviceDown' or 'steps' has changed. In it, the objects are animated on the form with the variables of the PLC.

The optFast_Click() and optSlow_Click() functions are activated whenever the user clicks the 'fast' or 'slow' marking box. In these functions, the 'switch' variable is set to TRUE or FALSE. The TwinCAT PLC server reads out this variable and appropriately modifies the behavior of the flashing ('DriveType' function block in the PLC program).

Parameter	Meaning
hVar	Handle of the PLC variable
cbLenght	Length of the data in bytes
pData	Visual Basic variable, which contains the data

When a form is closed, all variables that are no longer needed should be separately from the PLC variables. This is done in the Form1_Unload() function.

Download Sample:

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

7.4.4 Example: machine with Microsoft Visual C++ .NET

Microsoft Visual Studio 2005 is a development environment for creating C++ projects. The machine example is used for familiarisation with the integration of the TwinCAT ADS DLL .LIB component with the programming language C++.

Required software:

- Microsoft .NET Framework Version 2.0, for more information click here
- Microsoft Visual Studio 2005
- TwinCAT 2.10

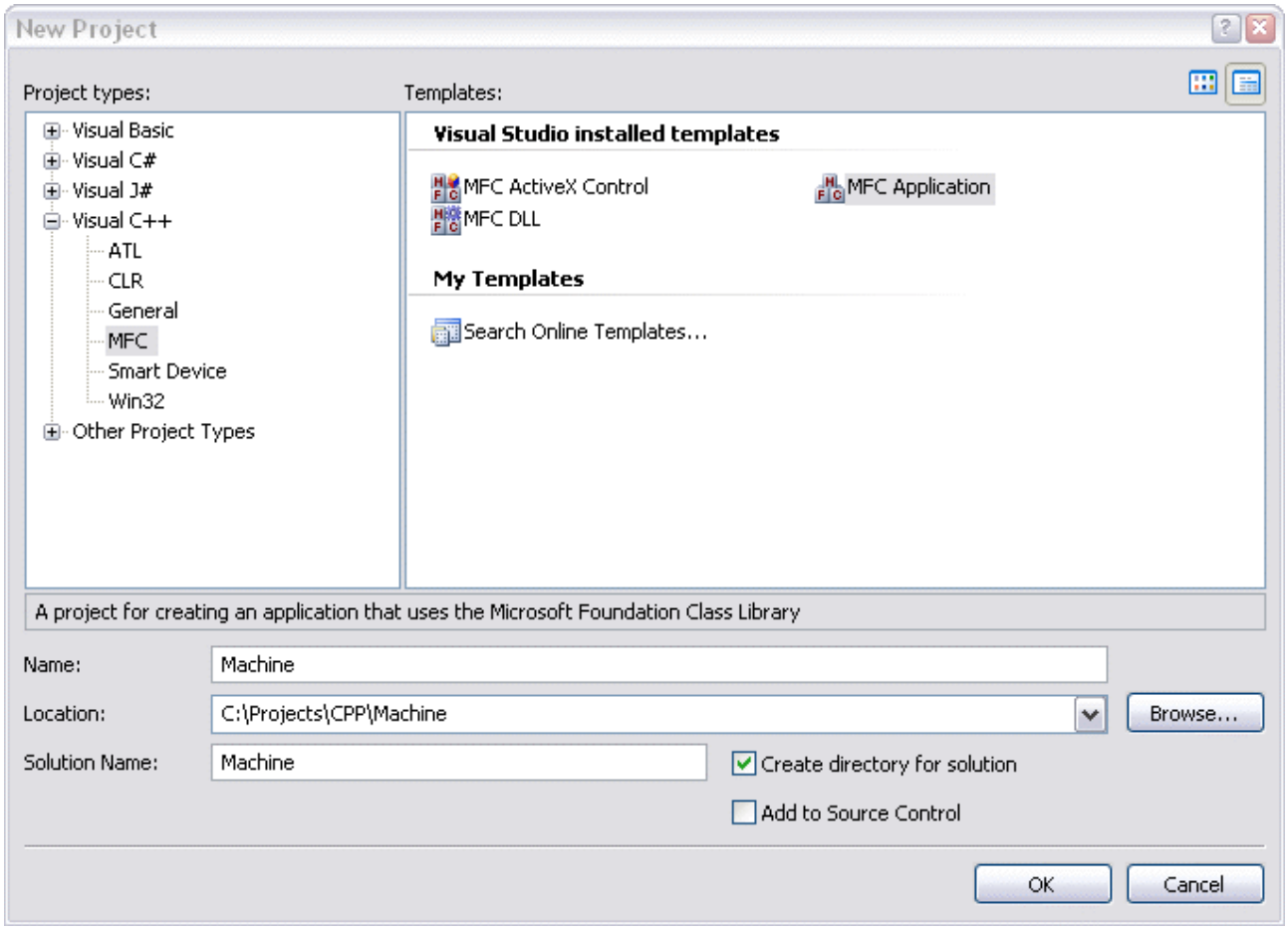
Before you can start the C++ program, TwinCAT and the PLC program must be active. If Microsoft Visual Studio 2005 is not installed on your computer, please install Microsoft .NET Framework Version 2.0. This sets up the required DLLs on your system.

First steps...

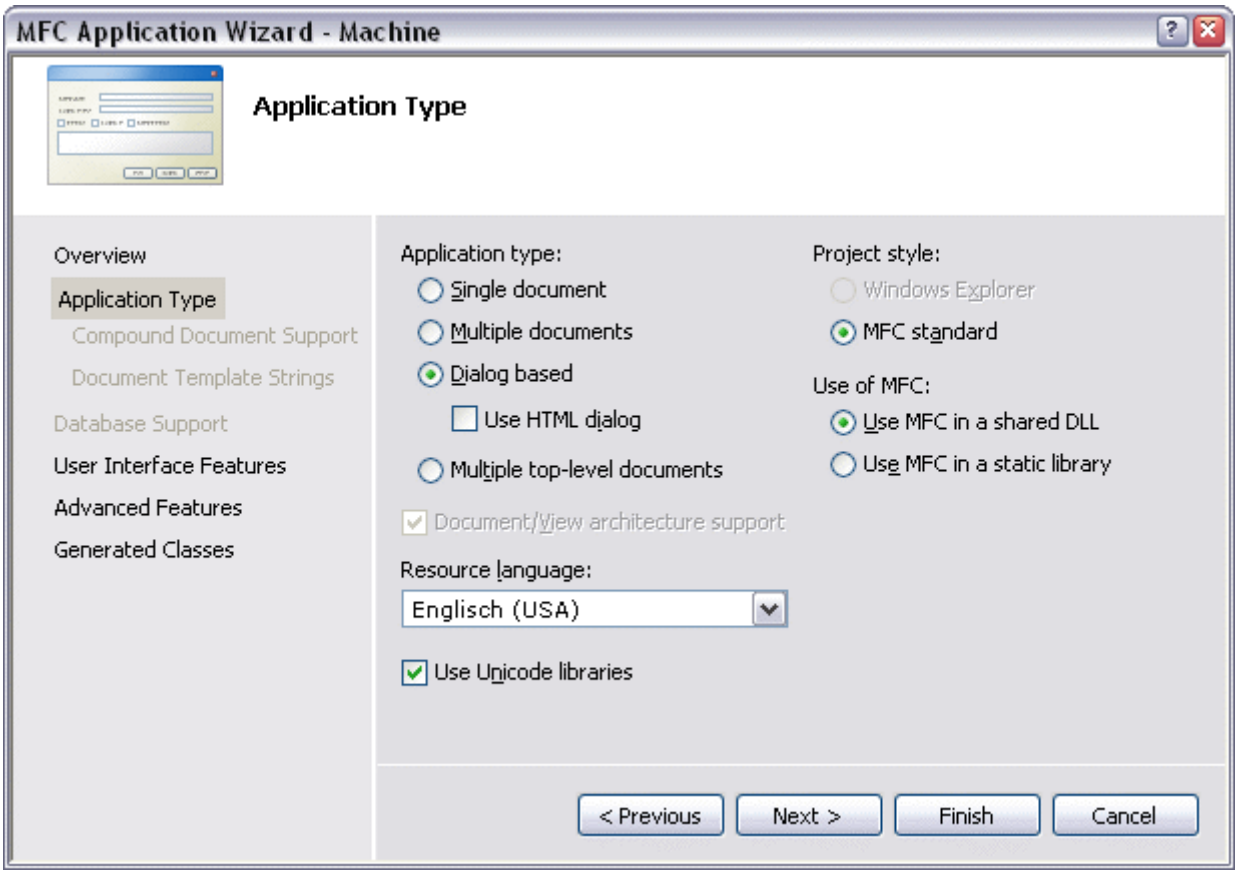
Step by step familiarisation with the development of a C++ program and integration of the TwinCAT ADS DLL .LIB component based on an example.

1. Create new project:

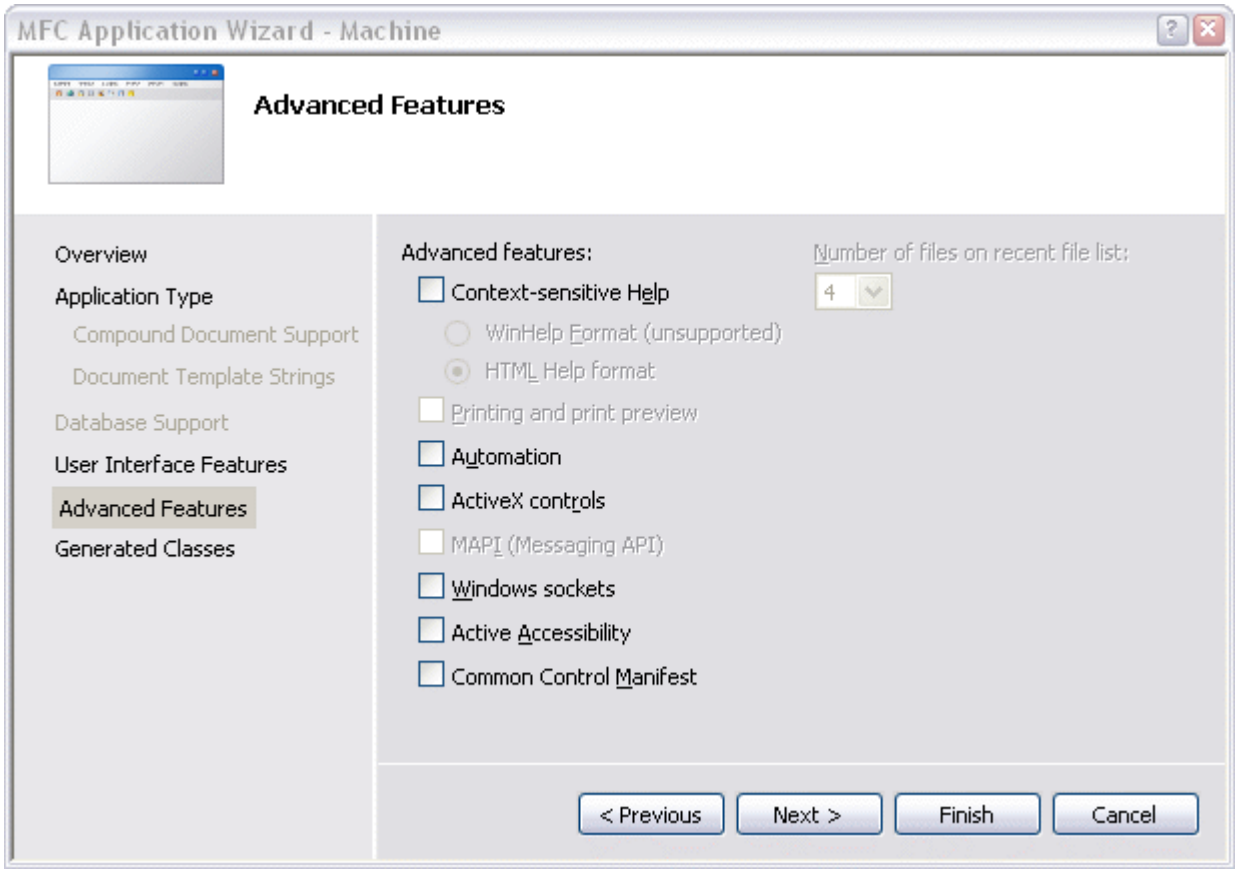
Create a new project by clicking 'File -> New -> Project...' in the menu. A new project based on different templates can be created via the dialog box *'New Project'*. Under *'Project Types'* select the programming language *'Visual C++'*, *'MFC'*, and under the templates select *'MFC Application'*. Enter a new name for your project, in this case please enter *'Machine'*. Then select the directory path for your project under *'Location'*.



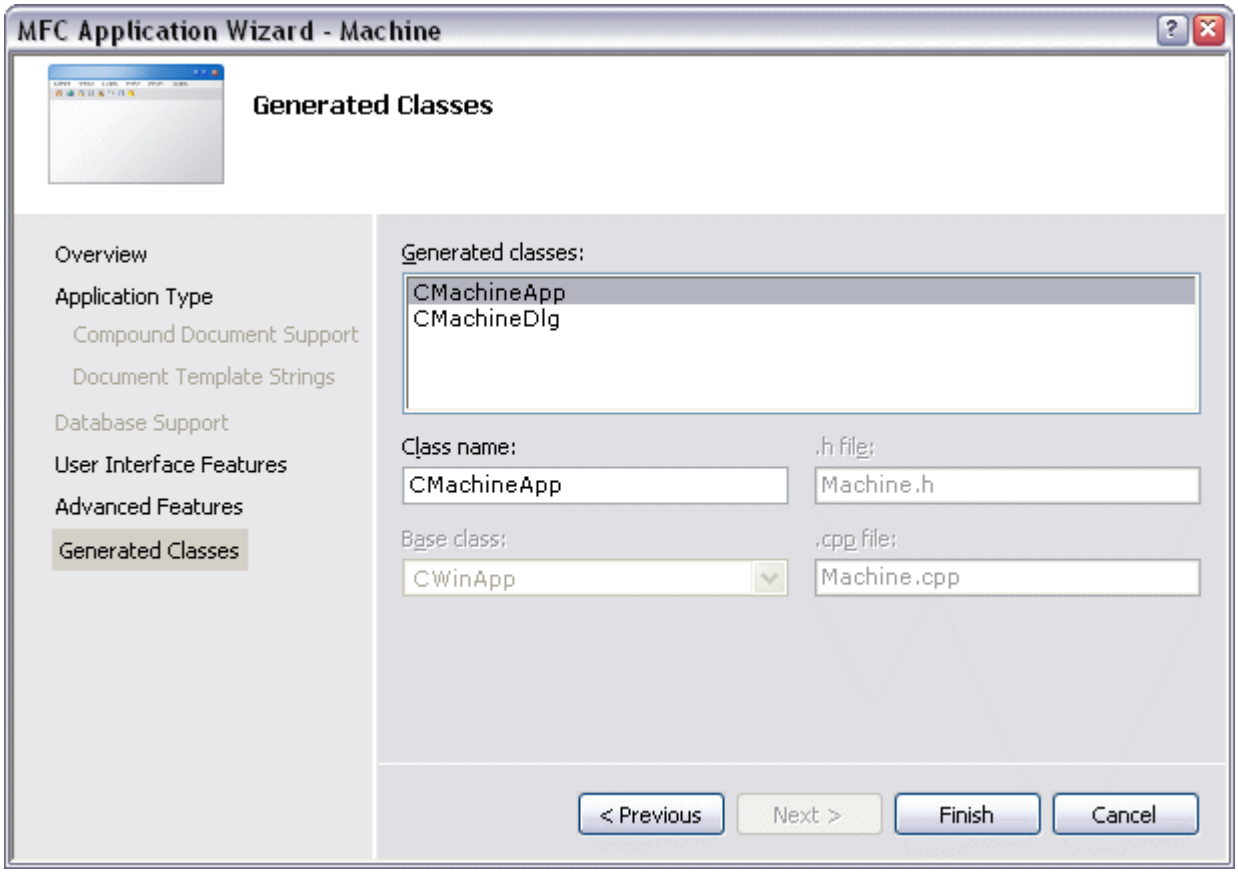
After the wizard welcome window please accept the following settings:



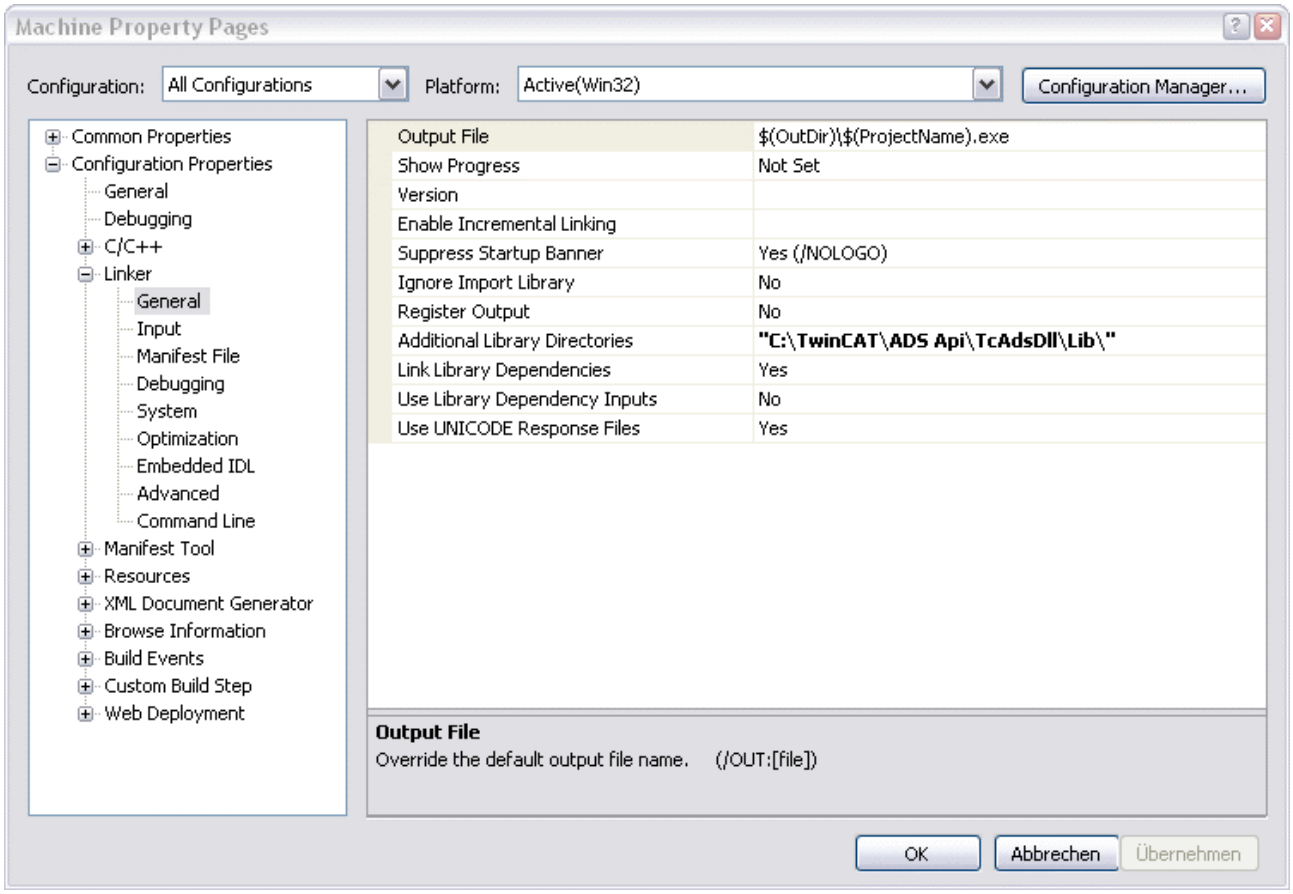
No further features are required.



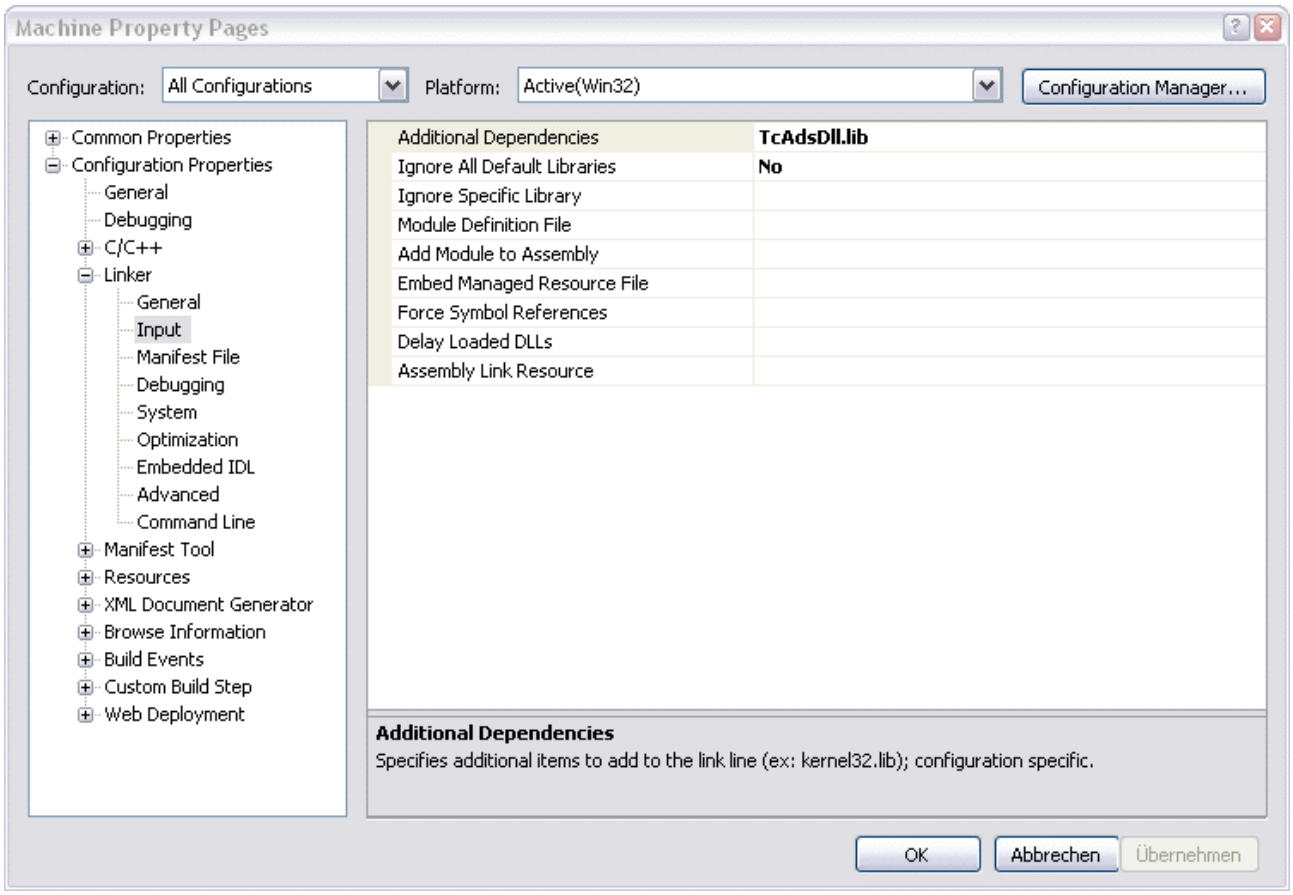
Last step in the wizard. No changes are required here.



Finally, link the required TcAdsDll.lib. Enter the path under 'Project -> Machine Properties... -> Configuration Properties -> Linker -> General -> Additional Library Directories'.



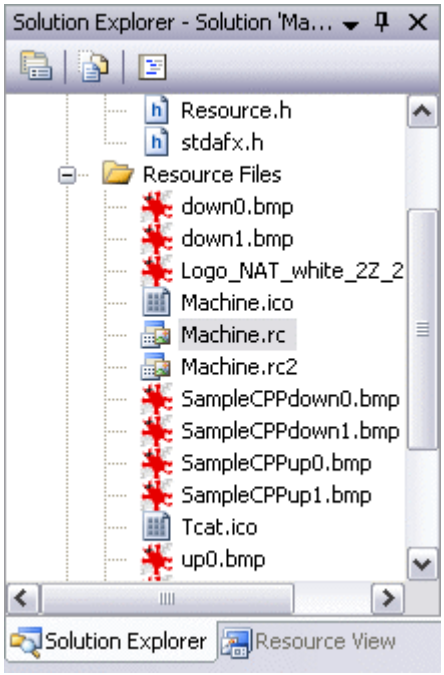
An additional entry is required under 'Project -> Machine Properties... -> Configuration Properties -> Linker -> Input -> Additional Dependencies'.



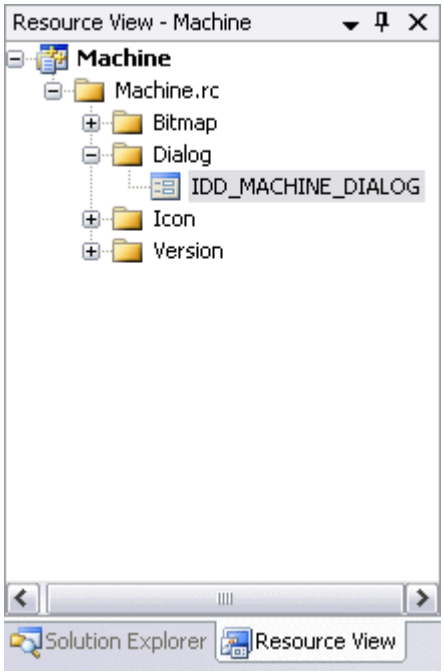
Please ensure that at the top left under 'Configuration:''*All Configurations*' is selected.

2. Creating a user interface

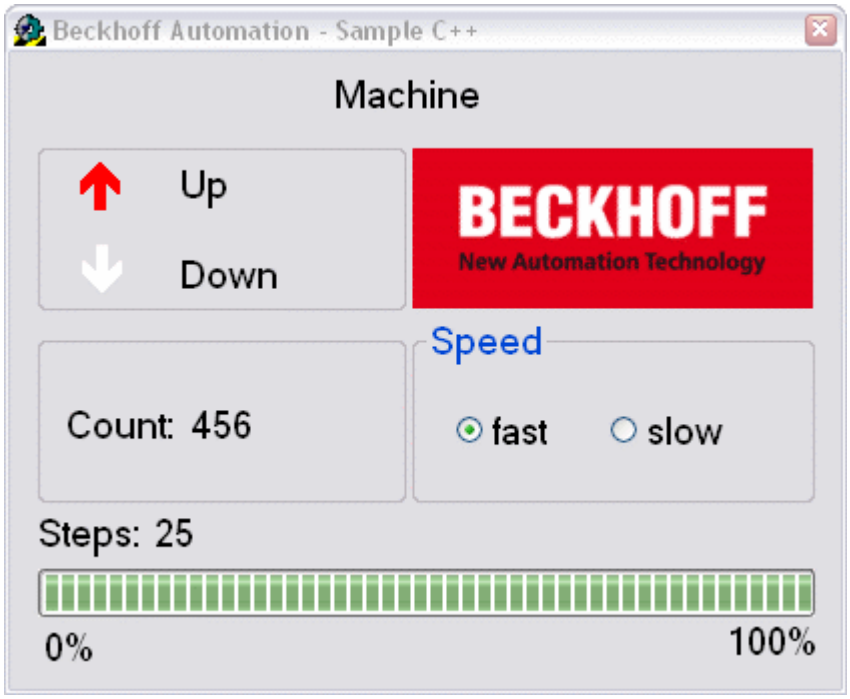
First create an interface in the design mode of the form 'Machine.rc' with the toolbox. Open *'Machine.rc'* by double-clicking in the *'Solution Explorer'*.



You are now in 'Resource View'. A further double-click on *'IDD_MACHINE_DIALOG'* brings up the form for positioning all required controls (such as 9 static texts, 3 picture boxes, 2 radio buttons, 1 progress bar, group boxes...).



The result should look similar to this.



In the upper left you see the two outputs that are also output to the Bus Terminals. The bottom left shows the variable for counting the workpieces. The cycle speed of the motor can be changed via the 'Speed' field on the right. The *'Steps'* display shows the number of cycles that are output on output 1.

3. Edit source text

Once the interface has been created and the TwinCAT DLL.LIB component has been integrated, you can change to the C++ source text. Declarations are implemented in the lower area of the source text for 'MachineDlg.h'.

```
[…]

// Variablen des Programms// Variables of the programmprivate:
     // 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;
};
```

The method 'CMachineDlg::OnInitDialog()' is called when the MFC application is started. It is used for generating instances of different classes, establishing a connection with runtime system 1 of component TcAdsDll.lib via port 801, loading required bitmaps and creating the handles for PLC variables.

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

     […]

     // Kommunikationsport für lokale SPS (Laufzeitsystem 1) öffnen// Open the communication port fo
r 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, ".ste
ps" );
     nErr = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hCount, 7, ".cou
nt" );
     nNotify = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hNotify, 8, "
.switch" );

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

Output PLC variables:

Variable output is organised in 'CMachineDlg::OnTimer()'. The parameter *'nElapse'* of the *'SetTimer()'* method (already specified in *'CMachineDlg::OnInitDialog()'*) determines the time (in ms) that should pass before the timer is executed again.

```
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);
};
```

The method 'AdsSyncReadWriteReq()' was used for linking the variables.

```
long AdsSyncReadWriteReq(
PAmsAddr pAddr, ULONG  nIndexGroup, ULONG nIndexOffset, ULONG nReadLength, PVOID pReadData, ULONG nW
riteLength,
 PVOID pWriteData );
```

- **pAddr:** Structure with NetId and port number of the ADS server.
- **nIndexGroup:** Index group.
- **nIndexOffset:** Index offset.
- **nReadLength:** Length of data returned by the ADS device in bytes.
- **pReadData:** Buffer with data returned by the ADS device.
- **nWriteLength:** Length of data written to the ADS device in bytes.
- **pWriteData:** Buffer with data being written to the ADS device.

The method 'AdsSyncReadWriteReq()' was also used for linking the variable *'hSwitch'*. The only difference is that *'AdsSyncReadReq()'* (reading of variables) is used instead of *'AdsSyncWriteReq()'* (writing of variables).

```
long AdsSyncReadReq( PAmsAddr  pAddr, ULONG nIndexGroup, ULONG nIndexOffset, ULONG nLength, PVOID pD
ata );
```

- **pAddr:** Structure with NetId and port number of the ADS server.
- **nIndexGroup:** Index group.
- **nIndexOffset:** Index offset.
- **nLength:** Length of the data, in bytes, written to the ADS server.
- **pData:** Pointer to data written to the ADS server.

Lastly, we require two methods for controlling the machine speed. To this end simply change to Design view, double-click on the radio button for 'fast' and *'slow'* and enter the associated code.

Code for *'fast'* radio button.

```
void CMachineDlg::OnBnClickedRadio1()
{
     // Anpassen der "Switch"-
Variable, bei klick auf Radio Button "fast"// Fits the switch variable if the radio button "fast" wa
s clicked
```

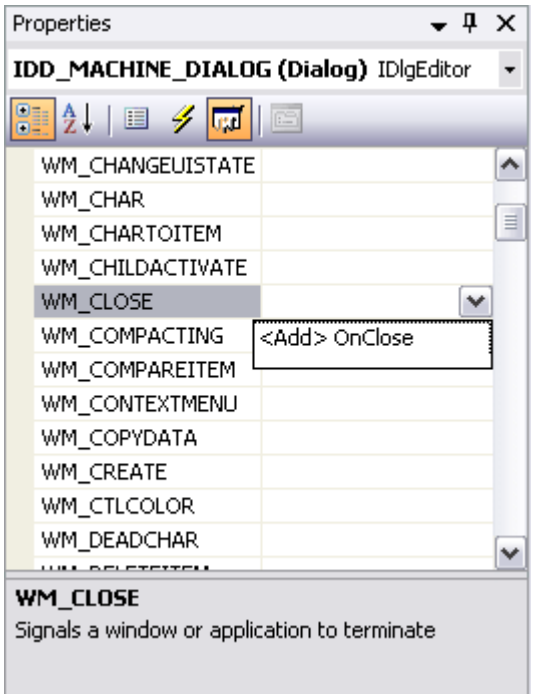
```
     bSwitch = true;
     nSwitch = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hSwitch, 8, "
.switch" );
     nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_VALBYHND, hSwitch, 0x1, &bSwitch );
     nSwitch = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hSwitch );
}
```

Code for 'slow' radio button.

```
void CMachineDlg::OnBnClickedRadio2()
{
     // Anpassen der "Switch"-
Variable, bei klick auf Radio Button "slow"// Fits the switch variable if the radio button "slow" wa
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 );
}
```

Deleting handles:

Finally, delete the handles created during the process. To this end, open the form 'Properties' and switch from '*Properties*' to *'Messages'* in the small icons at the top. Then select the *'WM_CLOSE'* message in the lower window and select *'<Add> OnClose'* in the dropdown menu.



This takes you directly to the point in the code where the following code has to be added.

```
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();
```

The PLC programMachine_Final.proshould start on runtime system 1, and the created C++ programMachine.execan be tested.

Download C++ program example:

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

7.4.5 Example: machine with Microsoft Expression Blend

Microsoft Expression Blend is a program for creating program interfaces for C# and Visual Basic. In this example an interface created with the program is linked with the Machine example. The programming language C# was used.

Required software:

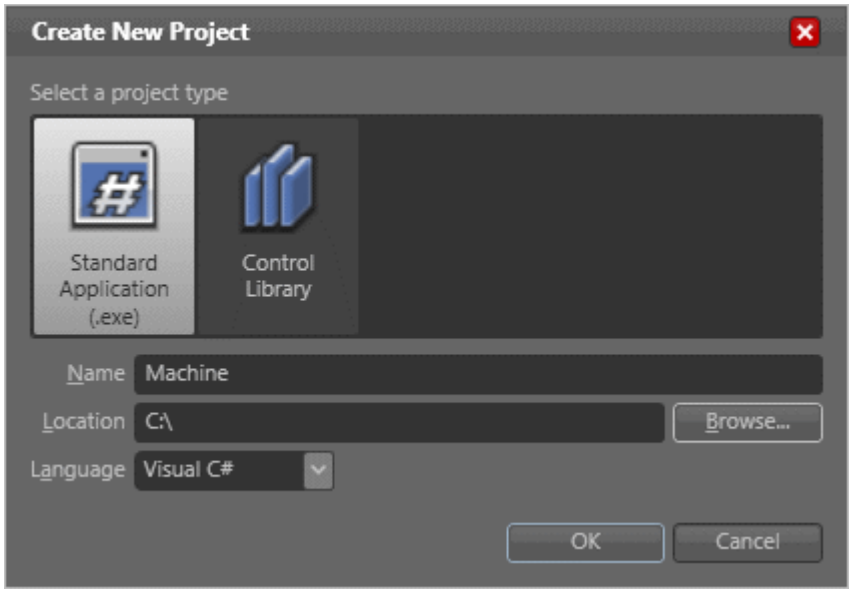
- Microsoft .NET Framework Version 3.0, for further information please refer to www.microsoft.com
- Microsoft Expression Blend, for further information please refer to www.microsoft.com
- Microsoft Visual Studio 2005
- TwinCAT 2.10

First steps ...

Program development with Microsoft Expression Blend and integration of the TwinCAT ADS .NET component is described based on an example.

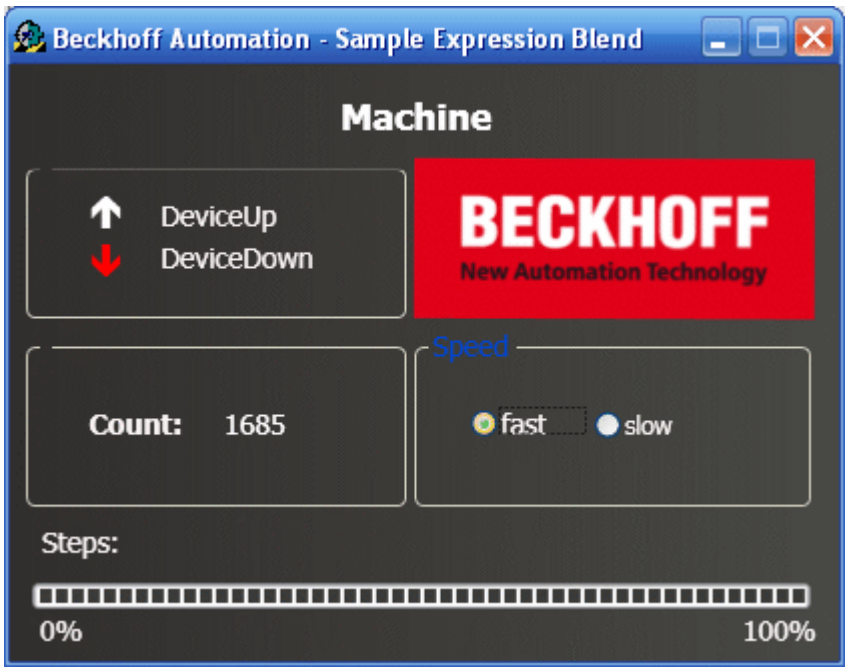
1. Create new project:

Start Expression Blend and create a new interface via Menu -> File -> New Project... . The 'Create New Application' dialog box opens and the type, name, location and programming language can be selected. In this case please select 'Standard Application' as the type, 'Machine' as the name and 'C#' as the programming language.



2. Creating a user interface

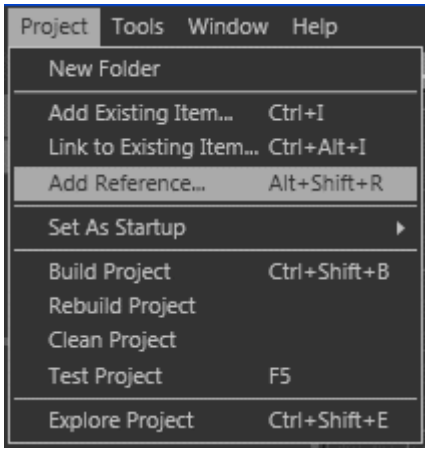
The interface settings are stored in file Window1.xaml.



In the upper left you see the two outputs that are also output to the Bus Terminals. The bottom left shows the variable for counting the workpieces. The cycle speed of the motor can be changed via the 'Speed' field on the right. The 'Steps' display shows the number of cycles that are output on output 1.

3. Adding a reference

Once the interface has been created, add a reference called TwinCAT.Ads.dll via the menu --> Project --> Add Reference.



4. Edit source text

Once the interface has been created and the TwinCAT ADS component has been integrated, you can change to the C# source text, which can be displayed in Visual Studio. The required namespaces 'System.IO' and 'TwinCAT.Ads' are added in the top row of the source text.

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

This is followed by the declarations.

```
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;
```

The first method is the load method. It is used to generate instances of different classes and create a link to port 801.

```
//-----------------------------------------------------//
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");
    }

    //...
```

The variables in the Load method are then linked and linked with a method (see "Write method 'tcClient_OnNotification'"), which is called when a variable changes.

```
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.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);

    // 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);
    }
}
```

Linking PLC variables:

The method AddDeviceNotification was used for linking the variables.

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

- **variableName:** Name of the PLC variable.
- **dataStream:** Data stream receiving the data.
- **offset:** Interval in the data stream.
- **length:** Length in the data stream.
- **transMode:** Event if the variable changes.
- **cycletime:** Time (in ms) after which the PLC server checks whether the variable has changed.
- **maxDelay:** Latest time (in ms) after which the event has finished.

- **userData:** Object that can be used for storing certain data.

The method CreateVariableHandle was used for linking the variable hSwitchWrite.

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

- **variableName:** Name of the PLC variable.

Write method 'tcClient_OnNotification':

The method is called if one of the PLC variables has changed.

```
//----------------------------------------------//
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 changedif(e.NotificationHandle == hDeviceUp)
    {
        //
Adapting the colours of the diagrams to match the variablesif (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 buttonif (binReader.ReadBoolean() == true)
        {
        optSpeedFast.IsChecked = true;
        }
        else
        {
        optSpeedSlow.IsChecked = true;
        }
    }
    }
    catch (Exception ex)
    {
    MessageBox.Show(ex.Message);
    }
}
```

Lastly, we require two methods for making the machine faster or slower. They are used to switch a virtual switch by writing a value to the PLC variable 'switch'.

```
//-------------------------------------------------------//
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);
    }
}
```

To ensure that the methods are called for the right event, select DocumentRoot in the Design view, click add under Events and select loaded, and select the Load method.

The same applies for the two radio button, although in this case select the event 'Click' and the method 'optSpeedFast_Click' or 'optSpeedSlow_Click_Click'.

Delete notifications and handles:

In the Close event of the window the links are activated again with the method DeleteDeviceNotification().

```
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();
}
```

The PLC program Machine_Final.pro should start on runtime system 1, and the created C# program *Machine.exe* can be tested.

Download:

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

7.4.6 Example: Machine with Microsoft Expression Blend in Microsoft Windows Vista Media Center

Microsoft Expression Blend is a program for creating program interfaces for C# and Visual Basic. In this example an interface created with the program is linked with the Machine example and subsequently integrated in the Microsoft Windows Vista Media Center. The programming language C# was used.

Target platform:

-Windows Vista

Implementation

-Visual C#

Required software:

- Microsoft .NET Framework Version 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 (November 2006 CTP)
- TwinCAT 2.10
- Notepad or other text editor

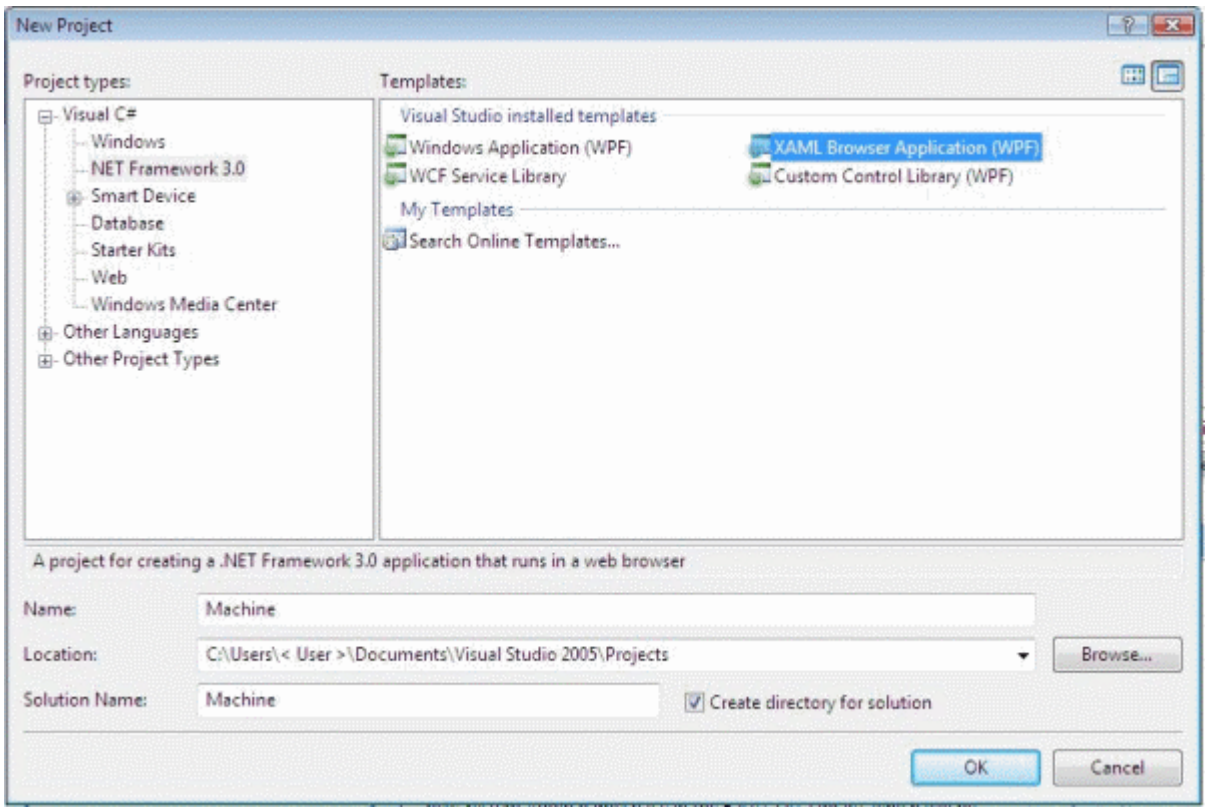
First steps ...

Step by step familiarisation with the development of a program with Microsoft Visual Studio and Microsoft Expression Blend, integration of the TwinCAT ADS .NET component based on an example, and integration in the Microsoft Windows Vista Media Center.

1. Create new project:

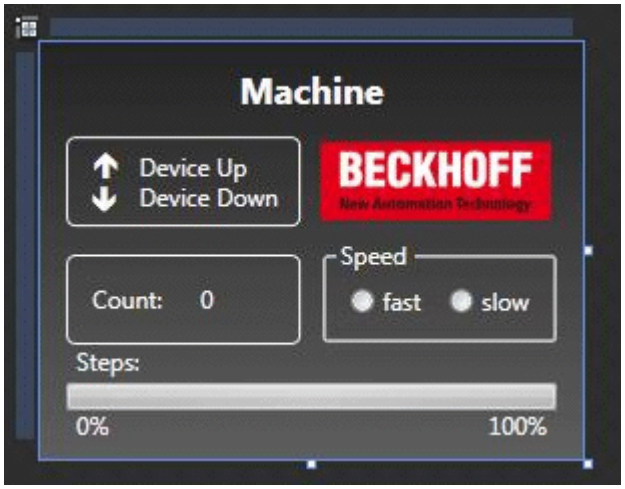
Start Microsoft Visual Studio and create new XAML browser application. Proceed via the menu 'File -> New -> Project...' . The dialog box 'New Project' opens.

First select the project type: 'Project types -> Visual C# -> Net Framework 3.0'. The project type templates appear on the right. Select 'XAML Browser Application'. Enter a name for your project (in this case Machine) and specify the location.



2. Creating a user interface

Now change to Microsoft Expression Blend and open the project you just created in order to create the user interface.

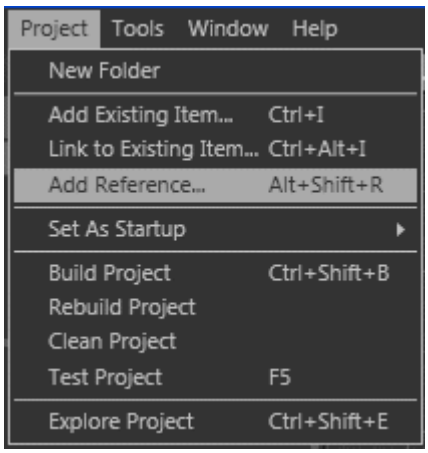


In the upper left you see the two outputs that are also output to the Bus Terminals. The bottom left shows the variable for counting the workpieces. The cycle speed of the motor can be changed via the 'Speed' field on the right. The 'Steps' display shows the number of cycles that are output on output 1.

To make the user interfaces constantly adapt their size, copy the upper grid and insert a view box instead of the grid. Now insert the grid into the view box. Now set the size of the page, the view box and the grid to 'Auto'. This may result in shifting of elements, in which case you have to reposition them. Please ensure that the size of the page, the view box and the grid is not inadvertently reset to "fixed".

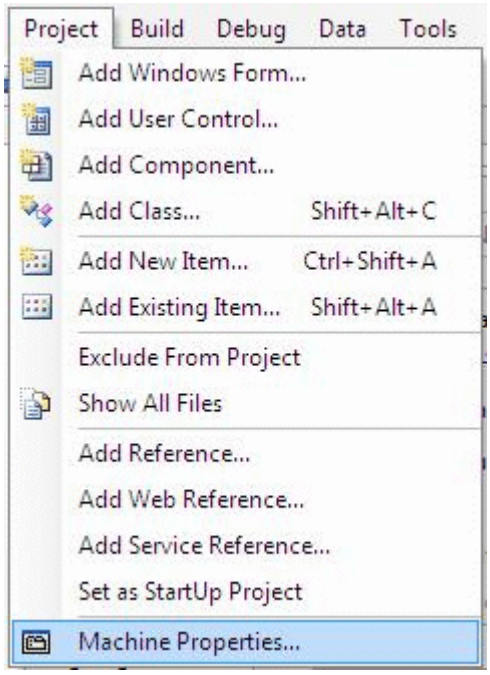
3. Adding a reference

Once the interface has been created, add a reference called 'TwinCAT.Ads.dll'. This can be done in Visual Studio or Expression Blend. In both cases proceed via the menu 'Project --> Add Reference'.

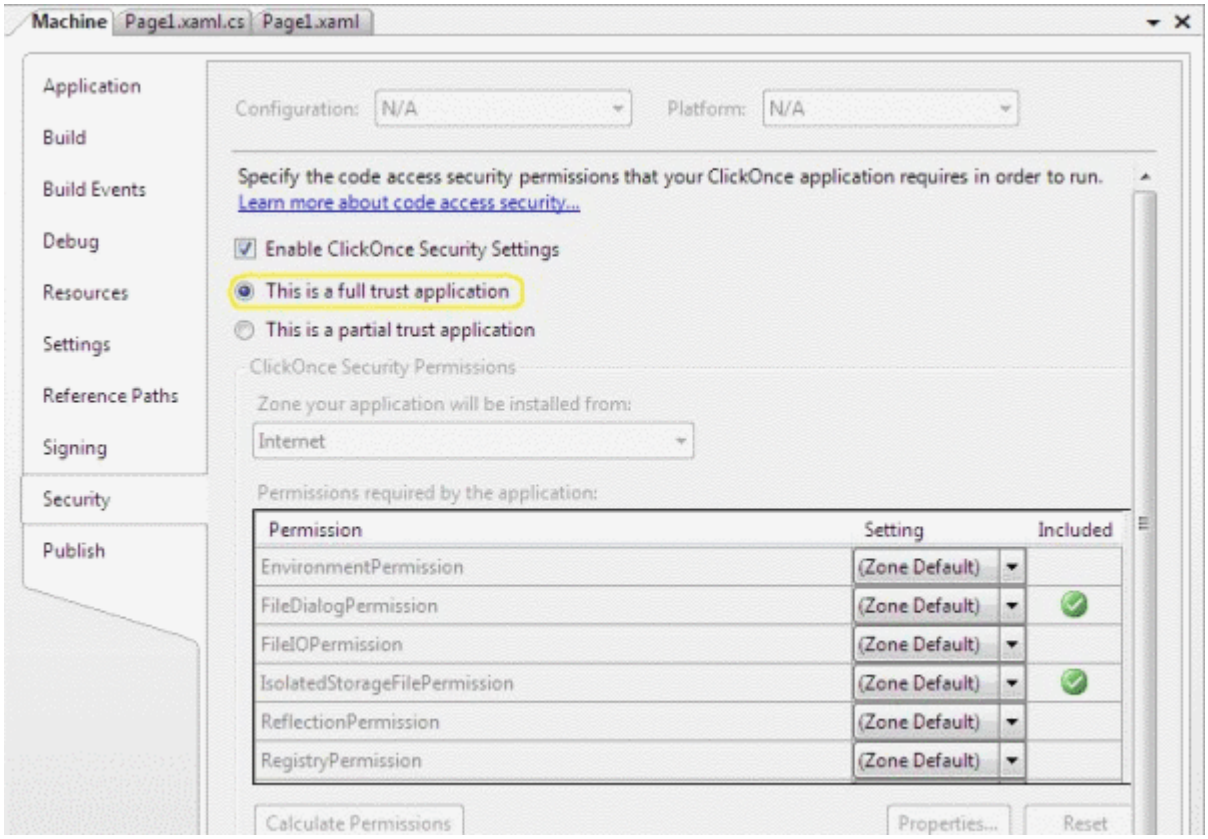


4. Security activation

Select 'Project -> <Project Name> Properties....' from the menu . .



A tab opens, in which you can specify the project properties. Select 'Security' and then 'this is a full trust application'.



5. Edit source text

You can now start creating the source text in C#.
The required namespaces 'System.IO' and 'TwinCAT.Ads' are added in the top row of the source text.

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

The next step involves the declarations.

```
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;
```

The first method is the 'Load' method. It is used to generate instances of different classes and create a link to port 801.

```
//------------------------------------------------------// 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 cl
ass
     dataStream = new AdsStream(7);

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

     // Eine neue Instanz der Klasse TcAdsClient erzeugen// Create an new instance of the TcAdsClien
t 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");
    }

    //...
```

The variables in the 'Load' method are then linked and linked with a method (that still has to be written), which is called when a variable changes.

```
try
    {
    // Initialisieren der Überwachung der SPS-
Variablen// Initalizing 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. Definition

Linking PLC variables:
The method AddDeviceNotification was used for linking the variables.

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

- **variableName:** Name of the PLC variable.
- **dataStream:** Data stream receiving the data.
- **offset:** Interval in the data stream.
- **length:** Length in the data stream.
- **transMode:** Event if the variable changes.
- **cycletime:** Time (in ms) after which the PLC server checks whether the variable has changed.
- **maxDelay:** Latest time (in ms) after which the event has finished.
- **userData:** Object that can be used for storing certain data.

The method CreateVariableHandle was used for linking the variable 'hSwitchWrite'.

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

- **variableName:** Name of the PLC variable.

7. Write method:

A method that does not exist yet was referred to above. This method ('tcClient_OnNotification') is written next. The method is called if one of the PLC variables has changed.

```
//-----------------------------------------------// 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);
    }
}
```

Lastly, we require two methods for setting the machine to fast or slow. They are used to switch a virtual switch by writing a value to the PLC variable 'switch'.

```
//-----------------------------------------------------// 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. Delete notifications and handles:

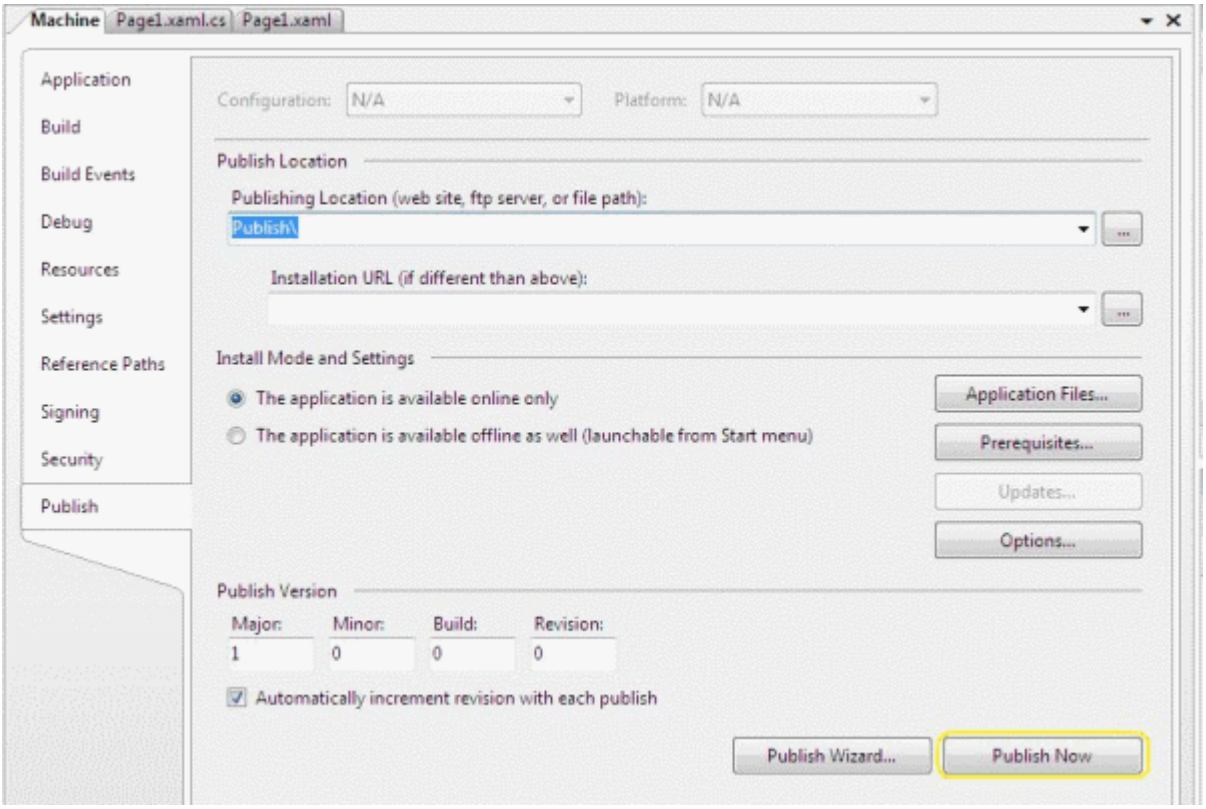
In the Close event of the window the links are activated again with the method DeleteDeviceNotification().

```
//-------------------------------------------------------// 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();
}
```

Last but not least we need to ensure that the methods are called for the right event. To this end open Expression Blend, select Page, change to Events in Properties and enter 'Load' under 'Loaded' and 'Close' under 'Unloaded'. Proceed accordingly for the two radio buttons, except that the 'Click' event should be selected and the method 'optSpeedFast_Click' or 'optSpeedSlow_Click'.
The Machine_Final.pro PLC machine program must run in runtime system 1, and the program can be tested in Internet Explorer 7.

9. Integration in Vista Media Center

Once you have tested your project sufficiently and no errors were detected, you can integrate it in the Microsoft Windows Vista Media Center.
In Visual Studio call up the project properties again, but this time select 'Publish'. Click on 'Publish Now'. This creates an xbap file which you subsequently call up in the Media Center. This step is always required whenever the program has been modified and the change is to be transferred to the Media Center.



Now open a text editor (e.g. Notepad) and enter:

```
<application
    URL = "C:
\Users\<User>\Documents\Visual Studio 2005\Projects\Machine\Machine\Publish\Machine.xbap">
</application>
```

Save this under: 'C:\Users\<User>\AppData\Roaming\Media Center Programs\Machine.mcl'. If you now start your Media Center you will find your program under 'Online Media -> program library -> programs by name -> Machine'.
This is the simplest form of integration in the Microsoft Windows Vista Media Center. Further information on integration in the Media Center can be found here.

10. Download Expression Blend example:

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

7.4.7 Sample Machine with Microsoft Silverlight and JavaScript

Microsoft Silverlight is a web-presentation-technology.

Target platform

-Windows XP, XPE, WES

-Windows Vista

-Windows 7

Implementation

-JavaScript

Required Software:

- runtime:
- Microsoft Silverlight 1.0 / 1.1
- developer tools:
- Microsoft Visual Studio 2008 Beta 2
- Microsoft Silverlight Tools Alpha Refresh for Visual Studio (July 2007)
or

- Microsoft Visual Studio 2005
- Microsoft Silverlight 1.0 Software Development Kit
For this sample Microsoft Visual Studio 2005 was used.

- designer tools:
- Expression Blend 2

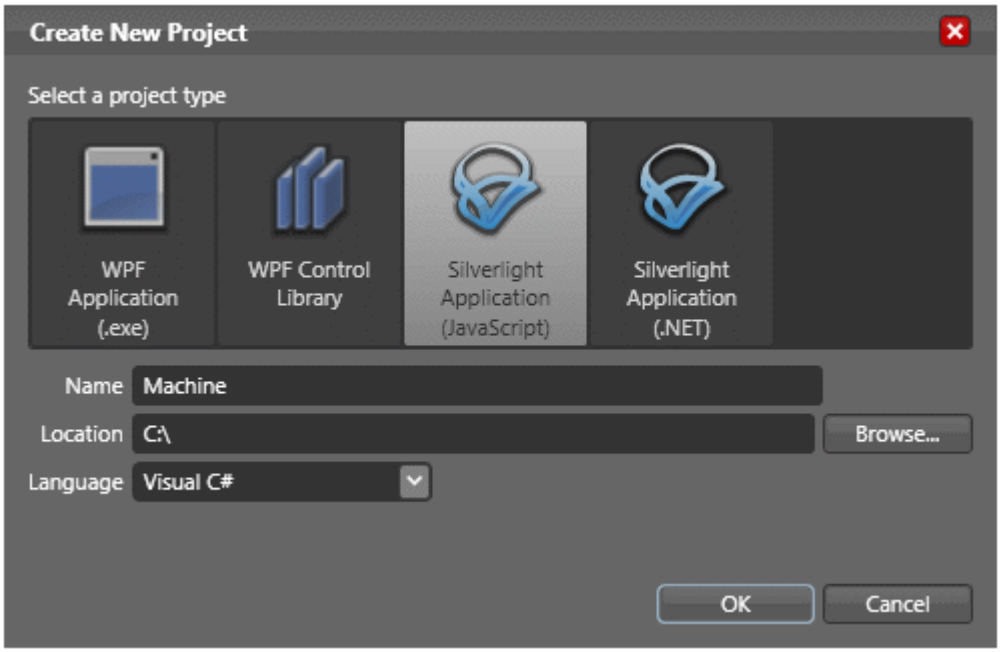
- others:
- TwinCAT 2.10
- Browser (for example Internet Explorer 7 or Mozilla Firefox 2)
- Microsoft .NET Framework Version 3.0

The first steps...

Step for step you learn to develop a Silverlight-Application and include the TwinCAT ADS Web Service on a sample.

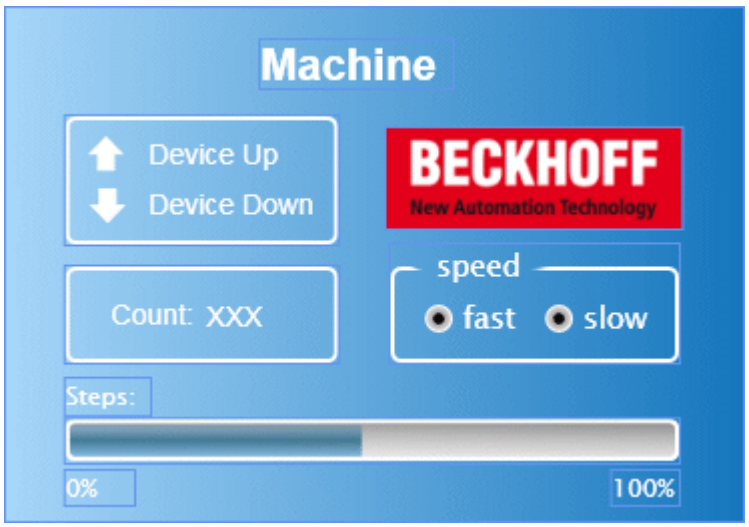
1. Create new Project:

Start the Microsoft Expression Blend 2 and create a new user interface over 'Menu->File -> new Project...' . If the Dialog 'Create New Project' open, now you can edit the type, the name, the location and the language. In this sample choose the type 'Silverlight Application (JavaScript)' and the name 'Machine'.



2. Create user interface:

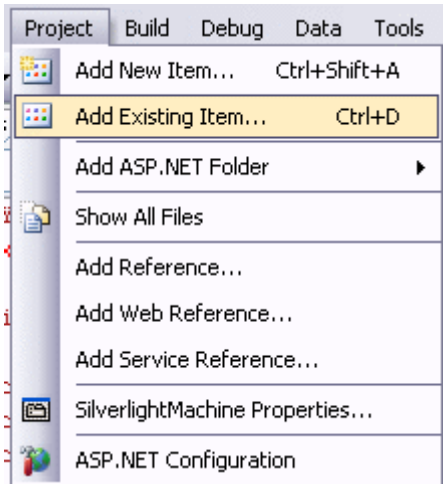
A few controls are available for the development of the user interfaces in Silverlight applications. You can create a *RadioButton* with a *Textbox* and two *Ellipses,* whitch are grouped into a *Canvas*. The *Progressbar* can be created with three *Rectangles* which are grouped into a separate *Canvas* again.
The properties of the interface are described in the *Page.xaml* file.
Note that you can not set the object size to 'Auto'. This may lead to errors later.



You can see the two outputs in the upper left corner. The variable, which counts the work pieces, is placed below. You can change the engine speed with the two RadioButtons on the right side. The displayed steps are equivalent to the number of clocks that are returned to output 1.

3. Add XMLHTTP.JS

In Visual Studio over 'Project Ã¢â€ â€™ Add Existing Item...' add the file 'XMLHTTP.JS'. This file contains general methods for reading and writing of PLC variables and converting data types.



4. Edit source code

Open the file 'Default.html' in Visual Studio and include the 'XMLHTTP.JS' file into the header.

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

Add a JavaScript-range in the HEAD-field of the HTML page. You find the following source code in it:

The global variables has to be declared first.

```
<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;

...
```

You have to adapt the url of the TcAdsWebService, the netID and the port.

The GUI objects can not be accessed directly. In order to do this the objects will be assigned to already declared variables after the start of the application.

```
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");
}
```

The loading method contains the assignation of the variables. For executing the Load method *Loaded="Load"* has to be added to the *Canvas* on top of the Page.xaml.

```
<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);
}
```

Read PLC variable

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

- **netId:** String indicating the AMS-Net-Id of the PLC
- **nPort:** Port-number of the runtime-system
- **indexGroup:** IndexGroup of the variables to read
- **indexOffset:** First byte to read
- **ncbLen:** Number of bytes to read

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

Add *onload="init()"* into *<body>*, so that the method will be executed during loading.

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

The next function reads values and displays them. It is important that the address of the variable in Machine.pro is correctly.

```
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
    }
}
```

With the last two methods the PLC variable, which controls the speed of the machine, is set to zero or one.

```
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');
}
```

Write SPS Variables

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

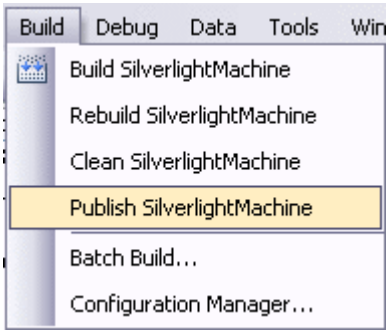
- **netId:** String indicating the AMS.Net.Id of the PLC
- **nPort:** Port-number of the runtime-system
- **indexGroup:** IndexGroup of the variable
- **indexOffset:** First byte to write to
- **ncbLen:** Number of bytes to write
- **pwrData:** Array containing the data to write
- **type:** "bool", "int" or "string"

It has to be changed to 'Expression Blend 2' as it has been done with the 'Load' function for declaring the according rows as 'Click' event for both buttons.

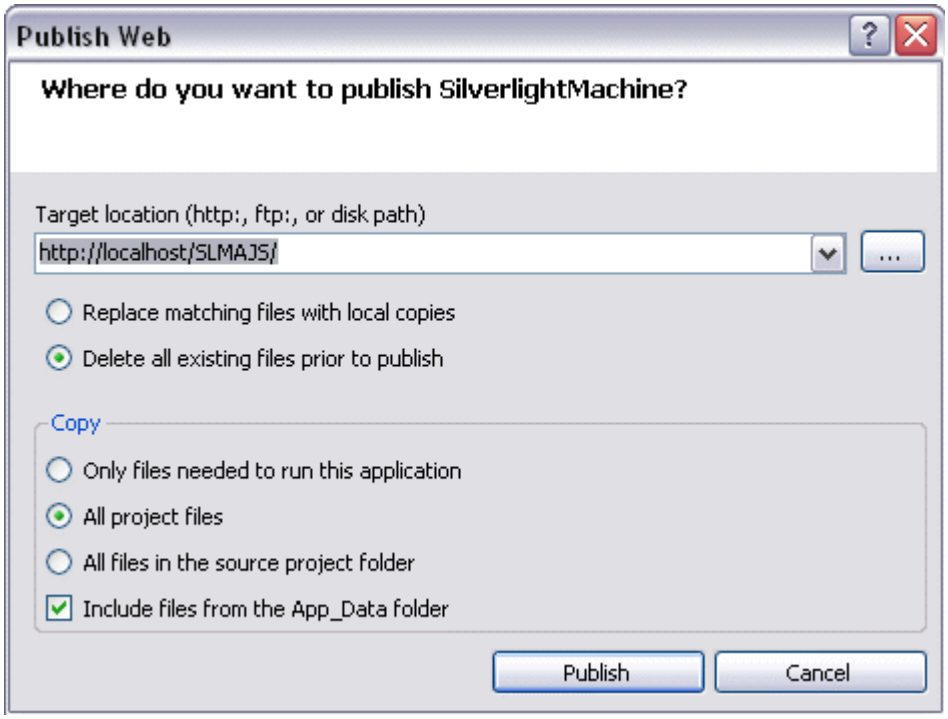
```
<Canvas x:Name="canvasFast" MouseLeftButtonDown="Fast_MouseLeftButtonDown" ...
<Canvas x:Name="canvasSlow" MouseLeftButtonDown="Slow_MouseLeftButtonDown" ...
```

5. Testing:

By debugging the application you will recognize that it will not work as expected. To prevent this got to 'Build -> Publish'.



Choose the 'Target location' in the dialog window and then 'All project files' under 'Copy'.



If the statusbar says 'Publish succeded', you can run and test the application in the browser.

Download:

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

7.4.8 Example for a machine with Microsoft Silverlight for Windows Embedded

A new item in Windows Embedded CE 6.0 R3 is Silverlight for Windows Embedded. With this new technology, user interfaces of CE devices can now be written in XAML and designed with tools such as Microsoft Expression Blend. On the basis of the machine example, the creation of a Silverlight for Windows Embedded application with integration of the ADS components is described here.

Target platform

- Windows CE 6 R3

Implementation

- C++

Required software

- Microsoft Visual Studio 2008
- Microsoft Expression Blend 2 SP1
- TwinCAT 2.11
- Beckhoff HMI 600 SDK

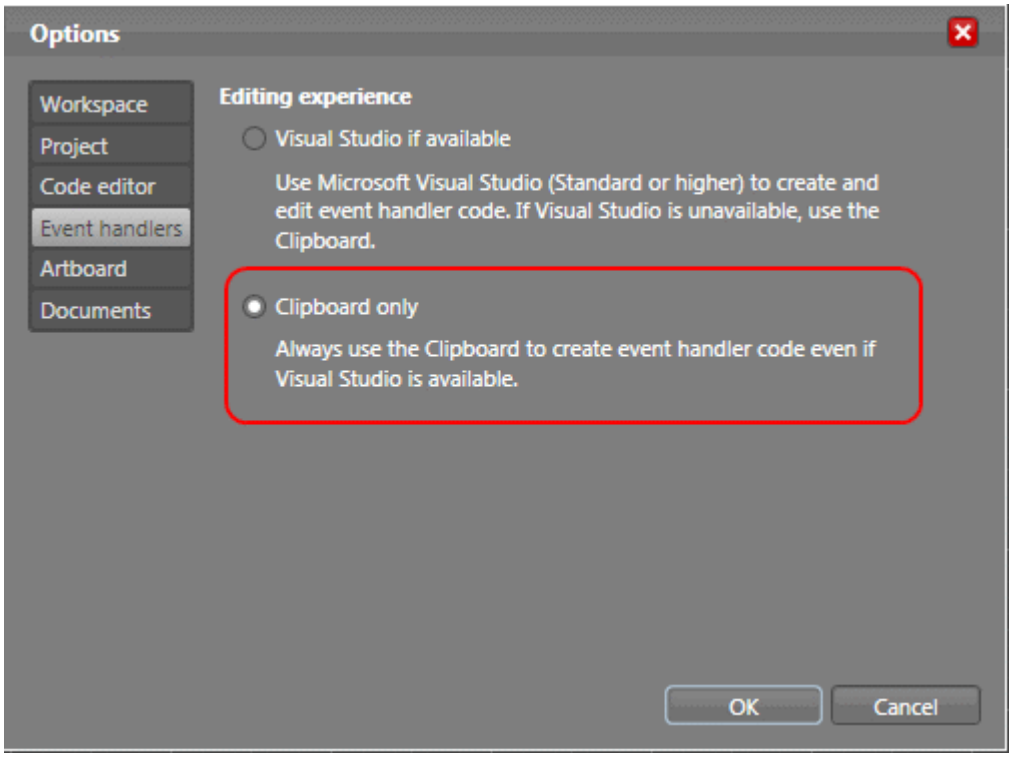
Required hardware

- Windows CE 6.0 R3 device (e.g. CX1020)

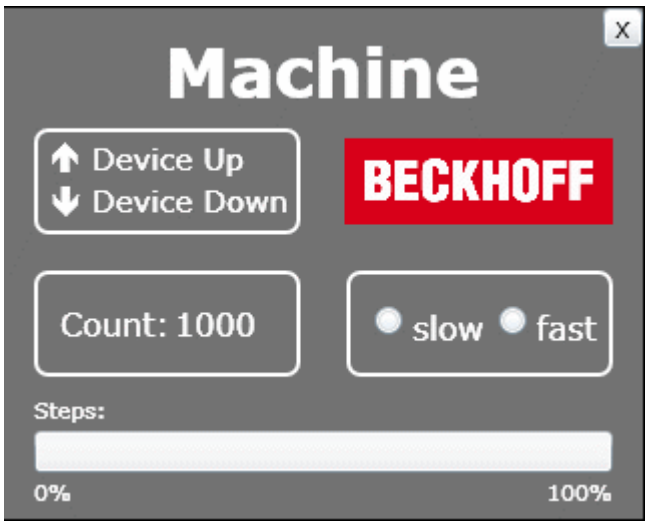
First steps...

1. Creating a new Silverlight 2 project:

The design of a Silverlight for Windows Embedded application is described in XAML. To this end a Silverlight 2 project is created with Microsoft Expression Blend 2 SP1 via *'File - > New Project'*. The Visual Studio Solution thereby created is not needed in this sample. In addition, the selection of the programming language (Visual C# or Visual Basic) can be ignored. Silverlight for Windows Embedded supports only Visual C++, which is not integrated in Expression Blend. It is therefore also not possible to use the source code generated by this tool. Disable Visual Studio integration in Expression Blend to avoid the unnecessary automatic generation of Visual C# and Visual Basic code. To do this, select: *'Options->Event handlers' 'Clipboard only'.*



2. Creating a user interface
The user interface can now be created in Expression Blend.



In the upper left area, the two outputs can be seen that are also output to the bus terminals. The bottom left shows the variable for counting the workpieces. The speed can be set on the right. The 'Steps' display corresponds to the number of cycles. In addition, a button for ending the program is created at the top right.

```
<UserControlxmlns="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"
```

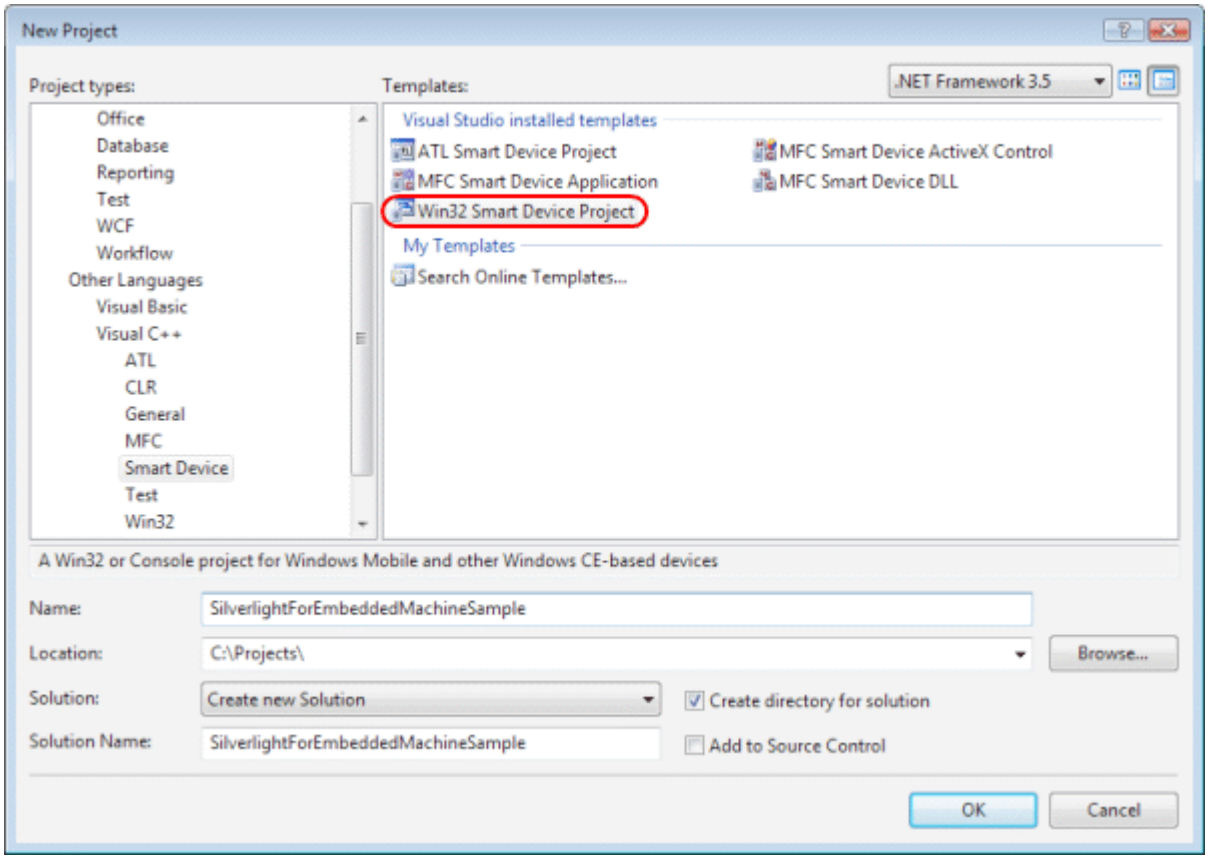
```
        Height="53" Width="132.532"><RectangleFill="{x:Null}" Stroke="#FFFFFFFF" StrokeThickness="2"

            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. Creating a new Win32 Smart Device project

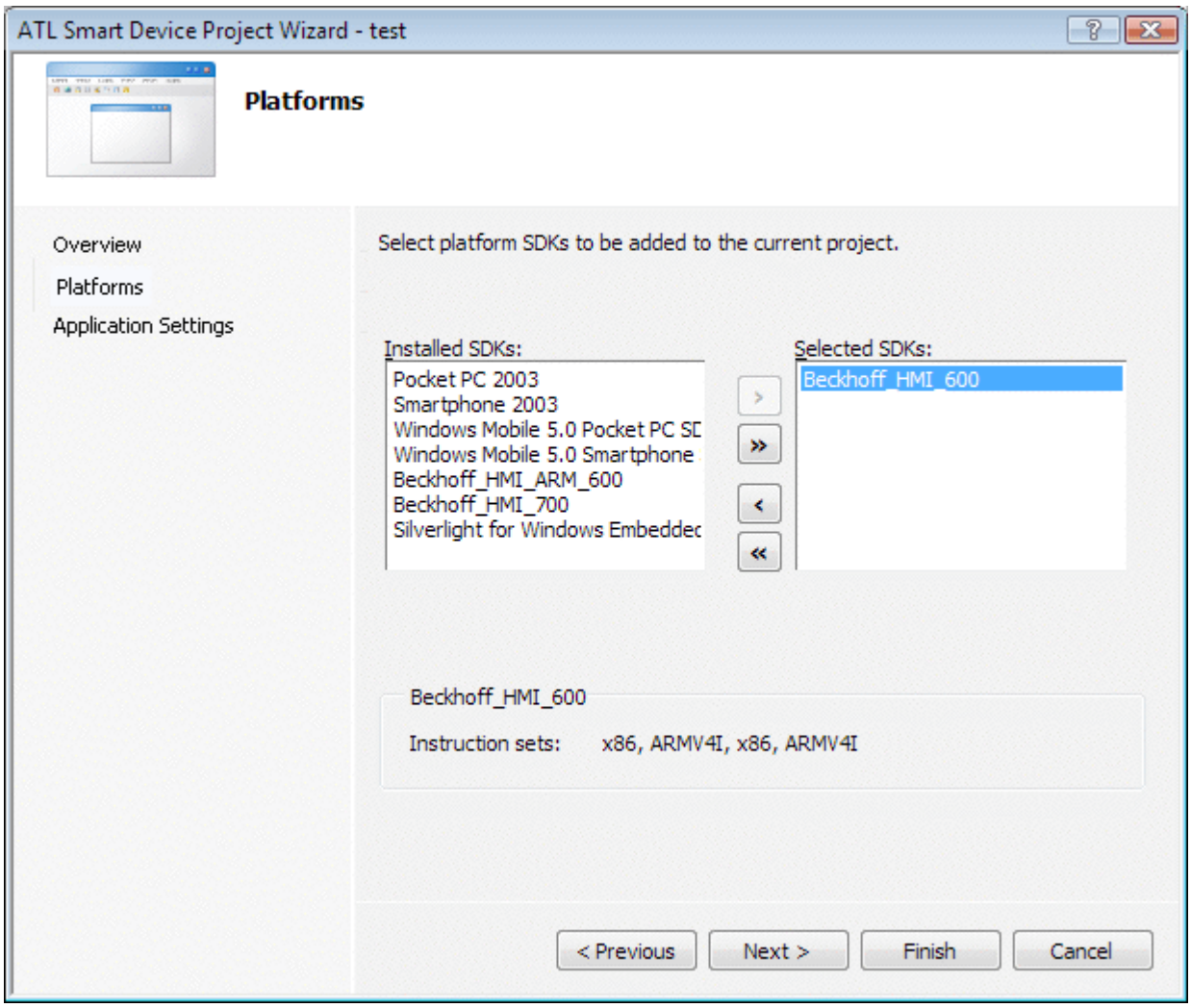
A new Win32 Smart Device project must now be created in Visual Studio 2008.

Beckhoff HMI 600 SDK installed?

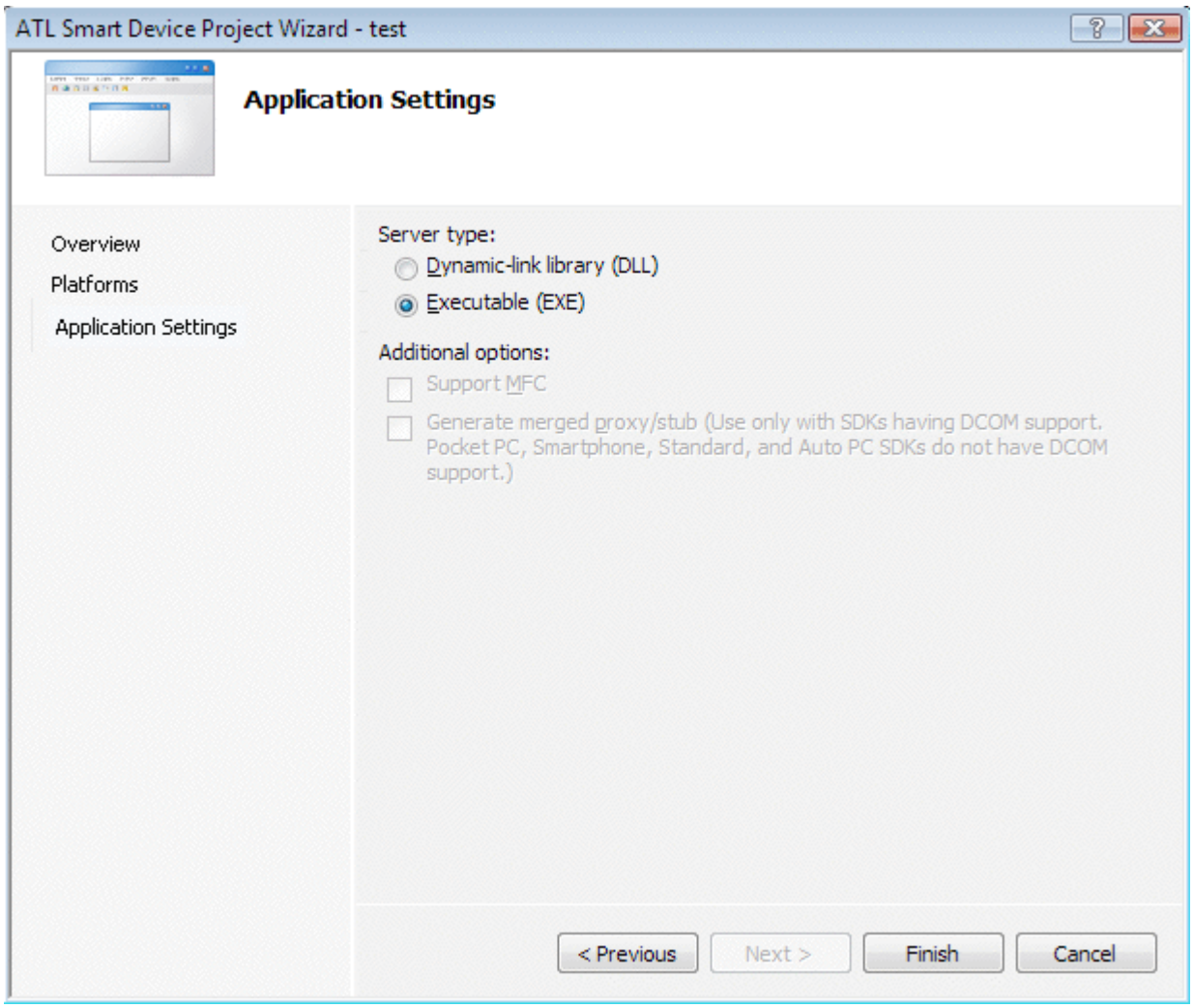
If the Beckhoff HMI 600 SDK is not yet installed on the computer, install it before creating a new Visual Studio project.



The Platform SDK of this project is the Beckhoff HMI 600 SDK.

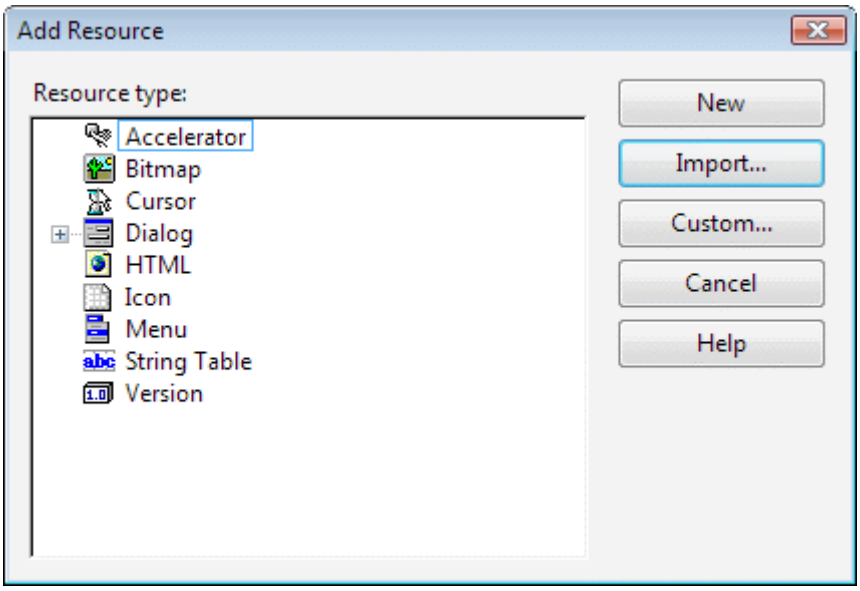


Executable (EXE) is selected as the server type.

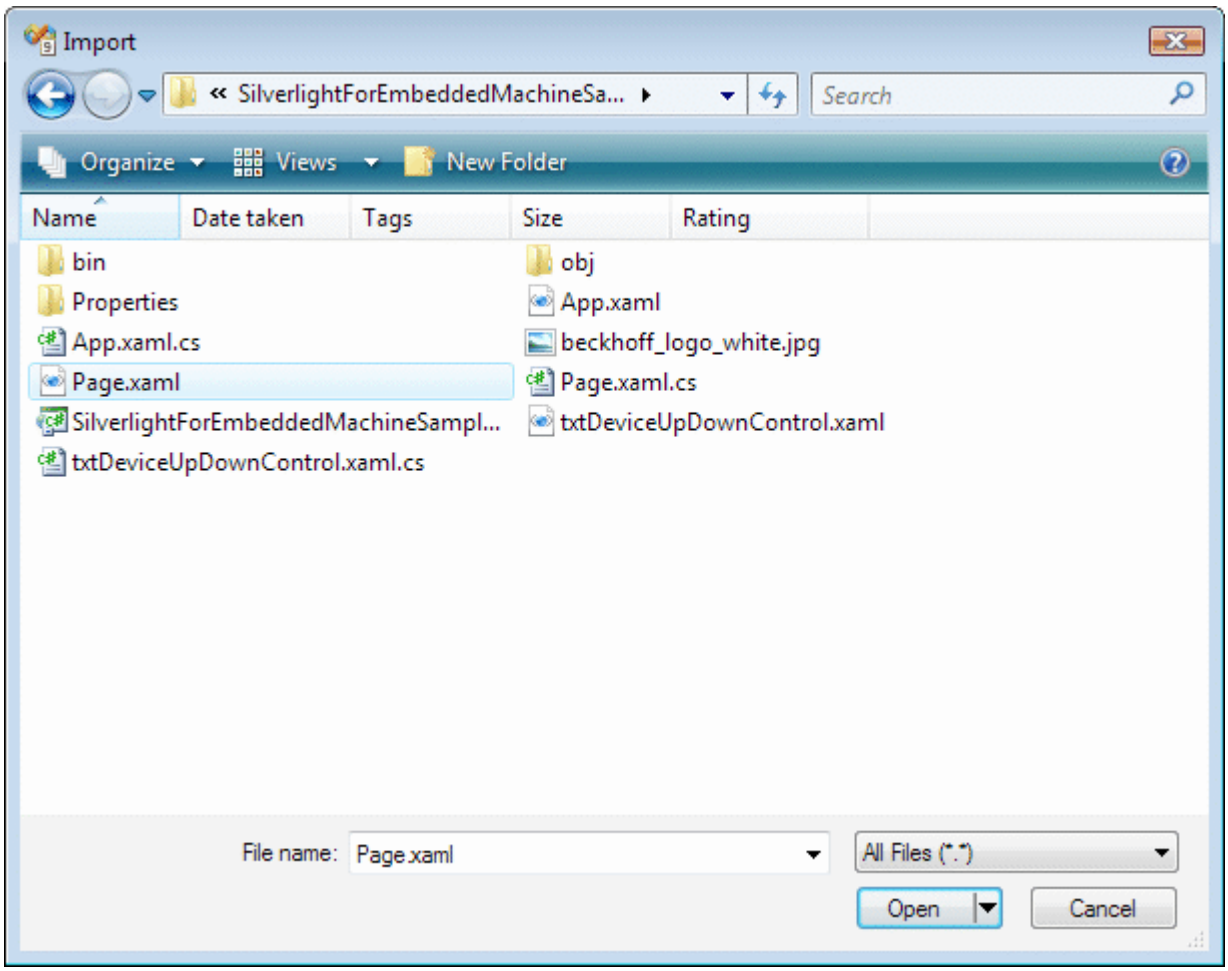


4. Integrating the XAML file as a resource

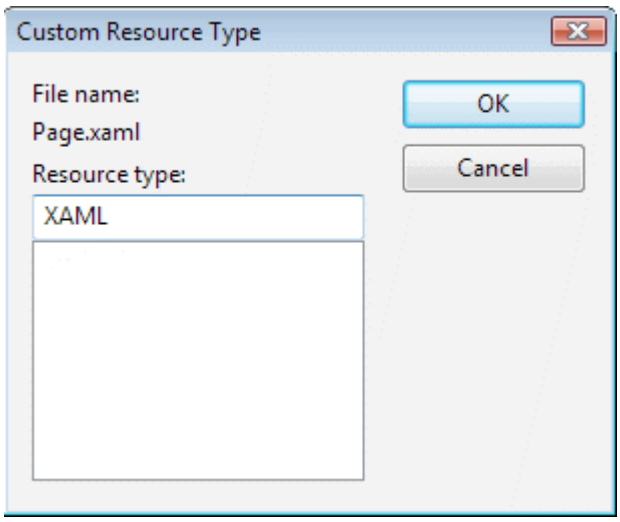
The user interface designed with Expression Blend can be integrated in the new project. To do this, open the resource file (.rc). A right mouse click on the resource in the Resource View tab and the selection of *'Add - > Resource…'* opens a dialog box via which the XAML file can be integrated.



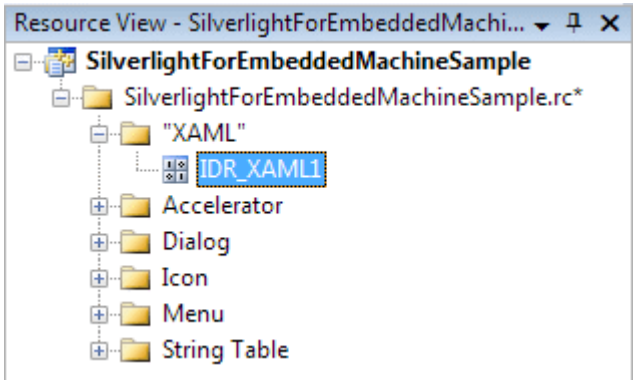
The XAML file can be imported into the project with the aid of the dialog.



Specify XAML as the resource type.



The standard resource ID (IDR_XAML1) can be retained in this example. In your own projects, however, it makes sense to rename them.



5. Creating an AdsHelper Class

The largest part of the Ads communication can be stored in a separate class, called AdsHelper here.

AdsHelper.h

The TcAds headers are integrated in the AdsHelper headers. Care must be taken to specify the correct path. In addition, the headers depend on the processor type. In the sample code the X86 headers are attached.

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

Furthermore, the declarations are made in the header.

```
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

The headers StdAfx.h and AdsHelper.h must be integrated in the AdsHelper.cpp,

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

and afterwards the global variables defined.

```
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(pServerAddrm, 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 =
    AdsSyncReadWriteReyEx2(port, pServerAddr, ADSIGRP_SYM_HNDBYNAME, 0x0,
    sizeof(long), pHandle, strlen(szVarname), (char*)szVarname, &read);

    return nErr;
}
```

Handles for PLC variables are released again with AdsFreeVarHandle and AdsFreeVarHandleEx.

```
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(pHandle), &pHandle);
}
```

The following two methods write the PLC variable ".switch" and thereby set the speed to slow or fast.

```
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;
}
```

In the connect method, a connection to the variables is created in the PLC.

```
long connect (long port, AmsAddr* addr, 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;
}
```

When disconnecting the Ads connection, the handles of the PLC variables must be released and the port closed.

```
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. Processing the source code

The headers are integrated first in the SilverlightForEmbeddedMachineSample.cpp file.

```
#include"pwinuser.h"#include"xamlruntime.h"#include"xrdelegate.h"#include"xrptr.h"    #include"resou
rce.h"
```

The declaration of the variables follows.

```
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);
```

The timer callback function checks the Ads connection and restores the connection if necessary.

```
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;
}
```

The following Ads Event handler is called if a PLC variable to which a link exists changes.

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

The corresponding arrows are colored red or not, depending on whether deviceUp, deviceDown or engine is set to TRUE.

This effect can be improved still further by the use of timelines.

```
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 indicates the number of cycles. The value is output via the progress bar *prgSteps*. For this the data must first be converted to a byte, since the associated PLC variable is of the type byte. Since the progress bar can only transfer data of the type float, conversion to the type float subsequently takes place.

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

In the case of *count*, as with steps, the data must first be converted to their original data type before they can be converted into a text and transferred to the text block *txtCount*

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

The output of the speed type is done via RadioButtons. The appropriate radio button is marked depending on the speed.

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

The OnClick event is triggered by the various instances and the name can be used to distinguish which instance was the trigger.

```
class BtnEventHandler
{
public:

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

    short state = 0;

    long adserror = 0;

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

    if (adserror != 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.
        adserror = connect(port, &addr, Callback);

        Sleep(1000);
    }

    SysFreeString(name);
    return S_OK;
    }
};
```

In the WinMain method, the XMAL runtime must be initialized first. If XamlRuntimeInitialize is successful, then Silverlight for Windows Embedded Runtime is started in the application.

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

Each Silverlight for Windows Embedded application has a single "application" object that can be used to access global properties.
To access this object the GetXRApplicationInstance API is used.

```
    HRESULT retcode;

    // Load an dinit XAML resource
    IXRApplicationPtr app;

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

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

After the initialization of the application object, the main window can be created and the administration of the object can be handed over to Silverlight for Windows Embedded.

```
    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;
```

The object within a Silverlight for Windows Embedded application is organized in object trees. In order to access this object, a pointer to the root element is required.

```
    IXRFrameworkElementPtr root;

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

Creating instances of the controls and timelines.

```
// Get controls by nameif (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 nameif (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;
```

Creating the "RadioButtonGroup" and assigning the two radio buttons from this group.

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

A redirecting object is needed to link the EventHandler with the buttons.

```
    BtnEventHandler handler;

    // Set the event handler for the buttonsif (FAILED(retcode=CreateDelegate(&handler, &BtnEventHan
```

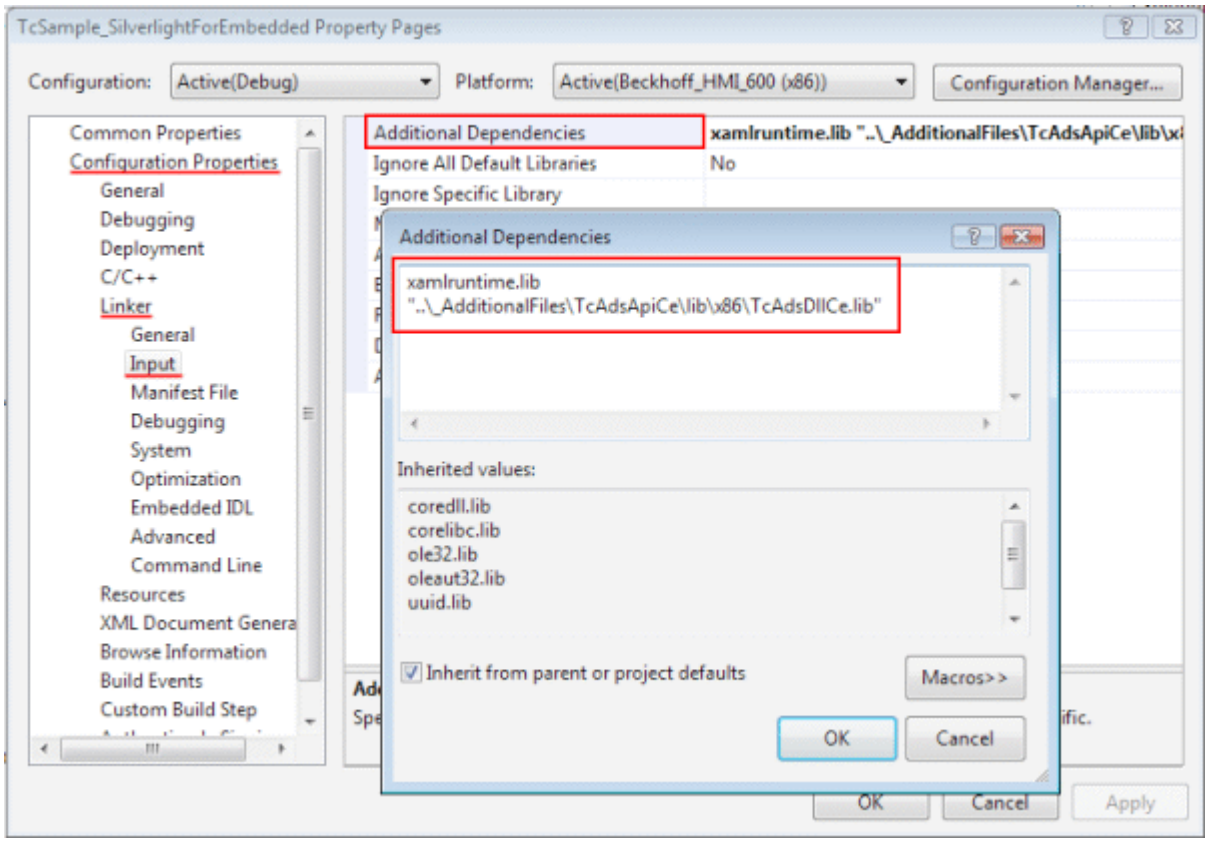
```
dler::OnClick, &clickdelegate)))
    return -1;

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

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

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

Integrating the Ads components.

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

    AdsGetLocalAddressEx(port, &addr);

    // connect to the PLC and register callbacks
    addr.port = 0x321;
    adserror = 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. Properties

A connection to *xamlruntime.lib* and *TcAdsDllCe.lib* must be made in the project properties.



Download Silverlight for Windows Embedded sample

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

Replacing X86 version with ARM

In the sample the X86 version of the TcAdsDllCe.lib is used. To build the sample for ARM devices, this library must be exchanged for the corresponding ARM version beforehand.

More Information:
www.beckhoff.com/automation

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