

Handbuch | DE

TwinCAT 2

Quick Start



TwinCAT 2 | System

Inhaltsverzeichnis

1	Vorwort.....	5
1.1	Hinweise zur Dokumentation	5
1.2	Zu Ihrer Sicherheit.....	6
1.3	Hinweise zur Informationssicherheit	7
2	Einführung in TwinCAT und Systemvoraussetzung.....	8
3	Installation	9
3.1	Installationsprogramm starten	9
3.2	Ende der Installation	17
4	TwinCAT PLC Control.....	21
4.1	SPS-Standard IEC 61131- 3	22
5	TwinCAT System Manager	25
6	TwinCAT Scope View.....	26
7	Beispielprogramm.....	27
7.1	Beispiel Machine.pro	27
7.2	Programmablauf verfolgen.....	35
7.3	Umsetzung des Beispielprogramms	40
7.3.1	Deklaration der Variablen.....	40
7.3.2	Lightbus - Einrichten der Busklemmen	41
7.3.3	Ethernet - Einrichten der Busklemmen	51
7.3.4	EtherCAT - Einrichten der Busklemmen	61
7.4	Beispiele Machine mit	73
7.4.1	Beispiel Maschine mit Microsoft Visual C#	73
7.4.2	Beispiel Maschine mit Microsoft Visual Basic .NET	79
7.4.3	Beispiel Maschine mit Microsoft Visual Basic 6.0	84
7.4.4	Beispiel Maschine mit Microsoft Visual C++ .NET	87
7.4.5	Beispiel Maschine mit Microsoft Expression Blend.....	96
7.4.6	Beispiel Maschine mit Microsoft Expression Blend in den Microsoft Windows Vista Media Center	101
7.4.7	Beispiel Maschine mit Microsoft Silverlight und JavaScript.....	108
7.4.8	Beispiel Maschine mit Microsoft Silverlight for Windows Embedded	114

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.

Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.

Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.

Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.

Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.

Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702

mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.

Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt



Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem <https://www.beckhoff.de/secguide>.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter <https://www.beckhoff.de/secinfo>.

2 Einführung in TwinCAT und Systemvoraussetzung

Ziel dieser Anleitung ist es, dem Leser einen schnellen Überblick über die Möglichkeiten der TwinCAT zu geben, ohne dabei auf Einzelheiten einzugehen. Eine Beispiel-Applikation, welche in den einzelnen Kapiteln Schritt für Schritt erweitert wird, ergänzt die Erklärungen.

Detaillierte Informationen sind aus den einzelnen Anleitungen der Programme zu entnehmen.

Systemvoraussetzung

x86 Prozessor

Zum Betrieb von TwinCAT 2 benötigen Sie einen PC mit einem x86 Prozessor.

Betriebssystem Windows 10 / 11

TwinCAT 2.11 läuft unter dem Betriebssystem Windows 10.

Die Runtime benötigt zwingend eine 32-bit Version von Windows 10.

Das Engineering wird auch für 64-bit Versionen von Windows 10 und 11 bereitgestellt.

Beispielprogramm

Um das Beispielprogramm nachvollziehen zu können, benötigen Sie TwinCAT ab Version 2.11.

Das hier vorliegende Beispiel bezieht sich auf die Feldbusse Lightbus, Ethernet oder EtherCAT. Zur Umsetzung werden benötigt:

- PC-Interfacekarte für I/O-Lightbus (FC2001) oder Ethernetkarte
- Buskoppler
- 2 Busklemmen mit je 2 digitalen Ausgängen (KL2032)
- Busendklemme (KL9010)
- Verdrahtungsmaterial (LWL-Kabel, Litze, usw...)
- 24V-Netzteil

Andere Feldbusse können ebenso nach dieser Anleitung verwendet werden.

Demokit:

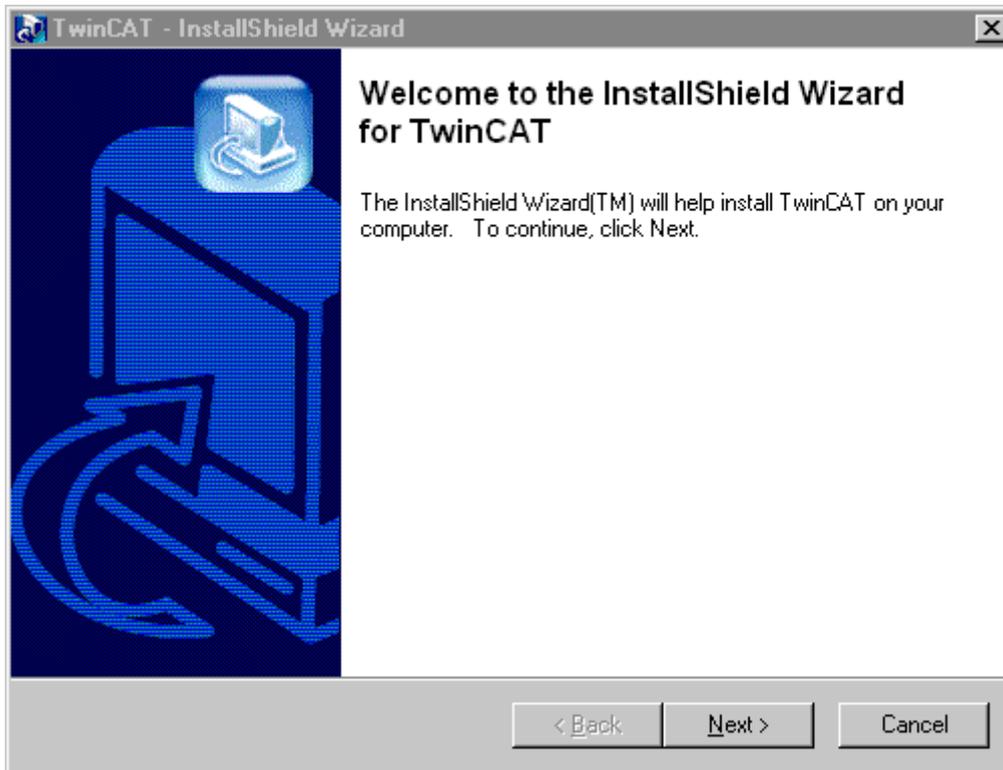
Die im Beispielprogramm benötigte Hardware ist in den Beckhoff Demokits enthalten.

3 Installation

3.1 Installationsprogramm starten

Starten Sie auf der CD das Programm SETUP.EXE. Öffnen Sie dazu den Windows Explorer, wechseln Sie auf das CD-ROM-Laufwerk und klicken Sie zweimal (Doppelklick) auf das Programm SETUP.EXE.

Die Installation beginnt mit dem folgenden Dialog, den sie mit 'Next' bestätigen müssen.



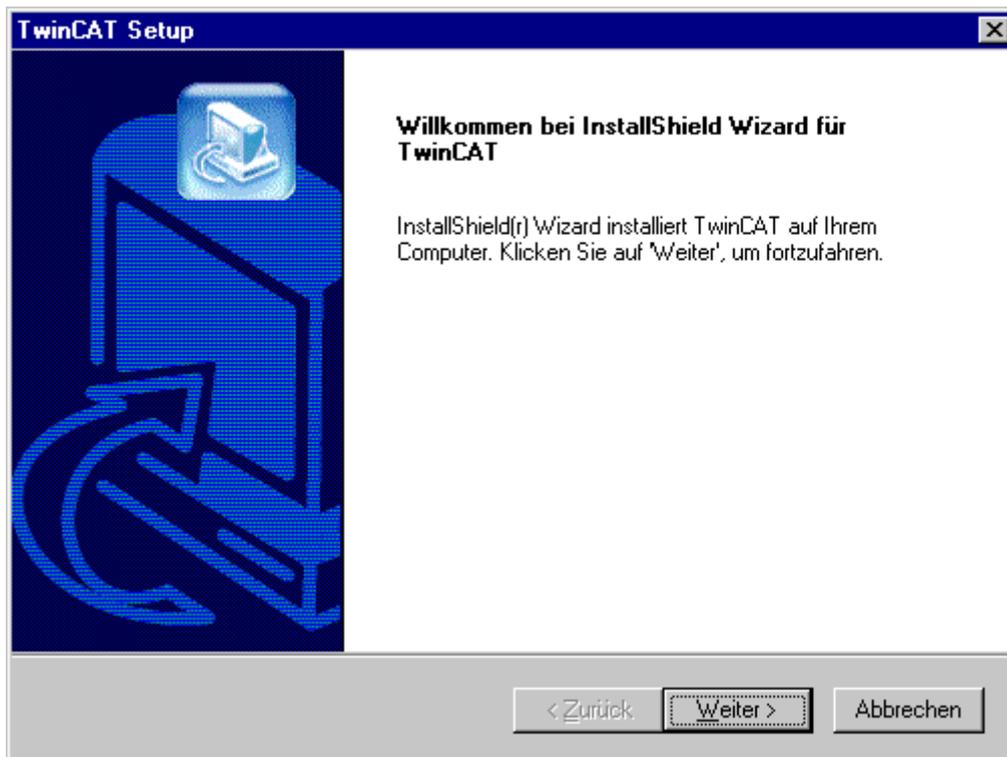
Sprache auswählen

Wählen Sie die Landessprache aus, in der TwinCAT installiert werden soll. Für die deutsche Installation wählen Sie den Eintrag 'Deutsch' und bestätigen Ihre Eingabe mit 'OK'. Der Installationsvorgang erfolgt komplett menügeführt.

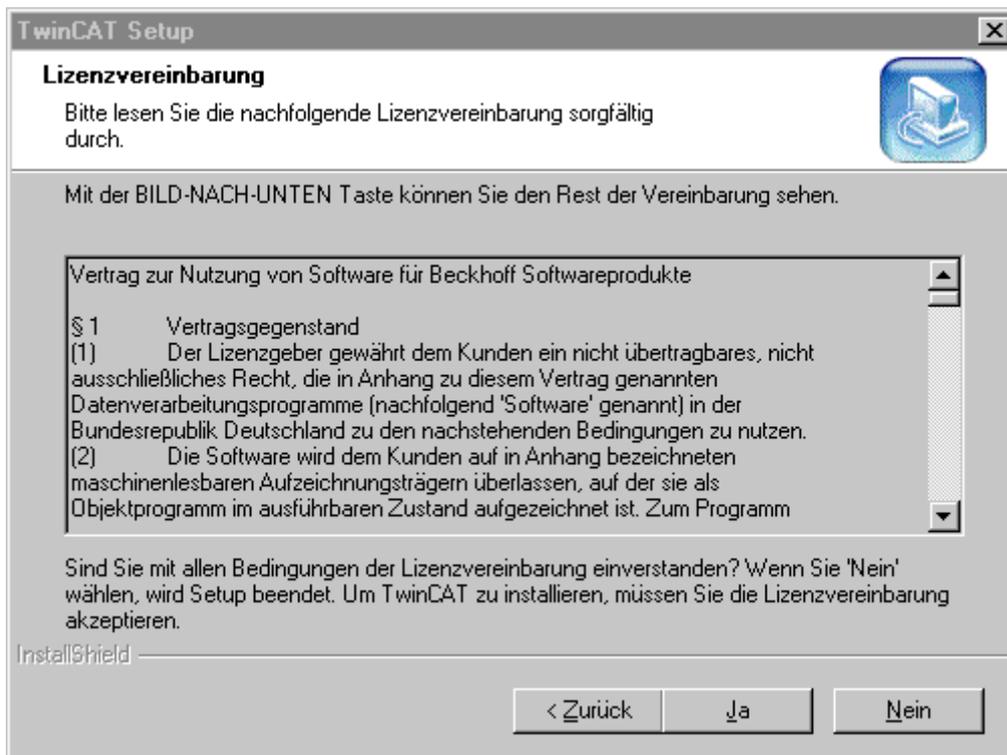


Programme beenden

Das Installationsprogramm empfiehlt, alle geöffneten Windows-Programme zu beenden, bevor das Setup-Programm durchgeführt wird.

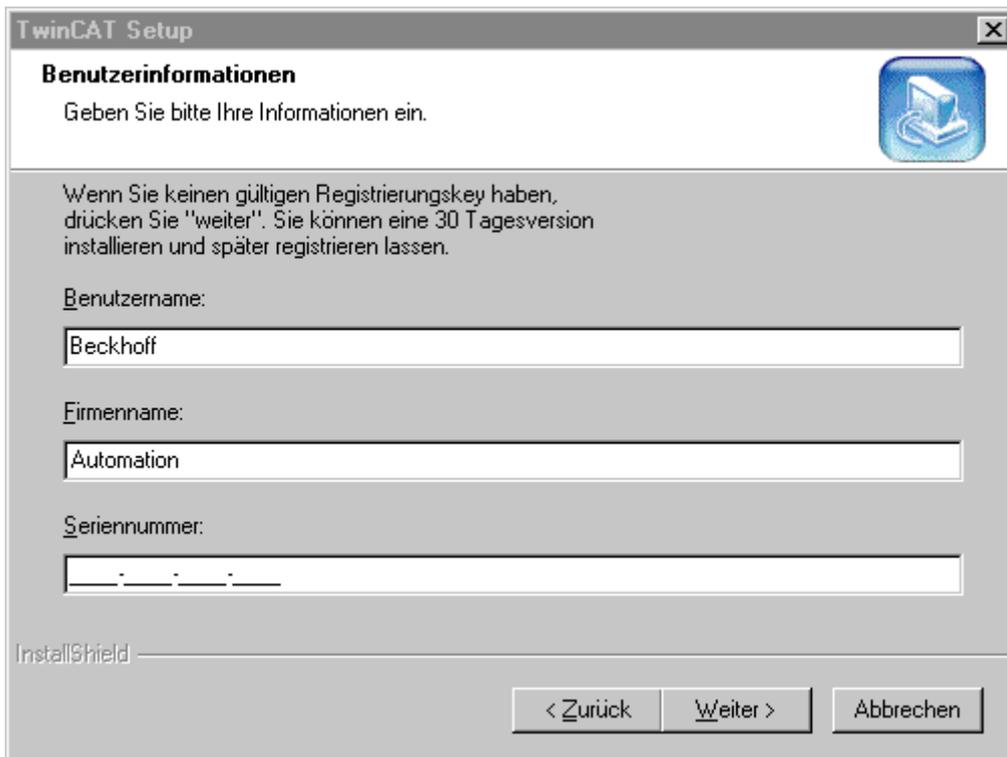


Lizenzvereinbarung



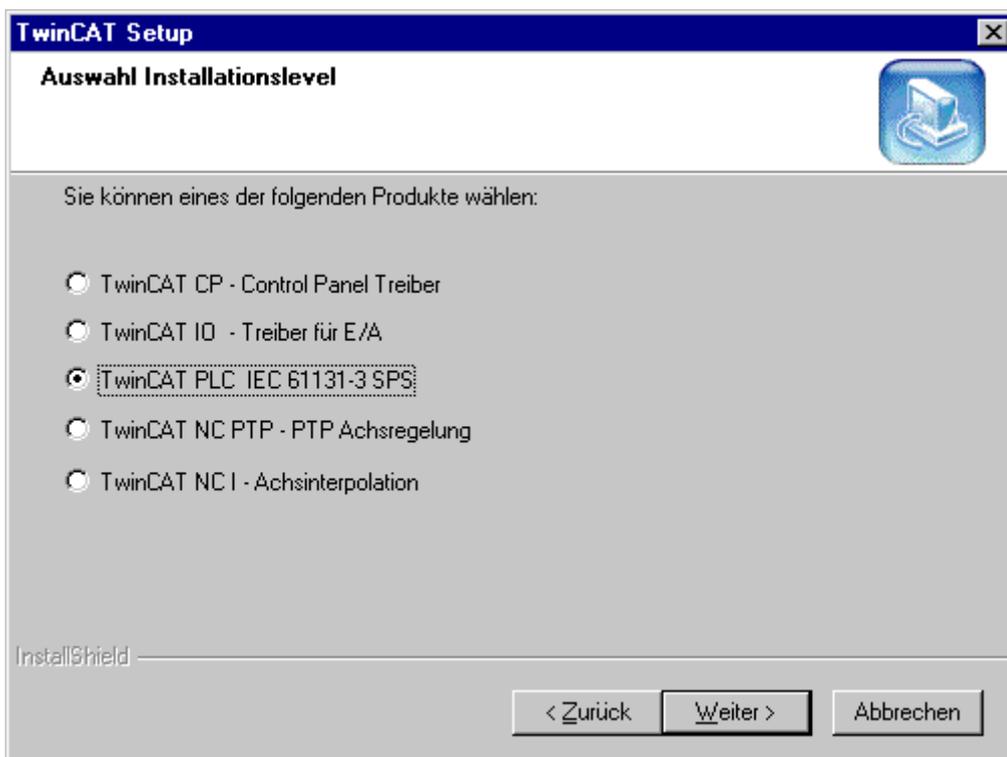
Benutzerinformationen eingeben

In diesem Dialogfenster müssen Sie Ihren Namen, die Firma und die Seriennummer eingeben. Wenn Sie derzeit noch keine Seriennummer besitzen, so lassen Sie das Feld für die Seriennummer leer und betätigen Sie 'Weiter'.



Installationslevel festlegen

TwinCAT kann in unterschiedlichen Ausbaustufen installiert werden:



Es steht Ihnen eine der folgenden Auswahlmöglichkeiten zur Verfügung:

TwinCAT CP

Enthält die nötigen Komponenten, um die Sonderfunktionen (USV, S-Tasten, ...) der Beckhoff Control Panels zu nutzen.

TwinCAT IO

Es können (User Mode-) Programme direkt auf die E/A-Geräte zugreifen. In diesem Level ist keine SPS enthalten.

TwinCAT PLC

In TwinCAT PLC ist die IEC61131-3 Entwicklungsumgebung enthalten.

TwinCAT NC PTP

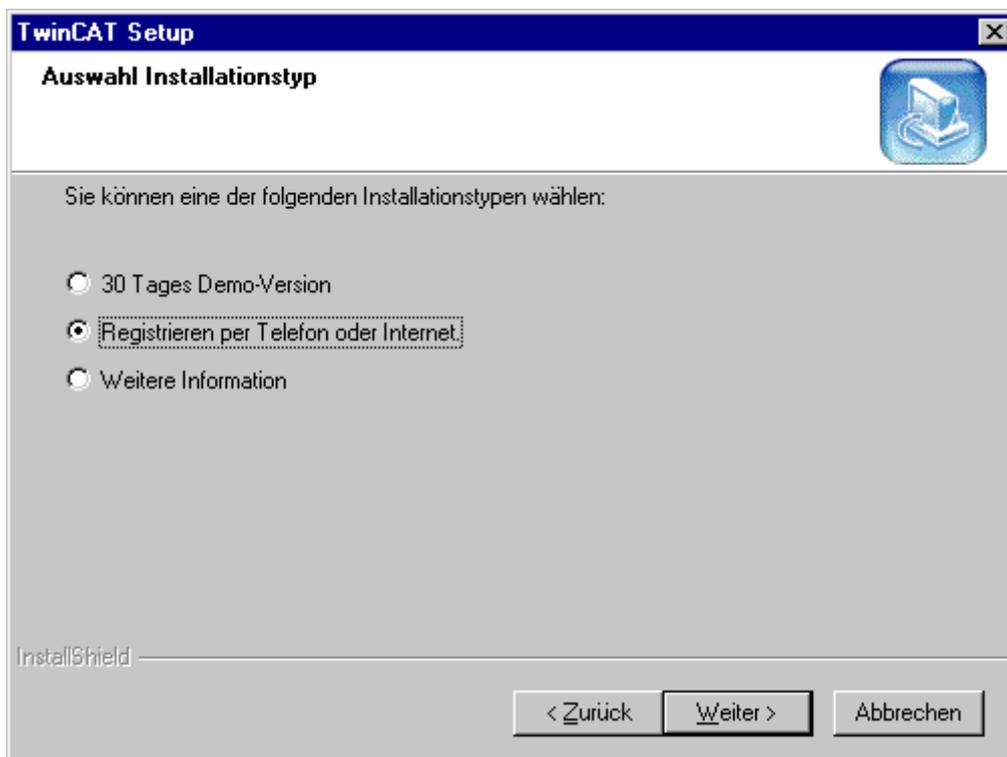
Es steht zusätzlich zur SPS noch die NC/CNC-Funktionalität zur Ansteuerung von PTP-Achsen zur Verfügung.

TwinCAT NC I

Zusätzlich zur SPS steht die NC-Funktionalität zur Interpolation von Antrieben im Raum zu Verfügung.

Installationstyp auswählen

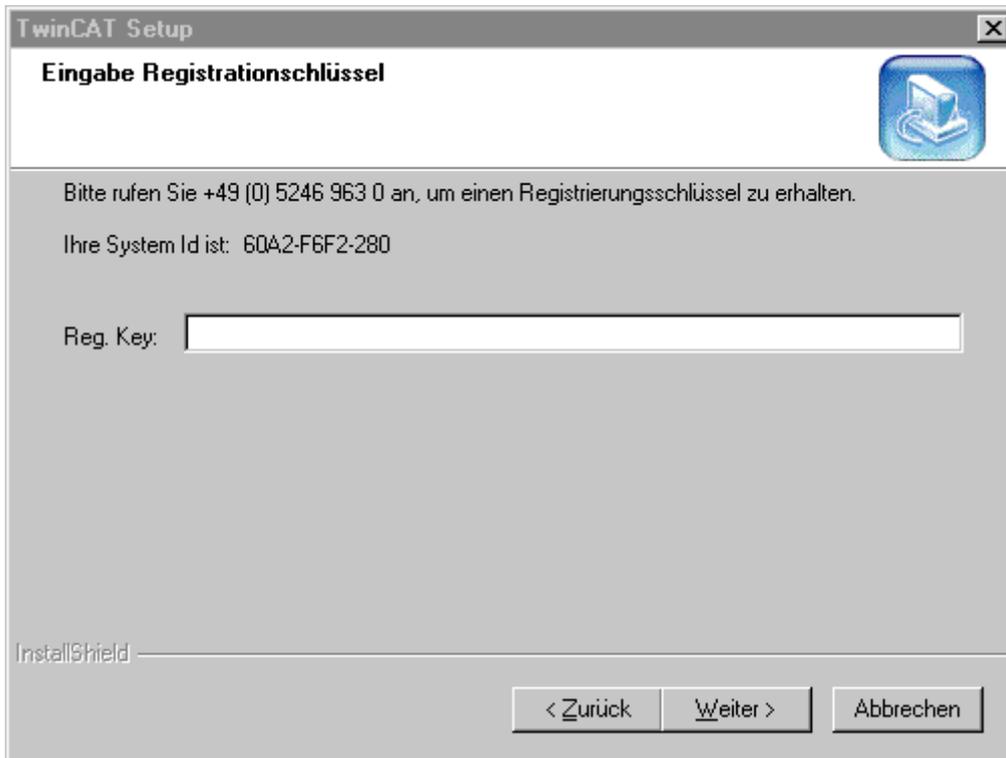
Entscheiden Sie an dieser Stelle, ob Sie TwinCAT als 30-Tages-Version oder mit direkter Registrierung installieren wollen. Haben Sie noch keine Registrierungsnummer, so können Sie erst eine 30-Tages-Version installieren und sich innerhalb der 30 Tage ohne Neuinstallation registrieren lassen.



Installationstyp	Einschränkungen
30 Tage Version	TwinCAT ist für 30 Tage ohne Einschränkungen lauffähig. Innerhalb dieser Zeit muss die Lizenznummer eingegeben werden. Falls dieses unterbleibt, kann TwinCAT nach 30 Tagen nicht mehr gestartet werden.
Registriermöglichkeit per Telefon oder Internet	Sie werden direkt bei der Installation nach der Lizenznummer gefragt. Siehe weiter unten.

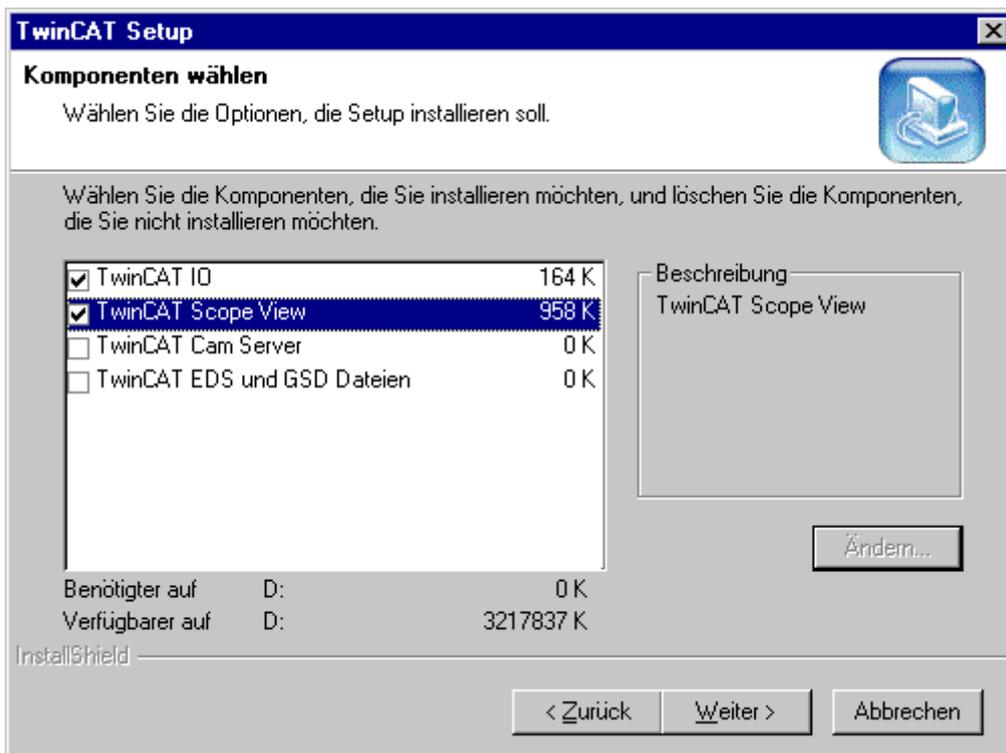
Registrationskey eingeben

Falls Sie sich entschieden haben TwinCAT zu registrieren, müssen Sie an dieser Stelle den Registrationskey eingeben. Diesen erhalten Sie von Beckhoff Automation. Die Telefonnummer ist unten auf dem Dialogfenster angegeben. Damit die Seriennummer errechnet werden kann, müssen Sie die System Id angeben. Diese ist ebenfalls auf dem nächsten Dialogfenster abgebildet.



Komponenten auswählen

Nicht alle Komponenten von TwinCAT werden standardmäßig mitinstalliert.



Voraussetzungen

Komponenten	Beschreibung
TwinCAT IO	Ermöglicht den direkten Zugriff auf das IO über eine DLL. Kann zusätzlich zu TwinCAT PLC oder TwinCAT NC PTP installiert werden.
TwinCAT Scope View	Zusatzprogramm für das grafische Darstellen von TwinCAT-Prozesswerten.
TwinCAT Cam Server	Schnelles Nockenschaltwerk.
TwinCAT EDS und GSD Dateien	Die EDS (DeviceNet) bzw. GSD Dateien (Geräte-Stamm-Dateien, Profibus) stellen dem Anwender alle notwendigen Einstellungen zur Konfiguration seines Systems bereit.

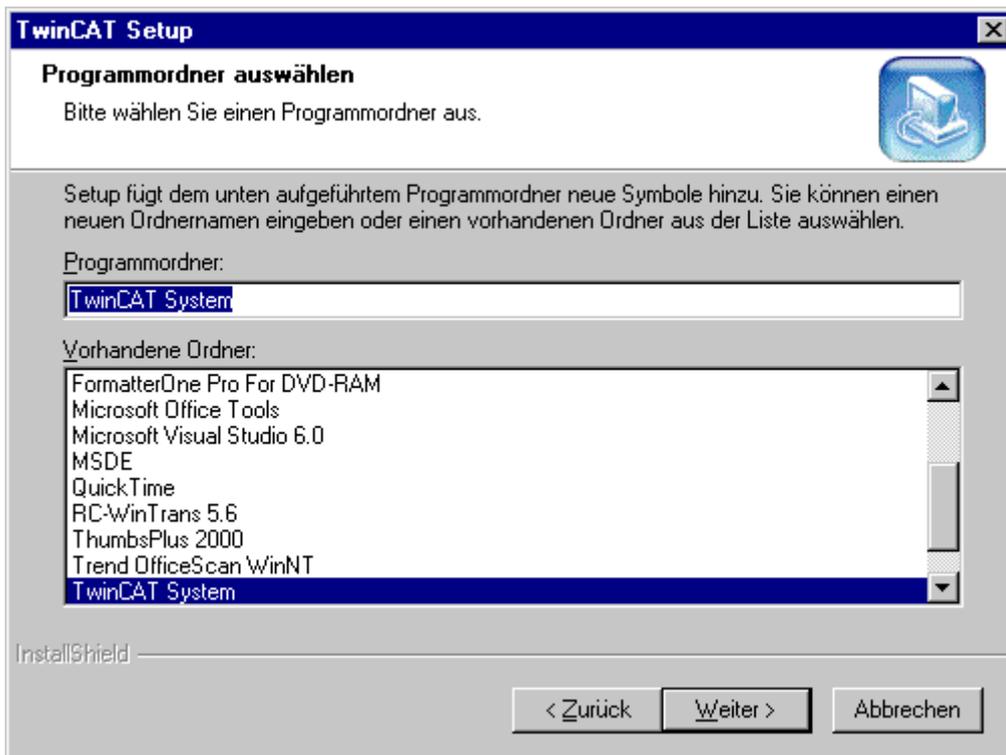
Zielpfad wählen

Im folgenden Dialog können Sie das Zielverzeichnis auswählen in dem TwinCAT installiert werden soll. In der Regel können Sie die Standardwerte übernehmen.



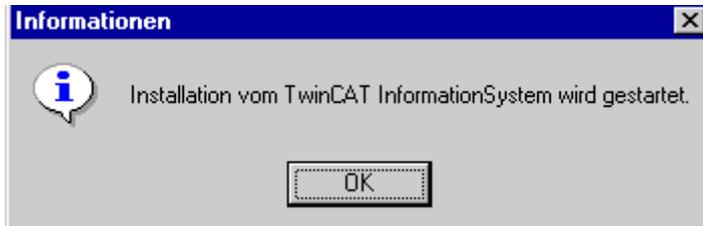
Programmordner auswählen

Anschließend muss der Programmordner eingegeben werden, in dem die Programmsymbole abgelegt werden sollen. Hier können Sie ebenfalls die Standardwerte übernehmen.



Nach der Installation von TwinCAT

wird automatisch die Installation des Beckhoff Information Systems gestartet. Das Beckhoff Information System enthält die gesamte Dokumentation zu TwinCAT. Bestätigen Sie bitte mit 'OK'.

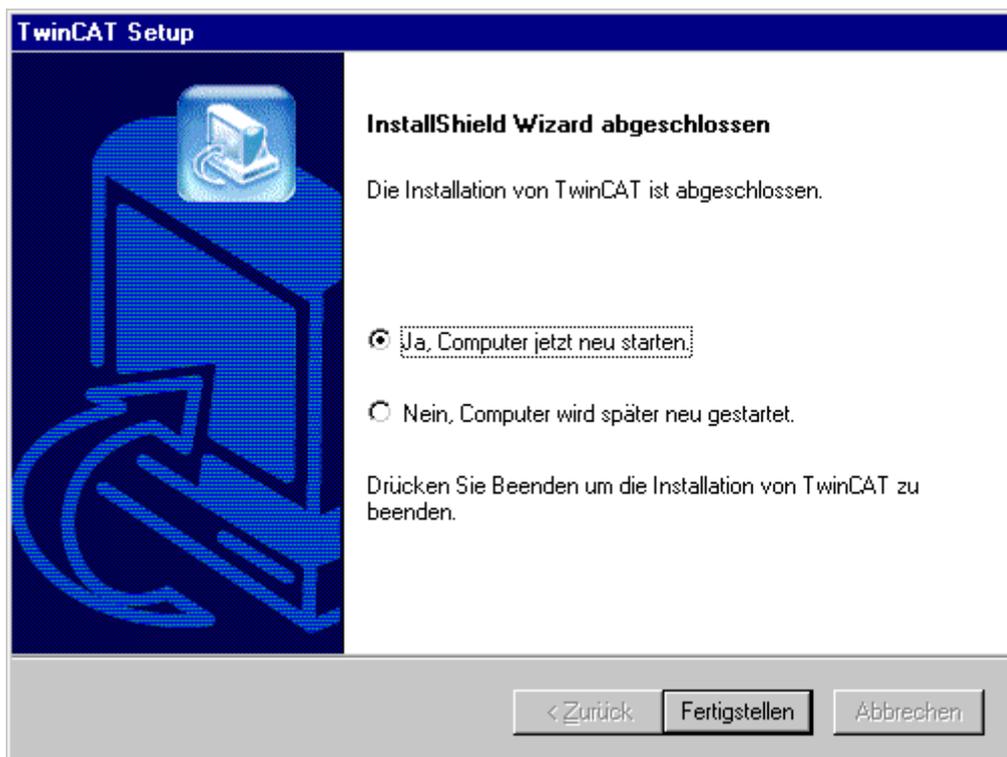


Ende der Installation des Informationssystems

Um die Installation des Informationssystems zu beenden, muss 'Finish' bestätigt werden. Danach wird auch die Installation von TwinCAT beendet.

Rechner neu starten

Nach der Installation von TwinCAT ist es notwendig, den Rechner neu zu starten.



Damit ist die Installation von TwinCAT abgeschlossen.

Silent-Installation

Es besteht die Möglichkeit, InstallShield die Antwortdatei für Sie erstellen zu lassen. Führen Sie Ihr Setup einfach mit dem Kommandozeilenparameter „Setup.exe -r“ aus. InstallShield zeichnet Ihre Einstellungen in der Datei Setup.iss auf und legt die Datei im Windows-Ordner ab.

Alle Sd-Dialogfensterfunktionen und in InstallShield integrierten Funktionen sind so konzipiert, dass sie Werte in die Datei Setup.iss schreiben, wenn InstallShield im Aufzeichnungsmodus (Setup -r) ausgeführt wird. Wenn Sie benutzerdefinierte Dialogfenster erstellen, müssen Sie SdMakeName und SilentWriteData aufrufen, um Abschnitte und Dialogfensterdaten in die Antwortdatei einzufügen, wenn das Setup im Aufzeichnungsmodus ausgeführt wird. Beispiele für die Verwendung dieser Funktionen zum Schreiben in Setup.iss finden Sie im Quellcode der Sd-Dialoge im Ordner <InstallShield>Include. Bitte lesen Sie den folgenden Abschnitt, um weitere Informationen darüber zu erhalten, welche Daten beim Aufruf von SdMakeName und SilentWriteData in Setup.iss eingefügt werden müssen.

Nachdem Sie Setup und Antwortdatei erstellt haben, gehen Sie wie folgt vor:

1. Legen Sie die Antwortdatei, die sich in Ihrem Windows-Ordner befindet, in den Ordner Advanced\Disk 1 des Fensters **Setup Files**.
 2. Erstellen Sie Ihre Medien. Wenn Sie eine selbstextrahierende ausführbare Datei für Ihr Setup erstellen, geben Sie „-s“ in das Bearbeitungsfeld **Setup-Kommandozeilenparameter** des Medienassistenten auf der Seite **Selbstextrahierendes Paket** oder auf der Packaging-Seite des Medieneigenschaftenblatts ein.
- ⇒ Jetzt können Sie das Setup im Silent-Modus mit InstallShield Silent ausführen.

Wenn Sie ein Setup im Silent-Modus ausführen, beachten Sie, dass keine Meldungen angezeigt werden. Stattdessen werden in einer Logdatei Setup.log Informationen über das Setup erfasst, einschließlich der Information, ob das Setup erfolgreich war. Sie können die Logdatei überprüfen und das Ergebnis des Setups ermitteln. (Beachten Sie, dass bei bestimmten Setup-Initialisierungsfehlern die Logdatei stattdessen Setupexe.log heißen kann und in SUPPORTDIR erstellt wird, wenn das Setup über das Internet ausgeführt wird, oder andernfalls in SRCDIR)

Um InstallShield Silent zu starten, führen Sie Setup.exe mit der Option -s aus.

InstallShield bietet außerdem die Schalter -f1 und -f2, mit denen Sie den Namen und den Speicherort der Antwortdatei sowie den Speicherort der Logdatei angeben können.

Der Wert **ResultCode** im Abschnitt **ResponseResult** in Setup.log gibt Aufschluss über ein erfolgreiches Silent-Setup. InstallShield schreibt einen entsprechenden Rückgabewert hinter den Schlüsselnamen **ResultCode**.

Setup.log ist der Standardname für die Silent-Setup-Logdatei, und ihr Standardspeicherort ist Disk1 - im selben Ordner wie Setup.inx. Sie können einen anderen Namen und Speicherort für Setup.log angeben, indem Sie die Schalter -f1 und -f2 mit Setup.exe verwenden.

Der zweite Abschnitt **Application**, gibt den Namen und die Version der installierten Anwendung sowie den Firmennamen an.

Der dritte Abschnitt **ResponseResult**, enthält den Ergebniscode, der angibt, ob Silent-Setup erfolgreich war oder nicht. Dem Schlüsselnamen **ResultCode** im Abschnitt **ResponseResult** wird ein Integer-Wert zugewiesen. InstallShield setzt einen der folgenden Rückgabewerte hinter den Schlüsselnamen **ResultCode**:

- 0 Erfolg.
- 1 Allgemeiner Fehler.
- 2 Ungültiger Modus.
- 3 Erforderliche Daten nicht in der Datei Setup.iss gefunden.
- 4 Nicht genügend Speicherplatz vorhanden.
- 5 Datei ist nicht vorhanden.
- 6 Es kann nicht in die Antwortdatei geschrieben werden.
- 7 Es kann nicht in die Logdatei geschrieben werden.
- 8 Ungültiger Pfad zur InstallShield Silent-Antwortdatei.
- 9 Kein gültiger Listentyp (String oder Zahl).
- 10 Ungültiger Datentyp.
- 11 Unbekannter Fehler während des Setup.
- 12 Die Dialogfenster sind nicht in Ordnung.
- 51 Der angegebene Ordner kann nicht erstellt werden.
- 52 Der Zugriff auf die angegebene Datei oder den angegebenen Ordner ist nicht möglich.
- 53 Ungültige Option ausgewählt.

3.2 Ende der Installation

Neue Programmsymbole

Im Startmenü von Windows NT/2000/XP befindet sich nach der Installation eine neue Programmgruppe mit fünf Programmsymbolen und zwei zusätzlichen Programmgruppen.



StartUp



Programme, die in dem Startup-Ordner liegen, werden automatisch von TwinCAT gestartet, wenn dieses durch die Auto-Boot Funktion aktiviert wurde. Nützlich ist dieses für Bedieneroberflächen, die auf Prozesswerte von TwinCAT zugreifen, da diese erst dann gestartet werden, wenn TwinCAT aktiv ist.

TwinCAT System Manager



Der TwinCAT System Manager ist das zentrale Verwaltungsprogramm von TwinCAT. Mit dessen Hilfe wird z.B. die Zuordnung der physikalischen I/O-Ebene (Feldbus) mit der logischen Prozessebene (SPS-Programm) durchgeführt. Diese Zuordnung wird auch als Mapping bezeichnet. Ebenfalls werden hiermit die Echtzeiteinstellungen vorgenommen.

TwinCAT PLC Control



Das TwinCAT PLC Control ist die Programmierumgebung für die IEC61131-3 - SPS. Hiermit werden die SPS-Programme geschrieben, verwaltet und ausgetestet.

TwinCAT System Control



Neben den auf der Arbeitsoberfläche sichtbaren Programmen, gibt es auch einige Programme und Treiber, die im Hintergrund, also nicht sichtbar, ablaufen. Die Konfiguration dieser Programm-Module wird mit Hilfe des TwinCAT System Control durchgeführt.

TwinCAT Scope View



Mit Hilfe des Scope View können TwinCAT Prozesswerte in Echtzeit aufgezeichnet und grafisch dargestellt werden. So kann z.B. die Dynamik von Achsen genau betrachtet werden.

In Windows NT/2000/XP integriert

Nach dem ersten Systemstart befindet sich in der rechten unteren Ecke der Arbeitsoberfläche das Symbol für den TwinCAT Realtime Server.



Anhand der Farbe ist der allgemeine Betriebszustand des Systems zu erkennen. Es gibt die Zustände 'gestartet' (grün), 'wird gestartet' (gelb) und 'gestoppt' (rot). Wenn Sie mit der Maus auf das Symbol klicken, öffnet sich ein Pop-Up-Menü, in dem Sie weitere Systemeinstellungen durchführen können. Sie können im Rahmen dieser Anleitung die Standardeinstellung übernehmen. Mit der rechten Maustaste ist es möglich, das System zu starten und zu stoppen, sowie die verschiedenen Systemprogramme direkt aufzurufen.

Beckhoff Information System



Das Beckhoff Information System ist eine stetig wachsende Referenz zu den TwinCAT Produkten. Es beinhaltet technische Informationen, Handbücher, Beispielcode, die TwinCAT Knowledge Base und vieles mehr. Die hierarchische Anordnung der Dokumente erleichtert das Finden der benötigten Informationen.

Vollversion:

Wenn Sie TwinCAT von der Beckhoff-Produkt-CD installiert haben, ist auf Ihrem Rechner nun das gesamte Beckhoff Information System installiert.

Basisversion:

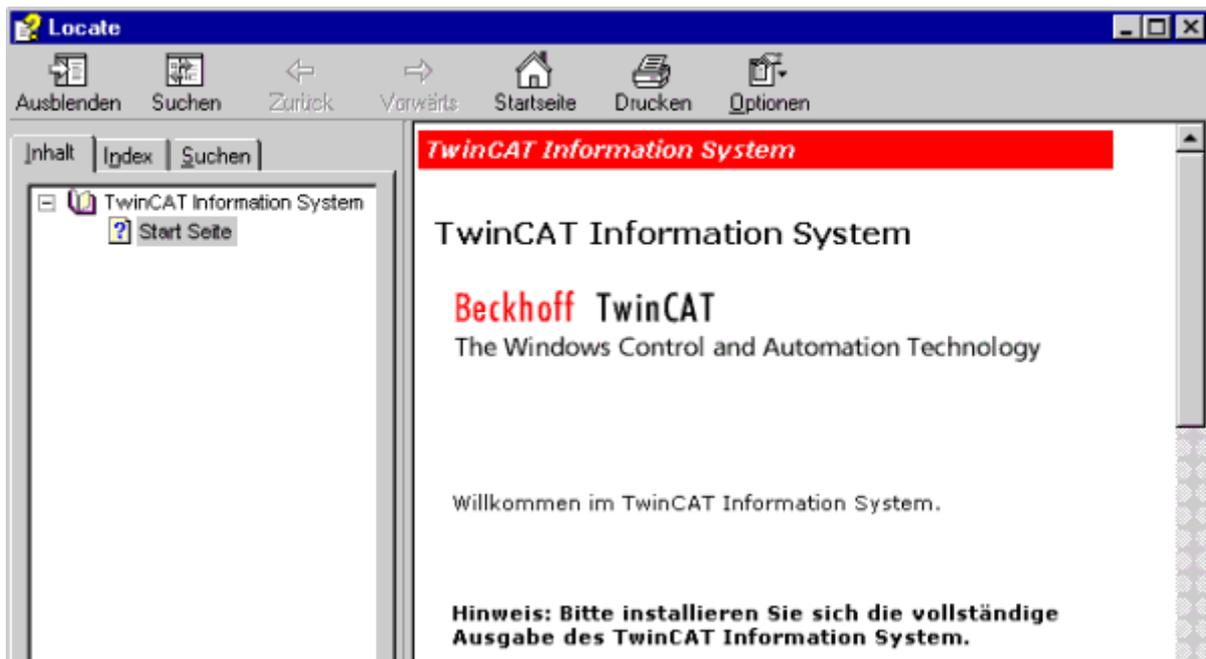
Haben Sie sich die Installation von TwinCAT aus dem Internet geladen, ist lediglich eine Basisversion des Beckhoff Information Systems auf Ihrem Rechner installiert.

Um eine vollständige Version (Umfang 200 MB) zu erhalten, haben Sie mehrere Möglichkeiten:

Sie finden das Beckhoff Information System

- auf sämtlichen Beckhoff-Produkt-CD's
- auf unserem FTP-Server
 - <http://download.beckhoff.com/download/Software/TwinCAT/TwinCAT2/InfoSystem/1031/Install/InfoSys.exe>
- und auf unserem Web-Server
 - <http://www.beckhoff.com>

Sie können die Anweisungen zur Installation auch der Startseite entnehmen, die Sie mit dem Startmenü, wie oben dargestellt, öffnen.



4 TwinCAT PLC Control



Was ist TwinCAT PLC Control?

Das TwinCAT PLC Control ist eine komplette Entwicklungsumgebung für Ihre Steuerung. Die Benutzung der Editoren und der Debugging-Funktionen hat die ausgereiften Entwicklungsumgebungen höherer Programmiersprachen zum Vorbild.

IEC 61131-3

TwinCAT PLC Control ermöglicht dem SPS-Programmierer einen einfachen Einstieg in die mächtigen Sprachmittel der IEC 61131-3. Bei der Entwicklung der TwinCAT PLC wurden folgende Leistungsmerkmale implementiert:

Mehrere Bausteinararten

Es werden von der TwinCAT PLC die Bausteinararten Anweisungsliste (AWL), strukturierter Text (ST), Ablaufsprache (AS), Funktionsplan (FUP), freigrafischer Funktionsplaneditor (CFC) und Kontaktplan (KOP) unterstützt.

Test ohne SPS

Durch die integrierte Software-SPS ist ein Testen des SPS-Programms ohne externe Hardware möglich.

Änderungen während des Betriebes

Änderung der Programme "online" in der SPS.

Wiederverwendbarkeit

Wiederverwendbarkeit von bestehenden SPS-Programmbausteinen.

Genormte Schnittstellen

Über genormte, offene Schnittstellen (OCX, DLL, usw.) ist eine Anbindung, auch über ein Netzwerk, zu anderen Programmen und Rechnern möglich.

Heterogene Umgebung

Durch die Verwendung von systemunabhängigen und weit verbreiteten Netzwerkprotokollen ist es möglich, die TwinCAT in eine heterogene Netzwerkumgebung einzubinden. So kann z.B. eine Oracle-Datenbank unter UNIX Daten mit der TwinCAT über TCP/IP austauschen und diese in einem BDE- oder PPS-System weiter verarbeiten oder Parameter innerhalb der TwinCAT PLC vorgeben, um den Produktionsprozess zu beeinflussen.

Hochsprach-Bibliotheken

Komplexe Algorithmen können z.B. in C++ oder Visual Basic entwickelt werden, um dann diese aus der TwinCAT PLC anzusprechen. Es gibt viele Drittanbieter solcher Bibliotheken die bestimmte Aufgabengebiete behandeln. Für komplexe, mathematische Aufgaben ist z.B. das Programm MathLab stark verbreitet.

SCADA-Systeme

Einige Hersteller von SCADA-Systemen (InTouch, Genie, Wizcon, usw.) bieten direkte Treiberunterstützung für die Anbindung an TwinCAT an.

Fernzugriff

Durch die Trennung von Programmierumgebung und Laufzeitumgebung, ist eine zentrale Programmierung verteilter Steuerungen über ein Netzwerk (auch ISDN) möglich.

Intuitive Entwicklungsumgebung

Simulation nach den Vorbild ausgereifter Hochsprach-Entwicklungsumgebungen (z.B. Visual C++). Breakpoint, Einzelschrittmodus, Tracen von Variablen, usw. sind mit der TwinCAT PLC möglich, so wie es bei modernen Entwicklungsumgebungen der Fall ist.

4.1 SPS-Standard IEC 61131- 3

Um ein SPS-Programm mit der TwinCAT PLC zu erstellen, stehen 5 verschiedene Sprachen nach IEC 61131-3 zur Verfügung.

Anweisungsliste (AWL)

Die Anweisungsliste ist der Programmiersprache STEP5 sehr ähnlich. Jede Anweisung beginnt in einer neuen Zeile und beinhaltet einen Operator und einen oder mehrere Operanden. Vor einer Anweisung kann sich eine Marke befinden, gefolgt von einem Doppelpunkt. Ein Kommentar muss das letzte Element in einer Zeile sein.

Beispiel:

Marke	Operator	Operand	Kommentar
Start:	LD	Beckenfuellstand	(* Füllstand laden *)
	GE	13	(* Grenzwert erreicht? *)
	JMPC	Pumpe_ein	
	R	Pumpe_Ansteuerung	(* Pumpe ausschalten *)
	JMP	Ende	
Pumpe_ein:	S	Pumpe_Ansteuerung	(* Pumpe einschalten *)
Ende:			

Strukturierter Text (ST)

Man spricht bei dieser Programmiersprache auch von einer höheren Programmiersprache, da nicht "maschinennahe" Befehle verwendet werden, sondern über abstrakte Befehle mächtige Befehlsfolgen aufgebaut werden können. Aus dem PC-Bereich sind Basic, PASCAL und C vergleichbare höhere Programmiersprachen.

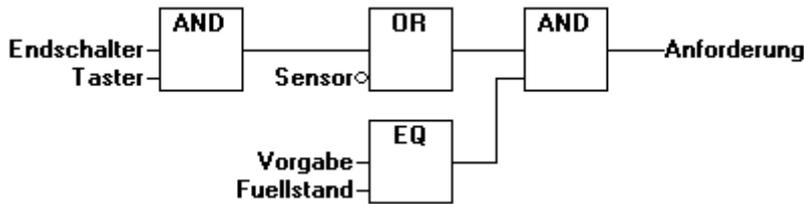
Beispiel:

Operator	Operand	Kommentar
CASE Temperatur_Ofen OF		(* Heizleistung regeln *)
	60..99: Heizung := 80;	(* 80% *)
	100..149: Heizung := 60;	(* 60% *)
	150..199: Heizung := 35;	(* 35% *)
	200..250: Heizung := 10;	(* 10% *)
ELSE Alarm := TRUE;		(* Alarm setzen *)
END_CASE;		

Funktionsplan (FUP)

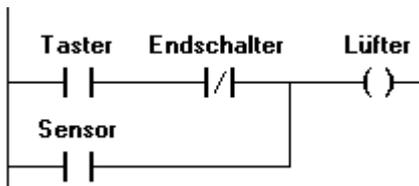
Der Grundgedanke bei der SPS-Programmierung mit Funktionsplan ist, dass der Aufbau in funktionsorientierten logischen Ablaufketten (Netzwerke) erfolgt. Innerhalb eines Netzwerks ist die Abarbeitungsrichtung stets von links nach rechts. Vor Ausführung eines Bausteins müssen alle Eingangswerte gebildet worden sein. Die Auswertung eines Netzwerks ist erst dann abgeschlossen, wenn die Ausgangswerte aller Elemente berechnet sind.

Beispiel:



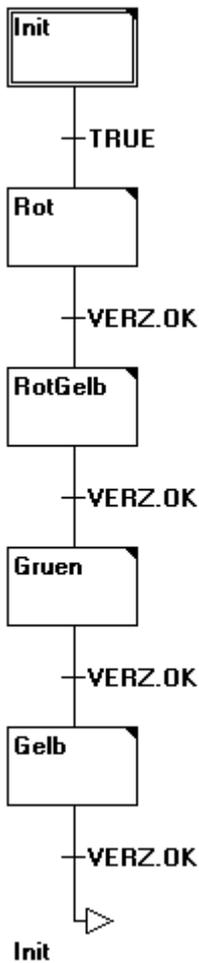
Kontaktplan (KOP)

Aus dem Bereich des elektrotechnischen Anlagenbaus stammt die Darstellung von logischen Abläufen in Form des Kontaktplans. Für die Umsetzung von Relaisschaltungen in SPS-Programmen ist diese Darstellung besonders gut geeignet. Die Bearbeitung beschränkt sich auf die booleschen Signale 1 und 0.



Ablaufsprache (AS)

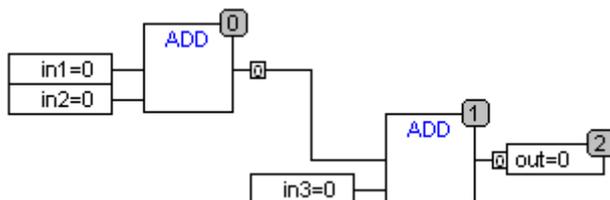
Die Ablaufsprache ist dort sinnvoll, wo eine Schrittkettenprogrammierung notwendig ist. Komplexe Aufgaben werden in übersichtliche Programmteile (Schritte) zerlegt. Anschließend wird der Ablauf zwischen diesen Schritten grafisch festgelegt. Die Schritte selbst werden in einer anderen Programmiersprache (ST, AWL, ...) erstellt oder können auch wieder in AS dargestellt werden.



AS-Programme bestehen im Wesentlichen aus Schritten, Transitionen (Übergangsbedingungen) und ihren Verknüpfungen. Jedem Schritt ist eine Menge von Befehlen zugeordnet; diese Befehle werden ausgeführt, wenn der Schritt aktiv ist. Eine Transition muss erfüllt sein, damit der nachfolgende Schritt ausgeführt wird. Die Schritte und die Transitionen können in einer beliebigen Sprache formuliert werden.

Freigrafischer Funktionsplaneditor (CFC)

Der freigrafische Funktionsplaneditor arbeitet nicht wie der Funktionsplan FUP mit Netzwerken, sondern mit frei platzierbaren Elementen. Dies erlaubt beispielsweise Rückkopplungen. Ein Beispiel für ein Netzwerk im Freigrafischen Funktionsplaneditor könnte typischerweise so aussehen:



5 TwinCAT System Manager



Was ist TwinCAT System Manager?

Der TwinCAT System Manager ist das zentrale Konfigurationswerkzeug des TwinCAT Systems. Hier werden die Ein- und Ausgänge der beteiligten Software Tasks und die physikalischen Ein- und Ausgänge der angeschlossenen Feldbusse verwaltet. Zusätzlich können Online-Werte der aktiven Konfiguration mit dem TwinCAT System Manager betrachtet werden.

Durch das Verknüpfen von Variablen der Software Tasks und Variablen der Feldbusse werden die logischen Ein- und Ausgänge den physikalischen zugeordnet.

Konfigurationsmodule des TwinCAT System Managers

Nachfolgend sind die verschiedenen Hauptkomponenten des TwinCAT System Managers aufgelistet. Die Existenz der Konfigurationskomponenten hängt vom Level des installierten TwinCAT Systems ab.

Echtzeit Konfiguration

Echtzeiteinstellungen und Erstellung benutzerdefinierter Tasks

SPS-Konfiguration

SPS Einbindung von bis zu 4 unabhängigen Laufzeitsystemen

Nocken-Konfiguration

Elektronisches Nockenschaltwerk und dessen Nockenkonfiguration

E/A-Konfiguration

Geräte- und Feldbuskomponenten-Verwaltung

6 TwinCAT Scope View



Was ist Scope View?

Das TwinCAT Scope View ist ein Analysewerkzeug zur grafischen Darstellung der Variablen aus verschiedenen SPS-Tasks.

Die Aufzeichnung kann dabei wahlweise gegen die Zeit oder auch als XY-Darstellung erfolgen.

Jedes Scope View kann über mehrere Kanäle verfügen, wobei die Anzahl nur durch den Speicher und die Rechengeschwindigkeit beschränkt ist. Bei der Darstellung gegen die Zeit ist jedem Kanal eine Variable zugeordnet.

Analyse des Scope Views

Zur Analyse des Scope Views stehen Werkzeuge bereit.

Datensicherung

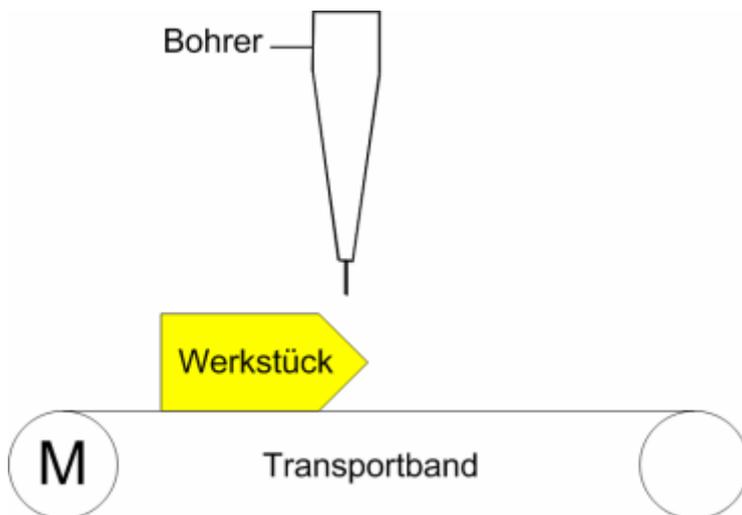
Das TwinCAT Scope View ermöglicht eine Sicherung der Daten in mehreren Dateiformaten, z. B. in einer Excel Tabelle.

7 Beispielprogramm

7.1 Beispiel Machine.pro

Das Erstellen von Applikationen mit der TwinCAT soll anhand eines Beispiel-Programms erläutert werden. Dieses Programm stellt eine Bearbeitungsmaschine für beliebige Werkstücke dar. Es befindet sich nach der Installation von TwinCAT im Verzeichnis '\\TwinCAT\Samples\First Steps\PLC' und hat den Namen 'Machine.pro'.

Skizze:



Funktionsbeschreibung:

- 1) Das Transportband wird beim Programmstart 5 Schritte angesteuert.
- 2) Der Bohrer wird für 2 sec. nach unten gefahren.
- 3) Der Bohrer wird für 2 sec. nach oben gefahren.
- 4) Bei Schritt 1 wird wieder begonnen.

TwinCAT starten:

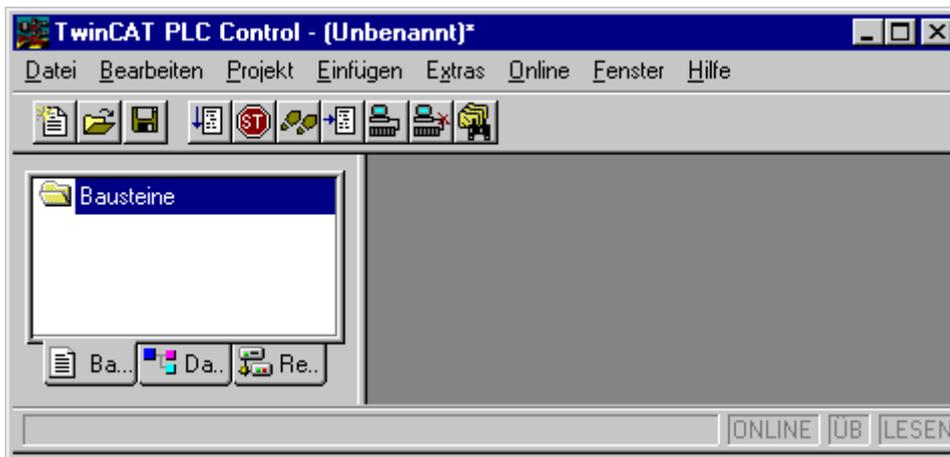
Bevor Sie das Programm ausführen können, müssen Sie den TwinCAT Realtime Server aktivieren.



Klicken Sie dazu mit der Maus auf das Symbol des TwinCAT Realtime Servers und aktivieren Sie den Befehl 'Start' aus dem Menü 'System'. Die Farbe des Symbols wechselt über gelb nach grün. Dies bedeutet, dass der Echtzeitkern von TwinCAT aktiv ist.

TwinCAT PLC Control starten:

Starten Sie jetzt die Programmieroberfläche von der TwinCAT SPS. Betätigen Sie dazu mit der Maus 'Start' -> 'Programme' -> 'TwinCAT System' -> 'TwinCAT PLC Control'.



Projekt öffnen:



Ein SPS-Projekt wird in einer Datei auf der Festplatte oder Diskette abgelegt, die den Namen des Projektes trägt. Um ein Projekt zu öffnen betätigen Sie den Menüpunkt 'Datei' und anschließend den Befehl 'Öffnen'.

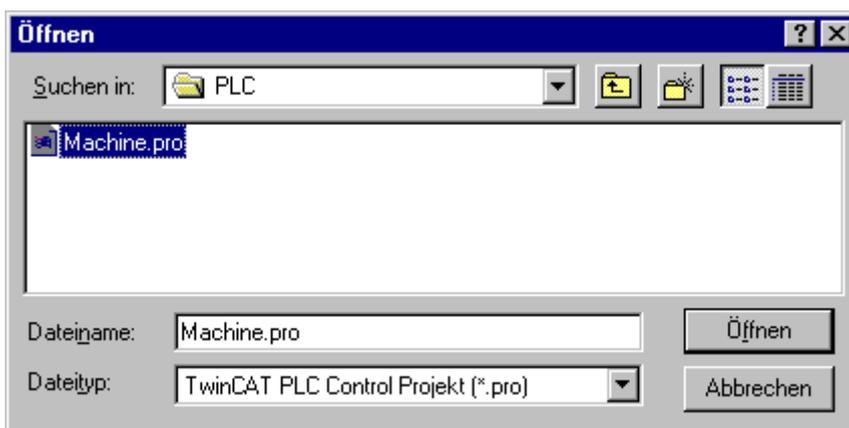
Verzeichnis auswählen:



Wechseln Sie in das oben angegebene Verzeichnis, indem Sie im Dialogfenster das abgebildete Symbol anklicken. Klicken Sie den Eintrag 'Samples' zweimal schnell hintereinander (Doppelklick). Verfahren Sie danach genauso mit dem Eintrag 'First Steps' und 'PLC'.

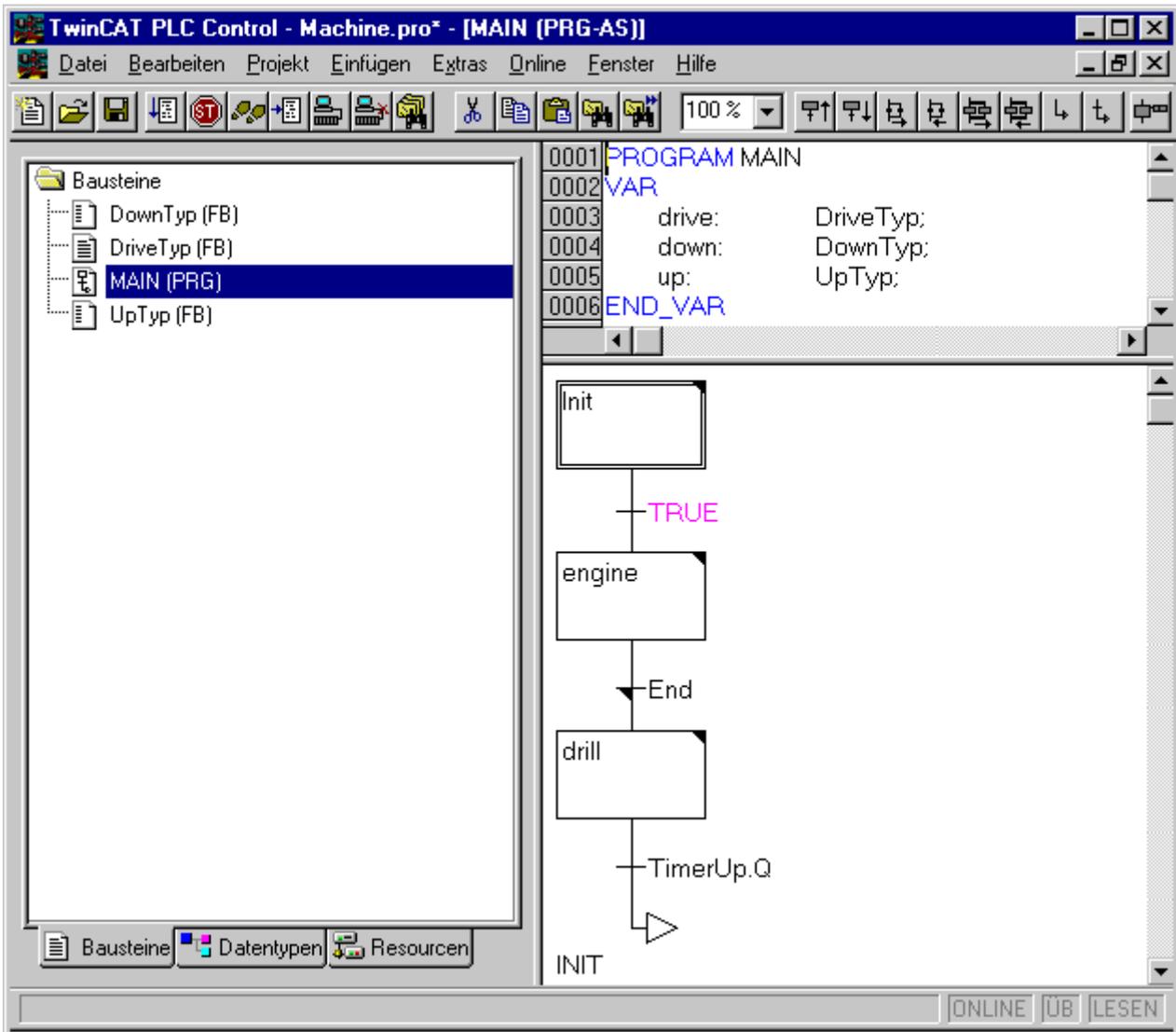
Projekt auswählen:

Wählen Sie das Projekt 'Machine.pro' aus, indem Sie den Eintrag im Dialogfenster mit der Maus anklicken und anschließend den Befehl 'Öffnen' ausführen.

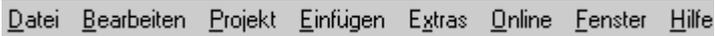


Die Elemente von PLC Control:

Nach dem Öffnen von Machine.pro und einem Doppelklick mit der linken Maustaste auf den Programmbaustein MAIN öffnet sich folgendes Dialogfenster.



Im oberen blauen Balken steht der Projektname Machine.pro. 

Direkt darunter befindet sich das Befehlsmenü 

und die Toolbar: 

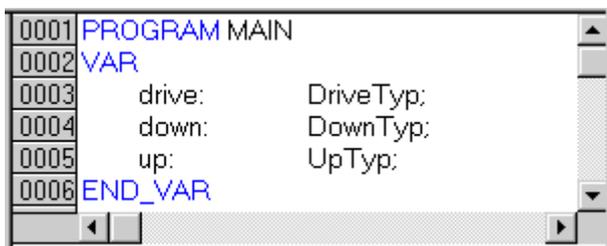
Der unterste graue Balken enthält die Statuszeile: 

Das Dialogfenster teilt sich in drei einzelne Fenster, sie enthalten die Objektliste, die Variablendeklaration und die Programmdarstellung.

Objektliste:

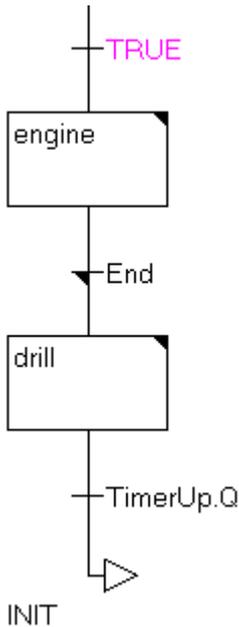
TwinCAT unterscheidet drei Arten von Grundobjekten in einem Projekt:

- (Programm-)Bausteine (POE)
- Datentypen
- Ressourcen

Variablendeklaration:

Ein SPS-Programm speichert seine Daten in Variablen ab. Variablen sind vergleichbar mit Merkerwörter oder Datenwörter. Bevor eine Variable benutzt werden kann, muss diese deklariert werden, d.h. ihre Zuordnung zu einem bestimmten Datentyp (z.B. BYTE oder REAL) bekannt gegeben werden. Bei der Deklaration werden auch bestimmte Attribute wie Batteriepufferung, Anfangswerte oder Zuordnung zu physikalischen Adressen festgelegt. Wird eine Variable nicht im Eingangsabbild oder Ausgangsabbild benötigt, also nur innerhalb des SPS-Programms, so braucht sich der SPS-Programmierer nicht um den Speicherort der Daten zu kümmern. Dieses übernimmt TwinCAT. Das ungewollte Überschneiden von Merkerwörter / Datenwörter, so wie es bei bisherigen Systemen möglich war, wird hierdurch vermieden (Seiteneffekte). Wie bei Variablen, müssen auch Funktionsbausteine deklariert (instanziiert) werden. In dem Beispiel werden von den drei Funktionsblöcken 'DriveTyp', 'DownTyp' und 'UpTyp' je eine Instanz (Drive, Down und Up) angelegt. Nach der Instanziierung können die Instanzen verwendet und aufgerufen werden.

Programmdarstellung:



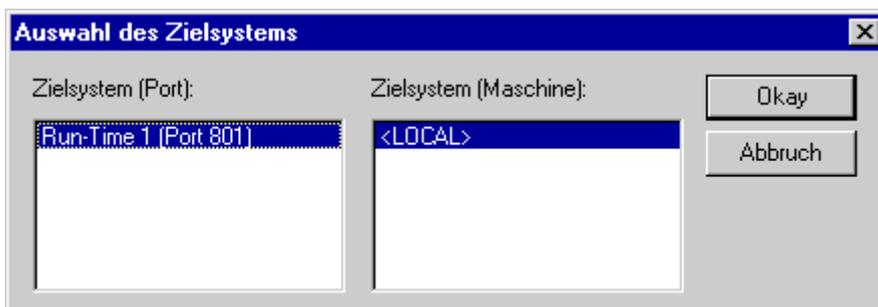
In diesem Bereich von der TwinCAT PLC Control wird das SPS-Programm eingegeben und dargestellt.

Zielsystem auswählen:

TwinCAT stellt bis zu 4 Laufzeitsysteme zur Verfügung. Jedes dieser Laufzeitsysteme kann unabhängig von den anderen Laufzeitsystemen SPS-Programme in IEC61131-3 ausführen. Auf welchem Laufzeitsystem Sie Ihre Programme ausführen wollen, entscheiden Sie durch den Befehl 'Auswahl des Zielsystems ...', im Menü 'Online'.

TwinCAT muss gestartet sein, um eine Auswahl treffen zu können.

Sofern sich TwinCAT im 'Run' oder 'Konfig'-Modus befindet, werden alle konfigurierten Laufzeitsysteme angezeigt.



Nach der Installation von TwinCAT ist eventuell nur ein Laufzeitsystem freigegeben. Bestätigen Sie die Auswahl mit 'OK'.

Falls sich die PLC-Runtime nicht auf dem lokalen Rechner befindet, müssen Sie zuvor eine Verbindung zum gewünschten Zielsystem herstellen (siehe TwinCAT-Dokumentation).

Einloggen:



Sie haben jetzt das SPS-Programm in der TwinCAT PLC Control geladen und wollen dieses ausführen. Vergewissern Sie sich, dass der TwinCAT Realtime Server aktiv ist. Sie erkennen das daran, dass rechts unten auf dem Bildschirm das TwinCAT Realtime Server Symbol grün dargestellt wird. Bevor Sie ein SPS-

Programm starten, müssen Sie die TwinCAT PLC Control mit dem Laufzeitsystem verbinden; sich in die Steuerung 'einloggen'. Führen Sie im Menü 'Online' den Befehl 'Einloggen' aus. Da sich im Laufzeitsystem noch kein SPS-Programm befindet, erscheint die Meldung: Kein Programm auf der Steuerung! Alles übersetzen?. Bestätigen Sie die Frage mit 'OK'.

In der Statuszeile wechselt daraufhin die Anzeige: Laufzeit: 1 ONLINE

SPS Programm starten:



Mit dem Befehl 'Start' aus dem Menü 'Online' wird das SPS-Programm im TwinCAT Realtime Server gestartet. Die Anzeige 'LÄUFT' in der Statuszeile wechselt von hellgrau auf schwarz. Sie sehen auch, dass innerhalb der Ablaufsprache einzelne Schritte zeitweise blau dargestellt werden. Ein blau dargestellter Schritt wird gerade ausgeführt, ist also der aktive Schritt.

SPS Programmverlauf verfolgen:

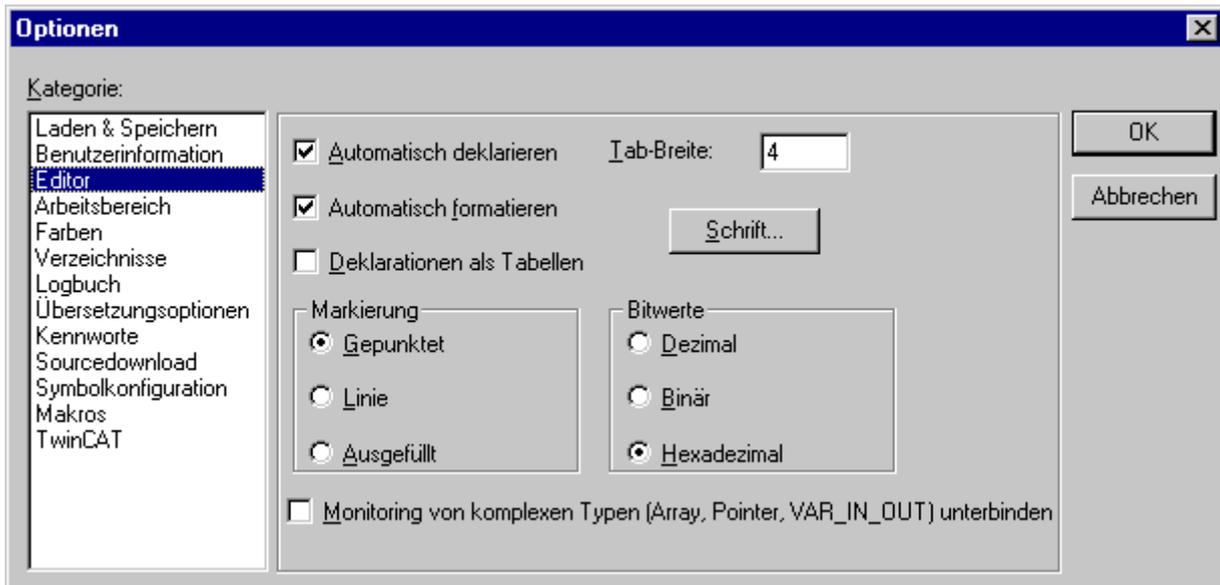
Über den Befehl 'Globale Variablen' in der Objektliste 'Ressourcen' sehen Sie alle global deklarierten Variablen. Globale Variablen können von allen Programmbausteinen (POE) gemeinsam benutzt werden.

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

Neben den Variablen werden dort auch die Funktionsbausteine 'timerUp', 'timerDown' und 'devTimer' angezeigt. Vor dem Funktionsnamen ist ein Pluszeichen. Durch einen Doppelklick mit der Maus auf dieses Zeichen öffnet sich eine baumartige Darstellung, in der alle Variablen der Funktion angezeigt werden.

Zahlendarstellung ändern:

Den Inhalt der Variablen können Sie in verschiedenen Zahlensystemen anzeigen lassen. Sie haben die Wahl zwischen Dezimal, Hexadezimal und Dual. Wenn Sie die Anzeige ändern wollen, müssen Sie im Menü 'Projekt' den Befehl 'Optionen...' ausführen. In der Kategorie 'Editor' können Sie den entsprechenden Eintrag auswählen.



Programm beenden:



Sie haben jetzt ein SPS-Programm in den TwinCAT PLC Control (IEC 61131-3 Programmierumgebung) geladen und auf den TwinCAT PLC Server (Laufzeitsystem) ausgeführt. Beenden Sie nun das SPS-Programm. Führen Sie dazu in den Menü 'Online' den Befehl 'Stop' aus.

Ausloggen:



In den nächsten Abschnitten werden wir das SPS-Programm ergänzen. Dazu ist es notwendig, dass Sie sich von den TwinCAT PLC Server abmelden (ausloggen). Im Menü 'Online' müssen Sie dazu den Befehl 'Ausloggen' ausführen.

Programmtext betrachten:

Dieses Beispiel wurde in den verschiedenen Programmiersprachen der IEC61131-3 programmiert. Der Hauptteil des Programms ist in der Ablaufsprache (AS) erstellt worden. Er beinhaltet die Schritte:

- Init
- Engine
- Drill

und die Transitionen:

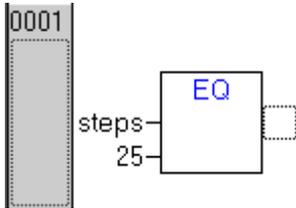
- TRUE
- End·

TimerUp.Q

Transitionen betrachten:

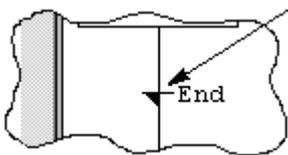
Die Transition 'TRUE' ist ständig erfüllt, da das Schlüsselwort 'TRUE' eine systemweite Konstante ist und ständig erfüllt ist (TRUE engl. wahr). Der Schritt 'Engine' wird bedingungslos nach dem Schritt 'Init' ausgeführt. TimerUp.Q bedeutet, dass die Variable Q in der Funktion Up TRUE (wahr oder auch 1) sein muss, damit diese Transition erfüllt ist.

'End' ist eine Transition, die weiteren Programmtext beinhaltet. Durch einen Doppelklick mit der Maus auf die Transition 'End' öffnet sich ein weiteres Fenster, in dem der entsprechende Programmtext angezeigt wird.



In der Transition 'End' wird verglichen, ob die 25 Schritte des Motors schon erreicht wurden. Ist dieses der Fall, wechselt das Programm bei dem nächsten Zyklus von dem Schritt 'Engine' zu dem Schritt 'Drill'.

Beinhaltet ein Schritt oder eine Transition weiteren Programmtext, so wird dieses durch ein kleines schwarzes Dreieck gekennzeichnet.

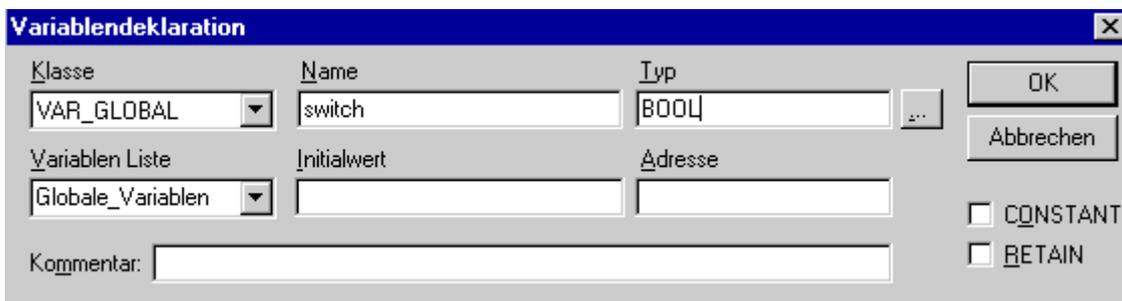


SPS Programm ändern:

Unter dem Menüpunkt Fenster können Sie wieder zum Fenster MAIN wechseln. An dieser Stelle soll das SPS-Programm von Ihnen so geändert werden, dass die Taktgeschwindigkeit des Motors in zwei Stufen (schnell/langsam) über eine Variable verändert werden kann. Öffnen Sie durch einen Doppelklick in der Objektliste den Funktionsblock 'DriveTyp'. Setzen Sie den Eingabecursor in die 1. Zeile und geben Sie folgenden Text ein:

```
IF switch = TRUE THEN
```

Nach betätigen der Eingabetaste (Return) erscheint ein Dialogfenster, welches Sie, wie unten abgebildet, ausfüllen müssen.



Nach Betätigen von 'OK' wird die Variable 'Schalter' an die Variablenliste von MAIN angefügt (evtl. ist es möglich das diese Dialogbox bei Ihnen nicht erscheint, da in dem Dialog 'Optionen' die automatische Variablendeklaration deaktiviert werden kann. Dann öffnen Sie im Menü Bearbeiten die Variablen Deklaration).

Geben Sie noch folgende Programmzeilen ein:

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

Das Fenster hat anschließend folgenden Inhalt:

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

Ist die Variable 'switch' gesetzt, wird die Variable 'devSpeed' auf 10 ms gesetzt, andernfalls auf 25 ms. Dieses hat zur Folge, dass in den folgenden Programmzeilen die Impuls- und die Pausendauer des Taktgebers entweder 25 ms oder 10 ms beträgt.

Programm speichern:



Speichern Sie das Programm durch den Befehl 'speichern' im Menü 'Datei'.

Programm übersetzen:

Bevor ein Programm in den TwinCAT PLC Server übertragen werden kann, muss dieses übersetzt werden, d.h. von der textuellen oder grafischen Darstellung in eine für die Steuerung verständliche Form konvertiert werden. Führen Sie dazu aus dem Menüpunkt 'Projekt' den Befehl 'Alles Übersetzen' aus.

Programm starten:

Loggen Sie sich an die Steuerung ein und starten Sie das SPS-Programm. Sie sehen, dass die Variable 'switch' beim Starten auf 'FALSE' steht.

Variablenwerte ändern:

Sie haben die Möglichkeit, Werte von Variablen zu verändern, während das SPS-Programm läuft. Öffnen Sie das Fenster 'globale Variablen' und führen Sie einen Doppelklick auf den Eintrag 'switch' aus. Die Anzeige ändert sich von FALSE auf TRUE und die Schriftfarbe wird rot. Der Wert in dem TwinCAT PLC Server hat sich bis zu diesem Zeitpunkt aber noch nicht geändert; dazu müssen Sie in dem Menüpunkt 'Online' den Befehl 'Werte schreiben' ausführen. Die Schriftfarbe wird wieder schwarz und die Variable 'devSpeed' ändert sich auf 10 ms.

Programmablauf verfolgen:

Um den Programmablauf verfolgen zu können müssen Sie TwinCAT Scope View öffnen.

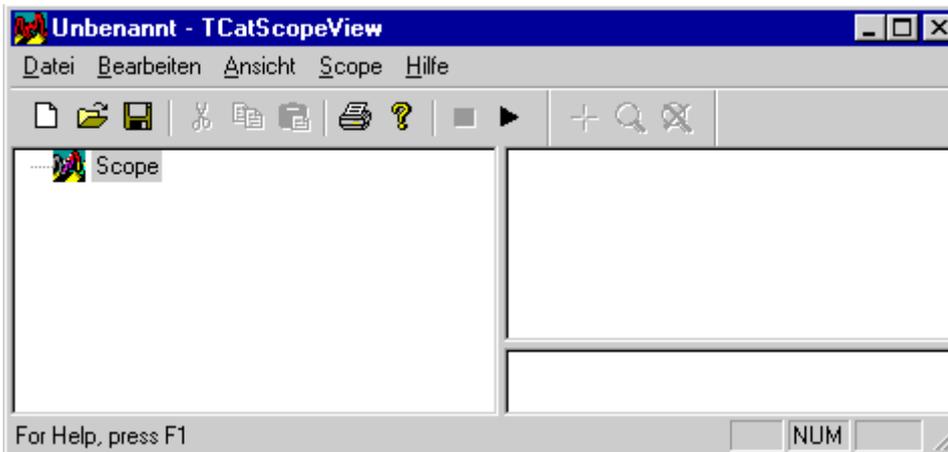
7.2 Programmablauf verfolgen

Zur Aufzeichnung und Analyse des Programms wird TwinCAT Scope View verwendet.

TwinCAT Scope View öffnen:

Sie können das Scope View nur über das Startmenü öffnen.

Betätigen Sie dazu mit der Maus 'Start' -> 'Programme' -> 'TwinCAT System' -> 'TwinCAT Scope View'.

**Die Elemente von TwinCAT Scope View:**

Das Fenster des TwinCAT Scope View ist ähnlich dem des TwinCAT PLC Control aufgebaut. In der oberen Zeile steht der Name des Projekts, darunter befindet sich die Menüleiste und die Toolbar. Darunter befinden sich drei große Fenster, die beim Öffnen noch leer sind. Im linken Fenster wird die Konfiguration des Scopes vorgenommen, wie nachfolgend beschrieben.

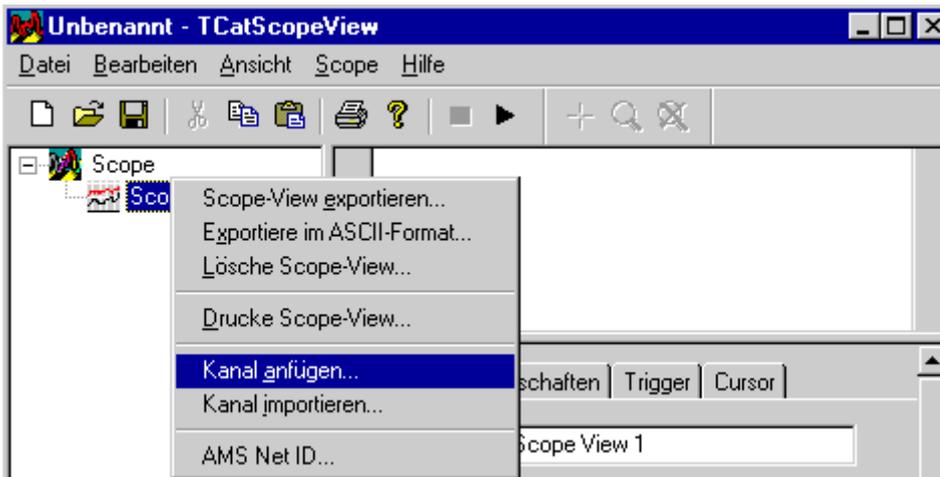
TwinCAT Scope View starten:

Um die Aufzeichnung des Beispielprogramms Machine.pro starten zu können, müssen sie zunächst ein Scope View, d.h. ein Projekt hinzufügen. Dazu klicken Sie mit der rechten Maustaste auf Scope, wählen dort Scope View hinzufügen und bestätigen mit 'OK'.

**Kanal einfügen:**

Zur Darstellung der einzelnen Signale müssen die jeweiligen Kanäle erzeugt werden.

Dazu klicken Sie mit der rechten Maustaste auf das Scope View 1, wählen Kanal hinzufügen und bestätigen mit 'Ok'.



Anschließend erscheinen folgende Karteireiter, mit denen die Variable beschrieben wird, deren Wert aufgezeichnet werden soll.



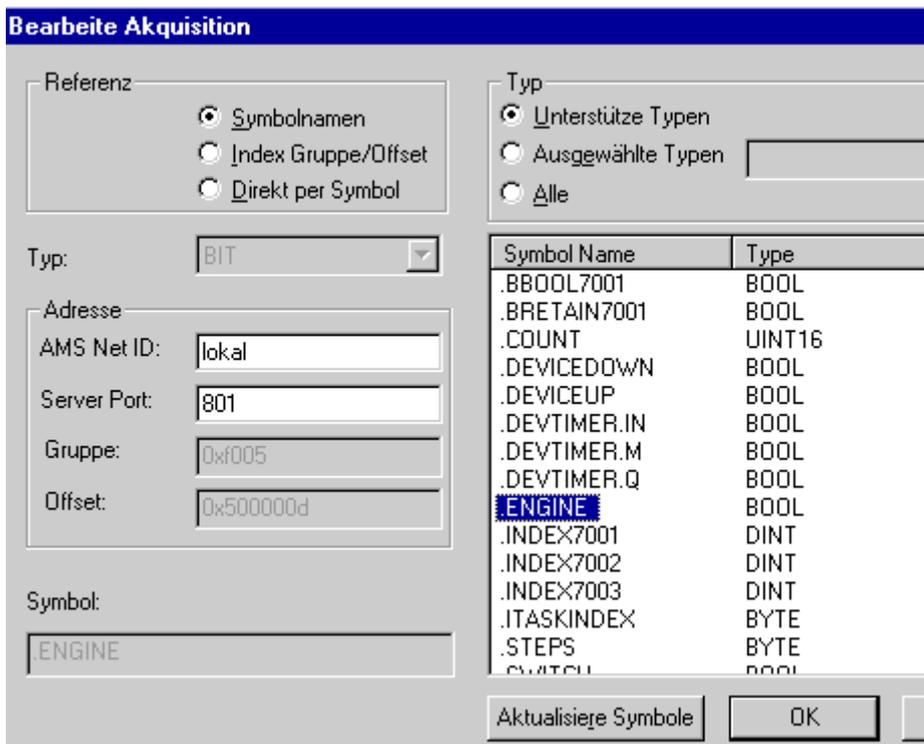
Um eine übersichtliche Darstellung zu erhalten, setzen Sie die Abtastzeit benutzerdefiniert auf 10 ms.

Server Port einrichten:

Als Nächstes stellen Sie den Server Port ein. Dazu klicken Sie unter Akquisition auf Verändern, geben die Nummer ein und bestätigen mit 'OK'. (Der Server Port ist dem PLC Control zu entnehmen, siehe unter Zielsystem auswählen [▶ 31].)

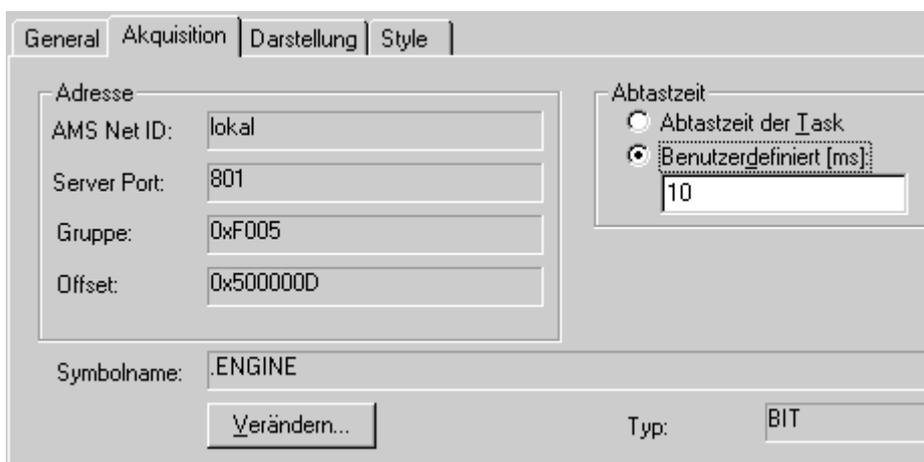
Kanal zuordnen:

Mit einem erneuten Klicken auf 'Verändern' ordnen Sie dem Kanal 1 das Signal .ENGINE zu und bestätigen dies mit OK.



Einstellungen

Danach hat der Karteireiter folgende Einstellungen:



Kanal umbenennen:

Durch einen langsamen Doppelklick auf Kanal1 kann eine Umbenennung in ENGINE erfolgen.

Weitere Kanäle zufügen:

Auf diese Art und Weise können Sie die weiteren Kanäle zuordnen.

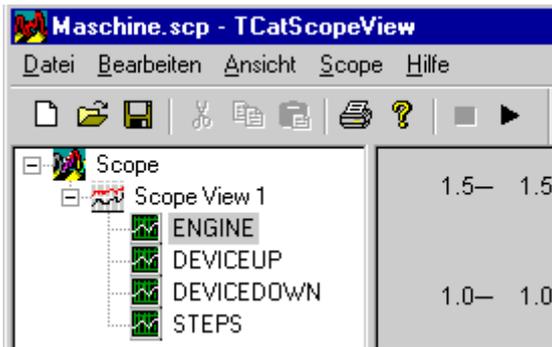
Kanal2=.DEVICEUP

Kanal3=.DEVICEDOWN

Kanal4=.STEPS

Der Server Port und die Abtastzeit bleiben für jeden Kanal gleich.

Nachdem Sie die vier Kanäle zugefügt und entsprechend umbenannt haben, speichern Sie das Scope View im Menü 'Datei', 'speichern unter' unter dem Namen Machine.scp ab.



Um die einzelnen Kurven voneinander unterscheiden zu können, kann jedem Kanal eine andere Farbe/Form bzw. Skala zugewiesen werden. Dies geschieht mit den Karteireitern Style bzw. Darstellung.

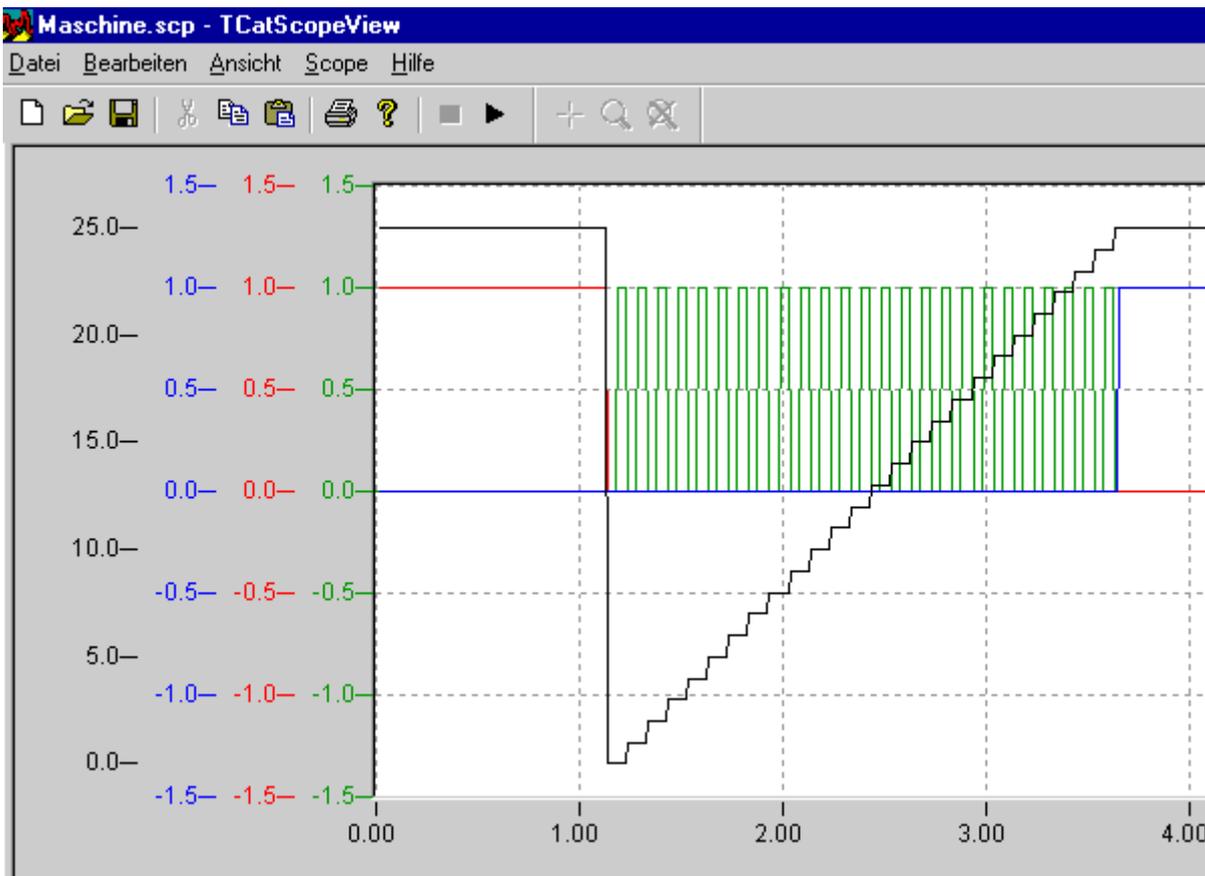
Aufzeichnung starten:



Sie starten die Aufzeichnung mit dem Menü 'Scope', Start Scope.

Machine.pro

Damit sieht das Beispielprogramm wie folgt aus:



7.3 Umsetzung des Beispielprogramms

7.3.1 Deklaration der Variablen

In diesem Kapitel werden Sie das vorherige SPS Programm an die Beckhoff Busklemmen anschließen. Dies geschieht mit dem TwinCAT System Manager. Dort werden alle Ein-/Ausgabeanschlüsse verwaltet, adressiert und den Ein- und Ausgangsdaten zugeordnet. Jeder E/A-Kanal ist mit einem logischen Namen ansprechbar. Der TwinCAT System Manager verwaltet mehrere Feldbusse an einem gemeinsamen Prozessabbild.

Hardware erforderlich

Dazu ist es erforderlich, dass Sie die erforderliche Hardware, d.h. das Demokit von Beckhoff für den Lightbus FC2001 besitzen. Das Demokit enthält die PC Interfacekarte für den I/O-Lightbus (FC2001), den Buskoppler BK 2000, diverse Busklemmen, 2 Lichtwellenleiter und Informations- bzw. Dokumentationsmaterial.

Ohne Hardware können Sie dieses Kapitel überspringen und zu den Applikationsbeispielen übergehen.

Variablendeklaration

Der Speicherort (Adresse) von Variablen wird vom System intern verwaltet. Der Programmierer braucht sich nicht um die Speicherverwaltung zu kümmern. Innerhalb der SPS-Programme wird mit symbolischen Variablennamen gearbeitet. Dadurch wird verhindert, dass Seiteneffekte (Überschneidungen) bei der Verwendung von Variablen auftreten können. Um auf die Ein-/Ausgabebene zuzugreifen, ist es notwendig, dass der Programmierer einzelnen Variablen eine feste Adresse zuweisen kann. Dieses wird durch das Schlüsselwort 'AT' erreicht, welches bei der Variablendeklaration mit angegeben werden muss. Hinter dem Schlüsselwort 'AT' stehen mehrere Parameter, die Auskunft über Datenort (Ein-/Ausgabe oder Merkerbereich) und Datenbreite (BIT, BYTE, WORD oder DWORD) geben. Die Variablendeklaration für das obige Beispiel hat folgenden Aufbau:

```
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
```

Dabei bedeutet:

	Datenart	Datenbreite	Bedeutung
%			Einleitendes Zeichen
	I		Eingang
	Q		Ausgang
	M		Merker
		X	Bit (1Bit)
		B	Byte (8 Bit)
		W	Wort (16 Bit)
		D	Doppelwort (32 Bit)

Die Ziffern hinter der Datenbreite geben die Adresse der Variable an. Für Bitvariablen muss die Adresse im Format x.y angegeben werden, für Byte, Wort und Doppelwort einfach x. Die Ein- und Ausgänge befinden sich in unterschiedlichen Speicherbereichen und können daher die gleiche Adresse enthalten.

7.3.2 Lightbus - Einrichten der Busklemmen

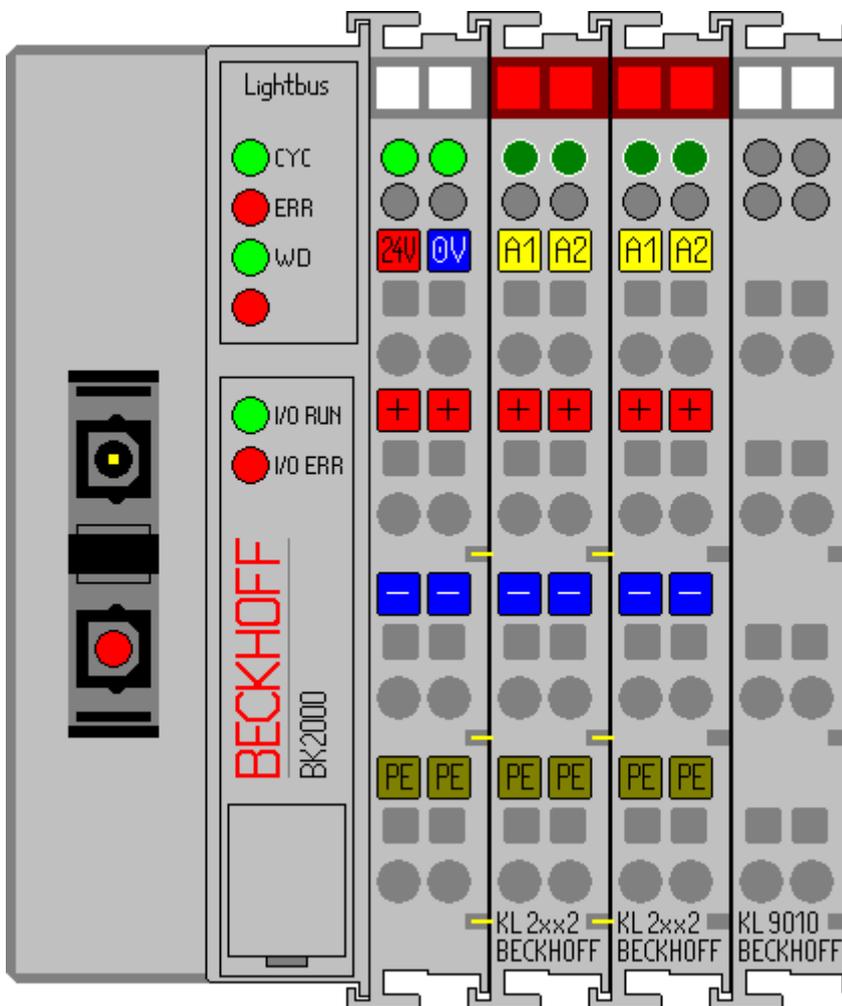
Benötigte Hardware:

- PC-Interfacekarte: FC2001
- Buskoppler: BK2000
- 3 digitale Ausgänge 24V: 2x KL2032
- Busendklemme: KL9010

Dieses Beispiel ist genau auf den Inhalt des Demokits TC9910-B200-0000 für den Beckhoff Lightbus ausgelegt. Die Hardware ist jedoch austauschbar. Dabei ist die Konfiguration der E/A-Geräte entsprechend zu ändern.

Aufbau der Klemmen:

Bauen Sie den Buskoppler und die Busklemmen auf, wie in der unteren Zeichnung dargestellt:



Schließen Sie den Buskoppler an die PC-Interface Karte an und versorgen Sie den Buskoppler mit 24 V.

Hardware Dokumentation

Nähere Informationen zum Hardware Anschluss entnehmen Sie bitte der gesonderten Hardware Dokumentation des oben genannten Demokits.

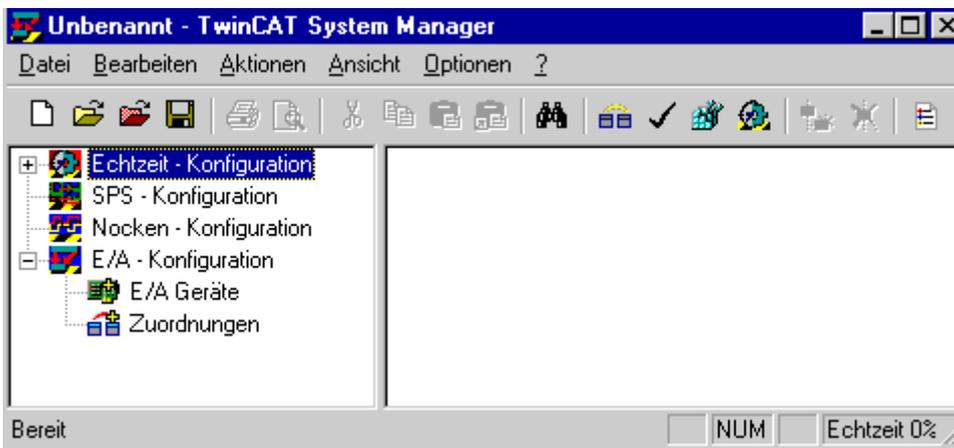
TwinCAT Realtime Server starten

Starten Sie nun, falls noch nicht geschehen, den TwinCAT Realtime Kernel, damit der TwinCAT Message Router aktiv ist.



TwinCAT System Manager starten

Nachdem das System gestartet wurde, ändert sich die Farbe des Icons von rot auf grün. Starten Sie jetzt den TwinCAT System Manager über 'Start'-> 'Programme' -> 'TwinCAT System' -> 'TwinCAT System Manager'.



Die Elemente des TwinCAT System Managers:

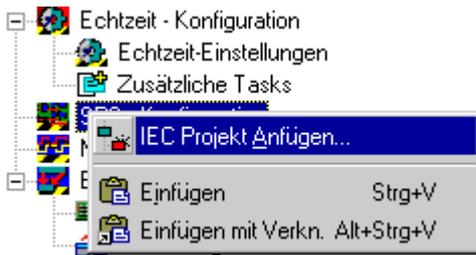
In der obersten Zeile steht der Name des Projektes, hier noch unbenannt, darunter befindet sich die Menüleiste und die Toolbar. Der untersten Zeile ist der Status des Systems zu entnehmen, in diesem Bild läuft das System (zu erkennen an der Einstellung 'Echtzeit'). Die beiden großen Fenster in der Mitte enthalten die Konfiguration des Systems. Nachfolgend wird mit dem Beispielprogramm diese Konfiguration durchgeführt.

Im linken Fenster ist die Systemkonfiguration als Baumstruktur dargestellt. Sie besteht aus den vier Hauptpunkten:

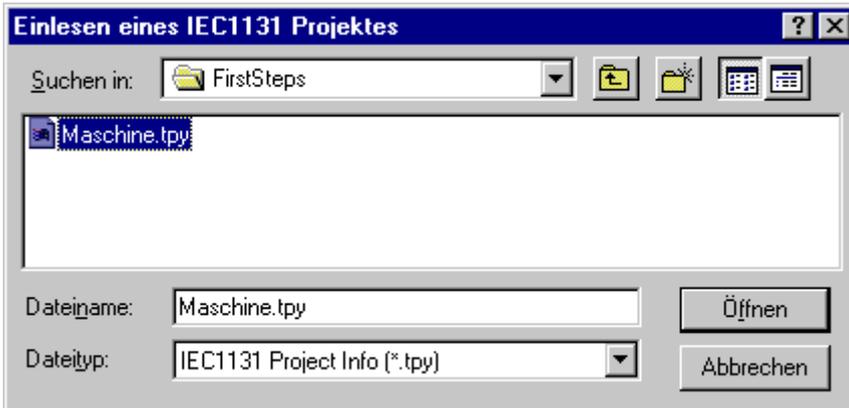
Konfiguration	Bedeutung
Echtzeit	Konfiguration der Echtzeitparameter
SPS	Unter diesem Eintrag werden sämtliche SPS-Projekte aufgelistet, die bei der Konfiguration berücksichtigt werden sollen.
Nocken	Hier können die Nockenschaltwerke angefügt werden.
E/A	Um die Steuerung mit der Prozessebene zu verbinden, sind entsprechende Interfacekarten und Schnittstellen notwendig. Unter diesem Eintrag ist aufgelistet, welche Karten verwendet werden.

SPS Konfiguration:

Damit TwinCAT auf die Variablen des Beispielprogramms Maschine zugreifen kann, muss das SPS-Projekt dem System Manager bekannt gegeben werden. Betätigen Sie dazu die rechte Maustaste, wenn sich der Mauszeiger über 'SPS - Konfiguration' befindet.



Wählen Sie dort 'IEC Projekt Anfügen'. Daraufhin öffnet sich folgendes Fenster, mit dem das SPS Projekt ausgewählt wird:

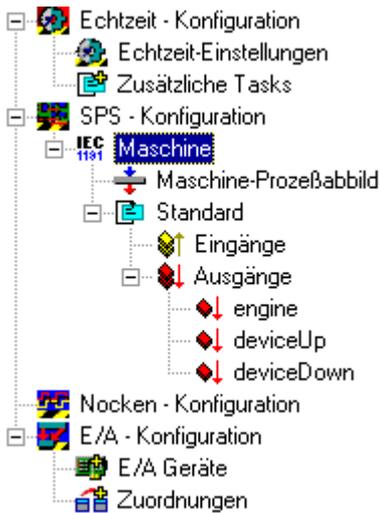


Wechseln Sie in das Verzeichnis '\TwinCAT\Samples\FirstSteps\' und wählen Sie die Datei 'Maschine.tpy' aus.

Unterhalb von 'SPS - Konfiguration' wurde ein weiterer Punkt ergänzt, der den Namen des SPS-Projektes trägt.



Ob ein Eintrag einen weiteren Unterpunkt enthält, erkennen Sie an den '-'/'+' Zeichen. Durch Anklicken dieser Symbole öffnen bzw. schließen sich die darunter liegenden Einträge. Wenn Sie den Baum so weit wie möglich öffnen, erhalten Sie folgende Struktur:



E/A Konfiguration:

Die E/A Geräte werden auf die gleiche Art wie die SPS Konfiguration angefügt.
 Wenn sich der Mauszeiger über E/A-Geräte befindet, können Sie mit der rechten Maustaste ein Kontextmenü öffnen, aus dem Sie den Eintrag 'Gerät Anfügen' auswählen müssen.



Gerät auswählen:

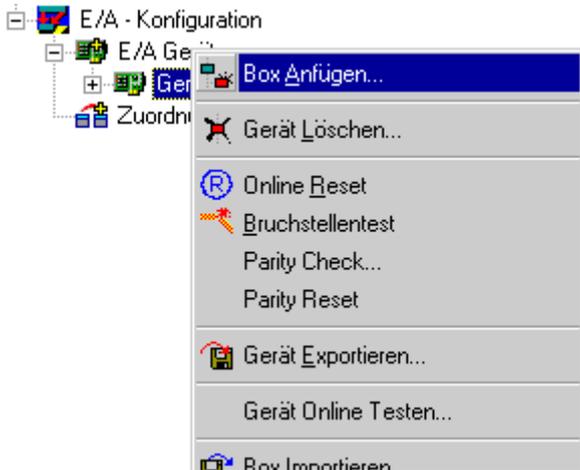
Daraufhin öffnet sich das untere Fenster. Wählen Sie den Gerätetyp FC200x, PCI aus. Der Gerätenamen ist frei wählbar.



Nach dem Einfügen des Geräts öffnet sich ein Dialogfeld, in dem die Konfiguration der Interfacekarte angegeben werden kann. Eine wichtige Einstellung ist z.B. unter dem Reiter ‚FC2001‘ die I/O-Adresse der Lightbus-Karte. Falls Sie an den Standardeinstellungen der Karte nichts geändert haben, können Sie die vorgegebenen Angaben übernehmen.

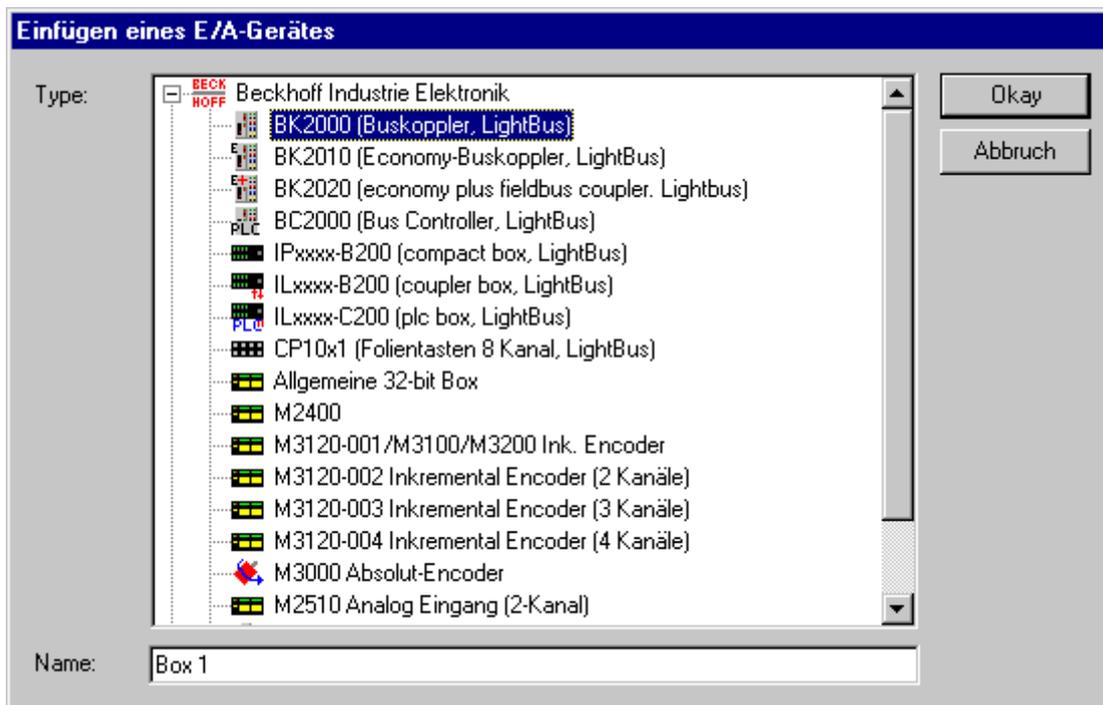
Buskoppler hinzufügen:

Öffnen Sie das Kontextmenü von der FC2001-Karte (Gerät 1) und wählen Sie den Befehl 'Box Anfügen...!'.



Buskoppler auswählen:

Wählen Sie den Buskoppler BK2000 aus und bestätigen Sie mit 'OK'. Der Name des Feldbusmoduls ist frei wählbar.



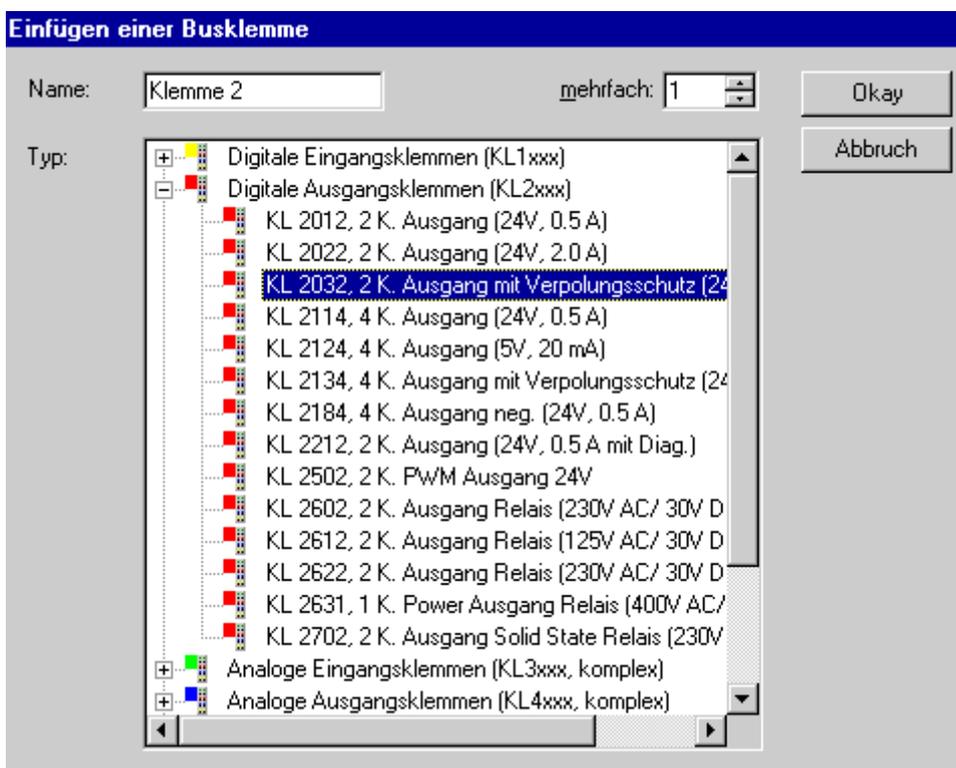
Busklemme anfügen:

Öffnen Sie das Kontextmenü des Buskopplers (BK2000) und wählen Sie den Befehl 'Klemme Anfügen'



Busklemme auswählen:

Wählen Sie die Busklemmen 2032 aus. Dazu gehen Sie mit dem Mauszeiger auf das '-' Zeichen und schließen mit einem Klick die Auswahl der digitalen Eingangsklemmen. Durch einen Klick auf das '+' Zeichen der digitalen Ausgangsklemmen öffnen Sie die neue Auswahl und können diese, mit einem Klick auf die gewünschte Busklemme KL2032, auswählen. Bestätigen Sie Ihre Wahl mit 'OK'.



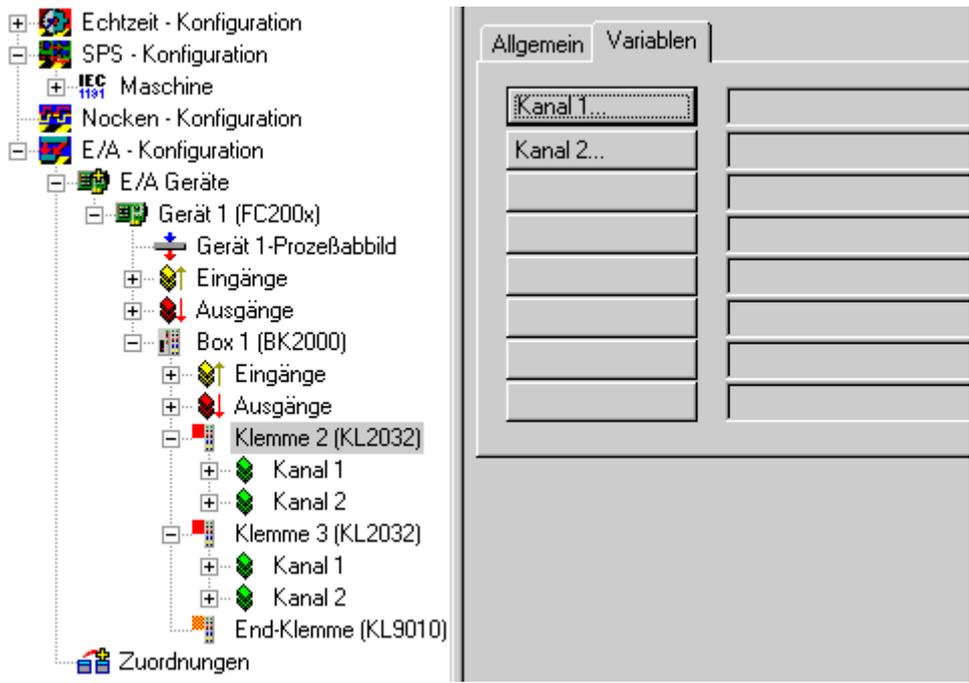
Der Name der Busklemme ist ebenfalls frei wählbar.

Da für das Beispielprogramm 3 digitale Ausgänge erforderlich sind, muss noch eine zweite KL2032 eingefügt werden. Dazu wiederholen Sie bitte die einzelnen Schritte.

Die Busendklemme KL9010 wird vom System Manager automatisch eingefügt.

Ende der Konfiguration:

Nach Beendigung der Konfiguration sollte folgender Aufbau zu sehen sein:

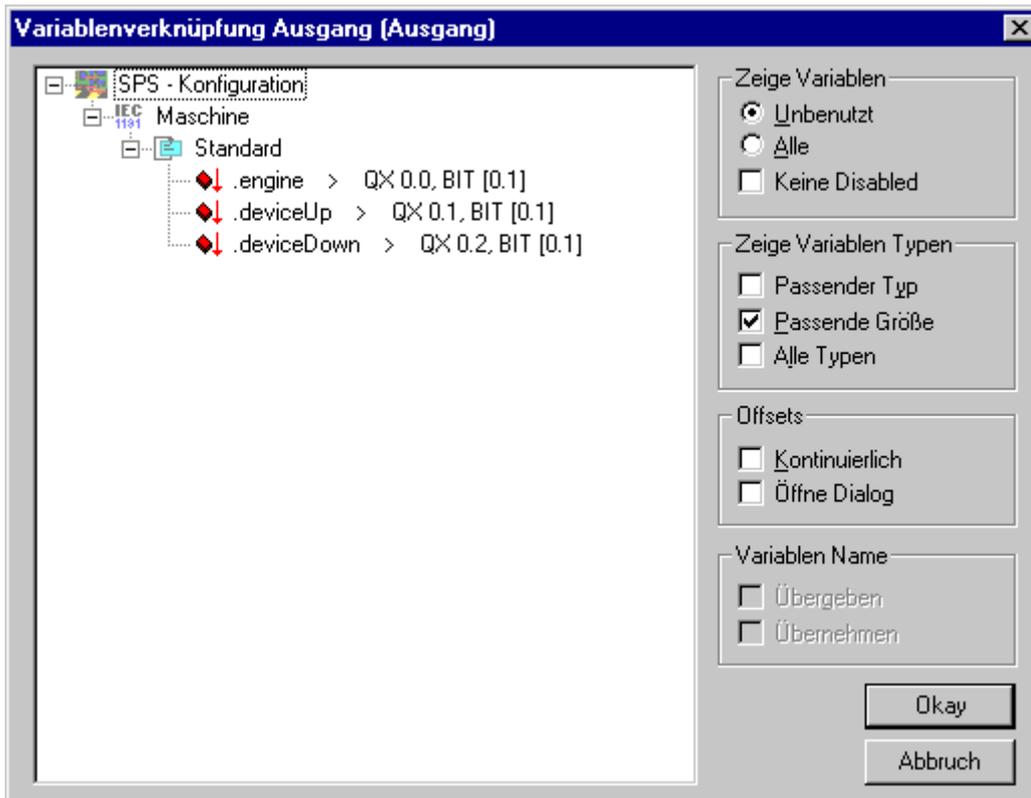


Sie können die Standardbezeichnungen (Gerät 1, Box 1, Klemme 2, usw.) natürlich umbenennen. Mit einem langsamen Doppelklick auf den entsprechenden Namen kann eine neue Bezeichnung vergeben werden.

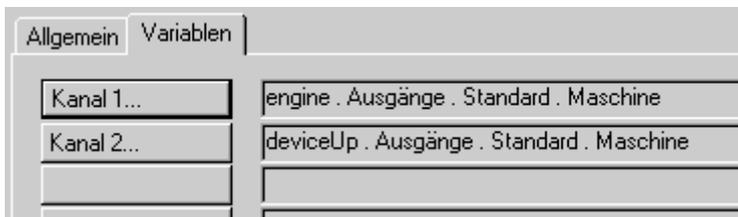
Variablen den Aus-/Eingangskanälen zuweisen

Bis zu dieser Stelle wurde die komplette Hardware, die für das obige Beispielprogramm benötigt wird, konfiguriert. Als nächstes müssen die einzelnen Variablen aus dem SPS-Projekt den einzelnen Ein-/Ausgangskanälen zugewiesen werden. Markieren Sie dazu die Klemme, die Sie konfigurieren wollen. Bei der Klemme 2 (2 digitale Ausgänge) öffnet sich auf der rechten Seite ein Dialogfenster mit den Reitern ‚Allgemein‘ und ‚Variablen‘ (siehe Bild oben). Wählen Sie den Reiter ‚Variablen‘ aus.

Sie sehen oben die 2 Ausgangskanäle aufgelistet. Beide Kanäle sind noch nicht belegt. Um den Kanal 1 zu konfigurieren, wählen Sie den entsprechenden Auswahlknopf (Kanal1). Es öffnet sich folgender Dialog:



In diesem Dialogfenster sind jetzt alle Ausgangsvariablen aufgelistet. Wählen Sie die erste Variable (engine) aus und bestätigen Sie Ihre Eingabe mit 'Okay'. Kanal2 belegen Sie mit der Variablen device.Up. Die erste Busklemme ist somit verknüpft:



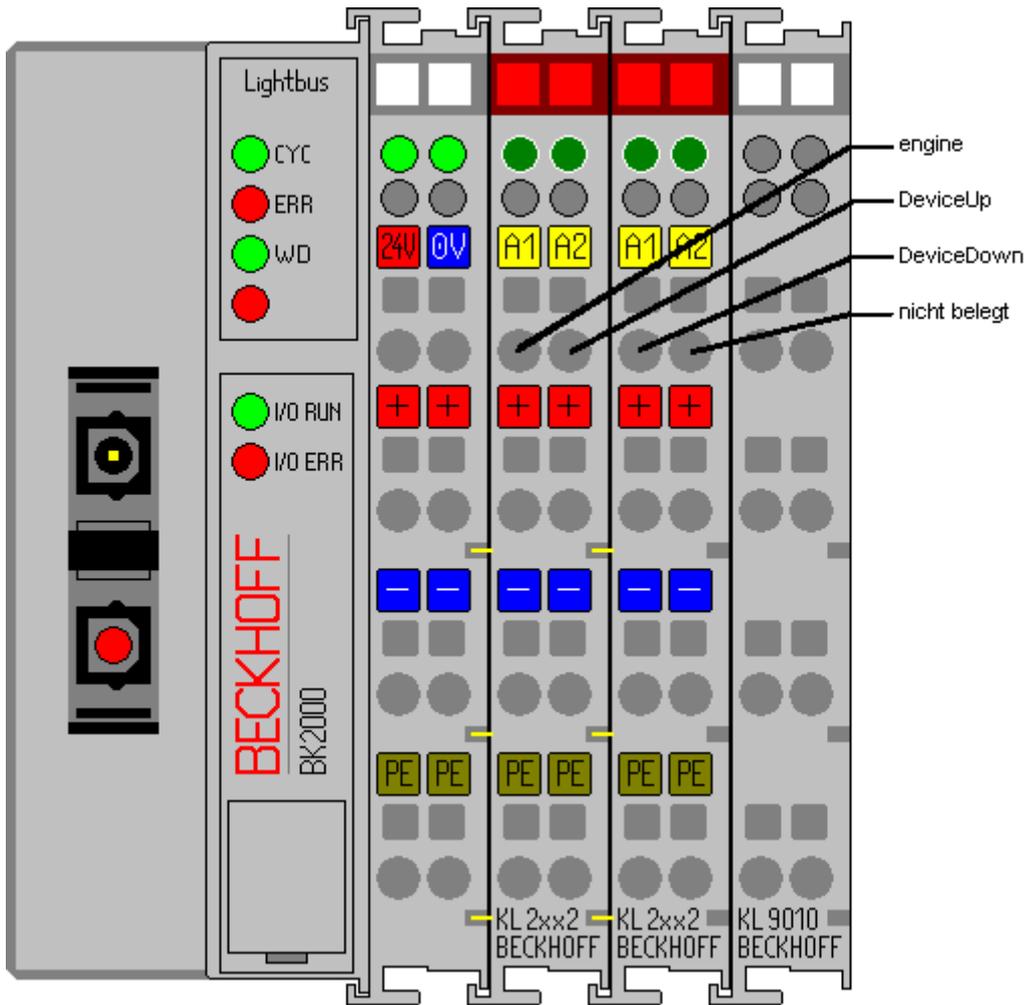
Die zweite Busklemme wird ebenso belegt, Kanal1 wird device.Down zugeordnet. Der Kanal2 dieser Klemme bleibt unbelegt.

Variablenverknüpfung:

Klemme2	SPS Variablen	Bedeutung
Kanal1(=Ausgang 1)	Engine	Ansteuerung Schrittmotor
Kanal2 (=Ausgang 2)	DeviceUp	Ansteuerung Bohrer hochfahren

Klemme3	SPS Variablen	Bedeutung
Kanal1 (=Ausgang 3)	DeviceDown	Ansteuerung Bohrer runterfahren

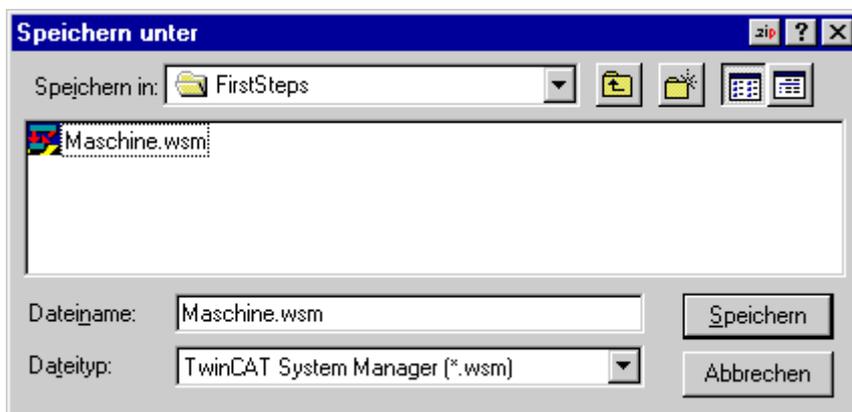
Belegung der Busklemmen:



Projekt speichern:



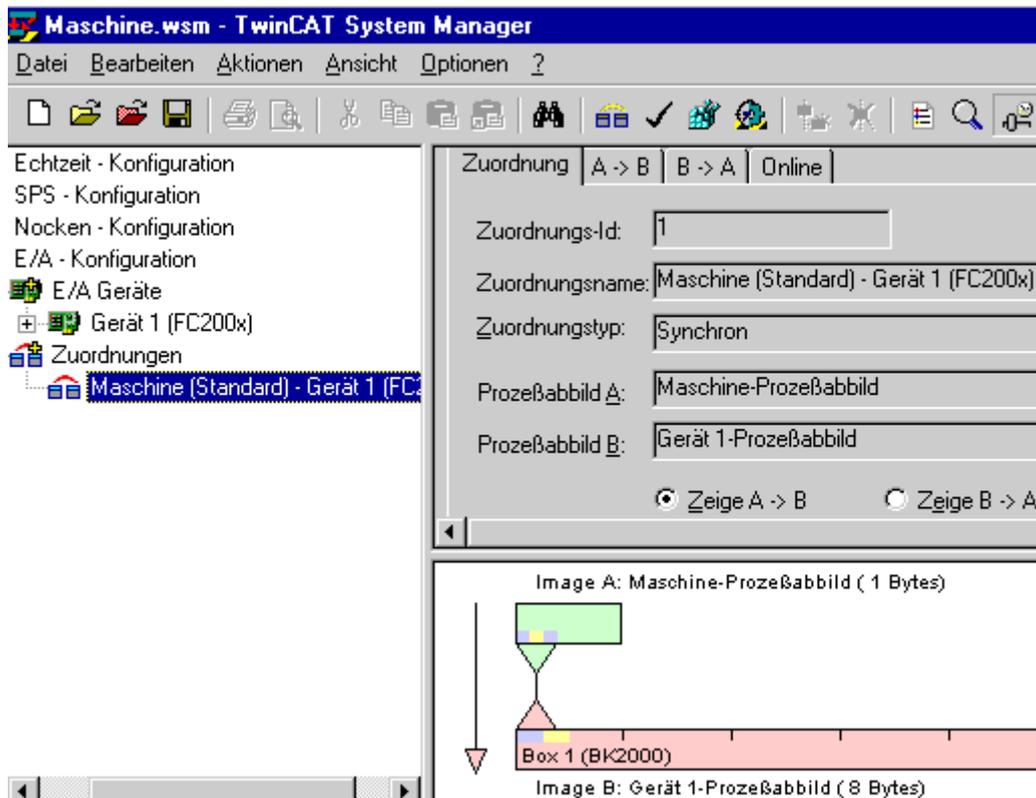
Sie sollten an dieser Stelle die Konfiguration abspeichern, damit Sie später wieder darauf zugreifen können. Führen Sie dazu aus dem Menü 'Datei' den Befehl 'Speichern unter...' aus.



Zuordnung erzeugen:

Sie haben jetzt die gesamte System-Konfiguration für das obige Beispielprogramm durchgeführt. Sie müssen jetzt die Zuordnung erzeugen. Gehen Sie dazu auf den Befehl 'Zuordnung erzeugen' im Menü 'Aktionen'.

Unterhalb des Baumeintrags 'Zuordnungen' sehen Sie jetzt 'Maschine (Standard) - Gerät 1 (FC2001)'. Klicken Sie diesen Eintrag an. Daraufhin öffnet sich folgendes Fenster:



Im Dialogfenster können Sie festlegen, ob der Datenfluss von A nach B oder von B nach A angezeigt werden soll. Hierbei entspricht das Image A dem Prozessabbild der SPS-Variablen, also den Ein-/Ausgangsvariablen. Image B enthält das Prozessabbild der E/A-Geräte, in diesem Fall vom Buskoppler BK2000. Jede Variable oder Busklemme ist farblich im Prozessabbild hervorgehoben. Wenn Sie mit der Maus auf eine dieser Flächen stehen bleiben, erscheint ein kleines Anzeigefeld, in dem die genaue Bezeichnung dargestellt wird.

Konfiguration in Registry schreiben:

Als letzter Schritt muss die Konfiguration in die Registry von Windows NT/2000/XP gespeichert werden, da beim Starten von TwinCAT die dort abgespeicherten Informationen ausgewertet werden. Führen Sie aus dem Menü 'Aktionen' den Befehl 'Sichern in Registry...' aus. Falls eine ältere Konfiguration dort schon abgespeichert ist, erscheint eine Sicherheitsabfrage, die Sie bestätigen können.

TwinCAT neu starten:

Damit TwinCAT die Änderungen übernimmt, muss das System neu gestartet werden.



An den Busklemmen KL2032 werden jetzt die einzelnen SPS-Variablen ausgegeben. Dies ist am Leuchten der LED an den Busklemmen zu erkennen.

7.3.3 Ethernet - Einrichten der Busklemmen

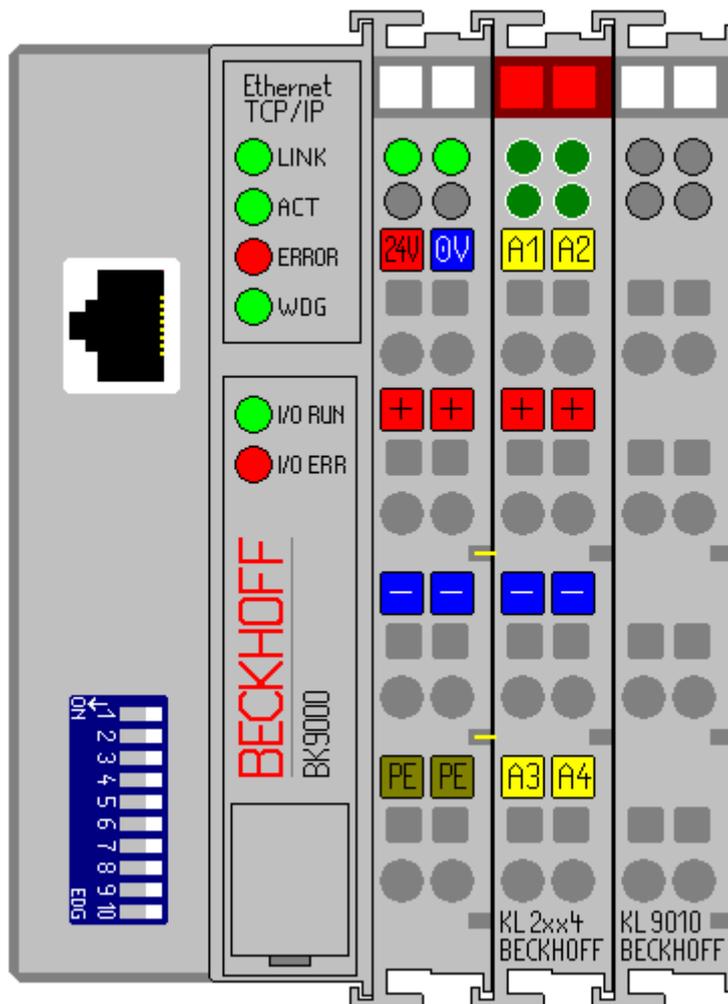
Benötigte Hardware:

- Ethernet-Buskoppler (BK9000)
- 1 Busklemme mit 4 digitalen Ausgängen (KL2114)
- Busendklemme (KL9010)

Dieses Beispiel ist genau auf den Ethernet Busstation ausgelegt. Die Hardware ist jedoch austauschbar. Dabei ist die Konfiguration der E/A-Geräte entsprechend zu ändern.

Aufbau des Buskopplers und der Busklemmen

Bauen Sie den Buskoppler und die Busklemmen auf, wie in der unteren Zeichnung dargestellt:



Hardware Konfiguration

Nähere Informationen zu Hardware-Anschluss und -Einstellung entnehmen Sie bitte der gesonderten Hardware Dokumentation des Buskopplers.

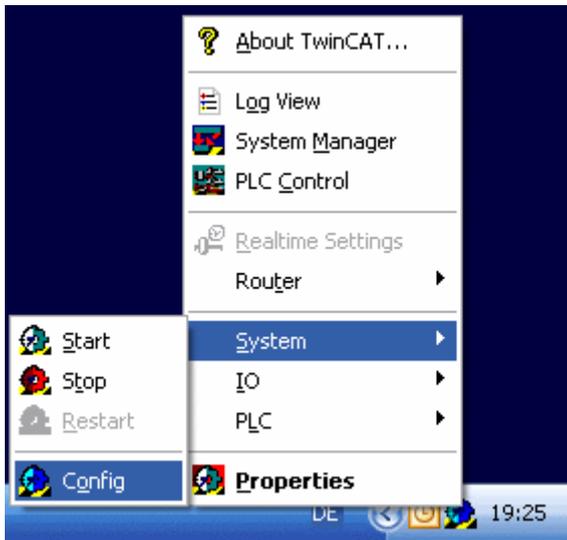
TwinCAT-SPS-Konfiguration

Es muss die erweiterte Version des 'Machine'-Projektes (C:\TwinCAT\Samples\FirstSteps\TwinCAT\Machine_Final.pro) auf der 1. Laufzeit der lokalen SPS eingerichtet sein. 'Machine_Final' entspricht den Änderungen die in [diesem Kapitel \[▶ 27\]](#) vorgenommen wurden.

Download SPS Beispiele: <https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281688587.zip>

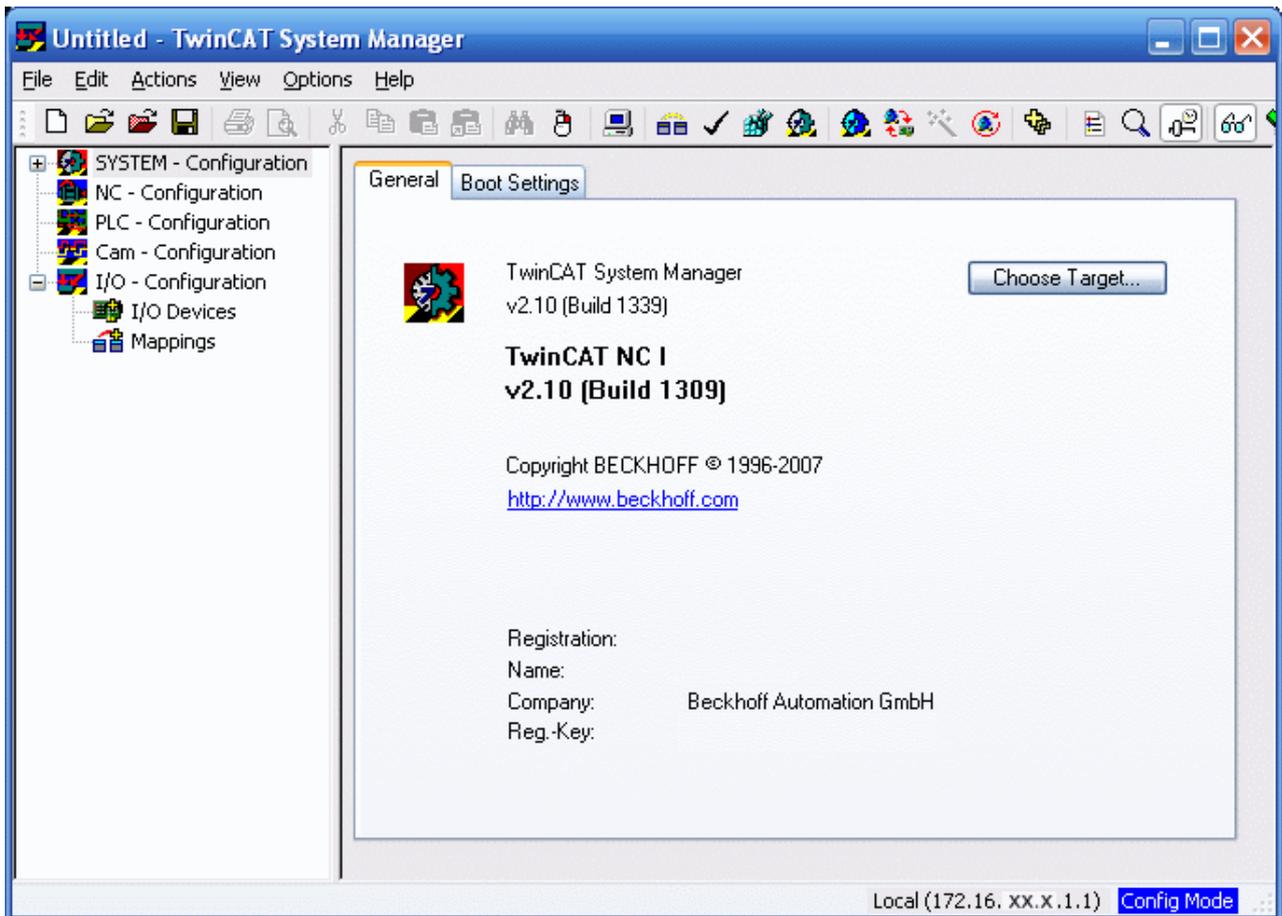
TwinCAT im Config-Modus starten

Klicken Sie in der Taskleiste rechts unten auf das TwinCAT-Symbol, danach auf System/Config. Damit wird TwinCAT im Config-Modus gestartet, der unter anderem das Einscannen der Boxen ermöglicht.



TwinCAT System Manager starten

Nachdem das System im Config-Modus gestartet wurde, ändert sich die Farbe des Icons auf blau. Starten Sie jetzt den TwinCAT System Manager über 'Start'-> 'Programme' -> 'TwinCAT System' -> 'TwinCAT System Manager'.



Die Elemente des TwinCAT System Managers

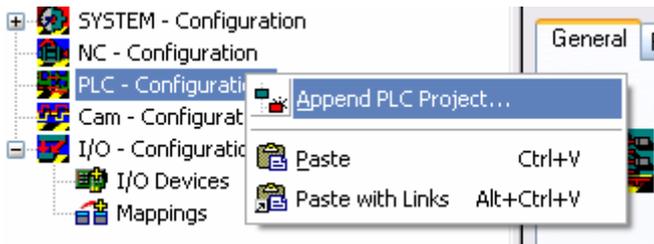
In der obersten Zeile steht der Name des Projektes, hier noch unbenannt, darunter befindet sich die Menüleiste und die Toolbar. Der untersten Zeile ist der Status des Systems zu entnehmen, in diesem Fall der Config-Modus. Die beiden großen Fenster in der Mitte enthalten die Konfiguration des Systems. Nachfolgend wird mit dem Beispielprogramm diese Konfiguration durchgeführt.

Im linken Fenster ist die Systemkonfiguration als Baumstruktur dargestellt. Sie besteht aus den fünf Hauptpunkten:

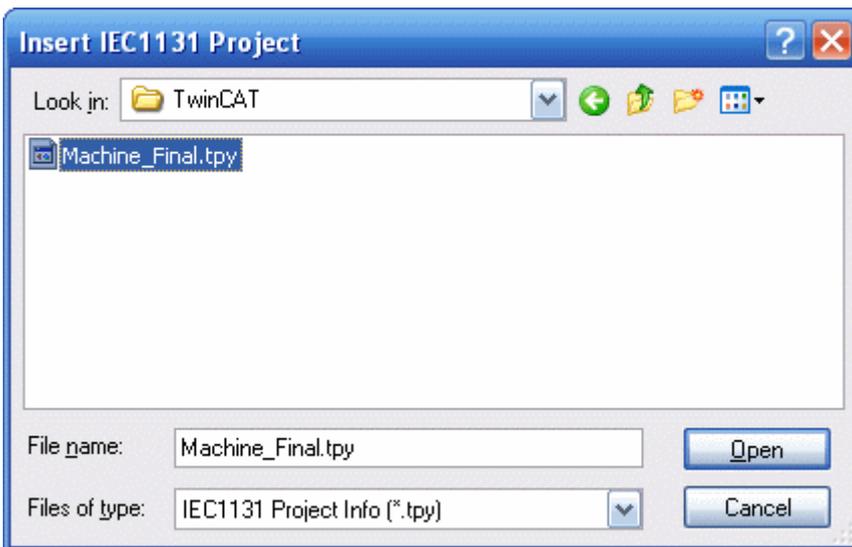
Konfiguration	Bedeutung
System	Konfiguration des Systems und der Echtzeitparameter
NC	(Optional) Hier können NC-Tasks eingebunden und verwaltet werden.
SPS	Unter diesem Eintrag werden sämtliche SPS-Projekte aufgelistet, die bei der Konfiguration berücksichtigt werden sollen.
Nocken	(Optional) Hier können die Nockenschaltwerke angefügt werden.
E/A	Um die Steuerung mit der Prozessebene zu verbinden, sind entsprechende Interfacekarten und Schnittstellen notwendig. Unter diesem Eintrag ist aufgelistet, welche Karten verwendet werden.

SPS-Konfiguration

Damit TwinCAT auf die Variablen des Beispielprogramms 'Machine_Final' zugreifen kann, muss das SPS-Projekt dem System Manager bekannt gegeben werden. Betätigen Sie dazu die rechte Maustaste, wenn sich der Mauszeiger über 'SPS - Konfiguration' befindet.

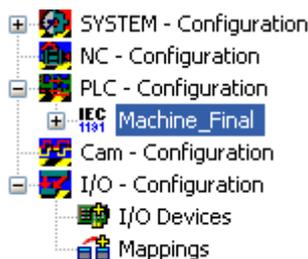


Wählen Sie dort 'SPS Projekt Anfügen'. Daraufhin öffnet sich folgendes Fenster, mit dem das SPS Projekt ausgewählt wird:

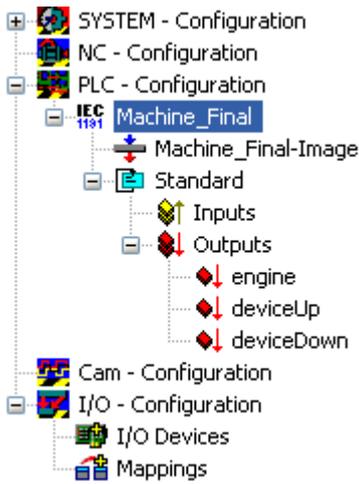


Wechseln Sie in das Verzeichnis '\TwinCAT\Samples\FirstSteps\TwinCAT\' und wählen Sie die Datei 'Machine_Final.tpy' aus.

Unterhalb von 'SPS - Konfiguration' wurde ein weiterer Punkt ergänzt, der den Namen des SPS-Projektes trägt.



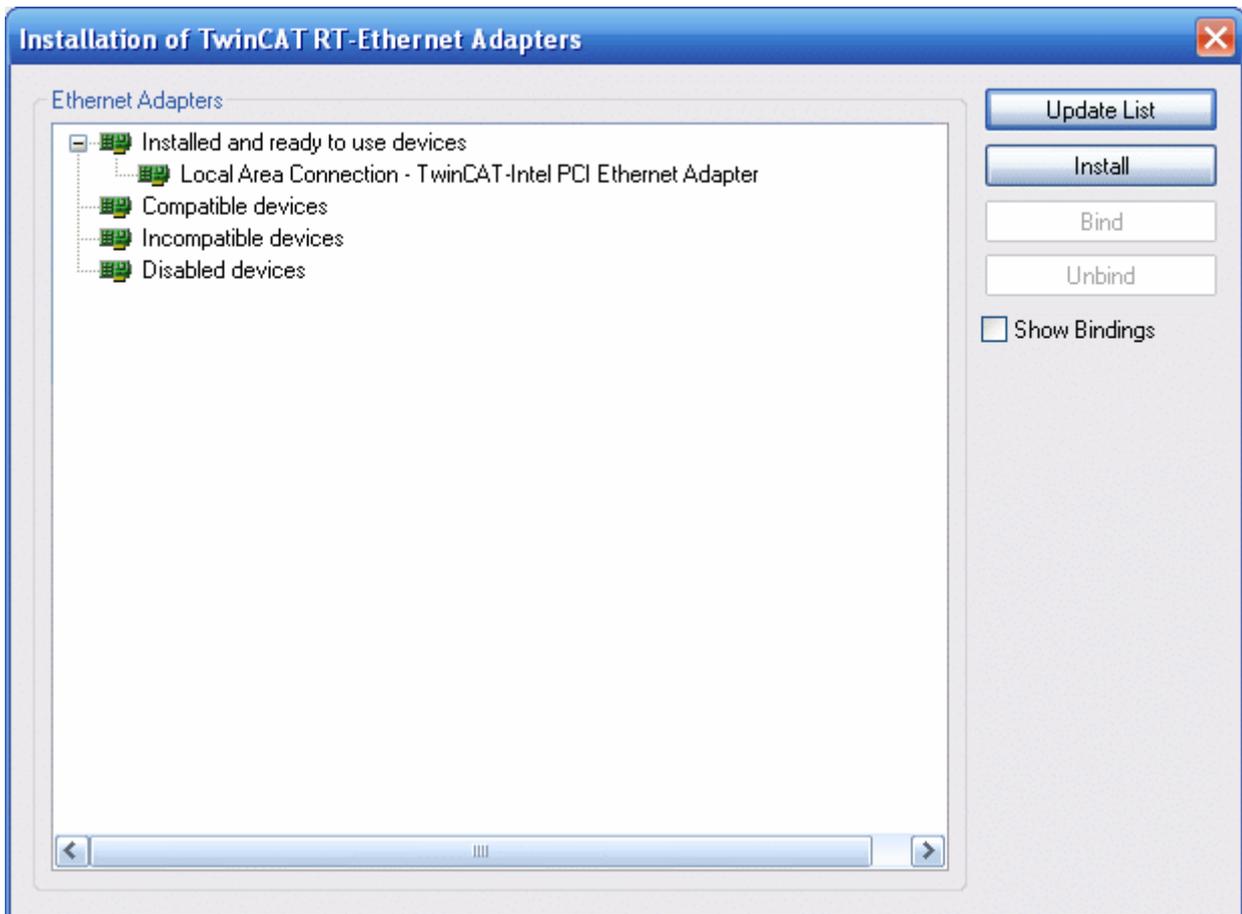
Ob ein Eintrag einen weiteren Unterpunkt enthält, erkennen Sie an den '-'/'+'-Zeichen. Durch Anklicken dieser Symbole öffnen bzw. schließen sich die darunter liegenden Einträge. Wenn Sie den Baum so weit wie möglich öffnen, erhalten Sie folgende Struktur:



Einrichten des Echtzeit-Ethernets

Bevor Sie Echtzeit-Ethernet-fähige Geräte (BK9000) verwenden, muss ihre Netzwerkkarte mit entsprechenden Treibern versehen werden.

Gehen Sie hierfür in der Menüleiste auf 'Optionen/Liste Echtzeit-Ethernet kompatible Geräte...'



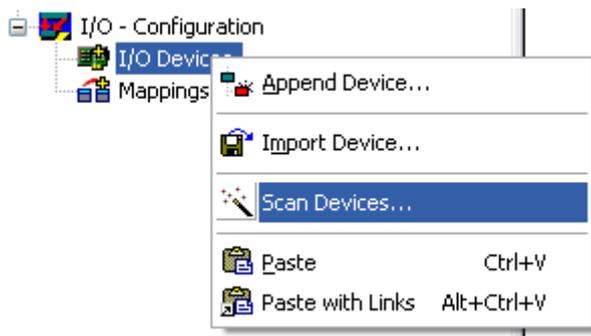
Im darauf folgenden Dialog gibt es vier Unterpunkte:

Installed and ready to use devices	Hier aufgeführte Netzwerkverbindungen sind Echtzeit-Ethernet-fähig und können verwendet werden
Compatible devices	Hier aufgeführte Netzwerkverbindungen sind prinzipiell Echtzeit-Ethernet-fähig, benötigen jedoch noch die entsprechenden Treiber. Klicken sie hierfür auf die Schaltfläche 'Install'. Läuft die Installation erfolgreich so wird die Netzwerkverbindungen zukünftig unter 'Installed and ready to use devices' geführt
Incompatible devices	Hier aufgeführte Netzwerkverbindungen sind nicht Echtzeit-Ethernet-fähig.
Disabled devices	Hier aufgeführte Netzwerkverbindungen sind deaktiviert und stehen momentan nicht zur Verfügung.

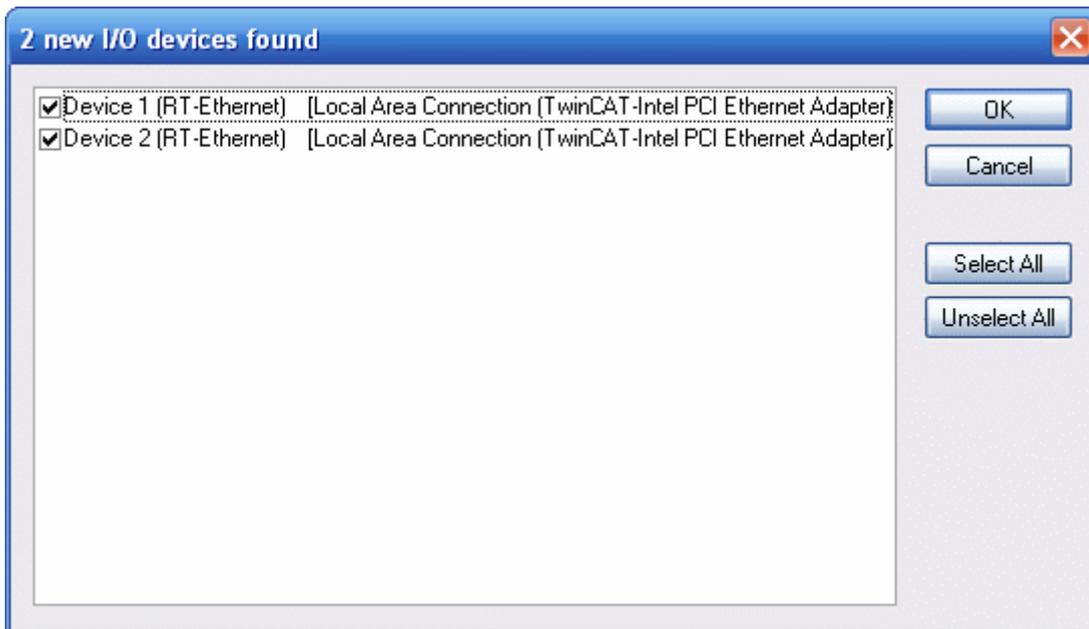
Sie benötigen mindestens eine Netzwerkverbindung unter 'Installed and ready to use devices' um fortzufahren.

E/A-Konfiguration: Gerät hinzufügen

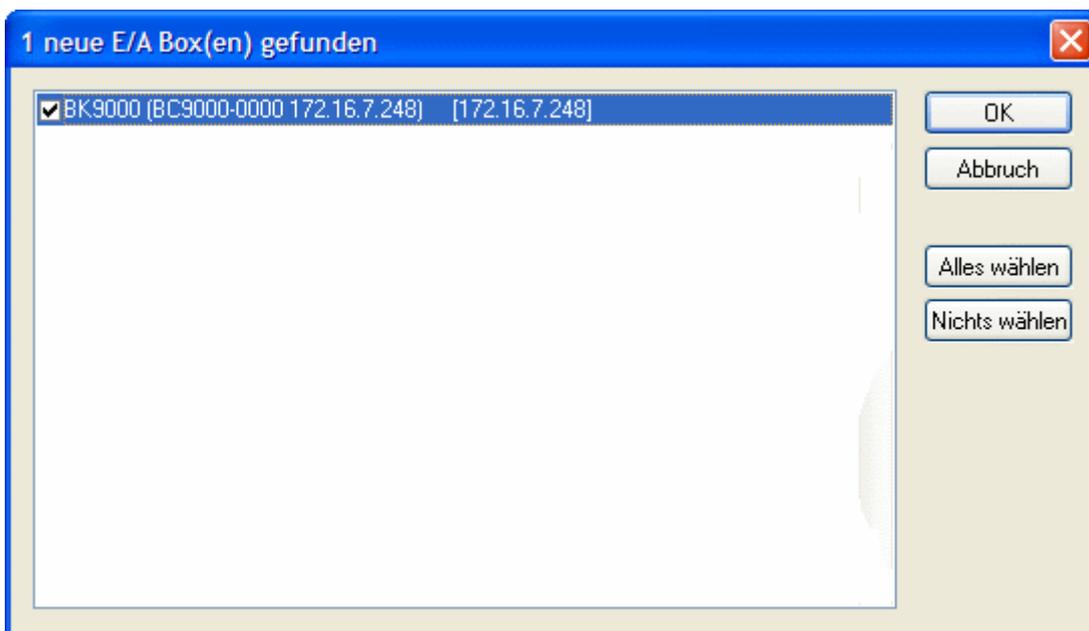
Die E/A Geräte werden auf die gleiche Art wie die SPS Konfiguration angefügt. Wenn sich der Mauszeiger über E/A-Geräte befindet, können Sie mit der rechten Maustaste ein Kontextmenü öffnen, aus dem Sie den Eintrag 'Geräte suchen...' auswählen müssen. Bestätigen Sie den Hinweis mit 'OK'.



Daraufhin öffnet sich das untere Fenster, welches alle gefundenen Geräte auflistet. Wählen Sie das 'RT-Ethernet'-Gerät aus. Sollten mehrere Geräte dieses Typs hier aufgelistet werden, so wählen Sie bitte das Gerät, welches der Netzwerkverbindung mit dem BK9000 entspricht.



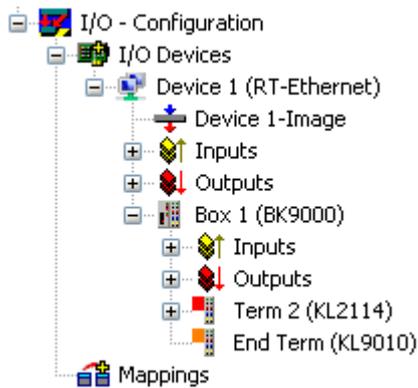
In anschließenden Dialog muss die Suche nach neuen Boxen mit 'OK' bestätigt werden.



Wählen Sie Ihren Buskoppler BK9000 aus und bestätigen Sie mit 'OK'. Aktivieren Sie nicht den 'Free Run'.

Erweitern Sie die Baumansicht durch Klicken auf das '+'-Symbol links daneben. Wie zu erkennen ist, wurden die an den Buskoppler angeschlossenen Klemmen automatisch erkannt und hinzugefügt.

Dies sollte mit dem Bild unten vergleichbar sein:

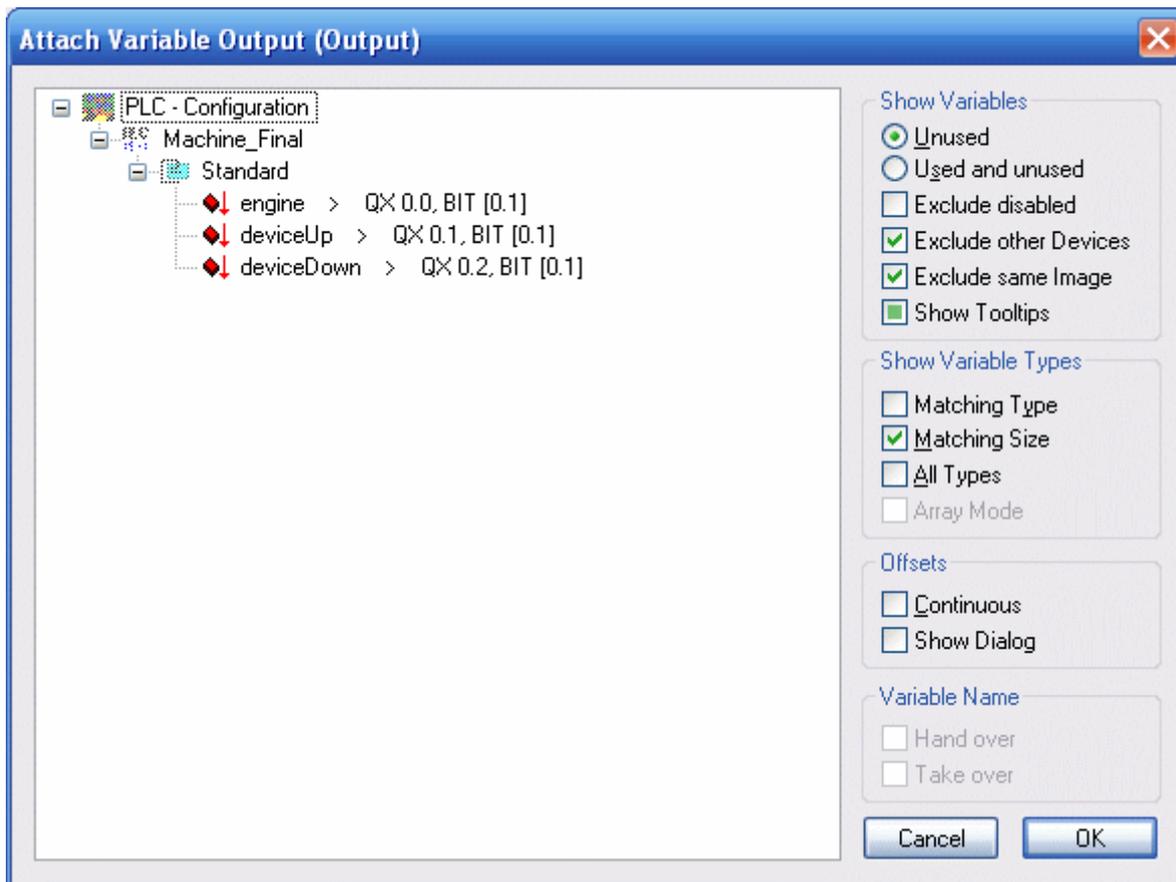


Der Name der Standardbezeichnungen (Gerät 1, Box 1, Klemme 1, usw.) ist frei wählbar. Mit einem langsamen Doppelklick auf den entsprechenden Namen kann eine neue Bezeichnung vergeben werden.

Variablen den Aus-/Eingangskanälen zuweisen

Bis zu dieser Stelle wurde die komplette Hardware, die für das obige Beispielprogramm benötigt wird, konfiguriert. Als nächstes müssen die einzelnen Variablen aus dem SPS-Projekt den einzelnen Ein-/Ausgangskanälen zugewiesen werden. Markieren Sie dazu die Klemme, die Sie konfigurieren wollen. Bei der Klemme 1 (4 digitale Ausgänge) öffnet sich auf der rechten Seite ein Dialogfenster mit den Reitern ‚Allgemein‘ und ‚Variablen‘. Wählen Sie den Reiter ‚Variablen‘ aus.

Diese 4 Ausgangskanäle sind noch nicht belegt. Um den Kanal 1 zu konfigurieren, wählen Sie den entsprechenden Auswahlknopf (‘Kanal 1‘). Es öffnet sich folgender Dialog:



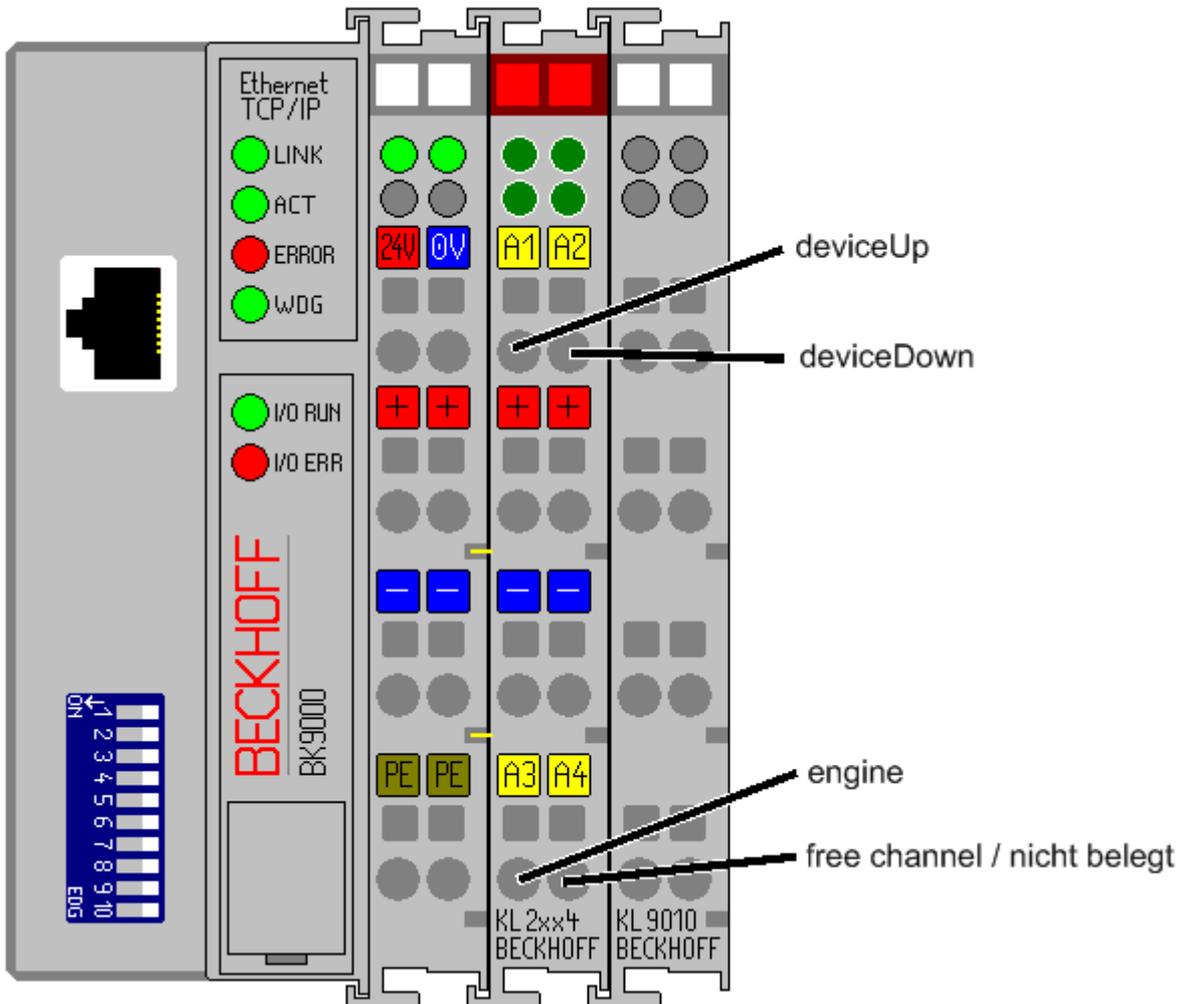
In diesem Dialogfenster sind jetzt alle Ausgangsvariablen aufgelistet. Wählen Sie die Variable 'deviceUp' aus und bestätigen Sie Ihre Eingabe mit 'OK'. Kanal 2 belegen Sie mit der Variablen 'deviceDown', Kanal 3 mit 'engine' und Kanal 4 bleibt unbelegt.

Die Busklemme ist somit verknüpft.

Variablenverknüpfung

Klemme 2	SPS Variablen	Bedeutung
Kanal 1 (=Ausgang 1)	DeviceUp	Ansteuerung Bohrer hochfahren
Kanal 2 (=Ausgang 2)	DeviceDown	Ansteuerung Bohrer herunterfahren
Kanal 3(=Ausgang 3)	Engine	Ansteuerung Schrittmotor

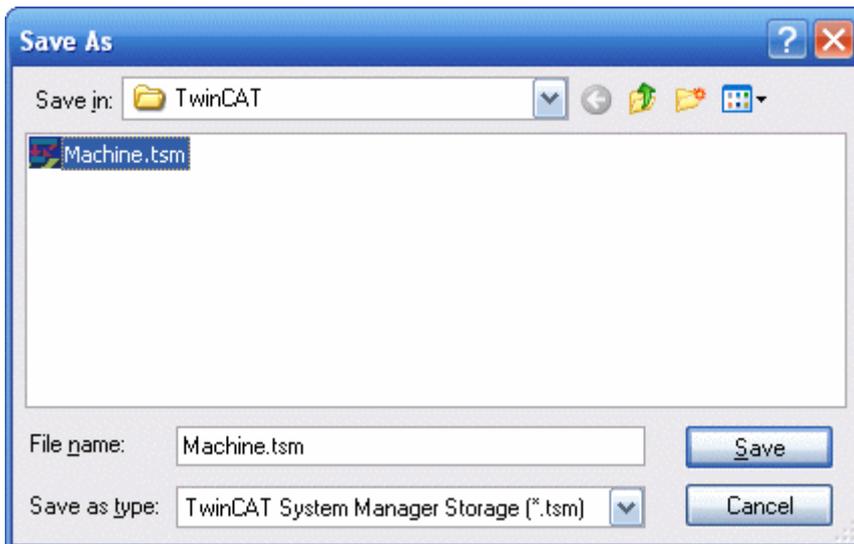
Belegung der Busklemmen:



Aktivierung des Projektes: Projekt speichern



Sie sollten an dieser Stelle die Konfiguration abspeichern, damit Sie später wieder darauf zugreifen können. Führen Sie dazu aus dem Menü 'Datei' den Befehl 'Speichern unter...' aus.

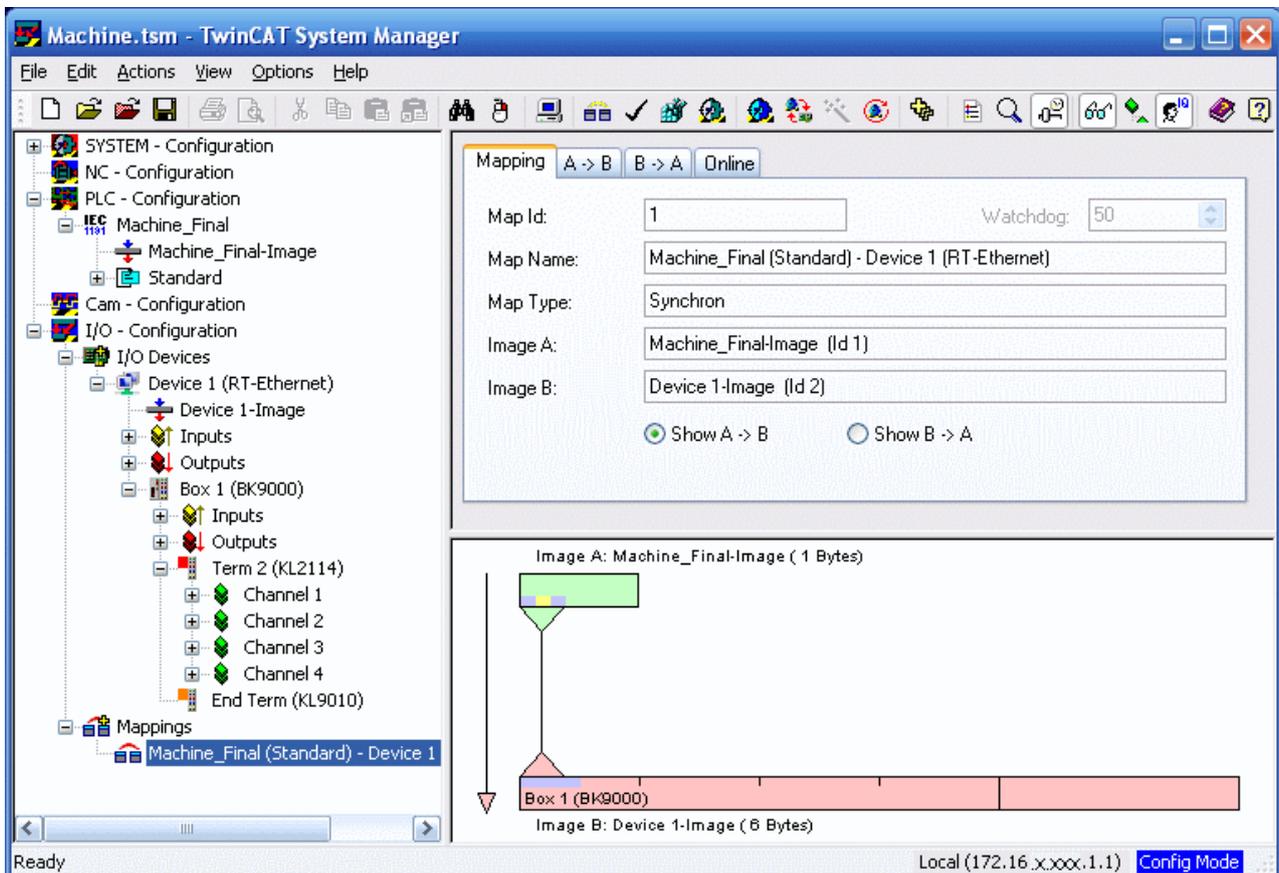


Zuordnung erzeugen:



Sie haben jetzt die gesamte System-Konfiguration für das obige Beispielprogramm durchgeführt. Sie müssen jetzt die Zuordnung erzeugen. Gehen Sie dazu auf den Befehl 'Zuordnung erzeugen' im Menü 'Aktionen'.

Unterhalb des Baumeintrags 'Zuordnungen' sehen Sie jetzt 'Machine (Standard) - Gerät 1 (RT-Ethernet)'. Klicken Sie diesen Eintrag an. Daraufhin öffnet sich folgendes Fenster:



Im Dialogfenster können Sie festlegen, ob der Datenfluss von A nach B oder von B nach A angezeigt werden soll. Hierbei entspricht das Image A dem Prozessabbild der SPS-Variablen, also den Ein-/Ausgangsvariablen. Image B enthält das Prozessabbild der E/A-Geräte, in diesem Fall vom Buskoppler BK9000. Jede Variable oder Busklemme ist farblich im Prozessabbild hervorgehoben. Wenn Sie mit der Maus auf eine dieser Flächen stehen bleiben, erscheint ein kleines Anzeigefeld, in dem die genaue Bezeichnung dargestellt wird.

Konfiguration in Registry schreiben



Als letzter Schritt muss die Konfiguration in die Registry von Windows NT/2000/XP gespeichert werden, da beim Starten von TwinCAT die dort abgespeicherten Informationen ausgewertet werden. Führen Sie aus dem Menü 'Aktionen' den Befehl 'Sichern in Registry...' aus. Falls dort bereits eine ältere Konfiguration abgespeichert ist, erscheint eine Sicherheitsabfrage, die Sie bestätigen können.

TwinCAT neu starten

Damit TwinCAT die Änderungen übernimmt, muss das System im 'Run'-Modus neu gestartet werden. Bestätigen Sie eine diesbezügliche Abfrage des System Managers mit 'OK'. Andernfalls verfahren Sie wie zu Beginn des Kapitels.

Sie müssen nun ggf. das SPS-Projekt 'Machine_Final.pro' erneut einlesen.

An den Busklemmen KL2404 werden jetzt die einzelnen SPS-Variablen ausgegeben. Dies ist am Leuchten der LEDs an den Busklemmen zu erkennen.

Weitere Informationen zum TwinCAT System Manager entnehmen Sie bitte der Dokumentation TwinCAT System Manager aus dem Beckhoff Information System.

7.3.4 EtherCAT - Einrichten der Busklemmen

Benötigte Hardware für die Beispielkonfiguration:

- Embedded PC CX1020
 - CPU Grundmodul C1020-0123: 512 MB Flash-Speicher, WindowsXP embedded
 - Systemschnittstelle CX1020-N010: eine DVI-I und zwei USB Schnittstellen.
 - Netzteil CX1100-0004 mit EtherCAT-Interface
- oder einen konventionellen IPC mit Windows 2000, XP oder Vista und mindestens einem freien Ethernet Port.
 - ein EK1100 EtherCAT Buskoppler
- EtherCAT I/O-Klemmen:
 - EL2004: 4-fach Digital-OUT, 24V DC, 0,5A, typische Schaltzeit 90 µs
 - evtl. sind weitere I/O-Klemmen vorhanden

Dieses Beispiel beschreibt den Vorgang an einem Embedded PC CX1020. Es ist aber auf jeden anderen PC (dann in Verbindung mit einem EtherCAT Koppler EK1100) übertragbar.

Aufbau der Klemmen:

Bauen Sie den CX1020 auf, wie in der unteren Zeichnung dargestellt. Die Abbildung zeigt eine Beispielkonfiguration.



Versorgen Sie den CX1020 mit 24 V.

Hardware Dokumentation

Nähere Informationen zum Hardware Anschluss entnehmen Sie bitte der gesonderten Hardware Dokumentation.

TwinCAT Realtime Server starten

Starten Sie nun, falls noch nicht geschehen, den TwinCAT Realtime Kernel im 'Konfig-Modus', damit der TwinCAT Message Router aktiv ist.



TwinCAT System Manager starten

Nachdem das System gestartet wurde, ändert sich die Farbe des Icons von rot auf blau. Starten Sie jetzt den TwinCAT System Manager.



Die Elemente des TwinCAT System Manager



In der obersten Zeile steht der Name des Projektes, hier noch unbenannt, darunter befindet sich die Menüleiste und die Toolbar. Der untersten Zeile ist der Status des Systems zu entnehmen, in diesem Bild läuft das System (zu erkennen an der Einstellung 'Echtzeit'). Die beiden großen Fenster in der Mitte enthalten die Konfiguration des Systems. Nachfolgend wird mit dem Beispielprogramm diese Konfiguration durchgeführt.

Im linken Fenster ist die Systemkonfiguration als Baumstruktur dargestellt. Sie besteht aus den drei Hauptpunkten:

Menü	Einstellungen
System - Konfiguration	Konfiguration des Systems und der Echtzeitparameter
SPS - Konfiguration	Konfiguration der SPS-Tasks
E/A - Konfiguration	Um die Steuerung mit der Prozessebene zu verbinden, sind entsprechende Schnittstellen notwendig. In diese Menü werden die Schnittstellen konfiguriert

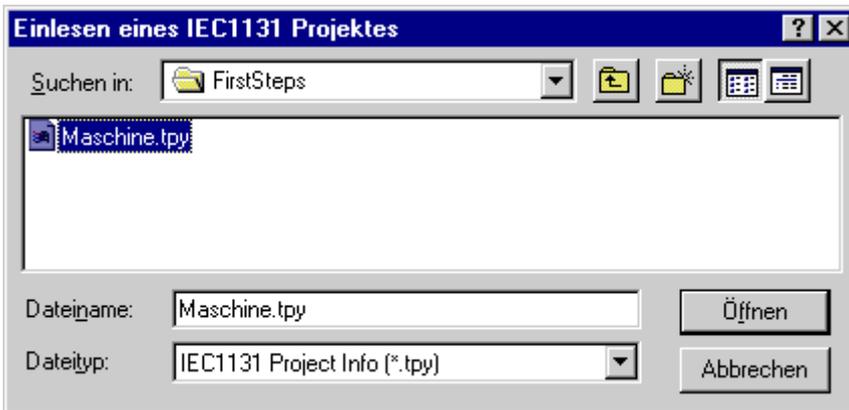
Ggf. sind noch weitere Systemkomponenten aufgelistet (z.B. Nocken - oder NC-Konfiguration).

SPS Konfiguration

Damit TwinCAT auf die Variablen des Beispielprogramms 'Maschine' zugreifen kann, muss das SPS-Projekt dem System Manager bekannt gegeben werden. Betätigen Sie dazu die rechte Maustaste, wenn sich der Mauszeiger über 'SPS - Konfiguration' befindet.



Wählen Sie dort 'IEC Projekt Anfügen'. Daraufhin öffnet sich folgendes Fenster, mit dem das SPS Projekt ausgewählt wird:

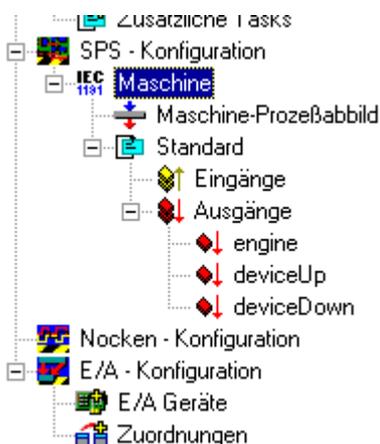


Wechseln Sie in das Verzeichnis '\TwinCAT\Samples\FirstSteps\' und wählen Sie das Beispielprogramm 'Maschine.tpy' aus.

Unterhalb von 'SPS - Konfiguration' wurde nun ein weiterer Punkt ergänzt, der den Namen des SPS-Projektes trägt.



Ob ein Eintrag einen weiteren Unterpunkt enthält, erkennen Sie an den -Zeichen und an den +Zeichen. Durch Anklicken dieser Symbole öffnen bzw. schließen sich die darunter liegenden Einträge. Wenn Sie den Baum so weit wie möglich öffnen, erhalten Sie folgende Struktur:



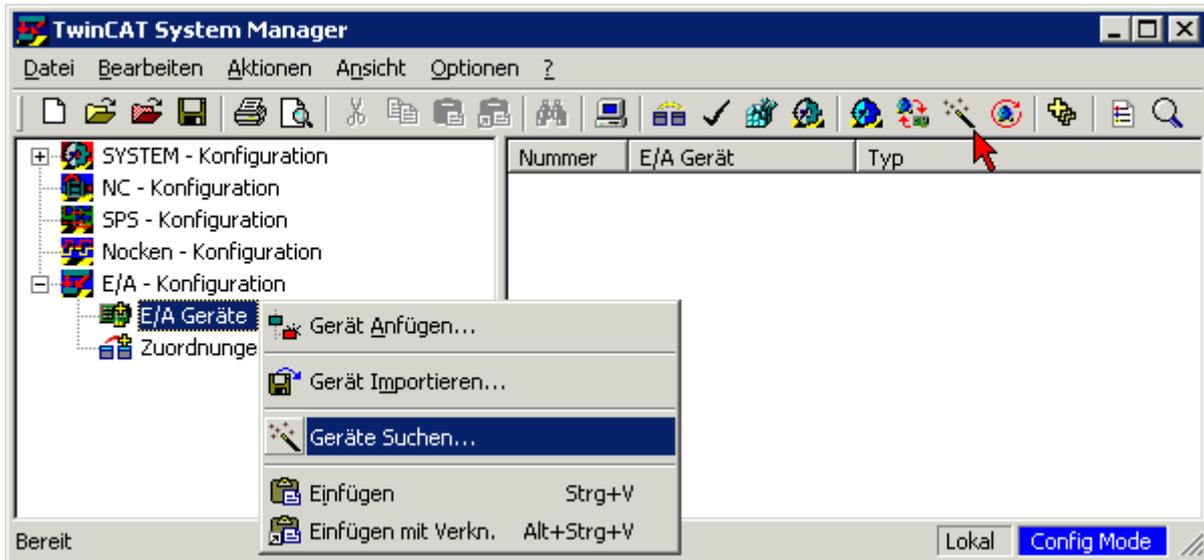
Im oberen Bild sehen Sie, dass Ihr Beispielprogramm eine Task namens 'Standard' enthält.

E/A Konfiguration

Gerät hinzufügen

Die E/A Geräte werden auf die gleiche Art wie die SPS Konfiguration angefügt. TwinCAT bietet Ihnen die Möglichkeit, die E/A-Komponenten automatisch zu einzulesen.

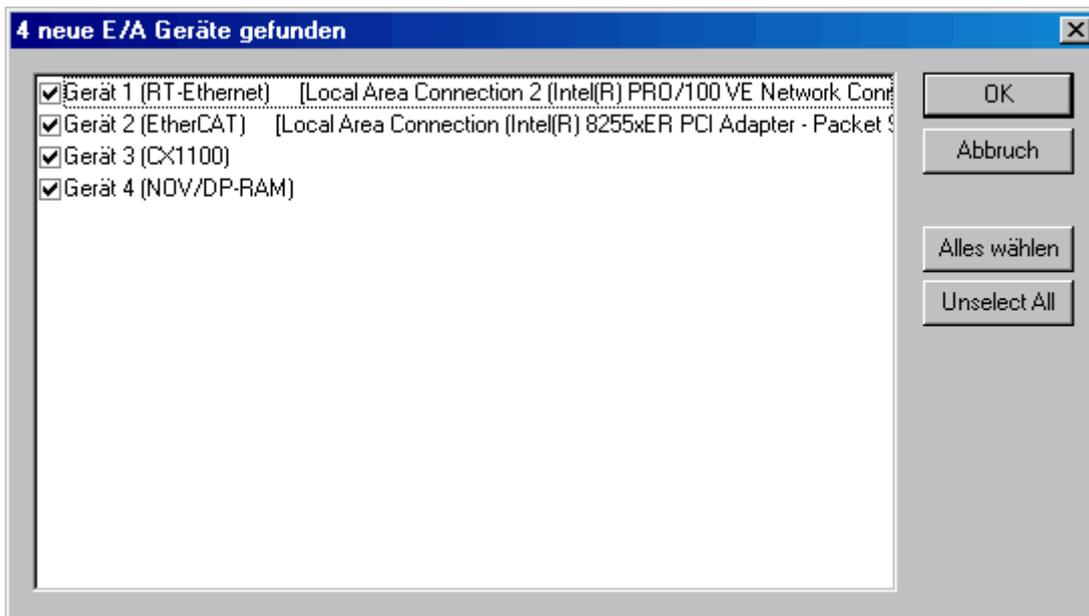
Wenn sich der Mauszeiger über E/A-Geräte befindet, können Sie mit der rechten Maustaste ein Kontextmenü öffnen, aus dem Sie den Eintrag 'Gerät Suchen' auswählen müssen. Sie können dies auch über die Menüleiste mit dem 'Zauberstab'-Symbol erledigen (roter Pfeil).



Es erscheint ein Hinweis, dass nicht alle Geräte automatisch erkannt werden können -> 'OK'.



TwinCAT durchsucht nun die angeschlossene Peripherie und listet die gefundenen Geräte auf -> 'OK'.



Der System-Manager will nun an den Kanälen des CX1020 nach Boxen (Buskopplern, Busklemmen Controllern oder Feldbus Box Modulen) suchen. -> 'OK'.

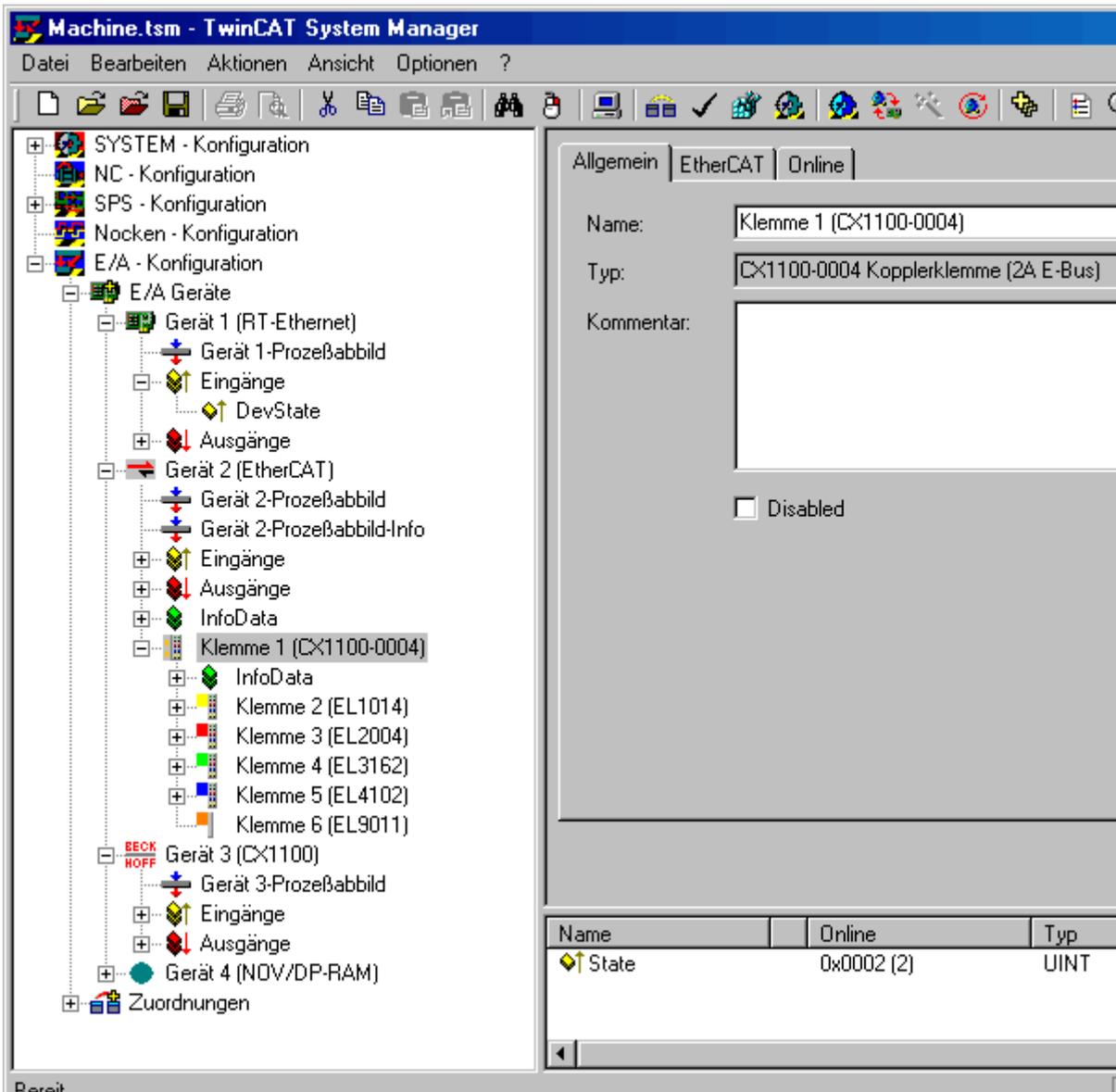


Der System Manager möchte nun den Free Run Modus aktivieren. Klicken Sie auf 'Ja'.



(Das Konfig-Modus Feature erlaubt das direkte Testen der I/O-Hardware, indem Ausgänge geschrieben werden können, ohne erst ein lauffähiges SPS Programm erstellen zu müssen.)

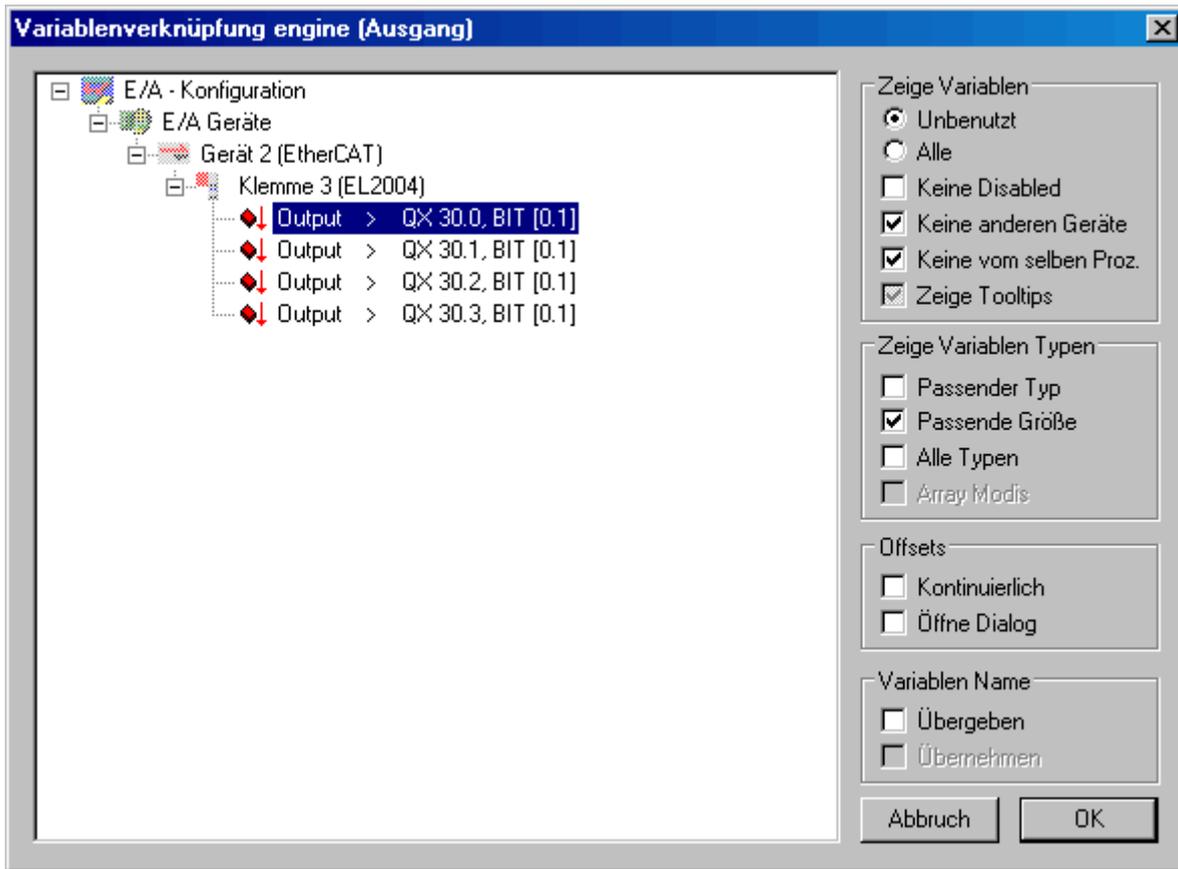
Der System Manager zeigt nun die gefundene I/O-Konfiguration an:



Sie können die Standardbezeichnungen (Gerät 1, Box 1, Klemme 2, usw.) umbenennen. Führen Sie dazu einen langsamen Doppelklick auf den entsprechenden Namen aus, und geben Sie die neue Bezeichnung ein.

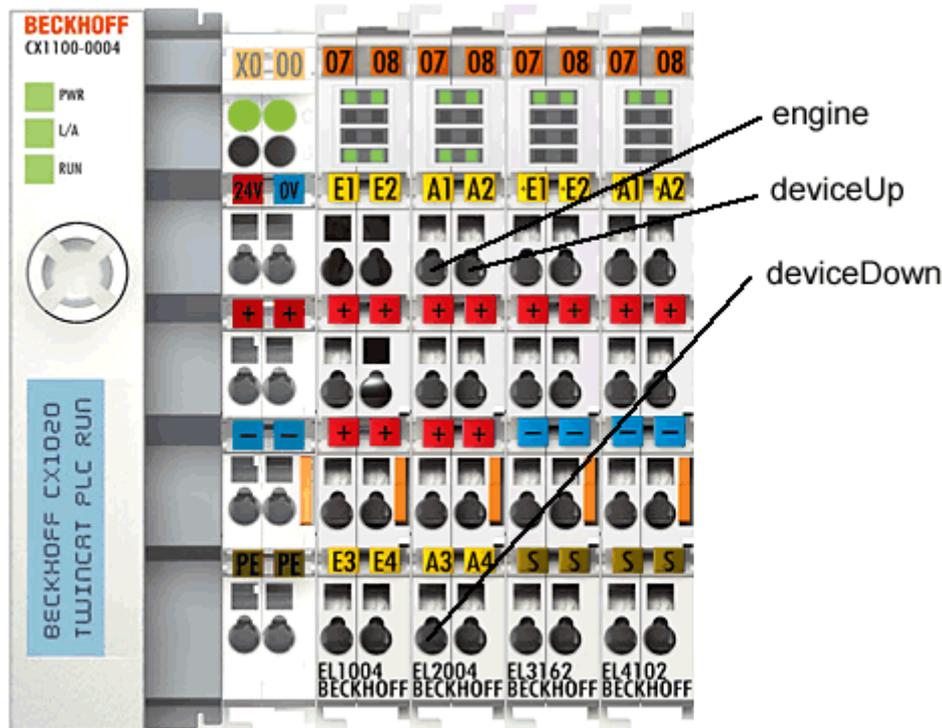
Variablen den Aus-/Eingangskanälen zuweisen

Bis zu dieser Stelle wurde die komplette Hardware, die für das obige Beispielprogramm benötigt wird, konfiguriert. Als nächstes müssen die einzelnen Variablen aus dem SPS-Projekt den einzelnen Ein-/Ausgangskanälen zugewiesen werden. Doppelklicken Sie dazu auf die Variablen des Programms, die Sie zuordnen möchten. Es öffnet sich ein Dialogfenster, in dem die passenden Ein-/Ausgangskanäle der angeschlossenen Peripherie aufgelistet sind. Sie sehen die 4 Ausgangskanäle aufgelistet, die noch nicht belegt sind. Wählen Sie für alle Variablen die gewünschten digitalen Ausgabe-Kanäle aus.



Klemme EL1014	SPS Variable	Bedeutung
Kanal 1 (= Ausgang 1)	engine	Ansteuerung Schrittmotor
Kanal 2 (= Ausgang 2)	deviceUp	Ansteuerung Bohrer hochfahren
Kanal 3 (= Ausgang 3)	deviceDown	Ansteuerung Bohrer runterfahren

Belegung der Busklemmen (an der Beispielkonfiguration)



Projekt speichern

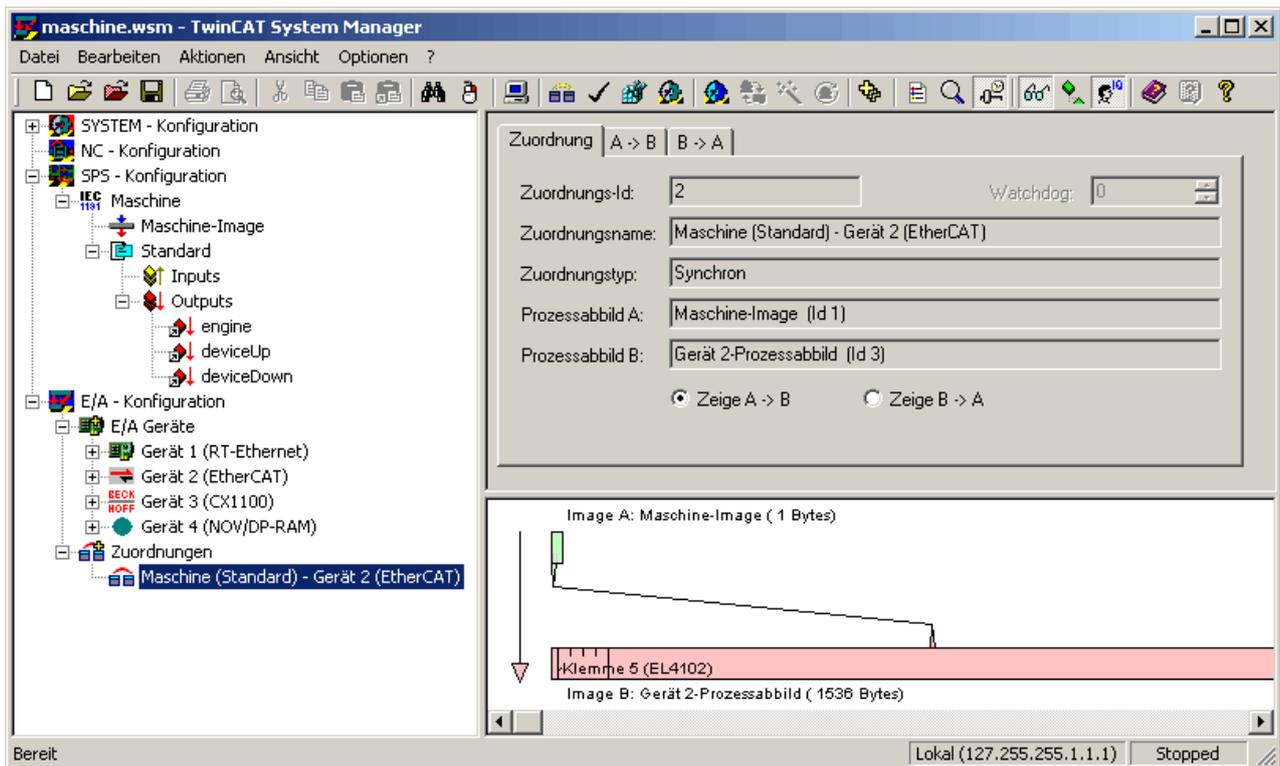


Sie sollten an dieser Stelle die Konfiguration abspeichern, damit Sie später wieder darauf zugreifen können. Führen Sie dazu aus dem Menü 'Datei' den Befehl 'Speichern unter...' aus.

Zuordnung erzeugen



Sie haben jetzt die gesamte System-Konfiguration für das obige Beispielprogramm durchgeführt. Sie müssen jetzt die Zuordnung für die Registry erzeugen. Gehen Sie dazu auf den Befehl 'Zuordnung erzeugen' im Menü 'Aktionen'. Unterhalb des Baumeintrags 'Zuordnungen' sehen Sie jetzt 'Maschine (Standard) - Gerät 2 (EtherCAT)'. Klicken Sie diesen Eintrag an. Daraufhin öffnet sich folgendes Fenster auf der rechten Seite:



Sie können festlegen, ob der Datenfluss von A nach B oder von B nach A angezeigt werden soll. Hierbei entspricht das Image A dem Prozessabbild der SPS-Variablen, also den Ein-/Ausgangsvariablen. Image B enthält das Prozessabbild der E/A-Geräte, in diesem Fall vom EtherCAT-Buskoppler. Jede Variable oder Busklemme ist farblich im Prozessabbild hervorgehoben. Wenn Sie mit der Maus auf eine dieser Flächen stehen bleiben, erscheint ein kleines Anzeigefeld, in dem die genaue Bezeichnung dargestellt wird. Mit der rechten Maustaste können Sie das Bild zoomen.

Konfiguration in Registry schreiben



Als letzter Schritt muss die Konfiguration in die Registry gespeichert werden, da beim Starten von TwinCAT nur die dort abgespeicherten Informationen ausgewertet werden. Führen Sie aus dem Menü 'Aktionen' den Befehl 'Aktiviert Konfiguration...' aus. Falls eine ältere Konfiguration dort schon abgespeichert ist, erscheint eine Sicherheitsabfrage, die Sie bestätigen können.

TwinCAT neu starten

TwinCAT möchte nun im 'Run Modus' neu starten um die Änderungen zu übernehmen.



Durch die neuen Konfiguration ist nun das zuvor auf die Steuerung geladene Programm nicht mehr vorhanden. Die LED an den Busklemmen blinken.

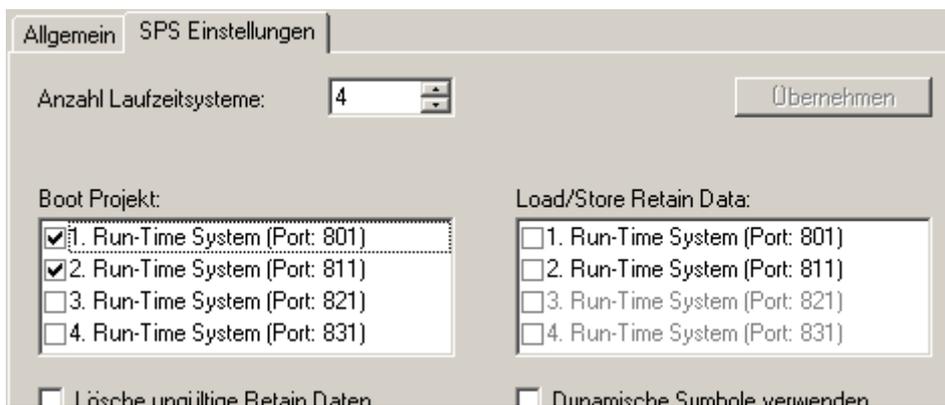
Öffnen Sie nun wieder das Programm 'PLC Control' und führen Sie im Menü 'Online' den Befehl 'Einloggen' aus. Es erscheint die Meldung: Kein Programm auf der Steuerung! Bestätigen Sie mit 'OK', woraufhin das Programm auf die SPS geladen wird.

Mit 'Online' -> 'Start' beginnt die Programmabarbeitung mit der neuen Konfiguration. An der Busklemme EL2004 werden jetzt die einzelnen SPS-Variablen ausgegeben. Dies ist am Leuchten der LEDs an den Busklemmen zu erkennen.

Boot-Einstellungen

Sobald Sie die SPS neu starten, müssen Sie ihr Projekt erneut laden. Sie können jedoch auch ein Boot-Projekt erzeugen. Diese bleibt in der Registry gespeichert und startet, sobald der TwinCAT-RealTime-Server gestartet wird, vorausgesetzt, die Konfiguration hat sich nicht geändert.

Konfigurieren Sie dafür zunächst im System Manager-Menü 'SPS-Konfiguration' unter der Registerkarte 'SPS Einstellungen', wo auch die Anzahl der zur Verfügung stehenden Laufzeitsysteme festgelegt werden kann, welches Laufzeitsystem ein Boot-Projekt aufnehmen soll.



Die entsprechenden Programme müssen Sie zunächst im System-Manager mit 'SPS-Projekt anfügen' laden, anschließend die E/As verknüpfen und die Konfiguration aktivieren. Nun müssen Sie mit TwinCAT PLC-Control für jedes zugehörige SPS-Programm nach dem Einloggen ('Online' -> 'Einloggen') noch den Befehl 'Online' -> 'Erzeugen eines Bootprojektes' ausführen. Ebenso können Sie es mit dem entsprechenden Befehl wieder löschen. Achten Sie hierbei darauf, dass die zugehörigen Laufzeitsysteme übereinstimmen. Loggen Sie sich wieder aus und schließen Sie PLC-Control. Speichern Sie Ihr System-Manager-Projekt.

● Boot-Projekt bleibt erhalten

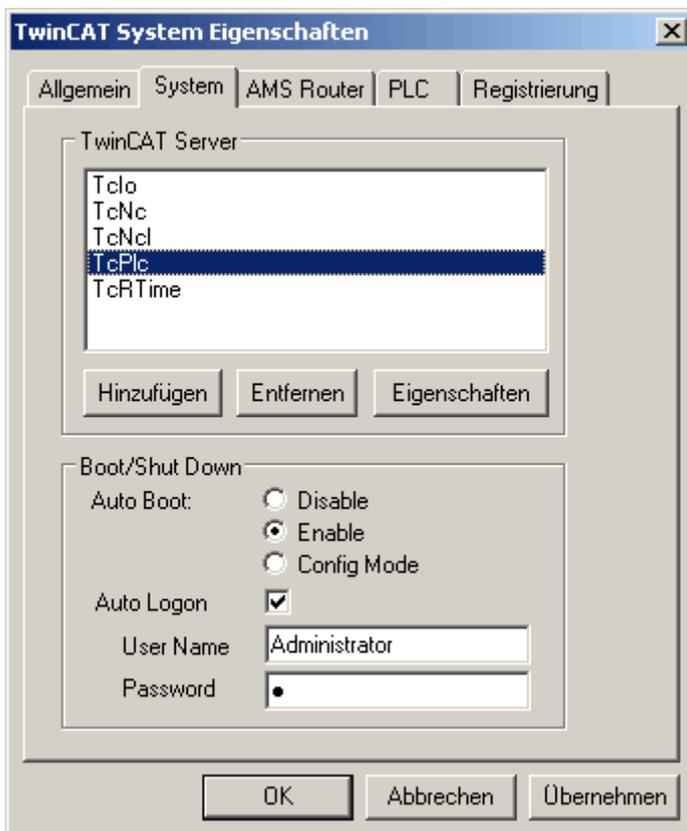
i Wird das Boot-Projekt nicht überschrieben oder gelöscht, bleibt es solange in der SPS geladen, bis Sie ein neues Boot-Projekt für dieses Laufzeitsystem erstellen oder es löschen.

Soll TwinCAT beim Systemstart automatisch ein oder mehrere SPS-Programm(e) abarbeiten, müssen zunächst die Anzahl der Laufzeitsysteme festgelegt werden und die Bootprojekte geladen werden (siehe oben).

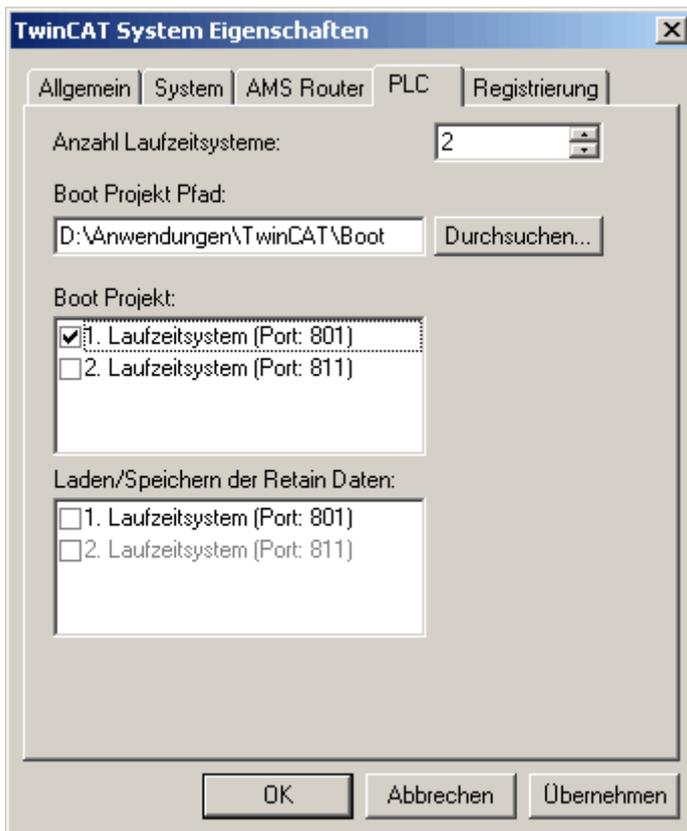
Öffnen Sie nun über den TwinCAT-Button in der Taskleiste das Menü 'Eigenschaften':



In der Registerkarte 'System' müssen Sie die Option 'Enable' unter 'Auto-Boot' aktivieren. Soll der SPS-Programmstart vollständig ohne Ihr Eingreifen von statten gehen, müssen Sie zudem noch Ihre Login-Daten für 'Auto Logon' eingeben. Ist dieser Punkt nicht aktiviert, bedarf es nach dem Systemstart zunächst einer Benutzer-Anmeldung:



Die Einstellungen für 'Auto-Boot' sind identisch mit der Einstellung 'System-Konfiguration' -> 'Boot-Einstellungen' im System Manager.
Ebenso die Einstellungen der Registerkarte 'PLC' mit den Einstellungen unter 'SPS-Konfiguration' -> 'SPS-Einstellungen' im System Manager:



Nach einem Neustart des Systems wird TwinCAT (Auto-Logon vorausgesetzt) im 'Run'-Modus starten und die geladenen und freigegebenen Boot-Projekte selbstständig beginnen abzuarbeiten.

7.4 Beispiele Maschine mit ...

7.4.1 Beispiel Maschine mit Microsoft Visual C#

Microsoft Visual Studio 2005 ist eine Entwicklungsumgebung zur Erstellung um C#-Projekten. Anhand des Maschine Beispiels werden Sie das Einbinden der TwinCAT ADS .NET Komponente mit der Programmiersprache C# (ausgeschrieben: CSharp) kennenlernen.

Erforderliche Software:

- Microsoft .NET Framework Version 2.0, mehr dazu [hier](#)
- Microsoft Visual Studio 2005
- TwinCAT 2.10

Bevor Sie das C#-Programm starten können, muss TwinCAT und das SPS-Programm aktiv sein. Ist auf Ihrem Rechner nicht Microsoft Visual Studio 2005 installiert, so müssen Sie das Microsoft .NET Framework Version 2.0 installieren. Dadurch werden die benötigten DLL's auf Ihrem System eingerichtet.

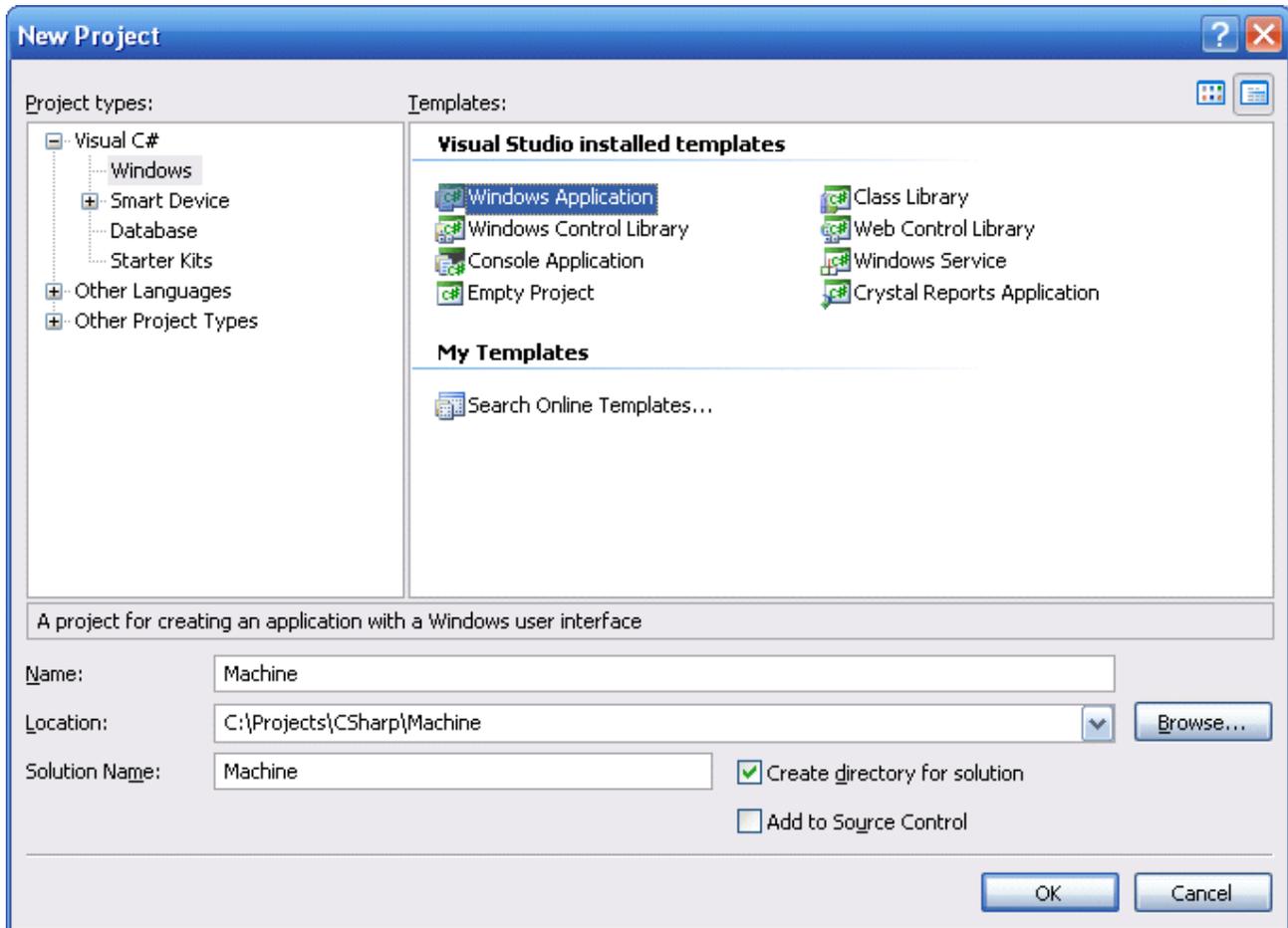
Die ersten Schritte...

Schritt für Schritt lernen Sie die Entwicklung eines C#-Programm und das Einbinden der TwinCAT ADS .NET Komponente anhand eines Beispiels kennen.

1. Neues Projekt erstellen:

Starten Sie das Visual Studio 2005. Erstellen Sie ein neues Projekt, indem Sie im Menü -> File -> New -> Project klicken. Im Dialogfeld 'New Project' können Sie ein neues Projekt auf der Basis verschiedener Vorlagen erstellen. Hier wählen Sie unter 'Project Types' die Programmiersprache Visual C# und unter den

Vorlagen 'Windows Application'. Geben Sie Ihrem Projekt ein neuen Namen, in diesem Fall bitte den Namen 'Machine' eingeben. Wählen sie dann den Verzeichnispfad unter 'Location' in dem Sie ihr Arbeitsprojekt erstellen wollen.



2. Bedienungsfläche erstellen

Hierzu wird zuerst eine Oberfläche in dem Design Modus der Form 'frmMachine' mit Hilfe der *Toolbox* erstellt. Die Einstellungen der verschiedenen Controls (wie z.B. 10 Labels, 2 Radiobuttons, 1 Progressbar, 1 PictureBox, GroupBox...) können Sie im Properties Window des Visual Studio einsehen und einstellen.

```
// Gruppenfelder (GroupBox)
grpDevice.Text = "";
grpCount.Text = "";
grpSpeed.Text = "Speed";

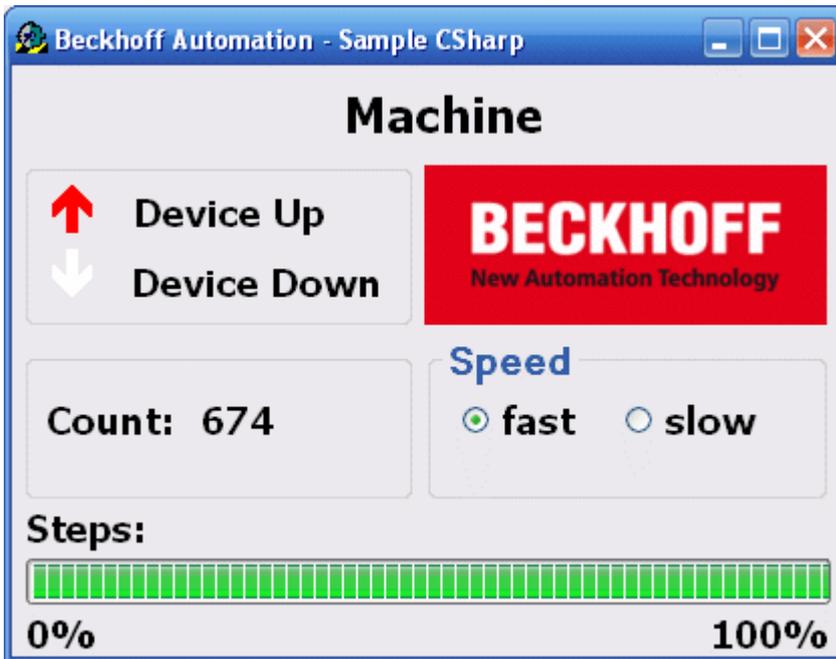
// Bezeichnungsfelder (Label)
lblMachine.Text = "Machine";
lblDeviceDown.Text = "Device Down";
lblDeviceUP.Text = "Device Up";
lblCount.Text = "0";
lblCountLabel.Text = "Count:";
lblSteps.Text = "Steps:";
lbl100Procent.Text = "100%";
lbl0Procent.Text = "0%";

// Das sind die Pfeile DeviceDown und DeviceUp mit einer anderen Schriftart und Schiftgröße (Label)
DeviceDown.Text = "ê";
DeviceDown.Font = new System.Drawing.Font("Wingdings", 20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)2));
DeviceUp.Text = "é";
DeviceUp.Font = new System.Drawing.Font("Wingdings", 20.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)2));

// ProgressBar
```

```
prgSteps.Name = "prgSteps";

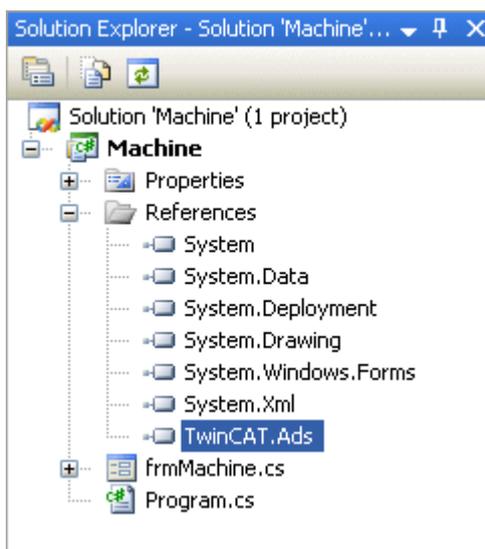
// Optionsfelder (RadioButton)
optSpeedSlow.Text = "slow";
optSpeedFast.Text = "fast";
```



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf die Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Speed' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Steps' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

3. Referenz hinzufügen

Hier muss zuerst eine Referenz namens TwinCAT.Ads.dll hinzugefügt werden. Die TwinCAT ADS .NET Komponente wird über das Menü -> Project -> Add Reference eingebunden. Um zu überprüfen ob tatsächlich die Referenz eingebunden wurde, können Sie sich im Solution Explorer (*STRG + W + S*) vergewissern.



4. Quelltext bearbeiten

Nach dem Erstellen der Oberfläche und das Einbinden der TwinCAT ADS Komponente kann zum C# Quelltext gewechselt werden. In die oberste Zeilen des Quelltextes werden die benötigten Namespaces 'System.IO' und 'TwinCAT.Ads' eingefügt.

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

Als nächstes sind die Deklarationen innerhalb der Klasse frmMachine dran.

```
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;
```

Die Methode 'frmMachine_Load' wird beim Aufruf der Windows Application gestartet. Diese Methode verbindet man, indem das Event 'Load' der Form mit der Methode im Properties Fenster verknüpft wird. In ihr werden Instanzen verschiedener Klassen erzeugt und eine Verbindung mit Laufzeitsystem 1 der Komponente TwinCAT.Ads über Port 801 hergestellt.

```
//-----//
Wird als erstes beim Starten des Programms aufgerufen//-----
-----private void frmMachine_Load(object sender, EventArgs e)
{
    try
    {
        // Eine neue Instanz der Klasse AdsStream erzeugen
        dataStream = newAdsStream(7);

        // Eine neue Instanz der Klasse BinaryReader erzeugen
        binReader = newBinaryReader(dataStream);

        // Eine neue Instanz der Klasse TcAdsClient erzeugen
        tcClient = newTcAdsClient();

        // Verbinden mit lokaler SPS - Laufzeit 1 - Port 801
        tcClient.Connect(801);
    }
    catch
    {
        MessageBox.Show("Fehler beim Laden");
    }
}
//...
```

SPS Variablen verbinden:

In dem frmMachine_Load-Ereignis der Form wird mit der Methode TcAdsClient.AddDeviceNotification()eine Verbindung zu jeder Variablen in der SPS erzeugt. Das Handle dieser Verbindung wird in einem Array gespeichert. Der Parameter TransMode spezifiziert die Art des Datenaustausches. In diesem Fall wird mit AdsTransMode.OnChange festgelegt, dass die SPS-Variable nur übermittelt wird falls sich der Wert in der SPS geändert hat(siehe AdsTransMode). Der Parameter *cycleTime* bestimmt, wie oft die PLC checken soll, ob sich die entsprechende Variable geändert hat. Dann werden die Variablen mit einer Methode '*tcClient_OnNotification*' (die noch geschrieben werden muss) verbunden, die bei einer Änderung einer Variablen aufgerufen wird.

```

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

    // Holen des Handles von "switch" - wird für das Schreiben des Wertes benötigt
    hSwitchWrite = tcClient.CreateVariableHandle(".switch");

    // Erstellen eines Events für Änderungen an den SPS-Variablen-Werten
    tcClient.AdsNotification += new AdsNotificationEventHandler(tcClient_OnNotification);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

Definition:

Zum Verbinden der Variablen wurde die Methode 'AddDeviceNotification()' verwendet.

```

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

```

- **variableName:** Name der SPS Variable.
- **dataStream:** Der Datenstrom, der die Daten empfängt.
- **offset:** Abstand der Daten im Stream.
- **length:** Länge der Daten im Stream.
- **transMode:** Das Ereignis, wenn sich die Variable ändert.
- **cycletime:** Die Zeit (in ms) nach der der SPS-Server überprüft, ob sich die Variable geändert hat.
- **maxDelay:** Die Zeit (in ms) nach der das Ereignis spätestens beendet ist.
- **userData:** Das Objekt, das zu ablegen bestimmter Daten verwendet werden kann.

Zum Verbinden der Variable 'hSwitchWrite' wurde die Methode 'CreateVariableHandle' verwendet.

```

int TcAdsClient.CreateVariableHandle(string variableName);

```

- **variableName:** Name der SPS-Variable.

Methode schreiben:

Oben wurde bereits auf eine Methode verwiesen, die noch gar nicht existiert. Daher wird diese Methode, die 'tcClient_OnNotification' genannt wurde, als nächstes geschrieben. Diese Methode wird aufgerufen, wenn sich eine der SPS-Variable geändert hat.

```

//-----//wird bei Änderung einer SPS-
//Variablen aufgerufen//-----
private void tcClient_OnNotification(object sender, AdsNotificationEventArgs e)
{
    try
    {
        // Setzen der Position von e.DataStream auf die des aktuellen benötigten Wertes
        e.DataStream.Position = e.Offset;

        // Ermittlung welche Variable sich geändert hat if(e.NotificationHandle == hDeviceUp)
        {
            //
            // Die Farben der Grafiken entsprechend der Variablen anpassen if (binReader.ReadBoolean() == true)
            {
                DeviceUp_LED.ForeColor = Color.Red;
            }
        }
    }
}

```

```

    }
    else
    {
        DeviceUp_LED.ForeColor = Color.White;
    }
}
else if(e.NotificationHandle == hDeviceDown)
{
    if (binReader.ReadBoolean() == true)
    {
        DeviceDown_LED.ForeColor = Color.Red;
    }
    else
    {
        DeviceDown_LED.ForeColor = Color.White;
    }
}
else if(e.NotificationHandle == hSteps)
{
    // Einstellen der ProgressBar auf den aktuellen Schritt
    prgSteps.Value = binReader.ReadByte();
}
else if(e.NotificationHandle == hCount)
{
    // Anzeigen des "count"-Werts
    lblCount.Text = binReader.ReadUInt16().ToString();
}
else if(e.NotificationHandle == hSwitchNotify)
{
    // Markieren des korrekten RadioButtonsif (binReader.ReadBoolean() == true)
    {
        optSpeedFast.Checked = true;
    }
    else
    {
        optSpeedSlow.Checked = true;
    }
}
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
}

```

Es fehlen noch zwei Methoden, mit denen die Maschine schnell oder langsam gestellt wird. In Ihnen wird ein virtueller Schalter umgelegt, hier wird ein Wert in die SPS-Variable switch geschrieben.

```

//-----//
wird aufgerufen, wenn das Feld 'slow' markiert wird//-----
-----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 'fast' markiert wird//-----
-----private void optSpeedSlow_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

Zu guter Letzt muss noch dafür gesorgt werden, dass die Methoden 'optSpeedFast_Click' und 'optSpeedSlow_Click' auch beim richtigen Ereignis aufgerufen werden. Dazu in die Design Ansicht wechseln, den Radiobutton 'optSpeedFast' markieren, im Properties Window bei Events auf 'Click' klicken und dann die Methode 'optSpeedFast_Click' wählen. Und das Gleiche muss auch noch mit dem RadioButton 'optSpeedSlow' mit der Methode 'optSpeedSlow_Click' gemacht werden.

Notifications und Handles löschen:

In dem frmMachine_FormClosing-Ereignis der Form werden die Verbindungen wieder mit der Methode DeleteDeviceNotification() freigegeben.

```
//-----//  
wird beim Beenden des Programms aufgerufen//-----  
private void frmMachine_FormClosing(object sender, FormClosingEventArgs e)  
{  
    try  
    {  
        // Löschen der Notifications und 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();  
}
```

Das SPS Programm Machine_Final.pro sollte auf dem Laufzeitsystem 1 starten und das erstellte C#-Programm Machine.exe kann getestet werden.

Download C# Programm Beispiel:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281689995.zip>

7.4.2 Beispiel Maschine mit Microsoft Visual Basic .NET

Microsoft Visual Studio 2005 ist eine Entwicklungsumgebung zur Erstellung von Visual Basic-Projekten. Anhand Anhand des Maschine Beispiels werden Sie das Einbinden der TwinCAT ADS .NET Komponente mit der Programmiersprache Visual Basic kennenlernen.

Erforderliche Software:

- Microsoft .NET Framework Version 2.0, mehr dazu unter <http://msdn2.microsoft.com>
- Microsoft Visual Studio 2005
- TwinCAT 2.10

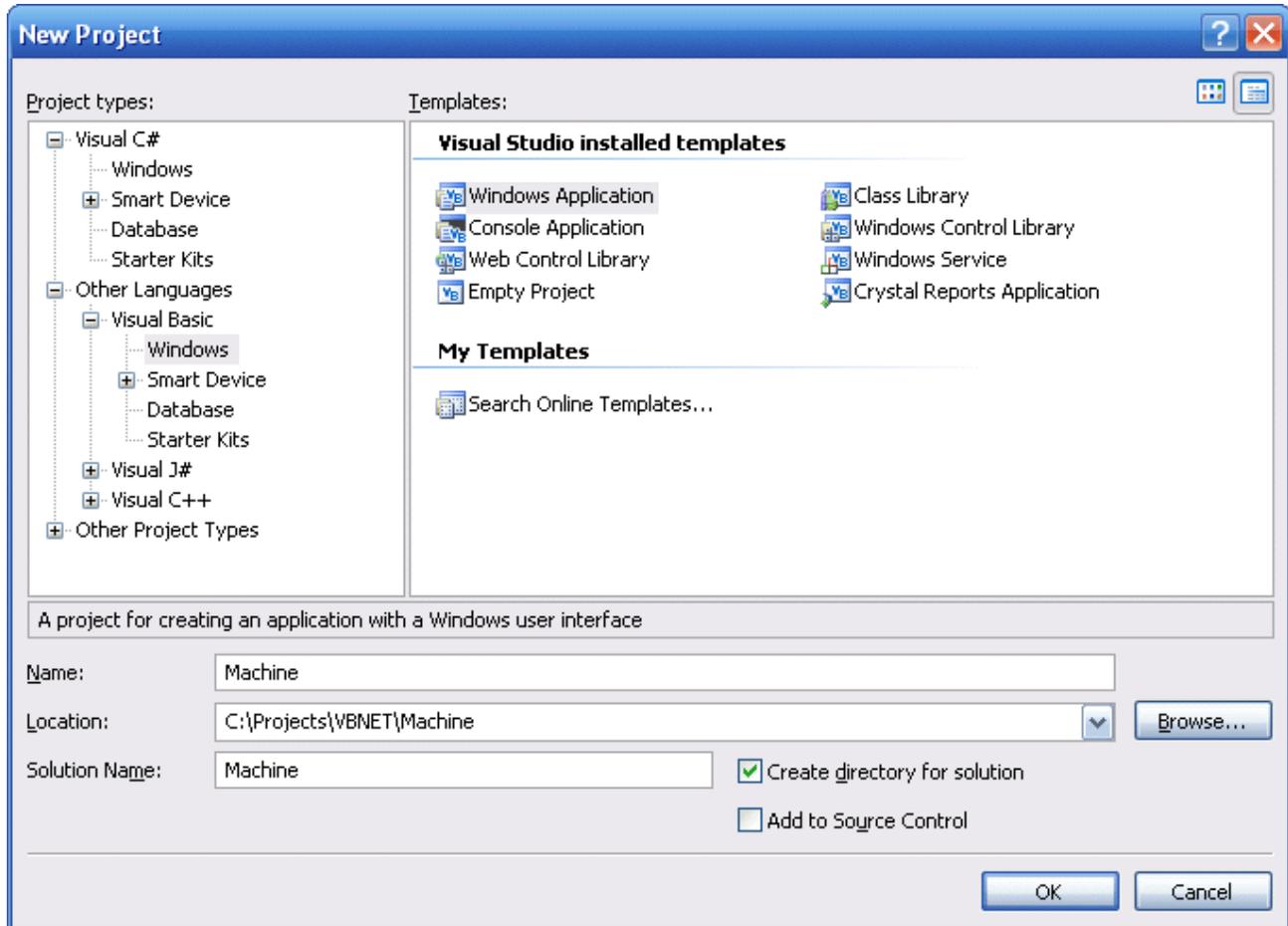
Um das Visual Basic-Programm starten zu können, muss TwinCAT und das SPS-Programm aktiv sein. Ist auf Ihrem Rechner kein Microsoft Visual Studio 2005 installiert, müssen Sie das Microsoft .NET Framework Version 2.0 installieren. Dadurch werden die benötigten DLL's auf Ihrem System eingerichtet.

Die ersten Schritte...

Die Entwicklung eines Visual Basic-Programms und das Einbinden der TwinCAT ADS .NET Komponente wird in den folgenden Schritten beschrieben.

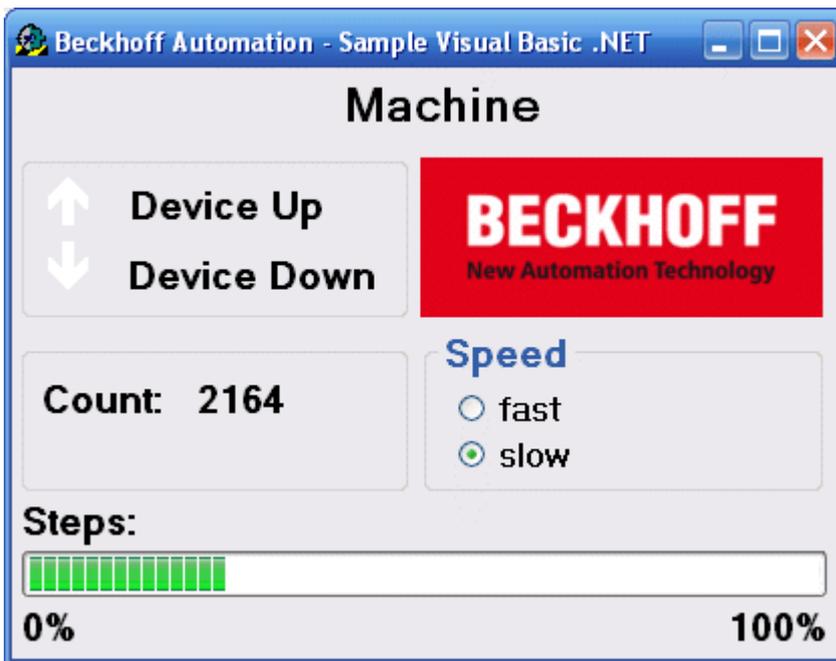
1. Neues Projekt erstellen:

Starten des Visual Studio 2005 und Erstellen eines neues Projekts über Menü -> File -> New -> Project. Im Dialogfeld 'New Project' kann eine neues Projekt auf Basis verschiedener Vorlagen erstellt werden. Wählen Sie unter 'Project Types' die Programmiersprache 'Visual Basic' und unter den Vorlagen 'Windows Application'. Geben Sie Ihrem Projekt ein neuen Namen, in diesem Fall bitte den Namen 'Machine' eingeben. Wählen Sie dann unter 'Location' den Verzeichnispfad in dem Sie ihr Arbeitsprojekt erstellen wollen.



2. Bedienungs Oberfläche erstellen

Hierzu wird zuerst eine Oberfläche in dem Design Modus der Form 'frmMachine' mit Hilfe der *Toolbox* erstellt. Die Einstellungen der verschiedenen Controls (wie z.B. 10 Labels, 2 Radiobuttons, 1 Progressbar, 1 PictureBox, GroupBox...) können Sie im Properties Window des Visual Studio einsehen und einstellen.



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf die Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Speed' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Steps' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

3. Referenz hinzufügen

Hier muss zuerst eine Referenz namens TwinCAT.Ads.dll hinzugefügt werden. Die TwinCAT ADS .NET Komponente wird über das Menü -> Project -> Add Reference eingebunden.

4. Quelltext bearbeiten

Nach dem Erstellen der Oberfläche und das Einbinden der TwinCAT ADS Komponente kann zum Visual Basic Quelltext gewechselt werden. In die oberste Zeilen des Quelltextes werden die benötigten Namespaces 'System.IO' und 'TwinCAT.Ads' eingefügt.

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

Danach folgen die Deklarationen innerhalb der Klasse frmMachine.

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

Die Methode 'frmMachine_Load' wird beim Aufruf der 'Windows Application' gestartet. Diese Methode wird durch Verknüpfung des Event 'Load' der Form mit der Methode im Properties Fenster verbunden. In ihr werden Instanzen verschiedener Klassen erzeugt und eine Verbindung mit Laufzeitsystem 1 der Komponente TwinCAT.Ads über den Port 801 hergestellt.

```
'-----
'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
        ' Eine neue Instanz der Klasse AdsStream erzeugen
        dataStream = New AdsStream(7)

        ' Eine neue Instanz der Klasse BinaryReader erzeugen
        binReader = New BinaryReader(dataStream)

        ' Eine neue Instanz der Klasse TcAdsClient erzeugen
        tcClient = New TwinCAT.Ads.TcAdsClient()
```

```

' Verbinden mit lokaler SPS - Laufzeit 1 - Port 801
tcClient.Connect(801)

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

' ...

```

SPS Variablen verbinden:

In dem `frmMachine_Load`-Ereignis der Form wird mit der Methode `TcAdsClient.AddDeviceNotification()` eine Verbindung zu jeder Variablen in der SPS erzeugt. Das Handle dieser Verbindung wird in einem Array gespeichert. Der Parameter `TransMode` spezifiziert die Art des Datenaustausches. In diesem Fall wird mit `AdsTransMode.OnChange` festgelegt, dass die SPS-Variablen nur übermittelt wird falls sich der Wert in der SPS geändert hat (siehe `AdsTransMode`). Der Parameter `cycleTime` bestimmt, wie oft die PLC prüfen soll, ob sich die entsprechende Variable geändert hat. Dann werden die Variablen mit der Methode `'tcClient_OnNotification'` (siehe "Methode 'tcClient_OnNotification' schreiben") verbunden, die bei einer Änderung einer Variablen aufgerufen wird.

```

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

```

Zum Verbinden der Variablen wurde die Methode `'AddDeviceNotification()'` verwendet.

```

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 der SPS Variablen.
- **dataStream:** Der Datenstrom, der die Daten empfängt.
- **offset:** Abstand der Daten im Stream.
- **length:** Länge der Daten im Stream.
- **transMode:** Das Ereignis, wenn sich die Variable ändert.
- **cycleTime:** Die Zeit (in ms) nach der der SPS-Server überprüft, ob sich die Variable geändert hat.
- **maxDelay:** Die Zeit (in ms) nach der das Ereignis spätestens beendet ist.
- **userData:** Das Objekt, das zum Ablegen bestimmter Daten verwendet werden kann.

Zum Verbinden der Variable `'hSwitchWrite'` wurde die Methode `'CreateVariableHandle'` verwendet.

```

Public Function CreateVariableHandle(ByVal variableName As String) As Integer

```

- **variableName:** Name der SPS Variablen.

Methode 'tcClient_OnNotification' schreiben:

Diese Methode wird aufgerufen, wenn sich eine der SPS-Variablen geändert hat.

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

        ' Ermittlung welche Variable sich geändert hat
        ' Detecting which variable has changed
        If e.NotificationHandle = hDeviceUp Then
            'Die Farben der Grafiken entsprechen der Variablen anpassen
            'Adapt colors of graphics according to the variables
            If binReader.ReadBoolean() = True Then
                DeviceUp_LED.ForeColor = Color.Red
            Else
                DeviceUp_LED.ForeColor = Color.White
            End If
        ElseIf e.NotificationHandle = hDeviceDown Then
            If binReader.ReadBoolean() = True Then
                DeviceDown_LED.ForeColor = Color.Red
            Else
                DeviceDown_LED.ForeColor = Color.White
            End If
        ElseIf e.NotificationHandle = hSteps Then
            ' Einstellen der ProgressBar auf den aktuellen Schritt
            ' Setting the ProgressBar to the current step
            prgSteps.Value = binReader.ReadByte()
        ElseIf e.NotificationHandle = hCount Then
            ' Anzeigen des "count"-Werts
            ' Displaying the "count" value
            lblCount.Text = binReader.ReadUInt16().ToString()
        ElseIf e.NotificationHandle = hSwitchNotify Then
            ' Markieren des korrekten RadioButtons
            ' Checking the correct RadioButton
            If binReader.ReadBoolean() = True Then
                optSpeedFast.Checked = True
            Else
                optSpeedSlow.Checked = True
            End If
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

Es fehlen noch zwei Methoden, mit denen die Maschine schneller oder langsamer fährt. In Ihnen wird ein virtueller Schalter umgelegt, hier wird ein Wert in die SPS-Variable switch geschrieben.

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

'-----
'wird aufgerufen, wenn das Feld 'fast' markiert wird
'is activated when the 'fast' field is marked
'-----
Private Sub optSpeedFast_Click(ByVal sender As Object, ByVal e As EventArgs) Handles optSpeedFast
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 optSpeedSlow
w.Click
    Try
        tcClient.WriteAny(hSwitchWrite, False)
    Catch ex As Exception
```

```

    MessageBox.Show(ex.Message)
End Try
End Sub

```

Damit die Methoden '*optSpeedFast_Click*' und '*optSpeedSlow_Click*' beim richtigen Ereignis aufgerufen werden, muss der Radiobutton '*optSpeedFast*' in der Design Ansicht markiert werden, im Properties Window bei Events auf '*Click*' geklickt und dann die Methode '*optSpeedFast_Click*' gewählt werden. Gleiches gilt für den Radiobutton '*optSpeedSlow*' mit der Methode '*optSpeedSlow_Click*'.

Notifications und Handles löschen:

In dem *frmMachine_FormClosing*-Ereignis der Form werden die Verbindungen wieder mit der Methode *DeleteDeviceNotification()* freigegeben.

```

'-----
'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

```

Das SPS Programm *Machine_Final.pro* auf dem Laufzeitsystem 1 starten und danach kann das erstellte VisualBasic-Programm *Machine.exe* gestartet werden.

Download Visual Basic Beispiel:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281691403.zip>

7.4.3 Beispiel Maschine mit Microsoft Visual Basic 6.0

TwinCAT ADS OCX

TwinCAT besitzt mehrere Programmierschnittstellen, um applikationsspezifische Programme in das System zu integrieren. Eine Programmiersprache die unterstützt wird, ist Visual Basic von Microsoft. Die Stärke dieser Programmiersprache ist das Erstellen von grafischen Bedieneroberflächen und die Möglichkeit diese an Datenbanken anzubinden. Visual Basic ist seit Jahren stark verbreitet und wird von Microsoft ständig weiterentwickelt. Viele Drittanbieter bieten Zusatzmodule für verschiedene Bereiche an. Der Oberbegriff für solche Module ist OCX. Beckhoff liefert für TwinCAT ein OCX, welches die Bezeichnung ADS OCX hat. Das ADS OCX stellt Methoden zur Verfügung, um mit anderen ADS-Geräten (PLC, NC/CNC, ...) über den TwinCAT Message Router zu kommunizieren. Das mitgelieferte Beispielprogramm zeigt, wie Sie von Visual Basic auf einzelne Variablen des TwinCAT PLC Servers zugreifen können.

Zugriffsmöglichkeiten:

Das ADS OCX beinhaltet verschiedene Verfahren um die Werte aus anderen ADS-Geräten auszulesen. Die Entscheidung welches Verfahren Sie einsetzen, hängt von der Umgebung ab, in der das Programm ausgeführt werden soll. Weitere Hinweise und eine ausführliche Referenz der einzelnen Funktionen finden Sie in der speziellen Anleitung zum ADS OCX. Hier soll nur eine kurze Übersicht über die einzelnen Zugriffsverfahren gezeigt werden:

Zugriff	Bedeutung
connect	Sobald eine Kommunikation zwischen einer SPS-Variablen und einer Visual Basic-Variablen benötigt wird, wird durch ein Methodenaufzuruf eine Verbindung zwischen diesen beide Variablen erstellt. Im weiteren Verlauf des Programms wird von TwinCAT die Visual Basic-Variable mit der SPS-Variablen abgeglichen. Diese Art des Datenaustauschs kann auch so benutzt werden, dass bei Änderung einer SPS-Variablen im Visual Basic-Programm eine Ereignisfunktion aufgerufen wird (ereignisgesteuerte Datenübertragung).
synchron	Nach Aufruf einer Schreib-/Lesemethode wird die Ausführung des Visual Basic-Programms so lange unterbrochen, bis die angeforderten Daten vorliegen. In den darauf folgenden Anweisungen kann mit den Daten sofort weiter gearbeitet werden.
asynchron	Bei dem asynchronen Zugriff wird die Ausführung des Visual Basic-Programms nicht unterbrochen, sondern sofort mit der Abarbeitung des nächsten Befehls fortgefahren. Treffen die angeforderten Daten beim AdsOCX ein, wird im Visual Basic-Programm eine Ereignisfunktion ausgelöst, in der als Parameter der Wert übergeben wird.

Beispielprogramm

TwinCAT und SPS Programm starten:

Bevor Sie das Visual Basic-Programm starten können, muss TwinCAT und das SPS-Programm aktiv sein. Ist auf Ihrem Rechner nicht Visual Basic 5 oder höher installiert, so müssen Sie auf der TwinCAT CD die ‚SamplesRuntime‘ installieren. Dadurch werden die benötigten DLL's auf Ihrem System eingerichtet. Das VB-Programm können Sie auch mit aktuelleren Versionen von Visual Basic bearbeiten; dazu benötigen Sie die ‚SamplesRuntime‘ nicht mehr.

Visual Basic-Programm starten:

Starten Sie das Programm 'Machine.exe'. (TwinCAT\ Samples\First Steps\VB\)



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf die Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Geschwindigkeit' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Schritte' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

Programmtext:

```
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
'-----'Wird als erstes beim Starten des Programms au
fgerufen'-----
Private Sub Form_Load()

    'Sprachabhängige Wörter aus Resource-Datei laden
    lblMachine.Caption = LoadResString(0 + GetLanguageId)
    lblDeviceUp.Caption = LoadResString(1 + GetLanguageId)
    lblDeviceDown.Caption = LoadResString(2 + GetLanguageId)
    lblCountLabel.Caption = LoadResString(3 + GetLanguageId)
    lblSteps.Caption = LoadResString(4 + GetLanguageId)
    fraSpeed.Caption = LoadResString(5 + GetLanguageId)
    optSpeedFast.Caption = LoadResString(6 + GetLanguageId)
    optSpeedSlow.Caption = LoadResString(7 + GetLanguageId)

    'SPS-Variablen mit Visual Basic-Variablen verbinden
    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)

    'Handle der Variablen ermitteln
    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

'-----'wird beim Beenden des Programms aufgerufen'-----
Private Sub Form_Unload(Cancel As Integer)
    'Visual Basic-Variablen von SPS-Variablen trennen
    Call AdsOcx1.AdsReadIntegerDisconnect(steps)
    Call AdsOcx1.AdsReadBoolDisconnect(deviceUp)
    Call AdsOcx1.AdsReadBoolDisconnect(deviceDown)
    Call AdsOcx1.AdsReadLongDisconnect(counter)

    'Handle der Variablen freigeben
    Call AdsOcx1.AdsDeleteVarHandle(hSteps)
    Call AdsOcx1.AdsDeleteVarHandle(hDeviceUp)
    Call AdsOcx1.AdsDeleteVarHandle(hDeviceDown)
    Call AdsOcx1.AdsDeleteVarHandle(hCounter)
    Call AdsOcx1.AdsDeleteVarHandle(hSwitch)
End Sub

'-----'wird aufgerufen, wenn das Feld 'schnell' mar
kiert wird'-----
Private Sub optSpeedFast_Click()
    Dim switch As Boolean
    'SPS-Variable switch auf TRUE setzen
    switch = True
    Call AdsOcx1.AdsSyncWriteBoolVarReq(hSwitch, 2&, switch)
End Sub

'-----'wird aufgerufen, wenn das Feld 'langsam' mar
kiert wird'-----
Private Sub optSpeedSlow_Click()
    Dim switch As Boolean
    'SPS-Variable switch auf FALSE setzen
    switch = False
    Call AdsOcx1.AdsSyncWriteBoolVarReq(hSwitch, 2&, switch)
End Sub

'-----'wird bei Änderung einer SPS-
Variablen aufgerufen'-----
Private Sub AdsOcx1_AdsReadConnectUpdate(ByVal nIndexGroup As Long, ByVal nIndexOffset As Long)
    Select Case nIndexOffset
        Case hCounter:
            'Stückzahl in Form anzeigen
            lblCount.Caption = counter
        Case hDeviceUp
            'Die Farben der Grafiken entsprechend der Variablen anpassen
            DeviceUp_LED.ForeColor = IIf(deviceUp = True, vbRed, vbWhite)
        Case hDeviceDown
            'Die Farben der Grafiken entsprechend der Variablen anpassen

```

```

DeviceDown_LED.ForeColor = IIf(deviceDown = True, vbRed, vbWhite)
Case hSteps
  'Position vom Werkstück anzeigen
  prgSteps.Width = steps * 240
End Select
End Sub

```

Funktionsweise:

Im Ereignis Form1_Load() werden als erstes die sprachabhängigen Wörter aus der Resourcedatei geladen. Je nach eingestellter Landessprache von Windows werden die deutschen oder die englischen Wörter angezeigt. Anschließend werden durch die Methode 'AdsReadVarConnect' alle notwendigen Visual Basic-Variablen mit den entsprechenden SPS-Variablen verbunden. Die Parameter der Methode sind:

Parameter	Bedeutung
adsVarName	Name der SPS-Variable
cbLenght	Länge der Daten in Byte
nRefreshType	Art des Datenaustauschs
nCycleTime	Refresh Zyklus in ms
pData	Visual Basic-Variable, in der die Daten geschrieben werden

Nachdem die Variablen 'verbunden' wurden, wird von jeder Variablen der Handle geholt. Dieser wird weiter unten in der Ereignisfunktion 'AdsOcx1_AdsReadConnectUpdate' benötigt. Die Ereignisfunktion 'AdsOcx1_AdsReadConnectUpdate()' wird dann aufgerufen, wenn sich die Variable 'Count', 'DeviceUp', 'DeviceDown' oder 'Steps' geändert hat. In ihr werden die Objekte auf der Form mit den Variablen der SPS animiert. Anhand des übergebenden Handles, kann festgestellt werden, welche SPS-Variable sich geändert hat.

Die Funktionen 'optFast_Click()' und 'optSlow_Click()' werden dann aufgerufen, wenn der Bediener das Markierungsfeld 'fast' oder 'slow' anklickt. In diesen Funktionen wird die Variable 'switch' auf TRUE oder FALSE gesetzt. Über die Methode 'AdsSyncWriteBoolVarReq' wird der Wert in die SPS-Variable 'switch' geschrieben. Die Parameter der Methode sind:

Parameter	Bedeutung
hVar	Handle der SPS-Variable
cbLenght	Länge der Daten in Byte
pData	Visual Basic-Variable, welche die Daten enthält

Wenn eine Form geschlossen wird, sollten alle Variablen, die nicht mehr benötigt werden, von den SPS-Variablen getrennt werden. Dieses wird in der Funktion 'Form1_Unload()' durchgeführt. Außerdem werden alle Handles der Variablen wieder freigegeben.

Download Beispiel:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281692811.zip>

7.4.4 Beispiel Maschine mit Microsoft Visual C++ .NET

Microsoft Visual Studio 2005 ist eine Entwicklungsumgebung um C++-Projekte zu erstellen. Anhand des Maschine Beispiels werden Sie das Einbinden der TwinCAT ADS DLL .LIB Komponente mit der Programmiersprache C++ (ausgeschrieben: CPlusPlus) kennen lernen.

Erforderte Software:

- Microsoft .NET Framework Version 2.0, mehr dazu [hier](#)
- Microsoft Visual Studio 2005
- TwinCAT 2.10

Bevor Sie das C++-Programm starten können, muss TwinCAT und das SPS-Programm aktiv sein. Ist auf

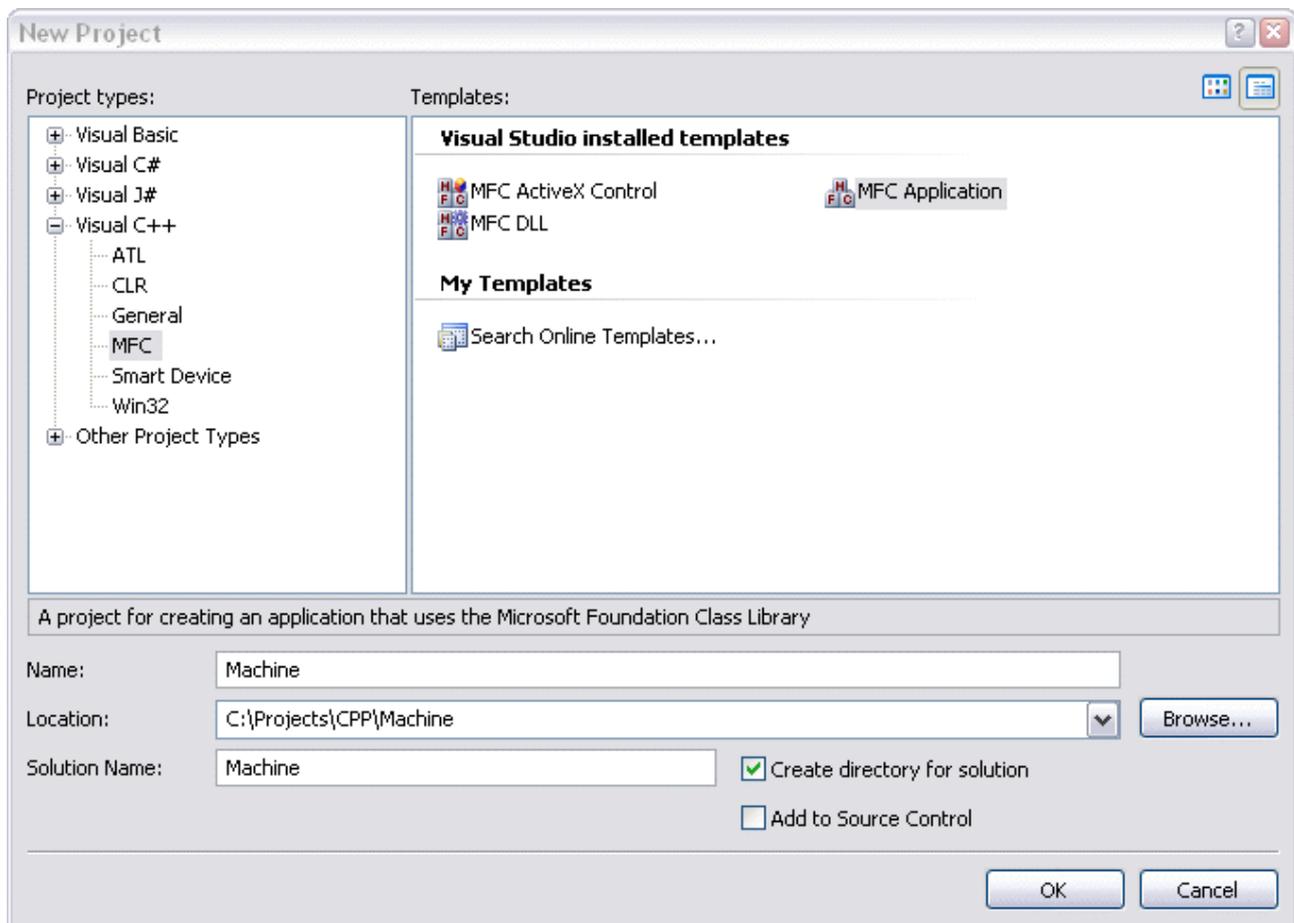
Ihrem Rechner nicht Microsoft Visual Studio 2005 installiert, so müssen Sie das Microsoft .NET Framework Version 2.0 installieren. Dadurch werden die benötigten DLL's auf Ihrem System eingerichtet.

Die ersten Schritte...

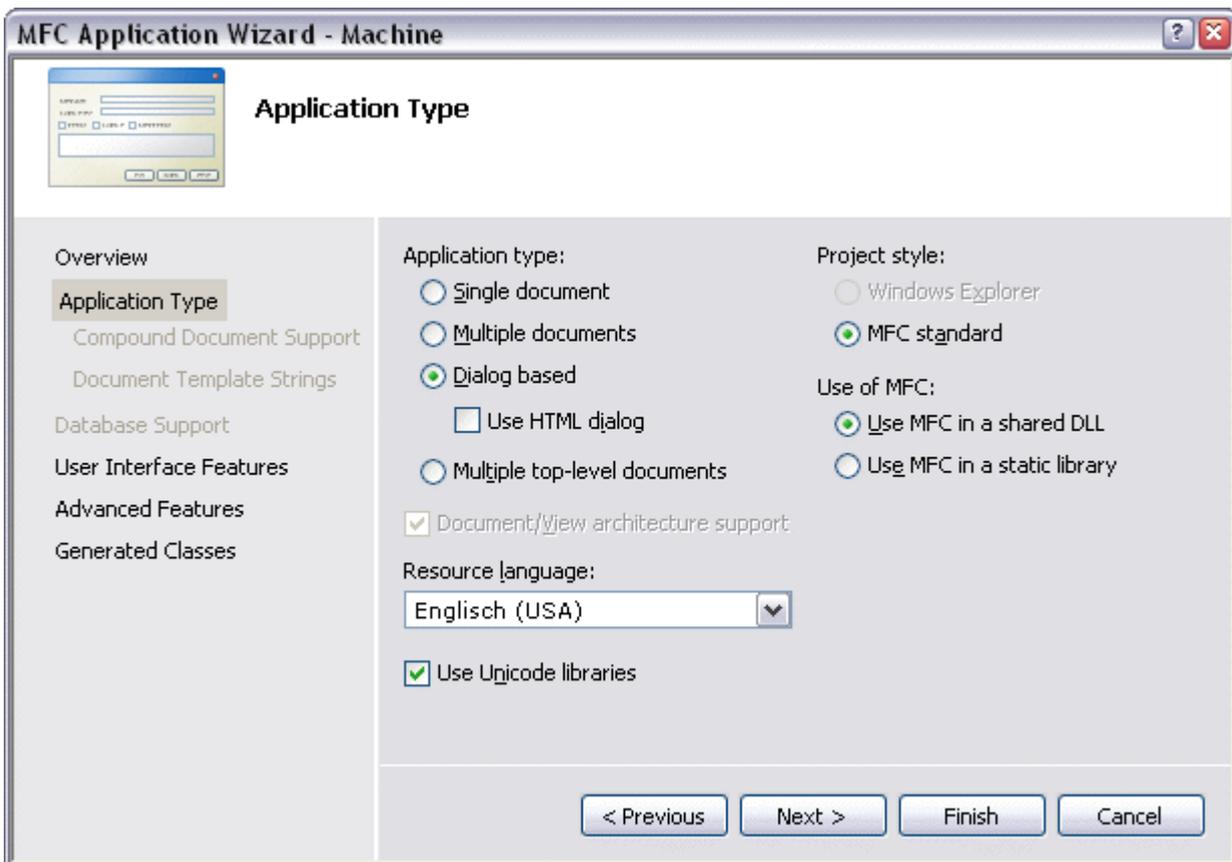
Schritt für Schritt lernen Sie die Entwicklung eines C++-Programms und das Einbinden der TwinCAT ADS DLL .LIB Komponente anhand eines Beispiels kennen.

1. Neues Projekt erstellen:

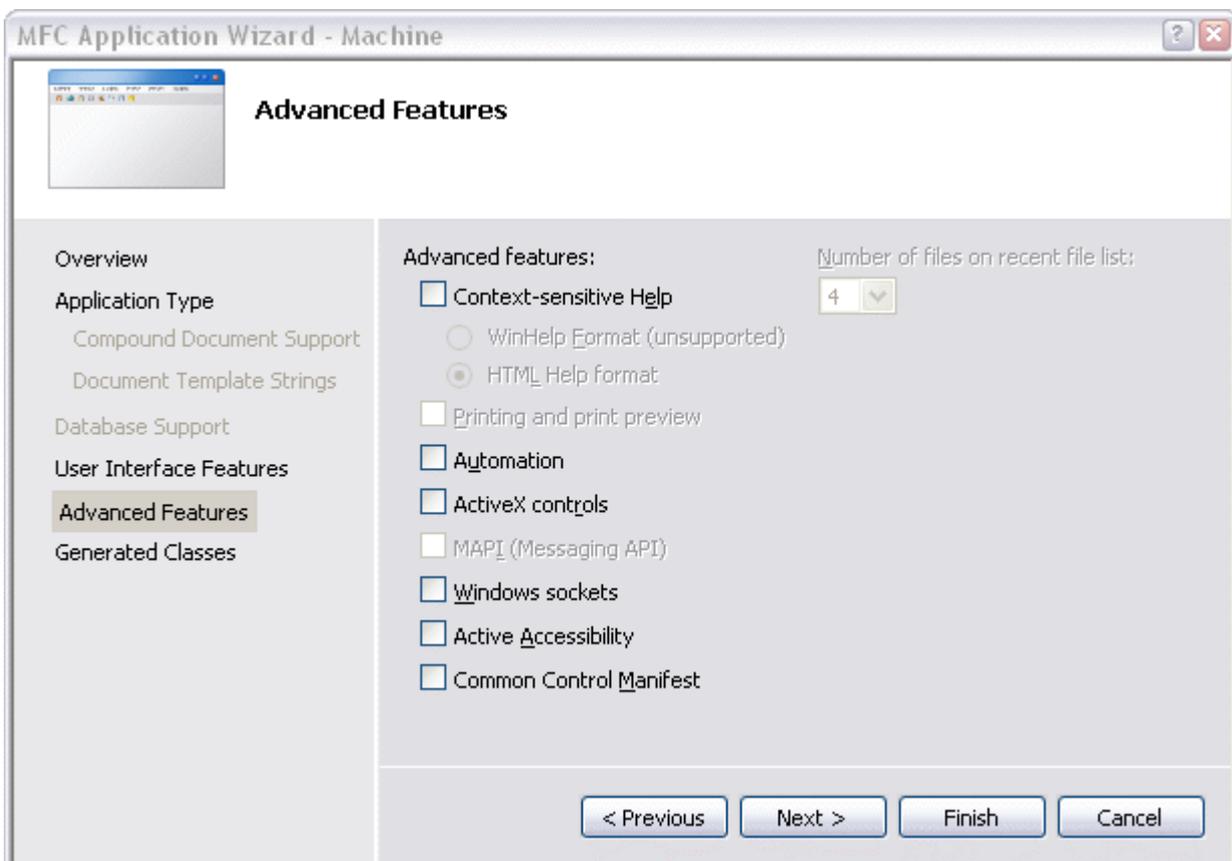
Starten Sie das Visual Studio 2005. Erstellen Sie ein neues Projekt, indem Sie im Menü 'File -> New -> Project...' anklicken. Im Dialogfeld 'New Project' können Sie ein neues Projekt auf der Basis verschiedener Vorlagen erstellen. Hier wählen Sie unter 'Project Types' die Programmiersprache 'Visual C++', 'MFC' und unter den Vorlagen 'MFC Application'. Geben Sie Ihrem Projekt einen neuen Namen, in diesem Fall bitte den Namen 'Machine' eingeben. Wählen sie dann den Verzeichnispfad unter 'Location' in dem Sie ihr Arbeitsprojekt erstellen wollen.



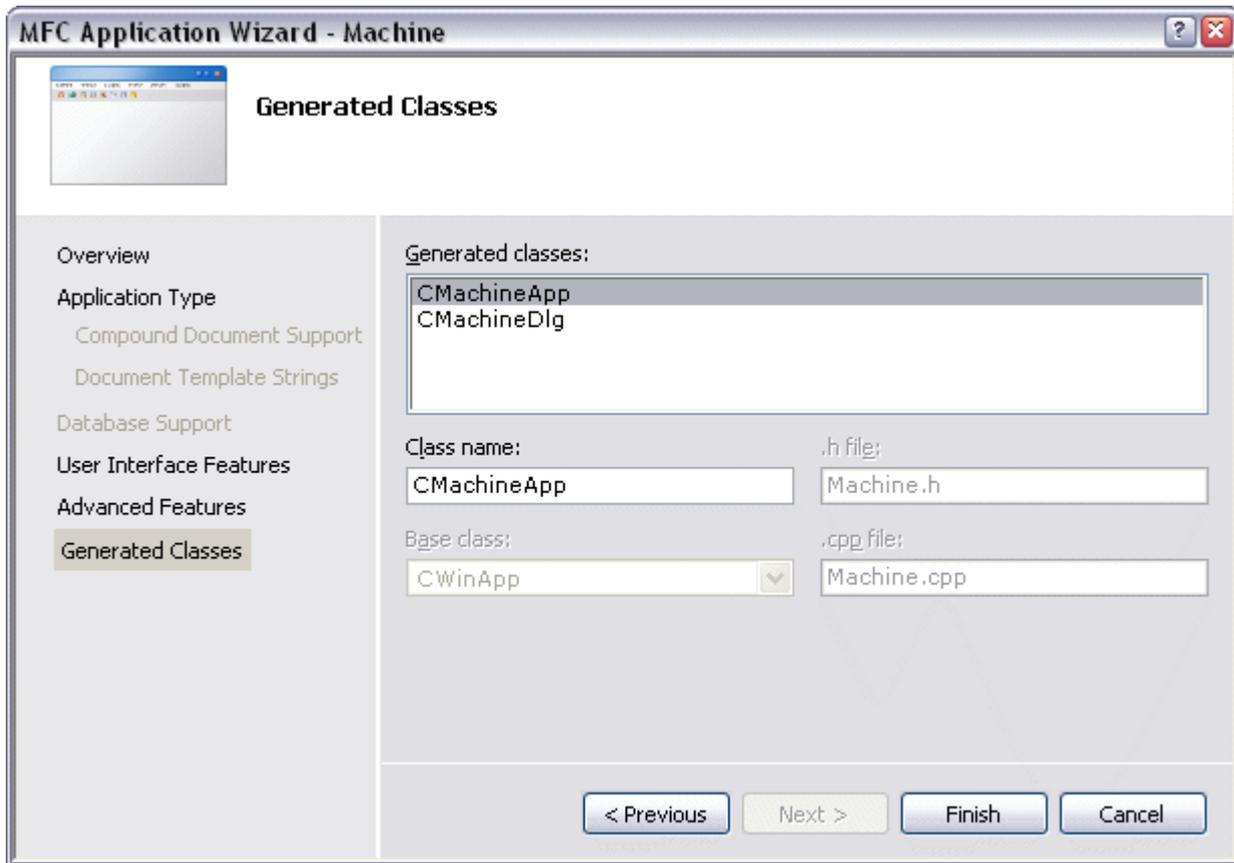
Nach der Begrüßung durch den Wizard, übernehmen Sie bitte folgende Einstellungen.



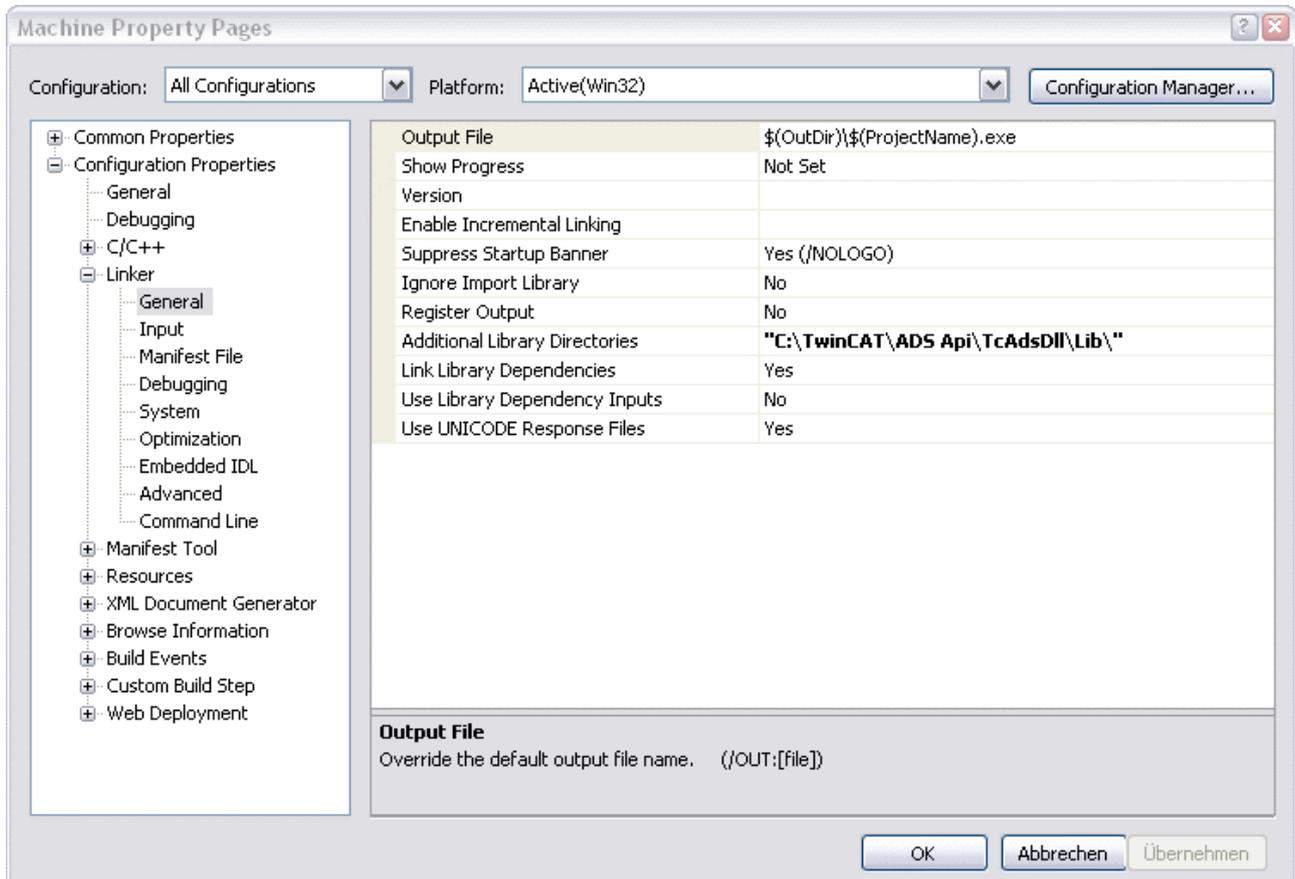
Weitere Features werden nicht benötigt.



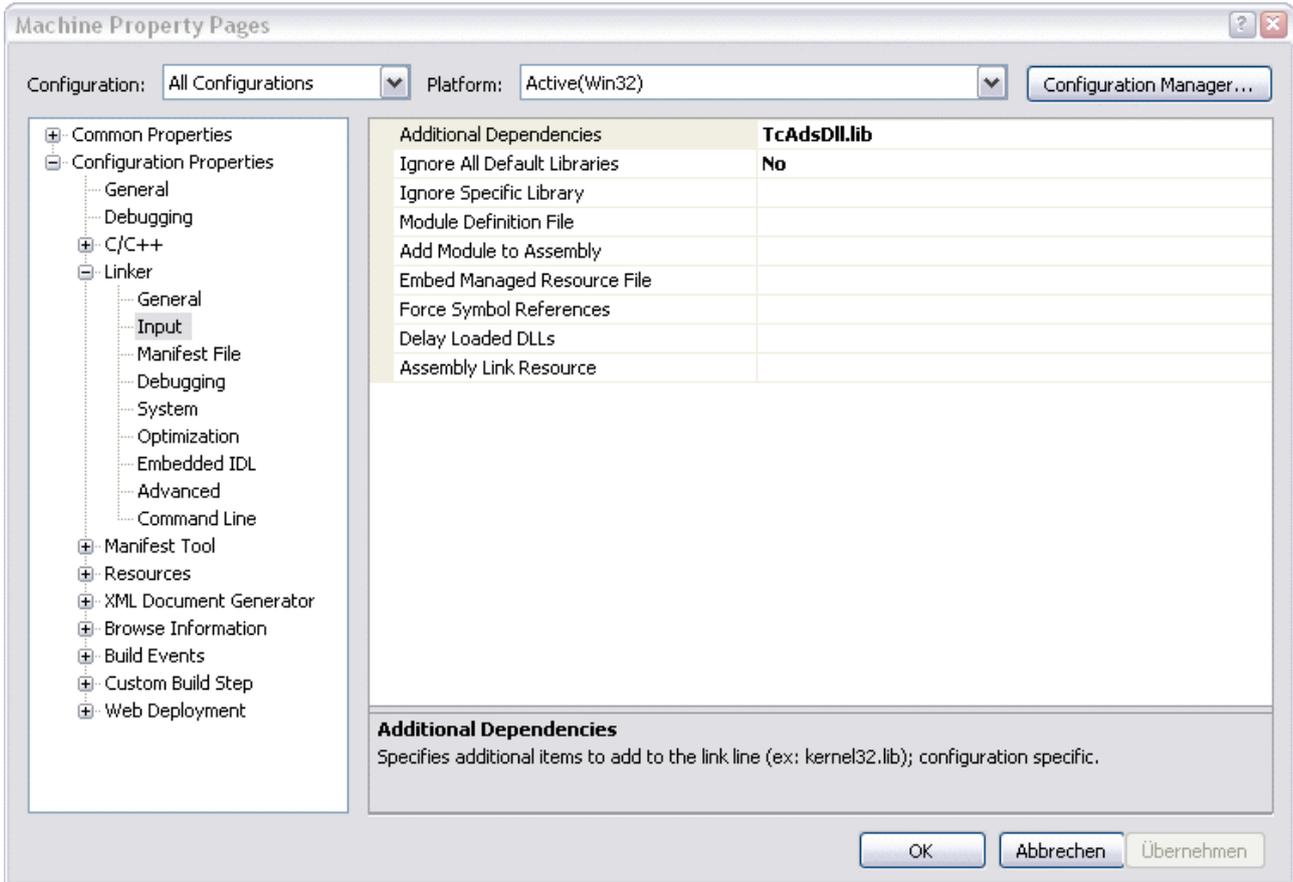
Letzter Schritt im Wizard. Hier kann alles so bleiben.



Um das Grundgerüst zu komplettieren und damit es zum Schluss nicht vergessen wird, verlinken wir gleich die benötigte TcAdsDll.lib. Der Pfad dazu wird unter 'Project -> Machine Properties... -> Configuration Properties -> Linker -> General -> Additional Library Directories' eingetragen.



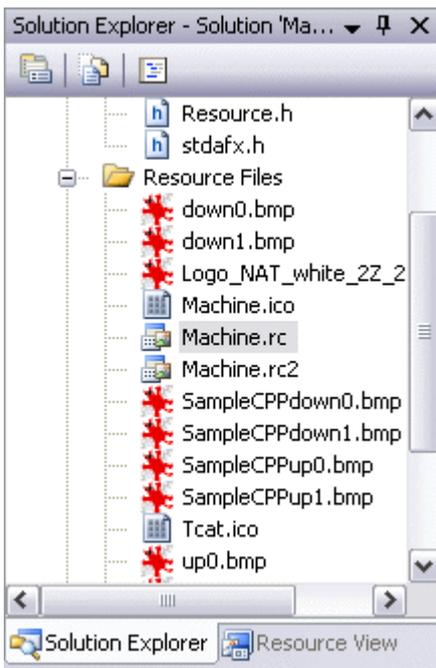
Zudem ist ein weiterer Eintrag unter 'Project -> Machine Properties... -> Configuration Properties -> Linker -> Input -> Additional Dependencies' notwendig.



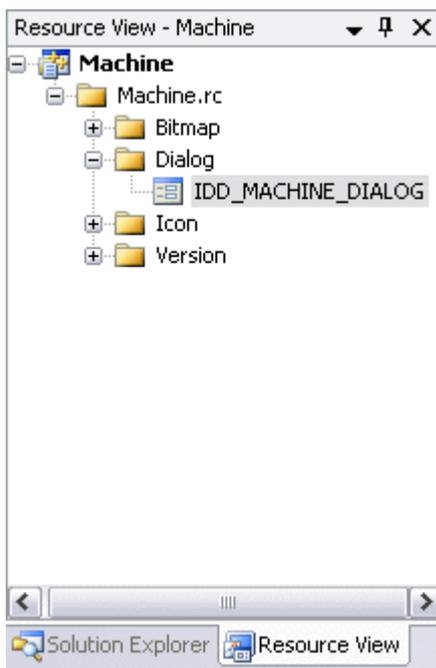
Bitte hierbei darauf achten, dass oben links bei 'Configuration:' auch 'All Configurations' ausgewählt ist!

2. Bedienungsoberfläche erstellen

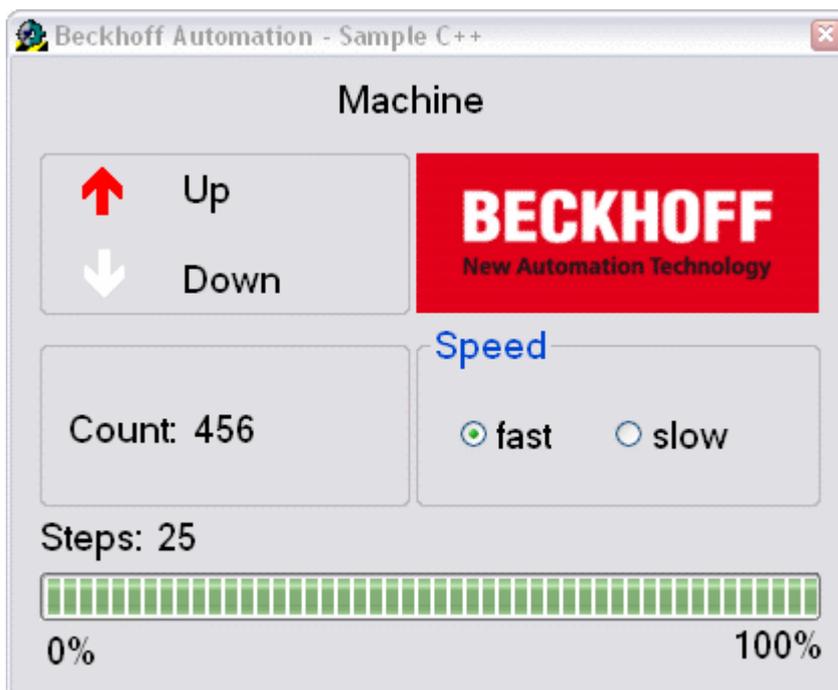
Hierzu wird zuerst eine Oberfläche im Design Modus der Form 'Machine.rc' mit Hilfe der Toolbox erstellt. Dazu öffnen Sie mit einem Doppelklick 'Machine.rc' im 'Solution Explorer'.



Sie befinden sich nun im 'Resource View'. Durch einen weiteren Doppelklick auf 'IDD_MACHINE_DIALOG' wird die Form sichtbar, auf der Sie jetzt alle benötigten Controls (wie z.B. 9 Static Texts, 3 PictureBoxes, 2 Radiobuttons, 1 Progressbar, GroupBoxes...) platzieren können.



Das Resultat sollte jetzt in etwa so aussehen.



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf die Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Speed' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Steps' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

3. Quelltext bearbeiten

Nach dem Erstellen der Oberfläche und dem Einbinden der TwinCAT DLL.LIB Komponente kann zum C++ Quelltext gewechselt werden. Im oberen Bereich von 'MachineDlg.h' binden wir die "TcAdsDef.h" und "TcAdsApi.h" ein. Weiter unten im Quelltext nehmen wir unsere Deklarationen vor.

```
#include "c:\twincat\ads_api\tcadsdll\include\tcadsdef.h" #include "c:\twincat\ads_api\tcadsdll\include\tcadsapi.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, hSwitch, hNotify;
};
```

Die Methode 'CMachineDlg::OnInitDialog()' wird beim Start der MFC Application aufgerufen. In ihr werden Instanzen verschiedener Klassen erzeugt, eine Verbindung mit dem Laufzeitsystem 1 der Komponente TcAdsDll.lib über den Port 801 hergestellt, benötigte Bitmaps geladen und die Handles der SPS-Variablen erstellt.

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

    [...]

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

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

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

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

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

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

```

    }
    else
    {
        pRB = ( CButton* )GetDlgItem( IDC_rdoFast ); pRB->SetCheck(0);
        pRB = ( CButton* )GetDlgItem( IDC_rdoSlow ); pRB->SetCheck(1);
    }
    nNotify = AdsSyncWriteReq( pAddr, ADSIGRP_SYM_RELEASEHND, 0x0, sizeof(ULONG), &hNotify );

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

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

```

SPS Variablen ausgeben:

In 'CMachineDlg::OnTimer()' ist die Ausgabe der Variablen organisiert. Der Parameter '*nElapse*' der Methode '*SetTimer()*' (der bereits im '*CMachineDlg::OnInitDialog()*' angegeben wurde) bestimmt die Zeit (in ms) die vergehen soll, bis der Timer erneut ausgeführt wird.

```

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
    anpassen 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 );
};

```

Zum Verbinden der Variablen wurde die Methode '*AdsSyncReadWriteReq()*' verwendet.

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

- **pAddr:** Struktur mit NetId und Portnummer vom ADS-Server.
- **nIndexGroup:** Index Group.
- **nIndexOffset:** Index Offset.
- **nReadLength:** Länge der Daten in Byte, die das ADS-Gerät zurückliefert.
- **pReadData:** Buffer mit Daten, die das ADS-Gerät zurückliefert.
- **nWriteLength:** Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden.
- **pWriteData:** Buffer mit Daten, die in das ADS-Gerät geschrieben werden.

Zum Verbinden der Variable 'hSwitch' wurde ebenfalls die Methode 'AdsSyncReadWriteReq()' verwendet. Einziger Unterschied ist hier, dass statt 'AdsSyncWriteReq()' (schreiben der Variablen) 'AdsSyncReadReq()' (lesen der Variablen) benutzt wird.

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

- **pAddr:** Struktur mit NetId und Portnummer vom ADS-Server.
- **nIndexGroup:** Index Group.
- **nIndexOffset:** Index Offset.
- **nLength:** Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **pData:** Zeiger auf Daten, die in den ADS-Server geschrieben werden.

Es fehlen noch zwei Methoden, um die Geschwindigkeit der Maschine zu regeln. Dazu einfach in die Design Ansicht wechseln und je einen Doppelklick auf den Radio Button für 'fast' und für 'slow' ausführen. Nun brauchen wir nur noch den entsprechenden Code einfügen.

Code für Radio Button 'fast'.

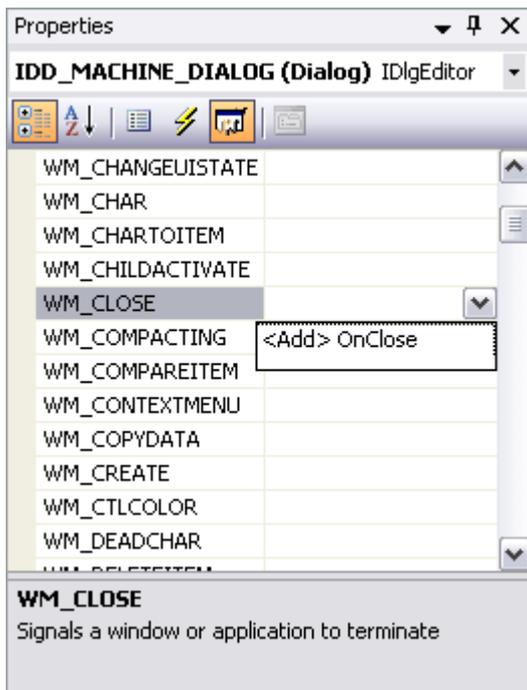
```
void CMachineDlg::OnBnClickedRadio1()
{
    // Anpassen der "Switch"-
    Variable, bei Klick auf Radio Button "fast"// Fits the switch variable if the radio button "fast" was
    s clicked
    bSwitch = true;
    nSwitch = AdsSyncReadWriteReq( pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(ULONG), &hSwitch,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

Code für Radio Button 'slow'.

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

Handles löschen:

Zum Schluss müssen wir nur noch die erstellten Handles wieder löschen. Hierzu öffnen wir die 'Properties' der Form und stellen bei den kleinen oberen Icons von 'Properties' auf 'Messages'. Danach suchen wir im unteren Fenster die 'WM_CLOSE' Message raus und wählen per DropDown '<Add> OnClose'.



Wir landen sofort an der entsprechenden Codestelle an der nur noch nachfolgender Code eingefügt werden muss.

```
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();
}
```

Das SPS Programm Machine_Final.prosolite auf dem Laufzeitsystem 1 starten und das erstellte C++-Programm Machine.exe kann getestet werden.

Download C++ Programm Beispiel:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281694219.zip>

7.4.5 Beispiel Maschine mit Microsoft Expression Blend

Der Microsoft Expression Blend ist ein Programm zum Erstellen von Programmoberflächen für C# und Visual Basic. In diesem Beispiel wird eine mit dem Programm erstellte Oberfläche mit dem Beispiel Machine verbunden. Dabei wurde die Programmiersprache C# verwendet.

Erforderliche Software:

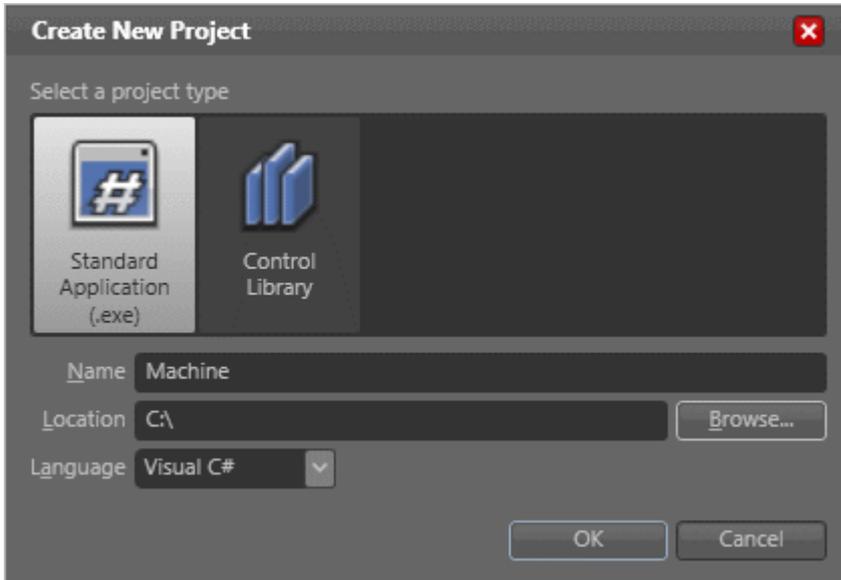
- Microsoft .NET Framework Version 3.0, mehr dazu unter www.microsoft.com
- Microsoft Expression Blend, mehr dazu unter www.microsoft.com
- Microsoft Visual Studio 2005
- TwinCAT 2.10

Die ersten Schritte ...

Die Entwicklung eines Programms mit dem Microsoft Expression Blend und das Einbinden der TwinCAT ADS .NET Komponente wird anhand eines Beispiels beschrieben.

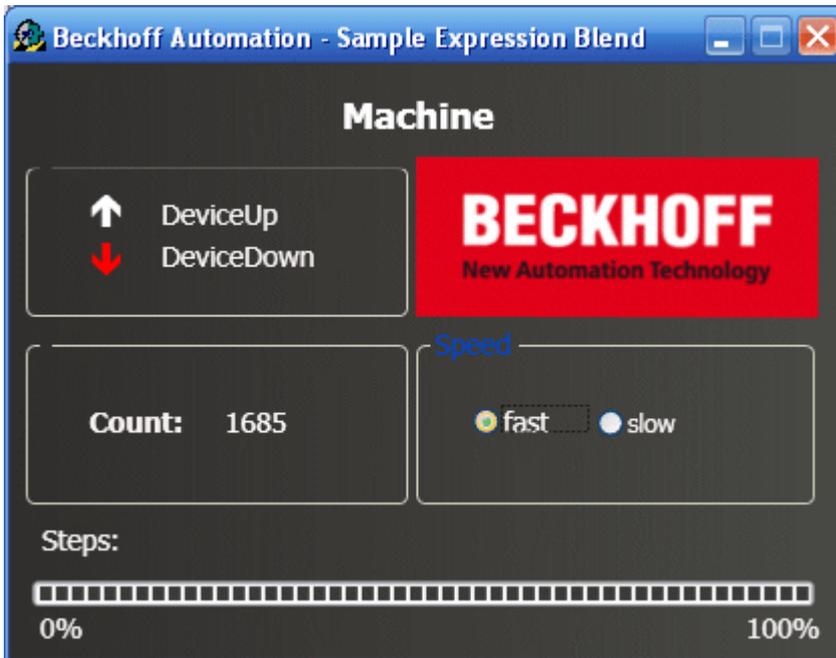
1. Neues Projekt erstellen:

Starten des Expression Blend und Erstellen einer neuen Oberfläche, über Menü -> File -> New Project... . Das Dialogfeld 'Create New Application' öffnet sich und es kann der Typ, der Name, die Location und die Programmiersprache ausgewählt werden. In diesem Fall bitte den Typ 'Standard Application', den Namen 'Machine' und die Programmiersprache 'C#' wählen.



2. Bedienungsoberfläche erstellen

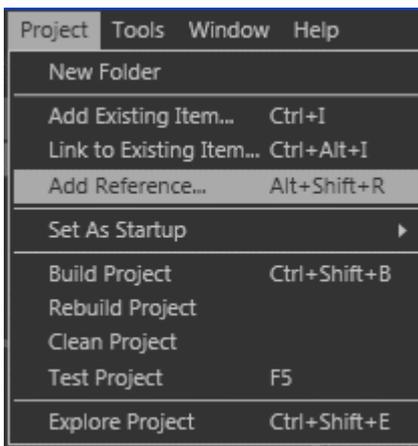
Die Einstellungen der Oberfläche werden in der Datei Window1.xaml abgelegt.



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf die Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Speed' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Steps' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

3. Referenz hinzufügen

Nach dem Erstellen der Oberfläche muss zuerst eine Referenz namens TwinCAT.Ads.dll hinzugefügt werden. Die erfolgt über das Menü --> Project --> Add Reference.



4. Quelltext bearbeiten

Nach dem Erstellen der Oberfläche und das Einbinden der TwinCAT ADS Komponente kann zum C# Quelltext, der in Visual Studio angezeigt werden kann, gewechselt werden. In die oberste Zeile des Quelltextes werden die benötigten Namespaces 'System.IO' und 'TwinCAT.Ads' eingefügt.

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

Danach folgen die Deklarationen.

```
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;
```

Die erste Methode ist die Load Methode. In ihr werden Instanzen verschiedener Klassen erzeugt und eine Verbindung zum Port 801 hergestellt.

```
//-----//
Wird als erstes beim Starten des Programms aufgerufen//-----
-----private void Load(object sender, EventArgs e)
{
    try
    {
        // Eine neue Instanz der Klasse AdsStream erzeugen
        dataStream = new AdsStream(7);

        // Eine neue Instanz der Klasse BinaryReader erzeugen
        binReader = new BinaryReader(dataStream);

        // Eine neue Instanz der Klasse TcAdsClient erzeugen
        tcClient = new TcAdsClient();

        // Verbinden mit lokaler SPS - Laufzeit 1 - Port 801
        tcClient.Connect(801);
    }
    catch
    {
        MessageBox.Show("Fehler beim Laden");
    }

    //...
}
```

Dann werden in der Methode Load die Variablen noch verbunden und mit einer Methode (siehe "Methode 'tcClient_OnNotification' schreiben") verbunden, die bei einer Änderung einer Variable aufgerufen wird.

```
try
{
    // Initialisieren der Überwachung der SPS-Variablen
    hEngine = tcClient.AddDeviceNotification(".engine", dataStream, 0, 1, AdsTransMode.OnChange, 10,
0, null);
    hDeviceUp = tcClient.AddDeviceNotification(".deviceUp", dataStream, 1, 1, AdsTransMode.OnChange,
```

```

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

    // Holen des Handles von "switch" - wird für das Schreiben des Wertes benötigt
    hSwitchWrite = tcClient.CreateVariableHandle(".switch");

    // Erstellen eines Events für Änderungen an den SPS-Variablen-Werten
    tcClient.AdsNotification += new AdsNotificationEventHandler(tcClient_OnNotification);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

SPS Variablen verbinden:

Zum Verbinden der Variablen wurde die Methode `AddDeviceNotification` verwendet.

```

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

```

- **variableName:** Name der SPS Variablen.
- **dataStream:** Der Datenstrom, der die Daten empfängt.
- **offset:** Abstand im Datenstrom.
- **length:** Länge im Datenstrom.
- **transMode:** Das Ereignis, wenn sich die Variable ändert.
- **cycleTime:** Die Zeit (in ms) nach der der SPS-Server überprüft, ob sich die Variable geändert hat.
- **maxDelay:** Die Zeit (in ms) nach der das Ereignis spätestens beendet ist.
- **userData:** Das Objekt, das zum Ablegen bestimmter Daten verwendet werden kann.

Zum Verbinden der Variable `hSwitchWrite` wurde die Methode `CreateVariableHandle` verwendet.

```

int TcAdsClient.CreateVariableHandle(string variableName);

```

- **variableName:** Name der SPS-Variablen.

Methode 'tcClient_OnNotification' schreiben:

Diese Methode wird aufgerufen, wenn sich eine der SPS-Variablen geändert hat.

```

//-----//wird bei Änderung einer SPS-
Variablen aufgerufen//-----
private void tcClient_OnNotification(object sender, AdsNotificationEventArgs e)
{
    try
    {
        // Setzen der Position von e.DataStream auf die des aktuellen benötigten Wertes
        e.DataStream.Position = e.Offset;

        // Ermittlung welche Variable sich geändert hat if(e.NotificationHandle == hDeviceUp)
        {
            //
            Die Farben der Grafiken entsprechen der Variablen anpassen if (binReader.ReadBoolean() == true)
            {
                DeviceUp_LED.Foreground = new SolidColorBrush(Colors.Red);
            }
            else
            {
                DeviceUp_LED.Foreground = new SolidColorBrush(Colors.White);
            }
        }
    }
    else if(e.NotificationHandle == hDeviceDown)

```

```

    {
        if (binReader.ReadBoolean() == true)
        {
            DeviceDown_LED.Foreground = new SolidColorBrush(Colors.Red);
        }
        else
        {
            DeviceDown_LED.Foreground = new SolidColorBrush(Colors.White);
        }
    }
else if(e.NotificationHandle == hSteps)
{
    // Einstellen der ProgressBar auf den aktuellen Schritt
    prgSteps.Value = binReader.ReadByte();
}
else if(e.NotificationHandle == hCount)
{
    // Anzeigen des "count"-Werts
    lblCount.Text = binReader.ReadUInt16().ToString();
}
else if(e.NotificationHandle == hSwitchNotify)
{
    // Markieren des korrekten RadioButtonsif (binReader.ReadBoolean() == true)
    {
        optSpeedFast.IsChecked = true;
    }
    else
    {
        optSpeedSlow.IsChecked = true;
    }
}
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
}

```

Es fehlen noch zwei Methoden, mit denen die Maschine schneller oder langsamer fährt. In Ihnen wird ein virtueller Schalter umgelegt, hier wird ein Wert in die SPS-Variable switch geschrieben.

```

//-----//
wird aufgerufen, wenn das Feld 'slow' markiert wird//-----
-----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 'fast' markiert wird//-----
-----private void optSpeedSlow_Click(object sender, EventArgs e)
{
    try
    {
        tcClient.WriteAny(hSwitchWrite, false);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
}

```

Damit die Methoden beim richtigen Ereignis aufgerufen werden, muss in der Design Ansicht DocumentRoot markiert werden, bei Events auf add geklickt und loaded gewählt, und die Methode Load ausgewählt werden.

Gleiches gilt für die beiden RadioButtons, allerdings wird hier das Event 'Click' ausgewählt, sowie die Methode 'optSpeedFast_Click' bzw. 'optSpeedSlow_Click_Click'.

Notifications und Handles löschen:

In dem Close-Ereignis des Fensters werden die Verbindungen wieder mit der Methode `DeleteDeviceNotification()` freigegeben.

```
private void Close(object sender, EventArgs e)
{
    try
    {
        // Löschen der Notifications und 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();
}
```

Das SPS Programm `Machine_Final.pro` sollte auf dem Laufzeitsystem 1 starten und das erstellte C#-Programm `Machine.exe` kann getestet werden.

Download:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281695627.zip>

7.4.6 Beispiel Machine mit Microsoft Expression Blend in den Microsoft Windows Vista Media Center

Microsoft Expression Blend ist ein Programm zum Erstellen von Programmoberflächen für C# und Visual Basic. In diesem Beispiel wird eine, mit dem Programm erstellte, Oberfläche mit dem Beispiel Machine verbunden und das Ganze anschließend in den Vista Media Center eingebunden. Dabei wurde die Programmiersprache C# verwendet.

Zielplattform

- Windows Vista

Implementierung

- Visual C#

Erforderliche Software

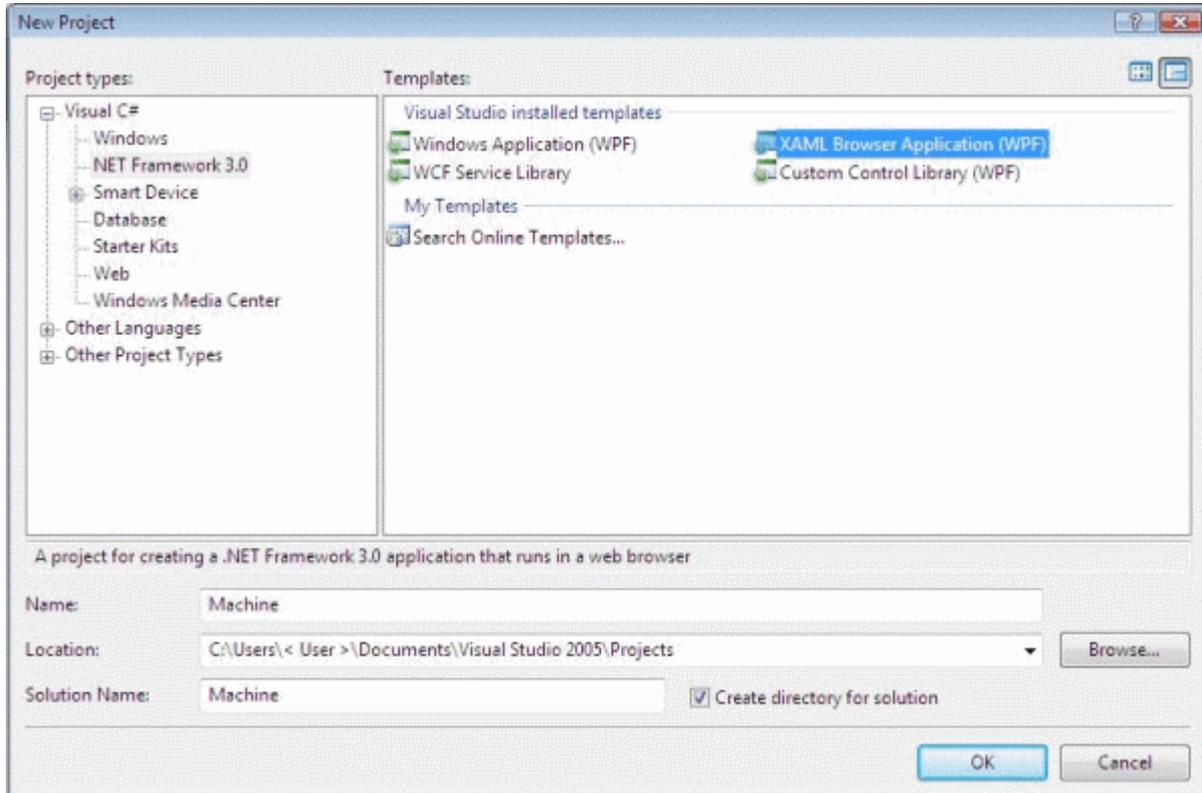
- Microsoft .NET Framework Version 3.0
- Microsoft Expression Blend
- Microsoft Visual Studio 2005
- Microsoft Windows Vista Media Center
- Microsoft Windows SDK für .Net Framework 3.0
- Microsoft Visual Studio 2005 extensions für .Net Framework 3.0 (November 2006 CTP)
- TwinCAT 2.10
- Notepad oder einen anderen Texteditor

Die ersten Schritte ...

Schritt für Schritt lernen Sie die Entwicklung eines Programms mit Microsoft Visual Studio und Microsoft Expression Blend und das Einbinden der TwinCAT ADS .NET Komponente anhand eines Beispiels kennen und binden dieses dann in den Vista Media Center ein.

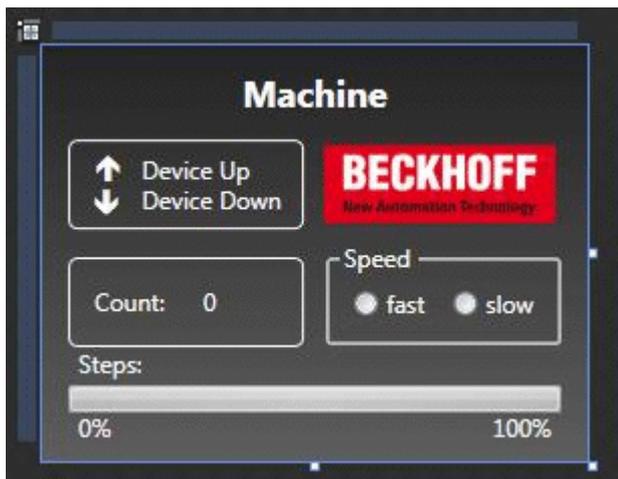
1. Neues Projekt erstellen:

Starten Sie Microsoft Visual Studio und erstellen Sie ein neue XAML Browser Applikation. Dazu gehen Sie über das Menü unter 'File -> New -> Project...' . Das Dialogfeld 'New Project' öffnet sich. Zuerst wählen Sie den Projekt Typ aus: 'Project types -> Visual C# -> Net Framework 3.0'. Rechts erscheinen dann die Templates des Projekt Typs. Dort wählen Sie 'XAML Browser Application'. Sie geben ihrem Projekt jetzt noch einen Namen, in diesem Fall ist das Machine und legen die Location fest.



2. Bedienungsoberfläche erstellen

Sie wechseln jetzt zu Microsoft Expression Blend, wo Sie Ihr eben erstelltes Projekt öffnen, um dort die Bedienoberfläche zu erstellen.

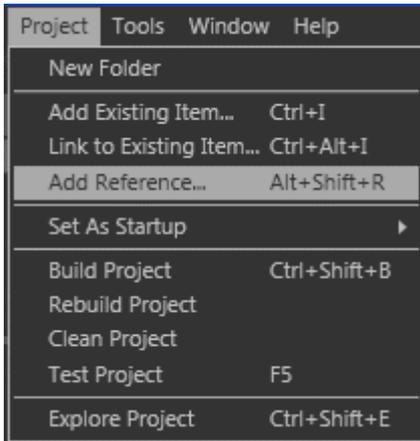


Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf den Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld 'Speed' die Taktgeschwindigkeit des Motors verändern. Die Anzeige 'Steps' entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

Wollen Sie, dass die Bedienoberfläche ständig ihre Größe anpasst, so kopieren sie den obersten Grid und fügen dann anstelle des Grids eine Viewbox ein. In diese Viewbox fügen Sie nun den Grid ein. Jetzt müssen Sie nur noch die Größe der Seite, der Viewbox und des Grids auf 'Auto' setzen. Hier bei kann es zu einer Verschiebung von Elementen kommen. Diese müssen Sie dann erneut positionieren. Achten Sie dabei darauf, dass Sie die Größe der Seite, der Viewbox und des Grids nicht ausversehen wieder fest setzen.

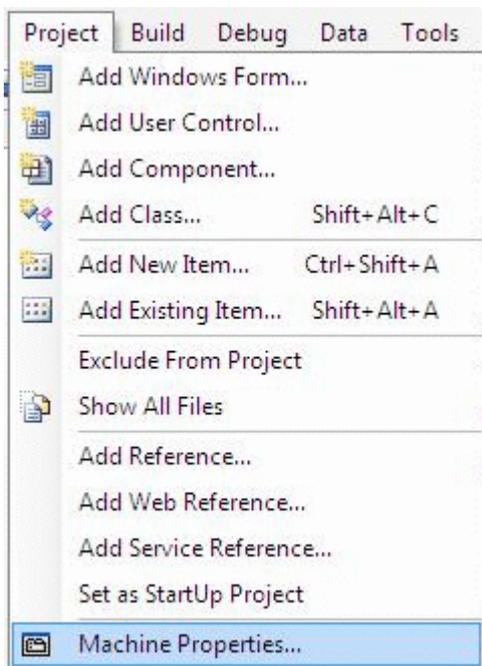
3. Referenz hinzufügen

Nach dem Erstellen der Oberfläche muss zuerst eine Referenz namens 'TwinCAT.Ads.dll' hinzugefügt werden. Dies kann sowohl in Visual Studio, als auch in Expression Blend erfolgen. In beiden Fällen geht man über das Menü unter 'Project --> Add Reference'.

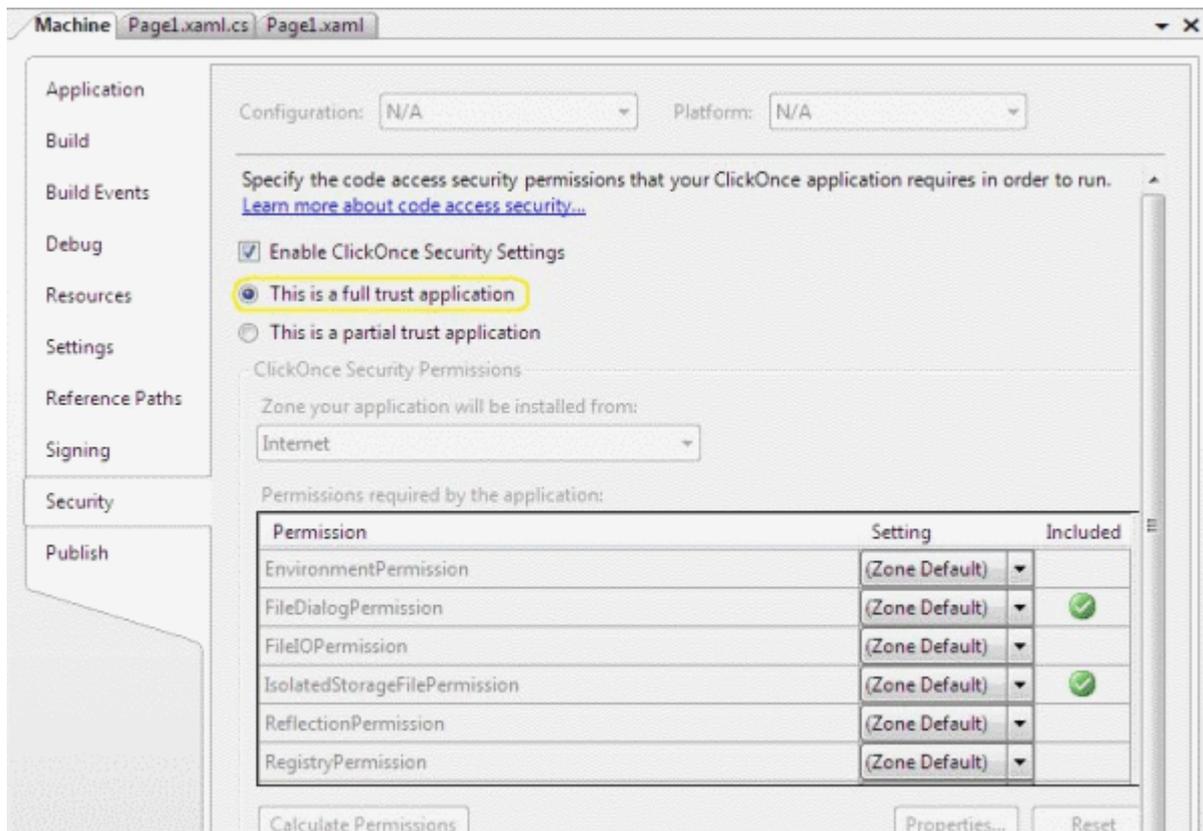


4. Sicherheitsfreigabe

Gehen Sie im Menü unter 'Project -> <Name ihres Projektes> Properties....' .



Es öffnet sich nun ein Tab, in dem Sie die Projekteigenschaften festlegen können. Gehen Sie dort auf 'Security' und wählen Sie 'this is a full trust application' aus.



5. Quelltext bearbeiten

Nun kann mit dem Erstellen des Quelltextes in C# begonnen werden.

In die oberste Zeile des Quelltextes werden die benötigten Namespaces 'System.IO' und 'TwinCAT.Ads' eingefügt.

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

Danach folgen die Deklarationen.

```
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;
```

Die erste Methode, ist die 'Load' Methode. In ihr werden Instanzen verschiedener Klassen erzeugt und eine Verbindung zum Port 801 hergestellt.

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

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

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

```
// Verbinden mit lokaler SPS - Laufzeit 1 - Port 801// Connecting to local PLC - Runtime 1 -
Port 801
tcClient.Connect(801);
}
catch
{
    MessageBox.Show("Fehler beim Laden");
}

//...
```

Dann werden in der Methode 'Load' die Variablen noch verbunden und mit einer Methode (die noch geschrieben werden muss) verknüpft, die bei einer Änderung einer Variable aufgerufen wird.

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

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

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

6. Definition

SPS Variablen verbinden:

Zum Verbinden der Variablen wurde die Methode AddDeviceNotification verwendet.

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

- **variableName:** Name der SPS Variablen.
- **dataStream:** Der Datenstrom, der die Daten empfängt.
- **offset:** Abstand im Datenstrom.
- **length:** Länge im Datenstrom.
- **transMode:** Das Ereignis, wenn sich die Variable ändert.
- **cycleTime:** Die Zeit (in ms) nach der der SPS-Server überprüft, ob sich die Variable geändert hat.
- **maxDelay:** Die Zeit (in ms) nach der das Ereignis spätestens beendet ist.
- **userData:** Das Objekt, das zum Ablegen bestimmter Daten verwendet werden kann.

Zum Verbinden der Variable 'hSwitchWrite' wurde die Methode CreateVariableHandle verwendet.

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

- **variableName:** Name der SPS-Variablen.

7. Methode schreiben:

Oben wurde bereits auf eine Methode verwiesen, die noch gar nicht existiert. Daher wird diese Methode, die 'tcClient_OnNotification' genannt wurde, als nächstes geschrieben. Diese Methode wird aufgerufen, wenn sich eine der SPS-Variable geändert hat.

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

Es fehlen noch zwei Methoden, mit denen die Geschwindigkeit der Maschine eingestellt werden kann. In Ihnen wird ein virtueller Schalter umgelegt, hier wird ein Wert in die SPS-Variable switch geschrieben.

```
//-----// 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. Notifications und Handles löschen:

In dem Close-Ereignis des Fensters werden die Verbindungen wieder mit der Methode `DeleteDeviceNotification()` freigegeben.

```

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

```

Zu guter Letzt muss noch dafür gesorgt werden, dass die Methoden auch beim richtigen Ereignis aufgerufen werden. Dazu in Expression Blend gehen, Page auswählen, bei Properties auf Events wechseln und bei 'Loaded' 'Load' reinschreiben und bei 'Unloaded' 'Close'.

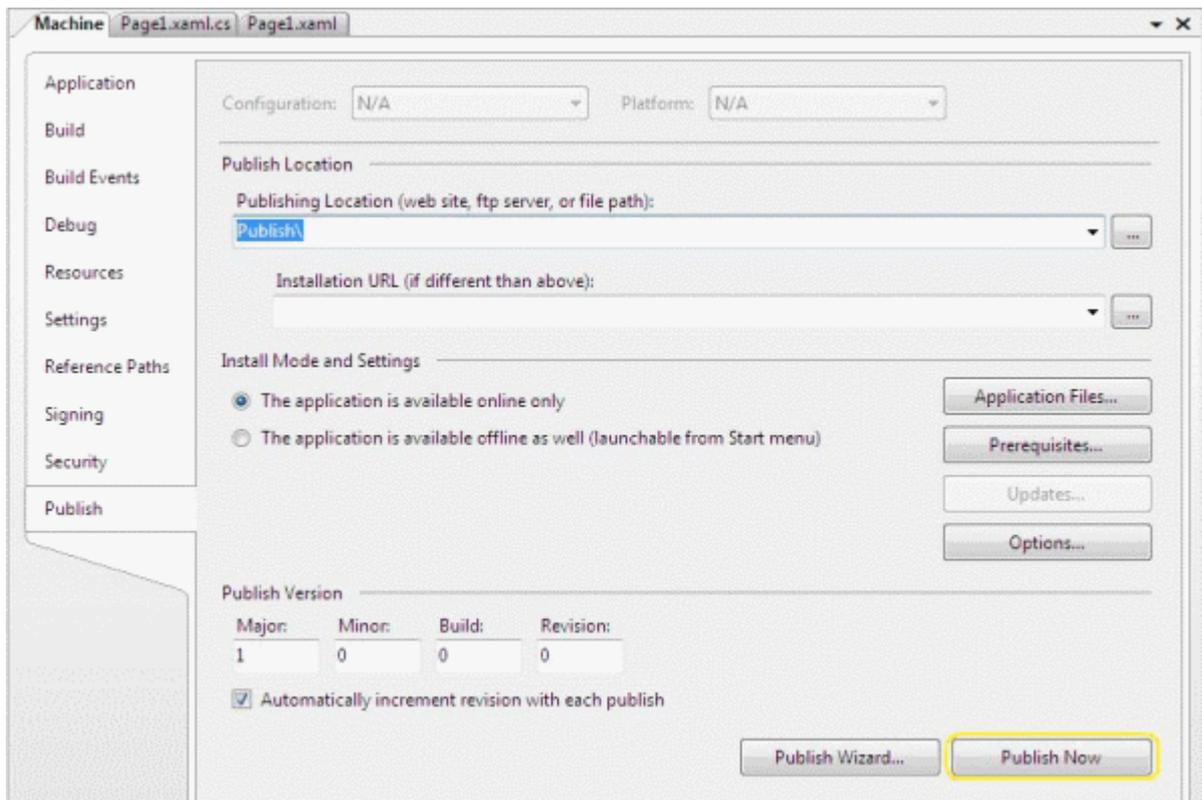
Das Gleiche muss auch noch mit den beiden RadioButtons gemacht werden, nur das hierbei das Event 'Click' ausgewählt wird, sowie die Methode 'optSpeedFast_Click' bzw. 'optSpeedSlow_Click'.

Das SPS Maschinenprogramm `Machine_Final.pro` muss auf dem Laufzeitsystem 1 laufen und das Programm kann im Internet Explorer 7 getestet werden.

9. Einbinden in den Vista Media Center

Habe Sie ihr Projekt hinreichend getestet und keinen Fehler festgestellt, dann können Sie es nun in den Media Center einbinden.

Rufen Sie wieder in Visual Studio die Projekteigenschaften auf, jedoch gehen Sie dann auf 'Publish'. Dort klicken Sie dann auf 'Publish Now'. Es wird nun die `xbap-Datei` erstellt die Sie später im Media Center aufrufen. Diesen Schritt müssen Sie immer machen, wenn Sie ihr Programm verändert haben und die Änderung auch in den Media Center übernommen werden soll.



Gehen Sie jetzt in einen Texteditor, z.B. Notepad und geben Sie folgendes ein:

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

Speichern Sie es unter: 'C:\Users\\AppData\Roaming\Media Center Programs\Machine.mcl'. Wenn Sie nun Ihren Media Center Starten finden Sie ihr Programm unter 'Online Media -> program library -> programs by name -> Machine'.

Dies ist die einfachste Form der Einbindung in den Windows Vista Media Center. Weitere Informationen zur Einbindung in den Media Center finden Sie [hier](#).

10. Download Expression Blend Beispiel:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281697035.zip>

7.4.7 Beispiel Machine mit Microsoft Silverlight und JavaScript

Microsoft Silverlight ist eine Web-Präsentationstechnik, die durch ein entsprechendes Plugin von allen gängigen Browsern (Internet Explorer 6/7, Mozilla Firefox, Apple Safari und Opera) dargestellt werden kann.

Zielplattformen

- Windows XP, XPE, WES
- Windows Vista
- Windows 7

Implementierung

- JavaScript

Erforderliche Software:**- Runtime:**

- Microsoft Silverlight 1.0
- Microsoft Silverlight 1.1

Welche Runtime Sie benötigen hängt davon ab ob Sie eine Silverlight 1.0 oder 1.1 Applikation in Ihrem Browser darstellen wollen.

- Developer Tools:

- Microsoft Visual Studio 2008 Beta 2
 - + Microsoft Silverlight Tools Alpha Refresh for Visual Studio (July 2007)
- oder

- Microsoft Visual Studio 2005
- + Microsoft Silverlight 1.0 Software Development Kit

Für dieses Beispiel wurde Microsoft Visual Studio 2005 verwendet.

- Designer Tools:

- Expression Blend 2 August Preview

- Sonstige:

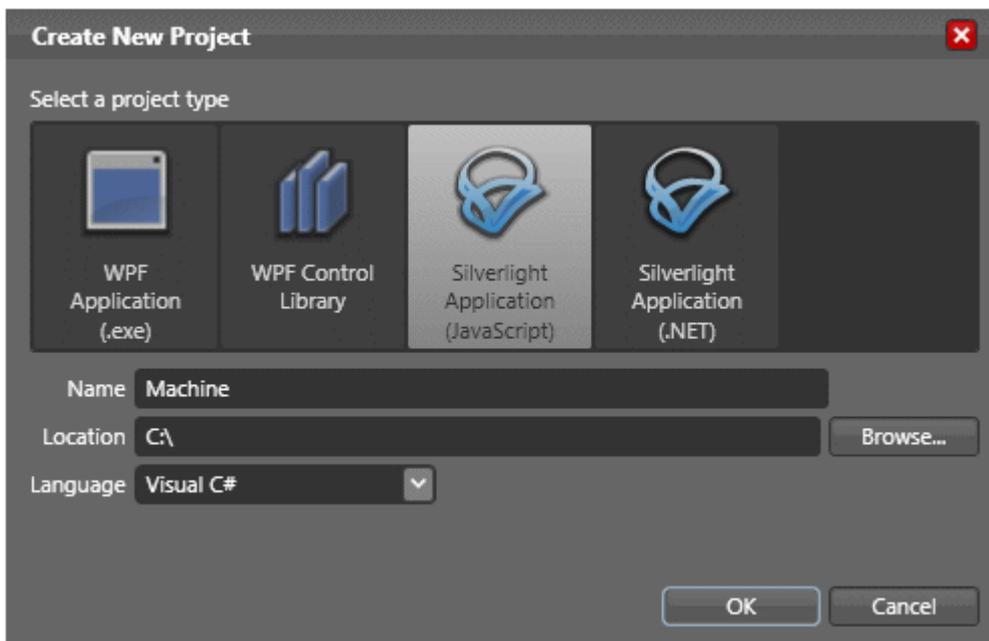
- TwinCAT 2.10
- Browser (z.B. Internet Explorer 7 oder Mozilla Firefox)
- Microsoft .NET Framework Version 3.0

Die ersten Schritte...

Schritt für Schritt lernen Sie die Entwicklung einer Silverlight-Applikation und das Einbinden des TwinCAT ADS Web Service anhand eines Beispiels kennen.

1. Neues Projekt erstellen:

Starten Sie Microsoft Expression Blend 2 und erstellen Sie eine neue Oberfläche über 'Menü → File → new Project...' . Das Dialogfeld 'Create New Projekt' öffnet sich und es kann nun der Typ, der Name, der Speicherort und die Programmiersprache ausgewählt werden. In diesem Beispiel wählen Sie den Typ 'Silverlight Application (JavaScript)' und den Namen 'Machine'.

**2. Bedienoberfläche erstellen:**

Für die Erstellung der Bedienoberfläche stehen nur wenige Controls zur Verfügung. Einen *RadioButton* können Sie mit einer *Textbox* und zwei *Ellipsen* erstellen, die in einem *Canvas* gepackt werden. Die *Progressbar* kann man mit drei *Rectangles* erstellen, die man auch wieder in ein separates *Canvas* packt. Die Einstellungen der Oberfläche werden in der Datei *Page.xaml* abgelegt.

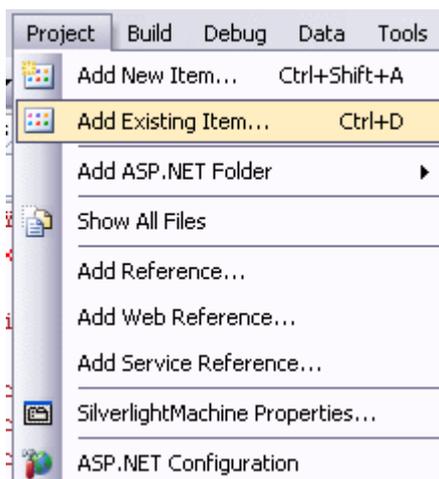
Beachten Sie, dass Sie bei keinem Objekt die Größe auf 'Auto' setzen. Dies kann sonst später zu Fehlern führen.



Im oberen linken Bereich sehen Sie die beiden Ausgänge, die auch auf den Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts können Sie mit dem Feld *Speed* die Taktgeschwindigkeit des Motors verändern. Die Anzeige *Steps* entspricht der Anzahl der Takte, die auf den Ausgang 1 ausgegeben werden.

3. Add XMLHTTP.JS

Im Visual Studio über 'Projekt → Add Existing Item...' fügen Sie die Datei 'XMLHTTP.JS' ein. Diese Datei enthält allgemeine Methoden zum Lesen und Schreiben von SPS-Variablen, sowie zum Konvertieren von Datentypen.



4. Quelltext bearbeiten

Öffnen Sie die Datei Default.html in Visual Studio und binden Sie im Kopf die Datei 'XMLHTTP.JS' ein.

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

Fügen Sie in der HTML Seite im HEAD-Bereich einen JavaScript-Bereich ein. Der folgende Quelltext muss dort eingefügt werden:

Zunächst müssen die wichtigsten Variablen deklariert werden.

```
<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;
...

```

Die URL des TcAdsWebService sowie die NetID und den Port müssen Sie entsprechend anpassen.

Auf die Objekte der Bedienoberfläche kann nicht ohne weiteres zugegriffen werden. Damit dies funktioniert werden die Objekte nach dem Start der Applikation den Variablen zugewiesen, die oben bereits deklariert wurden.

```
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");
}
```

Die Load Methode beinhaltet zwar das Zuweisen der Variablen, jedoch wird diese bisher nie aufgerufen. Damit dies geschieht muss in Expression Blend 2 der XAML-Code der Oberfläche geändert werden. Dazu muss das oberste *Canvas* um *Loaded="Load"* ergänzt werden.

```
<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);
}
```

SPS-Variable lesen

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

- **netId:** String, der die AMS-Net-Id angibt, auf der die SPS zu finden ist
- **nPort:** Port-Nummerdes Laufzeitsystems
- **indexGroup:** IndexGroup der SPS-Variable
- **indexOffset:** Erstes Byte, in das geschrieben werden soll
- **ncbLen:** Anzahl der Bytes, die geschrieben werden sollen

```
function init()
{
    b64s = "ABCDEFGHGIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-=";
    success = 0;
    errors = 0;
    loop(refresh); //send request every x seconds
}
```

Ergänzen Sie *<body>* um *onload="init()"*, damit die Methode beim Laden ausgeführt wird.

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

Die nächste Funktion sorgt dafür, dass die Werte ausgelesen und ausgegeben werden. Wichtig ist hier, dass die Adresse der Variable von Machine.pro zum Auslesen richtig angegeben wird.

```
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
    }
}

```

In den letzten beiden Methoden wird die SPS-Variable, über die die Geschwindigkeit der Maschine gesteuert wird, auf null bzw. eins gesetzt.

```
function Fast_MouseLeftButtonDown(sender, EventArgs)
{
    Write(netId, port, '16416', '6', '2', '0', 'int');
}
function Slow_MouseLeftButtonDown(sender, EventArgs)
{
    Write(netId, port, '16416', '6', '2', '1', 'int');
}
```

SPS Variablen schreiben

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

- **netId**: String, der die AMS-Net-Id angibt, auf der die SPS zu finden ist
- **nPort**: Port-Nummerdes Laufzeitsystems
- **indexGroup**: IndexGroup der SPS-Variable
- **indexOffset**: Erstes Byte, in das geschrieben werden soll
- **ncbLen**: Anzahl der Bytes, die geschrieben werden sollen
- **pwrData**: Array, dass die zu schreibenden Daten enthält
- **type**: "bool", "int" oder "string"

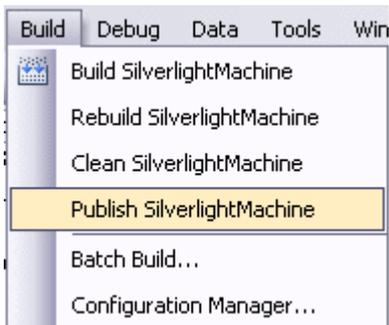
Wie auch bei der 'Load' Funktion muss auch hier nach Expression Blend 2 gewechselt werden, um in den entsprechenden Zeilen die beiden Methoden zum 'Klick-Event' ihrer beiden Buttons zu machen.

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

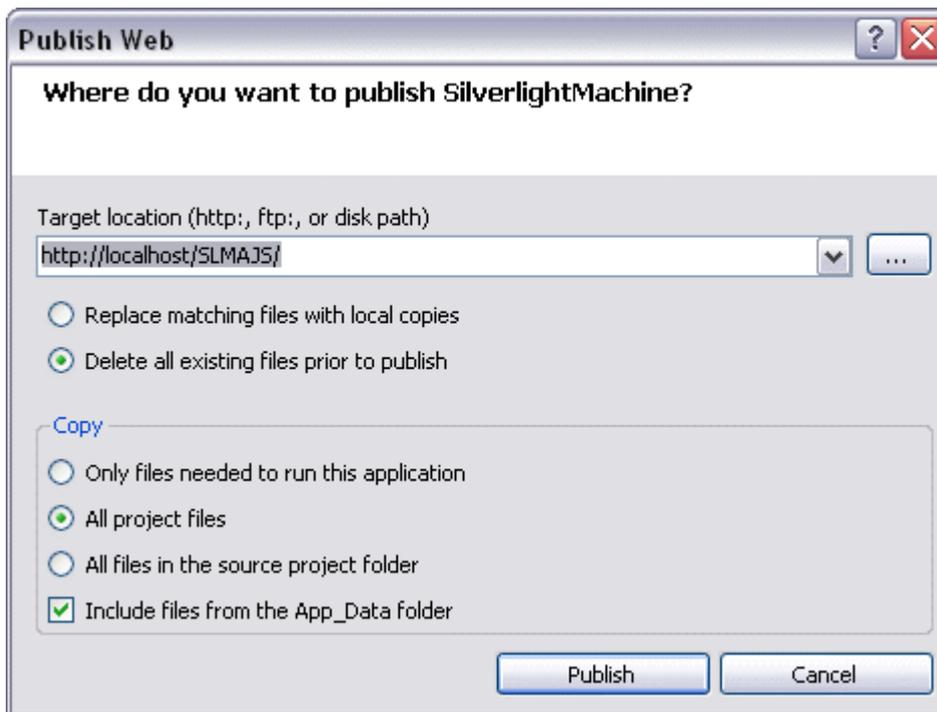
```
<Canvas x:Name="canvasSlow" MouseLeftButtonDown="Slow_MouseLeftButtonDown" ...
```

5. Testen:

Debuggen Sie zunächst Ihre Applikation. Sie werden feststellen, dass diese nicht so funktioniert wie Sie es erwarten. Anschließend gehen Sie über 'Build → Publish'.



Wählen Sie im Dialogfenster die 'Target location', sowie unter 'Copy' 'All project files' aus.



Wenn unten links in der Statusbar 'Publish succeeded' steht, können Sie Ihre Anwendung in einem Browser ausführen und testen.

Download:

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281698443.zip>

7.4.8 Beispiel Maschine mit Microsoft Silverlight for Windows Embedded

Eine Neuerung in Windows Embedded CE 6.0 R3 ist Silverlight for Windows Embedded. Mit dieser neuen Technologie können Bedienoberflächen von CE Geräten nun in XAML beschrieben werden und mit Tools wie Microsoft Expression Blend gestaltet werden. Anhand des Maschine Beispiels wird hier die Erstellung einer Silverlight for Windows Embedded Applikation mit Einbindung der ADS Komponente beschrieben.

Zielplattform

- Windows CE 6 R3

Implementierung

- C++

Erforderliche Software

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

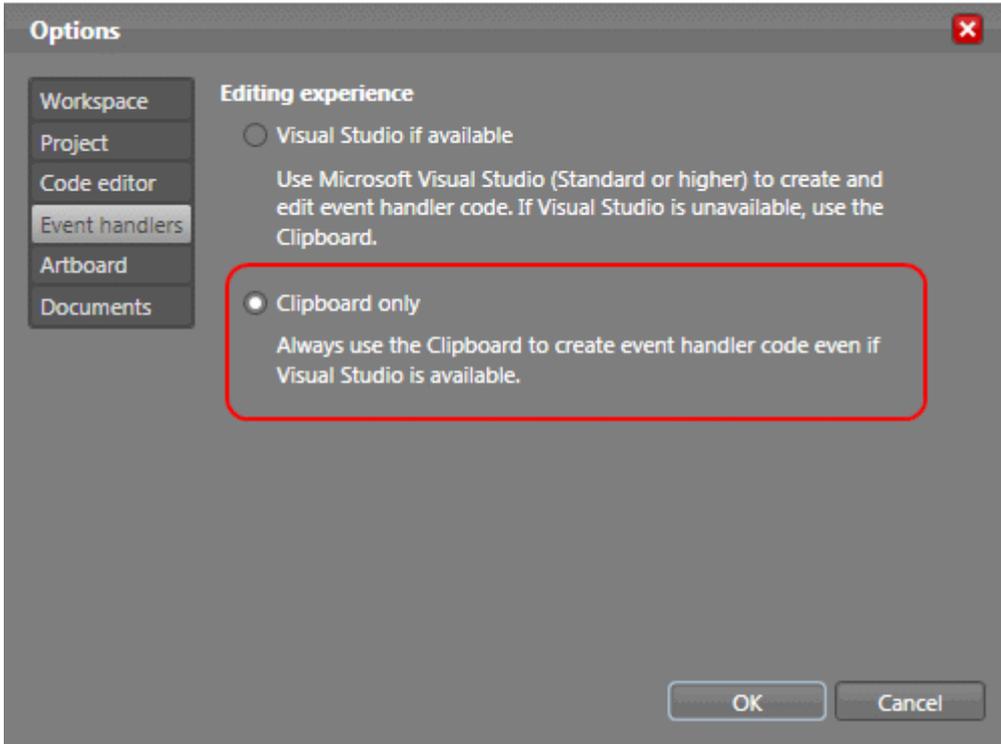
Erforderliche Hardware

- Windows CE 6.0 R3 Gerät (z.B. CX1020)

Die ersten Schritte...

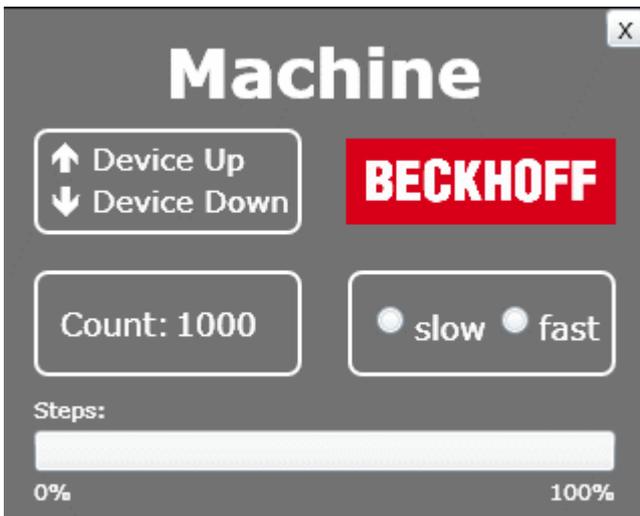
1. Neues Silverlight 2 Projekt erstellen:

Das Design einer Silverlight for Windows Embedded Applikation wird in XAML beschrieben. Dazu wird mit Microsoft Expression Blend 2 SP1 über 'File -> New Projekt' ein Silverlight 2 Projekt erstellt. Die dabei erstellte Visual Studio Solution wird in diesem Beispiel nicht benötigt. Zudem kann die Auswahl der Programmiersprache (Visual C# oder Visual Basic) außer Acht gelassen werden. Silverlight for Windows Embedded unterstützt nur Visual C++, welches nicht in Expression Blend integriert ist. Es ist daher auch nicht möglich den Quellcode zu verwenden, der von diesem Tool generiert wird. Deaktivieren Sie die Visual Studio Integration in Expression Blend um eine unnötige automatische Generation von Visual C# und Visual Basic Code zu vermeiden. Wählen Sie dazu: 'Options->Event handlers' 'Clipboard only'.



2. Bedienoberfläche erstellen

In Expression Blend kann nun die Bedienoberfläche erstellt werden.



Im oberen linken Bereich sind die beiden Ausgänge zu sehen, die auch auf den Busklemmen ausgegeben werden. Unten links ist die Variable abgebildet, welche die Werkstücke zählt. Rechts kann die Geschwindigkeit eingestellt werden. Die Anzeige 'Steps' entspricht der Anzahl der Takte. Oben rechts wurde zudem noch ein Button zum Beenden des Programms erstellt.

```
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="319" Height="255"><!-- Timelines --><UserControl.Resources><!-- Timeline Device Down --
><Storyboardx:Name="timelineDeviceDown"><ColorAnimationUsingKeyFramesBeginTime="00:00:00"
  Storyboard.TargetName="txtDeviceDown"
  Storyboard.TargetProperty="(TextBlock.Foreground) .
```

```

(SolidColorBrush.Color)"><SplineColorKeyFrameKeyTime="00:00:00.4000000" Value="#FFFF0000"/></
ColorAnimationUsingKeyFrames></Storyboard><!-- Timeline Device Up --
><Storyboardx:Name="timelineDeviceUp"><ColorAnimationUsingKeyFramesBeginTime="00:00:00"
Storyboard.TargetName="txtDeviceUp"
Storyboard.TargetProperty="(TextBlock.Foreground).
(SolidColorBrush.Color)"><SplineColorKeyFrameKeyTime="00:00:00.4000000" Value="#FFFF0000"/></
ColorAnimationUsingKeyFrames></Storyboard><!-- Timeline Engine --
><Storyboardx:Name="timelineEngine"/></UserControl.Resources><!-- Beginn der Layout Beschreibung --
><Gridx:Name="LayoutRoot" Background="#FF595959"><!-- Title Machine --
><TextBlockText="Machine" Margin="80,8.438,80,0" VerticalAlignment="Top"
FontWeight="Bold" Foreground="#FFFFFFFF" FontSize="34"/><!-- Device Up / Device Down --
><GridMargin="15,60,0,0" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="53" Width="132.532"><RectangleFill="{x:Null}" Stroke="#FFFFFFFF" StrokeThickness="2"

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

```

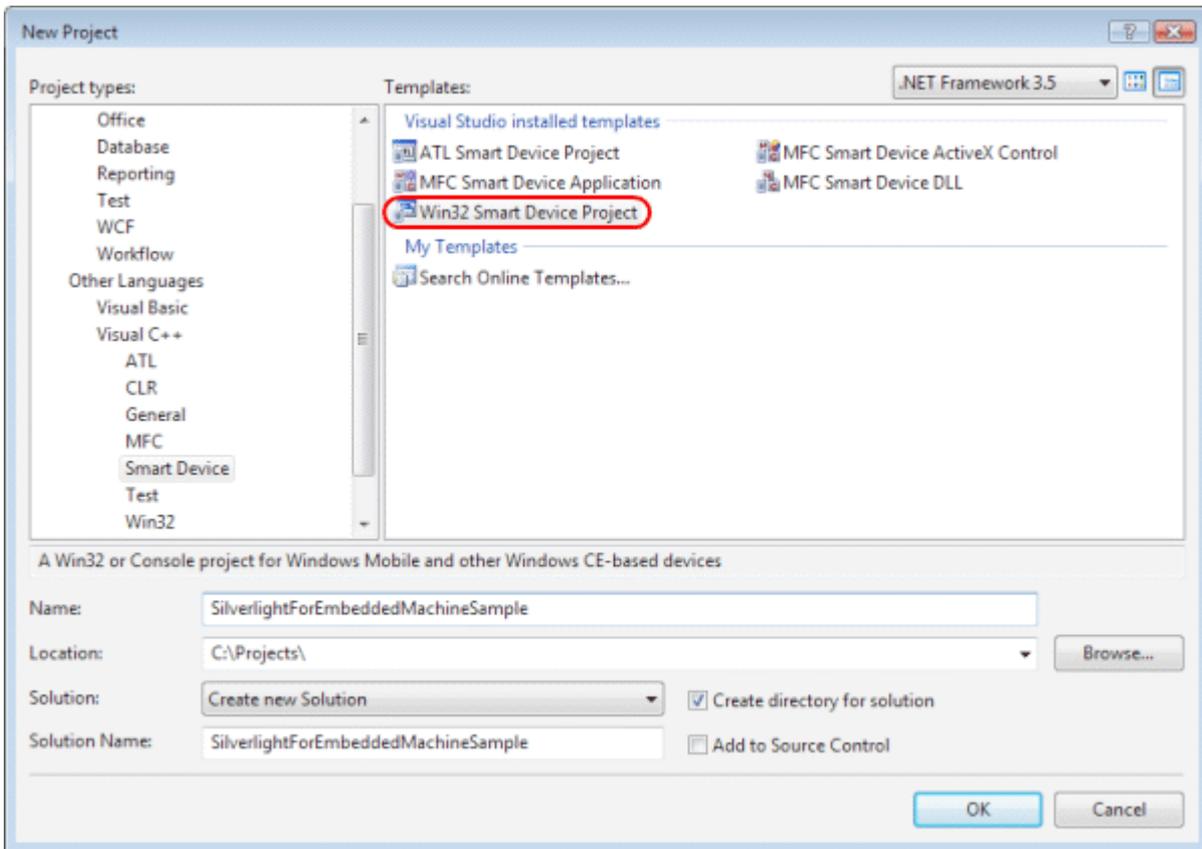
3. Neues Win32 Smart Device Projekt erstellen

In Visual Studio 2008 muss nun ein neues Win32 Smart Device Projekt erstellt werden.

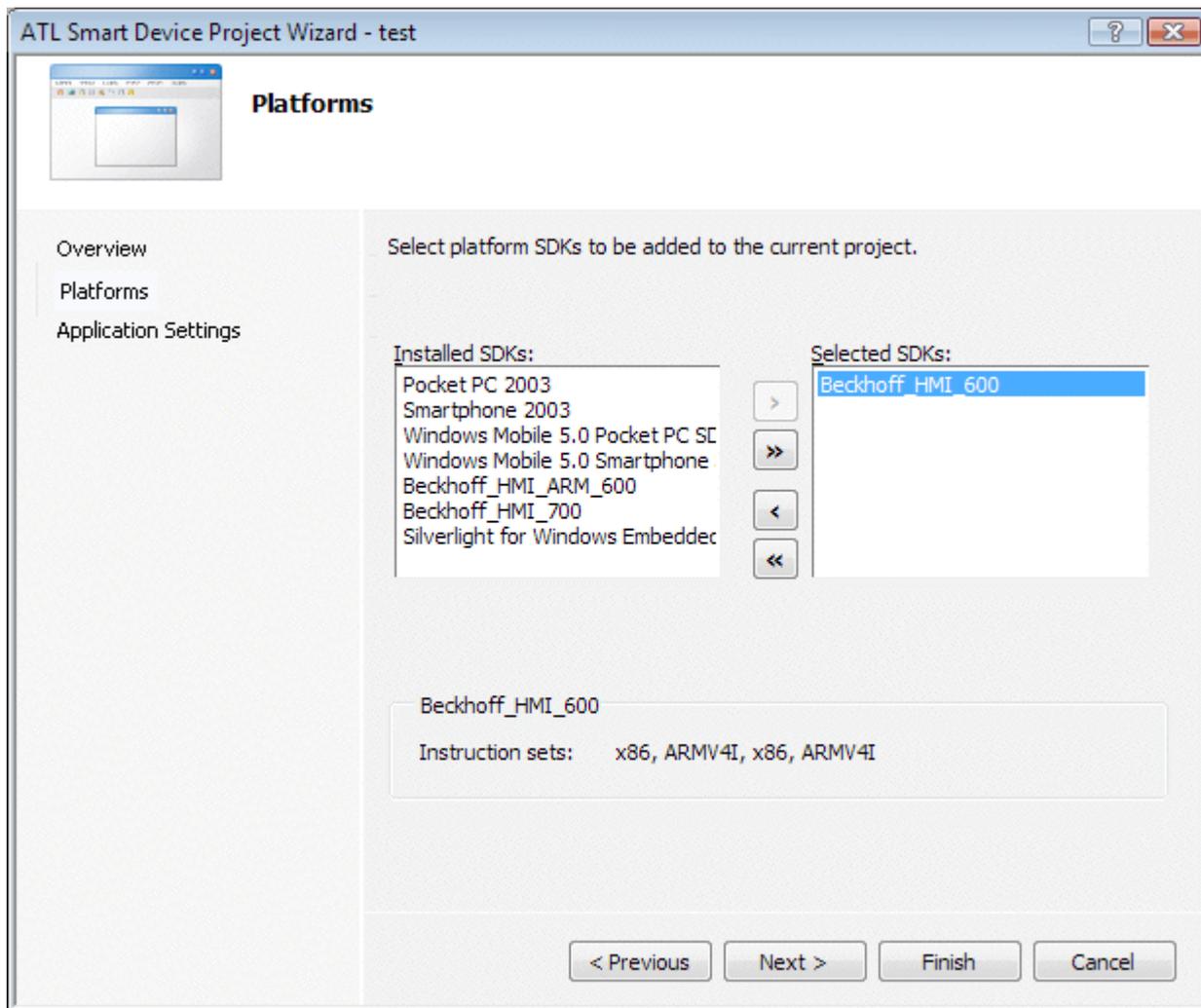


Beckhoff HMI 600 SDK installiert?

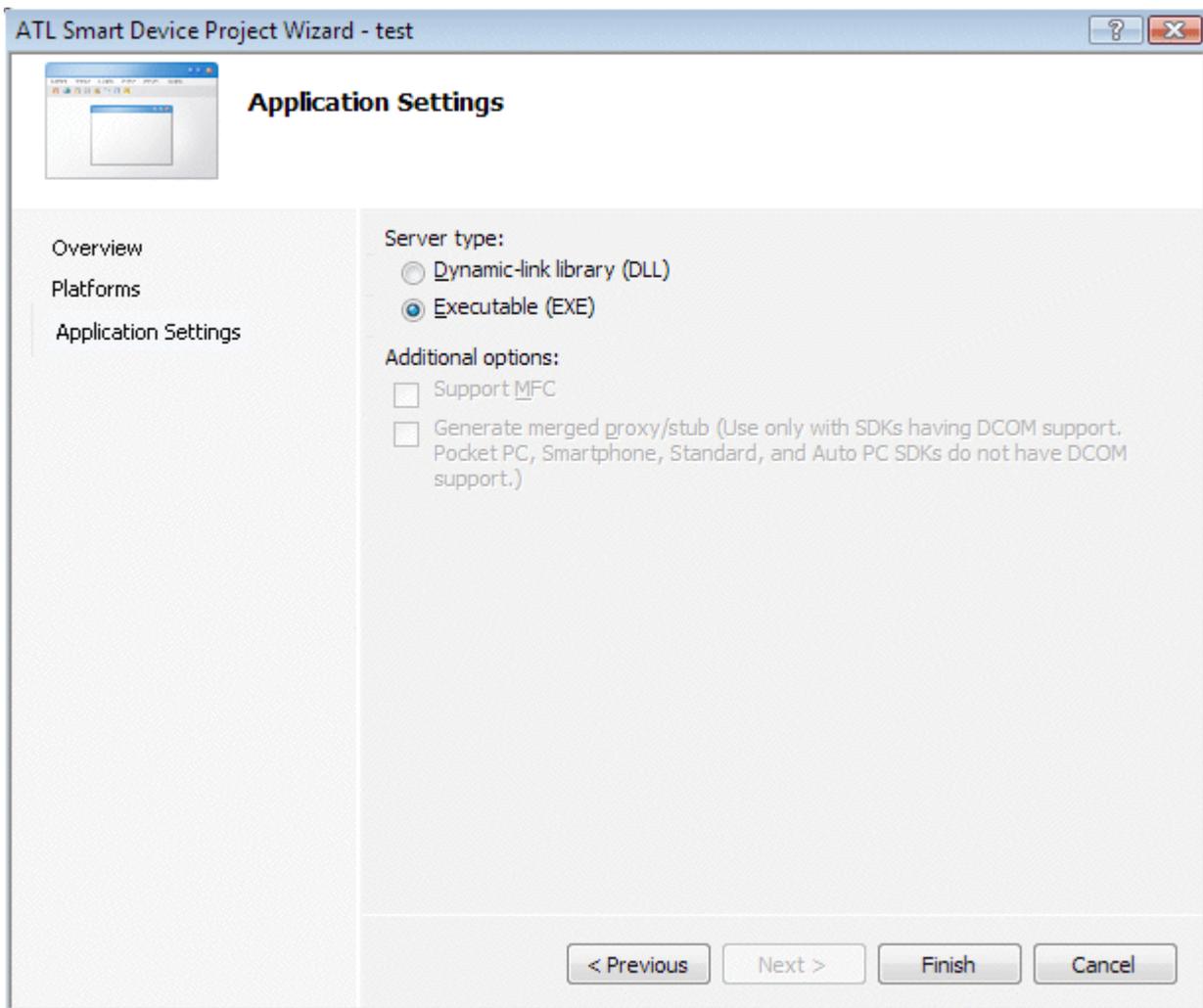
Sollte das Beckhoff HMI 600 SDK noch nicht auf dem Rechner installiert sein, so installieren Sie dies vor der Erstellung eines neuen Visual Studio Projektes.



Das Platform SDK dieses Projektes ist das Beckhoff HMI 600 SDK.

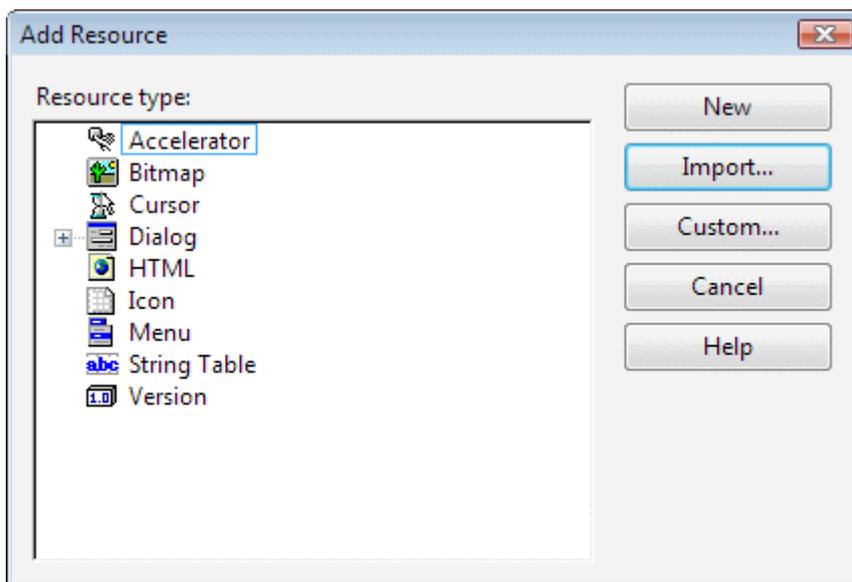


Als Servertyp wird Executable (EXE) ausgewählt.

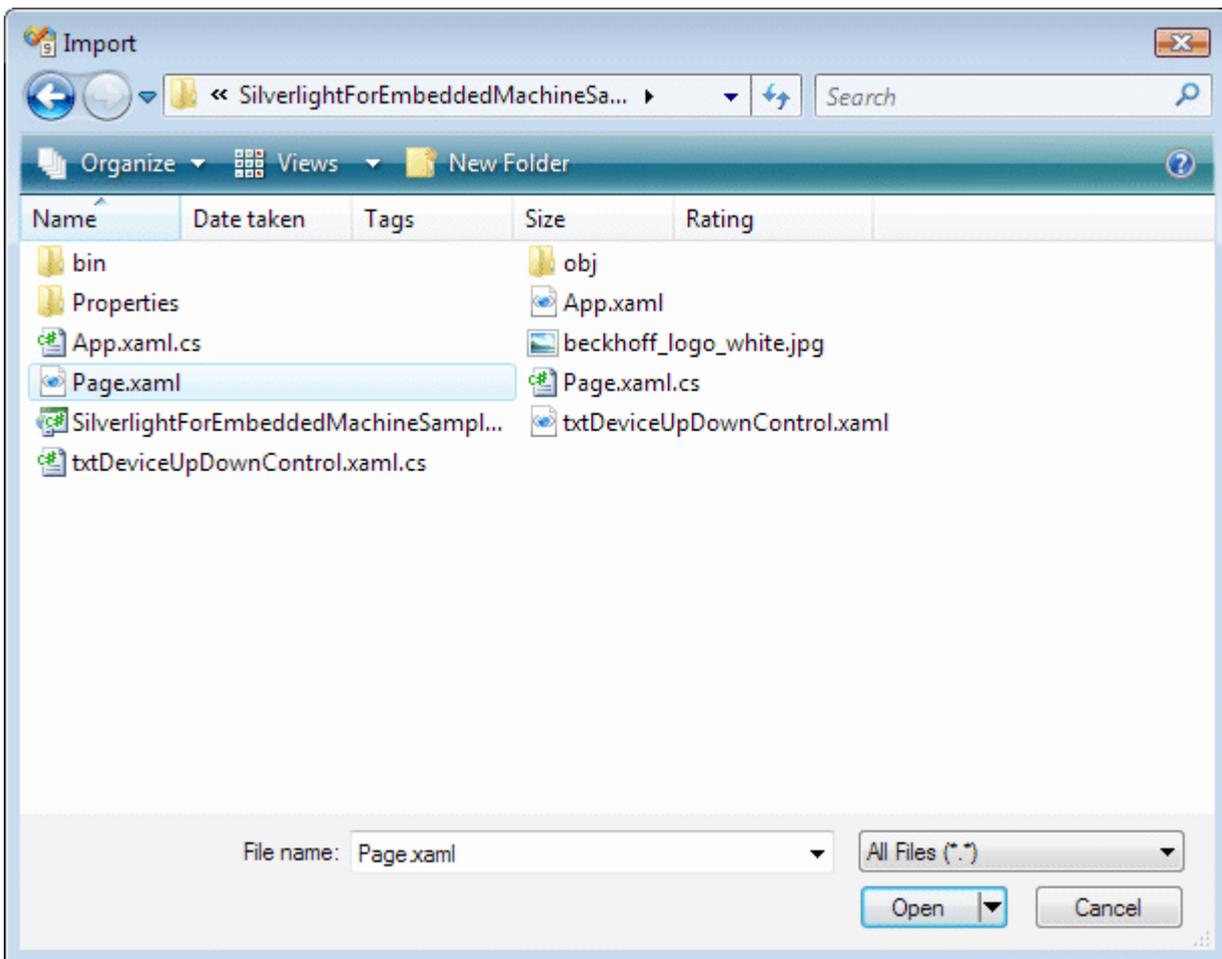


4. XAML-Datei als Resource einbinden

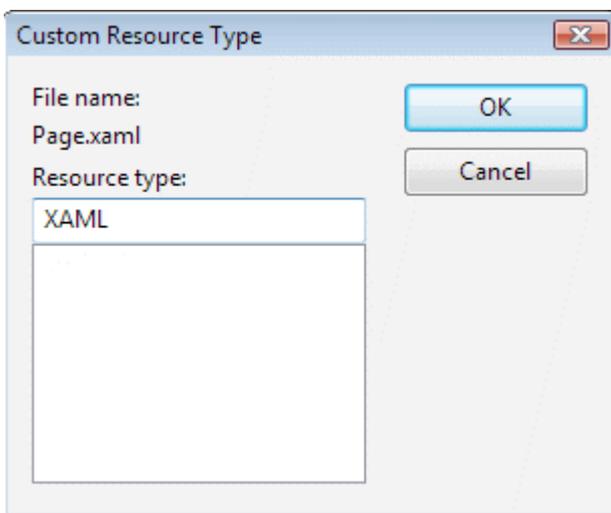
Die mit Expression Blend gestaltete Bedienoberfläche kann in das neue Projekt eingebunden werden. Öffnen Sie dazu die Resource-Datei (.rc). Mit einem Rechts-Klick auf die Resource im Resource View Tab und der Auswahl von 'Add -> Resource...' öffnet sich ein Dialog über den die XAML-Datei eingebunden werden kann.



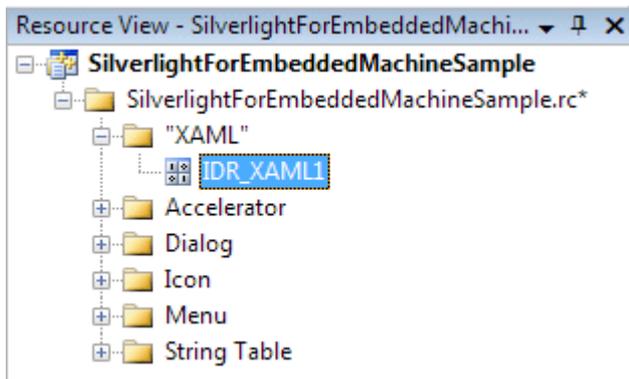
Mit Hilfe des Dialoges kann die XAML-Datei ins Projekt importiert werden.



XAML als Ressourcotyp angeben.



Die Standard ID (IDR_XAML1) der Resource kann in diesem Beispiel beibehalten werden. In eigenen Projekten ist es jedoch sinnvoll diese umzubenennen.



5. AdsHelper Class erstellen

Der größte Teil der Ads-Kommunikation kann in einer separaten Klasse, hier AdsHelper genannt, ausgelagert werden.

AdsHelper.h

In den AdsHelper-Header werden die TcAds-Headers eingebunden. Dabei ist auf die richtige Pfadangabe zu achten. Zudem sind die Headers abhängig vom Prozessortyp. Im Beispielcode sind die X86-Headers angefügt.

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

Des Weiteren werden im Header die Deklarationen vorgenommen.

```
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

Die Header StdAfx.h und AdsHelper.h müssen in die AdsHelper.cpp eingebunden werden,

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

und anschließend werden die globalen Variablen definiert.

```
long hEngine, hDeviceUp, hDeviceDown, hSteps, hCount, hSwitch;
unsigned long hEngineNotification, hDeviceUpNotification, hDeviceDownNotification,
    hStepsNotification, hCountNotification, hSwitchNotification;
```

Die Methoden AdsGetVarHandle und AdsGetVarHandleEx dienen dazu, Handles zu SPS-Variablen zu erstellen

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

    unsigned long read = 0;
    long nErr =
        AdsSyncReadWriteReqEx(pServerAddr, ADSIGRP_SYM_HNDBYNAME, 0x0,
```

```

    sizeof(long), pHandle, strlen(szVarname), (char*)szVarname, &read);

    return nErr;
}

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

    unsigned long read = 0;

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

    return nErr;
}

```

Mit `AdsFreeVarHandle` und `AdsFreeVarHandleEx` werden Handles zu SPS-Variablen wieder freigegeben.

```

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_RELEASEHND, 0,
        sizeof(pHandle), &pHandle);
}

```

Die folgenden beiden Methoden schreiben die SPS-Variablen `".switch"` und setzen dadurch die Geschwindigkeit auf langsam bzw. schnell.

```

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

    // Die SPS-Variablen ".switch" auf FALSE setzen
    bool datafalse = false;
    adsererror = AdsSyncWriteReqEx(port, pServerAddr, ADSIGRP_SYM_VALBYHND,
        handleSpeedSlow, 0x1, &datafalse);

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

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

    // Die SPS-Variablen ".switch" auf TRUE setzen
    bool datatrue = true;
    adsererror = AdsSyncWriteReqEx(port, pServerAddr, ADSIGRP_SYM_VALBYHND,
        handleSpeedFast, 0x1, &datatrue);

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

```

In der `connect`-Methode wird eine Verbindung zu den Variablen in der SPS erzeugt.

```

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 holen
    long 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 initialisieren
if(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;
}

```

Beim Trennen der Ads-Verbindung müssen die Handles der SPS-Variablen freigegeben und der Port geschlossen werden.

```

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. Quellcode bearbeiten

In der SilverlightForEmbeddedMachineSample.cpp-Datei werden als erstes die Headers eingebunden.

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

Anschließend folgt die Variablendeklaration.

```
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);
```

Die Timer-Callback-Funktion dient zur Überprüfung der Ads-Verbindung und stellt bei Bedarf die Verbindung wieder her.

```
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;
    }
}
```

Der folgende Ads-Event-Handler wird aufgerufen wenn sich eine SPS-Variable ändert, zu der eine Verknüpfung besteht.

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

Abhängig davon ob `deviceUp`, `deviceDown` oder `engine` auf `TRUE` gesetzt ist werden die entsprechenden Pfeile rot eingefärbt oder nicht.

Durch Verwendung von Timelines kann dieser Effekt noch verbessert werden.

```
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` gibt die Anzahl der Takte an. Der Wert wird über die Progressbar `prgSteps` ausgegeben. Hierzu müssen die Daten zunächst in ein Byte konvertiert werden, da die dazugehörige SPS Variable vom Typ Byte ist. Da der Progressbar nur Daten vom Typ float übergeben werden können erfolgt anschließend ein Konvertierung zum Typ float.

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

Wie auch schon bei Steps müssen bei `count` die Daten zunächst in ihren ursprünglichen Datentyp konvertiert werden, bevor sie in einen Text umgewandelt und dem Textblock `txtCount` übergeben werden können

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

Die Ausgabe des Geschwindigkeitstyps erfolgt über RadioButtons. Je nach Geschwindigkeit wird der entsprechende RadioButton markiert.

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

Der OnClick Event wird von den verschiedenen Instanzen angetriggert und über den Namen kann unterschieden werden, welche Instanz der Auslöser war.

```
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 der WinMain-Methode muss als erstes die XAML-Runtime initialisiert werden. Ist XamlRuntimeInitialize erfolgreich, so wird die Silverlight for Windows Embedded Runtime in der Applikation gestartet.

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

Jede Silverlight for Windows Embedded Applikation hat ein einzelnes "Applikations"-Objekt über welches der Zugriff auf globale Eigenschaften möglich ist.

Um auf dieses Objekt zuzugreifen wird die GetXRApplicationInstance API verwendet.

```
HRESULT retcode;

// Load an dinit XAML resource
IXRApplicationPtr app;
```

```

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

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

```

Nach der Initialisierung des Applikation Objekts kann das Hauptfenster erstellt und Silverlight for Windows Embedded die Verwaltung des Objektes übergeben werden.

```

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;

XR.XamlSource xamlsrc;

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

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

```

Das Objekt innerhalb einer Silverlight for Windows Embedded Applikation ist in Objektbäumen organisiert. Um auf dieses Objekt zuzugreifen wird ein Pointer zum Wurzelement benötigt.

```

IXRFrameworkElementPtr root;

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

```

Erzeugen von Instanzen der Oberflächenelemente und der Timelines.

```

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

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

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

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

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

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

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

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

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

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

```

Die Gruppe "RadioButtonGroup" erstellen und die beiden RadioButtons dieser Gruppe zuweisen.

```

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

```

Zum Verbinden des EventHandlers mit den Buttons wird ein weiterleitendes Objekt benötigt.

```
BtnEventHandler handler;

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

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

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

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

Einbinden der Ads-Komponenten.

```
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);

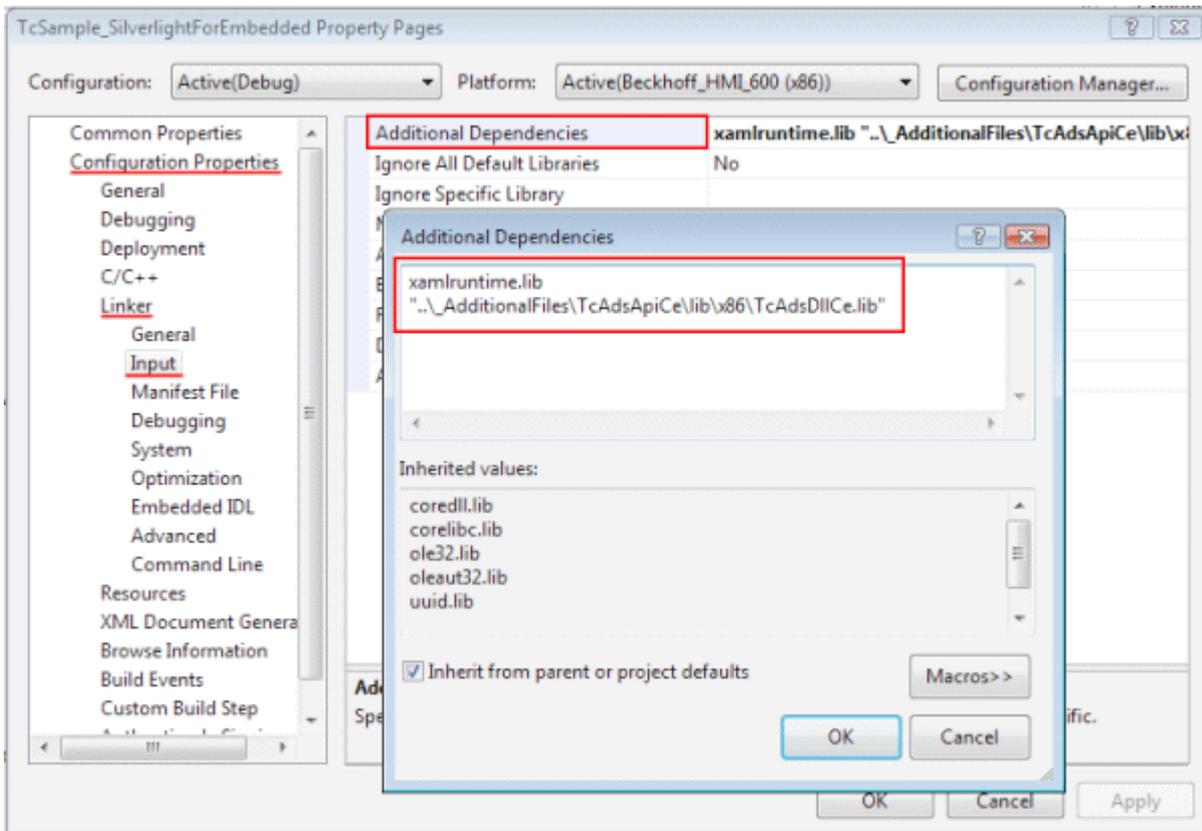
if (FAILED(retcode=vhost->StartDialoge(&exitcode)))
    return -1;

// cleanup
disconnect(port, &addr);
clickdelegate->Release();

return 0;
}
```

7. Eigenschaften

In den Projekteigenschaften muss eine Verbindung zur *xamlruntime.lib* und *TcAdsDllCe.lib* hergestellt werden.



Download Silverlight for Windows Embedded Beispiel

<https://infosys.beckhoff.com/content/1031/tcquickstart/Resources/5281699851.zip>

● X86 Version gegen ARM austauschen

i Im Sample wird die X86 Version der TcAdsDllCe.lib verwendet. Um das Sample für ARM-Geräte bauen zu können, muss diese Bibliothek vorher gegen die entsprechende ARM-Version ausgetauscht werden.

Mehr Informationen:
www.beckhoff.com/automation

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

