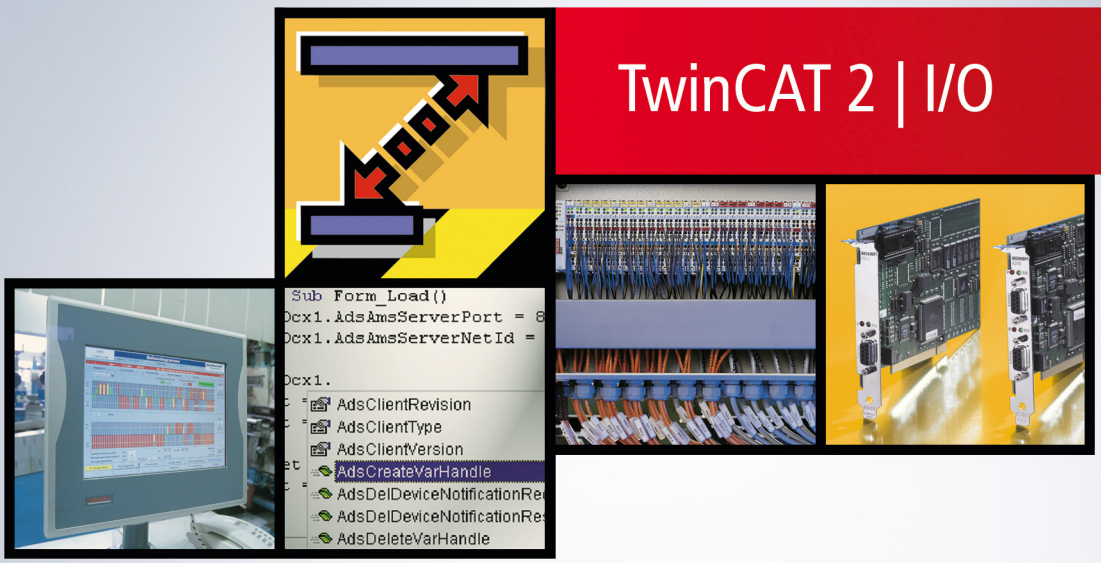
BECKHOFF New Automation Technology

Handbuch | DE

TX1100

TwinCAT 2 | TwinCAT I/O



28.09.2022 | Version: 1.1

Inhaltsverzeichnis

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, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche 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 weiter entwickelt.
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

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

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.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

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 Einleitung

Die grundlegende Idee hinter der TwinCAT I/O Schnittstelle ist der Zugriff der Windows NT/2000/XP/CE Anwendungsprogramme auf das TwinCAT I/O Subsystem, ohne die ADS Kommunikation zu verwenden.

Das Windows-Anwenderprogramm kann direkt das Prozessabbild bearbeiten, die Feldbusanbindung ist in unterlagerten Schichten gelöst.
Als Schnittstelle steht eine DLL Funktionen zur Verfügung um den Feldbus I/O zu triggern und einen Austausch der Prozessdaten mit dem TwinCAT Subsystem durchführen zu können.

Die Konfiguration erfolgt, wie mit TwinCAT üblich, im TwinCAT System Manager. Die Schnittstellen DLL ist für einen möglichst direkten Zugriff auf das Prozessabbild optimiert.

Mit Hilfe dieser Schnittstelle kann eine Windows NT Anwendungsapplikation alle Feldbus Typen, die für das TwinCAT System verfügbar sind (Lightbus, Profibus, Interbus, CANOpen, DeviceNet, Ethernet,…), verwenden.

Im letzten Kapitel dieser Dokumentation befindet sich ein Beispiel für die Verarbeitung von Feldbus I/Os.

3 Systemvoraussetzungen

Für Windows NT:

- TwinCAT 2.5 oder höher
- TCatIoDrv.dll
- TCatIoApi.h
- TCatIoDrv.lib

Für Windows CE:

- Image Version 1.90 oder höher.
- TwinCAT I/O datei : TCatIoW32Api.h, TCatIoDrvW32.lib
- TwinCAT Timer datei (optional) : TcTimerAPI.h, TcTimerWrap.lib
- TwinCAT CE "*Run as Device*" Option muss aktiviert sein.

Die für Windows CE https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082513675.zip .

4 TcTimer (CE) und TwinCAT IO

Zielplattform

Die TcTimer Funktionalität ist auf den CE basierenden Beckhoff Hardwaregeräten verfügbar (wie CX1000 / CX1020 / CX9000 / Ethernetpanels/ IPCs...)

https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082515083.pdf

Funktionaliät

Der TcTimer stellt einen deterministischen Timer dar, der die Ausführung z.B. eines kundenspezifischen C++ Codes erlaubt, anstatt die Logik innerhalb der IEC 61131-3 SPS Sprache zu implementieren.

Der Basis TcTimer ist ab 100 µs skalierbar und leitet sich vom höchsten CE Prioritätslevel ab. Das TwinCAT R3IO-API bietet einen Direktzugriff über Pointer auf das I/O Image einer TwinCAT IO Task.
Die erforderliche TwinCAT IO Task kann per Remote mit dem TwinCAT System Manager konfiguriert werden. Dieser erleichtert das Scannen der IOs auf dem verbundenen Feldbus und mappt die physikalischen IOs mit den logischen IOs der IO Task.

Datenkonsistenz: Nach dem Lesen der Eingangsdaten von der IO Task, Berechnung und Bereitstellung der neuen Daten im Ausgangsimage kann der C++ Code den Datenaustausch zwischen der IO Task und den gemappten physikalischen IOs veranlassen:

Als ein Ergebnis wird die Datenkonsistenz vom logischen C++ Code bis zum physikalischen IO Level unterstützt.

Hauptunterschiede zwischen TcTimer und TwinCAT IO R3

Generell sind die beiden Lösungen **TcTimer** und **R3IO** ähnlich: Beide Lösungen bieten mit API einen Zugriff aus dem C++ Code auf die Images der TwinCAT IO Task.

Die Hauptunterschiede:

- Die TcTimer Funktionalität wird **nur von der CE Plattform unterstützt** (CX1000 / CX1020 / CX9000 / Ethernetpanels / IPCs...)
- Die R3IO Funktionalität wird von den beiden Plattformen CE und XP/XPE unterstützt.
- TcTimer erlaubt die Ausführung von C++ Code in deterministischen Zyklen, außerdem kann der Datenaustausch zum Feldbus aus diesem deterministischen Zyklus angestoßen werden.
- R3IO erlaubt die Ausführung von C++ Code, der zyklisch durch einen Multimedia Timer (nicht hochdeterministisch) gestartet wird, um den Datenaustausch anzustoßen.
  Alternativ kann dieser Datenaustausch deterministisch durch die IO Task selbst gestartet werden : In diesem Fall sollte aus dem C++ Code per ADS in die IO-Task kommuniziert werden.

Notwendige Komponenten

Die für Windows CE https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082513675.zip .

Prioritäten

```
// Set the Priorities of the Thread
if (CeSetThreadPriority(hThreadTask1, 26) && CeSetThreadPriority(hThreadTask2, 27))
```

Im Beispiel wird die Thread-Priorität auf 26 und 27 gelegt.

Generell hat der Anwender die Freiheit, die Priorität des Threads zu bestimmen, hier Hinweise zu Prioritäten von anderen Prozessen:

- Device threads : Bereich 100-130- Beckhoff TwinCAT Timer thread : 1

- Microsoft MultiMedia Timer : 2

Abhängig von den Anforderungen kann somit eine Wahl der Priorität im Bereich 3 - 64 für Echtzeit threads und 131 - 255 für nicht Echtzeit threads sinnvoll erfolgen.

5 Funktionsreferenz

5.1 TCatIoOpen

Die Funktion *TCatIoOpen* stellt eine Verbindung zum TwinCAT I/O Server her. Bevor I/O Prozesse durchgeführt werden, sollte die Funktion *TCatIoOpen* aufgerufen werden.

```
LRESULT TCatIoOpen();
```

Parameter

Keine

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert -1.

Anmerkung:

TCatIoOpen ordnet Speicher und notwendige Systemresourcen zu.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoClose () [▶ 12]

Beachten Sie die Systemvoraussetzungen.

Sehen Sie dazu auch

Systemvoraussetzungen [▶ 9]

5.2 TCatIoClose

Die Funktion *TCatIoClose* beendet die Verbindung zum TwinCAT I/O Server. Bevor eine Anwendung beendet wird, sollte die Funktion *TCatIoClose* aufgerufen werden, um einen Verlust von Systemdaten zu verhindern.

```
LRESULT TCatIoClose();
```

Parameter

Keine

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert Null.

Anmerkung

TCatIoClose bereinigt und gibt zugeordnete Speicher und Systemresourcen frei.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoOpen () [▶ 12]

5.3 TCatIoInputUpdate

Die Funktion *TCatIoInputUpdate* initialisiert die Eingabe Zuordnungen der spezifizierten Task..

```
LRESULT TCatIoInputUpdate(
  USHORT nPort
);
```

Parameter

nPort

Port ID des I/O Tasks, dessen Prozessabbild zum Transfer der Eingabedaten benutzt werden sollte. Für weitere Informationen bezüglich der Definition der Prozessabbilder einer Task, siehe TwinCAT System Manager - Benutzerdefinierte Tasks.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null. Mögliche Fehlercodes:

-1: I/O Funktion ist nicht initialisiert

IOERR_IOSTATEBUSY [0x2075]: I/O Gerät ist nicht bereit.

Anmerkung

TCatIoInputUpdate prüft den Status des Eingabe Geräts, ob es bereit ist oder nicht. Durch den Aufruf von *TCatIoInputUpdate* wird kein Feldbus I/O gestartet. Um einen Feldbus I/O Zyklus zu starten. muss die Funktion TCatIoOutputUpdate [▶ 14] aufgerufen werden.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoOutputUpdate () [▶ 14]

5.4 TCatIoOutputUpdate

Die Funktion *TCatIoOutputUpdate* initialisiert die Ausgabe Zuordnungen der spezifizierten Task..

```
LRESULT TCatIoOutputUpdate(
  USHORT nPort
);
```

Parameter

nPort

Port ID des I/O Tasks, dessen Prozessabbild zum Transfer der Ausgabedaten benutzt werden sollte. Für weitere Informationen bezüglich der Definition der Prozessabbilder einer Task, siehe TwinCAT System Manager - Benutzerdefinierte Tasks.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null. Mögliche Fehlercodes:

-1: I/O Funktion ist nicht initialisiert

IOERR_IOSTATEBUSY [0x2075]: I/O Gerät ist nicht bereit.

Anmerkung

TCatIoOutputUpdate prüft den Status des I/O Geräts, ob es bereit ist oder nicht. Wenn das I/O Gerät bereit ist, schreibt *TCatIoOutputUpdate* die Ausgabedaten zum Gerät und startet den Feldbus I/O Zyklus.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.

- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoInputUpdate() [▶ 13]

5.5 TCatIoGetOutputPtr

Die Funktion *TCatIoGetOutputPtr* ordnet dem ausgehenden Prozessabbild ein Ausgabepuffer (buffer) zu.

```
LRESULT TCatIoGetOutputPtr (
  USHORT nPort,
  VOID** ppOutp,
  int nSize
);
```

Parameter

nPort

Port ID des I/O Tasks, dessen Prozessabbild zum Transfer der Ausgabedaten benutzt werden sollte. Für weitere Informationen bezüglich der Definition der Prozessabbilder einer Task, siehe TwinCAT System Manager - Benutzerdefinierte Tasks.

ppOutp

Pointeradresse um die Adresse des Ausgabepuffers zu bekommen. Wenn *TCatIoGetOutputPtr* folgt, hat der Pointer die Adresse des Ausgangsdatenpuffers initialisiert.

nSize

Byteanzahl des angeforderten Prozessabbildpuffers.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null.

Anmerkung

Die Funktion *TCatIoGetOutputPtr* ordnet dem Prozessabbild der spezifizierten Task einen Arbeitsspeicher zu und gibt die Adresse an *ppOutp* zurück. Wenn der Speicher bereits zugeordnet war, gibt *TCatIoGetOutputPtr* die Adresse des vorher zugeordneten Puffers zurück. Die Ausgabedaten werden durch diesen Puffer übertragen. Wenn TwinCAT gestoppt oder neu gestartet wird während der benutzerdefinierte Prozess läuft, bleibt die Ausgabe Adresse gültig, obwohl der I/O Transfer temporär gestoppt wird. Im Falle eines TwinCAT Neu Starts (Restart) kann der benutzerdefinierte Prozess die Ausführung ohne eine extra Berechnung wieder aufnehmen.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.

- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoGetInputPtr () [▶ 16]

5.6 TCatIoGetInputPtr

Die Funktion *TCatIoGetInputPtr* ordnet dem eingehenden Prozessabbild ein Eingangspuffer zu.

```
LRESULT TCatIoGetInputPtr (
  USHORT nPort,
  VOID** ppInp,
  int nSize
);
```

Parameter

nPort

Port ID des I/O Tasks, dessen Prozessabbild zum Transfer der Ausgabedaten benutzt werden sollte. Für weitere Informationen bezüglich der Definition der Prozessabbilder einer Task, siehe TwinCAT System Manager - Benutzerdefinierte Tasks.

ppInp

Pointeradresse um die Adresse des Ausgabepuffers zu bekommen. Wenn *TCatIoGetInputPtr* folgt, hat der Pointer die Adresse des Eingangsdatenpuffers initialisiert.

nSize

Byteanzahl des angeforderten Prozessabbildpuffers.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null.

Anmerkung

Die Funktion *TCatIoGetInputPtr* ordnet dem Prozessabbild der spezifizierten Task einen Arbeitsspeicher zu und gibt die Adresse an *ppInp* zurück. Wenn der Speicher bereits zugeordnet war, gibt *TCatIoGetInputPtr* die Adresse des vorher zugeordneten Puffers zurück. Die Eingabedaten werden durch diesen Puffer übertragen. Wenn TwinCAT gestoppt oder neu gestartet wird während der benutzerdefinierte Prozess läuft, bleibt die Ausgabe Adresse gültig, obwohl der I/O Transfer temporär gestoppt wird. Im Falle eines TwinCAT Neu Starts (Restart) kann der benutzerdefinierte Prozess die Ausführung ohne eine extra Berechnung wieder aufnehmen.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoGetOutputPtr () [▶ 15]

5.7 TCatIoReset

Die Funktion *TCatIoReset* setzt alle verfügbaren I/O Geräte im TwinCAT System zurück.

```
LRESULT TCatIoReset();
```

Parameter

Keine

Rückgabewert

- Win32: Wenn die Funktion fehlschlägt, ist der Rückgabewert ungleich Null.
- Win CE: Bei Erfolg True (1) , FALSE (0) im Fehlerfall.

Anmerkung

Die Funktion *TCatIoReset* initiiert das Rücksetzen der I/O Geräte durch die Anforderung des *AdsWriteControl.* Folglich kann TCatIoReset nur folgen, wenn das TwinCAT System läuft.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.

Benutzte Import Bibliothek: TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

5.8 TCatIoGetCpuTime

Die Funktion *TCatIoGetCpuTime* gibt die aktuelle CPU Zeit an.

```
LRESULT TCatIoGetCpuTime( __int64* pnTime );
```

Parameter

pnTime

Adresse einer 64 Bit Integer Variable zum Empfang der aktuellen CPU Zeit in Einheiten von 100Ns.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null.

Anmerkung

Die Funktion *TCatIoGetCpuTime* liest den Zählerstand der Pentuim CPU und skaliert ihn in Einheiten von 100Ns.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoGetCpuCounter () [▶ 18]

5.9 TCatIoGetCpuCounter

Die Funktion *TCatIoGetCpuCounter* gibt den aktuellen CPU Zählerstand an.

```
LRESULT TCatIoGetCpuCounter( __int64* pnCount );
```

Parameter

pnCount

Adresse einer 64 Bit Integer Variable zum Empfang des aktuellen CPU Zählerstands.

Rückgabewert

Wenn die Funktion ausfällt, ist der Rückgabewert nicht Null.

Anmerkung

Die Funktion *TCatIoGetCpuCounter* liest den Zählerstrand der Pentium CPU.

QuickInfo

Für Windows NT:

- **Windows NT:** Erforderliche Version 4.0 oder höher.
- **TwinCAT:** Erforderliche Version 2.5 oder höher.
- **Header:** Erklärt in TCatIoApi.h.
- **Benutzte Import Bibliothek:** TCatIoDrv.lib.

Für Windows CE:

- **Windows CE Version:** Erforderliche Version 4.2 oder höher.

- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

Siehe auch

TCatIoGetCpuTime () [▶ 17]

6 Function Reference (OS CE only)

6.1 TcTimerInitialize

The *TcTimerInitialize* function Initializes the TwinCAT Timer. Before using any TcTimer functionality *TcTimerInitialize* must be called.

```
DWORD TcTimerInitialize();
```

Parameters

None

Return Values

STATUS_SUCCESS - Indicates function succeeded in initializing TcTimer module.

STATUS_UNSUCCESSFUL - Indicates function failed in initializing TcTimer module.

Remarks

None

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h
- **Import Library:** Use TcTimerWrap.lib

6.2 TcTimerDeinitialize

The *TcTimerDeinitialize* function deinitializes the TwinCAT Timer.

```
DWORD TcTimerDeinitialize();
```

Parameters

None

Return Values

STATUS_SUCCESS - Indicates function succeeded.

STATUS_UNSUCCESSFUL - Indicates function failed.

Remarks

None

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h

- **Import Library:** Use TcTimerWrap.lib

6.3 TcTimerSetEvent

This function starts a specified timer event. The TwinCAT Timer runs in its own thread. After the event is activated, it sets or pulses the specified event object.

```
DWORD TcTimerSetEvent(
  UINT uDelay,
  LPTSTR lpEventName,
  UINT fuEvent
);
```

Parameters

uDelay

Event delay, in TcTimer Ticks.

lpEventName

Pointer to a null-terminated string that specifies the name of the event object. The name is limited to MAX_PATH characters and can contain any character except the backslash path-separator character (\). Name comparison is case sensitive. (Note: Event Names starting with "*TC_* "are reserved for Beckhoff)

fuEvent

Timer Event Type. Following Table shows the values that can be included by the fuEvent parameter.

Value	Description
TIME_ONESHOT	Event occurs once after uDelay ticks
TIME_PERIODIC	Event occurs after every uDelay ticks
TIME_CALLBACK_EVENT_SET	When the timer expires, the system calls the SetEvent() function to set the event with lpEventName.
TIME_CALLBACK_EVENT_PULSE	When the timer expires, the system calls the PulseEvent() function to pulse the event with lpEventName.

Return Values

Returns an identifier for the timer event if successful. This function returns NULL if it fails and the timer event was not created.

Remarks

Each call to **TcTimerSetEvent** for periodic timer events requires a corresponding call to the **TcTimerKillEvent** function.

The **TcTimer Ticks** are always configured by the "**TwinCAT System Service**". This can be done by configuring the "**Base Ticks**" in the Real Time Settings from the "**TwinCAT System Manager**" and activating the configuration on the Target System.

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h
- **Import Library:** Use TcTimerWrap.lib

6.4 TcTimerKillEvent

This function Destroys the specified timer event.

```
DWORD TcTimerKillEvent(
  UINT uTimerId
);
```

Parameters

uTimerId

Identifier of the timer event to cancel. This identifier was returned by the TcTimerSetEvnet function when the timer event was set up.

Return Values

STATUS_SUCCESS - Indicates function Succeeded

STATUS_UNSUCCESSFUL - Indicates function failed.

Remarks

None.

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h
- **Import Library:** Use TcTimerWrap.lib

6.5 TcTimerGetTickCount

The *TcTimerGetTickCount* function returns the Current Tick Count of the TwinCAT Timer.

```
DWORD TcTimerGetTickCount();
```

Parameters

None

Return Values

Tick count of the TwinCAT Timer.

Remarks

The **TcTimer Ticks** are always configured by the "**TwinCAT System Service**". This can be done by configuring the "**Base Ticks**" in the Real Time Settings from the "**TwinCAT System Manager**" and activating the configuration on the Target System.

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h

- **Import Library:** Use TcTimerWrap.lib

6.6 TcTimerGetTickTime

The *TcTimerGetTickTime* function returns the ticking time of the TwinCAT Timer.

```
DWORD TcTimerGetTickTime();
```

Parameters

None

Return Values

Tick time of the TwinCAT Timer in units of 100 nanoseconds. (i.e. 500µs Base tick returns 5000)

Remarks

The **TcTimer Ticks** are always configured by the "**TwinCAT System Service**". This can be done by configuring the "**Base Ticks**" in the Real Time Settings from the "**TwinCAT System Manager**" and activating the configuration on the Target System.

QuickInfo

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **CE Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TcTimerAPI.h
- **Import Library:** Use TcTimerWrap.lib

6.7 TCatGetState

Die Funktion TCatGetState gibt den aktuellen ADS - State von TwinCAT System Service.

```
LRESULT TCatGetState();
```

Return Values

Aktuellen AdsState von TwinCAT System Service. Wenn die Funktion ausfällt, ist der Rückgabewert ADSSTATE_ERROR.

QuickInfo

Für Windows CE:

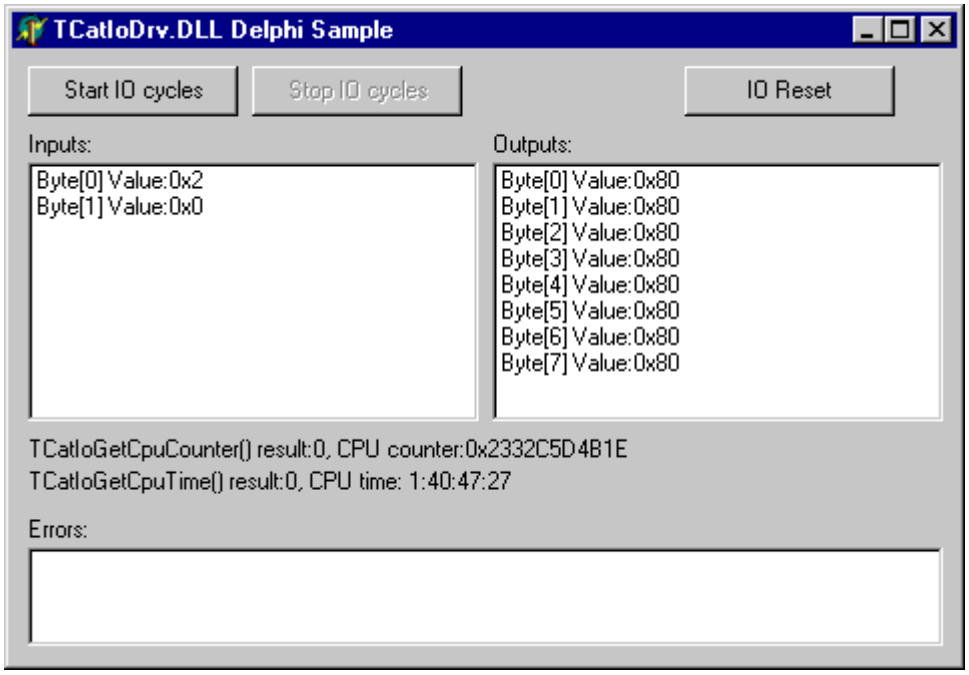
- **Windows CE Version:** Erforderliche Version 4.2 oder höher.
- **Image Version:** Erforderliche version 1.90 oder höher.
- **Header:** Erklärt in TCatIoW32Api.h.
- **Benutzte Import Bibliothek:** TCatIoDrvW32.lib.

7 Beispiele

7.1 TwinCAT I/0 Ring 3 DLL: Delphi-Applikation

Für dieses Beispiel wurden die Funktionen der TCatIoDrv-DLL nach Pascal portiert und in einer Unit **TCatIoDrv.pas** zusammengefasst.

Die Sourcen zu diesem Beispiel können sie hier entpacken: https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082516491.exe



Systemvoraussetzungen:

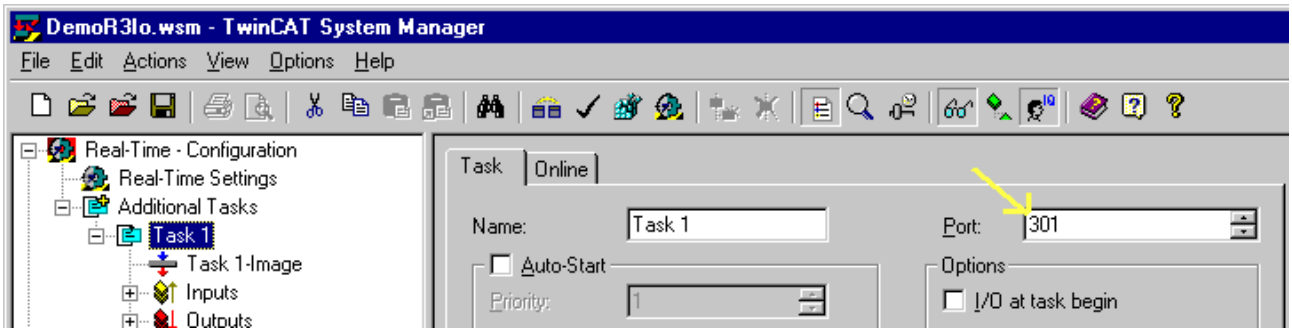
- TwinCAT 2.5 oder höher
- TCatIoDrv.Dll
- Delphi 5.0

Beschreibung

Aus der Delphi-Applikation soll das Mapping von dem Prozessabbild der Ein-/Ausgänge einer zusätzlichen Task im TwinCAT System Manager getriggert werden. Die Zykluszeit beträgt 100ms und wird mit Hilfe eines Multimedia-Timers erzeugt. Die Onlinewerte der Prozessabbilder werden in zwei ListBoxen angezeigt. Über den *IO Reset*-Button kann ein Reset der TwinCAT IO-Geräte durchgeführt werden. Das Mapping der Eingänge und Ausgänge kann über die Buttons *Start IO cycle* und *Stop IO cycle* gestartet bzw. gestoppt werden. Eventuelle Fehlermeldungen werden in einer weiteren ListBox ausgegeben.

Die Task-Konfiguration in TwinCAT System Manager

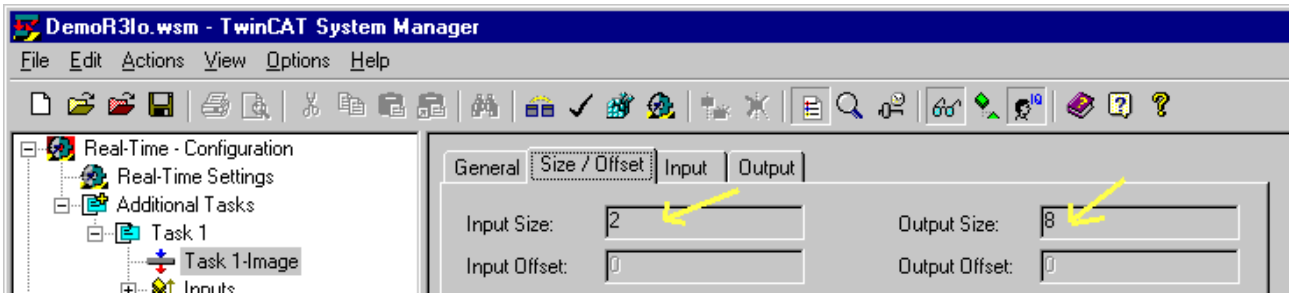
Für die zusätzliche Task wurde im TwinCAT System Manager die Portnummer 301 konfiguriert.



Das Prozessabbild hat folgende Größe:

Eingangsabbild: 2 Byte;

Ausgangsabbild: 8 Byte;



Die Portnummer und die Bytegröße des Prozessabbildes wurden in der **TCatIoDrv.pas**-Unit als Konstanten definiert und müssen bei anderen Werten entsprechend geändert werden.

Einbinden der TCatIoDrv-Unit

Die Deklarationen der benutzten TCatIoDrv.DLL-Funktionen befinden sich in der Pascal-Unit **TCatIoDrv.pas**. Diese wurde über eine Uses-Klausel in das Projekt eingebunden.

```
unit TCatIoDrvDelphiUnit;

interface
uses
TCatIoDrv,
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
StdCtrls, ExtCtrls, Grids;

...
```

Unit TCatIoDrv.pas

```
unit TCatIoDrv;

interface
uses sysutils,windows;
{$A-}
const
    // TwinCAT® System Manager->Additional Tasks->Task 1 tab:
    TASK_1_PORTNUMBER       = 301;
    // TwinCAT® System Manager->Additional Tasks->Task 1-Image->Size/Offset tab:
    MAX_INPUT_IMAGE_BYTESIZE    = 2;
    MAX_OUTPUT_IMAGE_BYTESIZE   = 8;
type
    TInputImage     = ARRAY[ 0 .. MAX_INPUT_IMAGE_BYTESIZE - 1 ] OF Byte;
    TOutputImage    = ARRAY[ 0 .. MAX_OUTPUT_IMAGE_BYTESIZE - 1 ] OF Byte;
    PInputImage     = ^TInputImage;
    POutputImage    = ^TOutputImage;
    function TCatIoOpen:longint; stdcall; external 'TCatIoDrv.dll' name '_TCatIoOpen@0';
    function TCatIoClose:longint;stdcall; external 'TCatIoDrv.dll' name '_TCatIoClose@0';
    function TCatIoInputUpdate( nPort : WORD ):longint;  stdcall; external 'TCatIoDrv.dll' name '_TC
atIoInputUpdate@4';
    function TCatIoOutputUpdate( nPort : WORD ):longint;  stdcall; external 'TCatIoDrv.dll' name  '_
TCatIoOutputUpdate@4';
    function TCatIoGetInputPtr( nPort : WORD; var pInput :Pointer; nSize :longint ):longint;   stdca
```

```
ll; external 'TCatIoDrv.dll' name  '_TCatIoGetInputPtr@12';
    function TCatIoGetOutputPtr( nPort : WORD; var pOutput :Pointer; nSize :longint ):longint;  stdc
all; external 'TCatIoDrv.dll' name  '_TCatIoGetOutputPtr@12';
    function TCatIoReset():longint;  stdcall; external 'TCatIoDrv.dll' name   '_TCatIoReset@0';
    function TCatIoGetCpuTime( var  pCpuTime : TFileTime ):longint;  stdcall; external 'TCatIoDrv.dl
l' name  '_TCatIoGetCpuTime@4';
    function TCatIoGetCpuCounter( var pCpuCount : int64 ):longint;  stdcall; external 'TCatIoDrv.dll
' name  '_TCatIoGetCpuCounter@4';

implementation
initialization
finalization
end.
```

Die Applikation

In der Event-Funktion *FormCreate* wird die DLL-Funktion TCatIoOpen [▶ 12] aufgerufen. Beim Erfolg liefert diese Funktion eine Null und es wird eine Verbindung zum TwinCAT I/O Subsystem aufgebaut. Ein Fehler wird in der ListBox an den Benutzer ausgegeben. Beim Beenden der Anwendung muß die Verbindung zum TwinCAT I/O Subsystem abgebaut werden. Dies geschieht in der Event-Funktion *FormDestroy,* dabei wird die DLL-Funktion TCatIotClose [▶ 12] aufgerufen. Wurde die Verbindung erfolgreich aufgebaut, dann werden zwei weitere Funktionen aufgerufen: TCatIoGetInputPtr [▶ 16] und TCatIoGetoutputPtr [▶ 15]. Diese Funktionen liefern Zeiger auf die Prozessabbilder der Ein-/Ausgänge. Über diese Zeiger können die Prozessdaten der Eingänge gelesen oder der Ausgänge geschrieben werden.

```
unit TCatIoDrvDelphiUnit;
interface
usesTCatIoDrv,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Grids;
type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Timer1: TTimer;
    StartButton: TButton;
    StopButton: TButton;
    Label1: TLabel;
    Button1: TButton;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    ListBox2: TListBox;
    ListBox3: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure StartButtonClick(Sender: TObject);
    procedure StopButtonClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    procedure InitControls();
    procedure CalculateNewOutputs(pData   : Pointer; cbSize   :integer);
    procedure ViewData( var ListBox : TListBox; pData   : Pointer; cbSize   :integer);
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  InputImagePtr      :PInputImage;
  OutputImagePtr     :POutputImage;

implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
var Result  : integer;
    InPtr, OutPtr :Pointer;
begin
    InitControls();

    Result := TCatIoOpen();
    if ( Result <> 0 ) then
    ListBox1.Items.Insert(0, Format('TCatIoOpen() error:%d', [ Result ] ) )
    else
    begin
    {get/initialize pointer to the input image}
```

```
    InPtr := NIL;
    Result := TCatIoGetInputPtr( TASK_1_PORTNUMBER, InPtr, MAX_INPUT_IMAGE_BYTESIZE );
    if (  Result = 0 ) And ( InPtr <> NIL ) then
        InputImagePtr := InPtr
    else
        ListBox1.Items.Insert(0, Format('TCatIoGetInputPtr() error:%d', [ Result ] ) );

    {get/initialize pointer to the output image}
    OutPtr := NIL;
    Result := TCatIoGetOutputPtr( TASK_1_PORTNUMBER, OutPtr, MAX_OUTPUT_IMAGE_BYTESIZE );
    if (  Result = 0 ) And ( OutPtr <> NIL ) Then
        OutputImagePtr := OutPtr
    else
        ListBox1.Items.Insert(0, Format('TCatIoGetOutputPtr() error:%d', [ Result ] ) );
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var Result : integer;
begin
    Result := TCatIoClose();
    if ( Result <> 0 ) then
    MessageBox(0, 'TCatIoClose() error!', 'Error', MB_OK);
end;
```

Wurde der Multimedia-Timer aktiviert, dann wird zyklisch die *Timer1Timer*-Ereignisroutine aufgerufen. In dieser Routine werden jedes mal folgende Funktionen aufgerufen:

- TCatIoInputUpdate [▶ 13] (DLL-Funktion: Triggert die Aktualisierung des Prozessabbildes der Eingänge (Eingänge werden gelesen));
- ViewData (Hilfsprozedur: Zeigt die aktuellen Werte der Eingänge in einer ListBox an);
- CalculateNewOutputs (Hilfsprozedur: Generiert/Verändert die Werte der Ausgänge (z.B. Lauflicht));
- TCatIoOutputUpdate [▶ 14] (DLL-Funktion: Triggert die Aktualisierung des Prozessabbildes der Ausgänge (Ausgänge werden geschrieben));
- ViewData (Hilfsprozedur: Zeigt die aktuellen Werte der Ausgänge in einer ListBox an);
- TCatIoGetCpuCounter [▶ 18] (DLL-Funktion: Liest den aktuellen Wert des CPU-Zählers);
- TCatIoGetCpuTime [▶ 17] (DLL-Funktion: Liest den Zählerstand der Pentium CPU);

```
procedure TForm1.Timer1Timer(Sender: TObject);
var Result : integer;
    CpuCounter  :int64;
    FileTime    :TFileTime;
    SystemTime  :TSystemTime;
begin
    Result := 0;
    try
    {Update input image}
    Result := TCatIoInputUpdate( TASK_1_PORTNUMBER );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoInputUpdate() error:%d', [ Result ] ) );
    {View inputs}
    ViewData( ListBox2, InputImagePtr, sizeof(TInputImage) );
    {Calculate new output values (running light)}
    CalculateNewOutputs( OutputImagePtr, sizeof(TOutputImage));
    {Update output image}
    Result := TCatIoOutputUpdate( TASK_1_PORTNUMBER );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOutputUpdate() error:%d', [ Result ] ) );
    {View outputs}
    ViewData( ListBox3, OutputImagePtr, sizeof(TOutputImage) );
    except
    Timer1.Enabled := false;
    ListBox1.Items.Insert(0, Format('TCatIoDrv exception error:%d', [ Result ] ) );
    end;
    Result := TCatIoGetCpuCounter(CpuCounter);
    Label4.Caption := Format('TCatIoGetCpuCounter() result:
%d, CPU counter:0x%x', [Result,CpuCounter] );
    Result := TCatIoGetCpuTime(FileTime);
    FileTimeToSystemTime( FileTime, SystemTime );
```

```
    Label5.Caption := Format('TCatIoGetCpuTime() result:%d, CPU time: %d:%d:%d:%d',
    [Result,SystemTime.wHour, SystemTime.wMinute, SystemTime.wSecond, SystemTime.wMilliseconds]);
end;
```

Die Routinen zum aktivieren/deaktivieren des Timers:

```
procedure TForm1.StartButtonClick(Sender: TObject);
begin
    StartButton.Enabled := false;
    StopButton.Enabled := true;
    Timer1.Enabled := true;
end;
procedure TForm1.StopButtonClick(Sender: TObject);
begin
    StartButton.Enabled := true;
    StopButton.Enabled := false;
    Timer1.Enabled := false;
end;
```

Die Routine für den IO-Reset:

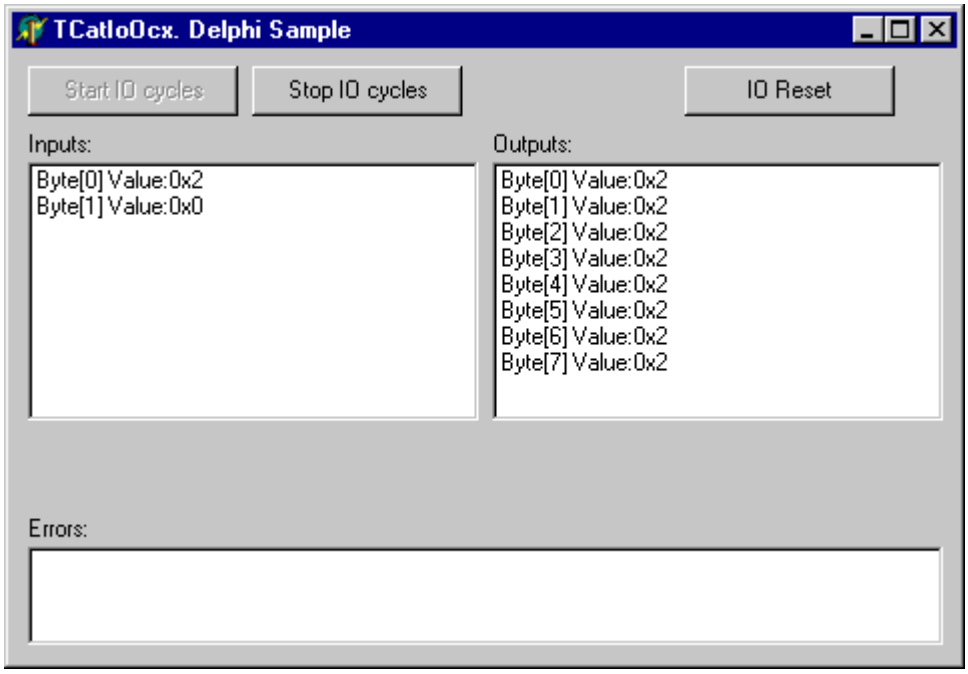
```
procedure TForm1.Button1Click(Sender: TObject);
var Result : integer;
begin
    Result := TCatIoReset();
    if ( Result <> 0 ) Then
    ListBox1.Items.Insert( 0, Format('TCatIoReset() error:%d', [ Result ] ) );
end;
```

```
procedure TForm1.InitControls();
var Row   :integer;
begin
    StartButton.Enabled := true;
    StopButton.Enabled := false;

    Timer1.Enabled := false;
    Timer1.Interval := 100; {100 ms}

    for Row:= 0 To MAX_INPUT_IMAGE_BYTESIZE - 1 do
    ListBox2.Items.Add( Format( 'Byte[%d]', [Row] ) );

    for Row:= 0 To MAX_OUTPUT_IMAGE_BYTESIZE - 1 do
    ListBox3.Items.Add( Format( 'Byte[%d]', [Row] ) );
end;

procedure TForm1.CalculateNewOutputs(pData   : Pointer; cbSize   :integer);
var i:integer;
    pByte : ^Byte;
begin
    if ( pData <> NIL ) And (cbSize > 0) then
    begin
    for i:= 0 to  cbSize - 1 do
    begin
        pByte := Pointer(Integer(pData) + i);
        if ( pByte^ = 0 ) then pByte^ := 1
        else  pByte^ := pByte^ shl 1;
    end;
    end;
end;

procedure TForm1.ViewData( var ListBox : TListBox; pData   : Pointer; cbSize   :integer );
var
  ByteOff    : Integer;
  pByte      : ^Byte;
begin
    if ( pData <> NIL ) And (cbSize > 0) then
    begin
    for ByteOff := 0 to cbSize - 1 do
    begin
        pByte:=  Pointer( integer(pData) + ByteOff );
        ListBox.Items.Strings[ByteOff] := Format('Byte[%d] Value:0x%x',[ ByteOff, pByte^ ] );
    end;
    end;
end;
end.
```

Die Sourcen zu diesem Beispiel können sie hier entpacken: https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082516491.exe

7.2 TwinCAT I/O Ring 3 OCX Delphi-Applikation

Die Sourcen zu diesem Beispiel können sie hier entpacken: https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082517899.exe



Systemvoraussetzungen:

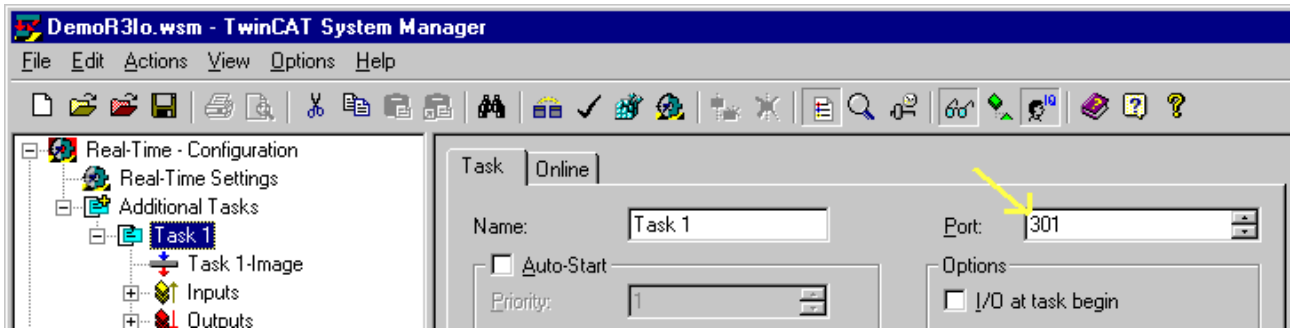
- TwinCAT 2.7 oder höher
- TCatIoOcx.Ocx
- Delphi 5.0

Beschreibung

Aus der Delphi-Applikation soll das Mapping von dem Prozeßabbild der Ein-/Ausgänge einer zusätzlichen Task im TwinCAT System Manager getriggert werden. Die Zykluszeit beträgt 100ms und wird mit Hilfe eines Multimedia-Timers erzeugt. Die Onlinewerte der Prozessabbilder werden in zwei ListBoxen angezeigt. Über den *IO Reset*-Button kann ein Reset der TwinCAT IO-Geräte durchgeführt werden. Das Mapping der Eingänge und Ausgänge kann über die Buttons *Start IO cycle* und *Stop IO cycle* gestartet bzw. gestoppt werden. Eventuelle Fehlermeldungen werden in einer weiteren ListBox ausgegeben.

Die Task-Konfiguration in TwinCAT System Manager

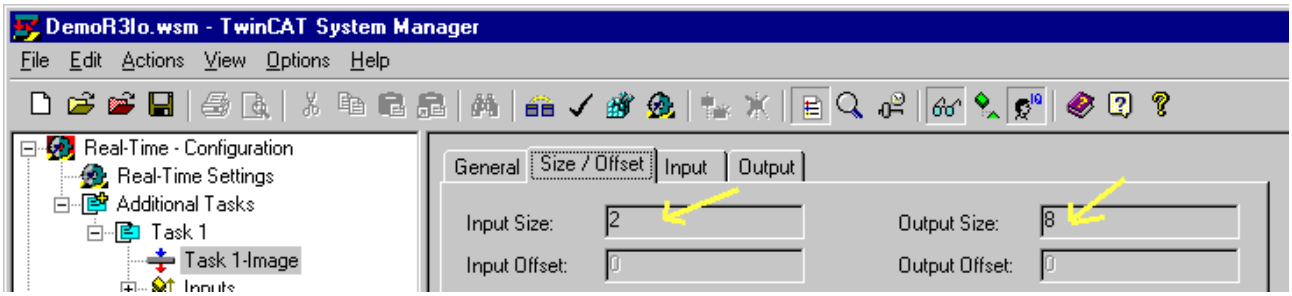
Für die zusätzliche Task wurde im TwinCAT System Manager die Portnummer 301 konfiguriert.



Das Prozessabbild hat folgende Größe:

Eingangsabbild: 2 Byte;

Ausgangsabbild: 8 Byte;



Die Portnummer und die Bytegröße des Prozessabbildes wurden in der *TCatIoOcxDelphiUnit.pas*-Unit als Konstanten definiert und müssen bei anderen Werten entsprechend geändert werden.

Einbinden der TCatIoOcx ActiveX-Komponente

Um die ActiveX-Komponente TCatIoOcx in Delphi-Applikationen benutzen zu können muß diese in die Komponentenpalette eingebuden werden. ActiveX-Komponenten können über den Menuebefehl: *Component->Import ActiveX Control...* eingebunden werden. Wählen Sie aus der Liste der installierten Komponenten das *TCatIoOcx ActiveX Control Module* aus und bestätigen mit *Install...* Bestätigen Sie dann die nachfolgenden Dialogfenster. Beim Erfolg befindet sich das TCatIoOcx in der Komponentenpalette der ActiveX-Komponenten.

Die Applikation

In der Event-Funktion *FormCreate* wird die Methode *TCatIoOcxOpen* aufgerufen. Beim Erfolg liefert diese Funktion eine Null und es wird eine Verbindung zum TwinCAT I/O Subsystem aufgebaut. Ein Fehler wird in der ListBox an den Benutzer ausgegeben. Beim Beenden der Anwendung muß die Verbindung zum TwinCAT I/O Subsystem abgebaut werden. Dies geschieht in der Event-Funktion *FormDestroy,* dabei wird die Methode *TCatIoOcxClose* aufgerufen. Wurde die Verbindung erfolgreich aufgebaut, dann werden zwei weitere Methoden aufgerufen: *TCatIoOcxGetInputPtr* und *TCatIoOcxGetOutputPtr*. Diese Methoden liefern Zeiger auf die Prozeßabbilder der Ein-/Ausgänge. Über diese Zeiger kann auf die Prozeßdaten der Eingänge lesend und der Ausgänge schreibend zugegriffen werden. In unserem Beispiel wird aber nicht über diese Zeiger auf die Prozeßdaten zugegriffen, sondern über zwei Datenpuffer: InputImage und OutputImage. Bei der Definition der Datenpuffer muß das 4 Byte-Alignment beachtet werden.

D.h. für jedes angefangene DWord der Prozeßdaten müssen 4 Byte Datenpuffer reserviert werden. In userem Beispiel ist der Datenpuffer der Eingänge 4 Byte (tatsächliche Größe 2 Byte) und der Ausgänge 12 Byte (tatsächliche Größe 8 Byte) groß. Die Datenpuffer dürfen größer, nicht aber kleiner sein.

```
unit TCatIoOcxDelphiUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Grids, OleCtrls, TCATIOOCXLib_TLB;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Timer1: TTimer;
    StartButton: TButton;
    StopButton: TButton;
    Label1: TLabel;
    Button1: TButton;
    Label2: TLabel;
    Label3: TLabel;
    ListBox2: TListBox;
    ListBox3: TListBox;
    TCatIoOcx1: TTCatIoOcx;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure StartButtonClick(Sender: TObject);
    procedure StopButtonClick(Sender: TObject);
```

```
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    procedure InitControls();
    procedure CalculateNewOutputs(pData   : Pointer; cbSize   :integer);
    procedure ViewData( var ListBox : TListBox; pData   : Pointer; cbSize   :integer);
  public
    { Public declarations }
  end;

const
 TASK_1_PORTNUMBER = 301;
    MAX_INPUT_IMAGE_BYTESIZE = 2;
    MAX_OUTPUT_IMAGE_BYTESIZE = 8;

type
    TInputImage       = ARRAY[ 0..MAX_INPUT_IMAGE_BYTESIZE DIV 4 ] Of Integer;
    TOutputImage      = ARRAY[ 0..MAX_OUTPUT_IMAGE_BYTESIZE DIV 4 ] Of Integer;

var
    Form1: TForm1;
InputImage    :TInputImage;
OutputImage       :TOutputImage;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var Result  : integer;
    InPtr, OutPtr :Integer;
begin
    InitControls();

    Result := TCatIoOcx1.TCatIoOcxOpen();
    if ( Result <> 0 ) then
    ListBox1.Items.Insert(0, Format('TCatIoOcxOpen() error:%d', [ Result ] ) )
    else
    begin
    {get/initialize pointer to the input image}
    Result := TCatIoOcx1.TCatIoOcxGetInputPtr( TASK_1_PORTNUMBER, InPtr, MAX_INPUT_IMAGE_BYTESIZE );
    if (  Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOcxGetInputPtr() error:%d', [ Result ] ) );

    {get/initialize pointer to the output image}
    Result := TCatIoOcx1.TCatIoOcxGetOutputPtr( TASK_1_PORTNUMBER, OutPtr, MAX_OUTPUT_IMAGE_BYTESIZE
 );
    if (  Result <> 0 ) Then
        ListBox1.Items.Insert(0, Format('TCatIoOcxGetOutputPtr() error:%d', [ Result ] ) );
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var Result : integer;
begin
    Result := TCatIoOcx1.TCatIoOcxClose();
    if ( Result <> 0 ) then
    MessageBox(0, 'TCatIoOcxClose() error!', 'Error', MB_OK);
end;
```

Wurde der Multimedia-Timer aktiviert, dann wird zyklisch die *Timer1Timer*-Ereignisroutine aufgerufen. In dieser Routine werden jedesmal folgende Methoden aufgerufen:

- TCatIoOcxInputUpdate (Die Methode triggert die Aktualisierung des Prozessabbildes der Eingänge (EIngänge werden gelesen));
- TCatIoOcxGetInputData (Die Methode liest die Prozeßdaten der Eingänge in den *InputImage*-Datenpuffer)
- ViewData (Hilfsprozedur: Zeigt die aktuellen Werte der Eingänge in einer ListBox an);
- CalculateNewOutputs (Hilsfsprozedur: Generiert/Verändert die Werte der Ausgänge (z.B. Lauflicht));
- TCatIoOcxSetOutputData (Die Methode setzt die Prozeßdaten der Ausgänge mit den Daten aus dem *OutputImage*-Datenpuffer)
- TCatIoOcxOutputUpdate (Die Methode triggert die Aktualisierung des Prozessabbildes der Ausgänge (Ausgänge werden geschrieben));
- ViewData (Hilfsprozedur: Zeigt die aktuellen Werte der Ausgänge in einer ListBox an);

```
procedure TForm1.Timer1Timer(Sender: TObject);
var Result : integer;
begin
    try

    {Update input image}
      Result := TCatIoOcx1.TCatIoOcxInputUpdate( TASK_1_PORTNUMBER );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOcxInputUpdate() error:%d', [ Result ] ) );

    {read input values}
      Result := TCatIoOcx1.TCatIoOcxGetInputData( TASK_1_PORTNUMBER,  InputImage[0], MAX_INPUT
_IMAGE_BYTESIZE );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOcxGetInputData() error:%d', [ Result ] ) );

    {View inputs}
    ViewData( ListBox2, @InputImage, MAX_INPUT_IMAGE_BYTESIZE );

    {Calculate new output values (running light)}
    CalculateNewOutputs( @OutputImage, MAX_OUTPUT_IMAGE_BYTESIZE );

    {write output values}
      Result := TCatIoOcx1.TCatIoOcxSetOutputData( TASK_1_PORTNUMBER,  OutputImage[0], MAX_OUT
PUT_IMAGE_BYTESIZE );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOcxSetOutputData() error:%d', [ Result ] ) );

    {Update output image}
      Result := TCatIoOcx1.TCatIoOcxOutputUpdate( TASK_1_PORTNUMBER );
    if ( Result <> 0 ) then
        ListBox1.Items.Insert(0, Format('TCatIoOcxOutputUpdate() error:%d', [ Result ] ) );

    {View outputs}
    ViewData( ListBox3, @OutputImage, MAX_OUTPUT_IMAGE_BYTESIZE );

    except
    Timer1.Enabled := false;
    ListBox1.Items.Insert(0, 'TCatIoOcx exception error:%d' );
    end;

end;
```

Die Routinen zum aktivieren/deaktivieren des Timers:

```
procedure TForm1.StartButtonClick(Sender: TObject);
begin
    StartButton.Enabled := false;
    StopButton.Enabled := true;
    Timer1.Enabled := true;
end;

procedure TForm1.StopButtonClick(Sender: TObject);
begin
    StartButton.Enabled := true;
    StopButton.Enabled := false;
    Timer1.Enabled := false;
end;
```

Die Routine für den IO-Reset:

```
procedure TForm1.Button1Click(Sender: TObject);
var Result : integer;
begin
    Result := TCatIoOcx1.TCatIoOcxReset();
    if ( Result <> 0 ) Then
    ListBox1.Items.Insert( 0, Format('TCatIoOcxReset() error:%d', [ Result ] ) );
end;
```

```
procedure TForm1.InitControls();
var Row   :integer;
begin
    StartButton.Enabled := true;
    StopButton.Enabled := false;

    Timer1.Enabled := false;
    Timer1.Interval := 100; {100 ms}

    for Row:= 0 To MAX_INPUT_IMAGE_BYTESIZE - 1 do
    ListBox2.Items.Add( Format( 'Byte[%d]', [Row] ) );
```

```
    for Row:= 0 To MAX_OUTPUT_IMAGE_BYTESIZE - 1 do
    ListBox3.Items.Add( Format( 'Byte[%d]', [Row] ) );
end;

procedure TForm1.CalculateNewOutputs(pData   : Pointer; cbSize   :integer);
var i:integer;
    pByte : ^Byte;
begin
    if ( pData <> NIL ) And (cbSize > 0) then
    begin
    for i:= 0 to  cbSize - 1 do
    begin
        pByte := Pointer(Integer(pData) + i);
        if ( pByte^ = 0 ) then pByte^ := 1
        else  pByte^ := pByte^ shl 1;
    end;
    end;
end;

procedure TForm1.ViewData( var ListBox : TListBox; pData   : Pointer; cbSize   :integer );
var
  ByteOff    : Integer;
  pByte      : ^Byte;
begin
    if ( pData <> NIL ) And (cbSize > 0) then
    begin
    for ByteOff := 0 to cbSize - 1 do
    begin
        pByte:=  Pointer( integer(pData) + ByteOff );
        ListBox.Items.Strings[ByteOff] := Format('Byte[%d] Value:0x%x',[ ByteOff, pByte^ ] );
    end;
    end;
end;



end.
```

Die Sourcen zu diesem Beispiel können sie hier entpacken: https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082517899.exe

7.3 C++ Beispiel

7.3.1 Konfiguration

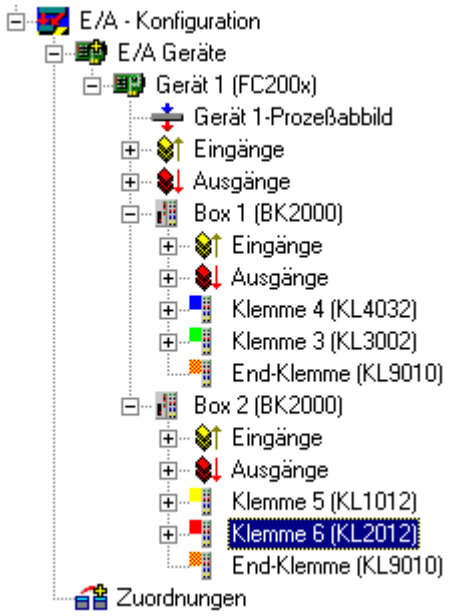
Der TwinCAT System Manager wird zur Konfiguration des TwinCAT Systems verwendet.

Für dieses Beispiel ist folgendes zu konfigurieren:

- Das I/O-Gerät [▶ 33] (z. B. Beckhoff FC2001)
- Zwei I/O Tasks [▶ 34]
- Die Prozessabbilder [▶ 34] dieser Tasks
- Das Mapping [▶ 36] zwischen den I/O Tasks und den I/O Geräten

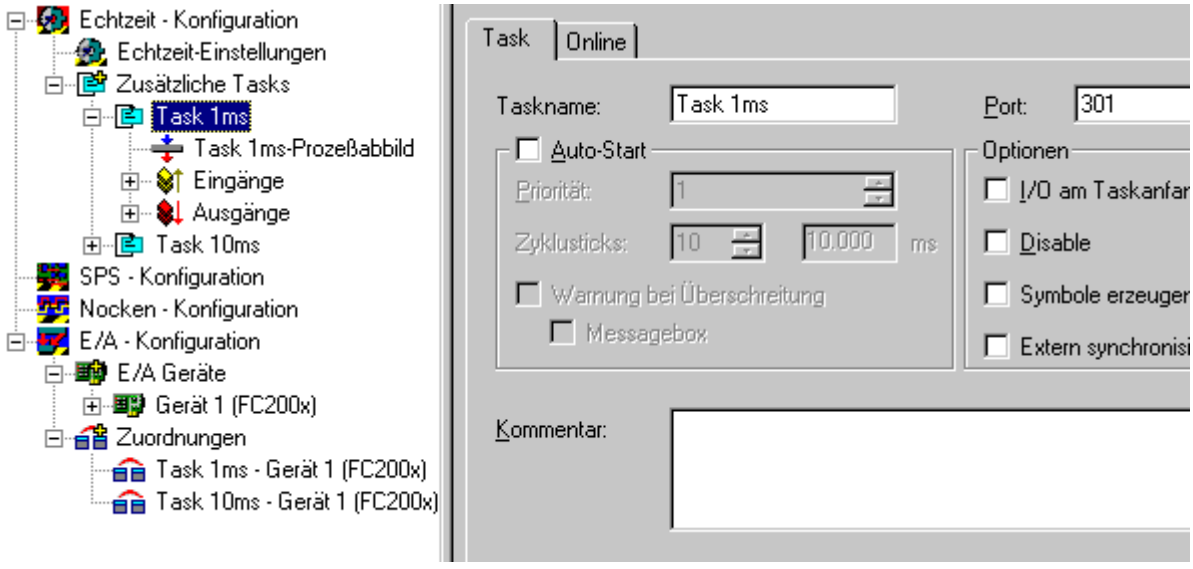
7.3.1.1 Das E/A Gerät

Das Zufügen von Geräten und Boxen ist in der Dokumentation: TwinCAT System Manager - E/A Konfiguration beschrieben.



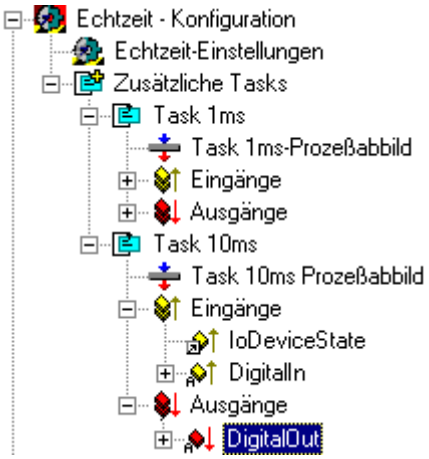
7.3.1.2 Die E/A Tasks

An dieser Stelle müssen die E/A Tasks zugefügt werden. Bis zu 5 Tasks sind verfügbar. Im Karteireiter "Task " auf der rechten Seite wird automatisch eine Portnummer von 301, für Task 1, bis 305 , für Task 5 eingetragen. Bitte vergewissern Sie sich, dass die Checkbox "Auto Start " deaktiviert ist. In diesem Programm laufen die Echtzeittasks nicht, lediglich ihre Prozessabbilder werden benutzt. Detaillierte Informationen über die Einstellungen der Tasks finden Sie unter: TwinCAT System Manager – Task-Einstellungen.



7.3.1.3 Die Prozessabbilder

Nachdem die beiden I/O Tasks zugefügt wurden, muss für jeden Task das Prozessabbild konfiguriert werden. Weitere Informationen dazu finden Sie ebenfalls in der Dokumentation TwinCAT System Manager.



Um sicher zu stellen, dass die im System Manager benutzten Datentypen der Taskvariablen zur C++ Applikation passen, generiert der System Manager für jeden Task eine Header Datei.

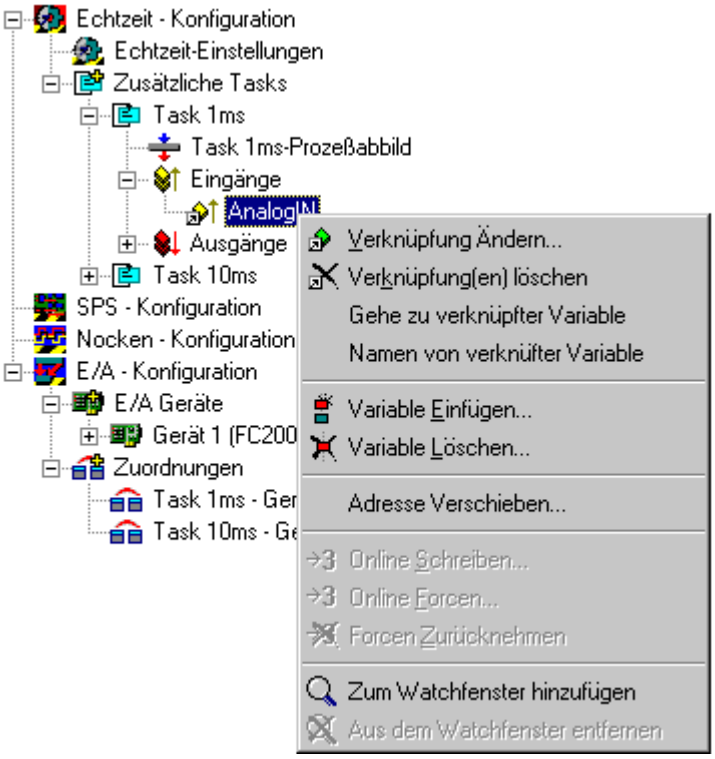
Das Beispiel einer solchen Header Datei ist nachfolgend dargestellt:

```
//////////////////////////////////////////////////////////////////////
//
// BECKHOFF Industrie Elektronik
//
// TwinCAT IO HeaderFile
//
//////////////////////////////////////////////////////////////////////
//
// Task1ms.h

#define TASK1MS_PORTNUMBER  301

#define TASK1MS_INPUTSIZE   2
#define TASK1MS_OUTPUTSIZE  4

#pragma pack(push, 1)

typedef struct
{
    short   AnalogIN;
} Task1ms_Inputs, *PTask1ms_Inputs;

typedef struct
{
    short   AnalogOut_1;
    short   AnalogOut_2;
} Task1ms_Outputs, *PTask1ms_Outputs;

#pragma pack(pop)
```

Die Header Datei enthält die ADS Port Nummer der Task, die Größe des Eingangs- und Ausgangsprozessabbildes (in Bytes) und die Variablen des Prozessabbilds als strukturierte Bestandteile.

7.3.1.4 Das Mapping (Zuordnung)

Verknüpfen Sie die Variablen mit dem korrespondierenden Feldbus I/O. Genauere Informationen zur Variablen Verknüpfung finden Sie in der Dokumentation TwinCAT System Manager - Variablen Verknüpfungen.



Nach der Konfiguration muss noch die Sicherung in der Registry erfolgen. Danach starten/ oder re-starten Sie das TwinCAT System...

... mit diesem Button.

7.3.2 Implementierung

7.3.2.1 Beispiel (OS NT/2000/XP)/CE

```
//////////////////////////////////////////////////////////////////////////////
//
// Beckhoff Automation
//
// TwinCAT® I/O sample program
//
//////////////////////////////////////////////////////////////////////////////
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include "mmsystem.h"
#include "TCatIoApi.h"    // header file shipped with TwinCAT® I/O
#include "Task1ms.h"  // TwinCAT® System Manager generated
#include "Task10ms.h"     // TwinCAT® System Manager generated

#define IOERR_IOSTATEBUSY               0x2075
#define TASK1MS_DELAY           1 // ms
#define TASK10MS_DELAY          10 // ms
#define DELAY_IN_100NS(x)               (x*10000)

PTask1ms_Outputs        pT1msOut;
PTask1ms_Inputs         pT1msIn;
PTask10ms_Outputs           pT10msOut;
PTask10ms_Inputs        pT10msIn;
```

```
////////////////////////////////////////////////////////////////////////////
void CALLBACK TimerProc1( UINT uID, UINT uMsg, DWORD dwUser, DWORD dw1, DWORD
dw2 )
{
    static int i=0;
    long nError;
    static __int64 nLastCpuTime=0, nActCpuTime=0;

    TCatIoGetCpuTime( &nActCpuTime );   // 0.5 ms, filter extra events
    if ( (nActCpuTime - nLastCpuTime) > (DELAY_IN_100NS(TASK1MS_DELAY)/2) )
    {
        nLastCpuTime =nActCpuTime;
        // first try to get the inputs and test if input update succeeded
        if ( (nError = TCatIoInputUpdate( TASK1MS_PORTNUMBER )) == 0 )
        {
            // do your calculation and logic
            pT1msOut->AnalogOut = pT1msIn->AnalogIn;
            // start the I/O update and field bus cycle
            TCatIoOutputUpdate( TASK1MS_PORTNUMBER );
            }
            else
                printf( "TCatInputUpdate(%d) %d failed with 0x%x !\n",
                    TASK1MS_PORTNUMBER, i++, nError );
        }
    }
```

```
////////////////////////////////////////////////////////////////////////////
void CALLBACK TimerProc2( UINT uID, UINT uMsg, DWORD dwUser, DWORD dw1, DWORD
dw2 )
{
    static int i=0;
    long nError;
    static __int64 nLastCpuTime=0, nActCpuTime=0;

    TCatIoGetCpuTime( &nActCpuTime ); // 5 ms, filter extra events
    if ( (nActCpuTime - nLastCpuTime) > (DELAY_IN_100NS(TASK10MS_DELAY)/2) )
    {
        nLastCpuTime = nActCpuTime;
        // try to get the inputs and test if input update succeeded
        if ( (nError = TCatIoInputUpdate( TASK10MS_PORTNUMBER )) == 0 )
        {
            // optionally test the device state, zero is ok.
            if ( pT10msIn->IoDeviceState != 0 )
                printf( "I Device Error !\n" );
            else
            {
                // do your calculation and logic
                pT10msOut->DigitalOut[0] = pT10msIn->DigitalIn[0];
                pT10msOut->DigitalOut[1] = pT10msIn->DigitalIn[1];
                // start the I/O update and field bus cycle
                TCatIoOutputUpdate( TASK10MS_PORTNUMBER );
            }
        }
        else
            printf("TCatInputUpdate(%d) %d failed with 0x%x !\n",
            TASK10MS_PORTNUMBER, i++, nError );
    }
}
```

```
////////////////////////////////////////////////////////////////////////////
void main()
{
    MMRESULT idTimer1, idTimer2;
    timeBeginPeriod(1);
    // always call TCatIoOpen first.
    if ( TCatIoOpen() == 0 )
    {
        // get the process image pointer
        if ( TCatIoGetOutputPtr(TASK1MS_PORTNUMBER, (void**)&pT1msOut,sizeof(Task1ms_Outputs) ) == 0
 &&
            TCatIoGetInputPtr( TASK1MS_PORTNUMBER,(void**)&pT1msIn, sizeof(Task1ms_Inputs) ) == 0 &&
            TCatIoGetOutputPtr(TASK10MS_PORTNUMBER,
(void**)&pT10msOut, sizeof(Task10ms_Outputs) )== 0 &&
            TCatIoGetInputPtr(TASK10MS_PORTNUMBER,
(void**)&pT10msIn, sizeof(Task10ms_Inputs) )== 0 )
        {
            //do some other initialization
            idTimer1 = timeSetEvent( TASK1MS_DELAY, 0,TimerProc1, 0, TIME_PERIODIC|
TIME_CALLBACK_FUNCTION );
            idTimer2 = timeSetEvent(TASK10MS_DELAY, 0, TimerProc2, 0, TIME_PERIODIC|
TIME_CALLBACK_FUNCTION );
```

```
            //just wait for the end
            getch();
            // events are no longer needed
            timeKillEvent(idTimer1 );
            timeKillEvent( idTimer2 );
        }
        // free resources
        TCatIoClose();
    }
    timeEndPeriod(1);
}
```

7.3.2.2 Beispiel (OS CE)

https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082519307.zip

```
////////////////////////////////////////////////////////////////////////////////
//
// Beckhoff Automation
//
// TwinCAT CE ® I/O sample program using TwinCAT Timer.
//
////////////////////////////////////////////////////////////////////////////////
```

```
// TcTimerWrap.lib and TCatIoDrvW32.lib must be linked for this project
//
#include "stdio.h"
#include "windows.h"
#include "TCatIoW32Api.h"     // header file shipped with TwinCAT® I/O
#include "TcTimerApi.h"       // header file shipped with TwinCAT® I/O
#include "Task1ms.h"  // TwinCAT® System Manager generated
#include "Task10ms.h"     // TwinCAT® System Manager generated

#define IOERR_IOSTATEBUSY               0x2075
#define TASK1MS_DELAY           1 // Ticks
#define TASK10MS_DELAY          10 // Ticks
#define DELAY_IN_100NS(x)               (x*10000)

PTask1ms_Outputs        pT1msOut;
PTask1ms_Inputs         pT1msIn;
PTask10ms_Outputs           pT10msOut;
PTask10ms_Inputs        pT10msIn;
HANDLE          g_hEvent1, g_hEvent2;
BOOL            g_bExit;
```

```
////////////////////////////////////////////////////////////////////////////////
static DWORD WINAPI TimerProc1( LPVOID lpParam)
{
    static int i=0;
    long nError;

    while(!g_bExit)
    {
        WaitForSingleObject(g_hEvent1,INFINITE);
        // first try to get the inputs and test if input update succeeded
        if ( (nError = TCatIoInputUpdate( TASK1MS_PORTNUMBER )) == 0 )
        {
            // do your calculation and logic
            pT1msOut->AnalogOut = pT1msIn->AnalogIn;
            // start the I/O update and field bus cycle
            TCatIoOutputUpdate( TASK1MS_PORTNUMBER );
        }
        else
            printf( "TCatInputUpdate(%d) %d failed with 0x%x !\n",
                    TASK1MS_PORTNUMBER, i++, nError );
    }
    return 0;
}
```

```
////////////////////////////////////////////////////////////////////////////////
static DWORD WINAPI TimerProc2( LPVOID lpParam)
{
    static int i=0;
    long nError;

    while(!g_bExit)
    {
        WaitForSingleObject(g_hEvent2,INFINITE);
```

```
        if ( (nError = TCatIoInputUpdate( TASK10MS_PORTNUMBER )) == 0 )
        {
            // optionally test the device state, zero is ok.
            if ( pT10msIn->IoDeviceState != 0 )
                printf( "I Device Error !\n" );
            else
            {
                // do your calculation and logic
                pT10msOut->DigitalOut[0] = pT10msIn->DigitalIn[0];
                pT10msOut->DigitalOut[1] = pT10msIn->DigitalIn[1];
                // start the I/O update and field bus cycle
                TCatIoOutputUpdate( TASK10MS_PORTNUMBER );
            }
        }
        else
            printf("TCatInputUpdate(%d) %d failed with 0x%x !\n",
            TASK10MS_PORTNUMBER, i++, nError );
    }
    return 0;
}
```

```
//////////////////////////////////////////////////////////////////////////////
int WINAPI WinMain( HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR lpCmdLine,
    int nCmdShow)
{
    INT idTimer1, idTimer2;
    TCHAR pszTask1Name[MAX_PATH] = TEXT("evt_TASK_1_TICK");
    TCHAR pszTask2Name[MAX_PATH] = TEXT("evt_TASK_10_TICK");
    HANDLE hThreadTask1, hThreadTask2;
    long m_nAdsState;
    DWORD m_dwBaseTime;
    // always call TCatIoOpen first.
    if ( TCatIoOpen() == 0 )
    {
         m_nAdsState = TCatGetState();
         if( (TcTimerInitialize() != STATUS_SUCCESS) || (m_nAdsState != ADSSTATE_RUN) )
        {
            TCatIoClose();
             printf("Failed to Initialize TcTimer or TwinCAT not in RUN mode\n");
             getchar();
            return FALSE;
         }
         m_dwBaseTime = TcTimerGetTickTime()/10;
        printf("Base Time = %d microseconds\n", m_dwBaseTime);

        // get the process image pointer
        if ( TCatIoGetOutputPtr(TASK1MS_PORTNUMBER, (void**)&pT1msOut,sizeof(Task1ms_Outputs) ) == 0
 &&
            TCatIoGetInputPtr( TASK1MS_PORTNUMBER,(void**)&pT1msIn, sizeof(Task1ms_Inputs) ) == 0 &&
            TCatIoGetOutputPtr(TASK10MS_PORTNUMBER,
(void**)&pT10msOut, sizeof(Task10ms_Outputs) )== 0 &&
            TCatIoGetInputPtr(TASK10MS_PORTNUMBER,
(void**)&pT10msIn, sizeof(Task10ms_Inputs) )== 0 )
        {
            // Create Events for the Tasks
            g_hEvent1 = CreateEvent(NULL,FALSE,FALSE,pszTask1Name);
            g_hEvent2 = CreateEvent(NULL,FALSE,FALSE,pszTask2Name);
             // Set Events in TcTimer
            idTimer1 = TcTimerSetEvent( TASK1MS_DELAY,  pszTask1Name, TIME_CALLBACK_EVENT_PULSE|
TIME_PERIODIC );
            idTimer2 = TcTimerSetEvent( TASK10MS_DELAY, pszTask2Name, TIME_CALLBACK_EVENT_PULSE|
TIME_PERIODIC );
             g_bExit = FALSE;
             //Create thread
            hThreadTask1 = CreateThread( NULL, 0, TimerProc1, NULL, CREATE_SUSPENDED, NULL);
            hThreadTask2 = CreateThread( NULL, 0, TimerProc2, NULL, CREATE_SUSPENDED, NULL);
             // Set the Priorities of the Thread
             if (CeSetThreadPriority(hThreadTask1, 26) && CeSetThreadPriority(hThreadTask2, 27))
            {
                ResumeThread(hThreadTask1);
                ResumeThread(hThreadTask2);
            }

            // Wait for the end
            printf("Press any key to End !!\n");
            getchar();
             g_bExit = TRUE;
             SetEvent(g_hEvent1);
```

```
            SetEvent(g_hEvent2);
            // events are no longer needed
           TcTimerKillEvent( idTimer1 );
           TcTimerKillEvent( idTimer2 );
           CloseHandle( g_hEvent1 );
           CloseHandle( g_hEvent2 );
       }
       // free resources
       TCatIoClose();
       TcTimerDeinitialize();
   }
    return 0;
}
```

Beachten Sie die Systemvoraussetzungen [▶ 9].

7.3.2.3 TcTimer Sample für VisualStudio 2005

https://infosys.beckhoff.com/content/1031/tcr3io/Resources/12082520715.zip

- Erzeugen Sie eine neue Konsolen Applikation im Visual Studio 2005.
- Linken Sie Ihr projekt mit TcTimerWrap.lib und TCatIoDrvW32.lib
- Fügen Sie folgenden Quellcode zu Ihrem Programm hinzu:

```
//////////////////////////////////////////////////////////////////////////////
//
// Beckhoff Automation
//
// TwinCAT CE ® I/O sample program using TwinCAT Timer.
//
//////////////////////////////////////////////////////////////////////////////
// TcTimerWrap.lib and TCatIoDrvW32.lib must be linked for this project
//

#include "stdafx.h"
#include "stdio.h"
#include "windows.h"
#include "TCatIoW32Api.h" // header file shipped with TwinCAT® I/O
#include "TcTimerApi.h" // header file shipped with TwinCAT® I/O
#include "Task1ms.h" // TwinCAT® System Manager generated
#include "Task10ms.h" // TwinCAT® System Manager generated


#define IOERR_IOSTATEBUSY 0x2075
#define TASK1MS_DELAY 1 // Ticks
#define TASK10MS_DELAY 10 // Ticks
#define DELAY_IN_100NS(x) (x*10000)

PTask1ms_Outputs pT1msOut;
PTask1ms_Inputs pT1msIn;
PTask10ms_Outputs pT10msOut;
PTask10ms_Inputs pT10msIn;
HANDLE g_hEvent1, g_hEvent2;
BOOL g_bExit;


//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
static DWORD WINAPI TimerProc1( LPVOID lpParam)
{
static int i=0;
long nError;

while(!g_bExit)
{
WaitForSingleObject(g_hEvent1,INFINITE);
// first try to get the inputs and test if input update succeeded
if ( (nError = TCatIoInputUpdate( TASK1MS_PORTNUMBER )) == 0 )
{
// do your calculation and logic
pT1msOut->AnalogOut = pT1msIn->AnalogIn;
```

```
 // start the I/O update and field bus cycle
TCatIoOutputUpdate( TASK1MS_PORTNUMBER );
}
else
printf( "TCatInputUpdate(%d) %d failed with 0x%x !\n",
TASK1MS_PORTNUMBER, i++, nError );
}
return 0;
}
////////////////////////////////////////////////////////////////////////////
static DWORD WINAPI TimerProc2( LPVOID lpParam)
{
static int i=0;
long nError;

while(!g_bExit)
{
WaitForSingleObject(g_hEvent2,INFINITE);
if ( (nError = TCatIoInputUpdate( TASK10MS_PORTNUMBER )) == 0 )
{
// optionally test the device state, zero is ok.
if ( pT10msIn->IoDeviceState != 0 )
printf( "I Device Error !\n" );
else
{
// do your calculation and logic
pT10msOut->DigitalOut[0] = pT10msIn->DigitalIn[0];
pT10msOut->DigitalOut[1] = pT10msIn->DigitalIn[1];
// start the I/O update and field bus cycle
TCatIoOutputUpdate( TASK10MS_PORTNUMBER );
}
}
else
printf("TCatInputUpdate(%d) %d failed with 0x%x !\n",
TASK10MS_PORTNUMBER, i++, nError );
}
return 0;
}


int _tmain(int argc, TCHAR* argv[])
{
INT idTimer1, idTimer2;
TCHAR pszTask1Name[MAX_PATH] = TEXT("evt_TASK_1_TICK");
TCHAR pszTask2Name[MAX_PATH] = TEXT("evt_TASK_10_TICK");
HANDLE hThreadTask1, hThreadTask2;
long m_nAdsState;
DWORD m_dwBaseTime;
// always call TCatIoOpen first.
if ( TCatIoOpen() == 0 )
{
m_nAdsState = TCatGetState();
if( (TcTimerInitialize() != STATUS_SUCCESS) && (m_nAdsState != ADSSTATE_RUN) )
{
TCatIoClose();
printf("Failed to Initialize TcTimer or TwinCAT not in RUN mode\n");
getchar();
return FALSE;
}
m_dwBaseTime = TcTimerGetTickTime()/10;
printf("Base Time = %d microseconds\n", m_dwBaseTime);

// get the process image pointer
if ( TCatIoGetOutputPtr(TASK1MS_PORTNUMBER, (void**)&pT1msOut,sizeof(Task1ms_Outputs) ) == 0
&&
TCatIoGetInputPtr( TASK1MS_PORTNUMBER,(void**)&pT1msIn, sizeof(Task1ms_Inputs) ) == 0 &&
TCatIoGetOutputPtr(TASK10MS_PORTNUMBER,(void**)&pT10msOut, sizeof(Task10ms_Outputs) )== 0 &&
TCatIoGetInputPtr(TASK10MS_PORTNUMBER,(void**)&pT10msIn, sizeof(Task10ms_Inputs) )== 0 )
{
// Create Events for the Tasks
g_hEvent1 = CreateEvent(NULL,FALSE,FALSE,pszTask1Name);
g_hEvent2 = CreateEvent(NULL,FALSE,FALSE,pszTask2Name);
// Set Events in TcTimer
idTimer1 = TcTimerSetEvent( TASK1MS_DELAY, pszTask1Name, TIME_CALLBACK_EVENT_SET|
TIME_PERIODIC );
idTimer2 = TcTimerSetEvent( TASK10MS_DELAY, pszTask2Name, TIME_CALLBACK_EVENT_SET|
TIME_PERIODIC );
g_bExit = FALSE;
// Create thread
hThreadTask1 = CreateThread( NULL, 0, TimerProc1, NULL, CREATE_SUSPENDED, NULL);
```

```
hThreadTask2 = CreateThread( NULL, 0, TimerProc2, NULL, CREATE_SUSPENDED, NULL);
// Set the Priorities of the Thread
if (CeSetThreadPriority(hThreadTask1, 26) && CeSetThreadPriority(hThreadTask2, 27))
{
ResumeThread(hThreadTask1);
ResumeThread(hThreadTask2);
}

// Wait for the end
printf("Press any key to End !!\n");
getchar();
g_bExit = TRUE;
SetEvent(g_hEvent1);
SetEvent(g_hEvent2);
// events are no longer needed
TcTimerKillEvent( idTimer1 );
TcTimerKillEvent( idTimer2 );
CloseHandle( g_hEvent1 );
CloseHandle( g_hEvent2 );
}
// free resources
TCatIoClose();
TcTimerDeinitialize();
}
return 0;
}
```

- Sollten Sie beim Kompilieren einen Linker Fehler bekommen, stellen Sie sicher dass die Compiler Option "/Zc:wchar_t-" eingeschaltet ist. (Offnen Sie die Projekt Eigenschaften -> Configuration Properties->C++->Language->Treat wchar_t as Built-in Type)

Debuggen der BeispielApplikation

Um das Programm zu debuggen führen sie die folgenden Schritte durch:

- In der Vs2005 IDE wählen Tools -> Options -> Device Tools -> Devices.
- Wählen Sie Ihr Gerät aus und öffnen der Properties Dialog.
- Wählen Sie "TCP Connect Transport" aus und geben Sie die IP Adresse Ihres Gerätes ein
- Auf Ihrem Cx Gertät navigieren sie zu \Hard Disk\System
- Starten Sie die Tools "conmanclient2" und "cmaccept"
- In der IDE wählen Sie Tools -> Connect to Device.
- Starten Sie das Programm.

8 TwinCAT IO und FCxxxx

TwinCAT IO mit FC310x, FC510x und FC520x

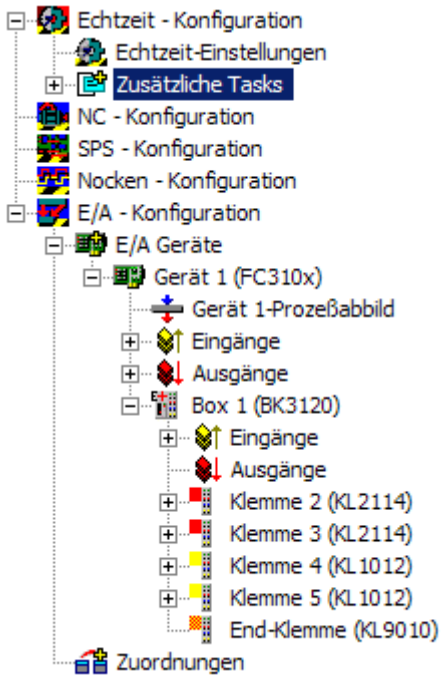
TwinCAT IO enthält die Gerätetreiber und die Konfigurationssoftware (TwinCAT System Manager) für die PC-Karten FC310x, FC510x und FC520x unter Windows NT, 2000 und XP.
Die FcIoApi-DLL bietet die Schnittstellen für beliebige Windows-Anwendungen zum Zugriff auf die Prozessdaten dieser PC-Karten.

Für die Konfiguration des Feldbusses mit dem TwinCAT System Manager sei auf die Beschreibungen der PC-Karten FC310x, FC510x und FC520x verwiesen. Auch die Diagnoseschnittstelle des jeweiligen Feldbusses ist dort ausführlich beschrieben.

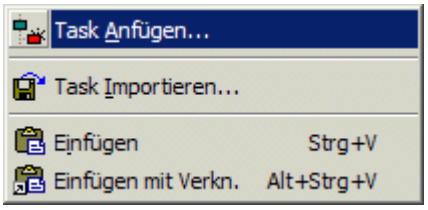
Neben der Konfiguration des Feldbusses ist im TwinCAT System Manager noch eine IO-Task einzurichten, mit der die entsprechende Feldbuskarte nachgetriggert wird. Die Zykluszeit der IO-Task entspricht der Zykluszeit auf dem Feldbus. Dazu ist mindestens eine Variable des Feldbus Devices (FC310x, FC510x bzw. FC520x) mit einer Variablen der IO-Task zu verknüpfen.

8.1 Einrichten einer IO-Task

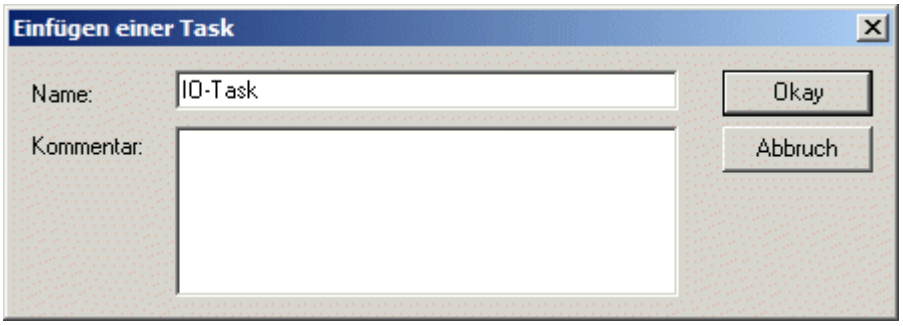
Im Baum des TwinCAT System Manager ist unter *Echtzeit-Konfiguration* eine IO-Task einzurichten:



Mit der rechten Maustaste ist auf *Zusätzliche Tasks* zu klicken, um eine Task (IO-Task) anzufügen:

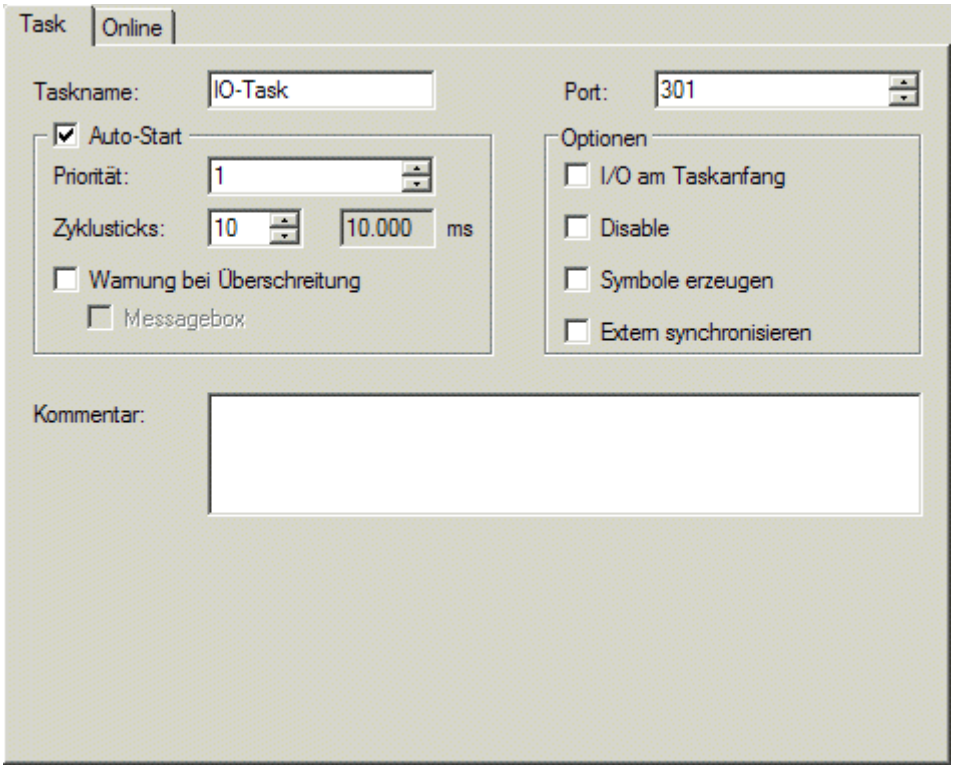


In dem erscheinenden Dialog kann der Name der Task angepasst werden:



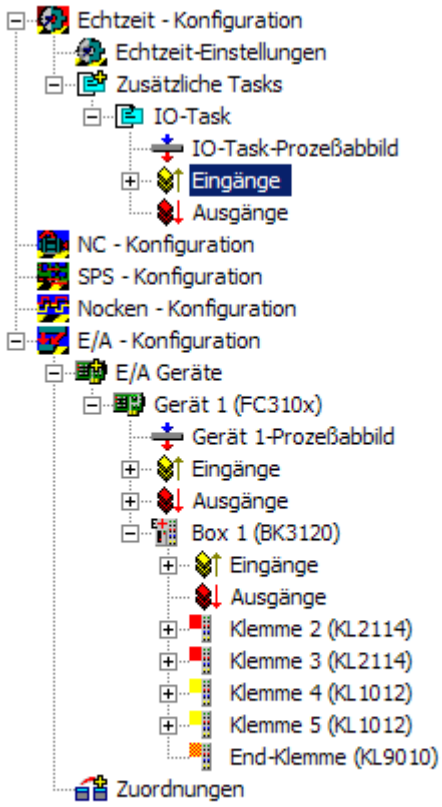
In der rechten Fensterhälfte können jetzt die Einstellungen der IO-Task angepasst werden:

Die Check-Box *Auto-Start* muss angeklickt, die Zykluszeit des Feldbusses kann dann unter *Zyklusticks* angepasst werden. Weiterhin wird noch der *Port* für die Funktionsaufrufe der FcIoApi-DLL benötigt, alle anderen Parameter brauchen nicht verändert werden.

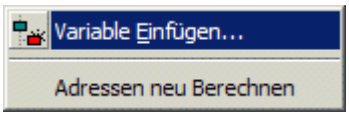


Verknüpfen der IO-Task mit dem Feldbus-Device

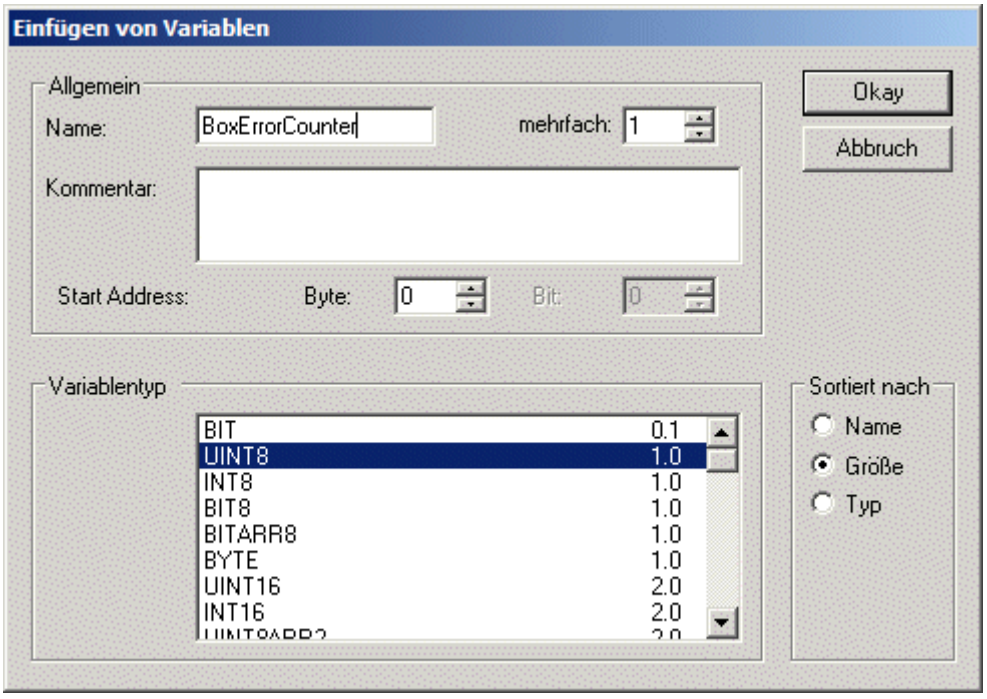
Weiterhin muss mindestens eine Variable der IO-Task mit dem Feldbus Device verknüpft werden. Dazu ist im Baum auf die Eingänge der IO-Task zu gehen:



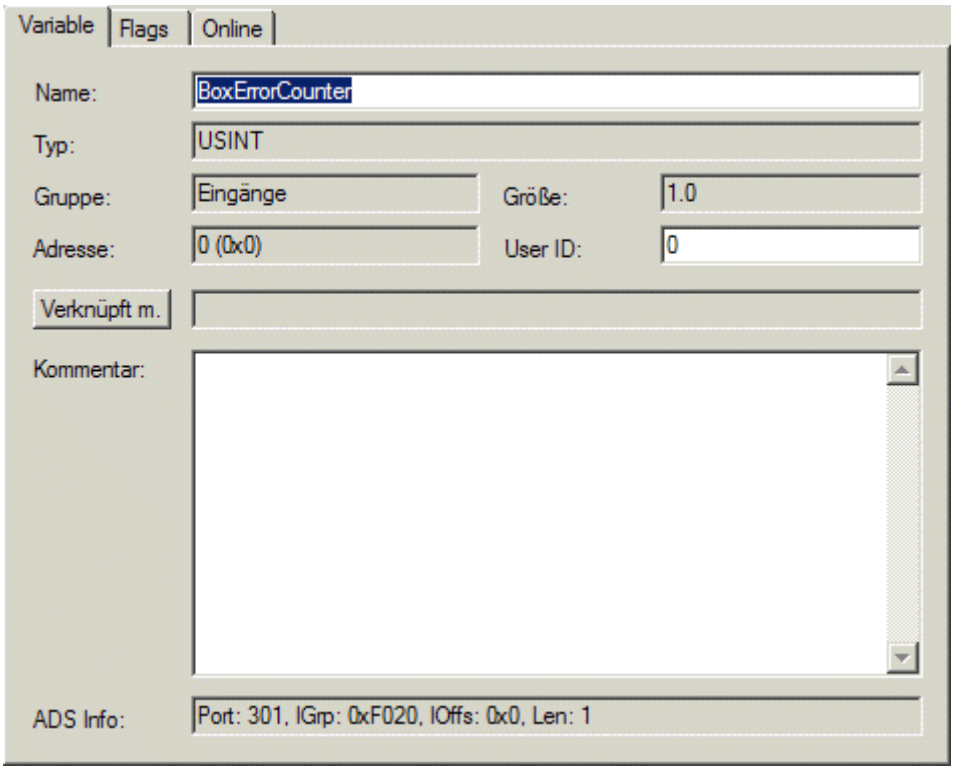
Durch Klicken der rechten Maustaste erscheint ein Pop-Up-Menü, über das eine neue Variable angefügt werden kann:



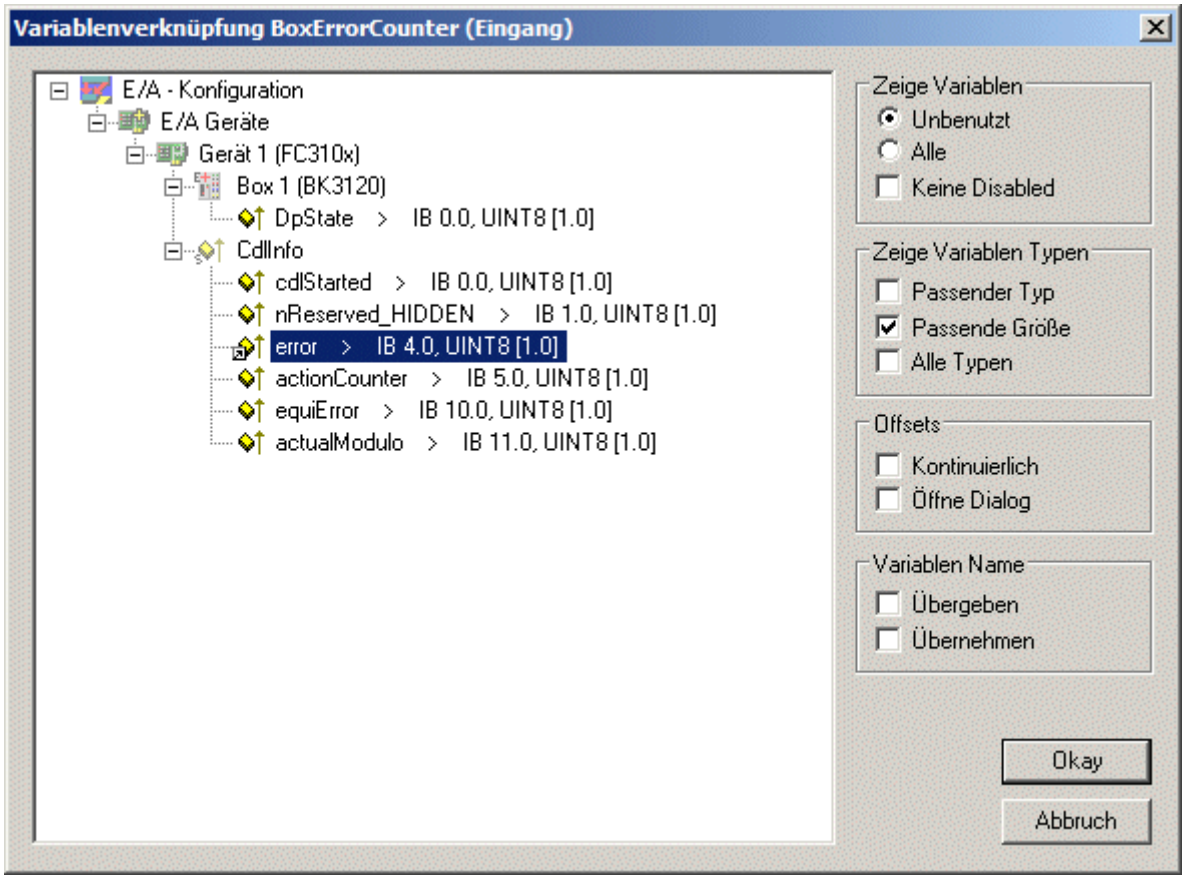
In dem Variablen-Dialog können *Name* der Variablen, *Start Address* (Adresse im Prozessabbild der IO-Task) und der *Variablentyp* eingegeben werden:



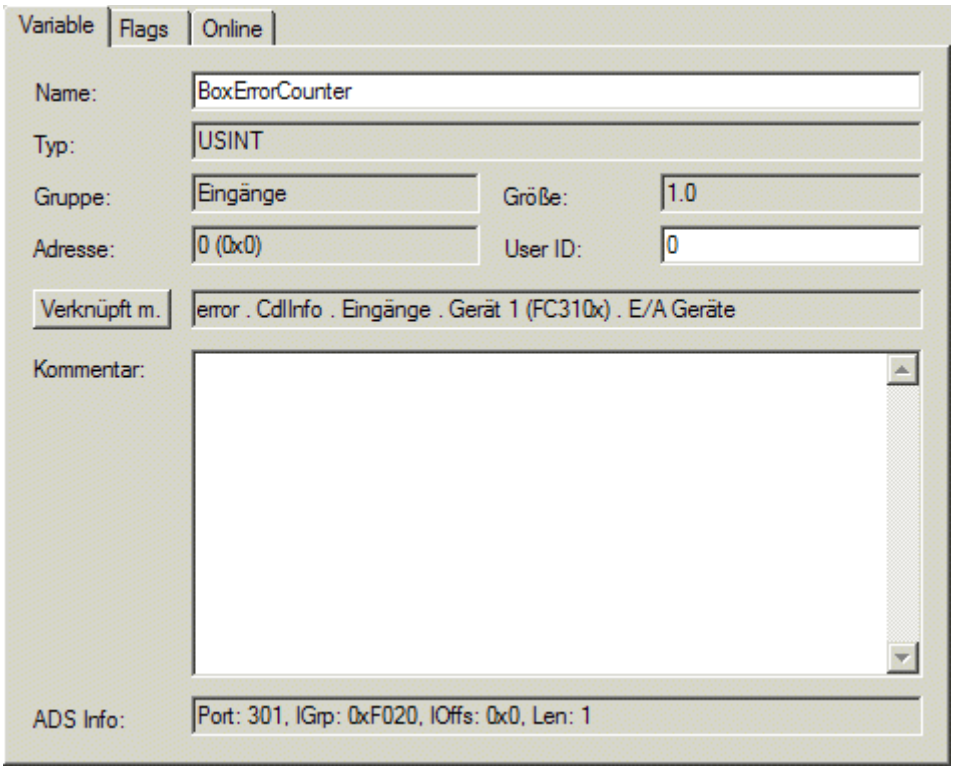
Auf dem Karteireiter "Variable" im rechten Fenster wird jetzt durch Klicken des Buttons *Verknüpft m.* die zugeordnete Variable des Feldbus Devices ausgewählt:



In dem Dialog "Variablenverknüpfung" werden jetzt die Variablen angezeigt, die verknüpft werden können. Die entsprechende Variable ist auszuwählen und der Dialog mit *Okay* zu verlassen:



Auf dem Karteireiter "Variable" ist jetzt die Verknüpfungsinformation eingetragen.



Starten des Feldbusses

Das Projekt ist schließlich noch in der Registry zu speichern (über das Regsitry-Icon im System-Manager) und TwinCAT zu starten (über das TwinCAT-Icon im TwinCAT System Manager oder in der Icon-Leiste unten rechts).

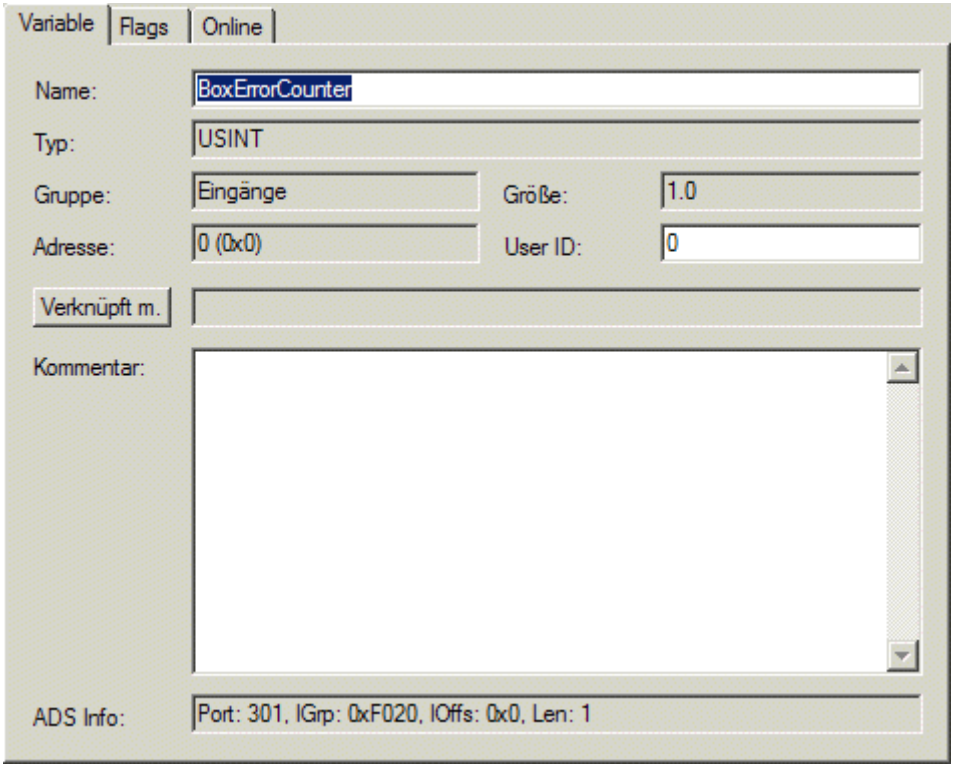
Dann sollte der Feldbus hochlaufen, über den System-Manager können die Zustände diagnostiziert oder Ausgänge gesetzt werden.

Dokumente

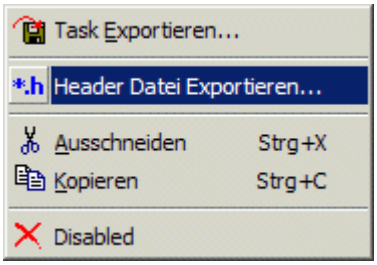
Im Folgenden werden die Funktionalitäten der Beckhoff PCI-Karte FC310x (als Master und Slave**)** für den Einsatz unter TwinCAT (NCI, PTP, PLC und IO) beschrieben.
Die folgenden Kapitel gelten auch für die PROFIBUS-Anschaltung des CX1000 (CX1500-M310 (Master) bzw. CX1500-B310 (Slave)), die Bezeichnung FC310x bezieht sich dann auch auf die CX1500-M310-Master bzw. CX1500-B310-Slave-Anschaltung.

8.2 Zugriff auf die Prozessdaten des Feldbusses

Der konsistente Zugriff über die IO-Task funktioniert nur für die Variablen des Feldbusses, die mit der IO-Task verknüpft sind. Wie weiter oben bei der Konfiguration für eine Eingangs-Variable gezeigt, müssen für alle Feldbus-Variablen, auf die konsistent zugegriffen werden soll, entsprechende Variablen der IO-Task angelegt und mit diesen verknüpft werden. Die Offsets der Zugriffsfunktionen beziehen sich auf die Adressen der Variablen der IO-Task (die Variable BoxErrorCounter befindet sich auf Adresse 0 des Eingangsprozessabbildes der IO-Task):



Das Prozessabbild der IO-Task kann in Form einer H-Datei exportiert werden, wenn man mit der rechten Maustaste auf die IO-Task im Baum klickt:



StartImageUpdate

long StartImageUpdate(int portNo, int time, int outpLength, int inpLength);

Die Funktion StartImageUpdate muss beim Hochlauf der Anwendung einmal aufgerufen werden, um den Timer zu starten, der den konsistenten Zugriff auf die Prozessdaten durchführt. Dabei sind der Port portNo der IO-Task, die Zykluszeit time des Timers in ms sowie die Länge des Eingangs- bzw. Ausgangs-Prozessabbildes inpLength bzw. outpLength der IO-Task (entsprechend der bei der Definition der Variablen erzeugten Adressen) anzugeben. Innerhalb der Timer-Routine wird das Eingangs-Prozessabbild der IO-Task konsistent gelesen und lokal zwischengespeichert, so dass der Zugriff auf die Eingangs-Prozessdaten mit der Funktion ReadInputs sehr schnell geht. Das Ausgangs-Prozessabbild der IO-Task wird nur mit dem lokalen Prozessabbild konsistent aktualisiert, wenn durch einen Aufruf von WriteOutputs auf das lokale Prozessabbild zugegriffen wurde. Auch der Aufruf von WriteOutputs geht daher sehr schnell.

Rückgabewerte:

0: kein Fehler

-1: Timer konnte nicht gestartet werden

-2: AMS-Adresse konnte nicht gelesen werden

-3: nicht genügend Speicher für lokales Prozessabbild

-4: Timer läuft bereits

StartImageUpdateWithWd

long StartImageUpdateWithWd(int portNo, int time, int outpLength, int inpLength, int wdTime);

Die Funktion StartImageUpdateWithWd beinhaltet die gleiche Funktionalität wie StartImageUpdate und startet zusätzlich noch einen Watchdog. Der Watchdog wird nachgetriggert, wenn die Funktionen ReadInputs oder WriteOutputs aufgerufen werden. Ist das innerhalb der Watchdog-Time nicht der Fall, wird ein IO-Reset auf allen Devices durchgeführt (das führt zum Abschalten der Ausgänge) und die Outputs im Prozessabbild werden auf 0 gesetzt.

StopImageUpdate

long StopImageUpdate(void);

Die Funktion StopImageUpdate muss nur aufgerufen werden, wenn beim Beenden der Anwendung die DLL nicht automatisch entladen wird (ist bei z.B. bei Anwendung auf LabView-CVI-Basis der Fall).

Rückgabewerte:

0: kein Fehler

-5: DLL nicht mehr aktiv

ReadInputs

long ReadInputs(int offset, int length, unsigned char * pData);

Mit der Funktion ReadInputs wird das lokale Eingangs-Prozessabbild bzw. Teile davon gelesen. Dabei sind der offset innerhalb des Eingangs-Prozessabbildes der IO-Task, die Länge length der zu lesenden Daten und ein Pointer pData auf einen Speicherbereich, in den die Eingangsdaten kopiert werden können, übergeben werden.

Rückgabewerte:

0: kein Fehler

-1: Timer läuft nicht

-2: Offset ist zu groß

-5: DLL nicht mehr aktiv

WriteOutputs

long WriteOutputs(int offset, int length, unsigned char * pData);

Mit der Funktion WriteInputs wird das lokale Ausgangs-Prozessabbild bzw. Teile davon beschrieben. Dabei sind der offset innerhalb des Ausgangs-Prozessabbildes der IO-Task, die Länge length der zu lesenden Daten und ein Pointer pData auf einen Speicherbereich auf die Ausgangsdaten, übergeben werden.

Rückgabewerte:

0: kein Fehler

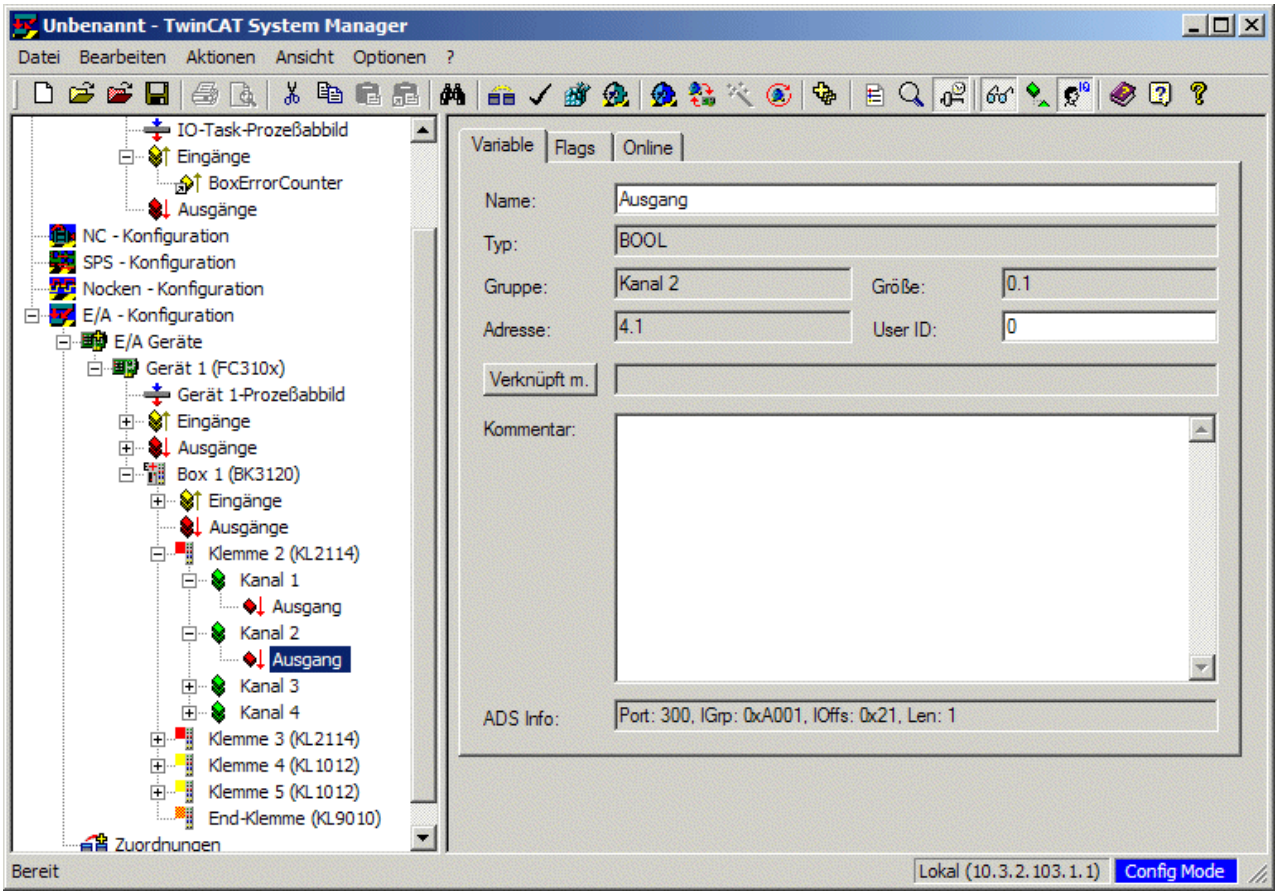
-1: Timer läuft nicht

-2: Offset ist zu groß

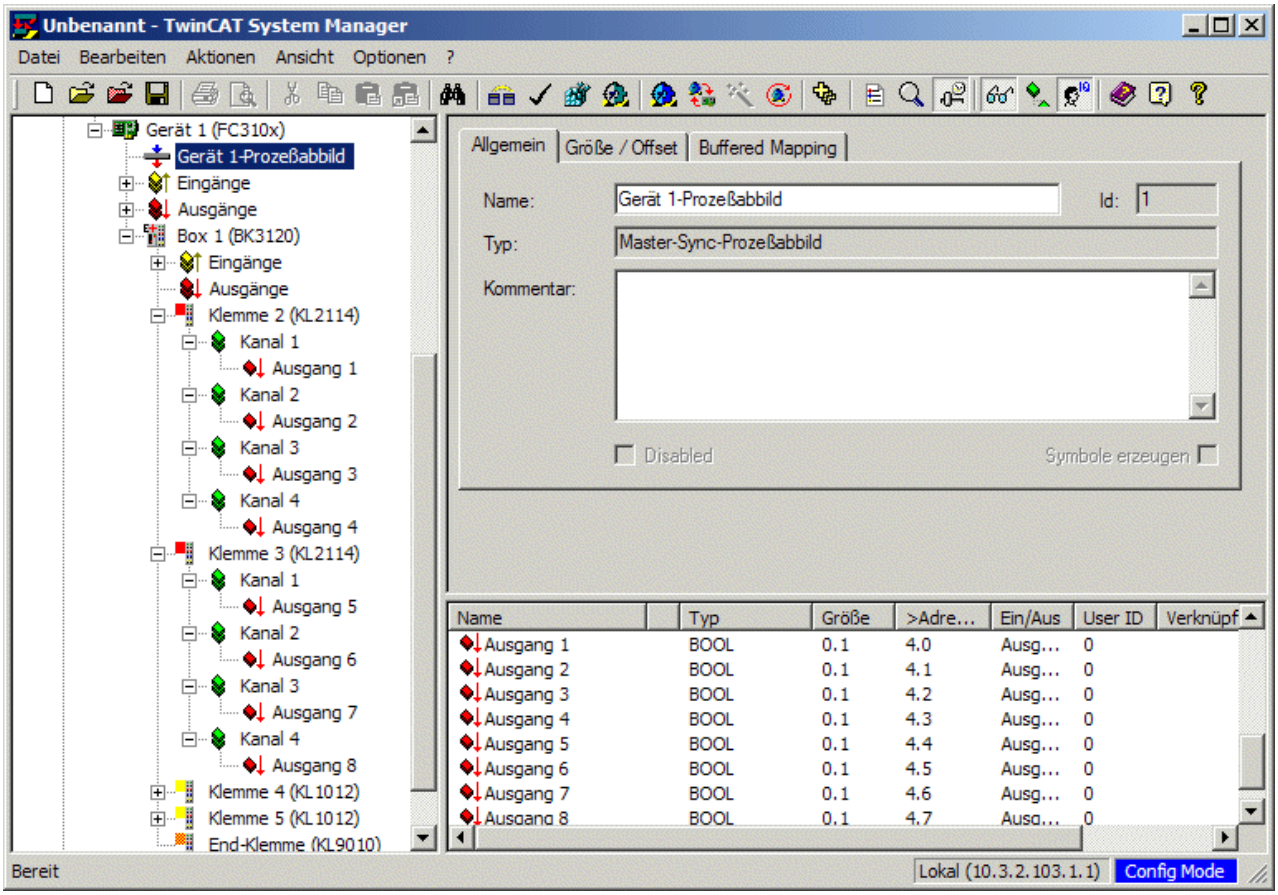
-5: DLL nicht mehr aktiv

8.3 Direkter Zugriff auf DP-RAM

Alternativ zum konsistenten Zugriff über die IO-Task [▶ 49] kann direkt auf das DP-RAM zugegriffen werden. Dabei ist nur WORD-Konsistenz möglich, dafür ist er maximal schnell, die Totzeit (Alter einer gelesenen Variable bzw. Verzögerungszeit bis eine geschriebene Variable auf dem Feldbus gesendet wird) ist maximal so groß wie die Zykluszeit der IO-Task. Es ist zu beachten, dass die Ausgangsvariablen des Feldbusses, die über direkten Zugriff beschrieben werden, nicht mit Variablen der IO- Task verknüpft sein dürfen, da diese sonst durch die IO-Task wieder überschrieben werden. Die Offsets der Zugriffsfunktionen beziehen sich auf die Variablenadressen des Feldbus Devices, d. h. die Adresse ist auf dem Karteireiter "Variable" bei der entsprechenden Variablen unter dem Feldbus Devices nachzuschauen (in dem abgebildeten Beispiel handelt es sich um die Adresse 4.1):

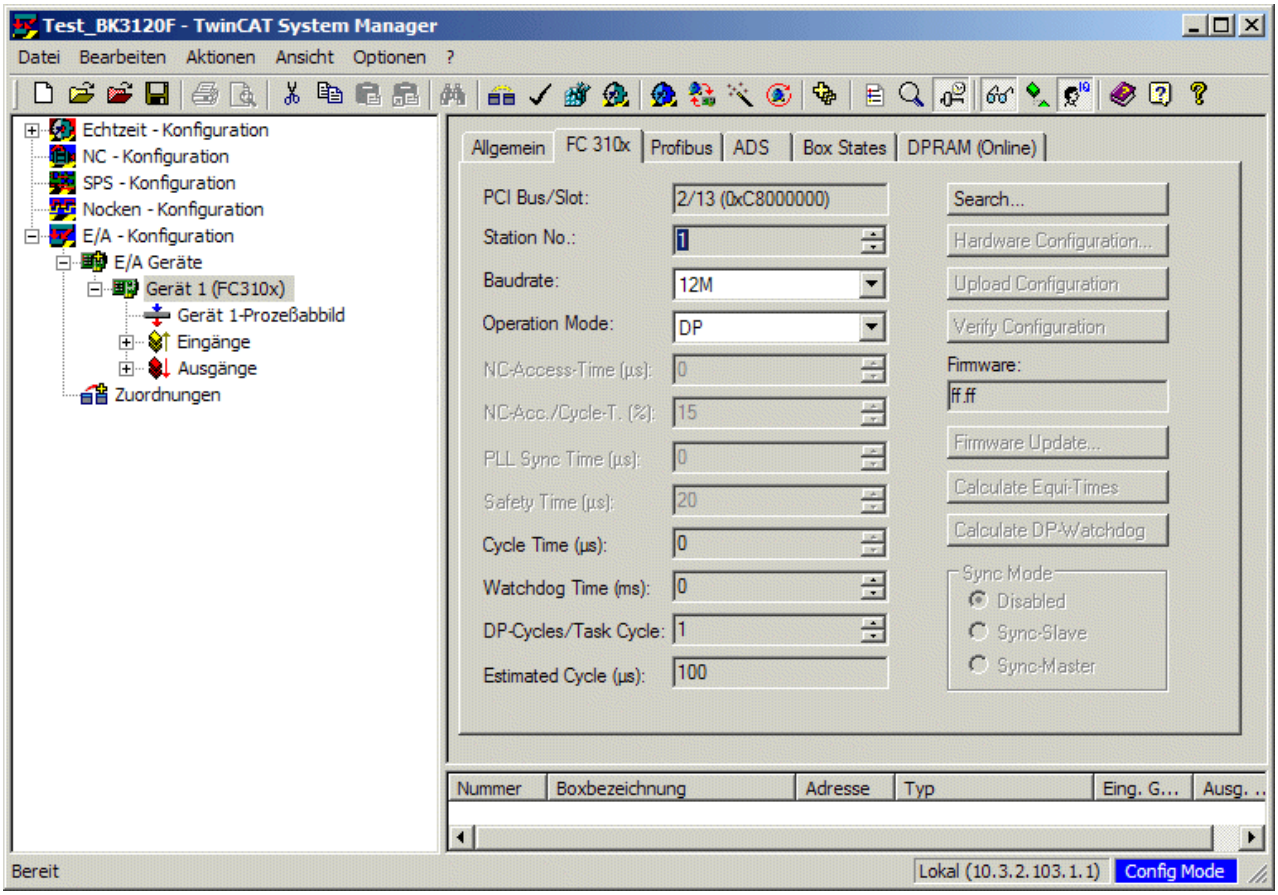


Bei den PROFIBUS-PC-Karten FC310x ist zu beachten, dass die 4 Bytes vor jeder ersten Eingangs- und Ausgangsvariable eines PROFIBUS-Slaves für den PROFIBUS-Protokollheader verwendet werden (daher ist der kleinste Offset einer Variablen an der FC310x auch 4), ein Beschreiben dieser jeweiligen 4 Bytes führt zu Störungen auf dem PROFIBUS. Daher sollte darauf geachtet werden, dass beim direkten Zugriff nur auf die Bereiche zugegriffen wird, in denen sich tatsächlich Variablen befinden, eine Übersicht kann man sich im rechten Fenster unten anzeigen lassen, wenn man im Baum das Prozessabbild des Devices anklickt:



Wenn man mit der rechten Maustaste auf die Liste klickt, kann diese auch gedruckt oder z.B. nach Excel kopiert werden.

Bei den Zugriffsfunktionen muss die DP-RAM-Adresse übergeben werden, die unter *PCI Bus/Slot* auf dem Karteireiter "FC310x", "FC510x" bzw. "FC520x" gelesen werden kann, wenn man im Baum auf das Feldbus Device klickt:



ReadInputsDirect

long _stdcall ReadInputsDirect(unsigned long dpRamAddress, int offset, int length, unsigned char * pData);

Mit der Funktion ReadInputsDirect werden Eingangsvariablen des Feldbus-Devices direkt aus dem DP-RAM gelesen, der Aufruf der Funktion geht sehr schnell (ca. je Wort ca. 1,5 µs). Dabei sind die DP-RAM-Adresse dpRamAddress des Feldbus-Devices, der offset und die Länge length der Eingangsvariablen im DP-RAM und ein Pointer pData auf einen Speicherbereich, in den die Eingangsdaten kopiert werden können, übergeben werden.

Rückgabewerte:

0: kein Fehler

-1: DP-RAM-Pointer konnte nicht alloziert werden

-2: Offset ist zu groß

-3: DP-RAM-Adresse anders als bei den vorherigen Aufrufen von ReadInputsDirect oder WriteOutputsDirect

-5: DLL nicht mehr aktiv

WriteOutputsDirect

long _stdcall WriteOutputsDirect(unsigned long dpRamAddress, int offset, int length, unsigned char * pData);

Mit der Funktion WriteOutputsDirect werden Ausgangsvariablen des Feldbus-Devices direkt in das DP-RAM geschrieben, der Aufruf der Funktion geht sehr schnell (ca. 1,5 µs je Bereich). Dabei sind die DP-RAM-Adresse dpRamAddress des Feldbus-Devices, der offset und die Länge length der Ausgangsvariablen im DP-RAM und ein Pointer pData auf die Ausgangsdaten, übergeben werden.

Rückgabewerte:

0: kein Fehler

-1: DP-RAM-Pointer konnte nicht alloziert werden

-2: Offset ist zu groß

-3: DP-RAM-Adresse anders als bei den vorherigen Aufrufen von ReadInputsDirect oder WriteOutputsDirect

-5: DLL nicht mehr aktiv

GetDirectInputPointer

void * GetDirectInputPointer(unsigned long dpRamAddress);

Mit der Funktion GetDirectInputPointer kann ein Pointer auf das Eingangsprozessabbild im DP-RAM geholt werden. Die Adressen der Eingangsvariablen müssen, wie oben beschrieben, aus dem System Manager entnommen werden. Dabei ist die DP-RAM-Adresse dpRamAddress des Feldbus Devices zu übergeben.

Rückgabewerte:

0: Fehler

!= 0: Pointer auf Eingangsvariablen im DP-RAM

GetDirectOutputPointer

void * GetDirectOutputPointer(unsigned long dpRamAddress);

Mit der Funktion GetDirectOutputPointer kann ein Pointer auf das Ausgangsprozessabbild im DP-RAM geholt werden. Die Adressen der Ausgangsvariablen müssen, wie oben beschrieben, aus dem System Manager entnommen werden. Dabei ist die DP-RAM-Adresse dpRamAddress des Feldbus Devices zu übergeben.

Rückgabewerte:

0: Fehler

!= 0: Pointer auf Eingangsvariablen im DP-RAM

Mehr Informationen:
www.beckhoff.de/tx1100

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