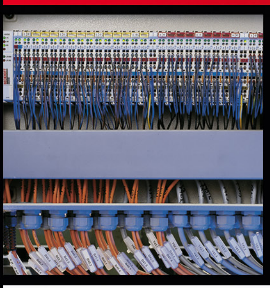
BECKHOFF New Automation Technology

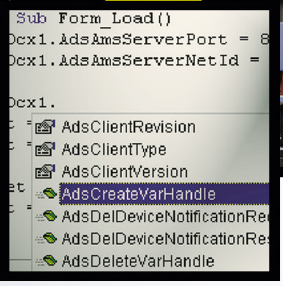
Manual | EN

TX1100

TwinCAT 2 | TwinCAT I/O

TwinCAT 2 | I/O





```
Sub Form_Load()
Dcx1.AdsAmsServerPort = 8
Dcx1.AdsAmsServerNetId =

Dcx1.
     AdsClientRevision
     AdsClientType
     AdsClientVersion
et   AdsCreateVarHandle
     AdsDelDeviceNotificationRe
     AdsDelDeviceNotificationRe
     AdsDeleteVarHandle
```

2022-09-28 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

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

Personnel qualification

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

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

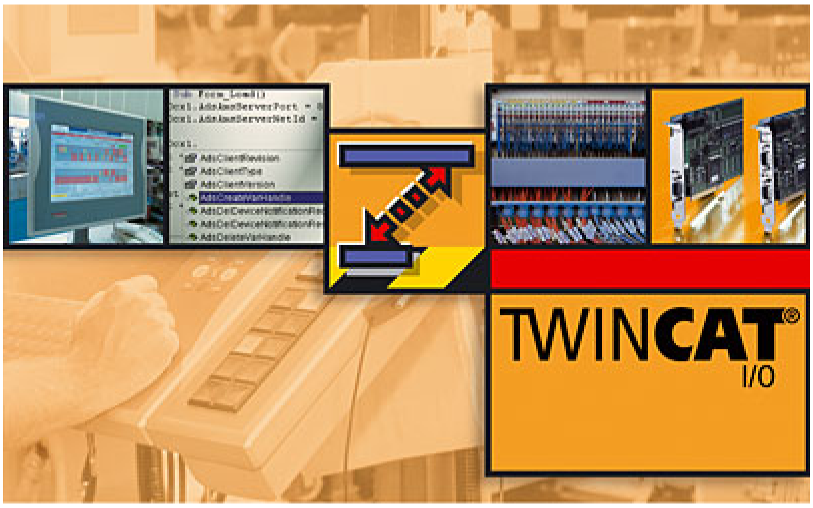
The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of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

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

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

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

2 Introduction



The basic idea behind the TwinCAT I/O interface is to grant Windows programs (Windows NT/2000/XP user mode and/or Windows CE application) access to the TwinCAT I/O subsystem without using ADS communication.
Therefore a user mode DLL is supplied which offers a set of interface functions to trigger the fieldbus I/O and to exchange the process data with the TwinCAT I/O subsystem.

The configuration is done in the standard TwinCAT way through the TwinCAT System Manager.

The interface DLL is optimized to access the process images as directly as possible. Through this interface a Windows NT user mode application is able to use all fieldbus types available for the TwinCAT system (Lightbus, Profibus, Interbus, CANOpen, DeviceNet, Ethernet,…).

There is an example for processing fieldbus I/O later in this document.

3 System Requirements

For Windows NT:

- TwinCAT 2.5 or later
- TCatIoDrv.dll
- TCatIoApi.h
- TCatIoDrv.lib

For Windows CE:

- Image Version 1.90 or above
- TwinCAT I/O Files : TCatIoW32Api.h, TCatIoDrvW32.lib
- TwinCAT Timer files (optional) : TcTimerAPI.h, TcTimerWrap.lib
- TwinCAT CE "*Run as Device*" option must be enabled. This can be done by using "**CxConfig Tool**" provided with the CE image.

The components https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082513675/.zip

4 TcTimer (CE) and TwinCAT IO

Target platforms

TcTimer functionality is available on BECKHOFF CE based devices (like CX1000 / CX1020 / CX9000 / Ethernet Panel/ IPC...)

https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082515083/.pdf

Functionality

TcTimer provides a deterministic timer allowing to execute e.g. customer C++ code instead of implementing logic within IEC61131-3 PLC languages.

The basic TcTimer is scalable from 100µs and is derived from highest CE proirity level. The TwinCAT-R3IO-API offers direct access via pointers to in/out-put image of TwinCAT IO task.

The required TwinCAT IO task can be configured remotely with TwinCAT System Manager. This tool helps to easy scan the IOs on connected fieldbusses and map the physical field IOs to logical IOs in IO-Task.

Data-consistency : After reading the input data from IO task, calculating and providing new data in the output image the C++ code can trigger the data exchange between IO task to mapped physical IOs : As a result data-consistency from logical C++ code down to physical IO level is provided.

Major differences between TcTimer and TwinCAT IO R3

General the two solutions **TcTimer** and **R3IO** are similar : Both solutions offer an API to access from C++ code the images from TwinCAT-IO-tasks.

The major differences are this :

- TcTimer functionallity **only supported by CE platform** (CX1000 / CX1020 / CX9000 / Ethernetpanels / IPCs...)
- R3IO functionality is supported on both platforms CE and XP/XPE

- TcTimer allows to execute C++ code in deterministic cycles, also data-exchange to fieldbus can be triggered out of this deterministic cycle
- R3IO allows to execute C++ code which is cyclic started by multimedia timer (not highly deterministic) to trigger data-exchange to fieldbus.
  As an alternative this data-exchange can be triggered deterministic by the IO-task itself, but in this case we recommend to access IO-Task with ADS.

Required Components

The components https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082513675/.zip

Priorities

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

The sample sets thread-prior to 26 and 27.

In general the user can set the thread-priority, see additional info about priorites from other threads :

- Device threads : 100-130 approx.- Beckhoff TwinCAT Timer thread : 1

- Microsoft MultiMedia Timer : 2

Depending on the requirements of your application it make sense to choose priority 3 - 64 for real time threads and priority 131 - 255 for non real time threads.

5 Function Reference

5.1 TCatIoOpen

The *TCatIoOpen* function opens a connection to the TwinCAT I/O Server. Before any I/O processing *TCatIoOpen* should be called.

```
LRESULT TCatIoOpen();
```

Parameters

None

Return Values

If the function fails, the return value is -1.

Remarks

TCatIoOpen allocates memory and necessary system resources

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoClose () [▶ 12]

Note the system requirements [▶ 9].

5.2 TCatIoClose

The *TCatIoClose* function closes a connection to the TwinCAT I/O Server. Before ending the application *TCatIoClose* should be called to avoid loss of system resources.

```
LRESULT TCatIoClose();
```

Parameters

None

Return Values

If the function fails, the return value is non zero.

Remarks

TCatIoClose does some cleanup and frees allocated memory and system resources.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoOpen () [▶ 12]

5.3 TCatIoInputUpdate

The *TCatIoInputUpdate* function initiates the input mapping of the specified task.

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

Parameters

nPort

Port id of the I/O task whose process image should be used to get the input data. For further information on defining the task process image see the TwinCAT System Manager documentation, chapter "Additional tasks".

Return Values

If the function fails, the return value is nonzero. Possible error codes:

-1: I/O connection is not initialized

IOERR_IOSTATEBUSY [0x2075]: I/O device is not ready

Remarks

TCatIoInputUpdate checks the state of the input device whether it is ready or not. No fieldbus I/O is initiated by calling *TCatIoInputUpdate*, to start the field bus I/O cycle call TCatIoOutputUpdate [▶ 14].

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.

Import Library: Use TCatIoDrvW32.lib.

See Also

TCatIoOutputUpdate () [▶ 14]

5.4 TCatIoOutputUpdate

The *TCatIoOutputUpdate* function initiates the output mapping of the specified task.

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

Parameters

nPort

Port id of the I/O task whose process image should be used to transfer the output data. For further information on defining the task process image, see the TwinCAT System Manager - Addtional Tasks.

Return Values

If the function fails, the return value is non zero. Possible error codes:

-1: I/O connection is not initialized

IOERR_IOSTATEBUSY [0x2075]: I/O device is not ready

Remarks

TCatIoOutputUpdate checks the state of the I/O device whether it is ready or not. If the I/O device is ready, *TCatIoOutputUpdate* writes the output data to the device and starts the fieldbus I/O cycle.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoInputUpdate() [▶ 13]

5.5 TCatIoGetOutputPtr

The *TCatIoGetOutputPtr* function allocates an output buffer for the outgoing process image.

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

Parameters

nPort

Port id of the I/O task whose process image should be used to transfer the output data. For further information on defining the task process image see the TwinCAT System Manager documentation, chapter "Additional tasks".

ppOutp

Address of the pointer to get the address of the output buffer. If *TCatIoGetOutputPtr* succeeds, the pointer is initialized to the address of the output buffer.

nSize

Number of bytes for the requested process image buffer.

Return Values

If the function fails, the return value is non zero.

Remarks

TCatIoGetOutputPtr allocates a user buffer for the process image of the specified task and returns the address in *ppOutp*. If the buffer was already allocated, *TCatIoGetOutputPtr* returns the address of the previously allocated buffer. The output data will be transferred through this buffer. If TwinCAT is stopped or restarted while the user mode process is running the output address remains valid, although the I/O transfer is temporarily stopped. In case of a TwinCAT Restart, the user mode process can resume the execution without any extra calculation.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoGetInputPtr () [▶ 16]

5.6 TCatIoGetInputPtr

The *TCatIoGetInputPtr* function allocates an input buffer for the incoming process image.

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

Parameters

nPort

Port id of the I/O task whose process image should be used to transfer the output data. For further information on defining the task process image see the TwinCAT System Manager documentation, chapter "Additional tasks".

ppInp

Address of the pointer to get the address of the output buffer. If *TCatIoGetInputPtr* succeeds, the pointer is initialized to the address of the input buffer.

nSize

Number of bytes for the requested process image buffer.

Return Values

If the function fails, the return value is non zero.

Remarks

TCatIoGetInputPtr allocates a user buffer for the process image of the specified task and returns the address in *ppInp*. If the buffer was already allocated, *TCatIoGetInputPtr* returns the address of the previously allocated buffer. The input data will be transferred through this buffer. If TwinCAT is stopped or restarted while the user mode process is running the input address remains valid, although the I/O transfer is temporary stopped. In case of a TwinCAT Restart, the user mode process can resume the execution without any extra calculation.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoGetOutputPtr () [▶ 15]

5.7 TCatIoReset

The *TCatIoReset* function resets all available I/O devices in the TwinCAT System.

```
LRESULT TCatIoReset();
```

Parameters

None

Return Values

- Win32: If the function fails, the return value is non zero.
- Win CE: True (1) indicates succes, FALSE (0) indicates failure.

Remarks

TCatIoReset initiates a reset in the I/O device(s) through a *AdsWriteControl* request. Therefore *TCatIoReset* can succeed only if the TwinCAT System is in run mode.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

5.8 TCatIoGetCpuTime

The *TCatIoGetCpuTime* function returns the current CPU time.

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

Parameters

pnTime

Address of a 64 bit integer variable to receive the current CPU time in units of 100 Ns.

Return Values

If the function fails, the return value is non zero.

Remarks

TCatIoGetCpuTime reads the counter value of the Pentium CPU and scales it to a time value in units of 100 Ns.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

TCatIoGetCpuCounter () [▶ 18]

5.9 TCatIoGetCpuCounter

The TCatIoGetCpuCounter function returns the current CPU counter value.

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

Parameters

pnCount

Address of a 64 bit integer variable to receive the current CPU counter value.

Return Values

If the function fails, the return value is non zero.

Remarks

TCatIoGetCpuCounter reads the counter value of the Pentium CPU.

QuickInfo

For Windows NT:

- **Windows NT Version:** Requires version 4.0 or later.
- **TwinCAT:** Requires version 2.5 or later.
- **Header:** Declared in TCatIoApi.h.
- **Import Library:** Use TCatIoDrv.lib.

For Windows CE:

- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

See Also

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

The TCatGetState function returns the current ADS - State of TwinCAT System Service.

```
LRESULT TCatGetState();
```

Return Values

Current AdsState of TwinCAT System Service. If the function fails, the return value is ADSSTATE_ERROR.

QuickInfo

For Windows CE:

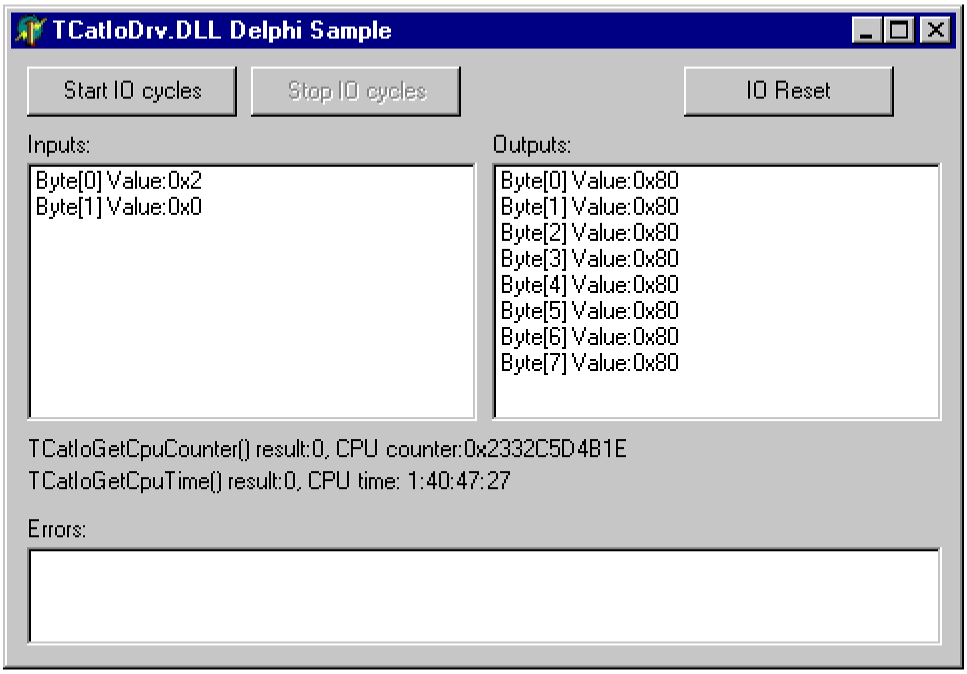
- **Windows CE Version:** Requires version 4.2 or later.
- **Image Version:** Requires version 1.90 or later.
- **Header:** Declared in TCatIoW32Api.h.
- **Import Library:** Use TCatIoDrvW32.lib.

7 Samples

7.1 TwinCAT I/0 Ring 3 DLL: Delphi Application

For this example, the functions of the TcatIoDrv DLL are ported to Pascal and assembled in a unit named **TCatIoDrv.pas**.

The source for this example can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082516491/.exe



```
Byte[0] Value:0x2
Byte[1] Value:0x0
```

```
Byte[0] Value:0x80
Byte[1] Value:0x80
Byte[2] Value:0x80
Byte[3] Value:0x80
Byte[4] Value:0x80
Byte[5] Value:0x80
Byte[6] Value:0x80
Byte[7] Value:0x80
```

```
TCatIoGetCpuCounter() result:0, CPU counter:0x2332C5D4B1E
TCatIoGetCpuTime() result:0, CPU time: 1:40:47:27
```

System requirements:

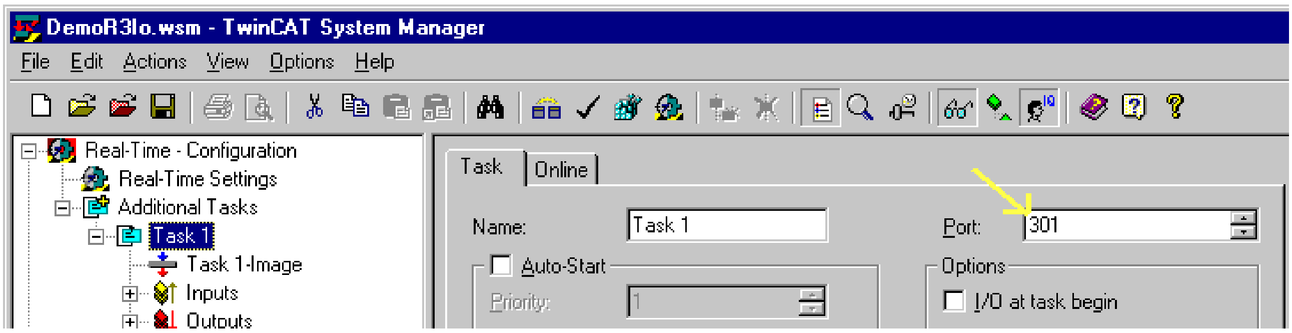
- TwinCAT 2.5 or higher
- TCatIoDrv.Dll
- Delphi 5.0

Description

The mapping of the process image of the inputs and outputs of an additional task in the TwinCAT System Manager is to be triggered from the Delphi application. The cycle time is 100 ms, and is generated with the aid of a multi-media timer. The online values of the process images are displayed in two list boxes. The TwinCAT I/O devices can be reset by means of the *I/O Reset* button. Mapping the inputs and outputs can be started or stopped using the *Start I/O cycle* and *Stop I/O cycle* buttons. If there are any error messages they are output in another list box.

Task Configuration in the TwinCAT System Manager

Port number 301 is configured for the additional task in the TwinCAT System Manager.


DemoR3Io.wsm - TwinCAT System Manager
File Edit Actions View Options Help
Real-Time - Configuration
Real-Time Settings
Additional Tasks
Task 1
Task 1-Image
Inputs
Outputs
Task
Online

Name: Task 1

Port: 301

□ Auto-Start

Priority: 1

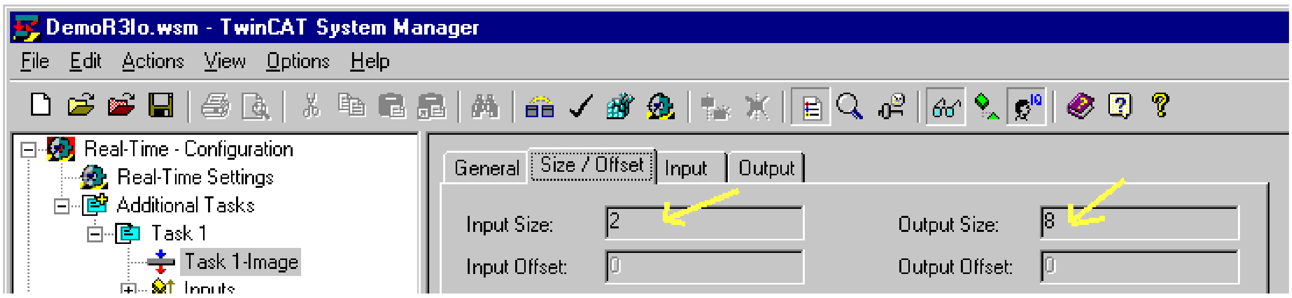
Options

□ I/O at task begin

The process image has the following size:

Input image: 2 bytes;

Output image: 8 bytes;


DemoR3Io.wsm - TwinCAT System Manager
File Edit Actions View Options Help
Real-Time - Configuration
Real-Time Settings
Additional Tasks
Task 1
Task 1-Image
Inputs
General
Size / Offset
Input
Output

Input Size: 2

Output Size: 8

Input Offset: 0

Output Offset: 0

The port number and the byte size of the process image are defined as constants in the **TCatIoDrv.pas** unit, and must be appropriately changed if they have any other value.

Linking the TCatIoDrv unit

The declarations of the TCatIoDrv.DLL functions used are found in the **TCatIoDrv.pas** Pascal unit. This is linked into the project by means of a "uses" clause.

```
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.
```

The Application

The event function *FormCreate* calls the DLL function TcatIoOpen [▶ 12]. If successful, the function returns a null, and a connection to the TwinCAT I/O sub-system is established. The user is informed of any errors in the list box. When the application is closed, the connection to the TwinCAT I/O sub-system must be removed. This is done in the *FormDestroy* event function, which calls the DLL function TCatIotClose [▶ 12]. Once the connection has been successfully established, two other functions are called: TCatIoGetInputPtr [▶ 16] and TCatIoGetoutputPtr [▶ 15]. These functions return pointers to the process images for the inputs and outputs. These pointers can be used to read the input process data or to write to the outputs.

```
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;
```

The *Timer1Timer* event routine is called cyclically when the multimedia timer is active. The following functions are called every time by this routine:

- TCatIoInputUpdate [▶ 13] (DLL function: triggers an update of the process image for the inputs (inputs are read));
- ViewData (auxiliary procedure: displays the current values of the inputs in a list box);
- CalculateNewOutputs (auxiliary procedure: generates/alters the values of the outputs (e.g. the running light));
- TCatIoOutputUpdate [▶ 14] (DLL function: triggers an update of the process image for the outputs (outputs are written));
- ViewData (auxiliary procedure: displays the current values of the outputs in a list box);
- TCatIoGetCpuCounter [▶ 18] (DLL function: reads the current value of the CPU counter);
- TCatIoGetCpuTime [▶ 17] (DLL function: reads the counter state of the 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;
```

The routines for activating/deactivating the timer:

```
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;
```

The routine for the I/O reset:

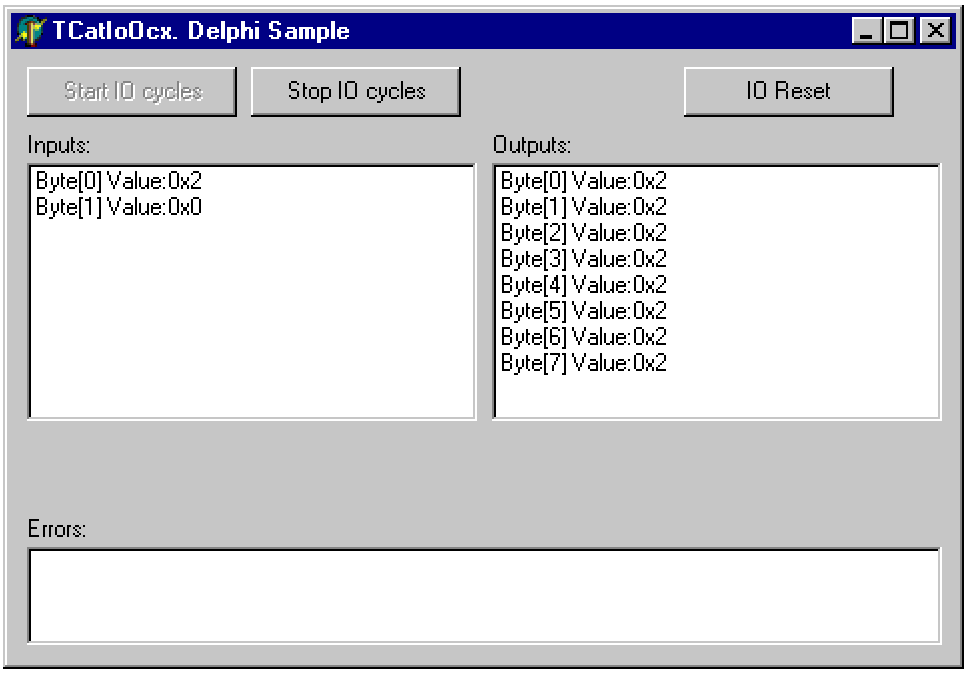
```
procedure TForm1.Button1Click(Sender: TObject);
var Result : integer;
begin
    Result := TCatIoReset();
    if ( Result <> 0 ) Then
    ListBox1.Items.Insert( 0, Format('TCatIoReset() error:%d', [ Result ] ) );
end;
```

```
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.
```

The source for this example can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082516491/.exe.

7.2 TwinCAT I/O Ring 3 OCX Delphi Application.

The source for this example can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082517899/.exe



```
Byte[0] Value:0x2
Byte[1] Value:0x0
```

```
Byte[0] Value:0x2
Byte[1] Value:0x2
Byte[2] Value:0x2
Byte[3] Value:0x2
Byte[4] Value:0x2
Byte[5] Value:0x2
Byte[6] Value:0x2
Byte[7] Value:0x2
```

System requirements:

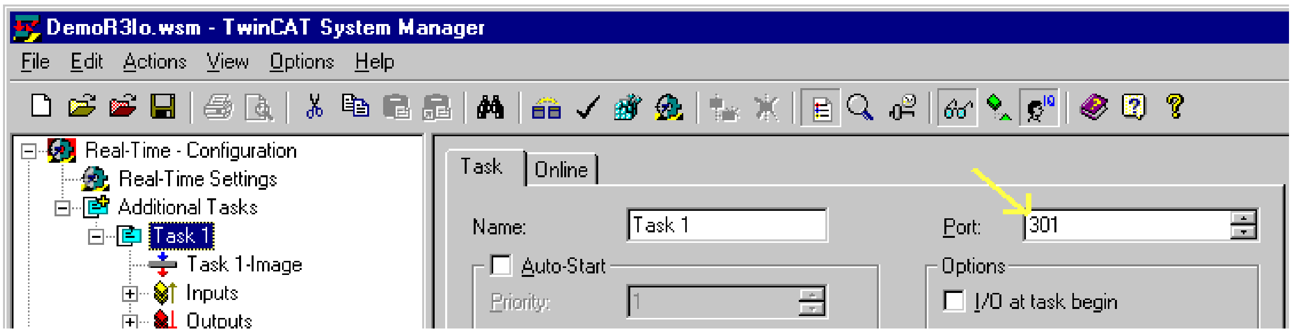
- TwinCAT 2.7 or higher
- TCatIoOcx.Ocx
- Delphi 5.0

Description

The mapping of the process image of the inputs and outputs of an additional task in the TwinCAT System Manager is to be triggered from the Delphi application. The cycle time is 100 ms, and is generated with the aid of a multi-media timer. The online values of the process images are displayed in two list boxes. The TwinCAT I/O devices can be reset by means of the *I/O Reset* button. Mapping the inputs and outputs can be started or stopped using the *Start I/O cycle* and *Stop I/O cycle* buttons. If there are any error messages they are output in another list box.

Task Configuration in the TwinCAT System Manager

Port number 301 is configured for the additional task in the TwinCAT System Manager.


DemoR3Io.wsm - TwinCAT System Manager

Name: Task 1

Port: 301

□ Auto-Start

Priority: 1

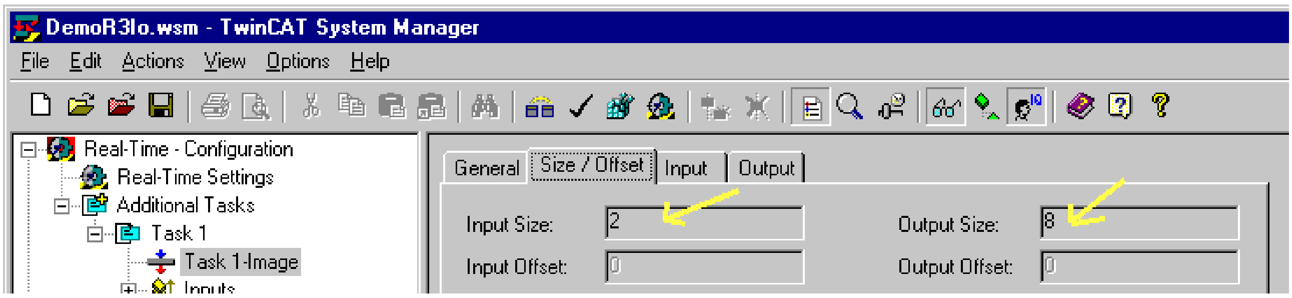
Options

□ I/O at task begin

The process image has the following size:

Input image: 2 bytes;

Output image: 8 bytes;


DemoR3Io.wsm - TwinCAT System Manager

Input Size: 2

Input Offset: 0

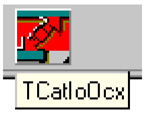
Output Size: 8

Output Offset: 0

The port number and the byte size of the process image are defined as constants in the *TCatIoOcxDelphiUnit.pas* unit, and must be appropriately changed if they have any other value.

Linking the TCatIoOcx ActiveX Components

In order to be able to use the TCatIoOcx ActiveX components in Delphi applications, it must be linked into the component palette. ActiveX components can be linked using the menu command: *Component->Import ActiveX Control ...* . Select the *TCatIoOcx ActiveX Control Module* from the list of installed components, and confirm with *Install...* Then confirm the dialogue window that follows. When successful, the TCatIoOcx will be found in the palette of ActiveX components.



The Application

The event function *FormCreate* calls the method *TCatIoOcxOpen*. If successful, the function returns a null, and a connection to the TwinCAT I/O sub-system is established. The user is informed of any errors in the list box. When the application is closed, the connection to the TwinCAT I/O sub-system must be removed. This is done in the *FormDestroy* event function, which calls the method *TCatIoOcxClose*. Once the connection has been successfully established, two other methods are called: *TCatIoOcxGetInputPtr* and *TCatIoOcxGetOutputPtr*. These methods return pointers to the process images for the inputs and outputs. These pointers can be used to obtain read access to the inputs and write access to the outputs. In our example, however, these pointers are not used to provide access to the process data. Two data buffers are used instead: InputImage and OutputImage. 4-byte alignment must be observed when defining the data buffers.

This means that 4 bytes must be reserved in the data buffer for every DWord of process data, whether complete or partial. In our example, the size of the data buffer for the inputs is 4 bytes (the actual size is 2 bytes) and for the outputs it is 12 bytes (the actual size is 8 bytes). The data buffers can be larger, but must not be smaller.

```
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;
```

The *Timer1Timer* event routine is called cyclically when the multimedia timer is active. The following methods are called by this routine every time:

- TCatIoOcxInputUpdate (the method triggers an update of the input process image (inputs are read));
- TCatIoOcxGetInputData (the method reads the process data from the inputs into the *InputImage* data buffer)
- ViewData (auxiliary procedure: displays the current values of the inputs in a list box);
- CalculateNewOutputs (auxiliary procedure: generates/alters the values of the outputs (e.g. the running light));
- TCatIoOcxSetOutputData (the method sets the output process data using the data from the *OutputImage* data buffer)
- TCatIoOcxOutputUpdate (the method triggers an update of the output processed image (outputs are written));
- ViewData (auxiliary procedure: displays the current values of the outputs in a list box);

```
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;
```

The routines for activating/deactivating the timer:

```
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;
```

The routine for the I/O 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.
```

The source for this example can be unpacked from here: https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082517899/.exe

7.3 C++ Sample

7.3.1 Configuration

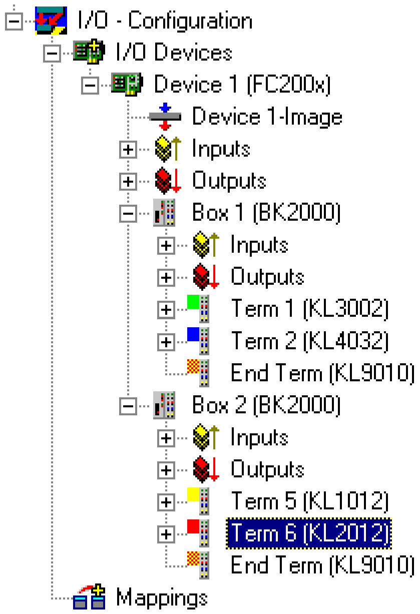
The TwinCAT System Manager is used for the configuration of the TwinCAT system.

For this sample you have to configure:

- The I/O device [▶ 33] (e.g. Beckhoff FC2001)
- Two I/O tasks [▶ 33]
- The process images [▶ 33] of that tasks
- The mapping [▶ 35] between the I/O tasks and the I/O device

7.3.1.1 The I/O Device

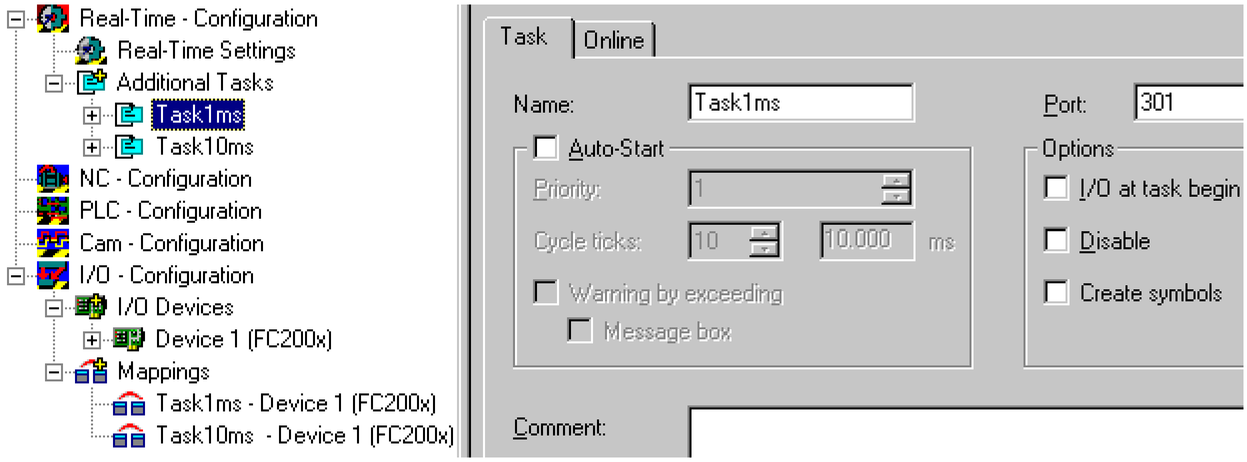
Add the device and the boxes as described under: TwinCAT System Manager - I/O Configuration.



- I/O - Configuration
  - I/O Devices
    - Device 1 (FC200x)
      - Device 1-Image
      - Inputs
      - Outputs
      - Box 1 (BK2000)
        - Inputs
        - Outputs
        - Term 1 (KL3002)
        - Term 2 (KL4032)
        - End Term (KL9010)
      - Box 2 (BK2000)
        - Inputs
        - Outputs
        - Term 5 (KL1012)
        - Term 6 (KL2012)
        - End Term (KL9010)
  - Mappings

7.3.1.2 The I/O Tasks

Now the I/O task(s) must be added. Up to 5 tasks are available. In the property page on the left side enter a port number from 301 (task 1) to 305 (task 5). Please ensure that the "Auto-Start" check box is left unchecked. In this example the real time tasks are not running, only their process images are used. Detailed information about *Additional Tasks* are available under TwinCAT System Manager - Addtional Tasks.



- Real-Time - Configuration
  - Real-Time Settings
  - Additional Tasks
    - Task1ms
    - Task10ms
- NC - Configuration
- PLC - Configuration
- Cam - Configuration
- I/O - Configuration
  - I/O Devices
    - Device 1 (FC200x)
  - Mappings
    - Task1ms - Device 1 (FC200x)
    - Task10ms - Device 1 (FC200x)

Task | Online

Name: Task1ms

Port: 301

□ Auto-Start

Priority: 1

Cycle ticks: 10 10.000 ms

□ Warning by exceeding

□ Message box

Options

□ I/O at task begin

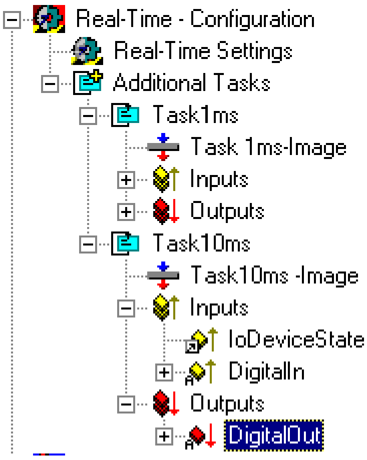
□ Disable

□ Create symbols

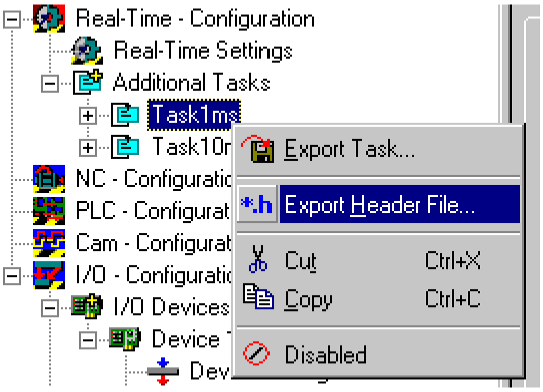
Comment:

7.3.1.3 The Process Image

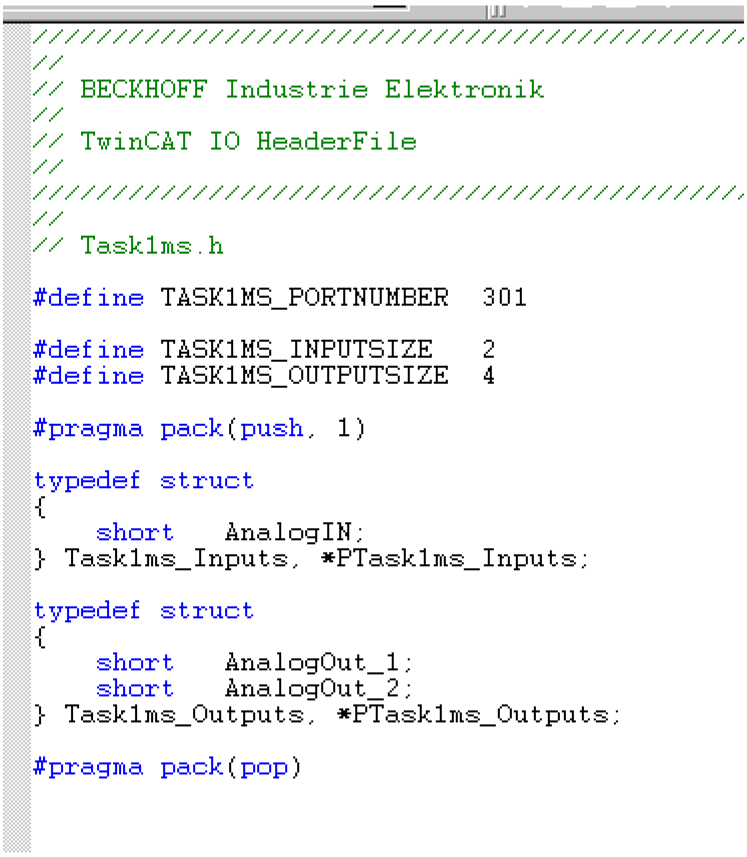
After adding the I/O tasks, the process image of each task must be configured. Please see the TwinCAT System Manager. documentation for details on configuring the I/O tasks.



To be sure, that the used data types of the task variables in the System Manager configuration is the appropriate one for the C++ application, the System Manager can generate a header file for each task. definition.



An example of such a header file is shown below:

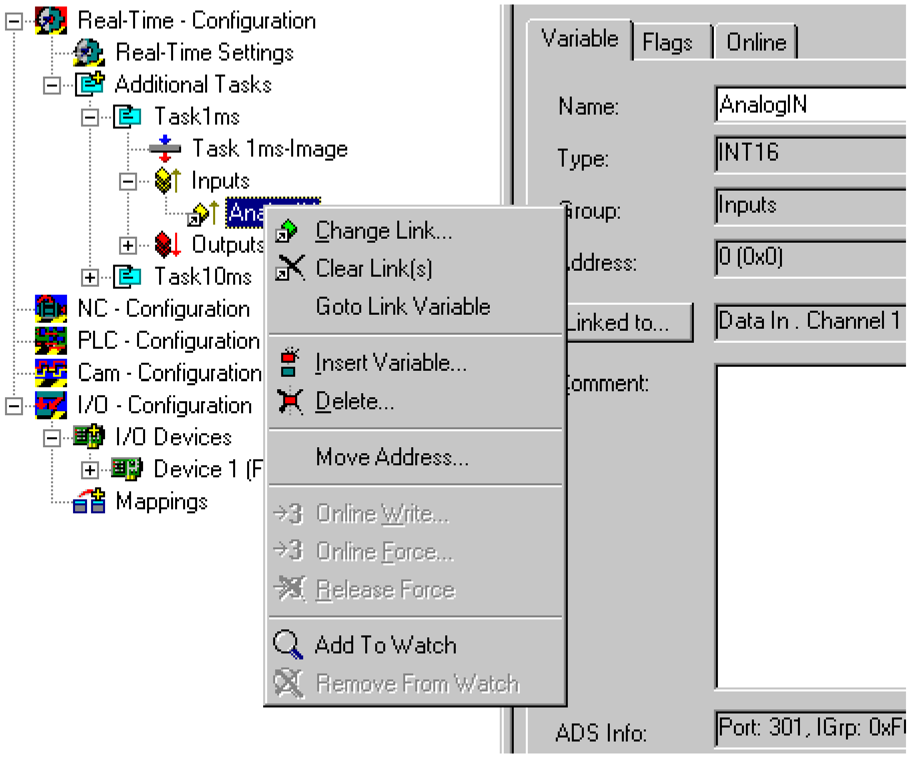


```
//////////////////////////////////////////////
//
// BECKHOFF Industrie Elektronik
//
// TwinCAT IO HeaderFile
//
//////////////////////////////////////////////
//
// Task1ms.h

#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)
```

The header file contains the ADS Port number of the task, the size of the input and output process image (in bytes) and the variables of the process image as structure members.

7.3.1.4 The Mappings

Link the variables with the corresponding fieldbus I/Os. Detailed information about mapping of variables is available under TwinCAT System Manager - Link Variables.



Name: AnalogIN
Type: INT16
Group: Inputs
Address: 0 (0x0)
Linked to... Data In . Channel 1
Comment:
ADS Info: Port: 301, IGrp: 0xF

After done configuration, save it to the registry and start / restart ...

... the TwinCAT system with this button.

7.3.2 Implementation

7.3.2.1 TwinCAT Sample Program

```
///////////////////////////////////////////////////////////////////////////////
//
// 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 TwinCAT Sample Program

https://infosys.beckhoff.com/content/1033/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;
}
```

Note the system requirements.

Also see about this

- System Requirements [▸ 9]

7.3.2.3 TcTimer Sample for VisualStudio 2005

https://infosys.beckhoff.com/content/1033/tcr3io/Resources/12082520715/.zip

- In VS2005 Create a new Console Application.
- Link your project to TcTimerWrap.lib and TCatIoDrvW32.lib
- Add the following code to the Application:

```
////////////////////////////////////////////////////////////////////////////////
//
// 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;
}
```

- If you receive a linker error when compiling make sure the compiler option "/Zc:wchar_t-" is set. (Goto Project properties -> Configuration Properties->C++->Language->Treat wchar_t as Built-in Type)

Debugging the sample Application

To debug the application perform the following steps :

- In the Vs2005 IDE select Tools -> Options -> Device Tools -> Devices.
- Chose your device and open the properties dialog.
- In the properties dialog select "TCP Connect Transport" and enter the ip address for your device
- On the Cx Device navigate to \Hard Disk\System
- Run the tools "conmanclient2" and "cmaccept"
- In the IDE select Tools -> Connect to Device.
- Start the Application.

8 TwinCAT IO and FCxxxx

TwinCAT IO with FC310x, FC510x and FC520x

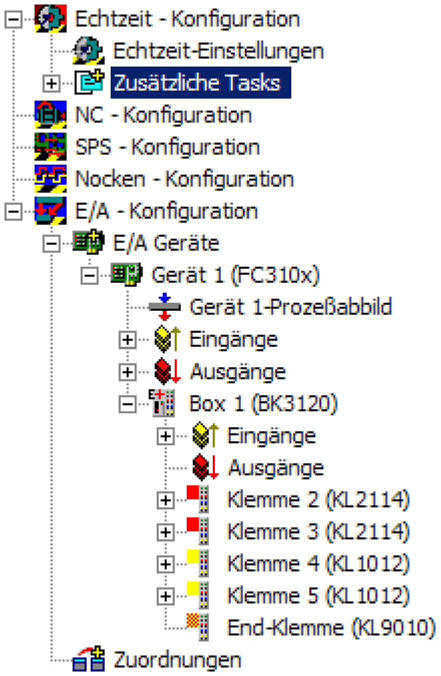
TwinCAT IO contains the device drivers and the configuration software (TwinCAT System Manager) for the FC310x, FC510x and FC520x PC cards under Windows NT, 2000 and XP.
The FcIoApi DLL provides the interfaces for freely selectable Windows applications for accessing the process data of these PC cards.

Regarding the configuration of the fieldbus via the TwinCAT System Manager please refer to the descriptions of the following PC cards: FC310x, FC510x, and FC520x, which also contain detailed descriptions of the respective diagnostics interfaces.

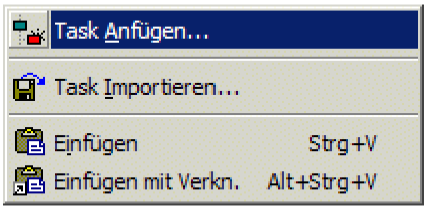
In addition to the fieldbus configuration, an IO task for re-triggering the associated fieldbus card has to be set up in the TwinCAT System Manager. The cycle time of the IO task corresponds to the fieldbus cycle time. To this end, at least one fieldbus device variable (FC310x, FC510x or FC520x) has to be linked with an IO task variable.

8.1 Setting up an IO task

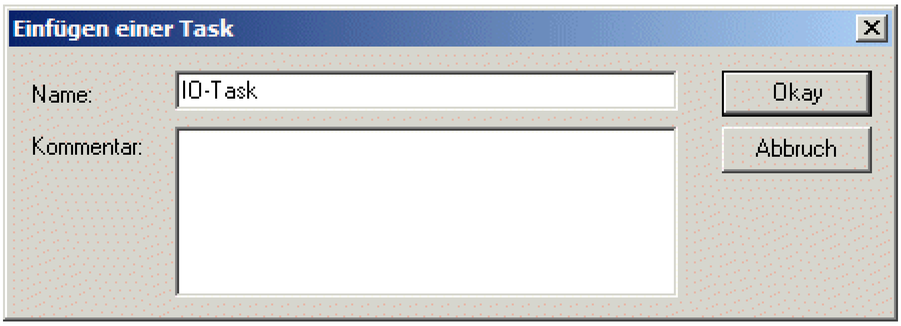
An IO task has to be set up in the TwinCAT System Manager tree under *Real-time configuration*:



Right-click on *Additional tasks* for adding a task (IO task):



The task name can be modified in the dialog that appears:

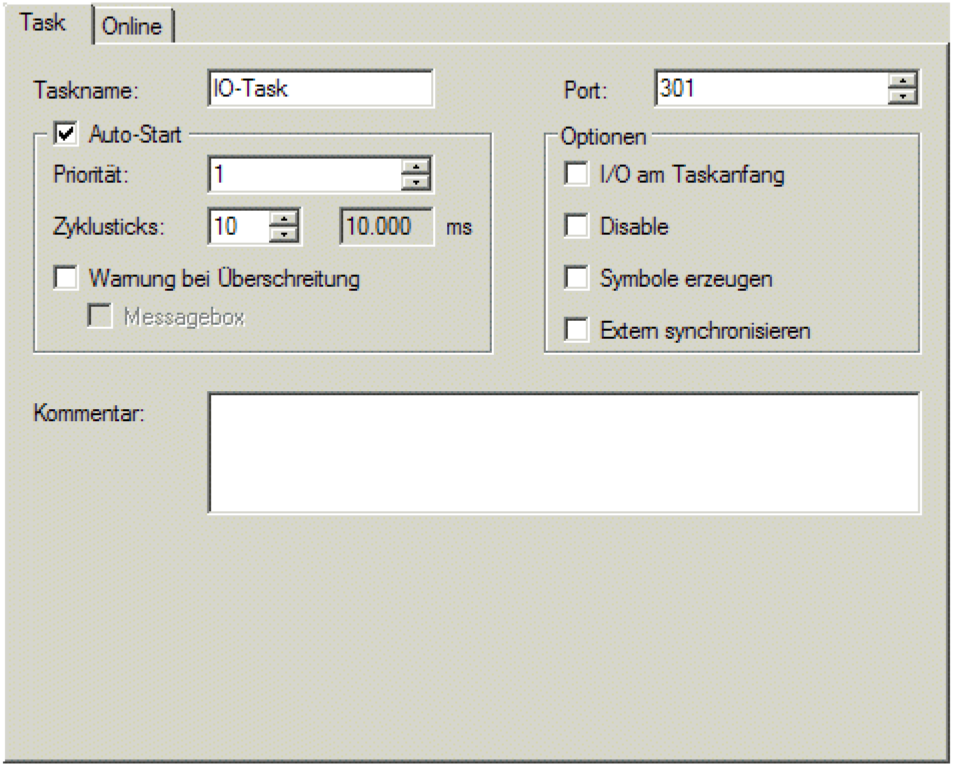

Einfügen einer Task

Name: IO-Task

Kommentar:

The IO task settings can now be adjusted in the right half of the window:

Click on the *Auto start* check box in order to adjust the fieldbus cycle time under *Cycle ticks*. The *port* for the FcIoApi DLL function calls is also required, all other parameters can remain unchanged.



Taskname: IO-Task

Port: 301

☑ Auto-Start

Priorität: 1

Zyklusticks: 10 10.000 ms

☐ Warnung bei Überschreitung

☐ Messagebox

Optionen

☐ I/O am Taskanfang

☐ Disable

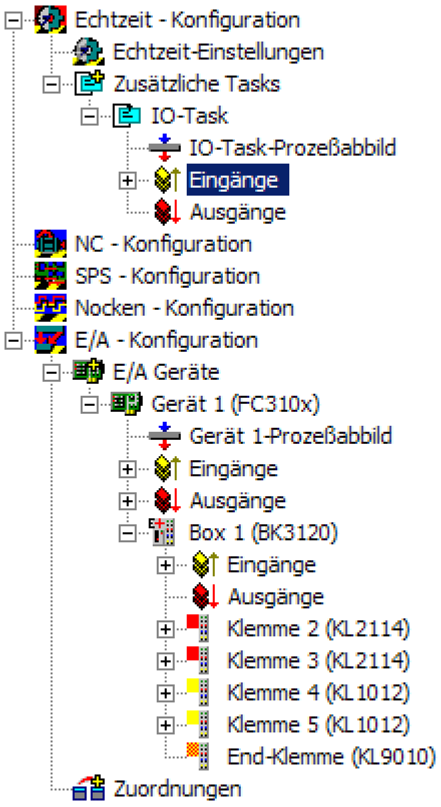
☐ Symbole erzeugen

☐ Extern synchronisieren

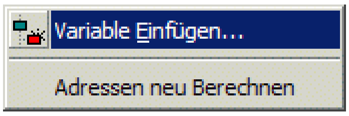
Kommentar:

Linking the IO task with the fieldbus device

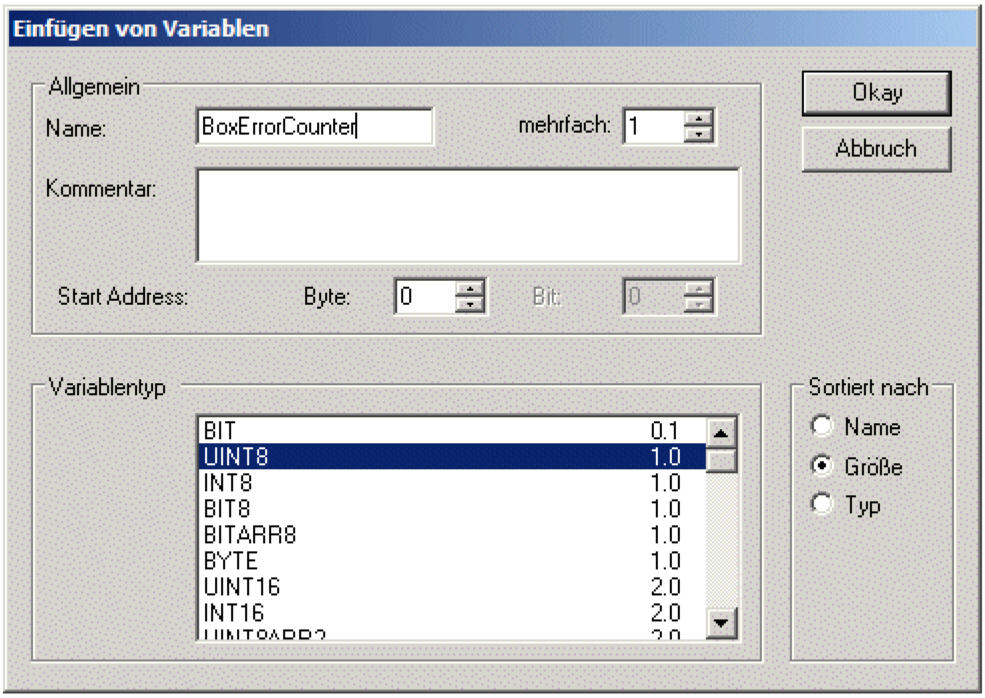
At least one IO task variable has to be linked with the fieldbus device. This is done via the IO task inputs in the tree:



Right-clicking brings up a pop-up menu through which a new variable can be appended:



In the variable dialog, the variable *name*, *start address* (address in the process image of the IO task), and *variable type* can be specified:


Einfügen von Variablen

Allgemein

Name: BoxErrorCounter mehrfach: 1

Kommentar:

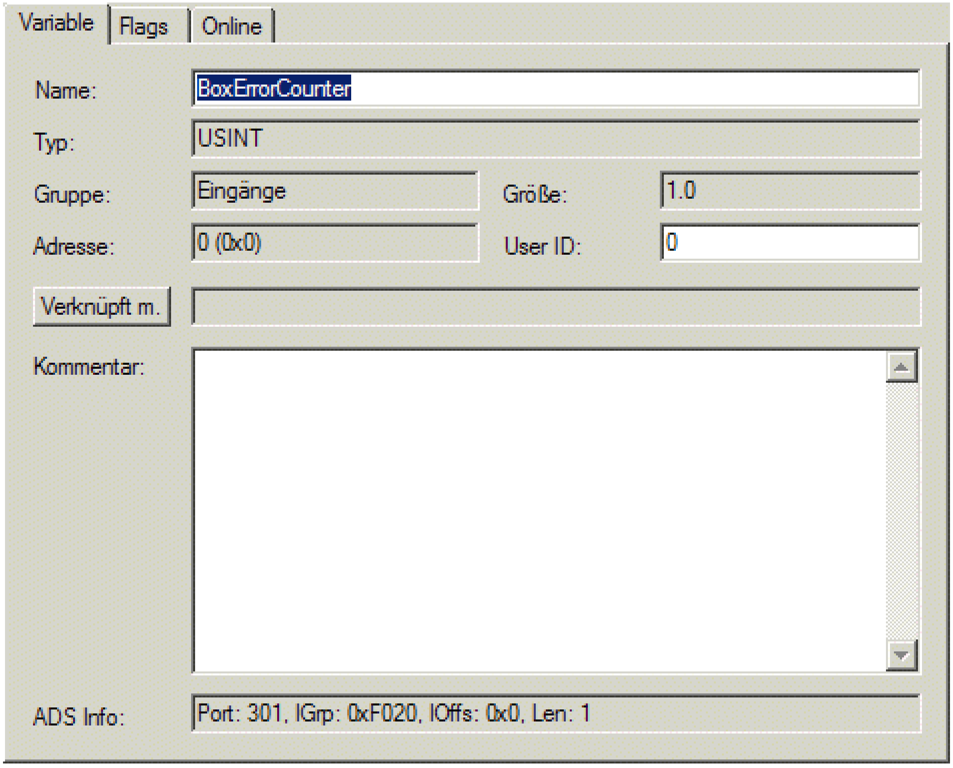
Start Address: Byte: 0 Bit: 0

Variablentyp

BIT	0.1
UINT8	1.0
INT8	1.0
BIT8	1.0
BITARR8	1.0
BYTE	1.0
UINT16	2.0
INT16	2.0
UINT8ARR2	2.0

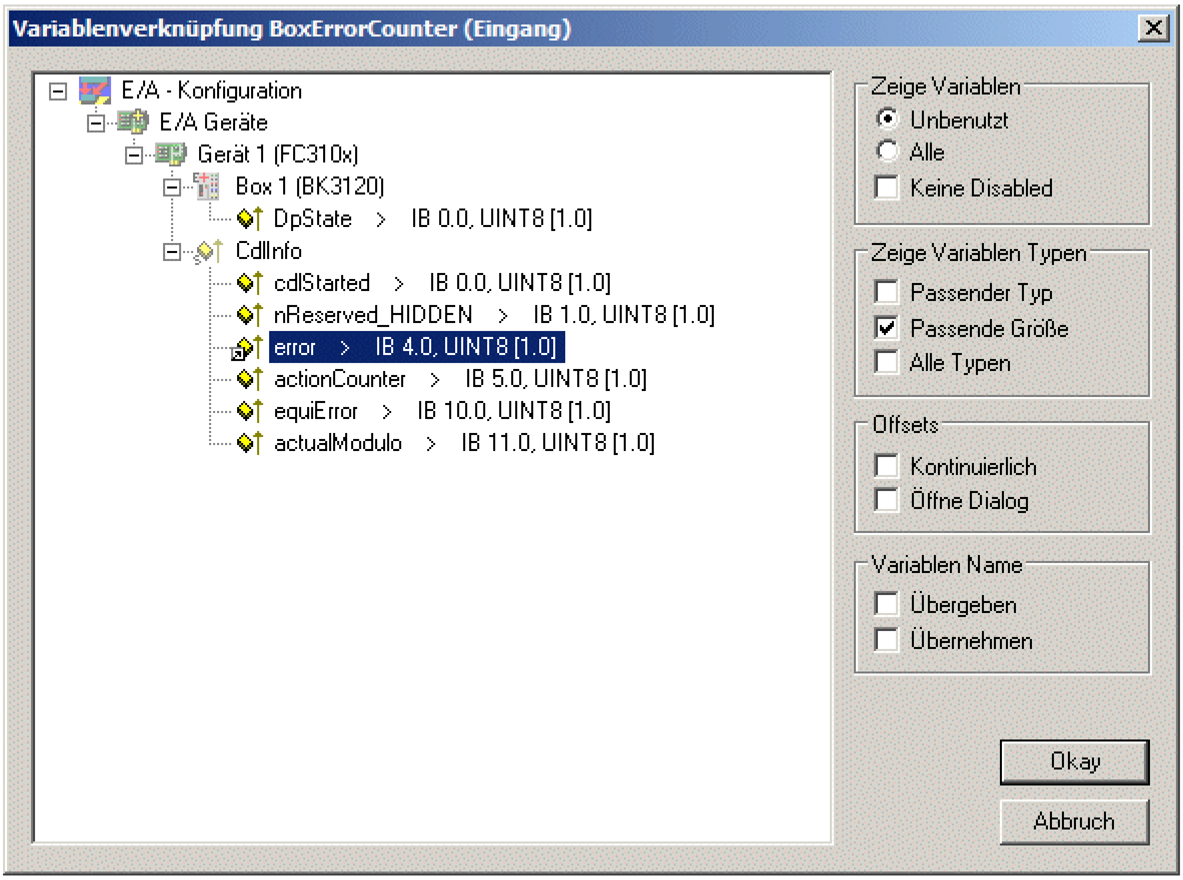
Sortiert nach
○ Name
● Größe
○ Typ

Now select the variable assigned to the fieldbus device by clicking on the *Linked w.* button in the "Variable" tab in the window on the right:



Name: BoxErrorCounter
Typ: USINT
Gruppe: Eingänge
Größe: 1.0
Adresse: 0 (0x0)
User ID: 0
Kommentar:
ADS Info: Port: 301, IGrp: 0xF020, IOffs: 0x0, Len: 1

The variables available for linking are now displayed in the "Variable link" dialog. Select the associated variable and confirm with *OK*:



Variablenverknüpfung BoxErrorCounter (Eingang)

```
E/A - Konfiguration
  E/A Geräte
    Gerät 1 (FC310x)
      Box 1 (BK3120)
        DpState  >  IB 0.0, UINT8 [1.0]
      CdlInfo
        cdlStarted  >  IB 0.0, UINT8 [1.0]
        nReserved_HIDDEN  >  IB 1.0, UINT8 [1.0]
        error  >  IB 4.0, UINT8 [1.0]
        actionCounter  >  IB 5.0, UINT8 [1.0]
        equiError  >  IB 10.0, UINT8 [1.0]
        actualModulo  >  IB 11.0, UINT8 [1.0]
```

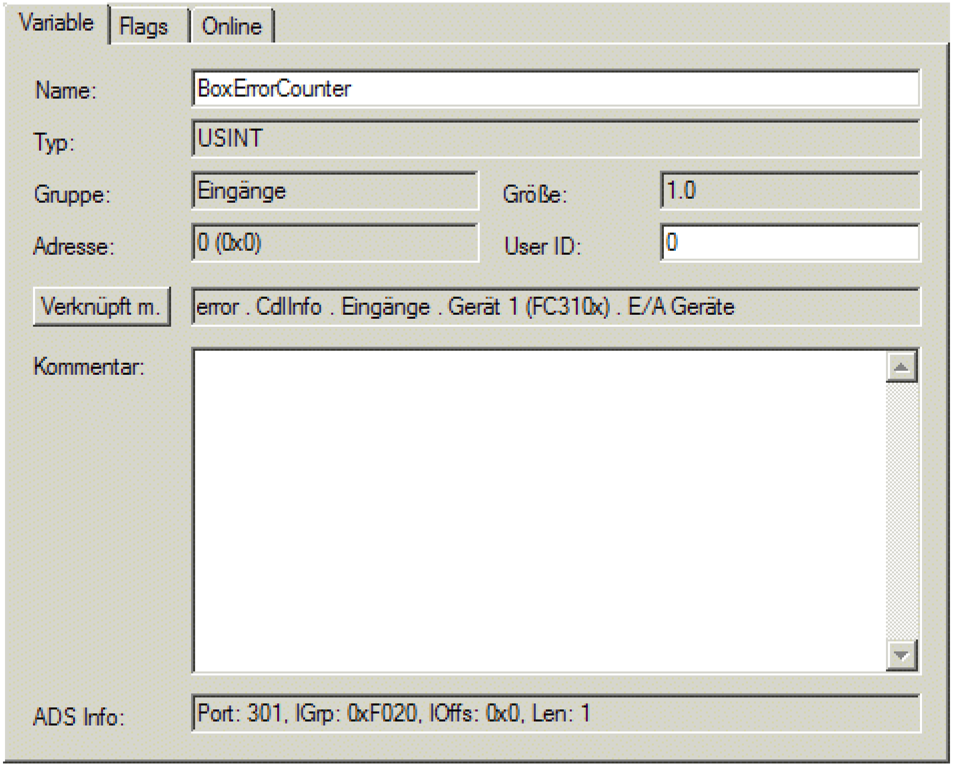
Zeige Variablen
● Unbenutzt
○ Alle
□ Keine Disabled

Zeige Variablen Typen
□ Passender Typ
☑ Passende Größe
□ Alle Typen

Offsets
□ Kontinuierlich
□ Öffne Dialog

Variablen Name
□ Übergeben
□ Übernehmen

The "Variable" tab now shows the link information.



Name: BoxErrorCounter
Typ: USINT
Gruppe: Eingänge
Größe: 1.0
Adresse: 0 (0x0)
User ID: 0
Verknüpft m.: error . CdlInfo . Eingänge . Gerät 1 (FC310x) . E/A Geräte
Kommentar:
ADS Info: Port: 301, IGrp: 0xF020, IOffs: 0x0, Len: 1

Starting the fieldbus

Finally, save the project in the registry (via the registry icon in the System Manager) and start TwinCAT (via the TwinCAT icon in the TwinCAT System Manager or in the Icon bar at the bottom right).

The fieldbus should now start up. The states can be diagnosed or outputs set via the System Manager.

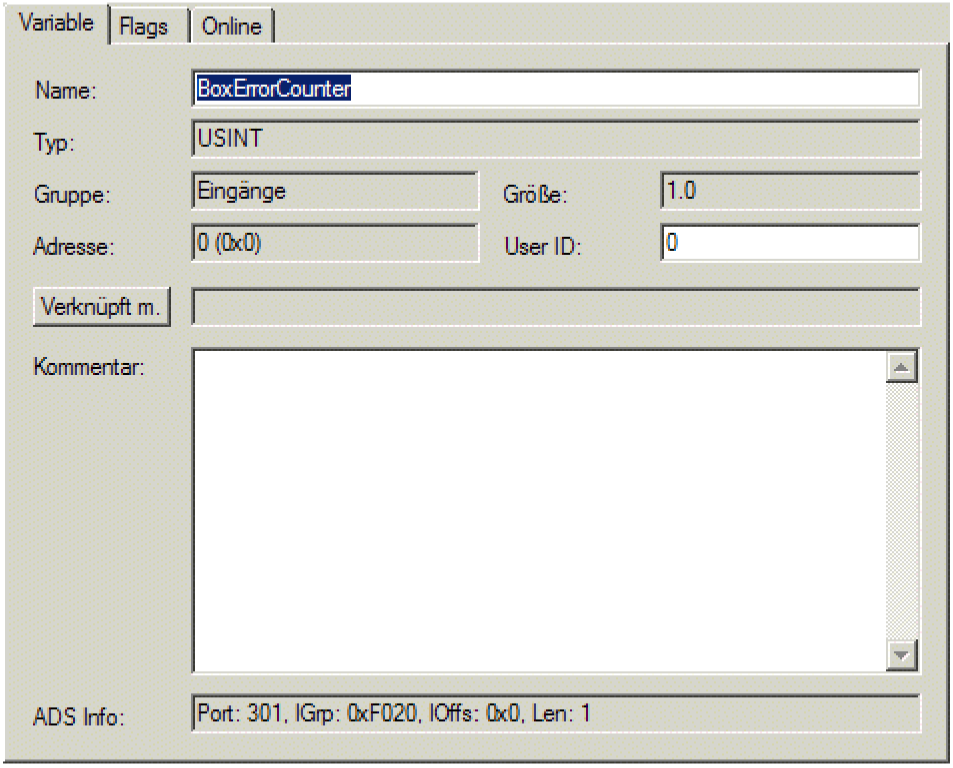
Documents

The functionalities of the Beckhoff PCI card FC310x (as Master and Slave) for use under TwinCAT (NCI, PTP, PLC and IO) is described below.

The following sections also apply to the PROFIBUS connection for the CX1000 (CX1500-M310 (master) or CX1500-B310 (slave)); the name FC310x then also refers to the CX1500-M310 master or CX1500-B310 slave connection.

8.2 Access to fieldbus process data

Consistent access via the IO task only works for fieldbus variables that are linked to the IO task. As shown above for an input variable configuration, for all fieldbus variables to be accessed consistently, associated IO task variables have to be created and linked with them. The access function offsets refer to the addresses of the IO task variables (the BoxErrorCounter variable is allocated to address 0 of the input process image of the IO task):



Name: BoxErrorCounter

Typ: USINT

Gruppe: Eingänge

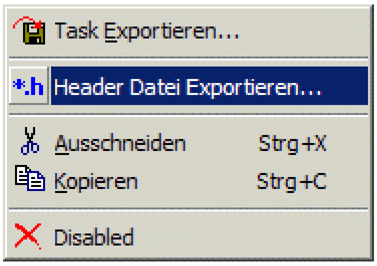
Größe: 1.0

Adresse: 0 (0x0)

User ID: 0

ADS Info: Port: 301, IGrp: 0xF020, IOffs: 0x0, Len: 1

The process image of the IO task can be exported in the form of an H file by right-clicking on the IO task in the tree:



StartImageUpdate

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

The StartImageUpdate function must be called once during start-up of the application for initiating the timer that ensures consistent access to the process data. The port portNo of the IO task, the time cycle time time in ms, and the length of the input or output process image inpLength or outpLength of the IO task (according to the addresses generated during variable definition) must be specified. Within the timer routine the input process image of the IO task is read consistently and held in the local buffer, enabling fast access to the input process data via the function ReadInputs. The output process image of the IO task is only consistently updated with the local process image, if the local process image was accessed by calling WriteOutputs. The WriteOutputs call is therefore very quick.

Return values:

0: no error

-1: Timer could not be started

-2: AMS address could not be read

-3: Not enough memory for local process image

-4: Timer already running

StartImageUpdateWithWd

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

The function StartImageUpdateWithWd has the same functionality as StartImageUpdate, although in addition it starts a watchdog. The watchdog is re-triggered when the functions ReadInputs or WriteOutputs are called. If this is not the case during the watchdog time, an IO reset is carried out for all devices (this leads to the outputs being disabled), and the outputs in the process image are set to 0.

StopImageUpdate

long StopImageUpdate(void);

The function StopImageUpdate only has to be called if the DLL is not automatically unloaded on termination of the application (e.g. for applications based on LabVIEW-CVI).

Return values:

0: no error

-5: DLL no longer active

ReadInputs

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

The function ReadInputs is used to read the local input process image or parts thereof. The following parameters are transferred: the offset within the input process image of the IO task, the length of the data to be read, and a pointer pData to a memory area into which the input data can be copied.

Return values:

0: no error

-1: Timer not running

-2: Offset too large

-5: DLL no longer active

WriteOutputs

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

The function WriteInputs is used to write to the local input process image or parts thereof. The following parameters are transferred: the offset within the output process image of the IO task, the length of the data to be read, and a pointer pData to a memory area into which the output data can be copied.

Return values:

0: no error

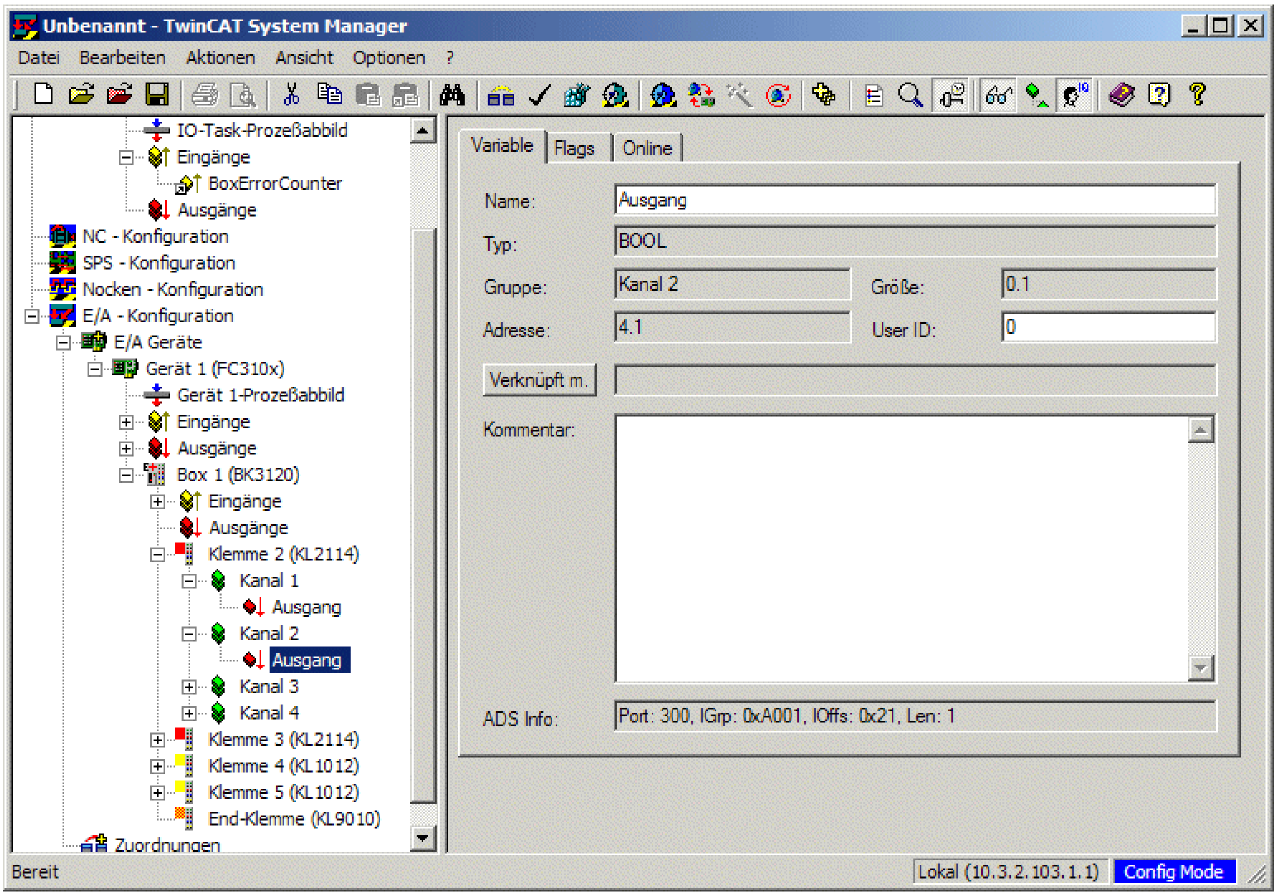
-1: Timer not running

-2: Offset too large

-5: DLL no longer active

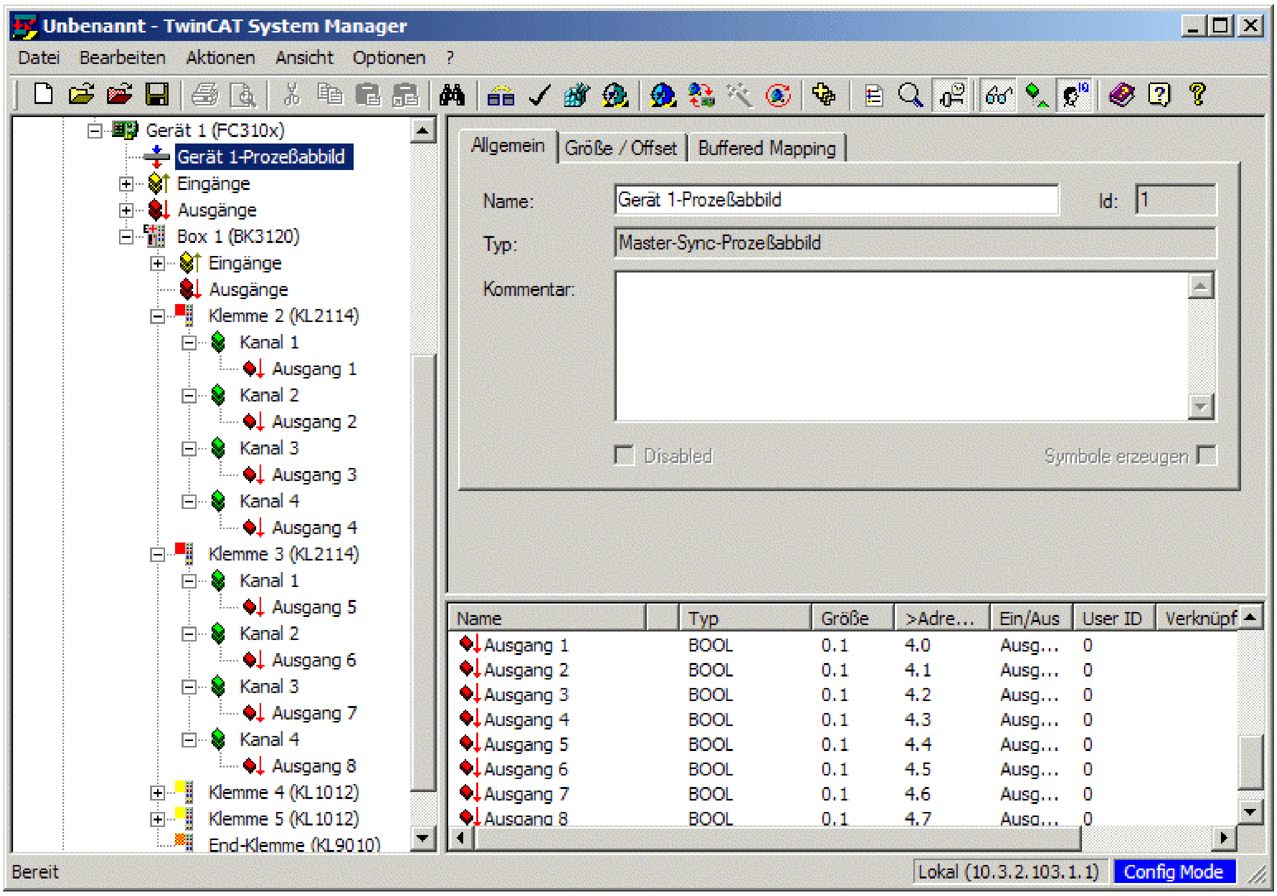
8.3 Direct DP-RAM access

As an alternative to consistent access via the IO task [▸ 47], direct access to the DP-RAM is also available. Only WORD consistency is possible, although access is very fast, and the maximum dead time (age of a read variable or delay time until a variable is sent to the fieldbus after writing) is the same as the cycle time of the IO task. Please note that that the fieldbus output variables that are written via direct access must not be linked with IO task variables, since they would be overwritten again by the IO task. The offsets of the access functions refer to the fieldbus device variable addresses, i.e. the address should be looked up under the "Variable" tab for the associated variable in the fieldbus devices (address 4.1 in the example shown):



Name: Ausgang
Typ: BOOL
Gruppe: Kanal 2 Größe: 0.1
Adresse: 4.1 User ID: 0
Kommentar:
ADS Info: Port: 300, IGrp: 0xA001, IOffs: 0x21, Len: 1

With FC310x PROFIBUS PC cards please note that the 4 bytes preceding the first input and output variable of a PROFIBUS slave are used for the PROFIBUS protocol header (i.e. the smallest variable offset for the FC310x is also 4). Writing of these respective 4 bytes would lead to PROFIBUS malfunction. Therefore, only those areas actually containing variables should be accessed directly. An overview can be displayed at the bottom of the window on the right by clicking on the process image of the device in the tree:



Name: Gerät 1-Prozeßabbild Id: 1

Typ: Master-Sync-Prozeßabbild

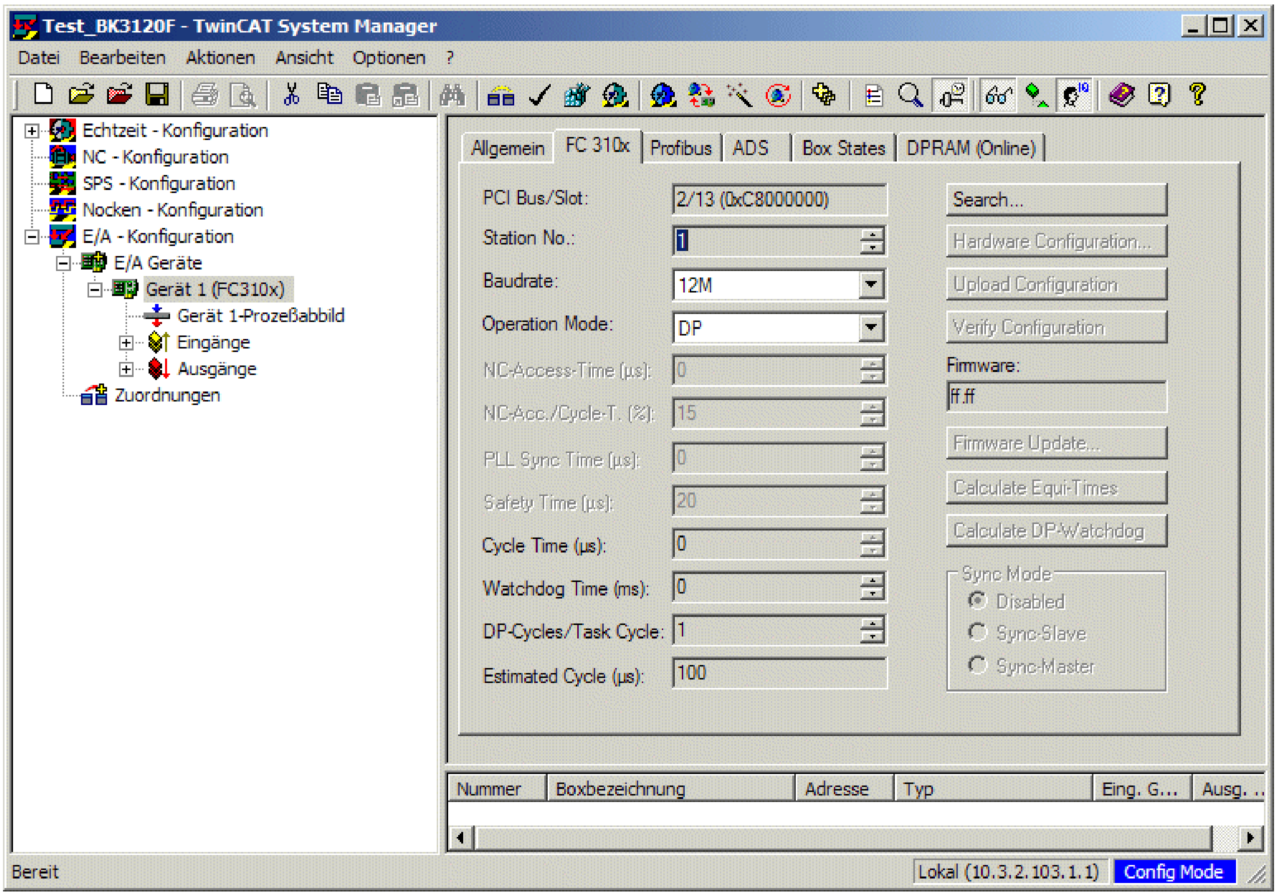
Kommentar:

□ Disabled Symbole erzeugen □

Name	Typ	Größe	>Adre...	Ein/Aus	User ID	Verknüpf
Ausgang 1	BOOL	0.1	4.0	Ausg...	0	
Ausgang 2	BOOL	0.1	4.1	Ausg...	0	
Ausgang 3	BOOL	0.1	4.2	Ausg...	0	
Ausgang 4	BOOL	0.1	4.3	Ausg...	0	
Ausgang 5	BOOL	0.1	4.4	Ausg...	0	
Ausgang 6	BOOL	0.1	4.5	Ausg...	0	
Ausgang 7	BOOL	0.1	4.6	Ausg...	0	
Ausgang 8	BOOL	0.1	4.7	Ausg...	0	

Options for printing the list or copying it to Excel, for example, are available through right-clicking on the list.

The access functions have to use the DP-RAM address that is shown under *PCI bus/slot* on the "FC310x", "FC510x" or "FC520x" tab when the fieldbus device is clicked in the tree:



PCI Bus/Slot: 2/13 (0xC8000000)

Station No.: 1

Baudrate: 12M

Operation Mode: DP

NC-Access-Time (µs): 0

NC-Acc./Cycle-T. (%): 15

PLL Sync Time (µs): 0

Safety Time (µs): 20

Cycle Time (µs): 0

Watchdog Time (ms): 0

DP-Cycles/Task Cycle: 1

Estimated Cycle (µs): 100

Firmware: ff.ff

Sync Mode

● Disabled

○ Sync-Slave

○ Sync-Master

ReadInputsDirect

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

The ReadInputsDirect function is used for reading fieldbus device input variables directly from the DP-RAM. Calling of this function is very fast (approx. 1.5 µs per word). The DP-RAM address dpRamAddress of the fieldbus device, the offset and the length of the input variables in the DP-RAM and a pointer pData to a memory area into which the input data can be copied are transferred.

Return values:

0: no error

-1: DP-RAM pointer could not be allocated

-2: Offset too large

-3: DP-RAM address is different than for previous calls of ReadInputsDirect or WriteOutputsDirect

-5: DLL no longer active

WriteOutputsDirect

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

The WriteOutputsDirect function is used for writing fieldbus device input variables directly into the DP-RAM. Calling of this function is very fast (approx. 1.5 µs per area). The DP-RAM address dpRamAddress of the fieldbus device, the offset and the length of the output variables in the DP-RAM and a pointer pData to the output data are transferred.

Return values:

0: no error

-1: DP-RAM pointer could not be allocated

-2: Offset too large

-3: DP-RAM address is different than for previous calls of ReadInputsDirect or WriteOutputsDirect

-5: DLL no longer active

GetDirectInputPointer

void * GetDirectInputPointer(unsigned long dpRamAddress);

The function GetDirectInputPointer can be used to obtain a pointer to the input process image in the DP-RAM. The addresses of the input variables have to be taken from the System Manager as described above. The DP-RAM address dpRamAddress of the fieldbus device has to be transferred.

Return values:

0: Error

!= 0: Pointer to DP-RAM input variables

GetDirectOutputPointer

void * GetDirectOutputPointer(unsigned long dpRamAddress);

The function GetDirectOutputPointer can be used to obtain a pointer to the output process image in the DP-RAM. The addresses of the output variables have to be taken from the System Manager as described above. The DP-RAM address dpRamAddress of the fieldbus device has to be transferred.

Return values:

0: Error

!= 0: Pointer to DP-RAM input variables

More Information:

www.beckhoff.com/tx1100

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