CANopen Tiny Manager

CANopen® Manager with CANopen Basic Functions for LabVIEW™ - CAN Applications

Software Manual

For Product C.1101.09

esd electronics gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
http://www.esd.eu
Phone: +49 (0) 511 3 72 98-0 • Fax: +49 (0) 511 3 72 98-68

Notes

The information in this document has been checked carefully and is considered to be entirely reliable. esd electronics makes no warranty of any kind regarding the material in this document and assumes no responsibility for any errors that may appear in this document. In particular, the descriptions and technical data specified in this document may not be constituted to be guaranteed product features in any legal sense.

esd electronics reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance, or design.

esd electronics gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Tel.: +49-511-37298-0
Fax: +49-511-37298-68
E-Mail: info@esd.eu
Internet: www.esd.eu

This manual contains important information and instructions on safe and efficient handling of the CANopen Tiny Manager. Carefully read this manual before commencing any work and follow the instructions.
The manual is a product component, please retain it for future use.

Document Information

Document file:	I:\Texte\Doku\MANUALS\PROGRAM\CAN\C.1101.09_CANopen_Tiny_Manager_LabVIEW\CANopen-Tiny-Manager_for_LabVIEW_software-manual_en_12.docx
Date of print:	2026-08-14
Document-type number:	QM.1012.C.1101.09

Software version:	0.5.0

Document History

The changes in the document listed below affect changes in the hardware as well as changes in the description of the facts, only.

Rev.	Chapter	Changes versus previous version	Date
1.0	-	First English manual	2021-10-28
1.1	all	All chapters revised, Names and icons of the VIs inserted, if applicable, figure of defines added	2024-06-18
	1.1	Figure 1 updated	
	2.1	Overview of VIs added	
	1.1, 2.3.1, 2.3.4, 2.4.1, 2.4.2	Note on the required CAN driver version included	
	3	Chapter revised	
1.2	all	All chapters revised, descriptions and figures updated and supplemented	2026-08-14
	2.5.1.1	New chapter: *co Tiny CANopen Error Text.vi*	
	2.5.1.2	New chapter: *co Tiny SDO abort Text.vi*	
	2.7.1	New chapter: *co Tiny NTCAN Error to String.vi*	

Technical details are subject to change without further notice.

Classification of Warning Messages and Safety Instructions

This manual contains noticeable descriptions for a safe use of the CANopen Tiny Manager and important or useful information.

NOTICE

Notice statements are used to notify people on hazards that could result in things other than personal injury, like property damage.

NOTICE

This NOTICE statement contains the general mandatory sign and gives information that must be heeded and complied with for a safe use.

INFORMATION

INFORMATION

Notes to point out something important or useful.

Data Safety

This software can be used to establish a connection to data networks. This may allow attackers to compromise normal function, get illegal access or cause damage.
esd does not take responsibility for any damage caused by the software if operated at any networks. It is the responsibility of the device's user to take care that necessary safety precautions for the device's network interface are in place.

Intended Use

The intended use of the CANopen Tiny Manager is the operation as CANopen® Manager with CANopen Basic Functions for LabVIEW™ - CAN Applications

---

Typographical Conventions

Throughout this manual the following typographical conventions are used to distinguish technical terms.

Convention	Example
File and path names	**`/dev/null`** or **`<stdio.h>`**
Function names	***open()***
Programming constants	`NULL`
Programming data types	`uint32_t`
Variable names	*Count*

Number Representation

All numbers in this document are base 10 unless designated otherwise. Hexadecimal numbers have a prefix of 0x. For example, 42 is represented as 0x2A in hexadecimal format.

Table of Contents

List of Tables

List of Figures

1 Overview

1.1 Description of CANopen Tiny Manager

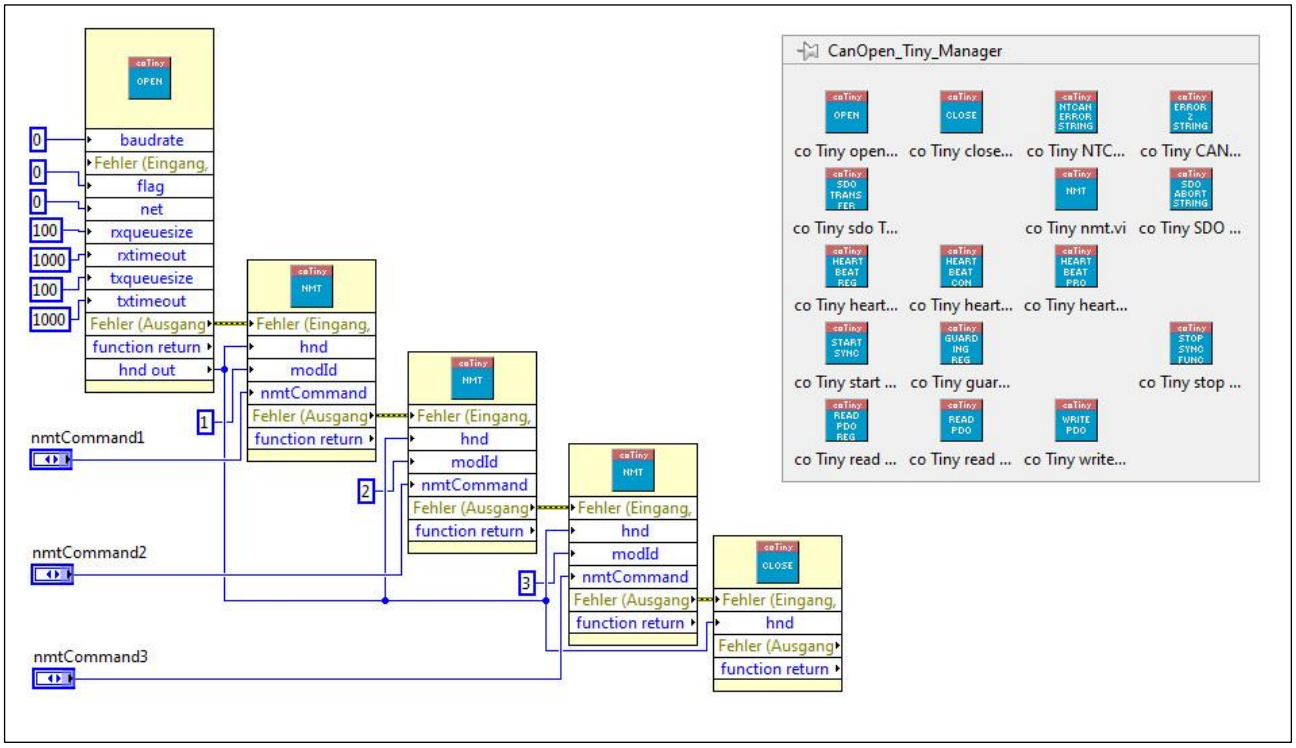


Figure 1: CANopen Tiny Manager with example VIs

The CANopen Tiny Manager for LabVIEW™ is a software library that simplifies the implementation of basic CANopen functionality in LabVIEW™ applications. It provides a dedicated set of Virtual Instruments (VIs) that enable straightforward access to the CANopen Tiny Manager library functions from within LabVIEW.
The library includes functions for CANopen network management as well as data communication services.

The Heartbeat and Guarding mechanisms can be used to monitor the operational status of CANopen devices and detect node failures. With the Heartbeat mechanism, each node periodically transmits its own status information. In contrast, the Guarding mechanism cyclically requests the node status by means of CAN remote frames.
The CANopen Tiny Manager for LabVIEW can also operate as a SYNC producer, periodically transmitting the CANopen Synchronization Object (SYNC) to all nodes in the network.

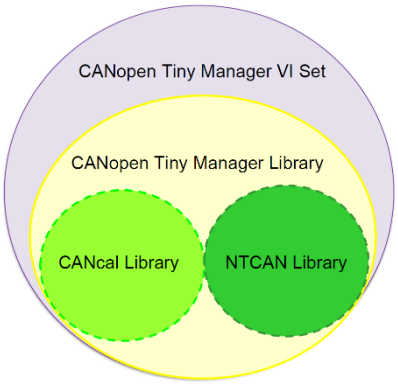
Service Data Objects and Process Data Objects (SDOs & PDOs)
The library supports the transmission and reception of Service Data Objects (SDOs) in expedited, segmented, and block transfer modes.
In addition, functions for transmitting and receiving Process Data Objects (PDOs) are provided, enabling efficient real-time data exchange between CANopen devices.
The CANopen Tiny Manager is compatible with all esd CAN interfaces supported under Windows®.

INFORMATION
Please note, that some of the VIs only work with CAN drivers from version 3.X.
(I.e. not with CAN driver V2 due to the lack of Tx scheduling functionality).

1.2 Basic Functions

The software comprises the following software components:

- CANopen Tiny Manager Library
- CANopen Tiny Manager VI Set



Name	Description
CANopen Tiny Manager **Library**	The CANopen Tiny Manager Library provides new functions based on the NTCAN Library and the CANcal Library.
CANopen Tiny Manager **VI Set**	The VI Set is used in LabVIEW to put the CANopen Tiny Manager Library functions into operation.

Figure 2: Software components

The CANopen Tiny Manager Library provides the underlying functionality for the CANopen Tiny Manager VI Set. The library is based on the CANcal Library and the NTCAN Library, which provide the CANopen protocol and CAN interface services.

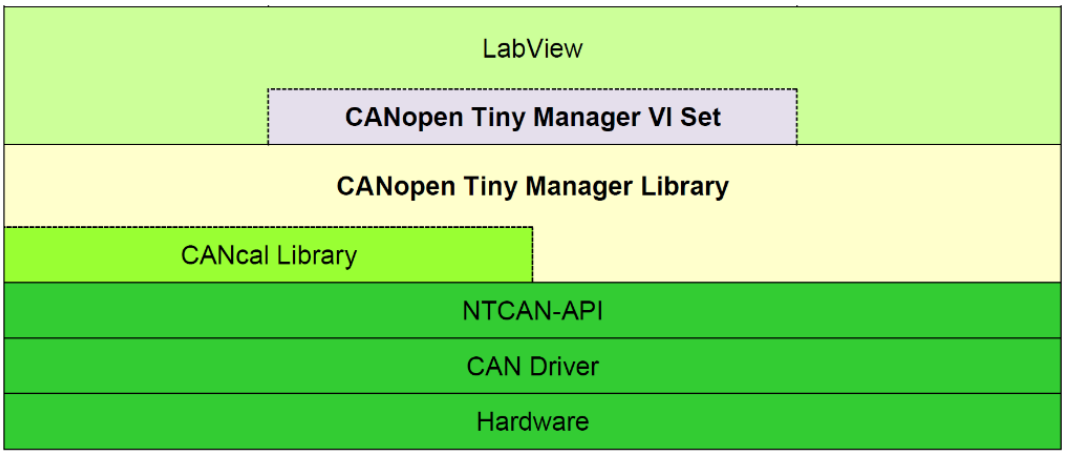


Figure 3: Software structure

1.3 CANopen Tiny Manager Library

CANopen Tiny Manager Library	CANopen Tiny Manager Library provides new functions based on NTCAN Library and CANcal Library.

Intention	Software interface between CANcal, NTCAN and LabVIEW.
Dependency	esd NTCAN library and esd CANcal library
Author / license holder	esd electronics gmbh
Delivery	Binary
Documentation	API description
License terms	See chapter License Terms, page 28

1.4 CANopen Tiny Manager VI Set

CANopen Tiny Manager VI Set	The functions of the CANopen Tiny Manager Library can be used in LabVIEW by means of “Virtual Instruments” (VI). The CANopen Tiny Manager VI Set (VI) supports all functions of the CANopen Tiny Manager Library.

Intention	CANopen Tiny Manager VI Set (VI) is provided for LabVIEW
Dependency	CANopen Tiny Manager library
Author / license holder	esd electronics gmbh
Delivery	Binary
Documentation	VI description
License terms	See chapter License Terms, page 28

1.5 Glossary

Abbreviations

Abbreviation	Term
API	Application Programming Interface
CAN	Controller Area Network
CiA	CAN in Automation
OS	Operating System
PDO	Process Data Object
SDK	Software Development Kit
SDO	Service Data Object

2 Description of VI Set

2.1 Overview of VIs

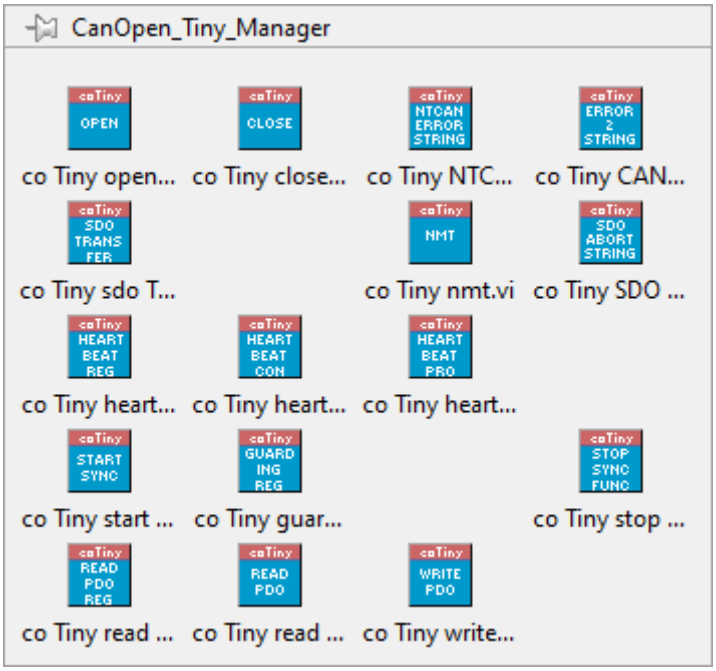


Figure 4: *CanOpen_Tiny_Manager*

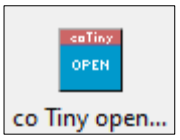
VI Name	Functionality	See page
co Tiny CANopen Error Text.vi	Convert *sdoReturn* of *co Tiny sdo Transfer.vi* to a textual description.	23
co Tiny close.vi	Close the CANopen Tiny Manager Library	13
co Tiny guarding Register.vi	Launch the Node Guarding function	15
co Tiny heart Beat Con.vi	Monitor the received Heartbeat messages	17
co Tiny heart Beat Pro.vi	Register a Heartbeat Producer with a given time interval and NMT status	18
co Tiny heart Beat Register.vi	Register a node as Heartbeat Consumer	16
co Tiny nmt.vi	Handle the network management	14
co Tiny NTCAN Error to String.vi	Convert the *function return* output of the VIs to a textual description.	27
co Tiny open.vi	Open the CANopen Tiny Manager Library	11
co Tiny read Pdo Register.vi	Read PDO register	25
co Tiny read Pdo.vi	Read PDOs	26
co Tiny SDO abort Text.vi	Convert *sdoAbortCode* of *co Tiny sdo Transfer.vi* to a textual description.	23
co Tiny sdo Transfer.vi	Enable SDO access	21
co Tiny start Sync.vi	Send SYNC messages	19
co Tiny stop Sync Functions.vi	Stop sending a defined SYNC message	20
co Tiny write Pdo.vi	Write PDOs	24

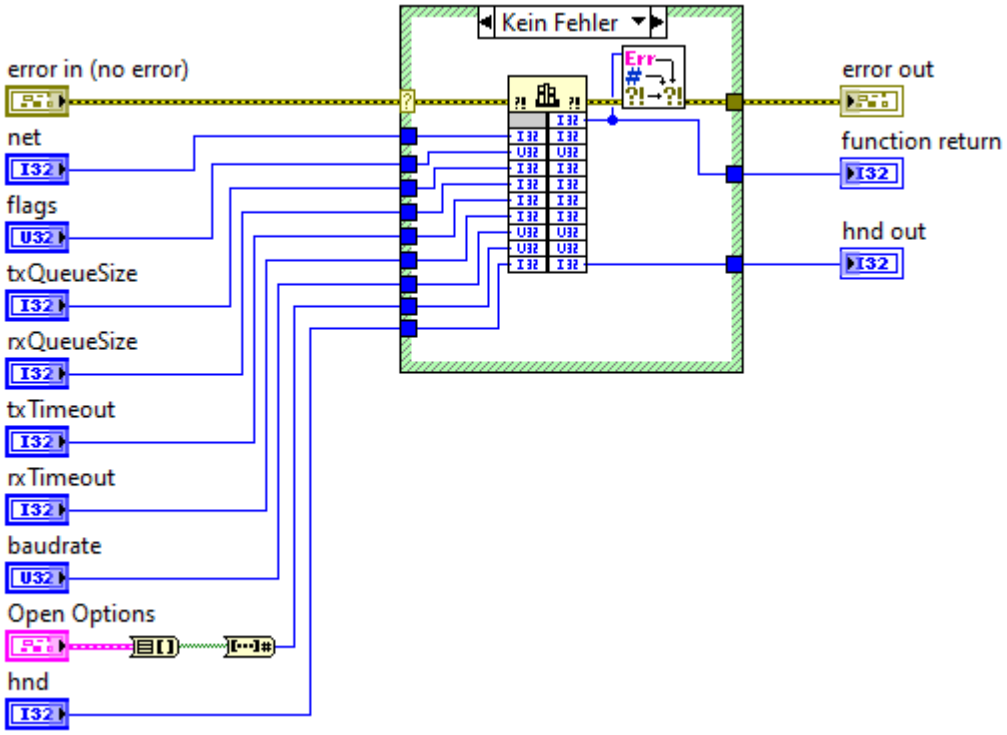
Table 1: VI Overview

2.2 Starting the Network

2.2.1 co Tiny open.vi

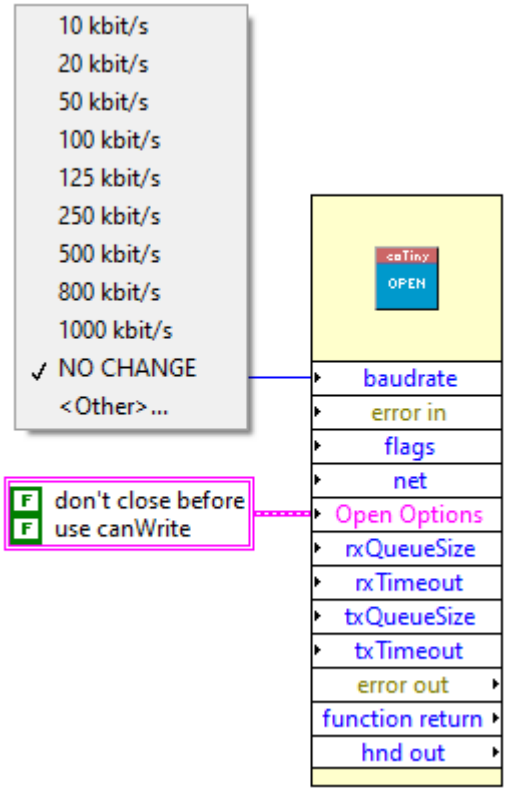
Use this VI to open the CANopen Tiny Manager Library.





Function
int32 coTiny_open(int32_t net, uint32_t flag, int32_t txqueuesize, int32_t rxqueuesize, int32_t txtimeout, int32_t rxtimeout, uint32_t baudrate, uint32_t openflags, COTINY_HANDLE *hnd);

INFORMATION
For further information about the parameters of *coTiny_open* see *canOpen()* and *canSetBaudrate()* in the Application Developers Manual: “NTCAN Part 1: C/C++ Software Design Guide” (CAN-API-ME, see Order Information, page 29).



Open Options:

- don't close before

- With this option disabled (= default) all instances opened for the same net number are closed before opening a new instance on this net.
  This behaviour is helpful if a VI incorrectly aborted without closing the instance.
  However, without the option it is not possible to intentionally open multiple instances with the same net number.

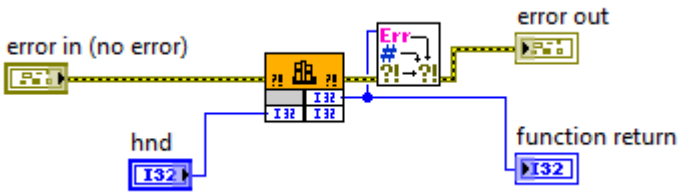
INFORMATION
Please note: There is only one useful scenario for opening the same network multiple times:
Simultaneous SDO access to different nodes on the same CAN net. To allow this, option "don't close before" set is required.

- use canWrite

Internally use the *canWrite()* function (see NTCAN manual) instead of the *canSend()* function for transmitting CAN messages.
The default *canSend()* function is non-blocking, but in the presence of bus faults, the fault diagnosis is limited.
In previous versions of the VI (up to lib 0.4) the internal *canWrite()* function was the default.

2.2.2 co Tiny close.vi

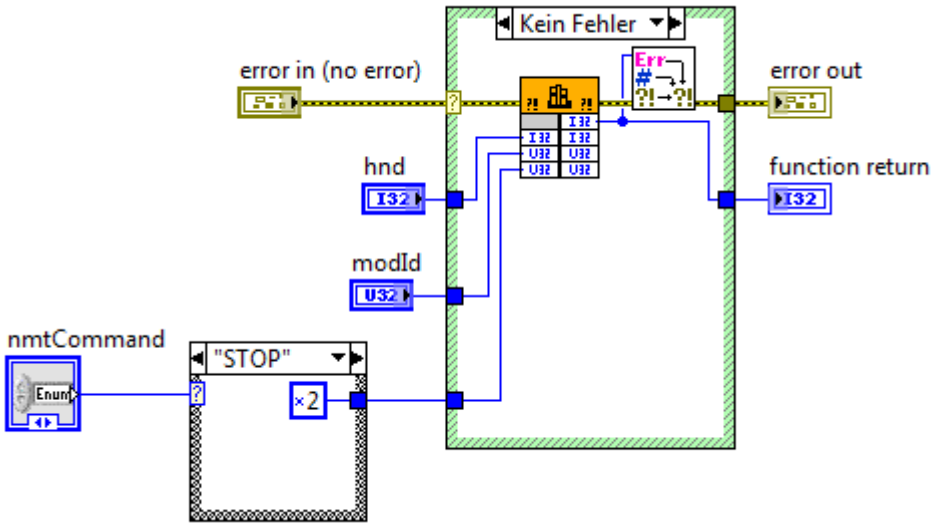
Close the CANopen Tiny Manager Library with this VI.



Function
int32 coTiny_close(COTINY_HANDLE hnd);

2.2.3 co Tiny nmt.vi

This VI handles the network management. The NMT object can be sent.
The function allows the NMT manager to change the status of the other nodes in the network.

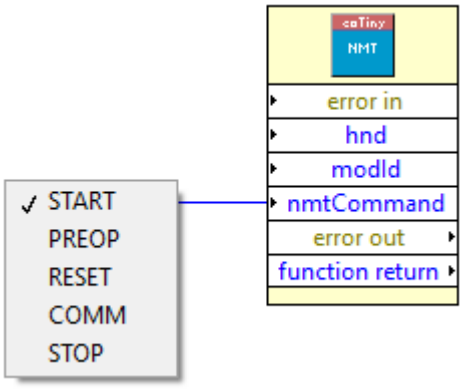


Function

```
int32 coTiny_nmt(COTINY_HANDLE    hnd,
                 uint32_t         modId,
                 uint32_t         nmtCommand);
```

Defines

```
enum NMT_COMMAND {
    NMT_START = 0x01,
    NMT_PREOP = 0x80,
    NMT_RESET = 0x81,
    NMT_COMM = 0x82,
    NMT_STOP = 0x02
};
```

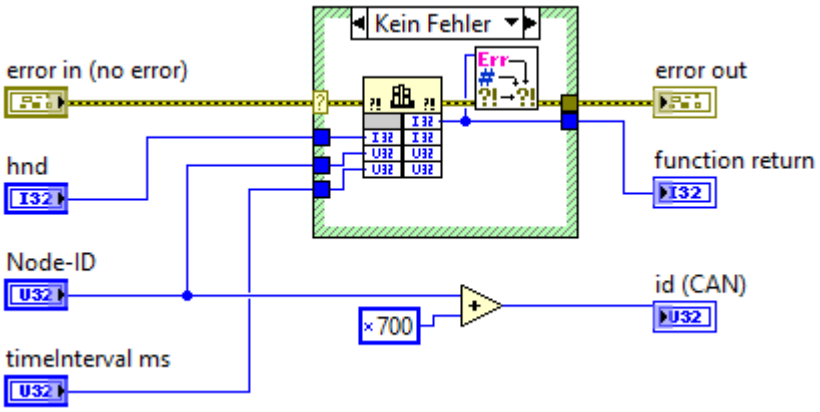


2.3 Network Monitoring

The Heartbeat and Guarding mechanism can be used to monitor CANopen devices, particularly for detecting communication or connection failures.
When using Guarding, the manager cyclically requests the status of each device by transmitting remote frames. In contrast, when the Heartbeat mechanism is used, each node cyclically transmits its own heartbeat message without the use of remote frames. This method is recommended by CiA® (CAN in Automation).

2.3.1 co Tiny guarding Register.vi

This VI is used to launch the Node Guarding function.



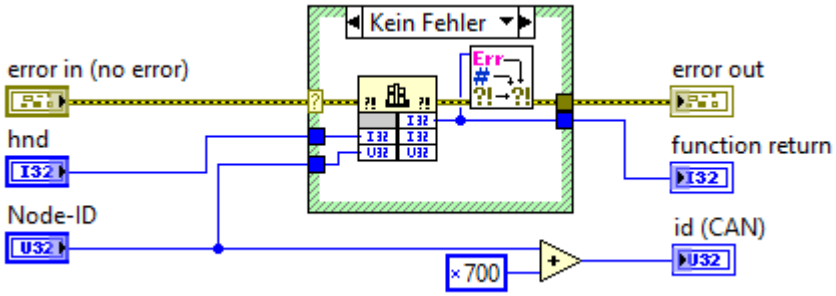
Function
int32 coTiny_guardingRegister(COTINY_HANDLE hnd, uint32_t Node-ID, uint32_t timeInterval ms);

Use *co Tiny heart Beat Con.vi* for the actual monitoring of the node.
Guarding and heartbeat producer messages are automatically stopped with *co Tiny close.vi.*
To stop these functions programmatically in advance, use the *id(CAN)* output as the *id* input for *co Tiny stop Sync Functions.vi.*

INFORMATION
This VI only works with CAN driver version V3.X or later.
(I.e. not with CAN driver V2 due to the lack of Tx scheduling functionality).

2.3.2 co Tiny heart Beat Register.vi

Registers a node as Heartbeat Consumer with this VI. The VI must be called before *co Tiny heart Beat Con.vi.*
The *id(CAN)* output is intended for informational purposes only and represents the CAN ID of the corresponding Heartbeat message.

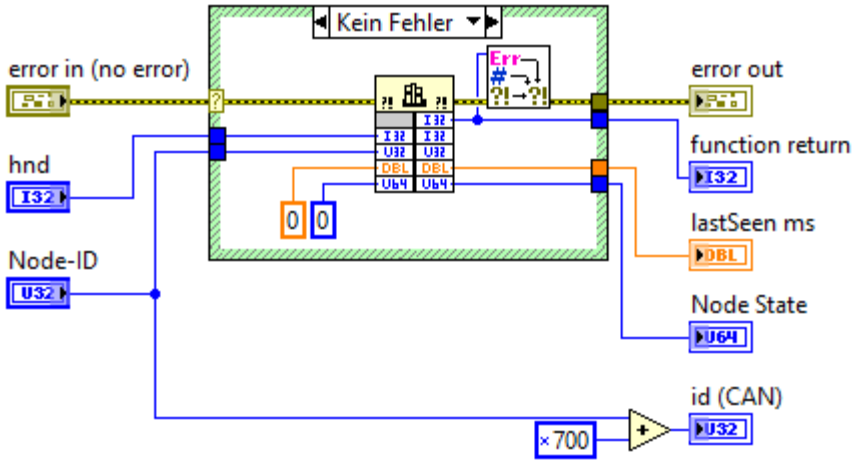


Function
int32 coTiny_heartBeatRegister(COTINY_HANDLE hnd, uint32_t Node-ID;

2.3.3 co Tiny heart Beat Con.vi

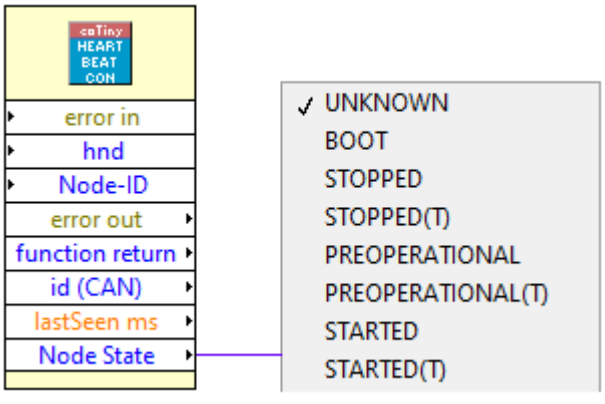
NOTICE
Before using this VI, the Heartbeat Register function *coTiny_heartBeatRegister()* must be used (see *co Tiny heart Beat Register.vi*).

Monitor the received heartbeat messages from a registered node with this VI.



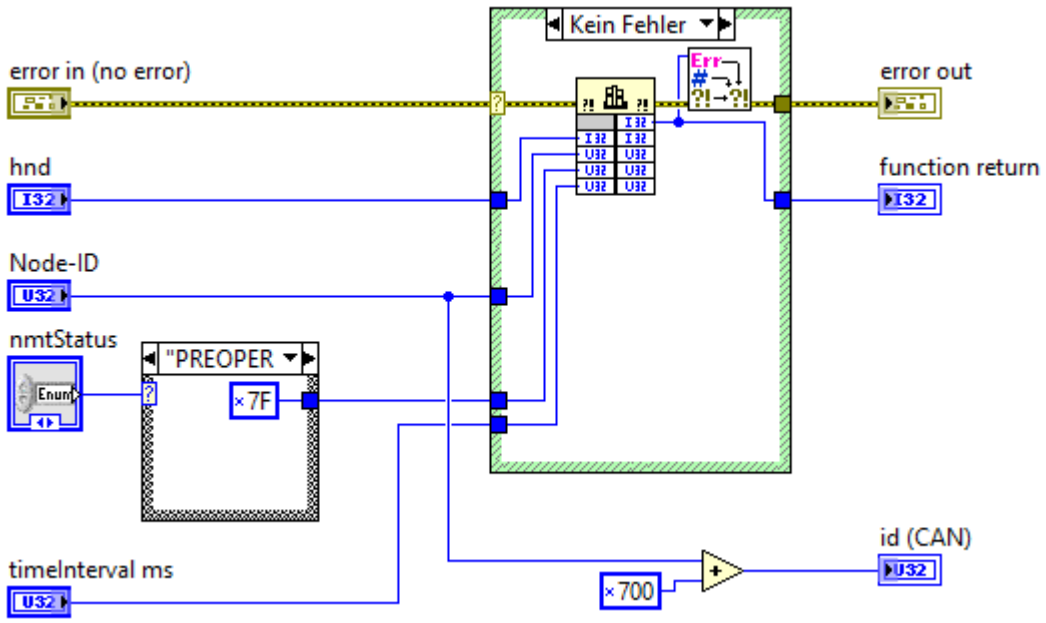
Function
int32 coTiny_heartBeatCon(COTINY_HANDLE hnd, uint32_t Node-ID double *lastSeenMs, uint64_t *data);

The *Node State* output indicates the most recent state reported by the node.
The states marked with *(T)* include the toggle bit used by the Node Guarding protocol. Therefore, when Node Guarding is enabled, the reported state typically alternates between successive responses from the node
The *id(CAN)* output is provided for informational purposes only and represents the CAN ID of the corresponding message.



2.3.4 co Tiny heart Beat Pro.vi

Register a heartbeat producer with a given time interval and NMT status via this VI.

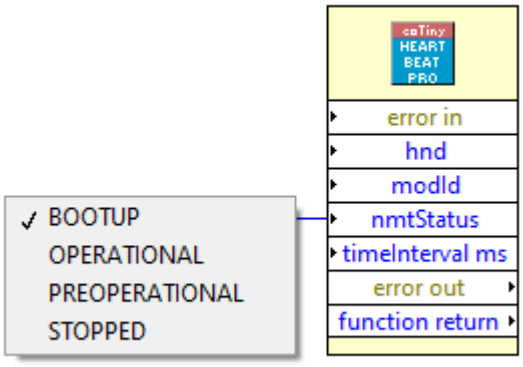


Function
int32 coTiny_heartBeatPro(COTINY_HANDLE hnd, uint32_t Node-ID, uint32_t nmtStatus, uint32_t timeIntervalms);

The heartbeat message for the specified *Node-ID* is sent periodically at the time interval specified by *timeIntervalms*.
To modify the cyclically transmitted *nmtStatus,* call this VI again with the same *timeIntervalms* value. (If the heartbeat interval needs to be changed, the current heartbeat producer must be stopped using *co Tiny stop Sync Functions.vi*!)
Heartbeat producer and node guarding messages are automatically terminated by *co Tiny close.vi*. To stop programmatically in advance, use the *id(CAN)* output as the *id* input for *co Tiny stop Sync Functions.vi*.

Defines

enum NMT_STATUS {
NMT_BOOTUP = 0x00,
NMT_OPERATIONAL = 0x05,
NMT_PREOPR = 0x7f,
NMT_STOPPED = 0x04
};

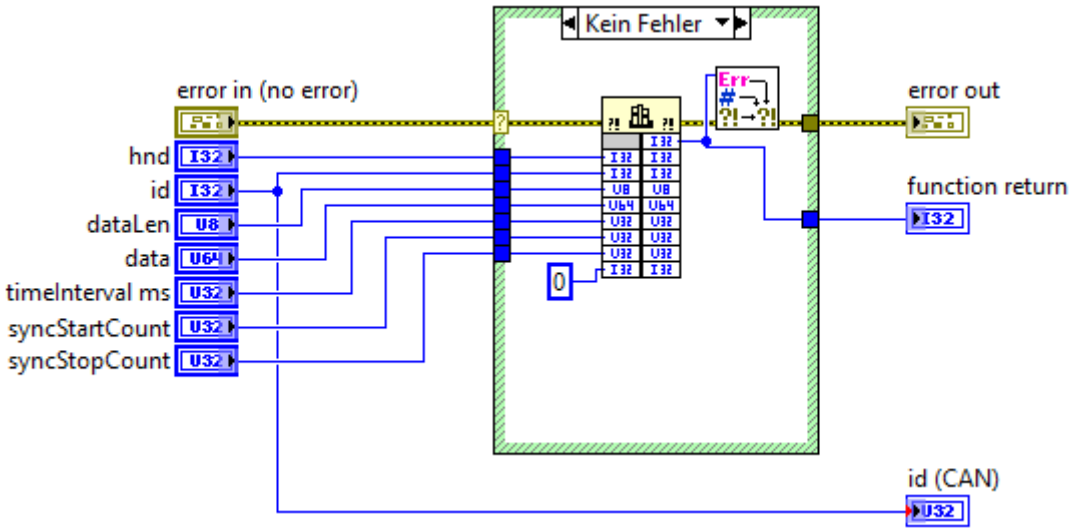


INFORMATION
This VI only works with CAN driver version V3.X or later.
(I.e. not with CAN driver V2 due to the lack of Tx scheduling functionality).

2.4 Network Synchronization

2.4.1 co Tiny start Sync.vi

Start cyclic broadcast of the SYNC messages using the default CAN ID 0x80 or specify a different Sync CAN ID *id*.



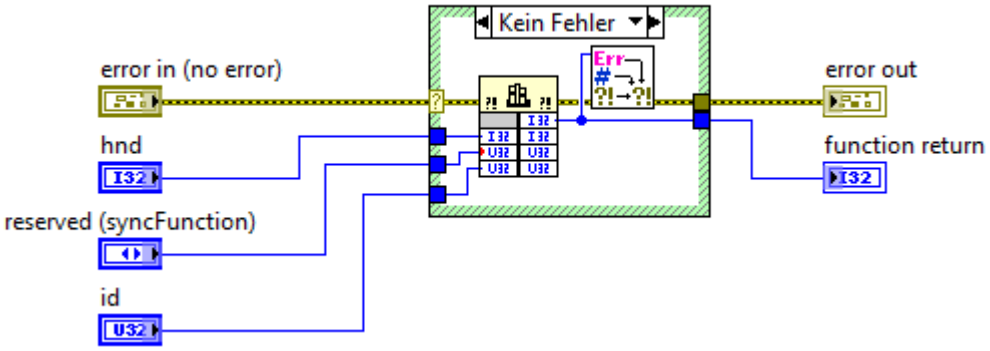
Function
int coTiny_startSync(COTINY_HANDLE hnd, int32_t id, uint8_t dataLen, uint64_t data, uint32_t timeIntervalms, uint32_t syncStartCount uint32_t syncStopCount uint32_t internal_flag);

The default CAN data length is 0 bytes. Optionally, *dataLen* can be set to 1 and specify non-zero values for *syncStartCount* and *syncStopCount.*
SYNC transmission is automatically terminated by *co Tiny close.vi*. For manual termination, use *co Tiny stop Sync Functions.vi* with the same CAN ID that was specified when starting the SYNC transmission.
To change the cycle time specified by *timeIntervalms,* the current SYNC transmission must first be stopped using *co Tiny stop Sync Functions.vi*. The VI can then be called again with the new interval value.

INFORMATION
This VI only works with CAN driver version V3.X or later.
(I.e. not with CAN driver V2 due to the lack of Tx scheduling functionality).

2.4.2 co Tiny stop Sync Functions.vi

Stops a cyclic transmission previously started by *co Tiny start Sync.vi*, *co Tiny heart Beat Pro.vi* or *co Tiny guarding Register.vi*.



Function
int32 coTiny_stopSyncFunctions(COTINY_HANDLE hnd, uint32_t reserved, (previously syncFunction) int32_t id);

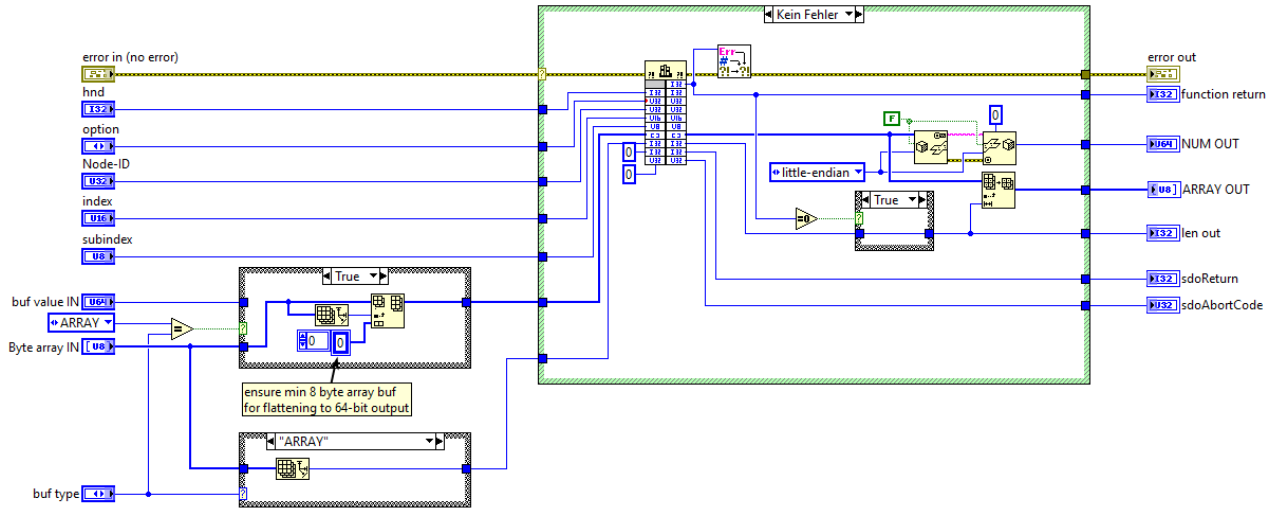
The *id* input must match the *id(CAN)* output returned by the corresponding *start* function.
All active cyclic transmissions are automatically stopped when *co Tiny close.vi* is executed.
(Legacy compatibility: The *syncFunction* input from previous VI implementations (library version 0.4 and earlier) is retained for compatibility reasons only and is no longer evaluated.)

INFORMATION
This VI only works with CAN driver version V3.X or later.
(I.e. not with CAN driver V2 due to the lack of Tx scheduling functionality).

2.5 SDO (Service Data Object)

2.5.1 co Tiny sdo Transfer.vi

This VI provides access to CANopen Service Data Objects (SDOs).
SDOs can be used to read from and write to CANopen object dictionary entries, for example for device configuration and parameterization.
For SDO communication, the CANopen Tiny Manager Library uses the *ClientUpload* and *ClientDownload* functions of the CANcal Library.



Function

```
int32 coTiny_sdoTransfer(COTINY_HANDLE    hnd,
                         uint32_t         options,
                         uint8_t          Node-ID,
                         uint16_t         index,
                         uint8_t          subindex,
                         void             *buf,
                         int32_t          *len;
                         int32_t          *sdoReturn,
                         uint32_t         *sdoAbortCode);
```

SDO Read Operations:
The *NUM_OUT* output returns the SDO data interpreted as an unsigned 64-bit integer.
For other data types, use LabVIEW's raw binary conversion functions to convert the data into the required representation (see the SDO example).
The ARRAY_OUT output provides the received data as a byte array.
To read SDO entries larger than 8 bytes, set *buf type* to *ARRAY* and provide a reserved and initialized byte array at *Byte array IN.* The array length must be greater than or equal to the expected SDO data length.

SDO Write Operations:
For SDO transfers up to 8 bytes, select a *buf type* that matches the data length (1 ... 8 bytes).
Provide the data to be written via *buf value IN*, which is interpreted as an unsigned 64-bit integer.
For data types other than unsigned integers, use raw binary conversion functions before writing the data (see the SDO example).
To write more than 8 bytes, set *buf type* to ARRAY and provide the data as a byte array at *Byte array IN*.

Parallel SDO Access
Each CANopen node can process only one SDO request (that is, one client/server CAN ID pair) at a time.
However, SDO requests to different nodes can be processed in parallel.
Due to the implementation of *co Tiny sdo Transfer.vi*, parallel access requires multiple handles opened with *co Tiny open.vi* on the same CAN network. These handles must be supplied to separate instances of *co Tiny sdo Transfer.vi*. Otherwise, all SDO requests, including requests to different nodes, are processed sequentially.

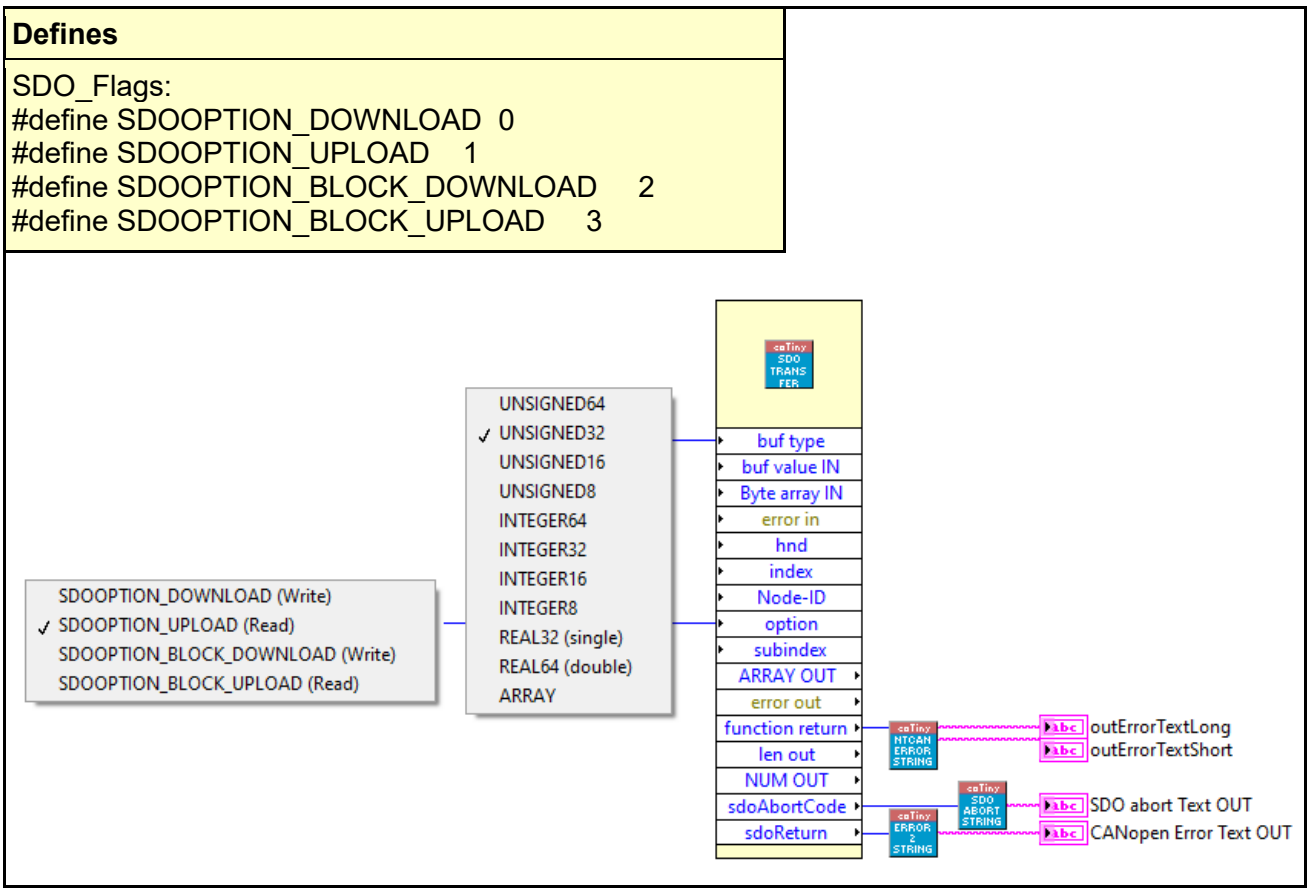
Error Handling
The VI provides three error outputs:
- *function return* – Overall function status and collective error information
- *sdoReturn* – Return code from the underlying SDO client functions.
- *sdoAbortCode* – SDO abort code returned by the target node.
  This value is valid only if *sdoReturn* indicates an Abort condition.

For a textual description of error codes, use the following VIs:
co Tiny NTCAN Error to String.vi
co Tiny CANopen Error Text.vi
co Tiny SDO Abort Text.vi

These VIs convert the respective error codes into readable error messages.

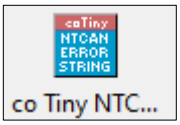
Defines

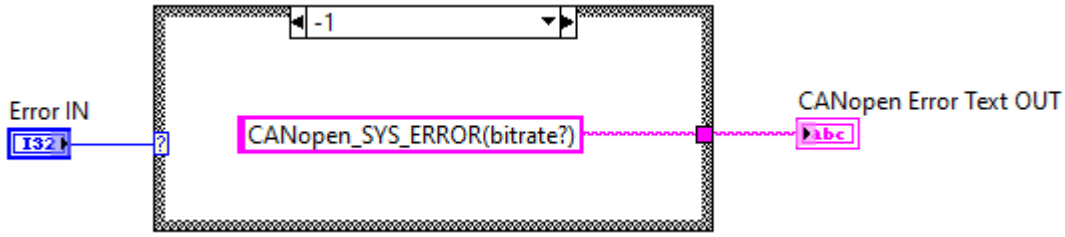
SDO_Flags:
#define SDOOPTION_DOWNLOAD 0
#define SDOOPTION_UPLOAD 1
#define SDOOPTION_BLOCK_DOWNLOAD 2
#define SDOOPTION_BLOCK_UPLOAD 3



2.5.1.1 co Tiny CANopen Error Text.vi

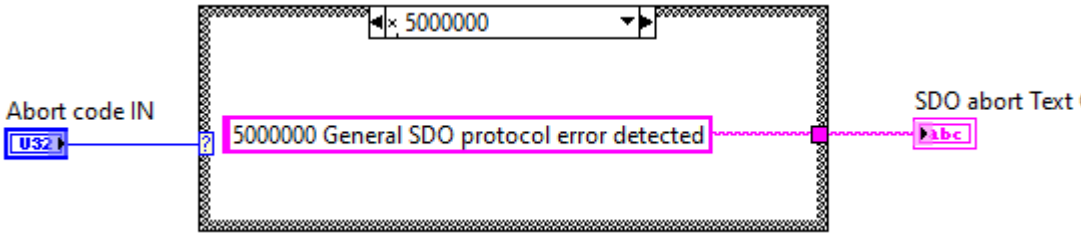
Converts the *sdoReturn* error code of *co Tiny sdo Transfer.vi* into a textual description.





2.5.1.2 co Tiny SDO abort Text.vi

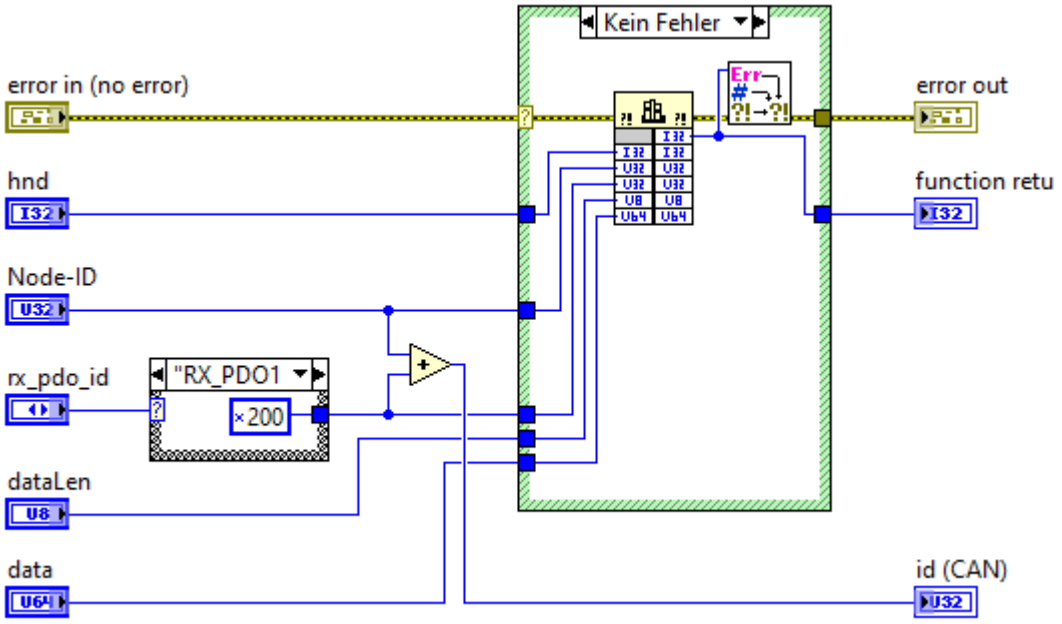
Converts the *sdoAbortCode from co Tiny sdo Transfer.vi* into a textual description of the corresponding SDO abort condition.



2.6 PDO (Process Data Object)

2.6.1 co Tiny write Pdo.vi

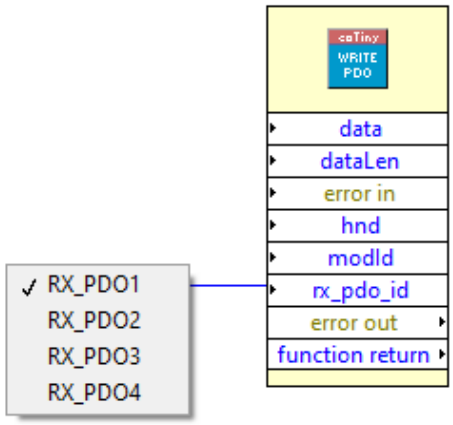
This VI can be used to write data in any PDO mapping.



Function
int32 coTiny_writePdo(COTINY_HANDLE object_hnd, uint32_t Node-ID, uint32_t rx_pdo_id, uint8_t dataLen, uint64_t data);

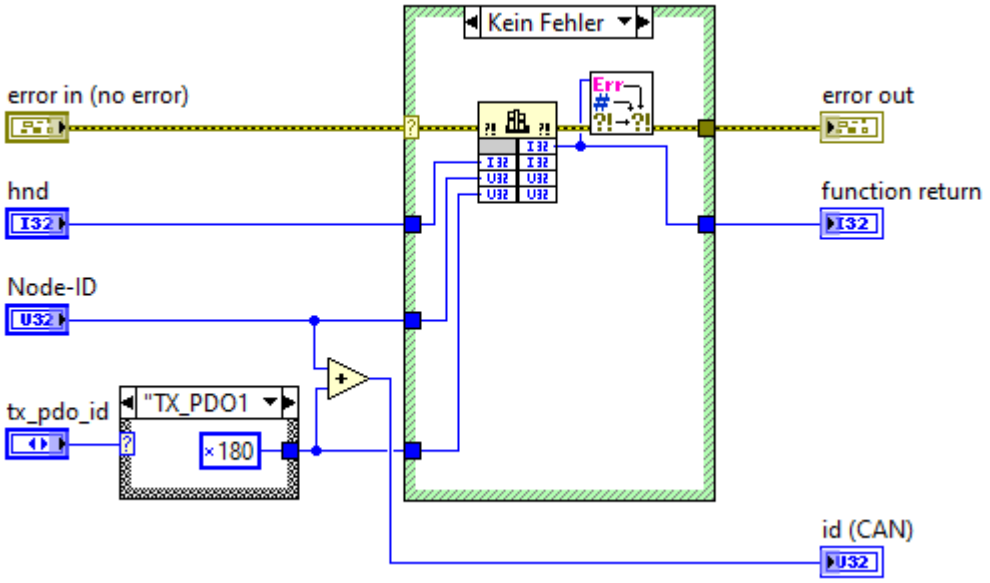
Defines

enum RX_PDO_ID {
RX_PDO1 = 0x200,
RX_PDO2 = 0x300,
RX_PDO3 = 0x400,
RX_PDO4 = 0x500
};



2.6.2 co Tiny read Pdo Register.vi

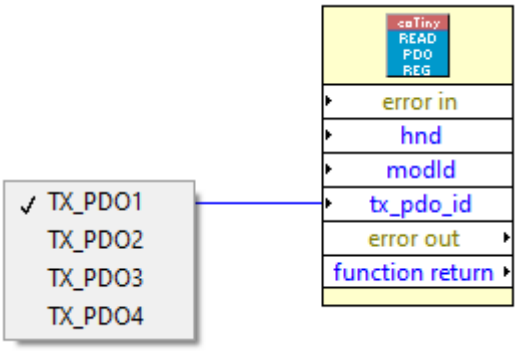
Before PDO messages can be received using *co Tiny read PDO.vi*, this VI must be called to register the corresponding PDOs.



Function
int32 coTiny_readPdoRegister(COTINY_HANDLE object_hnd, uint32_t Node-ID, uint32_t tx_pdo_id);

Defines

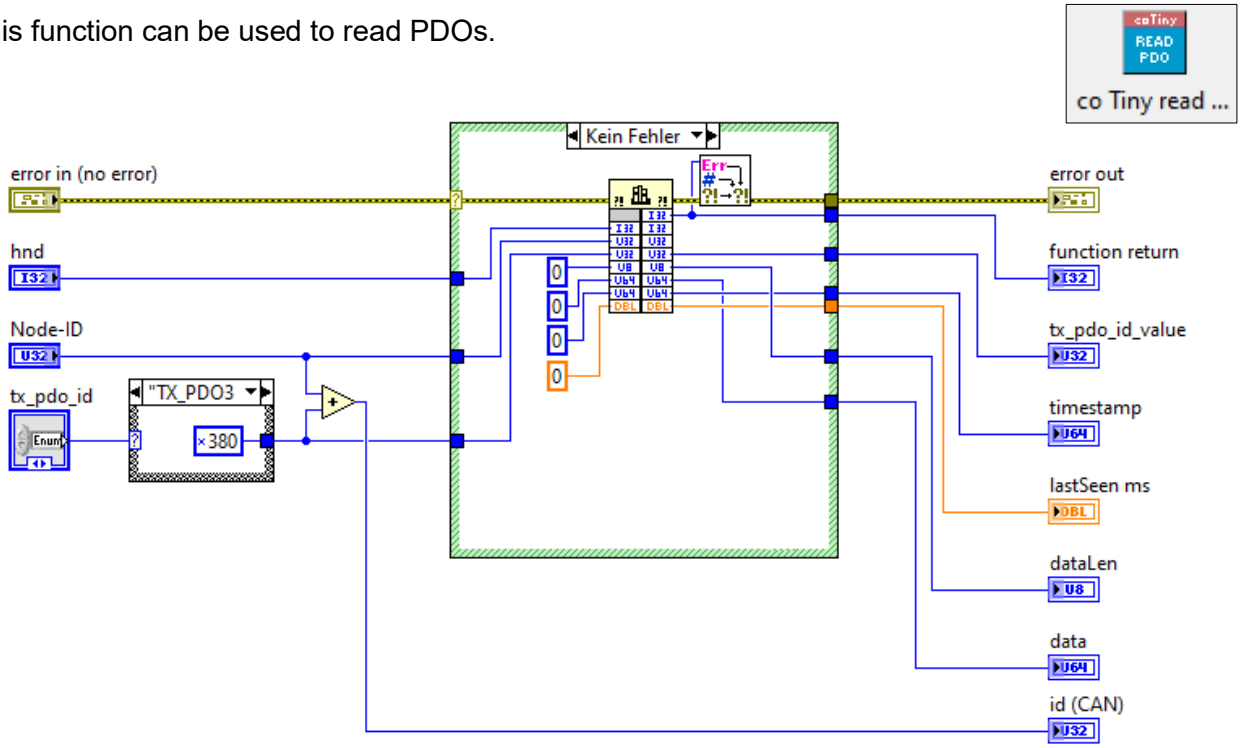
enum TX_PDO_ID {
TX_PDO1 = 0x180,
TX_PDO2 = 0x280,
TX_PDO3 = 0x380,
TX_PDO4 = 0x480
};



2.6.3 co Tiny read Pdo.vi

NOTICE
Before using this VI, use co Tiny read Pdo Register.vi to register the corresponding PDOs. The *co Tiny* read Pdo Register*()* function must be used before the function *coTiny_readPdo()* can be used.

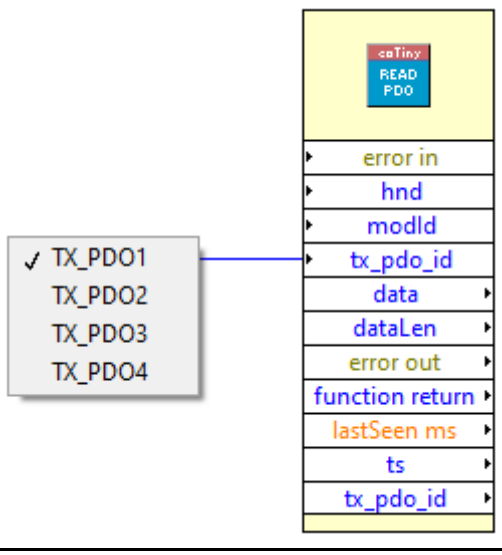
This function can be used to read PDOs.



Function
int32 coTiny_readPdo(COTINY_HANDLE hnd, uint32_t Node-ID, uint32_t tx_pdo_id, uint8_t *dataLen, uint64_t *data, uint64_t *timestamp, double *lastSeenMs);

Defines

enum TX_PDO_ID {
TX_PDO1 = 0x180,
TX_PDO2 = 0x280,
TX_PDO3 = 0x380,
TX_PDO4 = 0x480
};



2.7 Return Values

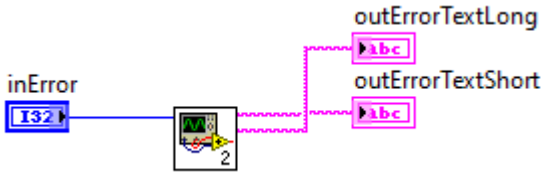
All NTCAN-API functions return a status code starting with the prefix 'NTCAN_', which should always be checked by the application.
If a function returns an error code, all output values provided through pointer parameters are considered undefined and must not be evaluated by the application.
The constants defining the NTCAN return codes are declared in `<ntcan.h>`.

Detailed information about the general return codes can be found in the Application Developers Manual: NTCAN Part 1: "C/C++ Software Design Guide", chapter "Return Codes" (CAN-API-ME, see Order Information, page 29).

Use *co Tiny NTCAN Error to String.vi* to convert the *function return* output of the VIs into a textual description.

2.7.1 co Tiny NTCAN Error to String.vi

Convert the *function return* output of the VIs to a textual description.



3 License Terms

NOTICE
The software from esd used in the CANopen Tiny Manager is subject to the license terms of the respective authors or rights holders. CANopen Tiny Manager may only be used in accordance with these license terms!
By using the CANopen Tiny Manager you agree to the terms of these software licenses.

The license terms of esd electronics gmbh (Software Licence Agreement) are displayed and installed on your system during installation of the CAN SDK.
You can also download the Software Licence Agreement from our website:

https://esd.eu/fileadmin/esd/software/licenses/esd/license_binary.txt

4 Order Information

Type	Properties	Order No.
CANopen Tiny Manager for LabVIEW	CANopen basic functions for NI LabVIEW Available for NI LabVIEW 2013 for Windows 7/8/10/11 (32-/64-bit) operating systems	C.1101.09

Table 2: Order information

PDF Manuals
For the availability of the manuals see table below.
Please download the manuals as PDF documents from our esd website https://www.esd.eu for free.

Manuals		Order No.
CANopen Tiny Manager-ME	Software manual in English	C.1101.10
CAN-API-ME	NTCAN-API Part 1: Application Developers Manual NTCAN-API Part 2: Driver Installation Guide	C.2001.21
CANopen-ME	CANopen Manuals in English	C.2002.21

Table 3: Available Manuals

Printed Manuals
If you need a printout of the manual additionally, please contact our sales team (sales@esd.eu) for a quotation. Printed manuals may be ordered for a fee.