SYS TEC ELECTRONIC

CANopen API for .NET

Software Manual

preliminary Edition March 2010

In this manual are descriptions for copyrighted products which are not explicitly indicated as such. The absence of the trademark (©) symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual

The information in this document has been carefully checked and is believed to be entirely reliable. However, SYS TEC electronic GmbH assumes no responsibility for any inaccuracies. SYS TEC electronic GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. SYS TEC electronic GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.

Additionally, SYS TEC electronic GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. SYS TEC electronic GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

Contact	**Direct**	**Your local distributor**
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	Please find a list of our distributors under http://www.systec-electronic.com/distributors
Ordering Information:	+49 (3661) 6279-0 info@systec-electronic.com	
Technical Support:	+49 (3661) 6279-0 support@systec-electronic.com	
Fax:	+49 (3661) 62 79 99	
Web Site:	http://www.systec-electronic.com	

3rd preliminary Edition March 2010

Index of Figures and Tables

1 Introduction

The CANopen API for .NET is a wrapper around the SYS TEC CANopen stack that is built on the Microsoft .NET framework.

The .NET framework provides a sophisticated way of implementing software libraries. These libraries are called assemblies.
The main advantages of implementing a software library as .NET assembly are:

- Common object oriented interface, which is easy to understand.
- The assembly can be used by many programming languages like but not limited to C#, Visual Basic .NET, C++/CLI.
- The object oriented interface is the very same in all .NET languages. There is no need for “wrappers” anymore.
- Deployment of the assembly is very simple.
- Versioning support for each assembly.

It is assumed that you are familiar with CANopen and its usage. This includes the CiA specification 301 [1].

1.1 Features of the CANopen API for .NET

The CANopen API for .NET provides a simple interface to the SYS TEC CANopen stack.

It has the following common characteristics:

- Object oriented class model
- Supports the .NET framework 2.0
- Uses the .NET framework event model for CANopen events
- Implements the .NET exception model
- Implemented in C++/CLI
- Provides XML file for IntelliSense documentation

The following CANopen functionality is currently supported:

- Multiple separate instances of CANopen (up to 16 instances with SO-1085 and SO-1088)
- SYS TEC CAN-Wrapper as CAN driver, which supports USB-CANmodul, CAN-Ethernet-Gateway and more
- NMT master and slave (selectable at run time)
- Fixed object dictionary (may-be user extendable in future versions)
- 128 SDO clients
- One SDO server (the default one)
- 126 heartbeat consumers
- Heartbeat producer
- 126 emergency consumers
- Emergency producer
- LSS master
- Reception and transmission of plain CAN layer 2 messages (at least 20 COBs per direction may exist at the same time)

1.2 Types of CANopen API for .NET

The CANopen API for .NET is available in various types with different capabilities.

1.2.1 SO-1088 CANopen API for .NET limited for SYS TEC CAN interfaces

This version is freely available for SYS TEC PC to CAN interfaces, e.g. all USB-CANmoduls.

1.2.2 SO-1085 CANopen API for .NET

This version may be used with CAN interfaces from other vendors which are supported by the SYS TEC CAN-Wrapper.

1.2.3 SO-877 CANopen Source Code

The CANopen Source Code includes also the source of the CANopen API for .NET. It may be adapted and extended to your needs. For example the object dictionary can be modified or the number of CANopen instances can be increased or decreased.

1.3 Requirements

To use the CANopen API for .NET you must ensure that the following software packages are installed:

- Microsoft .NET framework 2.0
- Microsoft Visual C/C++ Runtime 2005 SP 1 [3]
- SYS TEC CAN-Wrapper (file CDRVWRAP.DLL)
- Supported CAN interface with the appropriate driver, e.g.
  - SYS TEC USB-CANmodul
    with SO-387 USB-CANmodul Utility Disk
  - SYS TEC CAN-Ethernet-Gateway
    with SO-1027 CAN-Ethernet-Gateway Utility Disk

2 Directory structure

The software package has the following directory structure.

Directory	Content
/	Assembly CANopenDotNET.DLL, CAN-Wrapper CDRVWRAP.DLL, XML documentation for IntelliSense
/Demo.CppCLI/	Sample project for C++/CLI, which demonstrates SDO client, NMT master, COB, etc.
/Demo.Cs/	Sample project for C#, which demonstrates SDO client, NMT master, COB, etc.
/Demo_LSS_Master.CppCLI/	LSS Master project for C++/CLI
/Demo_LSS_Master.Cs/	LSS Master project for C#
/Docu/	Manuals, e.g. this manual

Table 1: Directory structure

Because the assembly is located directly in the installation directory, this software package can be installed into your application directory through the supplied setup file.

3 Integration and installation

The assembly CANopenDotNET.DLL can be used with programming languages that support the .NET framework 2.0. This includes for example all languages that are supported by Microsoft Visual Studio 2005.

Basically there are two ways how to integrate an assembly into your project: either as private or as shared assembly. Shared assemblies are stored in the global assembly cache (GAC) on the computer and are usable by multiple applications. To reference a shared assembly in the GAC they need a globally unique name, which is called strong name. Currently, CANopenDotNET.DLL does not have got a strong name. So it is not possible to install this assembly in the GAC.

The second and easiest way is to use the assembly privately. That means you just have to copy it to your application's directory and reference it from your application.

Additionally, the SYS TEC CAN-Wrapper CDRVWRAP.DLL must be accessible. That means this DLL has to reside either in your application's directory or the SYSTEM32 directory of your Windows installation. It may be that the CAN-Wrapper was already installed by another application. Then the condition mentioned above is already met.

3.1 Microsoft Visual Studio 2005

Adding a reference to an assembly in Visual Studio is very simple.

- Right click on the project entry in the Solution Explorer.
- Go to entry "References…" and "Add new reference…" if it is a C++/CLI project or just "Add reference…" otherwise.
- Open the "Browse" tab and select "CANopenDotNET.DLL".
- Press Ok to confirm
- Make sure that "local copy" is enabled for this assembly.

The CANopen API for .NET provides IntelliSense documentation. To use it you have to keep the XML file CANopenDotNET.XML with the DLL file.

3.2 Deployment

If you create a setup program for your application, just ensure that the two DLL files CANopenDotNET.DLL and CDRVWRAP.DLL will be copied to your application's program directory.

Additionally the Microsoft .NET framework 2.0 and the Microsoft Visual C/C++ Runtime 2005 SP1 must be installed on the target system (see 1.2.2).

If you use the software packages SO-1085 or SO-1088, you may use the supplied setup file to install the complete software package to your application's program directory. This assures all necessary preconditions.

4 Object model

4.1 Namespace CANopenDotNET

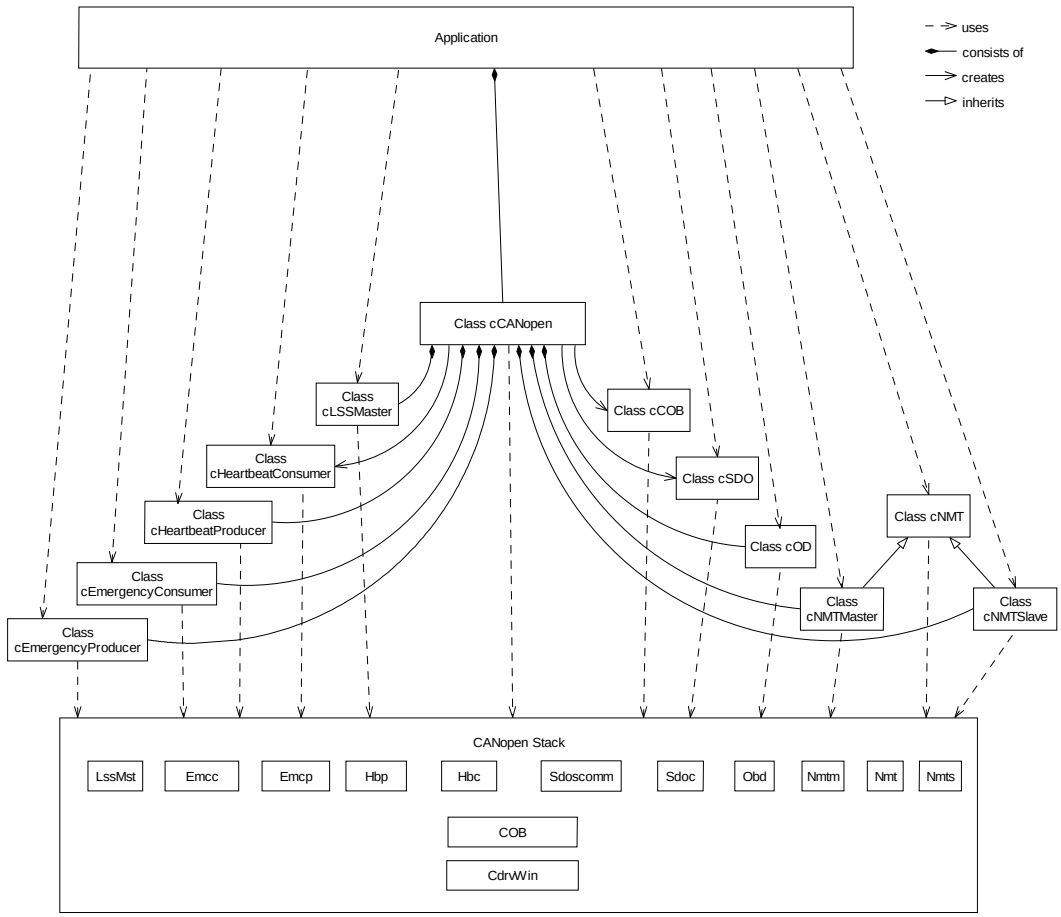

Figure 1 UML class diagram

The assembly CANopenDotNET.DLL comprises just one namespace CANopenDotNET.

This namespace contains all classes, enumerations and value types that implement the wrapper of the CANopen stack. The application is only able to create instances of the cCANopen class. The other classes are created by cCANopen instances. Some classes like the cOD class for the object dictionary exists only once per cCANopen instance. Others may be created multiple times per cCANopen instance.

The following sections provide a short introduction in what the assembly offers. For a complete reference see section 7 “Class reference”.

4.1.1 Class cCANopen

One object of this reference class represents one CANopen instance. It creates all related objects like cSDO, cOD, cNMT, etc. An object of this class can be created directly by the application. Furthermore, it has to be disposed by the application if it is no longer used anymore. The disposing will shut down the CANopen instance including the associated CAN driver instance and release all resources (i.e. managed and unmanaged).

See section 7.4.

4.1.2 Class cNMT

This abstract reference class models the local NMT state machine.

See section 7.5.

4.1.3 Class cNMTMaster

This reference class which provides the NMT master functionality like controlling and guarding of NMT slave nodes. It is derived from the abstract class cNMT.

See section 7.6.

4.1.4 Class cNMTSlave

This reference class which provides the NMT slave functionality. It is derived from the abstract class cNMT.

See section 7.7.

4.1.5 Class cOD

This reference class models the local object dictionary. It provides methods for accessing the local object dictionary.

See section 7.8.

4.1.6 Class cSDO

This reference class models one local SDO client. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

See section 7.9.

4.1.7 Class cCOB

This class provides the functionality to send and receive plain CAN layer 2 messages, i.e. communication objects (COB). An instance of this class represents one communication object. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

See section 7.10.

4.1.8 Class cHeartbeatConsumer

This reference class models one local heartbeat consumer. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

See section 7.11.

4.1.9 Class cHeartbeatProducer

This reference class models the local heartbeat producer.

See section 7.12.

4.1.10 Class cEmergencyConsumer

This reference class models the local emergency consumer.

See section 7.13.

4.1.11 Class cEmergencyProducer

This reference class models the local emergency producer.

See section 7.14.

4.1.12 Class cLSSMaster

This reference class which provides the LSS master functionality to configure LSS slaves.

See section 7.15.

4.1.13 Enumerations and value types

The namespace contains a bunch of enumerations and value types. These are used as arguments for class methods and described in detail when the corresponding methods are explained.

One important enumeration is enumCopKernel. It represents the error codes from the CANopen stack. The user gets in touch with this enumeration only in two situations: either it catches a cCANopenException or it consumes the event cCANopen.EventError via the appropriate event handler.

4.1.14 Exception cCANopenException

This exception is thrown whenever a CANopen stack function returns an enumCopKernel error code. This is the case if a severe error occurred and the function cannot continue the operation. The application should catch this exception whenever it calls a CANopen method. Otherwise the application would crash if such an exception was raised.

See section 7.2.

5 Thread model

5.1 Process thread

The SYS TEC CANopen stack for Microsoft Windows uses a multi-threaded approach. It creates a process thread for each initialized CANopen instance. This process thread is in charge of the following functions. It handles incoming messages over the CAN-Bus, like request for the SDO server and heartbeats from remote nodes. It monitors timeouts, e.g. SDO transfer timeouts, and cyclic task like the heartbeat producer. Furthermore, it processes more complex tasks like the switch mode selective command of the LSS master. Additionally, the SYS TEC CAN-Wrapper creates threads to process CAN message, but that is totally transparent to the application.

It is important for the application, that delegates which are registered for events are called within the CANopen instance's process thread. It is not allowed to call any methods of the CANopenDotNET namespace within the delegate if not stated otherwise. That is because the CANopen methods must be synchronized with the process thread and critical sections cannot be entered twice in the same thread without deadlock. Another reason is that it is not allowed to call CANopen functions within an event callback function even without multiple threads, because this may cause in unpredictable results.

To circumvent this problem you may start a worker thread in your delegate.

5.2 Reentrant and thread-safe methods

There is a difference between a reentrant method and a thread-safe method.

A reentrant method may be called simultaneously by multiple threads for different object instance. It is not safe to call a reentrant method by multiple threads for the very same object instance.

On the other hand a thread-safe method may be called simultaneously by multiple threads for the same object instance.

Most CANopen methods are reentrant, but not thread-safe. There exist some exceptions: The constructors of the cCANopen class and the ConnectToNet() resp. BeginConnectToNet() of the cNMT class are neither reentrant nor thread-safe. The application has to assure that these methods are not called simultaneously by multiple threads.

On the other hand the Get…() and Create…() methods of the cCANopen class are thread-safe.

6 Object dictionary

In the current version the CANopen API for .NET is provided with a fixed default object dictionary. This object dictionary should be sufficient for most applications.

In the future, the CANopen API for .NET may support dynamic object dictionaries.

If you have access to the CANopen Source Code (SO-877), you are able to extend the object dictionary to your needs.

7 Class reference

7.1 Enumeration enumCopKernel

The enumeration enumCopKernel represents the error codes from the CANopen stack. The user gets in touch with this enumeration only in two situations: either it catches a cCANopenException or it consumes the event cCANopen.EventError via the appropriate event handler.

This enumeration is derived from the C enum type tCopKernel from the CANopen stack. It uses a similar naming scheme. The constants of enumCopKernel just use the prefix “k” instead of “kCop” as tCopKernel.

Table lists the most common used constants and their meaning. If you encounter other constants, please have a look in the CANopen User Manual L-1020 [2].

Constant	Description
kSuccessful	No error occurred.
kIllegalInstance	The CANopen instance does not exist or was already initialized.
kNoFreeInstance	The maximum number of CANopen instances has been reached.
kInvalidNodeId	An invalid node ID was specified.
kNoResource	A resource of the operating system could not be created.
kInvalidParam	Invalid parameters were specified.
kCdrvInitError	An error occurred while initializing the CAN driver (e.g. the selected hardware is not present).
kCdrvInvalidDriverType	An invalid driver type (tCdrvWinParam:: m_VxDType) was requested. This error may be issued if the requested hardware is not present, no free device is available or SO-1088 is used with an unauthorized type.
kCdrvDriverNotFound	The necessary driver DLL (e.g.

Constant	Description
	USBCAN32.DLL or ETHCAN.DLL) was not found.
kCdrvInvalidDevNumber	An invalid device number was specified.
kCdrvDevAlreadyInUse	The device which was selected is already in use.
kCobAlreadyExist	The requested COB-ID exists already. This may occur if a SDO client to the very same SDO server has been created before and was not disposed yet.
kCobCdrvStateSet	The CAN driver changed its state.
kObdIllegalPart	The accessed part of the object dictionary is unknown.
kObdIndexNotExist	The specified object index does not exist.
kObdSubindexNotExist	The specified subindex does not exist in the object index.
kObdReadViolation	Reading of a write-only object is not allowed.
kObdWriteViolation	Writing of a read-only object is not allowed.
kObdAccessViolation	Access to the specified object is not allowed.
kNmtStateError	An error occurred in the NMT state machine.
kSdocInvalidParam	Invalid parameters were specified for the SDO client.
kSdocClientNotExist	The selected SDO client does not exist in the object dictionary.
kSdocBusy	The SDO client is busy, i.e. a transfer is already running.
kSdocNoFreeEntry	No free SDO client index available.
kHbcNoFreeEntry	No free heartbeat consumer entry available.
kLssmIllegalState	Method of cLSSMaster was called in illegal state, i.e. in wrong order. Some methods may be called only in LSS mode CONFIGURATION.

Table 2: Constants of enumCopKernel

7.2 SDO abort codes

The CANopen communication profile [1] and specifications based on it define several abort codes for SDO transfers. These abort codes may be sent by both communication partners. For convenience, some of them are explained below.

Value	Description
0	Transfer finished successfully.
0x05030000L	Toggle bit error.
0x05040000L	The SDO transfer timed out. Mostly the CANopen device is not available anymore or the connection is broken.
0x05040001L	Unknown command specifier.
0x05040002L	Invalid block size
0x05040003L	Invalid sequence number
0x05040004L	CRC error
0x05040005L	Out of memory
0x06010000L	Unsupported access
0x06010001L	Reading of a write-only object
0x06010002L	Writing of a read-only object
0x06020000L	Object does not exist
0x06040041L	Object is not mappable to PDO
0x06040042L	PDO length exceeded
0x06040043L	Generic parameter incompatibility
0x06040047L	Generic internal incompatibility
0x06060000L	Access failed due to hardware error
0x06070010L	Data type length does not match
0x06070012L	Data type length too high
0x06070013L	Data type length too low
0x06090011L	Sub-index of object does not exist
0x06090030L	Value range exceeded
0x06090031L	Value too high
0x06090032L	Value too low
0x06090036L	Maximum value is less than minimum value
0x060A0023L	Resource is not available
0x08000000L	General error
0x08000020L	Data not transferred or stored
0x08000021L	Data not transferred due to local control
0x08000022L	Data not transferred due to device state

Value	Description
0x08000023L	Object dictionary does not exist

Table 3: SDO abort codes

7.3 Exception cCANopenException

This exception inherits ApplicationException. It will be thrown whenever a CANopen stack function returns an enumCopKernel error code unequal to enumCopKernel.kSuccessful. This is the case if a severe error occurred and the function cannot continue the operation. The application should catch this exception whenever it calls a CANopen method. Otherwise the application would crash if such an exception was raised.

7.3.1 Field m_ErrorCode

The public field m_ErrorCode is of type enumCopKernel and represents the error code which was returned by the CANopen stack.

7.4 Class cCANopen

One object of this reference class represents one CANopen instance. It creates all related objects like cSDO, cOD, cNMT, etc. An object of this class can be created directly by the application. Furthermore, it has to be disposed by the application if it is no longer used anymore. The disposing will shut down the CANopen instance including the associated CAN driver instance and release all resources (i.e. managed and unmanaged).

7.4.1 Constructors

Syntax C#:

```
public cCANopen(
        byte                    bLocalNodeId_p,
        enumCdrvBaudIndex       BaudIndex_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tCdrvWinParam       CdrvParam_p,
        enumCdrvBaudIndex       BaudIndex_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        enumCdrvBaudIndex       BaudIndex_p,
        bool                    fMaster_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tIdentParam         Identity_p,
        enumCdrvBaudIndex       BaudIndex_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tCdrvWinParam       CdrvParam_p,
        enumCdrvBaudIndex       BaudIndex_p,
        bool                    fMaster_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tIdentParam         Identity_p,
        enumCdrvBaudIndex       BaudIndex_p,
        bool                    fMaster_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tIdentParam         Identity_p,
        ref tCdrvWinParam       CdrvParam_p,
        enumCdrvBaudIndex       BaudIndex_p);
public cCANopen(
        byte                    bLocalNodeId_p,
        ref tIdentParam         Identity_p,
        ref tCdrvWinParam       CdrvParam_p,
        enumCdrvBaudIndex       BaudIndex_p,
        bool                    fMaster_p);
```

Parameters:

bLocalNodeId_p: node ID of this CANopen instance

Identity_p:	identity of this CANopen device, e.g. device type, vendor ID, product code, etc. If not specified it defaults to the values of the object dictionary.
CdrvParam_p:	parameters for the SYS TEC CAN-Wrapper driver. If not specified an arbitrary USB-CANmodul will be used.
BaudIndex_p:	index of the baudrate which the CAN controller shall use.
fMaster_p:	indicates if this CANopen instance shall be NMT master (true) or slave (false). If not specified NMT master will be selected.

Return:

N/A.

Description:

Overloaded constructor, that creates a CANopen instance with the supplied parameters. The constructors are NOT thread-safe.

Field	Type	Description
m_dwDeviceType	int	Device type resp. profile (object 0x1000 of local OD)
m_dwVendorId	int	Vendor ID (object 0x1018/1 of local OD)
m_dwProductCode	int	Product code (object 0x1018/2 of local OD)
m_dwRevision	int	Revision number (object 0x1018/3 of local OD)
m_dwSerNum	int	Serial number (object 0x1018/4 of local OD)
m_sDevName	string	Device name (object 0x1008 of local OD)
m_sHwVersion	string	Hardware version (object 0x1009 of local OD)
m_sSwVersion	string	Software version (object 0x100A of local OD)

Table 4: Fields of tIdentParam

Field	Type	Description
m_VxDType	enumVxDType	CAN hardware type
m_ThreadPriority	enumThreadPriority	Priority of CAN-Wrapper thread

Field	Type	Description
m_bDeviceNr	byte	Device number (255 for an arbitrary device)
m_wIOBase	short	IO base address (only valid for ISA cards)
m_bIRQ	byte	IRQ (only valid for ISA cards)
m_IpAddress	Net::IPAddress	IP address (only valid for CAN-Ethernet-Gateway)
m_wIPPort	int	IP port in range 1 - 65535 (only valid for CAN-Ethernet-Gateway)
m_dwReconnectTimeout	int	Reconnect timeout (only valid for CAN-Ethernet-Gateway)
m_dwConnectTimeout	int	Connect timeout (only valid for CAN-Ethernet-Gateway)
m_dwDisconnectTimeout	int	Disconnect timeout (only valid for CAN-Ethernet-Gateway)
m_bIpProtocol	byte	IP protocol, 0 = TCP, 1 = UDP (only valid for CAN-Ethernet-Gateway)

Table 5: Fields of tCdrvWinParam

Constant	Description
kAutoDetect	Autodetect strategy
kPhyCAN	PhyCAN driver with pcNetCAN card
kPCAN_V1_ISA	PCAN 1.x driver with pcNetCAN card
kPCAN_V1_Dongle	PCAN 1.x driver with PCAN-Dongle
kUSBCAN	SYS TEC USB-CANmodul driver
kPCAN_PCI	PCAN-PCI driver with PCI card
kPCAN_V2_ISA	PCAN 2.x driver with pcNETCAN card
kPCAN_V2_Dongle	PCAN 2.x driver with PCAN-Dongle
kPCAN_V2_PCI	PCAN 2.x driver PCI card
kPCAN_Dongle	PCAN-Dongle-Driver
kPCAN_USB	PEAK USB-CAN-Modul
kSCANCONN_USB	SYSTEC CAN Connector with USB-CANmodul
kSCANCONN_ETH	SYSTEC CAN Connector with CAN Ethernet Gateway
kETHCAN	ETHERNET-CAN-GATEWAY-Driver
kPCAN_V2_USB	PCAN 2.x driver with PCAN-USB
kIniDetect	Read parameters from INI file

Table 6: Constants of enumVxDType

Constant	Description
kNormal	Normal thread priority
kHighest	Highest thread priority
kTimeCritical	Time critical thread priority

Table 7: Constants of enumThreadPriority

Constant	Description
k1MBaud	1 MBit/sec
k800kBaud	800 kBit/sec
k500kBaud	500 kBit/sec
k250kBaud	250 kBit/sec
k125kBaud	125 kBit/sec
k100kBaud	100 kBit/sec
k50kBaud	50 kBit/sec
k20kBaud	20 kBit/sec
k10kBaud	10 kBit/sec

Table 8: Constants of enumCdrvBaudIndex

7.4.2 Method Dispose()

Syntax C#:

public sealed override void **Dispose**();

Parameters:

N/A.

Return:

N/A.

Description:

The Dispose() method has to be called when this CANopen instance is no longer used anymore. This method shuts down this CANopen instance and releases all unmanaged resources.

7.4.3 Delegate EventErrorHandler()

Syntax C#:

```
public delegate void EventErrorHandler(
        object                  Sender_p,
        enumCopKernel           ErrorCode_p,
        object                  pArg_p);
```

Parameters:

Sender_p:	Sender of the error event, i.e. this object.
ErrorCode_p:	Error code from the CANopen stack
pArg_p:	Object which contains details of the error event. The class of the object depends on the error code.

Return:

N/A.

Description:

This is the delegate type for error events (EventError) from the CANopen stack.

7.4.4 Event EventError

Syntax C#:

```
public event EventErrorHandler EventError;
```

Description:

This event signals errors from the CANopen stack. Registered event handlers are called within the CANopen instance's process thread.

7.4.5 Method GetNMT()

Syntax C#:

public cNMT **GetNMT**();

Parameters:

N/A.

Return:

cNMT	Singleton object of either class cNMTMaster or cNMTSlave

Description:

This method returns the related cNMT object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.6 Method GetOD()

Syntax C#:

public cOD **GetOD**();

Parameters:

N/A.

Return:

cOD	Singleton object of class cOD

Description:

This method returns the related cOD object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.7 Method GetHeartbeatProducer()

Syntax C#:

public cHeartbeatProducer **GetHeartbeatProducer**();

Parameters:

N/A.

Return:

cHeartbeatProducer	Singleton object of class cHeartbeatProducer

Description:

This method returns the related cHeartbeatProducer object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.8 Method GetEmergencyConsumer()

Syntax C#:

public cEmergencyConsumer **GetEmergencyConsumer**();

Parameters:

N/A.

Return:

cEmergencyConsumer	Singleton object of class cEmergencyConsumer

Description:

This method returns the related cEmergencyConsumer object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.9 Method GetEmergencyProducer()

Syntax C#:

public cEmergencyProducer **GetEmergencyProducer**();

Parameters:

N/A.

Return:

cEmergencyProducer	Singleton object of class cEmergencyProducer

Description:

This method returns the related cEmergencyProducer object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.10 Method GetLSSMaster()

Syntax C#:

public cLSSMaster **GetLSSMaster**();

Parameters:

N/A.

Return:

cLSSMaster	Singleton object of class cLSSMaster

Description:

This method returns the related cLSSMaster object of this CANopen instance. This object must not be diposed. This method is thread-safe.

7.4.11 Method CreateCOB()

Syntax C#:

```
public cCOB CreateCOB(
        int                 dwCobId_p,
        enumCobType         CobType_p,
        object              pObject_p);
public cCOB CreateCOB(
        int                 dwCobId_p,
        enumCobType         CobType_p,
        object              pObject_p,
        int                 dwCycleTime_p);
```

Parameters:

dwCobId_p:	COB-ID of the CAN message
CobType_p:	Type of the CAN message
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte) with a maximum size of 8 bytes. Sending of boxed value types makes only sense if they can be updated without creating a new object (AFAIK this is only possible with C++/CLI). The only solution is to wrap value types in a new reference type and specify sequential layout.
dwCycleTime_p	Cycle time for cyclic Tx CAN messages specified in 100 µs.

Return:

cCOB	Newly created object of class cCOB

Description:

This method returns a cCOB object which was created with the supplied parameters. This object has to be disposed if it is no longer used anymore.

Constant	Description
kSend	standard CAN messages to send
kRecv	standard CAN messages to receive
kRmtSend	receive data as answer of RTR frame

Constant	Description
kRmtRecv	send data as answer of RTR frame
kForceSend	standard CAN messages to send and forced by received RTR frame
kForceRmtRecv	send data as answer of RTR frame and with CobSend(immediately=TRUE)
kCyclicSend	standard CAN message which is sent cyclically
kCyclicRmtSend	RTR frame which is sent cyclically
kFilter	filter for COB type
kExtended	extended CAN message (CAN2.0B)

Table 9: Constants of [Flags]enumCOBType

7.4.12 Method CreateHeartbeatConsumer()

Syntax C#:

```
public cHeartbeatConsumer CreateHeartbeatConsumer(
        byte                    bNodeId_p,
        short                   wHeartbeatTime_p);
```

Parameters:

bNodeId_p:	Node ID of the heartbeat producer
wHeartbeatTime_p:	Time of heartbeat in [ms] which should be larger than the one of the producer

Return:

cHeartbeatConsumer	Newly created object of class cHeartbeatConsumer

Description:

This method returns a cHeartbeatConsumer object which was created with the supplied parameters. This object has to be diposed if it is no longer used anymore.

7.4.13 Method CreateSDO()

Syntax C#:

```
public cSDO CreateSDO(
        byte                    bServerNodeId_p,
```

```
                short                   wClientIndex_p,
                int                     dwRxCanId_p,
                int                     dwTxCanId_p);
```

Parameters:

bServerNodeId_p:	Destination node ID
wClientIndex_p:	SDO client index to be used; 0 means that arbitrary client index will be used; other valid values are 0x1280 - 0x12FF
dwRxCanId_p:	receive CAN-ID; 0 means that default SDO server will be used
dwTxCanId_p:	transmit CAN-ID; 0 means that default SDO server will be used

Return:

cSDO	Newly created object of class cSDO

Description:

This method returns a cSDO object which was created with the supplied parameters. This object has to be diposed if it is no longer used anymore.

7.4.14 Method GetMaxInstances()

Syntax C#:

```
public static int GetMaxInstances();
```

Parameters:

N/A.

Return:

int	Number of supported object instances

Description:

This method returns the maximum supported numbers of CANopen instances of this assembly.

7.4.15 Method GetStackVersion()

Syntax C#:

```
public static void GetStackVersion(
        ref tVersion                    Version_p);
```

Parameters:

Version_p:	Contains the version number in format m_bMajor.m_bMinor.m_wRelease.

Return:

N/A.

Description:

This method returns the version number of the CANopen stack.

Field	Type	Description
m_bMajor	byte	Major version number
m_bMinor	byte	Minor version number
m_wRelease	short	Release number

Table 10: Fields of tVersion

7.5 Class cNMT

This abstract reference class models the local NMT state machine.

7.5.1 Delegate EventNmtHandler()

Syntax C#:

```
public delegate void EventNmtHandler(
        enumNMTEvent                    NmtEvent_p,
        enumNMTState                    NmtState_p);
```

Parameters:

NmtEvent_p:	Occured NMT event
NmtState_p:	Current NMT state

Return:

N/A.

Description:

This is the delegate type for local NMT events which result in NMT state changes.

Constant	Description
kEnterInitialising	Initialize NMT state machine
kResetNode	Reset node resp. application
kPreResetCommunication	Before entering reset communication
kResetCommunication	Reset communication
kPostResetCommunication	After reset communication
kEnterPreOperational	Enter NMT state PRE-OPERATIONAL
kEnterOperational	Enter NMT state OPERATIONAL
kEnterStopped	Enter NMT state STOPPED

Table 11: Constants of enumNMTEvent

Constant	Description
kInitialisation	NMT state INITIALISATION
kPreOperational	NMT state PRE-OPERATIONAL
kOperational	NMT state OPERATIONAL
kStopped	NMT state STOPPED

Table 12: Constants of enumNMTState

7.5.2 Event EventNmt

Syntax C#:

public event EventNmtHandler **EventNmt**;

Description:

This event signals local NMT state changes. Registered event handlers are called within the CANopen instance's process thread.

7.5.3 Delegate EventNmtSlaveHandler()

Syntax C#:

```
public delegate void EventNmtSlaveHandler(
        byte                        bNodeId_p,
        enumNMTErrorControlEvent    NmtmEvent_p,
        enumNMTState                NmtState_p);
```

Parameters:

bNodeId_p:	Node ID of the affected CANopen device
NmtmEvent_p:	Occured NMT error control event
NmtState_p:	Most recently transmitted NMT state of the specified node

Return:

N/A.

Description:

This is the delegate type for NMT error control events which indicate changes of guarded nodes. This delegate is used by the cNMTMaster and cHeartbeatConsumer class.

7.5.4 Method ConnectToNet()

Syntax C#:

```
public virtual void ConnectToNet();
```

Parameters:

N/A.

Return:

N/A.

Description:

This is a virtual method which initializes this CANopen instances and drives the NMT state machine until PRE-OPERATIONAL. There must not be called any CANopen methods until the NMT state machine is in state PRE-OPERATIONAL.

This method is neither thread-safe nor reentrant.

7.5.5 Method BeginConnectToNet()

Syntax C#:

```
public IAsyncResult BeginConnectToNet(
        AsyncCallback                   delegateAsyncCallback_p,
        object                          pAsyncState_p);
```

Parameters:

delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndConnectToNet() within the delegate.
pAsyncState_p:	Associated application specific object

Return:

IAsyncResult	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method which initializes this CANopen instances and drives the NMT state machine until PRE-OPERATIONAL by calling the virtual method ConnectToNet(). There must not be called any CANopen methods until the NMT state machine is in state PRE-OPERATIONAL.

The only exception is the method EndConnectToNet(), which must be called afterwards.

This method is neither thread-safe nor reentrant.

7.5.6 Method EndConnectToNet()

Syntax C#:

public void **EndConnectToNet**(
IAsyncResult pAsyncResult_p);

Parameters:

pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginConnectToNet()

Return:

N/A.

Description:

This method waits until this CANopen instances is initialized and the NMT state is PRE-OPERATIONAL.

This method is NOT thread-safe, but reentrant.

7.6 Class cNMTMaster

This reference class which provides the NMT master functionality like controlling and guarding of NMT slave nodes. It is derived from the abstract class cNMT.

7.6.1 Event EventNmtSlave

Syntax C#:

public event cNMT.EventNmtSlaveHandler **EventNmtSlave**;

Description:

This event signals changes of the guarded slave node, e.g. boot-up, connection loss or NMT state changes.
This event is of the same delegate as cHeartbeatConsumer.EventHeartbeat. Registered event handlers are called within the CANopen instance's process thread.

7.6.2 Method AddSlaveNode()

Syntax C#:

```
public void AddSlaveNode(
        byte                    bNodeId_p);
```

Parameters:

bNodeId_p:	Slave node ID

Return:

N/A.

Description:

This method adds the specified node ID as slave node. After execution of this method boot-up events are forwarded for this node and guarding may be configured.

This method is thread-safe.

7.6.3 Method ConfigureLifeGuard()

Syntax C#:

```
public void ConfigureLifeGuard(
        byte                    bNodeId_p,
        ref tLifeGuardParam     LgParam_p);
```

Parameters:

bNodeId_p:	Slave node ID
LgParam_p:	Life guarding parameters (time and factor)

Return:

N/A.

Description:

This method configures the specified life guarding parameters for the slave node.

This method is thread-safe.

Field	Type	Description
m_wTime	short	Guard time in 1 [ms] that is the interval in which the slave is polled by the master.
m_bFactor	byte	Factor multiplied by the guard time gives the live time of the slave. If the slave does not respond within that time an event is raised.

Table 13: Fields of tLifeGuardParam

7.6.4 Method GetSlaveInfo()

Syntax C#:

```
public void GetSlaveInfo(
        byte                    bNodeId_p,
        ref tSlaveInfo          SlaveInfo_p);
```

Parameters:

bNodeId_p:	Slave node ID
SlaveInfo_p:	Slave node information like NMT state and guarding state

Return:

N/A.

Description:

This method returns some information about the specified slave node.

This method is thread-safe.

Field	Type	Description
m_bLostMsgCount	byte	Counter of lost messages, i.e. responses from the slave node.
m_NMTState	enumNMTState	Recently transmitted NMT state of the slave node.
m_fLgActive	bool	Indicates if life guarding is currently active.
m_fNgActive	bool	Indicates if node guarding is currently active. It will be set if a single node guard request was sent to the slave node. And it

Field	Type	Description
		will be reset if that request is responded.

Table 14: Fields of tSlaveInfo

7.6.5 Method SendCommand()

Syntax C#:

```
public void SendCommand(
        byte                    bNodeId_p,
        enumNMTCommand          Command_p);
```

Parameters:

bNodeId_p: Destination node ID; value 0 means all nodes including ourselves (except for NMT commands kResetNode and kResetCommunication); the local node ID is also valid

Return:

N/A.

Description:

This method sends the specified NMT command to the specified node. Except kResetNode and kResetCommunication for all nodes, these commands are also executed on this CANopen instance if applicable.

This method is thread-safe.

Constant	Description
kStartRemoteNode	Start remote node, i.e. enter OPERATIONAL.
kStopRemoteNode	Stop remote node, i.e. enter STOPPED.
kEnterPreOperational	Enter PRE-OPERATIONAL.
kResetNode	Reset Node.
kResetCommunication	Reset Communication.

Table 15: Constants of enumNMTCommand

7.6.6 Method TriggerNodeGuard()

Syntax C#:

```
public void TriggerNodeGuard(
        byte                    bNodeId_p);
```

Parameters:

bNodeId_p: Slave node ID

Return:

N/A.

Description:

This method triggers one node guard for this slave node.

This method is thread-safe.

7.6.7 Method DeleteSlaveNode()

Syntax C#:

```
public void DeleteSlaveNode(
        byte                    bNodeId_p);
```

Parameters:

bNodeId_p: Slave node ID

Return:

N/A.

Description:

This method deletes the specified node ID as slave node. After execution of this method for example no boot-up events are forwarded for this node.

This method is thread-safe.

7.7 Class cNMTSlave

This reference class which provides the NMT slave functionality. It is derived from the abstract class cNMT.

7.7.1 Delegate EventNmtCommandHandler()

Syntax C#:

public delegate void **EventNmtCommandHandler**(
enumNMTCommand NmtCommand_p);

Parameters:

NmtCommand_p: Received NMT command

Return:

N/A.

Description:

This is the delegate type for received NMT commands. If the event handler throws a cCANopenException the NMT command will be rejected and not processed.

7.7.2 Event EventNmtCommand

Syntax C#:

public event EventNmtCommandHandler **EventNmtCommand**;

Description:

This Event notifies the application about received NMT commands. If any event handler throws a cCANopenException the NMT command will be rejected and not processed. Registered event handlers are called within the CANopen instance's process thread.

7.7.3 Method BootNetwork()

Syntax C#:

public void **BootNetwork**();

Parameters:

N/A.

Return:

N/A.

Description:

This method sends the NMT command Start Remote Node to all nodes. Afterwards it enters itself the NMT state OPERATIONAL. This method may be used by NMT slave devices with simple startup capability.

This method is thread-safe.

7.8 Class cOD

This reference class models the local object dictionary. It provides methods for accessing the local object dictionary.

7.8.1 Method ReadObject()

Syntax C#:

```
public void ReadObject(
        int                     wIndex_p,
        byte                    bSubIndex_p,
        object                  pObject_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary

pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).

Return:

N/A.

Description:

This method reads the specified object from the local object dictionary to the content of the specified reference type.

This method is thread-safe.

7.8.2 Method ReadObject(String)

Syntax C#:

```
public void ReadObject(
        int         wIndex_p,
        byte        bSubIndex_p,
        ref string  pString_p,
        int         dwMaxStringSize_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pString_p:	Reference to newly created String which will contain the read value.
dwMaxStringSize_p:	Maximum size of the String that will be created.

Return:

N/A.

Description:

This method reads the specified object of type VSTRING from the local object dictionary to a newly created String object.

This method is thread-safe.

7.8.3 Method WriteObject()

Syntax C#:

```
public void WriteObject(
        int                     wIndex_p,
        byte                    bSubIndex_p,
        object                  pObject_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).

Return:

N/A.

Description:

This method writes the content of the specified reference type to the specified object of the local object dictionary.

This method is thread-safe.

7.8.4 Method WriteObject(String)

Syntax C#:

```
public void WriteObject(
        int                     wIndex_p,
        byte                    bSubIndex_p,
        string                  pString_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pString_p:	String which will be written.

Return:
N/A.

Description:
This method writes the specified String to the specified object of the local object dictionary.

This method is thread-safe.

7.9 Class cSDO

This reference class models one local SDO client. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

7.9.1 Delegate EventSdoFinishedHandler()

Syntax C#:

```
public delegate void EventSdoFinishedHandler(
        object          Sender_p,
        byte            bServerNodeId_p,
        object          pObject_p,
        enumSDOState    SdoState_p,
        int             dwAbortCode_p,
        int             dwTransmittedBytes_p);
```

Parameters:

pSender_p:	Sender of this event
bServerNodeId_p:	Node ID of the associated SDO server.
pObject_p:	Object which contains the received or sent data.
SdoState_p:	State of the SDO transfer.
dwAbortCode_p:	SDO abort code (0 if transfer finished successfully).
dwTransmittedBytes_p:	Number of Bytes which were transmitted by the SDO transfer.

Return:
N/A.

Description:

This is the delegate type for SDO finished events.

Constant	Description
kNotActive	No transfer active.
kRunning	Transfer is running.
kTxAborted	Transmission was aborted.
kRxAborted	Reception was aborted.
kFinish	Transfer has finished successfully.

Table 16: Constants of enumSDOState

7.9.2 Event EventSdoFinished

Syntax C#:

public event EventSdoFinishedHandler **EventSdoFinished**;

Description:

This event notifies the application that the SDO transfer has finished and passes the information of the finished SDO transfer to the application. Registered event handlers are called within the CANopen instance's process thread.

7.9.3 Method Dispose()

Syntax C#:

public sealed override void **Dispose**();

Parameters:

N/A.

Return:

N/A.

Description:

The Dispose() method has to be called when this SDO client is no longer used anymore. This method releases the corresponding SDO client index in the object dictionary.

This method is NOT thread-safe, but reentrant.

7.9.4 Method ReadObject()

Syntax C#:

```
public void ReadObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p);
public void ReadObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers. It defaults to enumSDOType.kAuto if not specified.

Return:

N/A.

Description:

This method reads the specified object from the associated SDO server to the content of the specified reference type. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

Constant	Description
kAuto	First try to use block transfer and if that is rejected by the SDO server use segmented transfer.
kSegment	Use segmented transfer.
kBlock	Use block transfer.

Table 17: Constants of enumSDOType

7.9.5 Method ReadObject(String)

Syntax C#:

```
public void ReadObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        int                 dwStringSize_p);
public void ReadObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        int                 dwStringSize_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
dwStringSize_p:	Maximum size which will be transferred.
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers. It defaults to enumSDOType.kAuto if not specified.

Return:
N/A.

Description:
This method reads the specified object of type VSTRING from the associated SDO server to a newly created String object. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.6 Method BeginReadObject()

Syntax C#:

```
public IAsyncResult BeginReadObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the transfer.

Description:

This method reads the specified object from the associated SDO server to the content of the specified reference type. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. The application must call the method EndReadObject() with the returned IAsyncResult afterwards. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.7 Method BeginReadObject(String)

Syntax C#:

```
public IAsyncResult BeginReadObject(
        int             wIndex_p,
        byte            bSubIndex_p,
        int             dwStringSize_p,
        enumSDOType     Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
dwStringSize_p:	Maximum size which will be transferred.
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the transfer.

Description:

This method reads the specified object of type VSTRING from the associated SDO server to a newly created String object. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. The application must call the method EndReadObject() with the returned IAsyncResult afterwards.Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.8 Method EndReadObject()

Syntax C#:

```
public int EndReadObject(
        IAsyncResult                pAsyncResult_p);
public int EndReadObject(
        ref byte                    bServerNodeId_p,
        ref object                  pObject_p,
        ref enumSDOState            SdoState_p,
        ref int                     dwTransmittedBytes_p,
        IAsyncResult                pAsyncResult_p);
```

Parameters:

bServerNodeId_p:	Node ID of the associated SDO server.
pObject_p:	Object which contains the received or sent data.
SdoState_p:	State of the SDO transfer.
dwTransmittedBytes_p:	Number of Bytes which were transmitted by the SDO transfer.
pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginConnectToNet()

Return:

int:	SDO abort code (0 if transfer finished successfully).

Description:

This overloaded method waits until the transfer is finished and returns the SDO abort code.

This method is NOT thread-safe, but reentrant.

7.9.9 Method WriteObject()

Syntax C#:

```
public void WriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p);
public void WriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers. It defaults to enumSDOType.kAuto if not specified.

Return:

N/A.

Description:

This method writes the content of the specified reference type to the specified object of the associated SDO server. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.10 Method WriteObject(String)

Syntax C#:

```
public void WriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        string              pString_p);
public void WriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        string              pString_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pString_p:	String which will be transferred
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers. It defaults to enumSDOType.kAuto if not specified.

Return:

N/A.

Description:

This method writes the specified String to the specified object of the associated SDO server. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.11 Method BeginWriteObject()

Syntax C#:

```
public IAsyncResult BeginWriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        object              pObject_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pObject_p:	Object of a blittable type (e.g. value type like primitive types, reference types which are laid out sequential or array of Byte).
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the transfer.

Description:

This method writes the content of the specified reference type to the specified object of the associated SDO server. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. The application must call the method EndWriteObject() with the returned IAsyncResult afterwards. Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.12 Method BeginWriteObject(String)

Syntax C#:

```
public IAsyncResult BeginWriteObject(
        int                 wIndex_p,
        byte                bSubIndex_p,
        string              pString_p,
        enumSDOType         Type_p);
```

Parameters:

wIndex_p:	Index of object dictionary. It actually only has a range from 0 to 65535. The data type int is used because of CLS compliance.
bSubIndex_p:	Subindex of object dictionary
pString_p:	String which will be transferred
Type_p:	Type of SDO transfer. This type is ignored if there are transferred only up to 4 bytes, because these transfers are performed as expedited transfers.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the transfer.

Description:

This method writes the specified String to the specified object of the associated SDO server. When the transfer finishes the event EventSdoFinished will be raised.

This method does not block until the transfer is finished. The application must call the method EndWriteObject() with the returned IAsyncResult afterwards.Via the same SDO client only one transfer can be executed at the same time.

This method is NOT thread-safe, but reentrant.

7.9.13 Method EndWriteObject()

Syntax C#:

```
public int EndWriteObject(
        IAsyncResult                pAsyncResult_p);
public int EndWriteObject(
        ref byte                    bServerNodeId_p,
        ref object                  pObject_p,
        ref enumSDOState            SdoState_p,
        ref int                     dwTransmittedBytes_p,
        IAsyncResult                pAsyncResult_p);
```

Parameters:

bServerNodeId_p:	Node ID of the associated SDO server.
pObject_p:	Object which contains the received or sent data.
SdoState_p:	State of the SDO transfer.
dwTransmittedBytes_p:	Number of Bytes which were transmitted by the SDO transfer.
pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginConnectToNet()

Return:

int:	SDO abort code (0 if transfer finished successfully).

Description:

This overloaded method waits until the transfer is finished and returns the SDO abort code.

This method is NOT thread-safe, but reentrant.

7.9.14 Method AbortTransfer()

Syntax C#:

```
public void AbortTransfer(
        int                         dwAbortCode_p);
```

Parameters:

dwAbortCode_p:	SDO abort code which will be transmitted to the SDO server.

Return:
N/A.

Description:
This method aborts the running transfer with the specified SDO abort code.

This method is NOT thread-safe, but reentrant.

7.10 Class cCOB

This class provides the functionality to send and receive plain CAN layer 2 messages, i.e. communication objects (COB). An instance of this class represents one communication object. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

7.10.1 Delegate EventReceivedHandler()

Syntax C#:

```
public delegate void EventReceivedHandler(
        cCOB                    pSender_p,
        object                  pObject_p);
```

Parameters:

pSender_p:	Sender of this event
pObject_p:	Object which contains the received data. If pObject_p is null, too less data was received.

Return:
N/A.

Description:
This is the delegate type for COB received events.

7.10.2 Event EventReceived

Syntax C#:

public event EventReceivedHandler **EventReceived**;

Description:

This event passes the data of the received CAN message to the application. Registered event handlers are called within the CANopen instance's process thread.

7.10.3 Property Time

Syntax C#:

public property int Time;

Description:

The property Time describes the cycle time of cyclic Tx CAN messages. In case of received CAN messages it is the timestamp when the CAN message was received.
The property Time is measured in units of 100 µs.

Please note: In the current version of the USB-CANmodul driver the timestamp has only a width of 24 bits and is measured in units of 1 ms. This means that the USB-CANmodul timestamp will wrap around after 16,777,215 ms = approx. 4.66 h. Hence the receive timestamp stored in the property Time will wrap around after 167,772,150 * 100 µs.

7.10.4 Method Dispose()

Syntax C#:

public sealed override void **Dispose**();

Parameters:

N/A.

Return:
N/A.

Description:
The Dispose() method has to be called when this SDO client is no longer used anymore. This method releases the corresponding message object in the CANopen instance.

This method is NOT thread-safe, but reentrant.

7.10.5 Method Send()

Syntax C#:
public void **Send**();

Parameters:
N/A.

Return:
N/A.

Description:
This method sends the associated object..

This method is thread-safe.

7.11 Class cHeartbeatConsumer

This reference class models one local heartbeat consumer. There may exist multiple instances which were created by the same cCANopen instance. The application is responsible for disposing each instance when it is no longer used.

7.11.1 Event EventHeartbeat

Syntax C#:

public event cNMT.EventNmtSlaveHandler **EventHeartbeat**;

Description:

This event signals changes of the heartbeat producer, e.g. first heartbeat, connection loss and NMT state changes. This event is of the same delegate as cNMTMaster.EventNmtSlave. Registered event handlers are called within the CANopen instance's process thread.

7.11.2 Method Dispose()

Syntax C#:

public sealed override void **Dispose**();

Parameters:

N/A.

Return:

N/A.

Description:

The Dispose() method has to be called when this heartbeat consumer is no longer used anymore. This method releases the corresponding sub-index in the object dictionary.

This method is NOT thread-safe, but reentrant.

7.11.3 Method Configure()

Syntax C#:

public void **Configure**(short wHeartbeatTime_p);

Parameters:

wHeartbeatTime_p:	Time of heartbeat in [ms] which should be larger than the one of the producer.

Return:
N/A.

Description:
This method changes the heartbeat time of this consumer, that means the consumer must receive a heartbeat from the producer within this time, otherwise an event is raised.

This method is thread-safe.

7.12 Class cHeartbeatProducer

This reference class models the local heartbeat producer.

7.12.1 Method Configure()

Syntax C#:
public void **Configure**(short wHeartbeatTime_p);

Parameters:

wHeartbeatTime_p:	Time of heartbeat in [ms] which should be less than the one of any consumer.

Return:
N/A.

Description:
This method changes the heartbeat time of the local producer.

This method is thread-safe.

7.13 Class cEmergencyConsumer

This reference class models the local emergency consumer.

7.13.1 Delegate EventEmergencyHandler()

Syntax C#:

```
public delegate void EventEmergencyHandler(
        byte                                bNodeId_p,
        short                               wErrorCode_p,
        byte                                bErrorReg_p,
        byte[]                              abUserCode_p);
```

Parameters:

bNodeId_p:	Node ID of the emergency producer.
wErrorCode_p:	Error code of the emergency.
bErrorReg_p:	Value of the error register of the emergency producer.
abUserCode_p:	Additional manufacturer specific information that is an array of Byte with length 5.

Return:

N/A.

Description:

This is the delegate type for emergency events, i.e. when the consumer receives emergency messages.

7.13.2 Event EventEmergency

Syntax C#:

```
public event EventEmergencyHandler EventEmergency;
```

Description:

This event signals received emergency messages. Registered event handlers are called within the CANopen instance's process thread.

7.13.3 Method AddNode()

Syntax C#:

```
public void AddNode(
        byte                                bNodeId_p);
```

Parameters:

bNodeId_p:	Node ID of an emergency producer. 0 adds all nodes except the local node ID as emergency producer.

Return:

N/A.

Description:

This method adds the specified node ID as an emergency producer. After execution of this method emergency messages are forwarded for this node.

This method is thread-safe.

7.13.4 Method DeleteNode()

Syntax C#:

```
public void DeleteNode(
        byte                    bNodeId_p);
```

Parameters:

bNodeId_p:	Node ID of an emergency producer. 0 deletes all nodes except the local node ID as emergency producer.

Return:

N/A.

Description:

This method deletes the specified node ID as an emergency producer.

This method is thread-safe.

7.14 Class cEmergencyProducer

This reference class models the local emergency producer.

7.14.1 Method Send()

Syntax C#:

```
public void Send(
        short                   wErrorCode_p,
        byte                    bErrorReg_p,
        byte[]                  abUserCode_p,
        short                   wAdditionalInfo_p);
```

Parameters:

wErrorCode_p:	Error code of the emergency.
bErrorReg_p:	Value of the error register.
abUserCode_p:	Additional manufacturer specific information that is an array of Byte with length 5.
wAdditionalInfo_p	Additional information that is stored in the predefined error field if that exists.

Return:
N/A.

Description:
This method sends an emergency message over the CAN bus.

This method is thread-safe.

7.15 Class cLSSMaster

This reference class which provides the LSS master functionality to configure LSS slaves.

7.15.1 Method SwitchModeGlobal()

Syntax C#:

```
public void SwitchModeGlobal(
        enumLSSMode                    LSSMode_p);
```

Parameters:
LSSMode_p: LSS mode, i.e. either kOperation or kConfiguration.

Return:
N/A.

Description:
This method switches the LSS mode to the specified one for all LSS slaves. The LSS mode does not correlate with the NMT state.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Constant	Description
kOperation	LSS mode OPERATION.
kConfiguration	LSS mode CONFIGURATION.

Table 18: Constants of enumLSSMode

7.15.2 Method BeginSwitchMode()

Syntax C#:

```
public IAsyncResult BeginSwitchMode(
        ref tLSSAddress                 LSSAddress_p,
        AsyncCallback                   delegateAsyncCallback_p,
        object                          pAsyncState_p);
```

Parameters:

LSSAddress_p:	LSS address of a CANopen device which shall be switched to LSS mode CONFIGURATION.
delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndSwitchMode() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of switching the LSS mode selectively for the specified LSS address to CONFIGURATION. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndSwitchMode() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Field	Type	Description
m_dwVendorId	uint	Vendor ID (object 0x1018/1)
m_dwProductCode	uint	Produkt code (object 0x1018/2)

Field	Type	Description
m_dwRevision	uint	Revision number (object 0x1018/3)
m_dwSerNum	uint	Serial number (object 0x1018/4)

Table 19: Fields of tLSSAddress

7.15.3 Method EndSwitchMode()

Syntax C#:

```
public bool EndSwitchMode(
        IAsyncResult                    pAsyncResult_p);
```

Parameters:

pAsyncResult_p: Object of interface IAsyncResult which was returned by the method BeginSwitchMode().

Return:

bool: true if succeeded, false if process timed out, i.e. no LSS slave with the specified LSS address responded the switch mode command.

Description:

This method waits until the process is finished and returns true if it completes successfully.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

7.15.4 Method BeginInquireIdentity()

Syntax C#:

```
public IAsyncResult BeginInquireIdentity(
        enumLSSInquiryService           Services_p,
        AsyncCallback                   delegateAsyncCallback_p,
        object                          pAsyncState_p);
```

Parameters:

Services_p: The specified services, i.e. identity values like vendor or node ID, will be inquired.

delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndInquireIdentity() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of inquiring the specified services from the LSS slave which is in configuration mode. It is only allowed that exactly one LSS slave is in configuration mode, when this method is called. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndInquireIdentity() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Constant	Description
kNone	No service selected.
kVendorId	Inquire vendor ID.
kProductCode	Inquire product code.
kRevision	Inquire revision number.
kSerNum	Inquire serial number.
kAll	All services selected.

Table 20: Constants of [Flags]enumLSSInquiryService

7.15.5 Method EndInquireIdentity()

Syntax C#:

```
public bool EndInquireIdentity(
        ref tLSSAddress          LSSAddress_p,
        ref byte                 bNodeId_p,
        IAsyncResult             pAsyncResult_p);
```

Parameters:

pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginInquireIdentity().

Return:

bool:	true if succeeded, false if process timed out, i.e. no LSS slave responded the inquire commands.

Description:

This method waits until the process is finished and returns true if it completes successfully.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

7.15.6 Method BeginConfigure()

Syntax C#:

```
public IAsyncResult BeginConfigureSlave(
        byte                     bNodeId_p,
        bool                     fStore_p,
        AsyncCallback            delegateAsyncCallback_p,
        object                   pAsyncState_p);
```

Parameters:

bNodeId_p	Node ID which shall be configured. If it equals 0xFF it will not be configured.
fStore_p:	true if new configuration shall be stored on LSS slaves. false if configuration shall be changed temporarily only.
delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The

	delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndConfigureSlave() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of configuring the node ID of the LSS slave which is in configuration mode. It is only allowed that exactly one LSS slave is in configuration mode. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndConfigureSlave() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

7.15.7 Method BeginConfigure()

Syntax C#:

```
public IAsyncResult BeginConfigureSlave(
        byte                bNodeId_p,
        ref tLSSBitTiming   BitTiming_p,
        short               wSwitchDelay_p,
        bool                fStore_p,
        AsyncCallback       delegateAsyncCallback_p,
        object              pAsyncState_p);
```

Parameters:

bNodeId_p	Node ID which shall be configured. If it equals 0xFF it will not be configured.
BitTiming_p:	Reference to a value class which contains the bit timing.

wSwitchDelay_p:	Delay in [ms] between the configuration of the remote bit timing and the local activation of the new bit timing.
fStore_p:	true if new configuration shall be stored on LSS slaves. false if configuration shall be changed temporarily only.
delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndConfigureSlave() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of configuring the node ID of the LSS slaves which are in configuration mode. It is only allowed that exactly one LSS slave is in configuration mode, if the node ID shall be configured. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndConfigureSlave() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Field	Type	Description
m_bTableSelector	byte	Baudrate table selector (currently ignored, because only CiA BTR table is supported)
m_bTableIndex	byte	Index of baudrate table

Table 21: Fields of tLSSBiTiming

7.15.8 Method EndConfigureSlave()

Syntax C#:

```
public bool EndConfigureSlave(
        ref enumLSSConfigureState   State_p,
        ref byte                    bErrorCode_p,
        ref byte                    bSpecificErrorCode_p,
        IAsyncResult                pAsyncResult_p);
```

Parameters:

State_p:	State of the configuration. If this method returns false, this parameter contains the failed configure slave command.
bErrorCode_p:	Error code, which was passed by the LSS slave.
bSpecificErrorCode_p:	Specific error code, which was passed by the LSS slave. This is a manufacturer specific error code which is valid if bErrorCode_p equals 255.
pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginConfigureSlave().

Return:

bool:	true if succeeded, false if process timed out, i.e. no LSS slave responded the configure slave commands.

Description:

This method waits until the process is finished and returns true if it completes successfully.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Constant	Description
kIdle	Idle, i.e. no configuration active.
kNodeId	Configuration of node ID is/was active.
kConfigureBitTiming	Configuration of bit timing is/was active.
kActivateBitTiming	Activation of bit timing is/was active.
kStore	Storing of configuration is/was active.

Table 22: Constants of enumLSSConfigureState

7.15.9 Method BeginIdentifySlave()

Syntax C#:

```
public IAsyncResult BeginIdentifySlave(
        AsyncCallback       delegateAsyncCallback_p,
        object              pAsyncState_p);
```

Parameters:

delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndIdentifySlave() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of identifying LSS slaves without a valid node ID. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndIdentifySlave() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

7.15.10 Method BeginIdentifySlave()

Syntax C#:

```
public IAsyncResult BeginIdentifySlave(
        ref tLSSIdentifyParam   IdentifyParam_p,
        AsyncCallback           delegateAsyncCallback_p,
        object                  pAsyncState_p);
```

Parameters:

IdentifyParam_p:	LSS address range. Vendor ID and product code are fixed, but revision and serial number may be ranges.
delegateAsyncCallback_p:	Delegate which will be called when the process has finished. Specifying a null reference is allowed. The delegate may be called within the CANopen instance's process thread. Therefore it is only allowed to call the CANopen method EndIdentifySlave() within the delegate.
pAsyncState_p:	Associated application specific object.

Return:

IAsyncResult:	Object of interface IAsyncResult, which may be used to wait asynchronously for the end of the process.

Description:

This method starts the process of identifying LSS slaves with the specified LSS address range. When the process finishes the specified AsyncCallback delegate will be called.

This method does not block until the process is finished. The application must call the method EndIdentifySlave() with the returned IAsyncResult afterwards.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Field	Type	Description
m_dwVendorId	uint	Vendor ID (object 0x1018/1)
m_dwProductCode	uint	Produkt code (object 0x1018/2)
m_dwRevisionLow	uint	Lower bound of revision number (object 0x1018/3)
m_dwRevisionHigh	uint	Upper bound of revision number (object 0x1018/3)
m_dwSerNumLow	uint	Lower bound of serial number (object 0x1018/4)
m_dwSerNumHigh	uint	Upper bound of serial number (object 0x1018/4)

Table 23: Fields of tLSSIdentifyParam

7.15.11 Method EndIdentifySlave ()

Syntax C#:

```
public bool EndIdentifySlave(
        IAsyncResult                    pAsyncResult_p);
```

Parameters:

pAsyncResult_p:	Object of interface IAsyncResult which was returned by the method BeginIdentifySlave().

Return:

bool:	true if succeeded, false if process timed out, i.e. either no LSS slave with the appropriate LSS address responded the identify slave command or no LSS slave without a valid node ID exists in the network.

Description:

This method waits until the process is finished and returns true if it completes successfully.

This method is NOT thread-safe. It is only reentrant for different CANopen instances.

Glossary

CiA	CAN in Automation international users' and manufacturers' group (www.can-cia.org)
CCM	CANopen controlling module
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
CTS	Common Type System
COB	Communication object
DCF	Device configuration file (generated by configuration tools)
DLL	Dynamic linked library
GAC	Global assembly cache
HMI	Human machine interface
LSS	Layer setting services
NMT	Network Management
node	an arbitrary CANopen device. Often a NMT slave
OD	CANopen object dictionary
PDO	Process Data Object
SDO	Service Data Object
SRD	SDO requesting device

References

[1] CANopen Application Layer and Communication Profile, CiA DSP301, Version 4.1 21. February 2006, CAN in Automation e.V.

[2] CANopen User Manual, Software Manual, L-1020, SYS TEC electronic GmbH, Greiz

[3] Microsoft Visual C/C++ Runtime 2005 SP 1 (it is included as redistributable package in Visual Studio 2005 Professional Edition or available from http://www.microsoft.com/downloads/details.aspx?FamilyID=200b2fd9-ae1a-4a14-984d-389c36f85647)

Document:	CANopen API for .NET
Document number:	L-1114e_3, preliminary Edition March 2010

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Return to: SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY
Fax : +49 (0) 36 61 / 62 79 99

Published by

SYS TEC ELECTRONIC

Ordering No. L-1114e_3
Printed in Germany