
CANopen[®]

API for C++, C#/.NET

Available as *open source* (GPL v.3.0) or *commercial* license



www.datalink.se

©Ulrik Hagström 2008-2012

Contents

1	Product introduction	4
2	Application news	5
2.1	2013-01-27: Stuttgart Universität	5
3	Licensing conditions	6
3.1	GPL v.3.0 license	6
3.1.1	What is 'CAN dispatcher Library'?	6
3.2	Commercial license	7
3.2.1	Available value packages	7
4	Supported hardware	8
4.1	Available CAN interface plug-ins.	8
4.1.1	Zanthic Technologies CAN4USB FX.....	9
4.1.2	Lawicel CANUSB / CAN232	9
4.1.3	KVASER CANLIB (USB, PCI, PC104, PC104+, PCMICA,WLAN)	10
4.1.4	Movimento Castor (USB).....	10
4.1.5	EMS Dr. Thomas Wünsche (USB, Ethernet, PCI, PC104)	11
4.1.6	PEAK System (PCAN-Light API)	11
4.1.7	MHS Elektronik GmbH (Tiny-CAN API)	12
4.1.8	Create your own plug-in?	13
5	Software layout structure for C++ applications	14
6	Software layout structure for C#/.NET applications	15
7	Getting started.	16
7.1	Reference .NET DLL (canopenlib_NET.dll) in your Visual Studio project	16
7.2	No need to include any namespace.	16
7.3	Communication classes overview - .NET.....	16
7.3.1	ClientSDO_NET	17
7.3.2	NMT_Master_NET	19
7.3.3	CanMonitor_NET	19
7.3.4	EmcyServer_NET.....	21
7.3.5	EmcyClient_NET.....	22
7.3.6	ReceivePDO_NET	23
7.3.7	ReceivePDO_NET.....	24

CANopen[®]

Is a registered Community Trademarks of CAN in Automation.

<http://www.can-cia.org>

Checkout the source code from our Subversion-server on:

<https://canopen.no-ip.org/svn/demo/trunk>

!! The server is only up and running between 08.00-22.00 CET !!

How to get the login credentials:

=====

Send an empty e-mail to "subversion@datalink.se" and you will
automatically receive an e-mail with username and password within
a few seconds.

Thank you,

Datalink Engineering team

SERVER IS UP AND RUNNING FROM 08.00-22.00 CET (Central European Time)

Support for PDO:s,
Emergency and Sync-
messages since April 2013!

1 Product introduction

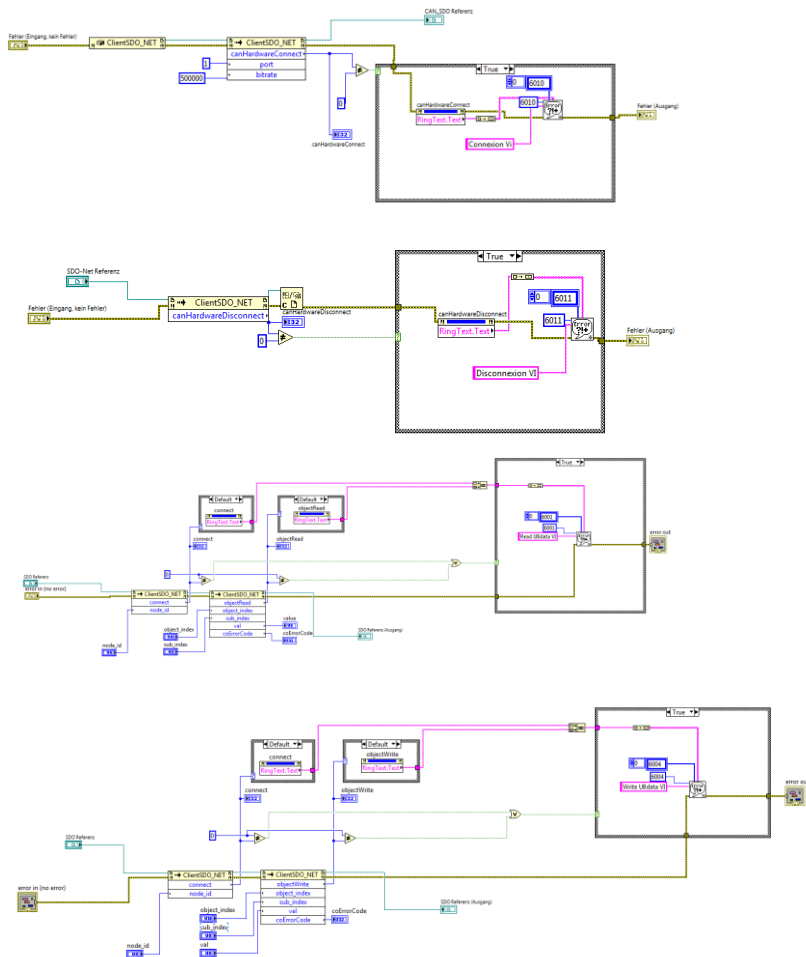
The purpose of this library is that it should be easy to create 3rd party CANopen[®] compliant applications in a high level language such as C# or C++. The library is well suited for helping develop diagnostic or tools for maintenance or service or developers. The driver supports;

- SDO expedited transfer (read and write 1-4 bytes in object dictionary of your CANopen node(s)).
- SDO segmented transfer (read and write typically strings in object dictionary of your CANopen node(s)).
- SDO block transfers (write only, use this for firmware upgrade).
- NMT Master Node control (start, stop, reset...)
- NMT Master error control (node-guard, heart-beat monitor)
- NMT Slave
- Transmit PDO
- Receive PDO
- Emergency producer
- Emergency consumer
- Sync message consumer
- **CAN raw mode (send and receive raw CAN frames), typically used to implement a CAN trace window in your application.**

2 Application news

2.1 2013-01-27: Stuttgart Universität

Stuttgart Universität is currently using this library to develop an application which controls servo controller for a test table in the institute for aeronautic construction. The application controls cylinder in position, speed and force. Labview's support for Microsoft .NET framework is being utilized.



3 Licensing conditions

CANopen API (C++/C#) is made available under the terms of the following dual license (like QT and OpenOffice):

3.1 GPL v.3.0 license

You may use CANopen API (C++/C#) in accordance with the terms of the GNU General Public License, version 3.0.

This license allows *for free use*, but also requires you to distribute your own code using CANopen API (C++/C#) under the terms of the GPL. See “[gpl-3.0.txt](#)” for complete information with additional permissions below since source code for “`can_dispatcher.lib`” is not included:

IMPORTANT: Additional permission under GNU GPL version 3 section 7:

If you modify this Program, or any covered work, by linking or combining it with 'CAN-dispatcher Library' (*`can_dispatcher.lib`*) (or a modified version of that library), containing parts covered by the terms of "CANopen API (C++/C#) Commercial License", the licensors of this Program grant you additional permission to convey the resulting work. {Corresponding Source for a non-source form of such a combination shall include the source code for the parts of 'CAN-dispatcher Library' used as well as that of the covered work.}

Read more about linking GPL-code to closed source here:

<http://www.gnu.org/licenses/gpl-faq.html#GPLIncompatibleLibs>
<http://www.gnu.org/licenses/gpl-faq.html#FSWithNFLibs>
<http://www.datalink.se/gpl-3.0.txt>

3.1.1 What is 'CAN dispatcher Library'?

The “CAN Dispatcher Library” (`can_dispatcher.lib`) is a vital library to make the CANopen driver package operate properly (and compile). Anyone committed to follow GPL v.3 are welcome to send us information about their project on sales@datalink.se and request the source code for the CAN-dispatcher Library. Please be very detailed on the project in the request.


3.2 Commercial license



In exchange for payment you may use CANopen API (C++/C#) in a closed-source application, in accordance with the terms of the CANopen API (C++/C#) Commercial License. CANopen API (C++/C#) Commercial License *also includes priority support*. **Commercial license also includes source code for `can_dispatcher.lib`**. Order on sales@datalink.se.

<http://www.datalink.se/license.txt>

3.2.1 Available value packages

Value package	Pricing and discounts	Price (Ex. VAT and shipping)
Commercial license only	1 commercial license only. 	200€
Lawicel commercial offer #1	1 commercial license + 5 Lawicel CANUSB (USB2.0 Mbits/s)      	549€
Lawicel commercial offer #2	1 commercial license + 1 Lawicel CANUSB (USB2.0 12 Mbits/s)  	245€
Kvaser commercial offer #1	1 commercial license + 5 Kvaser Leaf Light (USB2.0 480 Mbits/s)      	1349€
Kvaser commercial offer #2	1 commercial license + 1 Kvaser Leaf Light (USB2.0 480 Mbits/s)  	375€

These prices may change without prior notice – please ask for a quotation before ordering on



sales@datalink.se.



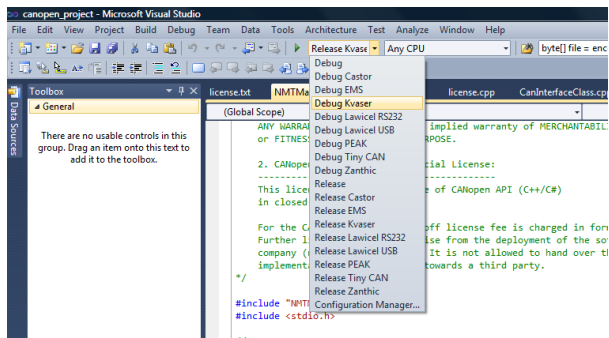
4 Supported hardware

The structure of the CANopen API (C++/C#) driver is that all CAN-layer dependent calls are made via a DLL called `canopenlib_hw.dll` (see chapter 4 in this document) – there are no direct dependencies from the CANopen® network to the CAN interface API.

This means that you can choose any of the listed hardware's below and just use the specific DLL for your hardware. You can also develop your own plug-in DLL; source code can be downloaded, free of charge for the listed hardware below as example how to implement your own plug-in for your preferred hardware. It has been kept in mind that this CANopen library should be possible to port to cheap CAN interfaces or even migrate to a embedded environment - without a fancy CAN API – therefore this library CAN-interface class holds a lot of intelligence (CAN message dispatcher, thread safe calls etc).

4.1 Available CAN interface plug-ins.

Today we can provide you with, free of charge, plug-in DLLs for the listed CAN interfaces below. You simply select your preferred hardware in the “Debug/Release” drop down in Visual Studio.



4.1.1 Zanthic Technologies CAN4USB FX

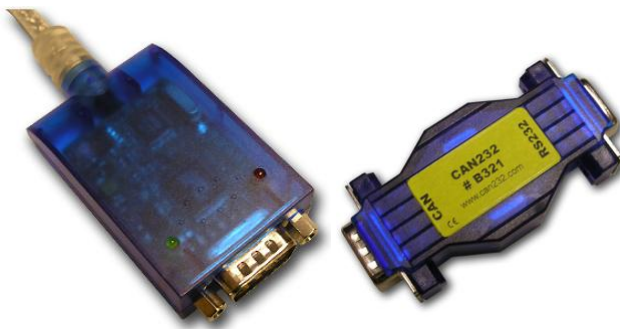
A plug-in for the *high performance* CAN4USBFX (<http://www.zanthic.com/can4usbfx.htm>) from Zanthic Technologies Inc is available which means that it will work with Zanthic Technologies USB adapters.



4.1.2 Lawicel CANUSB / CAN232

A plug-in for the Lawicel ASCII-API is available which means that it will work with both *low price* CANUSB and CAN232.

- CANUSB – http://www.canusb.com/documents/canusb_manual.pdf
- CAN232 – http://www.can232.com/can232_v3.pdf



4.1.3 KVASER CANLIB (USB, PCI, PC104, PC104+, PCMICA,WLAN)

All KVASER's hardware (www.kvaser.com) is using the same interface API – this means that you can choose any of the KVASER CAN-products as they are all supporting the CANLIB API.

Recommended Kvaser hardware is Kvaser Leaf Light!



4.1.4 Movimento Castor (USB)

Movimento Castor USB is low cost USB to CAN adapter. Contact distributors@datalink.se to reach all preferred retailing contacts and best price.

<http://www.movimentogroup.com/pdf/Movimento%20Castor%20CAN.pdf>



4.1.5 EMS Dr. Thomas Wünsche (USB, Ethernet, PCI, PC104)

All EMS hardware (<http://www.ems-wuensche.com>) is using the same interface API – this means that you can choose any of the EMS CAN-products to use with the EMS plug-in DLL.



Important notice: EMS windows drivers are not delivered with a free of charge API (SDK) – this means that you will have to purchase a proper developer license for your EMS device/devices to make the `canopen_hw.dll` working properly. **Contact EMS for further information about developer licenses.**

4.1.6 PEAK System (PCAN-Light API)

A plug-in for the PEAK System (www.peak-system.com) is available which means that it will work with all PEAK System hardware that is supporting the PCAN-Light API.



4.1.7 MHS Elektronik GmbH (Tiny-CAN API)

A plug-in for the Tiny-CAN II XL (www.mhs-elektronik.de) is available which means that it will work with all MHS System hardware that is supporting the Tiny-CAN API.



USB DRIVER NOTICE: Install the latest driver version **2.08.08** from the FTDI Homepage to get this adapter working stable on your OS – download: <http://www.ftdichip.com/Drivers/D2XX.htm> . The FTDI driver version 2.08.02 (WHQL Certified) doesn't work correctly on Windows 7 and Windows VISTA (Windows XP, or others?) is Datalink Engineering's experience.

To uninstall older/other driver packages you can use this application:
http://www.ftdichip.com/Support/Utilities/CDMUninstaller_v1.4.zip.

4.1.8 Create your own plug-in?

A number of 'C'-functions needs to be exported from canopenlib_hw.dll in order to make it work with higher layers of the CANopen® layer; they are described in the following list. Support for more than one handle per CAN channel is not required (a dispatcher in canopenlib.dll are dispatching one message to all objects that are listening to this CAN-port).

```
#define CAN_MSG_RTR           0x0001      // Message is a remote request
#define CAN_MSG_EXT           0x0002      // Message has a extended ID

canOpenStatus    __stdcall canPortLibraryInit(void);

canOpenStatus    __stdcall canPortOpen( int port, canPortHandle *handle );
canOpenStatus    __stdcall canPortClose( canPortHandle handle );

canOpenStatus    __stdcall canPortBtrRateSet( canPortHandle handle, int bitrate );

canOpenStatus    __stdcall canPortEcho( canPortHandle handle, bool enabled );

canOpenStatus    __stdcall canPortGoBusOn( canPortHandle handle );
canOpenStatus    __stdcall canPortGoBusOff( canPortHandle handle );

canOpenStatus    __stdcall canPortWrite(canPortHandle handle,
                                         long id,
                                         void *msg,
                                         unsigned int dlc,
                                         unsigned int flags);

canOpenStatus    __stdcall canPortRead(canPortHandle handle,
                                         long *id,
                                         void *msg,
                                         unsigned int *dlc,
                                         unsigned int *flags);
```

5 Software layout structure for C++ applications

Your C++ application in Visual Studio C++
canopenlib.lib (Visual Studio library file (Link)) + C++ header files .
canopenlib.dll (C++)
canopenlib_hw.dll ("Porting" for hardware being used, can be developed by you for your hardware.)
CAN hardware
CAN bus

6 Software layout structure for C#/.NET applications

Your C#/VB.NET application in Visual Studio.NET
canopen_NET.dll (.NET v.2.0 DLL) (Built with Common Language Infrastructure, CLI for Managed/Unmanaged handling)
canopenlib.dll (C++)
canopenlib_hw.dll (C) ("Porting" for hardware being used, can be developed by you for your hardware.)
CAN hardware
CAN bus

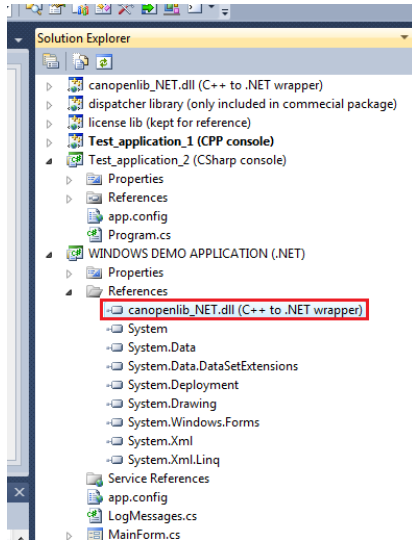
The syntax for the C# API version of this driver is very similar to the C++ and therefore left out in the examples. The appendix of this document includes a C# example program that should hopefully be sufficient.

7 Getting started.

This tutorial explains how to get started using the library together with Visual Studio 2010.

7.1 Reference .NET DLL (canopenlib_NET.dll) in your Visual Studio project

The .NET library file that needs to be included is the canopenlib_NET.dll. This DLL is simply added to your project by right-clicking the “References” folder of your C#/VB project and choose add. When successfully added it will look as follows:



7.2 No need to include any namespace.

The project that references the canopenlib.dll can create instances of the following classes without explicitly add any namespace; ClientSDO_NET, NMT_Master_NET and CanMonitor_NET. The ClientSDO_NET class is used for communication and configuration of CANOpen nodes (normally slaves) i.e. read from and writes to the object dictionary of the nodes. The NMT_Master_NET is used for network management of the network nodes (i.e. set them in different operational states and guard their presence and operational state). The CanMonitor_NET class is just easy access to the raw CAN-frames on the CAN-bus. The definitions of the classes are listed below.

7.3 Communication classes overview - .NET.

The classes of the library described more in detail in below. They have all in common that they connect to the very same CAN hardware channel but it will only be the first instance of any of the listed classes that will set the bit rate. It is also possible to create multiple instances of one single classes and let the asynchronously coexist beside other, simply no limitations to communicate with more than one node on the network at the very same time.

7.3.1 ClientSDO_NET

The ClientSDO-class is used for *write to* and *read from* the Object Dictionary of the nodes on the network. The class support expedited read/write, segmented read/write and block write.

7.3.1.1 Class overview

```
public class ClientSDO_NET : SDO_NET, IDisposable
{
    public ClientSDO_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public CANOPEN_LIB_ERROR.CanOpenStatus connect(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus connect(uint cobid_tx, uint cobid_rx);
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public bool isObjectReadResultCallbackEnabled();
    public bool isObjectWriteResultCallbackEnabled();
    public CANOPEN_LIB_ERROR.CanOpenStatus objectRead(ushort object_index, byte sub_index, out byte val, out uint
coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectRead(ushort object_index, byte sub_index, out uint val, out uint
coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectRead(ushort object_index, byte sub_index, out ushort val, out uint
coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectRead(ushort object_index, byte sub_index, byte[] val, out uint valid,
out uint coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectWrite(ushort object_index, byte sub_index, byte val, out uint coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectWrite(ushort object_index, byte sub_index, uint val, out uint coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectWrite(ushort object_index, byte sub_index, ushort val, out uint
coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectWrite(ushort object_index, byte sub_index, byte[] buf, uint valid,
out uint coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus objectWriteBlock(ushort object_index, byte sub_index, ushort crc, byte[] buf, uint
valid,
out uint coErrorCode);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerObjectReadResultCallback(CliReadResultDelegate readDelegate, object obj);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerObjectWriteResultCallback(CliWriteResultDelegate writeDelegate, object
obj);
    public void setNodeResponseTimeout(uint timeout);
    public void setReadObjectTimeout(uint timeout);
    public void setWriteObjectTimeout(uint timeout);
    public CANOPEN_LIB_ERROR.CanOpenStatus unregisterObjectReadWriteResultCallbacks();
}
```

7.3.1.2 Read from Object Dictionary example

Below is described how to initiate a read of a uint32 from object index = 0x1234 and sub index = 0x01 of node with id = 3.

```
ClientSDO_NET client_SDO;
client_SDO = new ClientSDO_NET();

stat = client_SDO.canHardwareConnect(0, 500000); //Port 0, bitr 500k.
if (stat != SDO_NET.CanOpenStatus.CANOPEN_OK)
    return -1; // Unexpected error.

stat = client_SDO.connect(3); // Connects client SDO to node 3.
if (stat != SDO_NET.CanOpenStatus.CANOPEN_OK)
    return -1; // Unexpected error.

SDO_NET.CanOpenStatus stat;
uint value;
uint error;

stat = client_SDO.objectRead(10, 10, out value, out error);
if (stat == SDO_NET.CanOpenStatus.CANOPEN_OK)
    return value; // Success, "value" contains read value.
```

7.3.1.3 Write to Object Dictionary example

If you want to write to (or read from) the object dictionary of a node you simply use the ClientSDO_NET class. Example below show a write attempt to object dictionary 0x1234 and subindex 0x12.

```
ClientSDO_NET client_SDO;
client_SDO = new ClientSDO_NET(); // Create instance.

CANOPEN_LIB_ERROR.CanOpenStatus stat;
stat = client_SDO.canHardwareConnect(0, 250000); // Connect CAN-hw ch 0, btr 250kbps.
if (stat != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
{
    // Something went wrong, is no CAN adapter connected?
}

stat = client_SDO.connect(12); // Connect to CANopen with node-id 12.
if (stat != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
{
    // Fatal error - could not connect to node 12.
}

uint error_code;
stat = client_SDO.objectWrite(0x1234, 0x12, 4, out error_code);
if (stat != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
{
    if (stat == CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_ERROR_HW_NOT_CONNECTED)
    {
        // Hardware error, is the adapter unplugged?
    }
    else
    {
        if (stat == CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_TIMEOUT)
        {
            // Write timeout - is node 12 powered off?
        }
        else
        {
            // Remote node didn't accept the value. Read node's error code in variable
            // variable error_code - see nodes documentation what this error code means.
        }
    }
}
else
{
    // Success!
}
```

7.3.2 NMT_Master_NET

The NMT_Master_NET class is used to network management. The library supports sending network management commands (i.e. start, stop, reset, etc. node).

7.3.2.1 Class overview

```
public class NMT_Master_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public NMT_Master_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int btr);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus heartbeatMonitorStart(byte node_id, uint heartbeat_consumer_time_ms);
    public CANOPEN_LIB_ERROR.CanOpenStatus heartbeatMonitorStop(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus NMTOperationalStateWrapperCPP(void* obj, byte node_id, byte state);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeGuardPollStart(byte node_id, uint node_life_time_ms);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeGuardPollStop(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodePreoperational(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeReadOperationalState(byte node_id, uint maxMsTimeout, out NodeState
node_state);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeReset(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeResetCommunication(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeStart(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeStop(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerNodeStateCallback(NMTOperationalStateDelegate opStateDelegate, object
obj);
}
```

7.3.2.2 Example how to start transmitting node-guard messages and verify node operational state.

```
class Program
{
    static void Main(string[] args)
    {
        NMT_Master_NET nmt_Master;
        nmt_Master = new NMT_Master_NET();
        nmt_Master.canHardwareConnect(0, 500000);
        nmt_Master.registerNodeStateCallback(
            new NMTOperationalStateDelegate(node_state_callback), (Object)nmt_Master);

        nmt_Master.nodeGuardPollStart(3, 3000); // Nodeguard node 3, node lifetime 3000ms
        nmt_Master.nodeGuardPollStart(5, 3000); // Nodeguard node 5, node lifetime 1500ms
        nmt_Master.nodeGuardPollStart(7, 3000); // Nodeguard node 5, node lifetime 1000ms

        ...
    }

    // Callback function that will be called from nmt_master object up on reception or no
    // reception of node guard response messages.

    static NMT_Master_NET.CanOpenStatus node_state_callback(object any, byte
node_id, byte state)
    {
        Console.WriteLine("Node State result : node_id: {0}, state: {1}", node_id, state);
        return NMT_Master_NET.CanOpenStatus.CANOPEN_OK;
    }
}
```

7.3.3 CanMonitor_NET

The CanMonitor_NET class is used to send and receive the raw CAN data ("CAN frames") on the bus.

7.3.3.1 Class overview

```
public class CanMonitor_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public CanMonitor_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public CANOPEN_LIB_ERROR.CanOpenStatus canWrite(uint id, byte[] data, byte dlc, uint flags);
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerCanReceiveCallback(object obj, CanReceiveDelegate can_receive_delegate);
}
```

7.3.3.2 Example how to send/receive raw CAN data (CAN frames) on hardware channel

```

class Program
{
    static void Main(string[] args)
    {
        CanMonitor.NET can_monitor;
        can_monitor = new CanMonitor.NET();
        can_monitor.registerCanReceiveCallback(
            (Object)this, new CanReceiveDelegate(canReceiveCallback));
        can_monitor.canHardwareConnect(0, 500000);
        ..

        ..

        byte[] data = new byte[8];

        data[0] = 0x01;
        data[1] = 0x02;
        data[2] = 0x03;
        data[3] = 0x04;
        data[4] = 0x05;
        data[5] = 0x06;
        data[6] = 0x07;
        data[7] = 0x018;

        can_monitor.canWrite(0x123, data, 4, CanMessageTypes.CAN_MSG_EXT_FLAG);
    }

    static CANOPEN_LIB_ERROR.CanOpenStatus canReceiveCallback(
        object obj, uint id, byte[] data, byte dlc, uint flags)
    {
        string temp_string;
        MainForm form = (MainForm)obj;

        temp_string = String.Format("0x{0:X3} {1}: ", id, dlc);
        if ( (flags & CanMessageTypes.CAN_MSG_RTR_FLAG) != 0 )
        {
            temp_string += "RTR";
        }
        else
        {
            for (int i = 0; i < dlc; i++)
            {
                temp_string += String.Format("{0:X2} ", data[i]);
            }
        }
        Console.Print(temp_string);
        return CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK;
    }
}

```

7.3.4 EmcyServer_NET

The EmcyServer_NET class is used to listen for and emergency messages on the network.

7.3.4.1 Class overview

```
public class EmcyServer_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public EmcyServer_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerEmcyServerMessageCallBack(object obj, EmcyServerDelegate
can_receive_delegate);
}
```

7.3.4.2 Example how receive emergency messages.

```
class Program
{
    static void Main(string[] args)
    {
        EmcyServer_NET emcy_server;

        emcy_server = new EmcyServer_NET();

        ret = emcy_server.canHardwareConnect(1, 500000);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = emcy_server.registerEmcyServerMessageCallBack((Object)emcy_server, new EmcyServerDelegate(emcy_callback));
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return

        ..
    }

    static EmcyServer_NET.CanOpenStatus emcy_callback(object obj, byte nodeId, ushort emcyErrorCode, byte errorRegister,
        byte[] manufacturerSpecificErrorField)
    {
        Console.WriteLine("EMERGENCY MESSAGE RECEIVED!");

        // Implement actions for the received emergency message here.

        return CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK;
    }
}
```

7.3.5 EmcyClient_NET

The EmcyClient_NET class is used to send network messages on the bus.

7.3.5.1 Class overview

```
public class EmcyClient_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public EmcyClient_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus nodeSetId(byte node_id);
    public CANOPEN_LIB_ERROR.CanOpenStatus sendEmcyMessage(ushort emcy_error_code, byte error_register,
        byte[] manufSpecificErrorField);
    public CANOPEN_LIB_ERROR.CanOpenStatus sendEmcyMessage(byte nodeId, ushort emcy_error_code, byte error_register,
        byte[] manufSpecificErrorField);
}
```

7.3.5.2 Example how send emergency messages.

```
class Program
{
    static void Main(string[] args)
    {
        EmcyClient_NET emcy_client;

        emcy_client = new EmcyClient_NET();

        ret = emcy_client.canHardwareConnect(0, 500000);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        byte[] manufSpecific = new byte[5];

        manufSpecific[0] = 0x10;
        manufSpecific[1] = 0x20;
        manufSpecific[2] = 0x30;
        manufSpecific[3] = 0x40;
        manufSpecific[4] = 0x50;

        ret = emcy_client.nodeSetId(3);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = emcy_client.sendEmcyMessage(0x10, 0x20, manufSpecific);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;
    }
}
```

7.3.6 ReceivePDO_NET

The ReceivePDO_NET class is used to listen to PDO data from the network messages on the bus.

7.3.6.1 Class overview

```
public class ReceivePDO_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public ReceivePDO_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus registerReceivePdoMessageCallBack(object obj, ReceivePdoDelegate
can_receive_delegate);
    public CANOPEN_LIB_ERROR.CanOpenStatus setCobid(uint cobid);
}
```

7.3.6.2 Example how receive PDO messages.

```
class Program
{
    static void Main(string[] args)
    {
        ReceivePDO_NET receive_pdo = new ReceivePDO_NET();

        ret = receive_pdo.setCobid(0x555);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = receive_pdo.registerReceivePdoMessageCallBack((Object)x, new ReceivePdoDelegate(rpdo_callback));
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = receive_pdo.canHardwareConnect(0, 500000);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ..
    }

    static ReceivePDO_NET.CanOpenStatus rpdo_callback(object x, UInt32 id, byte[] data, byte len)
    {
        // Handle the received PDO data here!

        return CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK;
    }
}
```

7.3.7 TransmitPDO_NET

The TransmitPDO_NET class is used to send PDO data to the network messages on the bus.

7.3.7.1 Class overview

```
public class TransmitPDO_NET : CANOPEN_LIB_ERROR, IDisposable
{
    public TransmitPDO_NET();

    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareConnect(int port, int bitrate);
    public CANOPEN_LIB_ERROR.CanOpenStatus canHardwareDisconnect();
    public override sealed void Dispose();
    protected virtual void Dispose(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus periodicTransmission(bool value);
    public CANOPEN_LIB_ERROR.CanOpenStatus setup(uint id, byte[] data, byte dlc);
    public CANOPEN_LIB_ERROR.CanOpenStatus transmit();
}
```

7.3.7.2 Example how send PDO messages (single and periodically).

```
class Program
{
    static void Main(string[] args)
    {
        TransmitPDO_NET transmit_pdo = new TransmitPDO_NET();

        ret = transmit_pdo.canHardwareConnect(1, 500000);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = transmit_pdo.setup(0x555, new byte[] { 1, 2, 3, 4, 5 }, 5);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        ret = transmit_pdo.transmit(); // Send one frame.
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        Thread.Sleep(1000);

        ret = transmit_pdo.periodicTransmission(true); // Send PDO periodically.
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        Thread.Sleep(5000);

        ret = transmit_pdo.setup(0x300, new byte[] { 8, 7, 6, 5, 4, 3, 2, 1 }, 8);
        if (ret != CANOPEN_LIB_ERROR.CanOpenStatus.CANOPEN_OK)
            return;

        Thread.Sleep(5000);
    }
}
```