CANopen Win API

Version 1.0.12

© TK Engineering Oy

Revision History

Date	Revision level	Author	Description
16.07.2014	1.0.0	SB/HL	Initial draft
04.08.2014	1.0.1	HL	Added COWA_AddLicense and example code
06.08.2014	1.0.2	HL	Added callbacks and updated example code
12.08.2014	1.0.3	HL	Updated the API according to latest code. Updated example code
18.08.2014	1.0.4	HL	Added supported network variables area
29.08.2014	1.0.5	HL	Restructured the document Added API constants and abbreviations
01.09.2014	1.0.6	HL	Updated function prototype COWA_TransmitEmcy()
04.09.2014	1.0.7	HL	Added COWA_INFO_CANOPENSTATUS
09.09.2014	1.0.8	HL	Added COWA_SetNodeInfo() Updated example
10.12.2014	1.0.9	HL	Updated contact information
17.12.2014	1.0.10	HL	Added hyper-links and more file description
21.11.2016	1.0.11	HL	Added COWA_GetRpdoStatus().
21.11.2016	1.0.12	HL	Added How to use network variables chapter

Support

Contact info@tke.fi for license

Contact support@tke.fi for technical support

Visit www.tke.fi for our products, service etc.

Important Information About This Guide

We believe that the information in this document was accurate at the time of printing. TK Engineering Oy cannot, however, assume responsibility for any errors or omissions in this document. The information in

this document is subject to change without notice and should not be taken as a commitment by TK Engineering Oy.

Table of Contents

1 Introduction

This document describes how to use CANopen Win API (COWA) DLL in your application. The available files are:

File	Description
CANopenWinAPI.h	The header file you need to include in your project
CANopenWinAPI32.dll	The 32-bit DLL
CANopenWinAPI64.dll	The 64-bit DLL for 64 bit application
CANopenWinAPI32.lib	The 32-bit static library
CANopenWinAPI64.lib	The 64-bit static library for 64 bit application

2 Abbreviation

Abbreviation	Definition
TKE	TK Engineering Oy
CiA	CAN in Automation
CAN	Controller Area Network
COWA	CANopen Win API
API	Application Programming Interface
OD	Object Dictionary
SDO	Service Data Object
Node id	Node identifier
PDO	Process Data Object
TPDO	Transmit PDO
RPDO	Receive PDO
EMCY	Emergency Object
NMT	Network Management

3 API Features

- CiA 301 compliant OD and NMT-master functionality
- CiA 302-4 compliant network variables
- CiA 302-5 compliant SDO manager(default SDO channels)
- Supports Kvaser hardware by default
- 10 TPDOs (object 0x1800 - 0x1809) and 10 RPDOs (object 0x1400 - 0x1409) are supported by default
- Supports RPDO timeout monitoring

- Supports 127 default client SDOs. The 128th client SDO is free to use
- One default SDO server is configured. 127 SDO server objects are free to use
- Supports network variable objects (range between object 0xA000 and 0xA8FF) with data type of INTEGER8, UNSIGNED8, INTEGER16, UNSIGNED16, INTEGER32 and UNSIGNED32. Each supported object has maximum 254 sub indexes
- One node is supported currently
- Supports both string based license and hardware license
- Contact us for customized features

4 API Functions

4.1 COWA_GetDllVersion()

unsigned long COWA_GetDllVersion();

Returns the version number of the CANopenWinAPI DLL. The bytes from left to right are:

1- Reserved

2- Major version

3- Minor version

4- Minor minor version

For example, If the version of the DLL is 1.2.3, the return value will be 0x00010203.

If the minor minor version is an odd number, then DLL is a developed version. If it is an even number, the DLL is a released version.

4.2 COWA_InitLibrary()

void COWA_InitLibrary();

Initializes internal data structures. **MUST** be called before any other functions in the API can be called.

Return nothing.

4.3 COWA_AddLicense()

cowaResult COWA_AddLicense(char * myLicense);

This function is only for **string based license**. If you have a license that is written to Kvaser hardware, this function can be ignored.

If you have a string based license, then this function **MUST** be the second call after calling COWA_InitLibrary(). Otherwise, no other functions will work. You can add only one license per calling. Call this function several times if you have more than one license.

Return COWA_OK as success. However, it doesn't mean the license is valid. The validation of the license will be reported in the following calls.

Return COWA_ERR_XXX on failure.

Contact us if you would like to purchase a license. Contact support team to get a 30-days demo license.

4.4 COWA_CreateNode()

cowaNodeHandle COWA_CreateNode();

Create a new node assigned to the first available CAN channel. Returns the handle to the node. Return -1 means invalid handle.

4.5 COWA_SetBitrate()

cowaResult COWA_SetBitrate(cowaNodeHandle handle, cowaBitrate bitrate);

Set the bitrate to the local node. Return COWA_OK on success, return COWA_ERR_XXX on failure. The call to this function will fail if the node was already running(i.e. after COWA_EnableNode() was called).

4.6 COWA_GetBitrate()

cowaResult COWA_GetBitrate(cowaNodeHandle handle, cowaBitrate *bitrate);

Get the birate of the local node. Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.7 COWA_SetNodeId()

cowaResult COWA_SetNodeId(cowaNodeHandle handle, uint8_t nodeId);

Set the node id of the local node. The valid node id is between 1 and 127. The call to this function will fail if the node was already running(i.e. after COWA_EnableNode() was called).

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.8 COWA_GetNodeId()

cowaResult COWA_GetNodeId(cowaNodeHandle handle, uint8_t *nodeId);

Get the node id of the local node. Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.9 COWA_SetNodeInfo()

cowaResult COWA_SetNodeInfo(cowaNodeHandle handle, uint8_t type, uint8_t *data, int32_t size);

This function is used to configure the basic information for the local node. e.g. Identity Object and Manufacture Device Name, Hareware Version and Software Version.

You **MUST** use this function to configure the node **before** calling COWA_EnableNode().

Arguments:

handle – The current handle of the node

type – One of COWA_INFO_XXX used for this function

data – Pointer to the buffer

size – The size of the buffer where data points to

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.10 COWA_EnableNode()

cowaResult COWA_EnableNode(cowaNodeHandle handle);

Enable a node on the first available channel. The node will go on bus with the configuted node id and bitrate. Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.11 COWA_SetLocalNmtState()

cowaResult COWA_SetLocalNmtState(cowaNodeHandle handle, cowaNmtCommand command);

Send the NMT command to the local node to enter specified NMT state.

Return COWA_OK as success, return COWA_ERR_XXX on failure.

4.12 COWA_SendNmtCommand()

cowaResult COWA_SendNmtCommand(cowaNodeHandle handle, uint8_t nodeId, cowaNmtCommand command);

Send the NMT command to the specified remote node on the network to enter specified NMT state.

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.13 COWA_GetNmtState()

cowaResult COWA_GetNmtState(cowaNodeHandle handle, int nodeId, cowaNmtState *state);

Get the NMT state of the specified node. If the nodeId is 0 or equal to the node id of the local node, the NMT state of the local node will be returned to *state. (see CiA314 version 0.0.3.5, section 6.5.2 Inquiring NMT state)

Otherwise, the status (operational state) of a node that is being monitored through the nodeguarding or heartbeat consumer protocols will be return.

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.14 COWA_GetNodeInfo()

COWA_GetNodeInfo(cowaNodeHandle handle, uint8_t type, int32_t *data);

Get runtime information about the node, e.g. CAN bus status.

Arguments:

handle – The current handle to the node

type – One of COWA_INFO_XXX that describes the information field we want to get.

data – Pointer to unsigned 32 bit variable that will get the data

For COWA_INFO_CANSTATUS, data will be the flag COWA_CANSTAT_XXX or the combination of them.

For COWA_INFO_CANOPENSTATUS, data will be the flag COWA_CANOPENSTAT_XXX or the combination of them.

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.15 COWA_GetRpdoStatus()

cowaResult COWA_GetRpdoStatus(cowaNodeHandle handle, uint16_t rpdoNumber, uint32_t *rpdoStatus);

Get the status of the monitored RPDO. e.g. if it's timeout or not.

Arguments:

handle – The current handle of the node

rpdoNumber – The number index of the RPDO. It should start from 1.

***rpdoStatus** – Pointer of the buffer to store the RPDO status flag. Check COWA_RPDO_XXX

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.16 COWA_ReadSdo()

cowaResult COWA_ReadSdo(cowaNodeHandle handle, uint8_t sdoNumber, uint16_t index, uint8_t subIndex, uint8_t * data, int32_t * size, uint32_t *sdoAbortCode);

The default SDO COB-ID is used in CANopenWinAPI. If sdoNumber is 0, read the Object Dictionary of the local node. The value will be stored in data. sdoAbortCode is assigned to an error code in case of error.

If sdoNumber is not 0, this function will send SDO read request of the Object Dictionary of the remote node.

Arguments:

handle – The current handle of the node

sdoNumber – 0 if access to local OD, other value if access remote OD

index – The index of the OD entry

subIndex – The sub index of the OD entry

data – Pointer to the buffer where the data will be stored

size – Size is both input and output. When calling the function size should contain the size (in bytes) of the memory pointed to by data. When the function returns successfully it will set size to the actual number of bytes stored in data.

sdoAbortCode – One of COWA_SDO_XXX. 0 if no error

Note: You need to check both sdoAbortCode and function return value. If the SDO abort message was received from the SDO server, then sdoAbortCode contains the abort code while function returns COWA_OK.

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.17 COWA_WriteSdo()

cowaResult COWA_WriteSdo(cowaNodeHandle handle, uint8_t sdoNumber, uint16_t index, uint8_t subIndex, uint8_t * data, int32_t size, uint32_t * sdoAbortCode);

If the sdoNumber is 0, write to the Object Dictionary of the local node. sdoAbortCode is assigned to an error code in case of error.

Otherwise, send the write access request of the Object Dictionary to the remote node.

Arguments:

handle – The current handle of the node

sdoNumber – 0 if access to local OD, other value if access remote OD

index – The index of the OD entry

subIndex – The sub index of the OD entry

data – Pointer to the buffer where the data will be written

size – The size(in bytes) of the buffer that **data** points to.

sdoAbortCode – One of COWA_SDO_XXX. 0 if no error or unspecified

Return COWA_OK on success, return COWA_ERR_XXX on failure. Remember to check the abort code when return value is not COWA_OK.

4.18 COWA_TransmitEmcy()

cowaResult COWA_TransmitEmcy(cowaNodeHandle handle, cowaEmcyErrorCode emcyErrorCode, uint16_t additionalInfo, uint8_t mb1, uint8_t mb2, uint8_t mb3, uint8_t mb4, uint8_t mb5);

Transmit an EMCY message to the CAN bus.

Arguments:

handle – The current handle of the node

emcyErrorCode – One of COWA_EMCY_XXX

additionalInfo – The additional information of the error history field in object 0x1003. See CiA-301 for more info

mb1 - mb5 – The 5 bytes of manufacture specific error code. See CiA-301 v 4.2.0 section 7.2.7.3 Emergency object protocol for more info

Return COWA_OK as function was called successfully, return COWA_ERR_XXX on failure.

Note: There is no guarantee that the EMCY message will be successfully transferred to the CAN bus even though it returns COWA_OK

4.19 COWA_CloseNode()

cowaResult COWA_CloseNode(cowaNodeHandle handle);

Goes off bus, closes CAN channel, frees handle. The handle will become invalid.

Return COWA_OK on success, return COWA_ERR_XXX on failure.

4.20 COWA_CloseLibrary()

void COWA_CloseLibrary();

Releases memories and frees all resources. **Must be the last function to be called.** Need to call **COWA_InitLibrary**() again to use the library.

5 API Constants

5.1 Definition of cowaResult

```
typedef int cowaResult;  // Result code for function calls
#define COWA_OK                          0   // OK, no errors
#define COWA_ERR_PARAM_RANGE             -1  // Parameter out of range
#define COWA_ERR_INVHANDLE               -2  // Handle is invalid
#define COWA_ERR_INTERNAL                -3  // Internal error
#define COWA_ERR_LIB_NOT_INITIALIZED     -4  // Library is not initialized
#define COWA_ERR_PARAM                   -5  // Bad parameter passed to the function
#define COWA_ERR_NODE_IS_RUNNING         -6  // The node is already started
#define COWA_ERR_NODE_NOT_ENABLED        -7  // The node is not started
#define COWA_ERR_NODE_NOT_CREATED        -8  // The node is not created yet
#define COWA_ERR_OTHER                   -9  // Other error
#define COWA_ERR_NO_CAN_HW               -10 // No CAN hardware channel
#define COWA_ERR_SW_BUFF_FULL            -11 // Internal software buffer is overflow
#define COWA_ERR_LICENSE                 -12 // No valid license
#define COWA_ERR_UNDEFINED_SDO_NO        -13 // The SDO number is undefined
#define COWA_ERR_BUSY                    -14 // The resource is in use, try again
                                             // later
#define COWA_ERR_TIMEOUT                 -15 // A timeout occurred
#define COWA_ERR_NOTSUPPORTED            -16 // Not supported
#define COWA_ERR_PDO_NOTFOUND            -17 // The PDO is not found


/* CAN error code */
#define COWA_ERR_CAN                     -17 // General CAN error
#define COWA_ERR_CAN_NOMSG               -18 // No CAN message to read
#define COWA_ERR_CAN_OVERFLOW            -19 // Receive message buffer is overflow
#define COWA_ERR_CAN_HARDWARE            -20 // CAN hardware error
#define COWA_ERR_CAN_TIMEOUT             -21 // CAN timeout
```

5.2 Definition of bus speed

```
typedef enum {
      COWA_BITRATE_1M = 0,
      COWA_BITRATE_800K = 1,
      COWA_BITRATE_500K = 2,
      COWA_BITRATE_250K = 3,
      COWA_BITRATE_125K = 4,
      COWA_BITRATE_100K = 5,
      COWA_BITRATE_50K = 6,
      COWA_BITRATE_25K = 7
} cowaBitrate;
```

5.3 Definition of cowaNmtCommand

```
typedef enum {
      COWA_NMT_CMD_START = 1,  // PreOperational -> Operational
      COWA_NMT_CMD_STOP = 2,   // -> Stopped
      COWA_NMT_CMD_ENTER_PREOP = 128,  // -> PreOperational State
      COWA_NMT_CMD_RESET_NODE = 129,       // -> Reset Node
      COWA_NMT_CMD_RESET_COMM = 130  // -> Reset Communication
} cowaNmtCommand;
```

5.4 Definition of cowaNmtState

```
typedef enum {
      COWA_NMT_STATE_BOOTUP = 0,  // Initialisation
      COWA_NMT_STATE_STOPPED = 4, // Stopped
      COWA_NMT_STATE_OPERATIONAL = 5, // Operational
      COWA_NMT_STATE_PREOP = 127    // Pre-operational
      COWA_NMT_STATE_UNKNOWN = 255  // Sate unknown is only used internally
} cowaNmtState;
```

5.5 Definition of cowaEmcyErrorCode

```
typedef enum{
      COWA_EMCY_NO_ERRORS                          =   0x0000,
      COWA_EMCY_ERROR_RESET                        =   0x0000,
      COWA_EMCY_GENERIC_ERRORS                     =   0x1000,
      COWA_EMCY_CURRENT_ERRORS                     =   0x2000,
      COWA_EMCY_CURRENT_DEVICE_INPUT_ERRORS        =   0x2100,
      COWA_EMCY_CURRENT_INSIDE_DEVICE_ERRORS       =   0x2200,
      COWA_EMCY_CURRENT_DEVICE_OUTPUT_ERRORS       =   0x2300,
      COWA_EMCY_VOLTAGE_ERRORS                     =   0x3000,
      COWA_EMCY_VOLTAGE_MAINS_ERRORS               =   0x3100,
      COWA_EMCY_VOLTAGE_INSIDE_DEVICE_ERRORS       =   0x3200,
      COWA_EMCY_VOLTAGE_OUTPUT_ERRORS              =   0x3300,
      COWA_EMCY_TEMPERATURE_ERRORS                 =   0x4000,
      COWA_EMCY_TEMPERATURE_AMBIENT_ERRORS         =   0x4100,
      COWA_EMCY_TEMPERATURE_DEVICE_ERRORS          =   0x4200,
      COWA_EMCY_HARDWARE_ERRORS                    =   0x5000,
      COWA_EMCY_SOFTWARE_ERRORS                    =   0x6000,
      COWA_EMCY_SOFTWARE_INTERNAL_ERRORS           =   0x6100,
      COWA_EMCY_SOFTWARE_USER_ERRORS               =   0x6200,
      COWA_EMCY_SOFTWARE_DATA_SET_ERRORS           =   0x6300,
      COWA_EMCY_ADDITIONAL_MODULES_ERRORS          =   0x7000,
      COWA_EMCY_MONITORING_ERRORS                  =   0x8000,
      COWA_EMCY_MONITORING_COMMUNICATION_ERRORS    =   0x8100,
      COWA_EMCY_MONITORING_COM_CAN_OVERRUN_ERROR   =   0x8110,
      COWA_EMCY_MONITORING_COM_CAN_PASSIVE_ERROR   =   0x8120,
      COWA_EMCY_MONITORING_COM_LIFE_GUARD_HB_ERROR =   0x8130,
      COWA_EMCY_MONITORING_COM_BUS_RECOVERY_ERROR  =   0x8140,
      COWA_EMCY_MONITORING_COM_COBID_COLLISION_ERROR        = 0x8150,
      COWA_EMCY_MONITORING_PROTOCOL_ERRORS                  = 0x8200,
      COWA_EMCY_MONITORING_PROTOCOL_LENGTH_ERROR            = 0x8210,
      COWA_EMCY_MONITORING_PROTOCOL_LENGTH_EXCEEDED_ERROR   = 0x8220,
      COWA_EMCY_EXTERNAL_ERRORS                             = 0x9000,
      COWA_EMCY_ADDITIONAL_FUNCTION_ERRORS                  = 0xF000,
      COWA_EMCY_DEVICE_SPECIFIC_ERRORS                      = 0xFF00
}cowaEmcyErrorCode;
```

5.6 Definition of SDO Abort Code

```
#define COWA_SDO_TOGGLE_BIT_NOT_ALTERED                0x05030000
/* Toggle bit not altered */

#define COWA_SDO_PROTOCOL_TIMED_OUT                    0x05040000
/* SDO protocol timed out */

#define COWA_SDO_UNKNOWN_CMD                           0x05040001
/* Client/server command specifier not valid or unknown */

#define COWA_SDO_UNSUPPORTED_ACCESS_TO_OBJECT          0x06010000
/* Unsupported access to an object */

#define COWA_SDO_OBJECT_READ_ONLY                      0x06010002
/* Attempt to write a read-only object */

#define COWA_SDO_OBJECT_WRITE_ONLY                     0x06010001
/* Attempt to read a write-only object */

#define COWA_SDO_OBJECT_DOES_NOT_EXISTS                0x06020000
/* Object does not exists in the Object Dictionary */

#define COWA_SDO_OBJECT_LENGTH                         0x06070010
/* Data type does not match, length of service parameter does not match */

#define COWA_SDO_SERVICE_PARAM_TOO_HIGH                0x06070012
/* Data type does not match, length of service parameter too high */

#define COWA_SDO_SERVICE_PARAM_TOO_LOW                 0x06070013
/* Data type does not match, length of service parameter too low */

#define COWA_SDO_INVALID_SUBINDEX                      0x06090011
/* Sub-index does not exist */

#define COWA_SDO_OBJDICT_DYN_GEN_FAILED                0x08000023
/* Object dictionary dynamic generation fails or no object dictionary is present */

#define COWA_SDO_GENERAL                               0x08000000
/* General error */

#define COWA_SDO_GENERAL_PARAMETER_INCOMPATIBILITY     0x06040043
/* General paramter internal incompabilitity */

#define COWA_SDO_GENERAL_INTERNAL_INCOMPABILITY        0x06040047
/* General internal incompatibility in the device */

#define COWA_SDO_NOT_MAPABLE                           0x06040041
/* Try to map un-mapable object to PDO. */

#define COWA_SDO_DATA_CAN_NOT_BE_STORED                0x08000020
/* the requested data can not be stored */

#define COWA_SDO_OUT_OF_MEMORY                         0x05040005
/* Node is out of memory. */

#define COWA_SDO_PDO_LENGHT                            0x06040042
/* Leght of PDO is exceeded  */

#define COWA_SDO_VAL_RANGE_FOR_PARAM_EXCEEDED          0x06090030
/* Param out of range. */
```

```
#define COWA_SDO_VALUE_TOO_HIGH                          0x06090031
/* Value of parameter written too high (download only). */

#define COWA_SDO_VALUE_TOO_LOW                           0x06090032
/* Value of parameter written too low (download only). */

#define COWA_SDO_ABORT_SDO_TRANSFER                      0x06060000
/* Abort SDO transfer */
```

5.7 Definition of Flags used in COWA_GetNodeInfo()

```
// Data types for COWA_GetNodeInfo
#define COWA_INFO_CANSTATUS 1
#define COWA_INFO_CANOPENSTATUS 2


// Flags for CAN status for function COWA_GetNodeInfo with type COWA_INFO_CANSTATUS
#define COWA_CANSTAT_ERROR_PASSIVE  1
#define COWA_CANSTAT_BUS_OFF        2
#define COWA_CANSTAT_ERROR_WARNING  4
#define COWA_CANSTAT_ERROR_ACTIVE   8


/** Flags for CANopen status for function COWA_GetNodeInfo with type
COWA_INFO_CANOPENSTATUS. */
#define COWA_CANOPENSTAT_SDOABORT   1  // There was an SDO abort since the last time
we read the CANopen status. This flag is reset on every read
```

5.8 Definition used in COWA_SetNodeInfo()

```
/** Data types used in COWA_SetNodeInfo(). */
#define COWA_INFO_MFG_DEVICENAME       3
#define COWA_INFO_MFG_HW_VERSION       4
#define COWA_INFO_MFG_SW_VERSION       5
#define COWA_INFO_VENDORID             6
#define COWA_INFO_PRODUCTCODE          7
#define COWA_INFO_REVNUM               8
#define COWA_INFO_SERIALNUM            9
```

5.9 Definition used in COWA_GetRpdoStatus()

```
#define COWA_RPDO_TIMEOUT   1 // The monitored RPDO is timeout
```

6 How to use Network Variables

The network variable objects (range between object 0xA000 and 0xA8FF) with data type of INTEGER8, UNSIGNED8, INTEGER16, UNSIGNED16, INTEGER32 and

UNSIGNED32 can be accessed by the application through API function COWA_ReadSdo() and COWA_WriteSdo() with sdoNumber set to 0.

7 Example

Below is a simple example that shows you the basic sequence to use the CANopen Win API.

```
//...
#include "CANopenWinAPI.h"
//...

int ExampleCode(){

	cowaResult res;
	cowaNodeHandle myHandle;
	cowaBitrate myBitrate = COWA_BITRATE_250K;
	uint8_t myNodeId = 2;
	uint8_t NodeA_Id = 4;



	COWA_InitLibrary();
	printf("\n Init library done!\n");

	/* Add License for string based license, if you have Kvaser
	hardware licnese, this function can be ignored */
	char *myLicense = "E24393C 3DCDB71C B8CE377E 709C0602 2912006A|0|
	10078";
	res = COWA_AddLicense( myLicense);
	if( res != COWA_OK){
		// handle error.
		printf("\n COWA_AddLicense failed. res = %d!\n", res);
		return -1;
	}
	printf("\n Add License success!\n");



	myHandle = COWA_CreateNode();
	if( myHandle < 0 ){
		// handle error.
		printf("\n COWA_CreateNode failed. res = %d!\n", myHandle);
		return -3;
```

```
}
printf("\n Create Node success!\n");

res = COWA_SetBitrate( myHandle, myBitrate);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_SetBitrate failed. res = %d!\n", res);
      return -4;
}
printf("\n Set Bitrate success!\n");

res = COWA_SetNodeId( myHandle, myNodeId);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_SetNodeId failed. res = %d!\n", res);
      return -5;
}
printf("\n Set Node ID success!\n");

char deviceName[] = "CANopen DLL";
int32_t len = (int32_t)strlen(deviceName);
res = COWA_SetNodeInfo( myHandle, COWA_INFO_MFG_DEVICENAME,
(uint8_t *)&deviceName, len);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_SetNodeInfo failed. res = %d!\n", res);
      return -6;
}
printf("\n Set Device Name success!\n");


res = COWA_EnableNode( myHandle);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_EnableNode failed. res = %d!\n", res);
      return -7;
}
printf("\n Enable Node success, node goes on CAN-bus!\n");

int32_t canStatus;
```

```
res = COWA_GetNodeInfo( myHandle, COWA_INFO_CANSTATUS,
&canStatus);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_GetNodeInfo failed. res = %d!\n", res);
      return -8;
}
printf("\n Get CAN status success! CAN status = %d\n", canStatus);


/* Set/Get Local NMT State Example */

cowaNmtState state;
res = COWA_GetNmtState( myHandle, myNodeId, &state);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_GetNmtState failed. res = %d\n", res);
      return -9;
}
printf("\n Get Local NMT state success!\n");
printf("\n Local NMT state is = %d\n", state);

res = COWA_SetLocalNmtState( myHandle, COWA_NMT_CMD_START);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_SetLocalNmtState failed. res = %d\n", res);
      return -10;
}
printf("\n Set Local NMT state success!\n");


int32_t size;
uint32_t err;
uint8_t data_8 = 0;
uint16_t data_16 = 0;
uint32_t data_32 = 0;

/* SDO read access to the local OD, the SDO number should be 0 */

data_32 = 0;
```

```
size = sizeof(data_32);
res = COWA_ReadSdo( myHandle, 0, 0x1280, 0x02, (uint8_t
*)&data_32, &size, &err);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_ReadSdo to local OD 0x1280 subIdx 0x02
failed. res = %d, err = %d\n", res, err);
      return -11;
}
printf("\n SDO read-access to local OD 0x1280 subIdx 0x02
success.\n");



/* SDO Write-access to local OD Example */


// Configure local OD 0x1016 consumer heartbeat time
size = 4;
data_32 = 0x403E8; // consumer heartbeat 1000 ms from node 4
res = COWA_WriteSdo( myHandle, 0, 0x1016, 0x01, (uint8_t
*)&data_32, size, &err);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_WriteSdo to local OD 0x1016 subIdx 0x01
failed. res = %d, err = %d\n", res, err);
      return -12;
}
printf("\n SDO write-access to local OD 0x1016 subIdx 0x01
success.\n");


// SDO write-access to remote OD : Configure Node A producer
heartbeat time 1000 ms
// Note : we are using Default client SDOs, therefore, sdoNumber
and the node id are the same
data_16 = 0x3E8;
res = COWA_WriteSdo( myHandle, 4, 0x1017, 0x00, (uint8_t
*)&data_16, 2, &err);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_WriteSdo to remote OD 0x1017 subIdx 0x00
failed. res = %d, err = %d\n", res, err);
      return -13;
}
```

```
/* SDO Read-access to remote OD Example */


// SDO read-access to remote OD 0x1017 sub index 0x00 to see if
0x3E8 was written to.
data_16 = 0;
size = sizeof(data_16);
res = COWA_ReadSdo( myHandle, 4, 0x1017, 0x00, (uint8_t
*)&data_16, &size, &err);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_ReadSdo to remote OD 0x1017 subIdx 0x00
failed. res = %d, err = %d\n", res, err);
      return -14;
}
if( data_16 != 0x3E8){
      // handle error.
      printf("\n Value in remote OD 0x1017 subIdx 0x00 is invalid.
value(should be 1000) = %d\n", data_16);
      return -15;
}


Sleep(1000);


// Get the NMT state of the Node A(been monitored)
res = COWA_GetNmtState( myHandle, 4, &state);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_GetNmtState failed. res = %d\n", res);
      return -16;
}
printf("\n COWA_GetNmtState Node A's state = %d\n", state);



// Set the NMT state (operational) to the remote node over CAN
network
res = COWA_SendNmtCommand( myHandle, NodeA_Id,
COWA_NMT_CMD_START);
if( res != COWA_OK){
      // handle error.
      printf("\n COWA_SendNmtCommand to node A failed. res =
%d\n", res);
```

```
	return -17;
}


// Now we set Node A's producer heartbeat time to 0. (disable
heartbeat)
data_16 = 0;
size = 2;
res = COWA_WriteSdo( myHandle, 4, 0x1017, 0x00, (uint8_t
*)&data_16, size, &err);
if( res != COWA_OK){
	// handle error.
	printf("\n COWA_WriteSdo to node A 0x1017, sub index 0x00
failed. res = %d\n", res);
	return -18;
}


/* Transmit EMCY Example */


// You can call this function to send a EMCY message
res = COWA_TransmitEmcy( myHandle, COWA_EMCY_SOFTWARE_ERRORS, 0,
0, 0,0,0,0,0);
if( res != COWA_OK){
	// handle error.
	printf("\n COWA_TransmitEmcy failed. res = %d\n", res);
	return -19;
}



// Now we close the node, the node will go off CAN-bus. And then
close the library, free all the resources
res = COWA_CloseNode( myHandle);
if( res != COWA_OK){
	// handle error.
	printf("\n COWA_CloseNode failed. res = %d\n", res);
	return -20;
}
printf("\n Close local Node success.\n");


COWA_CloseLibrary();
printf("\n Close Library done\n");
return 0;
```

```
}
```

8 References

CiA – 301 CANopen application layer and communication, version 4.2.0
CiA – 302 CANopen additional application layer functions Part 4: Network variables and process image, version 4.1.0
CiA – 314 Accessing CANopen services in devices programmable in IEC 61131-3 languages, version 0.0.3.5