# CANopen Master API for Windows

## Software version 1.2

### Revision history

| Version | Revised Date | Author | Description |
|---|---|---|---|
| 1.2 | 2015/10/13 | Rocky | Added SYNC function content<br>Added TIME function content<br>Added AX_SYNC_Enable function<br>Added AX_SYNC_Disable function<br>Added AX_SYNC_SetConfig function<br>Added AX_SYNC_GetConfig function<br>Added AX_TIME_Enable function<br>Added AX_TIME_Disable function<br>Added AX_TIME_SetConfig function<br>Added AX_TIME_GetConfig function<br>Added AX_PDO_GetUpdObj function<br>Added AX_EVT_ERROR to event list<br>Added AX_ERR_THREADFAIL to error list<br>Added AX_ERR_EVENT_SYNC to error list<br>Added AX_ERR_EVENT_PDO to error list<br>Added AX_ERR_EVENT_TIME to error list<br>Modified the parameter of AX_PDO_AddObj function<br>Modified the parameter of AX_PDO_DelObj function<br>Modified the parameter of AX_PDO_GetObj function<br>Modified the parameter of AX_PDO_SetObj function<br>Modified the maximum value of PDO No to 16.<br>Modified the parameter of RPDO event |
| 1.1 | 2015/09/16 | Rocky | Added AX_ERR_PDO_CANIDEXIST in error list.<br>Added AX_ERR_SDO_ABORT in error list.<br>Added AX_ERR_SDO_FORMAT in error list<br>Modified the maximum value of PDO number to 512.<br>Modified the parameter of all PDO function.<br>Modified the parameter of all SDO function.<br>Modified the parameter of event. |
| 1.0 | 2015/08/03 | Rocky | - 1st release |
| | | | |

# Table of contents

# 12 Appendix

# 1. Introduction

The CANopen Master Application Programming Interface (API) is a programming library for connecting to CANopen network in from of a single monolithic file named CANopenMasterAPI.dll.

## 1.1. API supported

Windows 8 32-bit/64-bit
Windows 7 32-bit/64-bit
Windows Vista 32-bit/64-bit
Windows XP 32-bit/64-bit

## 1.2. Basic specifications

CiA301 CANopen application layer and communication profile

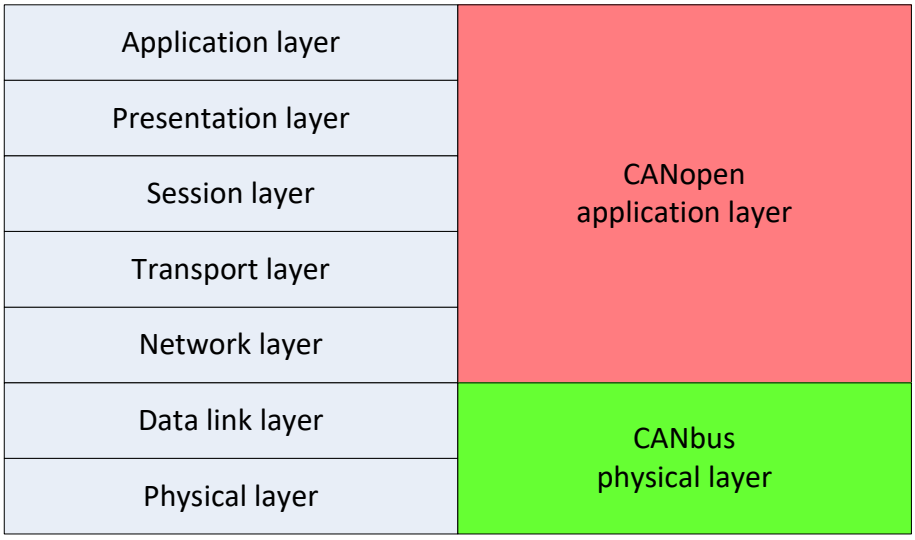
## 1.3. Support CANbus board

The CANopen Master API primarily works together with so-called active CANbus board. There are supported CANopen Master API such as the following list of the CANbus board.

| **Board** | **Interface** | **Note** |
|---|---|---|
| AX93700-CS | COM to CAN | Module |
| AX93700-CU | USB to CAN | Module |
| iCON101-CS | COM to CAN | Device |
| iCON101-CU | USB to CAN | Device |
| AX92903 | USB to CAN | Mini-card |
| | | |

## 1.4. Communication reference model

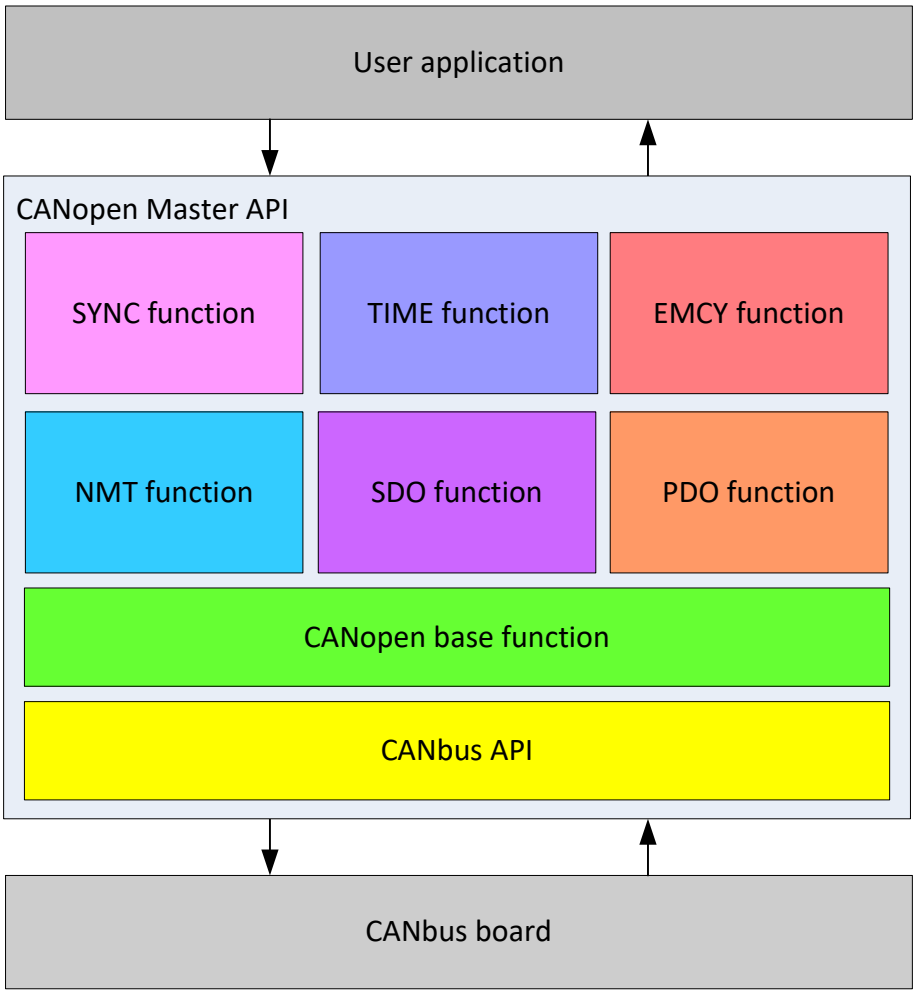
The communication concept conforms to the ISO-OSI reference model.



## 1.5. Architecture

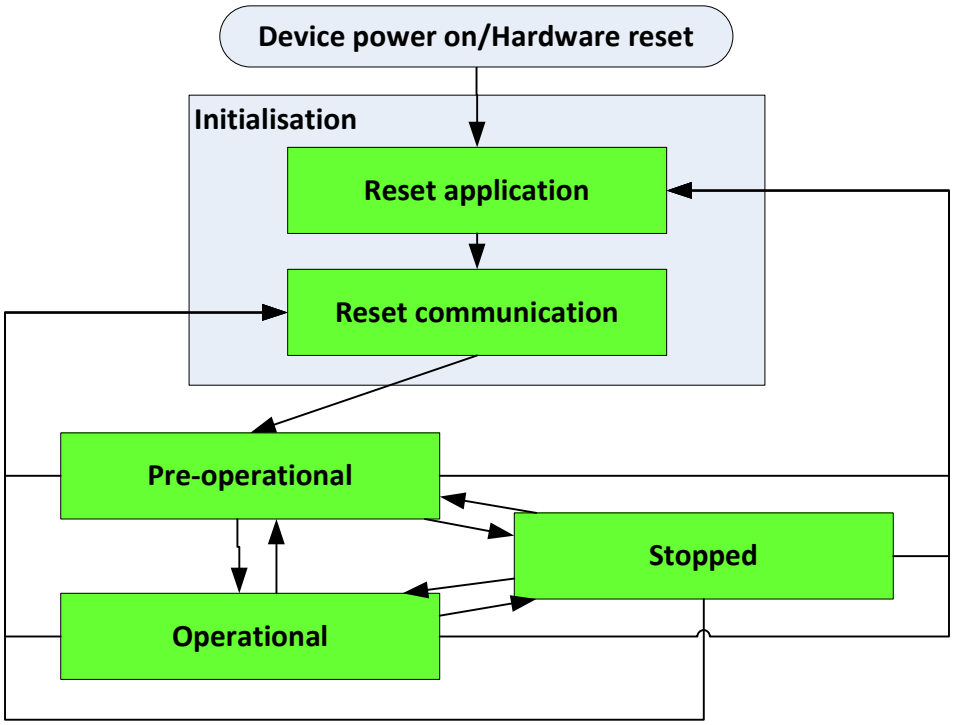
CANopen Master overview.

The below figure shows an overview of the CANopen Master functionality. The functionality of the different CANopen services can be configured individually in order to achieve an optimal performance for different platforms and applications.



## 1.6. Device states flow

The state diagram of a CANopen device is specified. CANopen devices enter the state Pre-operational directly after finishing the CANopen devices initialization. During this state CANopen device parameterization and CAN-ID-allocation via SDO is possible. Then the CANopen devices may be switched directly into the state Operational. The state determines the behavior of the communication function unit.




**Reset application:**

In this sub-state the parameters of the anufacturer-specific profile area and of the standardized device profile area are set to their power-on values. After setting of the power-on values the sub-state reset communication is entered autonomously.

**Reset communication:**

In this sub-state the parameters of the communication profile area are set to their power-on values. After this the state Initialisation is finished and the CANopen device executes the service boot-up write and enters the state Pre-operational.

**Pre-Operational**

In the state Pre-operational, communication via SDOs is possible. PDOs do not exist, so PDO communication is not allowed. Configuration of PDOs, parameters and also the allocation of application objects (PDO mapping) may be performed by a configuration

application. The CANopen device may be switched into the state Operational directly by sending the NMT service start remote node or by means of local control.

**Operational:**

In the state Operational all communication objects are active. Transitioning to the state Operational creates all PDOs; the constructor uses the parameters as described in the object dictionary. Object dictionary access via SDO is possible. Implementation aspects or the application state machine however may require to limit the access to certain objects whilst being in the state Operational, e.g. an object may contain the application program which cannot be changed during execution.

**Stopped:**

By switching a CANopen device into the state Stopped it is forced to stop the communication altogether (except node guarding and heartbeat, if active). Furthermore, this state may be used to achieve certain application behavior. The definition of this behavior falls into the scope of device profiles and application profiles.
If there are EMCY messages triggered in this state they are pending. The most recent active EMCY reason may be transmitted after the CANopen device transits into another state.

## 1.7. Device states and communication objects

The below table specifies the relation between states and communication objects. Services on the listed communication objects may only be executed if the CANopen devices involved in the communication are in the appropriate states.

| | Pre-operational | Operational | Stopped |
|---|---|---|---|
| PDO function | | **X** | |
| SDO function | **X** | **X** | |
| SYNC function | **X** | **X** | |
| TIME function | **X** | **X** | |
| EMCY function | **X** | **X** | |
| NMT control<br>NMT error control | **X** | **X** | **X** |

**(X:Support)**

# 2. CANBus control function

## 2.1. AX_GetLastError

**Summary**

Get last error of CANopen API.

**Definition**

int AX_GetLastError(void)

**Parameters**

None

**Return value**

Error code. **(see 10.1)**

**Example**

```
BOOL bRst;
int iErr;
iErr = AX_GetLastError();
if (iErr != AX_ERR_NONE)
{
	printf(“Error code = %d\r\n”, iErr);
}
```

## 2.2. AX_SetFunTimeout

**Summary**

Set timeout of function of API.

**Definition**

int AX_SetFunTimeout(DWORD iTimeout)

**Parameters**

iTimeout in timeout of function (ms)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_SetFunTimeout(100);    // 100ms
if (bRst)
{     // success
      printf("set timeout scuuess\r\n",);
}
else
{     // failure
      printf("set timeout failure (0x%X)\r\n", AX_GetLastError());
}
```

## 2.3. AX_Open

**Summary**

Open serial port of CAN-board by auto-search.

**Definition**

BOOL AX_Open(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_Open();
if (bRst)
{     // success
      printf("open success\r\n");
}
else
{     // failure
      printf("open failure (0x%X)\r\n", AX_GetLastError());
}
```

## 2.4. AX_OpenEx

**Summary**

Open serial port of CAN-board.

**Definition**

BOOL AX_OpenEx(char* pSerial, int iBaudrate)

**Parameters**

| | | |
|---|---|---|
| pSerial | in | Name of serial port |
| iBaudrate | in | Baudrate of serial port |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_OpenEx(“COM3”, 115200);
if (bRst)
{     //success
      printf(“open success\r\n”);
}
else
{     //failure
      printf(“open failure (0x%X)\r\n”, AX_GetLastError());
}
```

## 2.5. AX_IsOpen

**Summary**

Gets a value indicating the open or closed status of the serial port.

**Definition**

BOOL AX_IsOpen(void)

**Parameters**

None

**Return value**

TRUE means the device is opened, FALSE means the device is closed.

**Example**

```
BOOL bRst;
bRst = AX_IsOpen();
if (bRst)
{     //opened
      printf(“device is opened\r\n”);
}
else
{     //closed
      printf(“device is closed (0x%X)\r\n”, AX_GetLastError());
}
```

## 2.6. AX_Close

**Summary**

Close serial port of CAN-board.

**Definition**

BOOL AX_Close (void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_Close();
if (bRst)
{     //success
      printf("close success\r\n");
}
else
{     //failure
      printf("close failure (0x%X)\r\n", AX_GetLastError());
}
```

# 3. CANBus information function

## 3.1. AX_GetModel

**Summary**

Get model name of CAN-board.

**Definition**

BOOL AX_GetModel(char* pModel)

**Parameters**

pModel out Model name of CAN-board

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cModel[512];
memset(cModel, 0x0, sizeof(cModel));
bRst = AX_GetModel(cModel);
if (bRst)
{   //success
    printf("Model=%s\r\n", cModel);
}
else
{   //failure
    printf("get model name failure (0x%X)\r\n", AX_GetLastError());
}
```

## 3.2. AX_GetUUID

**Summary**

Get UUID of CAN-board.

**Definition**

BOOL AX_GetUUID(char* pUUID)

**Parameters**

pUUID out UUID of CAN-board

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cUUID[512];
memset(cUUID, 0x0, sizeof(cUUID));
bRst = AX_GetUUID(cUUID);
if (bRst)
{    //success
     printf("UUID=%s\r\n", cUUID);
}
else
{    //failure
     printf("get UUID failure (0x%X)\r\n", AX_GetLastError());
}
```

## 3.3. AX_GetSN

**Summary**

Get serial number of CAN-board.

**Definition**

BOOL AX_GetSN(char* pSN)

**Parameters**

| | | |
|---|---|---|
| pSN | out | Serial number of CAN-board |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cSN[512];
memset(cSN, 0x0, sizeof(cSN));
bRst = AX_GetSN(cSN);
if (bRst)
{     //success
      printf("SN=%s\r\n", cSN);
}
else
{     //failure
      printf("get SN failure (0x%X)\r\n", AX_GetLastError());
}
```

## 3.4. AX_GetManufacturer

**Summary**

Get manufacturer of CAN-board.

**Definition**

BOOL AX_GetManufacturer(char* pManufacturer)

**Parameters**

pManufacturer out Manufacturer of CAN-board

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cManufacturer[512];
memset(cManufacturer, 0x0, sizeof(cManufacturer));
bRst = AX_GetManufacturer(cManufacturer);
if (bRst)
{    //success
     printf("Manufacturer=%s\r\n", cManufacturer);
}
else
{    //failure
     printf("get manifacturer failure (0x%X)\r\n", AX_GetLastError());
}
```

## 3.5. AX_GetProductionDate

**Summary**

Get production date of CAN-board.

**Definition**

BOOL AX_GetProductionDate(char* pDate)

**Parameters**

pDate out Production date of CAN-board

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cDate[512];
memset(cDate, 0x0, sizeof(cDate));
bRst = AX_GetProductionDate(cDate);
if (bRst)
{   //success
    printf("Production date=%s\r\n", cDate);
}
else
{   //failure
    printf("get production date failure (0x%X)\r\n", AX_GetLastError());
}
```

## 3.6. AX_GetVersion

**Summary**

Get firmware version of CAN-board.

**Definition**

BOOL AX_GetVersion(char* pVersion)

**Parameters**

pVersion out Firmware version of CAN-board

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cVersion[512];
memset(cVersion, 0x0, sizeof(cVersion));
bRst = AX_GetVersion(cVersion);
if (bRst)
{   //success
    printf("fw ver=%s\r\n", cVersion);
}
else
{   //failure
    printf("get fw version failure (0x%X)\r\n", AX_GetLastError());
}
```

# 4. CANopen basic function

## 4.1. AX_SetCommTimeout

**Summary**

Set timeout of communication.

**Definition**

int AX_SetCommTimeout(DWORD iTimeout)

**Parameters**

iTimeout in timeout of communication (ms)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_SetCommTimeout(500);       // 500ms
if (bRst)
{     // success
      printf("set timeout scuuess\r\n",);
}
else
{     // failure
      printf("set timeout failure (0x%X)\r\n", AX_GetLastError());
}
```

## 4.2. AX_InitCANopen

**Summary**

Initializes the CANopen Master API.

**Definition**

BOOL AX_InitCANopen(int iDevNum, int iBitrate, int iNodeID, BOOL bTermination)

**Parameters**

| | | |
|---|---|---|
| iDevNum | in | Selection of the CAN line to be use (0) |
| iBitrate | in | Number of the bitrate table of CAN to be use<br>The following values are permitted: **(see 10.2)**<br>**AX_CAN_BITRATE_5K**<br>**AX_CAN_BITRATE_10K**<br>**AX_CAN_BITRATE_20K**<br>**AX_CAN_BITRATE_40K**<br>**AX_CAN_BITRATE_50K**<br>**AX_CAN_BITRATE_80K**<br>**AX_CAN_BITRATE_100K**<br>**AX_CAN_BITRATE_125K**<br>**AX_CAN_BITRATE_200K**<br>**AX_CAN_BITRATE_250K**<br>**AX_CAN_BITRATE_400K**<br>**AX_CAN_BITRATE_500K**<br>**AX_CAN_BITRATE_640K**<br>**AX_CAN_BITRATE_800K**<br>**AX_CAN_BITRATE_1000K** |
| iNodeID | in | Node ID of the CANopen Master API between 1 to 127 |
| bTermination | in | Enable/Disable termination state |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iDevNum=0;                          // CANbus number
int iBitrate=AX_CAN_BITRATE_1000K;      // 1M bitrate
int iNodeID=1;                          // Node ID=1
BOOL bTermination=TRUE;                 // enable termination state
bRst = AX_InitCANopen(iDevNum, iBitrate, iNodeID, bTermination);
if (bRst)
{     //success
      printf("Initial CANopen success\r\n");
```

```
}
else
{     //failure
      printf("Initial CANopen failure (0x%X)\r\n", AX_GetLastError());
}
```

## 4.3. AX_ReleaseCANopen

**Summary**

Releases CANopen Master API.

**Definition**

BOOL AX_ReleaseCANopen(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_ReleaseCANopen();
if (bRst)
{     //success
      printf(“release CANopen success\r\n”);
}
else
{     //failure
      printf(“release CANopen failure (0x%X)\r\n”, AX_GetLastError());
}
```

## 4.4. AX_IsInitCANopen

**Summary**

Gets a value indicating the initialization or release status of the CANopen.

**Definition**

BOOL AX_IsInitCANopen(void)

**Parameters**

None

**Return value**

TRUE means the CANopen is initialization, FALSE means the CANopen is release.

**Example**

```
BOOL bRst;
bRst = AX_IsInitCANopen();
if (bRst)
{     //initialization
      printf("CANopen is initialization\r\n");
}
else
{     //release
      printf("CANopen is release\r\n");
}
```

## 4.5. AX_SetEventCallback

**Summary**

Register callback function after trigger event.

**Definition**

BOOL AX_SetEventCallback(int iIndex, PCANEVENT pFunCallback)

**Parameters**

| | | |
|---|---|---|
| iIndex | in | Index of the event table<br>The following values are permitted: **(see 10.3)**<br>**AX_EVT_NMT_NODE_GAURDING**<br>**AX_EVT_NMT_HEARTBEAT**<br>**AX_EVT_SYNC_WRITE**<br>**AX_EVT_TIME_WRITE**<br>**AX_EVT_EMCY_WRITE**<br>**AX_EVT_PDO_RX** |
| pFunCallback | in | Callback function<br>**typedef int (*PCANEVENT) (void *pParam)** |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
int EventHeartbeat(void *pParam);
BOOL bRst;
bRst = AX_SetEventCallback(AX_EVT_NMT_HEARTBEAT, EventHeartbeat);
if (bRst)
{     //success
      printf(“added new node success\r\n”);
}
else
{     //failure
      printf(“added new node failure (0x%X)\r\n”, AX_GetLastError());
}

// heartbeat error
int EventHeartbeat(long lParam)
{
      printf("Heartbeat error, Node ID=0x%x\r\n", (int)lParam);
}
```

# 5. CANopen NMT function

## 5.1. AX_NMT_SearchNode

**Summary**

Checks CANopen-device is available in the network with the specified node ID.

**Definition**

BOOL AX_NMT_SearchNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device to be searched for (1-127)

**Return value**

TRUE means the remote node is existed, FALSE means the remote node isn't existed.

**Example**

```
BOOL bRst;
int iNodeID=2;
bRst = AX_NMT_SearchNode(iNodeID);
if (bRst)
{     //existed
      printf("remote node is existed\r\n");
}
else
{     //not existed
      printf("remote node is not existed\r\n",);
}
```

## 5.2. AX_NMT_AddNode

**Summary**

Added and registered a new node into node table.

**Definition**

BOOL AX_NMT_AddNode(int iNodeID, int iType, int iTime, int iLifeTime)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iType | in | Definition of the monitoring mechanism applicable<br>The following values are permitted: **(see 10.4)**<br>**AX_RMN_TYPE_GUARDING**<br>**AX_RMN_TYPE_HEARTBEAT** |
| iTime | in | Guardtime in milliseconds<br>0: Disable guarded |
| iLifeTime | in | Defines the number of unsuccessful attempts<br>at a Guard-request by the Master to be permitted (1-255) |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;                          // Node ID
int iNodeType=AX_RMN_TYPE_GUARDING;  // Guarding
int iNodeTime=500;                      // 500ms
int iLifeTimeFactor=1;                  // 1 times
bRst = AX_NMT_AddNode(iNodeID, iNodeType, iNodeTime, iLifeTimeFactor);
if (bRst)
{     //success
      printf("added new node success\r\n");
}
else
{     //failure
      printf("added new node failure (0x%X)\r\n", AX_GetLastError());
}
```

## 5.3. AX_NMT_DelNode

**Summary**

Remove a node from node table.

**Definition**

BOOL AX_NMT_DelNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_DelNode(iNodeID);
if (bRst)
{     //success
      printf("deleted remote node success\r\n");
}else
{     //failure
      printf("deleted remote node failure (0x%X)\r\n", AX_GetLastError());
}
```

## 5.4. AX_NMT_GetNodeInfo

**Summary**

Get node information from node table.

**Definition**

BOOL AX_NMT_GetNodeInfo(int iNodeID, int* pType, int* pTime, int* pLifeTime)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| pType | out | Definition of the monitoring mechanism applicable<br>The following values are possible: **(see 10.4)**<br>**AX_RMN_TYPE_GUARDING**<br>**AX_RMN_TYPE_HEARTBEAT** |
| pTime | out | Guardtime in milliseconds<br>0: Disable guarded |
| pLifeTime | out | Defines the number of unsuccessful attempts<br>at a Guard-request by the Master to be permitted (1-255) |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
int iType=0;
int iTime=0;
int iLifeTimeFactor=0;
bRst = AX_NMT_GetNodeInfo(iNodeID, &iType, &iTime, &iLifeTimeFactor);
if (bRst)
{     //success
      printf("NodeID=%d, Type=%d, GuardTime=%d, LifeTime=%d\r\n",
            iNodeID, iType, iTime, iLifeTimeFactor);
}
else
{     //failure
      printf("Failure to get information from remote node (0x%X)\r\n",
            AX_GetLastError());
}
```

## 5.5. AX_NMT_GetNodeState

**Summary**

Get node state from remote node.

**Definition**

BOOL AX_NMT_GetNodeState(int iNodeID, int* pState)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| pState | out | Current state of the remote node |
| | | The following values are possible: **(see 10.6)** |
| | | **AX_DEV_STATE_NONE** |
| | | **AX_DEV_STATE_BOOTUP** |
| | | **AX_DEV_STATE_STOPED** |
| | | **AX_DEV_STATE_OPERATIONAL** |
| | | **AX_DEV_STATE_PREOPERATIONAL** |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
int iState=AX_DEV_STATE_NONE;
bRst = AX_NMT_GetNodeState(iNodeID, &iState);
if (bRst)
{     //success
      if (iState == AX_DEV_STATE_BOOTUP)
            printf("device state : boot-up\r\n");
      else if (iState == AX_DEV_STATE_STOPED)
            printf("device state : stop\r\n");
      else if (iState == AX_DEV_STATE_OPERATIONAL)
            printf("device state : operational\r\n");
      else if (iState == AX_DEV_STATE_PREOPERATIONAL)
            printf("device state : pre-operational\r\n");
      else
            printf("device state : unknow\r\n");
}
else
{     //failure
      printf("Failure to get status from remote node (0x%X)\r\n", AX_GetLastError());
}
```

## 5.6. AX_NMT_StartNode

**Summary**

Transfer a remote node into OPERATIONAL state.

**Definition**

BOOL AX_NMT_StartNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_StartNode(iNodeID);
if (bRst)
{     //success
      printf("start remote node success\r\n");
}
else
{     //failure
      printf("start remote node failure (0x%X)\r\n", AX_GetLastError());
}
```

## 5.7. AX_NMT_StopNode

**Summary**

Transfer a remote node into STOPPED state.

**Definition**

BOOL AX_NMT_StopNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_StopNode(iNodeID);
if (bRst)
{     //success
      printf("stop remote node success\r\n");
}
else
{     //failure
      printf("stop remote node failure (0x%X)\r\n", AX_GetLastError());
}
```

## 5.8. AX_NMT_EnterPreOper

**Summary**

Transfer a remote node into PRE-OPERATIONAL state.

**Definition**

BOOL AX_NMT_EnterPreNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_EnterPreNode(iNodeID);
if (bRst)
{     //success
      printf("Success to enter pre-operational of remote node\r\n");
}
else
{     //failure
      printf("Failure to enter pre-operational of remote node (0x%X)\r\n",
            AX_GetLastError());

}
```

## 5.9. AX_NMT_ResetComm

**Summary**

Reset the communication profile of a remote node.

**Definition**

BOOL AX_NMT_ResetComm(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_ResetComm(iNodeID);
if (bRst)
{     //success
      printf("Success to reset communication of remote node\r\n");
}
else
{     //failure
      printf("Failure to reset communication of remote node (0x%X)\r\n",
            AX_GetLastError());
}
```

## 5.10. AX_NMT_ResetNode

**Summary**

Reset the application of a remote node.

**Definition**

BOOL AX_NMT_ResetNode(int iNodeID)

**Parameters**

iNodeID in Node ID of the CANopen device (1-127)

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;        // Node ID
bRst = AX_NMT_ResetNode(iNodeID);
if (bRst)
{     //success
      printf("reset remote node success\r\n");
}
else
{     //failure
      printf("reset remote node failure (0x%X)\r\n" , AX_GetLastError());
}
```

# 6. CANopen SDO function

## 6.1. AX_SDO_GetObj

**Summary**

Read the Object Dictionary entry from a remote node.

**Definition**

BOOL AX_SDO_GetObj(int iNodeID, int iMode, int iIdx, int iSubIdx, int* pLen,
char* pData, UINT *pAbortCode)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iMode | in | Definition of the SDO-transmission protocol<br>The following values are permitted: **(see 10.9)**<br>**AX_SDO_MODE_NORMAL**<br>**AX_SDO_MODE_BLOCK** |
| iIdx | in | Index of the object dictionary entry |
| iSubIdx | in | Sub-index of the object dictionary entry |
| pLen | in/out | (in) pData buffer size<br>(out) Read size |
| pData | out | Data of the object dictionary entry |
| pAbortCode | out | Possible Abort-Code of the SDO-Transfer In case of an abort, AX_ERR_SDO_ABORT is returned as return value. |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cData[512];                        // buffer
int iNodeID=2;                          // Node ID
int iMode= AX_SDO_MODE_NORMAL;          // SDO mode
int iIdx=0x1000;                        // Index of OD
int iSubIdx=0x00;                       // Sub-index of OD
int iLen=sizeof(cData);                 // length
UINT iAbort=0;                          // abort code
memset(cData, 0x0, sizeof(cData));
bRst = AX_SDO_GetObj(iNodeID, iMode, iIdx, iSubIdx, &iLen, cData, &iAbort);
if (bRst)
{     //success
      printf("Success to get SDO from remote node\r\n");
}
else
```

```
{       //failure
        printf("Failure to get SDO from remote node (0x%X)\r\n", AX_GetLastError());
}
```

## 6.2. AX_SDO_SetObj

**Summary**

Write data into an object dictionary entry of a remote node.

**Definition**

BOOL AX_SDO_GetObj(int iNodeID, int iMode, int iIdx, int iSubIdx, int iLen,
char* pData, UINT *pAbortCode)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iMode | in | Definition of the SDO-transmission protocol<br>The following values are permitted: **(see 10.9)**<br>**AX_SDO_MODE_NORMAL**<br>**AX_SDO_MODE_BLOCK** |
| iIdx | in | Index of the object dictionary entry |
| iSubIdx | in | Sub-index of the object dictionary entry |
| iLen | in | pData buffer size |
| pData | in | Data of the object dictionary entry |
| pAbortCode | out | Possible Abort-Code of the SDO-Transfer In case of an abort, AX_ERR_SDO_ABORT is returned as return value. |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cData[512];                        // buffer
int iNodeID=2;                          // Node ID
int iMode= AX_SDO_MODE_NORMAL;          // SDO mode
int iIdx=0x1017;                        // Index of OD
int iSubIdx=0x00;                       // Sub-index of OD
UINT iAbort=0;                          // abort code
memset(cData, 0x0, sizeof(cData));
cData[0] = (500&0xFF00)>>8;
cData[1] = 500&0xFF;
bRst = AX_SDO_SetObj(iNodeID, iMode, iIdx, iSubIdx, 2, cData, &iAbort);
if (bRst)
{     //success
      printf("Success to set SDO into remote node\r\n");
}
else
{     //failure
```

```
        printf("Failure to set SDO into remote node (0x%X)\r\n", AX_GetLastError());
}
```

# 7. CANopen PDO function

## 7.1. AX_PDO_AddObj

**Summary**

Add a new PDO into PDO table.

**Definition**

BOOL AX_PDO_AddObj(int iNodeID, int iPDONo, int iPDOType, int iPDOMode,
int iSyncCount, int iLen, int iCANID)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iPDONo | in | Number of PDO (1-16) |
| iPDOType | in | Transmission direction of the PDO from point of view<br>The following values are permitted: **(see 10.7)**<br>**AX_PDO_TYPE_TX**<br>**AX_PDO_TYPE_RX** |
| iPDOMode | in | Mode of PDO<br>The following values are permitted: **(see 10.8)**<br>**AX_PDO_MODE_ASYNC**<br>**AX_PDO_MODE_SYNC** |
| iSyncCount | in | Reserved |
| iLen | in | Data length of PDO |
| iCANID | in | Identifier of the CAN-object used by the PDO<br>The following values are permitted: **(see 10.5)**<br>**AX_CANID_MASTER_TPDO1**<br>**AX_CANID_MASTER_RPDO1**<br>**AX_CANID_MASTER_TPDO2**<br>**AX_CANID_MASTER_RPDO2**<br>**AX_CANID_MASTER_TPDO3**<br>**AX_CANID_MASTER_RPDO3**<br>**AX_CANID_MASTER_TPDO4**<br>**AX_CANID_MASTER_RPDO4** |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=2;                          // Node ID
int iPDONo=1;                           // PDO no
int iPDOType=AX_PDO_TYPE_RX;            // rx type
int iPDOMode=AX_PDO_MODE_ASYNC;         // async mode
```

```
int iLen=1;                                    // length
int iCANID=AX_CANID_MASTER_RPDO1;      // CANID
// added RPDO1 object
bRst = AX_PDO_AddObj(iNodeID, iPDONo, iPDOType, iPDOMode, 0, iLen, iCANID);
if (bRst)
{      //success
       printf("Success to add PDO information into PDO table\r\n");
}
else
{      //failure
       printf("Failure to add PDO information into PDO table (0x%X)\r\n",
              AX_GetLastError());
}
```

## 7.2. AX_PDO_DelObj

**Summary**

Delete PDO from PDO table.

**Definition**

BOOL AX_PDO_DelObj(int iNodeID, int iPDONo, int iPDOType)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iPDONo | in | Number of PDO (1-16) |
| iPDOType | in | Transmission direction of the PDO from point of view<br>The following values are permitted: **(see 10.7)**<br>**AX_PDO_TYPE_TX**<br>**AX_PDO_TYPE_RX** |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
Int iNodeID=2;                          // node id
int iPDONo=1;                           // PDO no
int iPDOType=AX_PDO_TYPE_RX;            // rx type
// delete RPDO1 object
bRst = AX_PDO_DelObj(iNodeID, iPDONo, iPDOType);
if (bRst)
{     //success
      printf("Success to delete PDO information from PDO table\r\n");
}
else
{     //failure
      printf("Failure to delete PDO information from PDO table (0x%X)\r\n",
            AX_GetLastError());
}
```

## 7.3. AX_PDO_GetObjInfo

**Summary**

Get PDO information from PDO table.

**Definition**

BOOL AX_PDO_GetObjInfo(int iNodeID, int iPDONo, int iPDOType,
int* pPDOMode, int* pSyncCount, int* pLen, int* pCANID)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iPDONo | in | Number of PDO (1-16) |
| iPDOType | in | Transmission direction of the PDO from point of view<br>The following values are permitted: **(see 10.7)**<br>**AX_PDO_TYPE_TX**<br>**AX_PDO_TYPE_RX** |
| pPDOMode | out | Mode of PDO<br>The following values are possible: **(see 10.8)**<br>**AX_PDO_MODE_ASYNC**<br>**AX_PDO_MODE_SYNC** |
| pSyncCount | out | Reserved |
| pLen | out | Data length of PDO |
| pCANID | out | Identifier of the CAN-object used by the PDO<br>The following values are possible: **(see 10.5)**<br>**AX_CANID_MASTER_TPDO1**<br>**AX_CANID_MASTER_RPDO1**<br>**AX_CANID_MASTER_TPDO2**<br>**AX_CANID_MASTER_RPDO2**<br>**AX_CANID_MASTER_TPDO3**<br>**AX_CANID_MASTER_RPDO3**<br>**AX_CANID_MASTER_TPDO4**<br>**AX_CANID_MASTER_RPDO4** |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=0;                      // Node ID
int iPDONo=1;                       // PDO number
int iPDOType= AX_PDO_TYPE_RX;       // rx type
int iPDOMode=0;
int iSyncCount=0;
```

```
int iLen=0;
int iCANID=0;                          // CANID
// Get RPDO1 object info
bRst = AX_PDO_GetObjInfo(iNodeID, iPDONo, iPDOType,
                                &iPDOMode, &iSyncCount, &iLen, &iCANID);

if (bRst)
{     //success
      printf("Success to get PDO information from PDO table\r\n");
}
else
{     //failure
      printf("Failure to get PDO information from PDO table (0x%X)\r\n",
            AX_GetLastError());
}
```

### 7.4. AX_PDO_GetUpdObj

**Summary**

Read the updated data of a PDO received from the RPDO-queue.

**Definition**

```
BOOL AX_PDO_GetUpdObj(int* pNodeID, int* pPDONo,
                      int* pLen, char* pData, int *pSyncCount)
```

**Parameters**

| | | |
|---|---|---|
| iNodeID | out | Node ID of the CANopen device (1-127) |
| iPDONo | out | Number of PDO (1-16) |
| pLen | in/out | (in) Data length of pData (1-8)<br>(out) Read size |
| pData | out | PDO data |
| pSyncCount | out | Reserved |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cData[1];
int iNodeID=0;
int iPDONo=0;
int iSyncCount=0;
int iLen=sizeof(cData);    // length
memset(cData, 0x0, sizeof(cData));
// get RPDO data
bRst = AX_PDO_GetObj(&iNodeID, &iPDONo, &iLen, cData, &iSyncCount);
if (bRst)
{     //success
      printf("Success to get RPDO\r\n");
}
else
{     //failure
      printf("No RPDO update\r\n");
}
```

## 7.5. AX_PDO_GetObj

**Summary**

Read the data of a PDO received from the RPDO-queue.

**Definition**

```
BOOL AX_PDO_GetObj(int iNodeID, int iPDONo,
                   int* pLen, char* pData, int *pSyncCount)
```

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iPDONo | in | Number of PDO (1-16) |
| pLen | in/out | (in) Data length of pData (1-8)<br>(out) Read size |
| pData | out | PDO data |
| pSyncCount | out | Reserved |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cData[1];
int iNodeID=2;        // Node id
int iPDONo=1;         // PDO number
int iSyncCount=0;
int iLen=sizeof(cData);    // length
memset(cData, 0x0, sizeof(cData));
// get RPDO1 data
bRst = AX_PDO_GetObj(iNodeID, iPDONo, &iLen, cData, &iSyncCount);
if (bRst)
{     //success
      printf("Success to get PDO from remote node\r\n");
}
else
{     //failure
      printf("Failure to get PDO from remote node (0x%X)\r\n", AX_GetLastError());
}
```

## 7.6. AX_PDO_SetObj

**Summary**

Write the data of a PDO transmitted to the TPDO-queue.

**Definition**

BOOL AX_PDO_SetObj(int iNodeID, int iPDONo, int iLen, char* pData)

**Parameters**

| | | |
|---|---|---|
| iNodeID | in | Node ID of the CANopen device (1-127) |
| iPDONo | in | Number of PDO (1-16) |
| iLen | in | Data length of pData (1-8) |
| pData | in | PDO data |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
char cData[1];
int iNodeID=2;              // node id
int iPDONo=1;               // PDO number
int iLen=sizeof(cData);     // length
cData[0]=0xFF;
bRst = AX_PDO_SetObj(iNodeID, iPDONo, iLen, cData);
if (bRst)
{     //success
      printf("Success to set PDO into remote node\r\n");
}
else
{     //failure
      printf("Failure to set PDO into remote node (0x%X)\r\n", AX_GetLastError());
}
```

# 8. CANopen SYNC function

## 8.1. AX_SYNC_Enable

**Summary**

Enable synchronization.

**Definition**

BOOL AX_SYNC_Enable(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_SYNC_Enable();
if (bRst)
{     //success
      printf(“Success to enable sync object\r\n”);
}
else
{     //failure
      printf(“Failure to enable sync object(0x%X)\r\n”, AX_GetLastError());
}
```

## 8.2. AX_SYNC_Disable

**Summary**

Disable synchronization.

**Definition**

BOOL AX_SYNC_Disable(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_SYNC_Disable();
if (bRst)
{     //success
      printf("Success to disable sync object\r\n");
}
else
{     //failure
      printf("Failure to disable sync object(0x%X)\r\n", AX_GetLastError());
}
```

## 8.3. AX_SYNC_GetConfig

**Summary**

Get configuration of synchronization.

**Definition**

BOOL AX_SYNC_GetConfig(PUINT32 pPeriod, PUINT32 pWindows, int* pCount)

**Parameters**

| | | |
|---|---|---|
| pPeriod | out | The cycle time between sync object (milliseconds) |
| pWindows | out | Reserved |
| pCount | out | Maximum value of the sync counter |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
UINT32 iPeriod=0;
UINT32 iWindows=0;
int iCount=0;
bRst = AX_SYNC_GetConfig(&iPeriod, &iWindows, &iCount);
if (bRst)
{     //success
      printf("Success to get configuration of synchronization\r\n");
}
else
{     //failure
      printf("Failure to get configuration of synchronization (0x%X)\r\n",
            AX_GetLastError());
}
```

## 8.4. AX_SYNC_SetConfig

**Summary**

Set configuration of synchronization.

**Definition**

BOOL AX_SYNC_SetConfig(UINT32 iPeriod, UINT32 iWindows, int iCount)

**Parameters**

| | | |
|---|---|---|
| iPeriod | in | The cycle time between sync object (milliseconds) |
| iWindows | in | Reserved |
| iCount | in | Maximum value of the sync counter (2-240) |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
UINT32 iPeriod=3000;    // 3 sec
UINT32 iWindows=0;
int iCount=2;           // max count
bRst = AX_SYNC_SetConfig(iPeriod, iWindows, iCount);
if (bRst)
{    //success
     printf("Success to set configuration of synchronization\r\n");
}
else
{    //failure
     printf("Failure to set configuration of synchronization (0x%X)\r\n",
          AX_GetLastError());
}
```

# 9. CANopen TIME function

## 9.1. AX_TIME_Enable

**Summary**

Enable timestamp.

**Definition**

BOOL AX_TIME_Enable(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_TIME_Enable();
if (bRst)
{     //success
      printf("Success to enable timestamp object\r\n");
}
else
{     //failure
      printf("Failure to enable timestamp object(0x%X)\r\n", AX_GetLastError());
}
```

## 9.2. AX_SYNC_Disable

**Summary**

Disable timestamp.

**Definition**

BOOL AX_TIME_Disable(void)

**Parameters**

None

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
bRst = AX_TIME_Disable();
if (bRst)
{     //success
      printf("Success to disable timestamp object\r\n");
}
else
{     //failure
      printf("Failure to disable timestamp object(0x%X)\r\n", AX_GetLastError());
}
```

## 9.3. AX_TIME_GetConfig

**Summary**

Get configuration of timestamp.

**Definition**

BOOL AX_TIME_GetConfig(PUINT16 pDays, PUINT32 pMSec, PUINT32 pPeriod)

**Parameters**

| | | |
|---|---|---|
| pDays | out | Days since 1984/1/1 |
| pMSec | out | Milliseconds after midnight |
| pPeriod | out | The cycle time between timestamp object (milliseconds) |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
UINT16 iDays=0;
UINT32 iMSec=0;
UINT32 iPeriod=0;
int iCount=0;
bRst = AX_TIME_GetConfig(&iDays, &iMSec, &iPeriod);
if (bRst)
{     //success
      printf("Success to get configuration of timestamp\r\n");
}
else
{     //failure
      printf("Failure to get configuration of timestamp (0x%X)\r\n",
            AX_GetLastError());
}
```

## 9.4. AX_TIME_SetConfig

**Summary**

Set configuration of timestamp.

**Definition**

BOOL AX_TIME_SetConfig(UINT16 iDays, UINT32 iMSec, UINT32 iPeriod)

**Parameters**

| | | |
|---|---|---|
| iDays | in | Days since 1984/1/1 |
| iMSec | in | Milliseconds after midnight |
| iPeriod | in | The cycle time between timestamp object (milliseconds) |

**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
UINT16 iDays=11315;          // 2015/1/1
UINT32 iMSec=36000000;       // 10:00
UINT32 iPeriod=1000;         // 1000ms
bRst = AX_TIME_SetConfig(iDays, iMSec, iPeriod);
if (bRst)
{     //success
      printf("Success to set configuration of timestamp \r\n");
}
else
{     //failure
      printf("Failure to set configuration of timestamp (0x%X)\r\n",
            AX_GetLastError());
}
```

# 10. CANopen EMCY function

## 10.1. AX_EMCY_GetObj

**Summary**

Read the data of a EMCY object received from the EMCY-queue.

**Definition**

BOOL AX_EMCY_GetObj(int* pNodeID, int* pErrCode, int* pErrReg, char* pErrData)

**Parameters**

| | | |
|---|---|---|
| pNodeID | out | Number of the node that has issued the error message |
| pErrCode | out | Error code of the error message |
| pErrReg | out | Error register of the error message |
| pErrData | out | Error data of the error message |

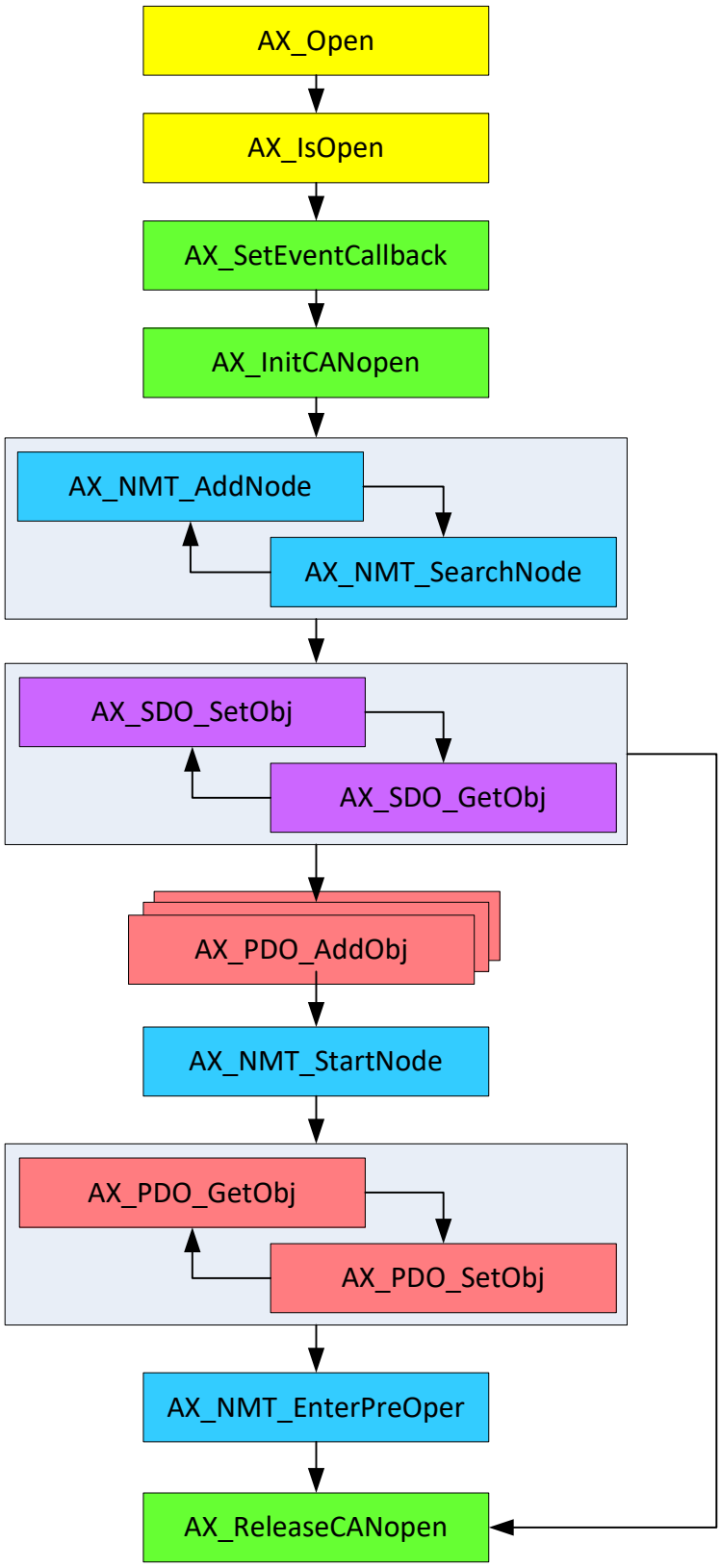
**Return value**

TRUE means success and FALSE means failure.

**Example**

```
BOOL bRst;
int iNodeID=0;              // Node ID
int iErrCode=0;             // Error code
int iErrReg=0;              // Error register
char cErrData[8];           // Error data
memset(cErrData, 0x0, sizeof(cErrData));
bRst = AX_EMCY_GetObj(&iNodeID, &iErrCode, &iErrReg, cErrData);
if (bRst)
{     //success
      printf("Success to get EMCY object from remote node\r\n");
}
else
{     //failure
      printf("Failure to get EMCY object from remote node (0x%X)\r\n",
            AX_GetLastError());
}
```

## 11. Typical sequence of function calls



# 12. Appendix

## 12.1. Appendix - Error code list

| Name | Value | Description |
|---|---|---|
| AX_ERR_NONE | 0x0000 | No error |
| AX_ERR_BUSY | 0x0001 | Function busy |
| AX_ERR_PARA | 0x0002 | Parameters error |
| AX_ERR_CONNECTED | 0x0010 | Device has been connected |
| AX_ERR_DISCONNECTED | 0x0011 | Device has been disconnected |
| AX_ERR_QUEUEEMPTY | 0x0012 | Queue empty |
| AX_ERR_UNINITIALIZED | 0x0013 | CANopen has not been initalized |
| AX_ERR_THREADFAIL | 0x0014 | Resume thread error |
| AX_ERR_CAN_OPENED | 0x0020 | CAN-port has been opened |
| AX_ERR_CAN_CLOSED | 0x0021 | CAN-port has been closed |
| AX_ERR_CAN_BITRATE | 0x0022 | Set bitrate error of CAN |
| AX_ERR_CAN_CONFIG | 0x0023 | Set configuration error of CAN |
| AX_ERR_CAN_TERMIN | 0x0024 | Set termination error of CAN |
| AX_ERR_CAN_ID | 0x0025 | Set CAN-ID error of CAN |
| AX_ERR_CAN_MASK | 0x0026 | Set CAN-mask error of CAN |
| AX_ERR_CAN_TX | 0x0027 | Send message error of CAN |
| AX_ERR_CAN_RX | 0x0028 | Receive message error of CAN |
| AX_ERR_CAN_TIMEOUT | 0x0029 | Response timeout of CAN |
| AX_ERR_NMT_NODEEXIST | 0x0030 | NMT-node is exist |
| AX_ERR_NMT_NODENOTEXIST | 0x0031 | NMT-node is not exist |
| AX_ERR_PDO_TIMEOUT | 0x0040 | PDO timeout |
| AX_ERR_PDO_OBJEXIST | 0x0041 | Object is exist in PDO list |
| AX_ERR_PDO_OBJNOTEXIST | 0x0042 | Object is not exist in PDO list |
| AX_ERR_PDO_CANIDEXIST | 0x0043 | CANID is exist of object in PDO list |
| AX_ERR_SDO_TIMEOUT | 0x0050 | SDO timeout |
| AX_ERR_SDO_ABORT | 0x0051 | SDO abort |
| AX_ERR_SDO_FORMAT | 0x0052 | SDO format error |
| AX_ERR_EVENT_SYNC | 0x1000 | Sync object error |
| AX_ERR_EVENT_PDO | 0x1001 | Sync PDO error |
| AX_ERR_EVENT_TIME | 0x1002 | Time object error |
| | | |

## 12.2. Appendix - CANbus bitrate list

| Name | Value | Description |
|---|---|---|
| AX_CAN_BITRATE_5K | 0x0 | 5k |
| AX_CAN_BITRATE_10K | 0x1 | 10k |
| AX_CAN_BITRATE_20K | 0x2 | 20k |
| AX_CAN_BITRATE_40K | 0x3 | 40k |
| AX_CAN_BITRATE_50K | 0x4 | 50k |
| AX_CAN_BITRATE_80K | 0x5 | 80k |
| AX_CAN_BITRATE_100K | 0x6 | 100k |
| AX_CAN_BITRATE_125K | 0x7 | 125k |
| AX_CAN_BITRATE_200K | 0x8 | 200k |
| AX_CAN_BITRATE_250K | 0x9 | 250k |
| AX_CAN_BITRATE_400K | 0xA | 400k |
| AX_CAN_BITRATE_500K | 0xB | 500k |
| AX_CAN_BITRATE_640K | 0xC | 640k |
| AX_CAN_BITRATE_800K | 0xD | 800k |
| AX_CAN_BITRATE_1000K | 0xE | 1000k |

## 12.3. Appendix - Event list

| Name | Value | Description | Return value |
|---|---|---|---|
| AX_EVT_NMT_NODE_GAURDING | 0x0101 | Guarding error | INT(Node ID) |
| AX_EVT_NMT_HEARTBEAT | 0x0102 | Heartbeat error | INT(Node ID) |
| AX_EVT_EMCY | 0x0401 | EMCY event | NULL |
| AX_EVT_PDO_RX | 0x0501 | RPDO event | NULL |
| AX_EVT_ERROR | 0xFF01 | Thread error event | INT(Error code) |
| | | | |

## 12.4. Appendix - Monitoring type list of NMT

| Name | Value | Description |
|---|---|---|
| AX_RMN_TYPE_GUARDING | 0x0001 | Guarding |
| AX_RMN_TYPE_HEARTBEAT | 0x0002 | Heartbeat |

## 12.5. Appendix - CANID list for Master of NMT

| Name | Value | Description |
|---|---|---|
| AX_CANID_MASTER_RPDO1 | 0x180 | RPDO1 of CANID |
| AX_CANID_MASTER_TPDO1 | 0x200 | TPDO1 of CANID |
| AX_CANID_MASTER_RPDO2 | 0x280 | RPDO2 of CANID |
| AX_CANID_MASTER_TPDO2 | 0x300 | TPDO2 of CANID |
| AX_CANID_MASTER_RPDO3 | 0x380 | RPDO3 of CANID |
| AX_CANID_MASTER_TPDO3 | 0x400 | TPDO3 of CANID |
| AX_CANID_MASTER_RPDO4 | 0x480 | RPDO4 of CANID |
| AX_CANID_MASTER_TPDO4 | 0x500 | TPDO4 of CANID |

## 12.6. Appendix - Status list of NMT

| Name | Value | Description |
|---|---|---|
| AX_DEV_STATE_NONE | 0x00 | Un-know |
| AX_DEV_STATE_BOOTUP | 0x01 | Boot-up |
| AX_DEV_STATE_STOPPED | 0x02 | Stopped |
| AX_DEV_STATE_OPERATIONAL | 0x03 | Operational |
| AX_DEV_STATE_PREOPERATIONAL | 0x04 | Pre-operational |

## 12.7. Appendix - Transmission type list of PDO

| Name | Value | Description |
|---|---|---|
| AX_PDO_TYPE_TX | 0x01 | PDO tx |
| AX_PDO_TYPE_RX | 0x02 | PDO rx |

## 12.8. Appendix - Synchronous mode list of PDO

| Name | Value | Description |
|---|---|---|
| AX_PDO_MODE_ASYNC | 0x00 | Synchronous mode |
| AX_PDO_MODE_SYNC | 0x01 | Asynchronous mode |

## 12.9. Appendix - Transmission protocol list of SDO

| Name | Value | Description |
|---|---|---|
| AX_SDO_MODE_NORMAL | 0x00 | Use of the normal transfer protocol |
| AX_SDO_MODE_BLOCK | 0x01 | Use of the block transfer protocol |