CANopen Library User Manual

V1.03

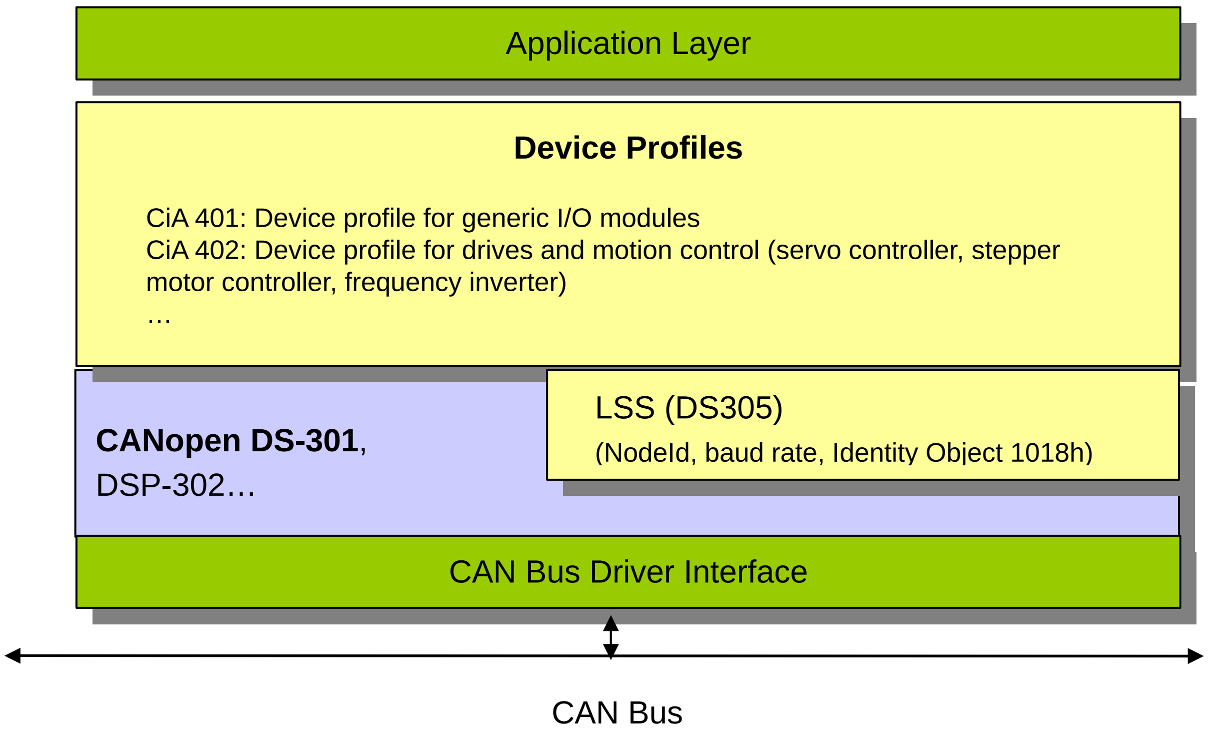
June 2010

Table of Contents

1. Introduction

1.1. CANopen architecture

CANopen, application layer communication protocol based on CAN bus, is widely used in distributed industrial automation system, medical system, maritime system, etc. CANopen is made up of a series of sub-protocol sets which can be divided into two parts. The first part is communication sub-protocol set that defines the basic communication modes and objects of all the devices. This part consists of DS-301, DS-302, DS-305, etc., among which DS-301 mainly describes specifications and definitions of CANopen application layer communication objects and other protocols are supplements to CANopen network on the basis of DS-301. The other part, device sub-protocol set, defines the function and data definition of standard devices which are of different types. These sub-protocols contain DS-401, DS-402, etc. Each device of CANopen describes its property and functions in standard Electronic Data Sheets (EDS) files. CANopen devices must strictly take the definitions in EDS files as their specifications and support perfect device exchange.


Application Layer
Device Profiles
CiA 401: Device profile for generic I/O modules
CiA 402: Device profile for drives and motion control (servo controller, stepper motor controller, frequency inverter)
…
CANopen DS-301, DSP-302…
LSS (DS305)
(NodeId, baud rate, Identity Object 1018h)
CAN Bus Driver Interface
CAN Bus

Basic communication objects of CANopen are:

OD (Object Dictionary), generated from the devices' EDS files, and describes all properties and communication objects of the device. It's the core definition of CANopen devices. Object Dictionary is composed of sequential object lists. Each object applies a 16-bit index as its addressing. Each object can be made up of several elements or a single element uses an 8-bit sub-index as its addressing.

SDO (Service Data Object) can access and configure Object Dictionary in remote nodes via index and sub-index of Object Dictionary. The object that requests to access is regarded as Client, and the object which is requested is Server. The length of request messages and response messages remains 8 bytes, including SDO command (1 byte), index of the object to be accessed (2 bytes), sub-index of the object to be accessed (1 byte) and 4 bytes of data to be transmitted. The SDO protocol can transmit data of any size. If the data is over 4 bytes, the message will be segmented into several parts.

The term SDO Read means SDO upload protocol, the client of a SDO uploads data from the server. SDO Write means SDO download protocol; the client of a SDO downloads data to the server.

PDO (Process Data Object) is used for high speed data exchange. The length of transmitted data is limited to 1-8 bytes. PDO contains many transmitting trigger ways, such as cycle transmitting (synchronized, time driven), triggered by remote frame, triggered by particular events, etc.

NMT (Network Management) belongs to master-slave mode. One NMT master can correspond to several NMT Slaves. NMT master detects status of each node of the network and completes status conversion. It supports the function of monitoring the device's status by heartbeat, Node guarding or Life guarding. The diagram in Figure 1 illustrates the major states a slave node can be in. Starts the CANopen network or power on a node, it goes into NMT state Initialisation. At the end of initialization the slave node tries to transmit its boot-up message to NMT master to report that boot-up has been finished and has entered pre-operation status. An NMT master can switch individual slave node or all nodes back and forth between the three major states: Pre-operational, Operational, and Stopped.

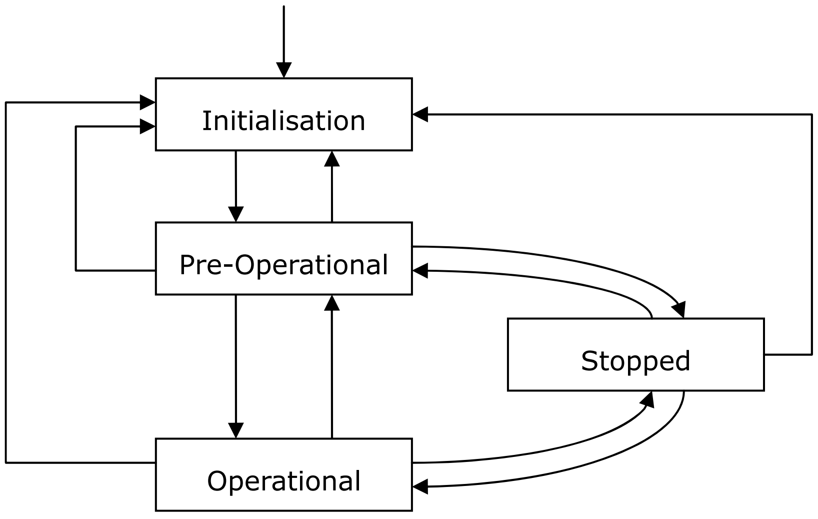


Figure 1: State diagram of a node

SYNC (Synchronization) makes the devices in the network possess synchronous capability. Only one node can produces the SYNC signal in a system as SYNC producer, which can be NMT master or other slave node. It is the time period in microseconds with which the SYNC signal occurs; the node has this value available in the OD entry [1006h, 00h].

Heartbeat or **Node Guarding** is a method to detect the node is live or not, and retrieves the status of a node. Recommend using heartbeat instead of node guarding less bandwidth, is more flexible. The node as heartbeat producer transmits heartbeat message cyclically, consisting of a 1-byte the current NMT state the node is in.

EMCY (Emergency) will be triggered if fatal errors occur inside the device, similar to error interrupt mode.

The relation between NMT states and communication objects shows in Table 1: NMT states and communication objects. It defines the communication objects can be serviced in the appropriate NMT state if the CANopen devices support. For example: the PDO is a critical message can be transmitted only in NMT state Operational otherwise can not.

Table 1: NMT states and communication objects

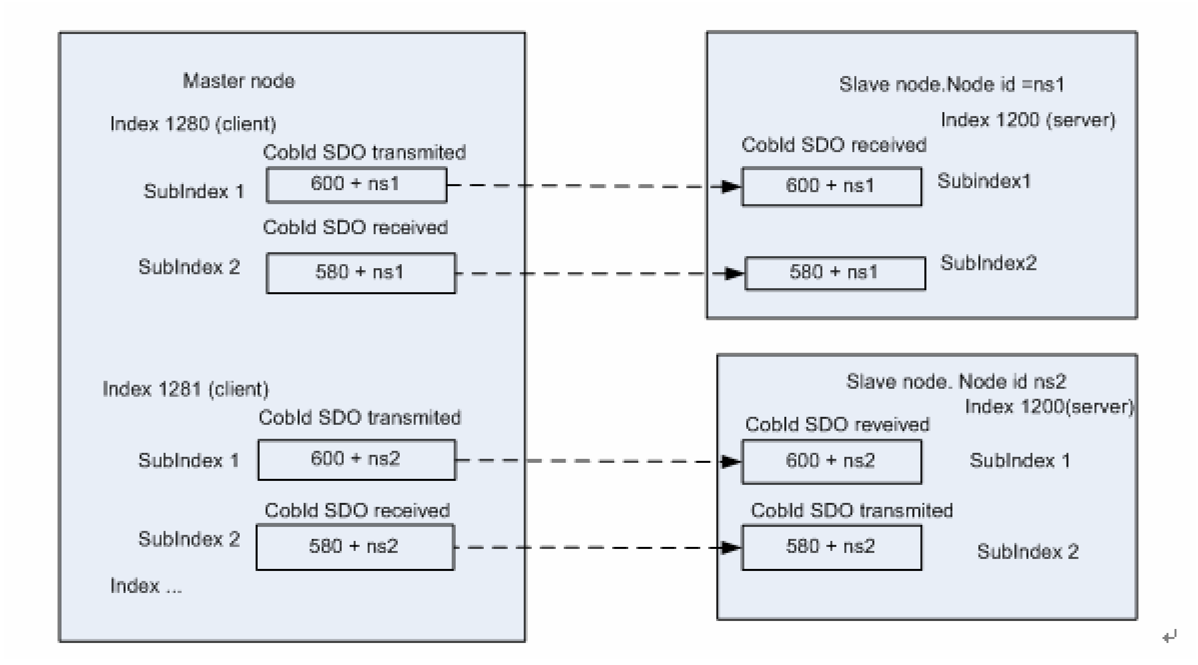
State / Object	Pre-operational	Operational	Stopped
PDO		V	
SDO	V	V	
SYNC	V	V	
TIME	V	V	
EMCY	V	V	
NMT	V	V	V

1.2. Object Dictionary setting

The object dictionary is generated from the EDS file, describes all properties and communication object of the CANopen device. It is the core that the behavior of the CANopen device depends on. Detail information reference CiA DS301 [1] Communication profile. This section describes how to configure the Object Dictionary for a node to be a SDO client or SDO server, can transmit/receive PDO message, and send SYNC message and heartbeat.

1.2.1.Service Data Objects (SDO)

Generally in CANopen network, each node implements only one SDO server (object 1200h) that handles read and write request its object dictionary from other nodes as SDO client. SDO client defined at object 1280h to 12FFh.



1.2.2.Process Data Objects (PDO)

PDO can be distinguishing between Transmit Process Data Object (TPDO) and Receive Process Data Object (RPDO). One node produces a PDO, which is a TPDO for that node. Other nodes receive the PDO, which is an RPDO (consumer).

The communication parameters for a TPDO are at object dictionary of index 1800h to 19FFh (indicating which CAN message is used for the PDO and how is it triggered), but index 1804h – 19FFh default are disabled by CiA DS301 [1]. And the mapping parameters are at index 1A00h to 1BFFh. acoapi library supports above 4 PDOs up to 512 PDOs. The COB-ID of above 4 PDOs can be self-defined and/or using CAN2.0B CAN message format that means the 29-bit of COB-ID could be 1, and lower 7 bit is defined as NodeId.

The transmit trigger method is defined in sub-index 02h transmission type. The value is 0-240 and 252 means that the PDO is transferred synchronously and cyclically. The transmission type 254 means that defined by manufacturer specific (manufacturer specific part of the object dictionary), 255 means defined by device profile. In transmission type 254 and 255 that can use event timer for trigger in an elapsed timer that are in sub-index 05h of the TPDO, and also support Change-Of-State (COS) transmission method simply transmits a TPDO message if the process data is in changes else not transmits even the event timer or inhibited timer is expired .

PDO Parameters	
Sub-index 00h	Number of sub-index
01h	COB-ID, 180h + NODE-ID
02h	Transmission type: 0-255
03h	Inhibit time, multiple of 100 μs
04h	Reserved
05h	Event timer, multiple of 1 ms 0: disable

For example: A node that node identifier is 2h, another node is 5h. Figure 2: PDO Linking illustrates the relation of the PDO linking that Node 2h transmits TPDO_1 to Node 5h, Node 5h receives the PDO as RPDO and update the data to object dictionary of specified index and sub-index.

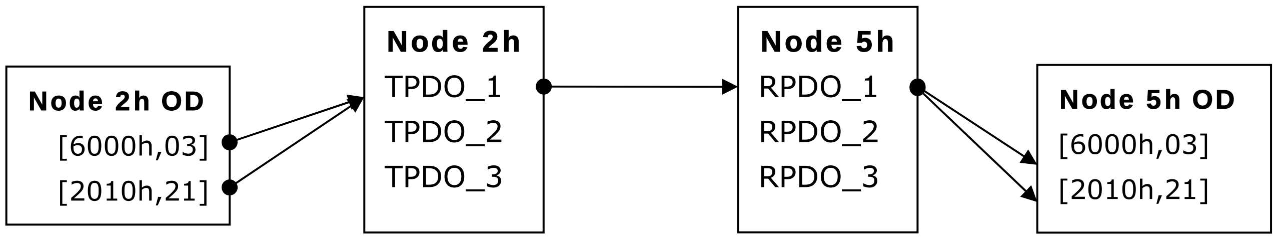


Figure 2: PDO Linking

Following is the example to describe how to set the value to the PDO in object dictionary.

■ TPDO setting example

A node with node identifier 0x02, the PDO is transmitted on synchrony. It contains 6 bytes of data: DataX (2 bytes) and DataY (4 bytes). DataX is defined at index 6000h sub-index 03h. DataY is defined at index 0x2010 sub-index 21h. The result of object dictionary setting about TPDO1 is shown in Table 2: TPDO1 setting example.

Table 2: TPDO1 setting example

Index [1800h]	value	description
Sub-index 00h	2	
01h	0x182	180h + Node-ID 0x2

02h	200	the PDO is transmitted every reception of <200> SYNC
Index [1A00h]	value	description
Sub-index 00h	2	write the number of data embedded in the PDO (1byte)
01h	0x60000308	Where to find of data embedded and the size. (8 bytes) Format: index(2 bytes) – sub-index(1 byte) – size in bits(1 byte) DataX is at object [6000h, 03h] with 8 bits
02h	0x20102120	where to find the second data embedded an the size (8bytes) DataY is at object [2010h, 21h] with 32 bits

- RPDO setting example

Another node 0x05 needs to be configured to directly listen for the TPDO1 transmitted by node 0x02. RPDO1 of node 0x05 should be used to receive TPDO1 of node 0x02.

Table 3: RPDO setting example

Index [1400h]	value	description
Sub-index 00h	2	
01h	0x182	180h + Node-ID 0x2
02h	200	the PDO is transmitted every reception of <200> SYNC
Index [1600h]	value	description
Sub-index 00h	2	write the number of data embedded in the PDO (1byte)
01h	0x60000308	Where to find of data embedded and the size. (8 bytes) Format: index(2 bytes) – sub-index(1 byte) – size in bits(1 byte) DataX is at object [6000h, 03h] with 8 bits
02h	0x20102120	where to find the second data embedded an the size (8bytes) DataY is at object [2010h, 21h] with 32 bits

1.2.3.SYNC setting

A node as SYNC producer broadcasts the synchronization object periodically, which provides the basic network synchronization. The time period in μs between SYNC is defined at index 1006h Communication cycle period of object dictionary. And mandatory if the node generates the SYNC object, the allow bit 30 at object 1005h must to be set.

Table 4: SYNC COB-ID (1005h) setting example

Index [1005h]	value	description
DefaultValue	0x00000080 or 0x40000080	Example: 11-bit SYNC COB-ID is 80h. Bit 30 set to 0 means the node does not generate the SYNC object. Bit 30 set to 1 means the node generates the SYNC object.

Table 5: Communicate Cycle Period (1006h) setting example

Index [1006h]	value	description
DefaultValue	0 or t	The time period is 0 means do not transmit the SYNC object. Other value in μs means the node generate the SYNC object in every <t> μs if allow bit 30 is set at index 1005h. Example: DefaultValue=0x001E8480, transmit every 2 seconds

1.2.4.Heartbeat

According to CiA DS301 [1], a CANopen node must support either heartbeat or node guarding protocol that can be monitoring the node is live or not. The heartbeat protocol is preferred since with the less bandwidth, so that the producer heartbeat time at index 1017h must be implemented.

Table 6: Producer Heartbeat Time (1017h) setting example

Index [1017h]	value	description
DefaultValue	0 or t	The time period is 0 means disable transmission of heartbeat message by the node. Other value specifies in milliseconds the time between transmissions of heartbeat messages. Example: DefaultValue=0x1388, the node transmits the heartbeat message every 5 seconds.

Other nodes (heartbeat consumers) can monitor the node whether the heartbeat is transmitted in specified time as heartbeat consumer time. If the heartbeat is not received within the heartbeat consumer time, a heartbeat error event will be trigger. The index 1016h of object dictionary specifies the maximum time to wait for a heartbeat from a specific node, which maximum 7Fh nodes. The consumer heartbeat time should be greater than the producer heartbeat time.

Table 7: Consumer Heartbeat Time (1016h) setting example

Index [1016h]	value	Description
Sub-index 00h	1	Number of entries
01h	0x005A1122	Consumer heartbeat time The format is: bit 0-15 heartbeat consumer time, bit 16-23 monitored node id, bit 24-31 reserved (set to 0). Example: DefaultValue=0x00051B58, This node must receives the heartbeat message from the node 5h within 7 seconds, else trigger a heartbeat error event for node 5.
02h – 7Fh		

1.3. Reference

1 [CiADS301] CANopen Application Layer and Communication Profile, CiA Draft Standard Proposal 301 Version 4.2, 7. Dec 2007

2 [CiADS203-1] CAN Application Layer for Industrial Application, CiA/DS203-1, Feb 1996

3 [CiADS306] Electronic data sheet specification for CANopen, CiA Draft Standard 306 Version 1.3, 01 Jan 2005

2. Advantech CANopen Protocol Library

2.1. Overview

Advantech CANopen Protocol Library (acoapi) provides a C application programming interface (API) for accessing the CANopen network protocol stack of nodes. It is easy to use, configure, start and monitor the CANopen devices careless CAN bus, developer just focused on CANopen application functionality. The acoapi library architecture is shown in Figure 3, at present, the library practices the specification DSP 301 v4.2 [1] defined by CiA, communication profile.

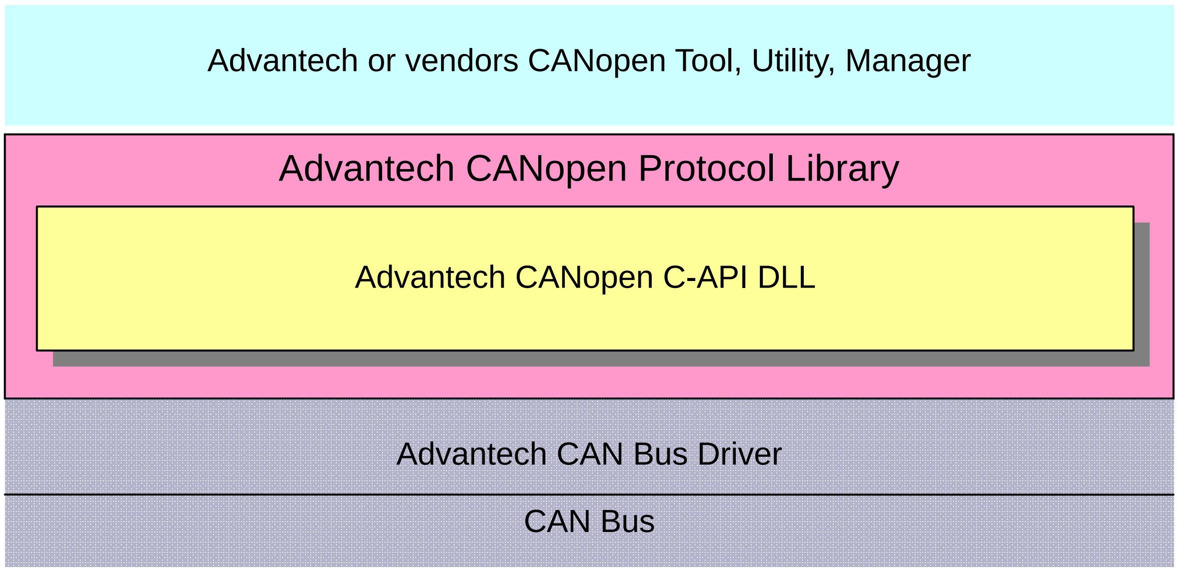

Advantech or vendors CANopen Tool, Utility, Manager
Advantech CANopen Protocol Library
Advantech CANopen C-API DLL
Advantech CAN Bus Driver
CAN Bus

Figure 3: Advantech CANopen Protocol Architecture

Base on the acoapi library to develop a CANopen node as master or slaves, the functionality of slaves and of a master specified by CANopen can configure and manage remote nodes is covered by the library:

- Read and write object dictionary (local or by SDO)
- Control or monitor the node NMT state (NMT master)
- PDO transmission mode: on request, by SYNC, time driven, event driven
- Support 512 TPDOs and 512 RPDOs
- SYNC producer and consumer
- Heartbeat producer and consumer

- Emergency objects

The acoapi library uses event-driven to notify the application to complete the tasks, indicate that an event has occurred. For example: Notify the application when the NMT state of a master/slave node has changed that can decide to do something in state changing, notify SYNC message received or transmitted, or PDO frame transmitted, the CANopen node object dictionary of object index and sub-index data has changed, etc.

2.2. Object Diagram

The acoapi library references CiA DS203-1 NMT Service Specification [2] to deal with the CANopen network aspect. Figure 4: The NMT model illustrates the NMT model of a CANopen network the acoapi library implemented.

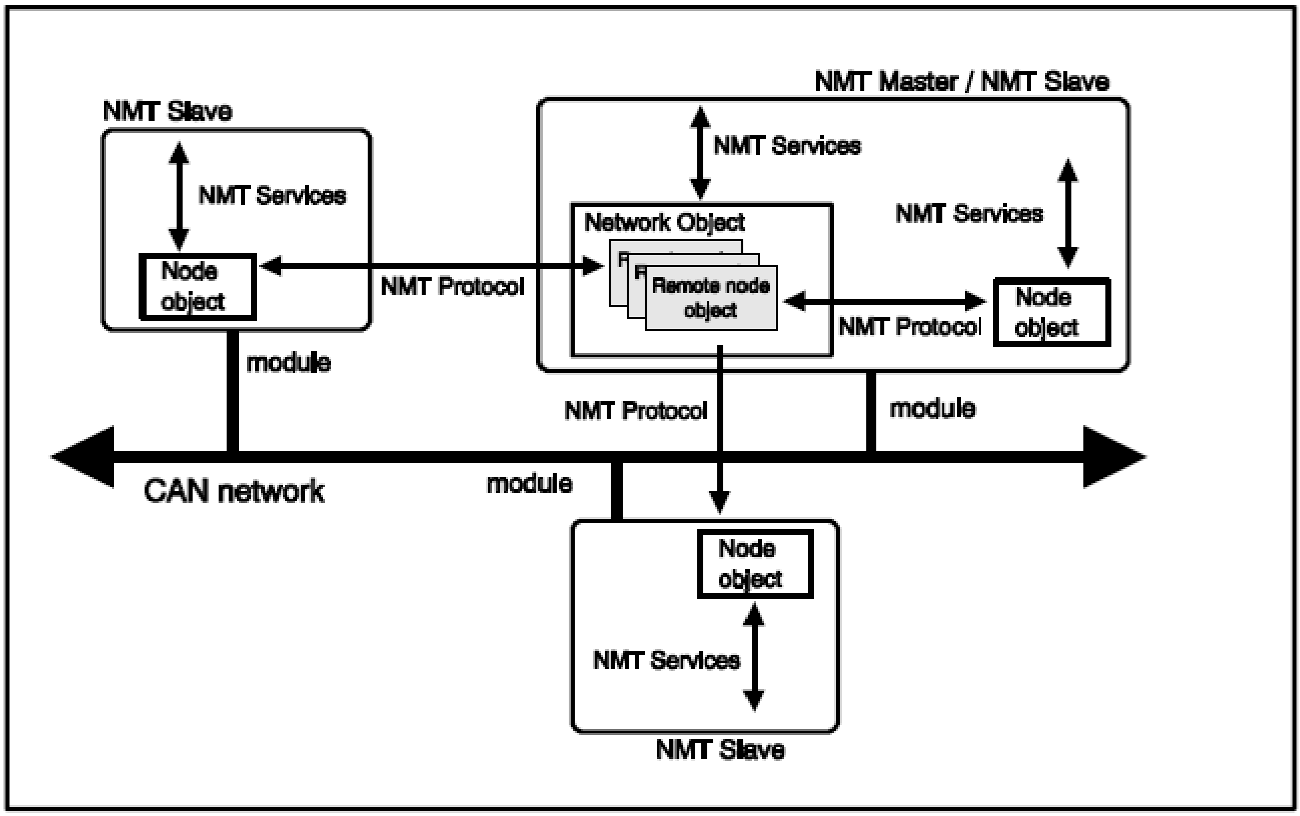


Figure 4: The NMT model

There are three objects to model a CANopen network:

- The network object
  The network object represents the set of all modules in a CANopen network that must include one node, called master node, and at most 126 node objects specified by CiA DS301 [1] (totally 127 nodes in CANopen network). It administers whole the life cycle of a CANopen network through the master. In other word, the network object exists in master side.

- The remote node object
  Each slave node is managed by the NMT master is represented by a remote node object on the network object.

- The node object
  A CANopen device in the network is considered as a node. The role of node can be master or slave depended on its capability. Each node has uniquely node identifier in the network defined in CiA DS301 [1].

For each slave there must exist one remote node object with the same node identifier on the network object that master node in. A slave node object and the remote node object that has the same node identifier are called peers. A unique node identifier is assigned to a slave node object and its peer by the master (network object). Master in the network object communicates with each remote node object via CANopen protocol to its peer slave node.

2.3. Runtime Environment

Windows 2000

Windows xp (32 bit and 64 bit)

Windows 2003 (32 bit and 64 bit)

Windows vista (32 bit and 64 bit)

Window 7 (32 bit and 64 bit)

2.4. API Functions

2.4.1. Overview

The acoapi DLL implements a set of functions which together provide CANopen functionality. Each function of first parameter is an object handles that pointer to master/slave node, remote node or network object. Network object is in master side for management the CANopen network; it must need a master node for creating a CANopen network and at most 126 remote slave nodes included, totally nodes is 127 in CANopen network. Remote slave node can be inserted to network object first before starting network, or inserted later if receiving the slave node boot-up message.

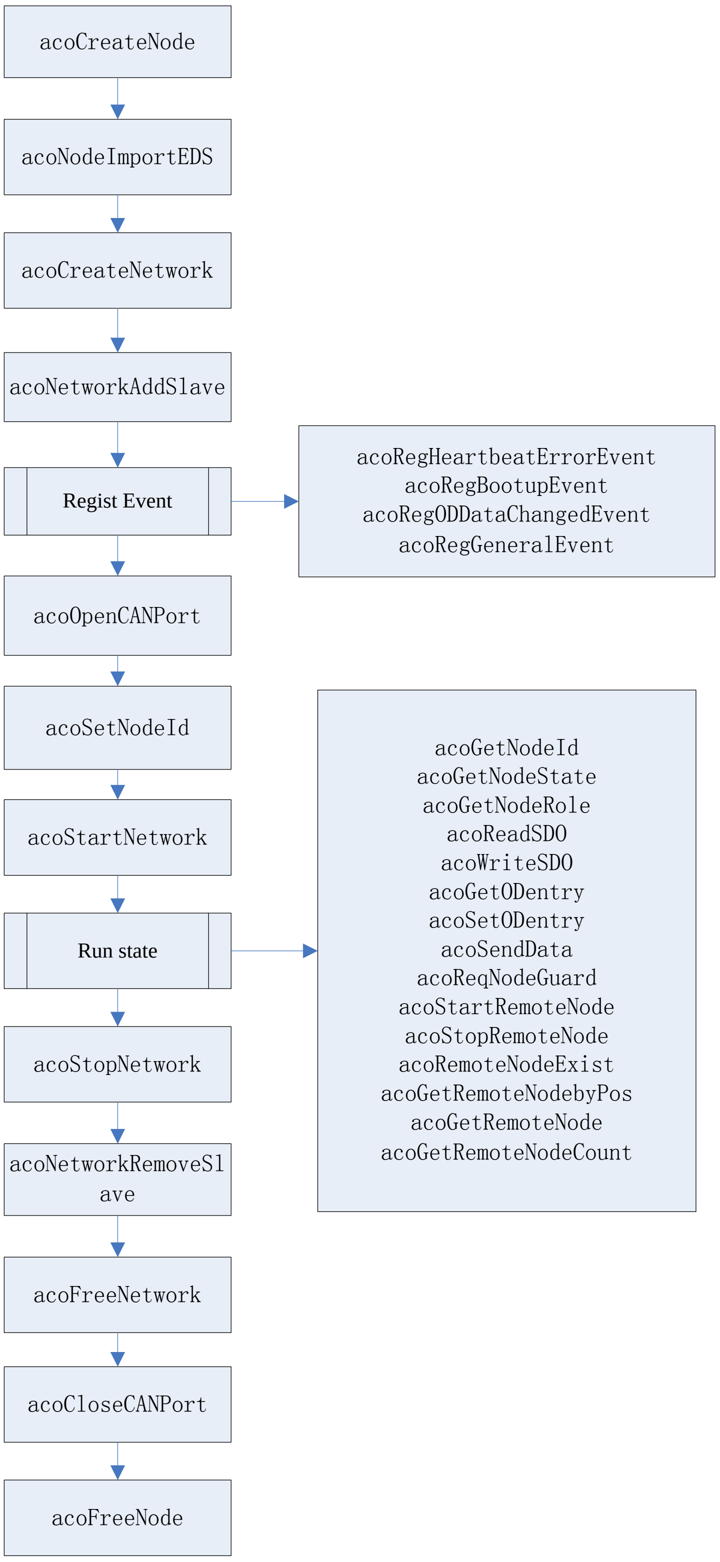
Table 8: acoapi function list

Function	Description
Network object (Master side)	
acoCreateNetwork	Create a CANopen network topology, must input a Master Node
acoFreeNetwork	Free network object instance and remote nodes
acoStartNetwork	Start network, all nodes will be into Operational state
acoStopNetwork	Stop network, all nodes will be into Stop state
acoStartRemoteNode	Start a remote node by NodeId
acoStopRemoteNode	Stop a remote node by NodeId
acoGetRemoteNodeId	Get node id of remote object
acoGetRemoteNodeState	Get the node state of remote object
acoGetRemoteNodeCount	Get remote nodes count
acoGetRemoteNode	Get remote node handle by NodeId
acoGetRemoteNodebyPos	Get Remote node handle by position
acoRemoteNodeExist	Check the Remote node is already exist in network object
acoNetworkAddSlave	Add a Remote node into network object
acoNetworkRemoveSlave	Remove a Remote node from network object
acoNetworkSetState	Send a NMT message to a Slave to change its state
acoReqNodeGuard	Request a Slave node the Node Guard message
acoRegBootupEvent	Register an event while receiving slave boot-up frame
acoNetworkReadSDO	Send SDO frame in async mode to read the object dictionary of remote slave node, it will call specified callback function after receiving the remote slave's response
acoNetworkWriteSDO	Send SDO frame in async mode to write the data to the object dictionary of remote slave node, it will call specified callback function after completing

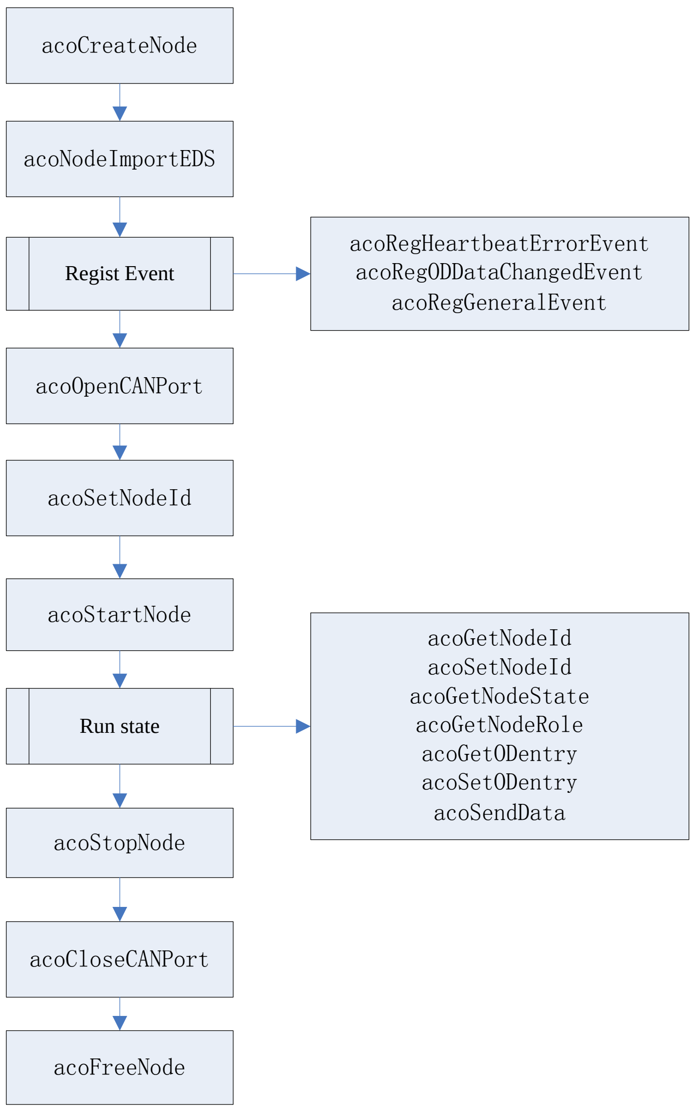
	the process
Node	
acoCreateNode	Create a node (Master/Slave) according
acoFreeNode	Free a node and close the CAN port
acoOpenCANPort	open the CAN port
acoCloseCANPort	Close the CAN port
acoSetBaudrate	Change the CAN port by specified baud rate.
acoNodeImportEDS	Import the device profile and assign the node id
acoGetNodeId	Get the node Id of the node object
acoSetNodeId	Set the node id of the node object
acoGetNodeState	Get the state of the node
acoStartNode	Start a slave node into Pre-Operational state that waiting NMT message
acoStopNode	Stop a slave node into Stop state
acoGetNodeRole	Get node role (Master or Slave) by node handle
acoGetODentry	Read local object dictionary info (SDO upload) or read remote slave node of OD by SDO in sync method
acoSetODentry	Write data to local object dictionary (SDO download) or write remote slave node of OD by SDO in sync method
acoReadSDO	A slave node reads other slave node of OD by sending SDO frame in async mode
acoWriteSDO	A slave node writes the data to other slave node of OD by sending SDO frame in async mode
acoRegODDataChangedEvent	Register an event for the entry [index, subindex] of Object Dictionary changed
acoUnRegODDataChangedEvent	Un-register an event for the entry changed
acoRegGeneralEvent	Register a call back function that for a general event
acoRegEmcyEvent	Register an event for slave received EMCY message
acoSendEmcy	Slave sends emergency message actively
acoSendData	Transmit a specified data to other nodes
acoRegHeartbeatEvent	Registers a callback function that will be called while receiving a heartbeat message
acoRegHeartbeatErrorEvent	Registers a callback function that will be called while detecting a heartbeat error or node guard error occurs
acoRegRecvPDOEvent	Registers an event is for receiving a PDO message.
acoSendPDOwithCOS	Send TPDOs if any data has changed (Change-Of-State), only transmission type 255 (0xFF) is supported yet.

2.4.2.Flow chart

■ Master side



- Slave side


acoCreateNode
acoNodeImportEDS
Regist Event
acoRegHeartbeatErrorEvent
acoRegODDataChangedEvent
acoRegGeneralEvent
acoOpenCANPort
acoSetNodeId
acoStartNode
Run state
acoGetNodeId
acoSetNodeId
acoGetNodeState
acoGetNodeRole
acoGetODentry
acoSetODentry
acoSendData
acoStopNode
acoCloseCANPort
acoFreeNode

2.4.3.Definition and Structures

Definition	Description
UNS8	A 8-bit unsigned char
UNS16	A 16-bit unsigned short integer
UNS32	A 32-bit unsigned long integer
NODEID_t	Type definition of node identifier
HACONODE	Pointer to slave node object
HACOMASTER	Pointer to master node object
HACOREMOTE	Pointer to remote node object
HACONODEOBJ	Pointer to a node object, may be slave node object, master node object or remote node object
HACONETWORK	Pointer to network object
CBROADCAST_NODEID	Node identifier is 0 for broadcast CANopen message
CINVALID_NODEID	-1 indicates invalid node identifier
MAX_PORTNAME	Maximum length of CAN port name
Object Dictionary Data Types	
ACODT_BOOLEAN	0x01, Boolean
ACODT_INT8	0x02, a 8-bit integer
ACODT_INT16	0x03, a 16-bit integer
ACODT_INT32	0x04, a 32-bit integer
ACODT_UINT8	0x05, a 8-bit unsigned integer
ACODT_UINT16	0x06, a 16-bit unsigned integer
ACODT_UINT32	0x07, a 32-bit unsigned integer
ACODT_REAL32	0x08, a 32-bit real
ACODT_VISIBLE_STRING	0x09, visible string
ACODT_OCTET_STRING	0x0A, octet string
ACODT_UNICODE_STRING	0x0B, Unicode string
ACODT_TIME_OF_DAY	0x0C, TIME_OF_DAY structure
ACODT_TIME_DIFFERENCE	0x0D, TIME_DIFFERENCE structure
ACODT_DOMAIN	0x0F, domain, an arbitrary large block of data
ACODT_INT24	0x10, a 24-bit integer
ACODT_REAL64	0x11, a 64-bit real
ACODT_INT40	0x12, a 40-bit integer
ACODT_INT48	0x13, a 48-bit integer

ACODT_INT56	0x14, a 56-bit integer
ACODT_INT64	0x15, a 64-bit integer
ACODT_UINT24	0x16, a 24-bit unsigned integer
ACODT_UINT40	0x18, a 40-bit unsigned integer
ACODT_UINT48	0x19, a 48-bit unsigned integer
ACODT_UINT56	0x1A, a 56-bit unsigned integer
ACODT_UINT64	0x1B, a 64-bit unsigned integer

Enumerate and structures	

enum enum_nodeRole {
Slave = 0x00,
Master = 0x01,
RemoteNode = 0x02,
UnknownRole = 0x0F,
};
typedef enum enum_nodeRole e_nodeRole;
	The role of a node object:
	Slave node object
	Master node object
	Remote node object
	Unknown node object

enum enum_acoNodeState {
ns_Initialisation = 0x00,
ns_Stopped = 0x04,
ns_Operational = 0x05,
ns_Pre_operational = 0x7F,
ns_Unknown_state = 0x0F
};
typedef enum enum_acoNodeState e_acoNodeState;
	The NMT state of a node:
	NMT state Initialisation,
	NMT state Stoppped,
	NMT state Operational,
	NMT state Pre-operational,
	Unknown state

enum enum_GeneralEventType {
 	
et_STATE_INITIALIZE = 0x00,
et_STATE_PREOPERATIONAL = 0x01,
et_STATE_OPERATIONAL = 0X02,
et_STATE_STOPPED = 0x03,
et_POST_SYNC = 0x04,
et_POST_PDO = 0x05,
};
typedef enum enum_GeneralEventType e_GeneralEventType;
	Identity a general event that with the same function definition:
	the node enter NMT state Initialisation,
	the node enter NMT state Pre-operational,
	the node enter NMT state Operational,
	the node enter NMT state Stopped,
	the node transmitted or received a sync frame,
	the node transmitted a PDO frame

Application-defined callback function
Notify application-defined when a special event has occurs typedef void (*TCOnGeneralEvent)(HACONODE hNode , LPVOID pvArg);

Notify if the value with specified object index and sub-index of object dictionary has changed typedef UNS32 (*TCOnODDataChangedEvent) (HACONODE hNode, UNS16 wIndex, UNS8 SubIndex, UNS8 DataType, UNS32 DataLen, void * pData , LPVOID pvArg);
Notify if SDO READ request is responded or timeout occurs. Typedef void (*TCOnReadSDOResult) (HACONODEOBJ hNode, NODEID_t NodeId, UNS16 wIndex, UNS8 SubIndex, UNS8 dataType, UNS32 dataLen, void* pData, UNS32 abortCode , LPVOID pvArg);
Notify if SDO WRITE request is responded or timeout occurs. Typedef void (*TCOnWriteSDOResult) (HACONODEOBJ hNode, NODEID_t NodeId, UNS16 wIndex, UNS8 SubIndex, UNS32 abortCode , LPVOID pvArg);
Notify if the slave node receives the emergency message from other nodes typedef void (*TCOnPostEmcyEvent) (HACONODE hNode, NODEID_t NodeID, UNS16 errCode, UNS8 errReg, UNS8 errManufacField[5], LPVOID pvArg);
Notify while not receive heartbeat message within the consumer heartbeat time. Typedef void (*TCOnHeartbeatErrorEvent)(HACONODE hNode, NODEID_t NodeId , LPVOID pvArg);
Notify while receiving a heartbeat message from heartbeat producer (remote slave nodes.) typedef void (*TCOnHeartbeatEvent) (HACONODE hNode, NODEID_t NodeId, e_acoNodeState currState , LPVOID pvArg);
Notify while master receiving a slave boot up message. Typedef void (*TCOnBootupEvent)(HACONODE hNode, NODEID_t NodeId , LPVOID pvArg);
Notify while receiving a PDO message from another node. typedef void (*TCOnRecvPDOEvent) (HACONODE hNode, int rpdoIndex, UNS32 cobId, UNS8 bRTR, UNS8 size, void *pData, LPVOID pvArg);

2.4.4. Error codes

acoapi error codes

Table 9: acoapi error codes

Definition	Value	Description
ACOERR_SUCCESS	0	Success
ERROR_INVALID_DATA	13	The data or input parameter is invalid or NULL
ERROR_OUTOFMEMORY	14	Not enough memory to allocate

ACOERR_INVALID_EDS_CONTENT	0x25100001	Wrong EDS file content
ACOERR_SDO_READ_FAILED	0x25100002	SDO read operation failed. Ex: no SDO entry in EDS
ACOERR_SDO_WRITE_FAILED	0x25100003	SDO write operation failed
ACOERR_STATE_CHANGED_FAILED	0x25100004	Fail to change node state
ACOERR_INVALID_MASTER	0x25100005	Master node is not exist
ACOERR_INVALID_NODEID	0x25100006	Invalid node id, for example: node id is over NMT_MAX_NODE_ID
ACOERR_INVALID_NODE	0x25100007	Invalid node object handle
ACOERR_REMOTE_NODEID_EXIST	0x25100008	The ID of remote node is exist
ACOERR_REMOTE_STATE_OPER	0x25100009	Remote node is in operational state that can not do this operation
ACOERR_CANPORT_NOT_OPEN	0x2510000A	CAN port can not open
ACOERR_NOT_SUPPORT	0x2510000B	Do not support the function
ACOERR_REMOTE_NODES_OVER	0x2510000C	Remote nodes count is over the limit in list, maximum nodes are 127
ACOERR_NODEID_EXIST	0x2510000D	The node id is exist in list
ACOERR_EMCY_FAILED	0x2510000E	Generate a emergency code failed
ACOERR_LICENSE_INVALID	0x2510000F	License failed. The acoapi library only supports Advantech allowed products
ACOERR_TIMER_FULL	0x25100010	The timer is full that can not set alarm.
ACOERR_PDO_IN_INHIBIT_TIME	0x25100011	The TPDO exist inhibit time that can not transmit PDO immediately.
ACOERR_PDO_COBID_29BIT	0x25100012	The 29-bit COB-ID should be 1.
ACOERR_PDO_TRANS_EVENT_INVALID	0x25100013	The transmission type of PDO is invalid to send. It should be 255 for COS.
ACCERR_EXCEPTION	0x25100014	Exception occurs.
ACCERR_PDO_TRANS_FAILED	0x25100015	Can not transmit PDO, the node could be not in Operational state, PDO offset invalid, or the object dictionary of PDO data invalid.

SDO abort codes

SDO abort codes please see DS 301 v4.02 p.48.

Table 10: SDO abort codes

Definition	Value	Description

ACOOD_SUCCESSFUL	0x00000000	Success
ACOOD_UNSUPPORT_OBJECT	0x06010000	Unsupported access to an object
ACOOD_READ_NOT_ALLOWED	0x06010001	Attempt to read a write-only object
ACOOD_WRITE_NOT_ALLOWED	0x06010002	Attempt to write a read-only object
ACOOD_NO_SUCH_OBJECT	0x06020000	Object does not exist in the Object Dictionary
ACOOD_NOT_MAPPABLE	0x06040041	Object cannot be mapped to the PDO
ACOOD_LENGTH_EXCEED	0x06040042	The number and length of the objects to be mapped would exceed PDO length
ACOOD_PARAM_INCOMPATIBILITY	0x06040043	General parameter incompatibility
ACOOD_INTERNAL_INCOMPATIBILITY	0x06040047	General internal incompatibility in the device
ACOOD_HW_ERROR	0x06060000	Access failed due to hardware error
ACOOD_LENGTH_DATA_INVALID	0x06070010	Data type does not match. Length of service parameter does not match
ACOOD_LENGTH_DATA_TOO_HIGH	0x06070012	Data type does not match. Length of service parameter is too high
ACOOD_LENGTH_DATA_TOO_LOW	0x06070013	Data type does not match. Length of service parameter is too low.
ACOOD_NO_SUCH_SUBINDEX	0x06090011	Subindex does not exist
ACOOD_VALUE_RANGE_EXCEED	0x06090030	Value range of parameter exceeded (write access only)
ACOOD_VALUE_TOO_HIGH	0x06090031	Value of parameter written is too high
ACOOD_VALUE_TOO_LOW	0x06090032	Value of parameter written is too low
ACOOD_INVALID_MAX_VALUE	0x06090036	Maximum value is less than the minimum value.
ACOSDOABT_TOGGLE_NOT_ALTERNED	0x05030000	
ACOSDOABT_TIMED_OUT	0x05040000	
ACOSDOABT_UNKNOW_COMMAND	0x05040001	Client/Server command specifier not valid or unknown
ACOSDOABT_INVALID_BLOCK_SIZE	0x05040002	Invalid block size (block mode)
ACOSDOABT_INVALIC_SEQ_NUM	0x05040003	Invalid sequence number (block mode)
ACOSDOABT_CRC_ERROR	0x05040004	CRC error (block mode)
ACOSDOABT_OUT_OF_MEMORY	0x05040005	Size data exceed SDO_MAX_LENGTH_TRANSFERT
ACOSDOABT_GENERAL_ERROR	0x08000000	Error size of SDO message
ACOSDOABT_DATA_TRANS_STORE_ERROR	0x08000020	Data cannot be transferred or stored to the application.
ACOSDOABT_LOCAL_CTRL_ERROR	0x08000021	Data cannot be transferred or stored to the application because of local control

Emergency error codes

Emergency error codes please see DS 301 v4.02 p.60.

Table 11: Emergency error codes

Definition	Value	Description
ACOEMCYERR_RESEST	0x0000	Error reset or No error
ACOEMCYERR_GENERIC	0x1000	Start code of Generic error
ACOEMCYERR_CURRENT	0x2000	Start code of Current
ACOEMCYERR_CURRENT_DEVICE_INPUT	0x2100	Current, device input side
ACOEMCYERR_CURRENT_DEVICE_INSIDE	0x2200	Current inside the device
ACOEMCYERR_CURRENT_DEVICE_OUTPUT	0x2300	Current, device output side
ACOEMCYERR_VOLTAGE	0x3000	Start code of Voltage
ACOEMCYERR_VOLTAGE_MAINS	0x3100	Main Voltage
ACOEMCYERR_VOLTAGE_INSIDE_DEVICE	0x3200	Voltage inside the device
ACOEMCYERR_VOLTAGE_OUTPUT	0x3300	Output voltage
ACOEMCYERR_TEMPERATURE	0x4000	Temperature
ACOEMCYERR_TEMPERATURE_AMBIENT	0x4100	Ambient temperature
ACOEMCYERR_TEMPERATURE_DEVICE	0x4200	Device temperature
ACOEMCYERR_DEVICE_HW	0x5000	Device hardware
ACOEMCYERR_DEVICE_SW	0x6000	Device software
ACOEMCYERR_DEVICE_SW_INTERNAL	0x6100	Internal software
ACOEMCYERR_DEVICE_SW_USER	0x6200	User software
ACOEMCYERR_DEVICE_SW_DATA	0x6300	Data set
ACOEMCYERR_ADDITIONAL_MODULES	0x7000	Additional modules
ACOEMCYERR_MONITOR	0x8000	Monitoring
ACOEMCYERR_MONITOR_COMMUN	0x8100	Start code of Monitoring Communication
ACOEMCYERR_MONITOR_CAN_OVERRUN	0x8110	CAN Overrun (Objects lost)
ACOEMCYERR_MONITOR_PASSIVE_MODE	0x8120	CAN in Error Passive Mode
ACOEMCYERR_MONITOR_GUARD	0x8130	Life guard error or heartbeat error
ACOEMCYERR_MONITOR_RECOVER_BUSOFF	0x8140	Recovered from bus off
ACOEMCYERR_MONITOR_COBID_COLLISION	0x8150	Transmit COB-ID collision
ACOEMCYERR_PROTOCOL	0x8200	Start code of Protocol error
ACOEMCYERR_PDO_LENGTH_INVALID	0x8210	PDO not processed due to length error
ACOEMCYERR_PDO_LENGTH_EXCEEDED	0x8220	PDO length exceeded
ACOEMCYERR_EXTERNAL	0x9000	External error

ACOEMCYERR_ADDITIONAL_FUNC	0xF000	Additional functions
ACOEMCYERR_DEVICE_SPECIFIC	0xFF00	Device specific

2.4.5.acoapi Functions

acoCreateNode

Create Master or Slave CANopen node.

```
HACONODE acoCreateNode(
   char *pCANDriverName
);
```

Parameters

pCANDriverName

[in] Input the CAN driver dll path and name. (Ignore)

Return Values

If the function succeeds, the return value is the master or slave node object handle. If the function fails, the return value is NULL. The generality of failed reason could be allocating memory failed. Could call GetLastError() to get the error code.

Remarks

Use the acoFreeNode() function to close an node object handle.

Example

```
HACONODE hSlaveNode = NULL;
hSlaveNode = acoCreateNode(NULL);
if (hSlaveNode)
 printf("Create Node SUCCESS.");
else
 printf("Create Node failed: %x", GetLastError());
```

acoFreeNode

Free the handle of a node that created by acoCreateNode(), the function will close the CAN port first if it is opening .

```
void acoFreeNode(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode()

Return Values

None

Example

```
extern HACONODE hSlaveNode;
acoFreeNode(hSlaveNode);
hSlaveNode = NULL;
```

acoOpenCANPort

Open the CAN port by specified port name and then set the baud rate. Should be open the CAN port first before start the CANopen network.

```
UNS32 acoOpenCANPort(
   HACONODE hNode,
   char *pPortName,
   int kbits
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that is the returned by acoCreateNode()

pPortName

[in] Specifies the CAN port name that is null-terminated string to open. For example, specify CAN1 , CAN2 as the CAN port.

kbits

[in] The baud rate (Kbits) of the CAN port. According to CiA Draft Standard 301 [1] recommend bit rates are listed below:

Target value	Setting value
10 Kbit/s	10

20 Kbit/s	20
50 Kbit/s	50
125 Kbit/s	125
250 Kbit/s	250
500 Kbit/s	500
800 Kbit/s	800
1000 Kbit/s	1000

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Use the acoCloseCANPort() function to close the CAN port.

Example

```
UNS32 ret;
HACONODE hSlaveNode;
//Use acoOpenCANPort() to set CAN port name and baud rate.
hSlaveNode = acoCreateNode(NULL);
if (hSlaveNode)
{
  printf("Create Node SUCCESS.");
  if (acoNodeImportEDS(hSlaveNode, “\\slavedict-ok.eds”, 0x02) == ACOERR_SUCCESS)
  {
    printf("Import EDS file ok.");
    // should be ImportEDS first than OpenCANPort
    if ((ret = acoOpenCANPort(hSlaveNode, "CAN1:", 1000)) == ACOERR_SUCCESS)
      printf("Open CAN1 ok.");
    else
      printf("Open CAN1 failed. %x", ret);
  }else
      printf("Import EDS file failed.");
}
```

acoCloseCANPort

Close the CAN port.

```
UNS32 acoCloseCANPort(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that is the returned by acoCreateNode()

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
extern HACONODE hSlaveNode;
if(hSlaveNode)
{
     acoStopNode(hSlaveNode);
      acoCloseCANPort(hSlaveNode);
      acoFreeNode(hSlaveNode);
      hSlaveNode = NULL;
 }
```

acoSetBaudrate

Set the CAN port baud rate.

```
UNS32 acoSetBaudrate(
   HACONODE hNode,
   int kbits
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that is the returned by acoCreateNode()

kbits

[in] The baud rate (Kbits) of the CAN port. According to CiA Draft Standard 301 [1] recommend bit rates are listed below:

Target value	Setting value
10 Kbit/s	10
20 Kbit/s	20
50 Kbit/s	50
125 Kbit/s	125
250 Kbit/s	250
500 Kbit/s	500
800 Kbit/s	800
1000 Kbit/s	1000

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

acoNodeImportEDS

Import the device profile for this node object handle and assign the node id.

```
UNS32 acoNodeImportEDS(
   HACONODE hNode,
   char * pProfile,
   NODEID_t * pNodeId
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that is the returned by acoCreateNode().

pProfile

[in] Device profile path and file name.

pNodeId

[in, out] Specified the node id, or return the Id if included in Profile. Also can be changed by acoSetNodeId(). Maximum the number of node id is 127.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, the returns value is acoapi error codes.

Remarks

EDS describes the behavior of CANopen device with respect to the contents of object dictionary. Its content should be conforming to CANopen standard, necessary to avoid incomplete or erroneous data sheets. Else the working model of the device in CANopen network may not running well as you expected. Use external test tool called CANchkEDS.exe to check the valid content of EDS first before importing EDS, or use a configuration utility to modify the content.

Example

Reference acoOpenCANPort() example.

acoGetNodeId

Get node identifier of the Master or Slave node object handle.

```
NODEID_t acoGetNodeId(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that is the returned by acoCreateNode()

Return Values

Return the node identifier of the node object handle that be set by acoNodeImportEDS() or acoSetNodeId(). Return CINVALID_NODEID if node identifier does not set yet.

Remarks

Use acoGetRemoteNodeId() to get remote slave node identifier.

The node identifier is unique identifier for each CANopen device in the network, a value in the range of [1..127] specified by CiA DS-301.

Example

```
extern HACONODE hSlaveNode;
```

```
	UNS32 NodeId;
	if (hSlaveNode)
		NodeId = acoGetNodeId(hSlaveNode);
```

acoSetNodeId

Assign a unique node identifier to the Master or Slave node object handle.

```
UNS32 acoSetNodeId(
   HACONODE hNode,
   NODEID_t NodeId
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

NodeId

[in] Specified a unique node identifier in a range of [1..127].

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

The node identifier is unique identifier for each CANopen device in the network, a value in the range of [1..127] specified by CiA DS-301.

Changing the node identifier, the related COB-ID included Node-ID of object dictionary will be updated too.

Example

```
	extern HACONODE hSlaveNode;
	UNS32 NodeId;
	if (hSlaveNode)
	{
		if(acoSetNodeId(hSlaveNode, 0x02) == ACOERR_SUCCESS)
		{
			NodeId = acoGetNodeId(hSlaveNode);
		}
```

```
    }
```

acoGetNodeState

Get the NMT state of the Master or Slave node object handle.

```
e_acoNodeState acoGetNodeState(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

Return Values

Return the NMT state of the Master or Slave node object handle.

Remarks

The NMT state of the Slave node will be auto into Pre-Operational state after calling acoStartNode(). Later the state of the Slave node can be changed by receiving the NMT messages from a NMT master. If using acoRegGeneralEvent() function to register a state changed event function, the event will be called by notify the state has been changed and understand which current state is in.

Use acoGetRemoteNodeState() to get remote slave node of NMT state.

Example

```
extern HACONODE hSlaveNode;
e_acoNodeState NodeState;
if (hSlaveNode)
{
      NodeState = acoGetNodeState(hSlaveNode);
      if(NodeState == ns_Operational)
      {
            printf("The node'state is in Operational state.");
      }
      else if(NodeState == ns_Pre_operational)
      {
            printf("The node'state is Pre_operational.");
      }
```

```
    }
```

acoStartNode

Start the Slave node into Initialization and then Pre-Operational state to join the CANopen network.

```
UNS32 acoStartNode(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Slave node object handle that returned by acoCreateNode().

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Before starting the Slave node, calls acoImportEDS() to import EDS of device profile and calls acoOpenCANPort() to open CAN port is necessary. The Slave node transmits a boot-up message to signal that it has entered the state Pre-operational after Initializing, then it joins the CANopen network. Depending on NMT state machine, the Slave node later waits the NMT command specifier from NMT master to switch its state.

Uses acoStopNode() function to enter the Stop state by local control.

Master node auto started by acoStartNetwork() and stopped by acoStopNetwork() that does not necessary to call acoStartNode() or acoStopNode().

Example

```
HACONODE hSlaveNode=NULL;
//Use acoOpenCANPort() to set CAN port name and baud rate.
hSlaveNode = acoCreateNode(NULL);
if (hSlaveNode)
{
      printf("Create Node SUCCESS.");
      if (acoNodeImportEDS(hSlaveNode, “\\slavedict-ok.eds”, 0x02) == ACOERR_SUCCESS)
      {
            printf("Import EDS file ok.");
```

```
                // should be ImportEDS first than OpenCANPort
                if (acoOpenCANPort(hSlaveNode, "CAN1:", 1000) == ACOERR_SUCCESS)
                {
                        printf("Open CAN1 ok.");
                        if (acoStartNode(hSlaveNode) == ACOERR_SUCCESS)
                                printf("StartNode success!");
                        else
                        {
                                printf("StartNode failed!");
                                acoCloseCANPort(hSlaveNode);
                                acoFreeNode(hSlaveNode);
                                hSlaveNode = NULL;
                        }
                }
                else
                {
                        printf("Open CAN1 failed.");
                        acoFreeNode(hSlaveNode);
                        hSlaveNode = NULL;
                }
        }
        else
        {
                printf("Import EDS file failed.");
                acoFreeNode(hSlaveNode);
                hSlaveNode = NULL;
        }
}
```

acoStopNode

Stop the slave node into Stop state by local control.

```
UNS32 acoStopNode(
   HACONODE hNode
);
```

Parameters

hNode

[in] Pointer to the Slave node object handle that returned by acoCreateNode().

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

The Slave node can be call acoStopNode() that enter into Stop state by local control. But also in CANopen network that means NMT master can transmit any NMT command to switch its state. Use acoCloseCANPort() function that can exit the CANopen network without receiving any CAN frames.

Master node is stopped by acoStopNetwork() function that does not necessary to call acoStartNode() or acoStopNode().

Example

```
extern HACONODE hSlaveNode;
If(hSlaveNode)
{
        acoStopNode(hSlaveNode);
        acoCloseCANPort(hSlaveNode);
        acoFreeNode(hSlaveNode);
        hSlaveNode = NULL;
 }
```

acoGetNodeRole

Identifier the role of a node is Master, Slave or remote node.

```
e_nodeRole acoGetNodeRole(
   HACONODEOBJ hNode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode() or pointer to remote node object handle that returned by acoNetworkAddSlave().

Return Values

Return the node role: Master, Slave, Remote Node or unknown role.

Example

```
extern HACONODE hSlaveNode;
if (acoGetNodeRole(hSlaveNode) == Master)
{
        //To do.
}
else if (acoGetNodeRole(hSlaveNode) == Slave)
{
        //To do.
}
```

acoGetODentry

Read an entry with specified index and sub-index from the local object dictionary. This function is a synchronous method, it returns the result until completed or timeout occurs.

```
UNS32 acoGetODentry(
   HACONODE hNode,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 *pDataType,
   UNS32 *pDataLen,
   void* pData
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode()

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to read.

pDataType

[out] the data type of the [index, sub-index] value. See data type constant in acodef.h.

pDataLen

[in, out] Input is the memory size of the pData allocated; output is the real data length of pData. The input data length should be equal to the length of [index, sub-index] value in object dictionary or zero for retrieving the data as expect.

pData

[out] the data of [index, sub-index] value in object dictionary. If input is NULL, do not output the value, return current data type and data length that the entry [index, sub-index] is.

Return Values

If the function succeeds, the return value is ACOOD_SUCCESSFUL. If the function fails, please reference acoapi error codes and SDO abort codes.

Remarks

According to the content of object dictionary, need to check access type before reading the entry. Does not allow to read the entry if the access type of [index, sub-index] is write-only (WO). Retrieve the data if the [index, sub-index] permission of access type is read-only (RO) or read-write (RW).

Use acoSetODentry() function to write the data of [index, sub-index] in local object dictionary.

Asynchronous method, acoReadSDO() and acoWriteSDO() function is to read/write the data of [index, sub-index] to remote object dictionary.

Example

```
extern HACONODE hSlaveNode;
UNS32 ret;
UNS8 data[16]={0};
UNS8 DataType=0;
UNS32 DataLen=8;
ret = acoGetODentry(hSlaveNode, 0x1017, 0, &DataType, &DataLen, &data);
if (ACOOD_SUCCESSFUL == ret)
        printf ("ReadObjDict succ: DataType=%x, DataLen=%d ", DataType, DataLen);
else
        printf ("ReadObjDict failed: %X", ret);
```

acoSetODentry

Write an entry to the local object dictionary. This function is a synchronous method, it is blocked and returns the result until the operation is complete or timeout occurs.

```
UNS32 acoSetODentry(
   HACONODE hNode,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 DataType,
   UNS32 DataLen,
   void* pData
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to write.

DataType

[in] the data type of the pData value. See data type constant in acodef.h.

DataLen

[in] the data length of the pData.

pData

[in] data

Return Values

If the function succeeds, the return value is ACOOD_SUCCESSFUL. If the function fails, please reference acoapi error codes and SDO abort codes.

Remarks

According to the content of object dictionary, need to check access type before writing data to the entry. Does not allow to update data of the entry if the access type of [index, sub-index] is read-only (RO). Can be writeable if the [index, sub-index] permission of access type is write-only (WO) or read-write (RW).

Use acoGetODentry() function to read the data of [index, sub-index] in local object dictionary.

Synchronous read/write object dictionary means that the method is blocked until the operation is complete, and then the method returns its data. Recommend that use synchronous method is well access in the local object dictionary, but use asynchronous method is better by access remote object dictionary. Reference acoReadSDO() and acoWriteSDO() functions.

Example

```
    extern HACONODE hSlaveNode;
    UNS32 ret;
    UNS8 data=255;
    UNS8 DataType=ACODT_UINT8;
    UNS32 DataLen=sizeof(data);

    ret = acoSetODentry(hSlaveNode, 0x1800, 2, DataType, DataLen, &data);
    if (ACOOD_SUCCESSFUL == ret)
        printf ("WriteObjDict succ: DataType=%x, DataLen=%d, Data=%x ", DataType, DataLen, data);
    else
        printf ("WriteObjDict failed: %X", ret);
```

acoReadSDO

Read an entry with specified index and sub-index from remote object dictionary of specified node identifier through SDO protocol. This function is an asynchronous method, it starts a task is returned immediately without waiting for a result. When the task finishes, the library notifies the application that the message was successfully processed and pass the result to the callback function.

```
UNS32 acoReadSDO (
   HACONODE hNode,
   NODEID_t NodeId,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 DataType,
   TCOnReadSDOResult pf
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode()

NodeId

[in] specified a SDO server node identifier, read its object dictionary

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to read.

DataType

[in] the data type of the [index, sub-index] value. See data type constant in acodef.h.

pf

[in] Pointer to the application-defined function of type TCOnReadSDOResult; represents the starting address of the function. The function will be executed if SDO request is responded or timeout occurs.

Return Values

If the function succeeds, the return value is ACOOD_SUCCESSFUL. If the function fails, please reference acoapi error codes or SDO abort codes.

Remarks

Asynchronous read/write object dictionary means that the function does not block the procedure, return immediately. Application can continue doing other work. To be informed the callback function when receiving the SDO responded or timeout occurs, pass the result of entry [index, sub-index] of object dictionary whose node identifier is NodeId to the callback function, application would get the value of entry and SDO abort code to understand the result of SDO request. If the operation is success, the abort code is ACOOD_SUCCESSFUL; otherwise, SDO abort code is returned that defined by CiA DS-301 v4.02. The node with specified node identifier must have the capability of SDO server; it is described in index 1200h – 127Fh of EDS.

According to the content of object dictionary, also need to check access type before reading the entry in remote object dictionary. Does not allow to read the entry if the access type of [index, sub-index] is write-only (WO). Retrieve the data if the [index, sub-index] permission of access type is read-only (RO) or read-write (RW).

Use acoWriteSDO() function to write the data of entry [index, sub-index] to remote object dictionary which specific node identifier.

A synchronous method, reference acoGetODentry() and acoSetODentry(). acoNetworkReadSDO() and acoNetworkWriteSDO() are for network object to request the remote slave node.

Example

```
extern HACONODE hSlaveNode;

void OnReadSDOResult(
            HACONODEOBJ hNode, NODEID_t NodeId,
            UNS16 wIndex, UNS8 bSubIndex, UNS8 dataType, UNS32 dataLen, void *pData, UNS32 abortCode,
LPVOID pvArg)
{
      if (ACOOD_SUCCESSFUL == abortCode) {
            UNS8 *ptr = (UNS8 *)pData;
            printf("ReadSDOResult: succ: [%04x:%02x] DataLen=%d, Data=%02x" \
                  , wIndex, bSubIndex, dataLen, *ptr);
      } else
            printf("ReadSDOResult failed: %x", abortCode);
```

```
    }



    UNS32 ret;
    UNS8 DataType=ACODT_UINT8;
    NODEID_t OtherNode=0x05;
    //*** access Object Dictionary: async method
    ret = acoReadSDO(hSlaveNode, OtherNode, 0x1800, 0x02, DataType, OnReadSDOResult, NULL);
    if (ACOOD_SUCCESSFUL == ret)
            printf ("acoReadSDO succ.");
    else
            printf ("acoReadSDO failed: %X", ret);
```

acoWriteSDO

Write the value at the index and sub-index to remote object dictionary of the node whose node identifier is NodeId through SDO protocol. This function is an asynchronous method, it starts a task is returned immediately without waiting for a result. When the task finishes, the function notifies the application that the message was successfully processed and pass the result to the callback function.

```
UNS32 acoWriteSDO (
   HACONODE hNode,
   NODEID_t NodeId,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 DataType,
   UNS32 DataLen,
   void* pData
   TCOnWriteSDOResult pf
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode()

NodeId

[in] specified a SDO server node identifier, write to its object dictionary

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to write.

DataType

[in] the data type of the pData value. See data type constant in acodef.h.

DataLen

[in] the data length of pData.

pData

[in] the data of [index, sub-index] value.

pf

[in] Pointer to the application-defined function of type TCOnWriteSDOResult; represents the starting address of the function. The function will be executed if SDO request is responded or timeout occurs.

Return Values

If the function succeeds, the return value is ACOOD_SUCCESSFUL. If the function fails, please reference acoapi error codes and SDO abort codes.

Remarks

Asynchronous read/write object dictionary means that the function does not block the procedure, return immediately. Application can continue doing other work. To be informed the callback function when receiving the SDO responded or timeout occurs, pass the result to the callback function, application would get the SDO abort code to understand the result of SDO request. If the operation is success, the abort code is ACOOD_SUCCESSFUL; otherwise, SDO abort code is returned defined by CiA DS-301 v4.02. The node with specified node identifier must have the capability of server SDO; it is described in index 1200h – 127Fh of EDS.

According to the content of object dictionary, also need to check access type before writing the data to the entry in remote object dictionary. Does not allow to write the entry if the access type of [index, sub-index] is read-only (RO). Access type should be write-only (WO) or read-write (RW).

Use acoReadSDO() function to read the data of entry [index, sub-index] from remote object dictionary which specific node identifier.

A synchronous method, reference acoGetODentry() and acoSetODentry(). acoNetworkReadSDO() and acoNetworkWriteSDO() are for network object to request the remote slave node.

Example

```
void OnWriteSDOResult(
            HACONODEOBJ hNode, NODEID_t NodeId,
            UNS16 wIndex, UNS8 bSubIndex, UNS32 abortCode, LPVOID pvArg)
{
        if (ACOOD_SUCCESSFUL == abortCode) {
                UNS8 *ptr = (UNS8 *)pData;
                printf("WriteSDOResult: succ: [%04x:%02x] ", wIndex, bSubIndex);
        } else
                printf("WriteSDOResult failed: [%04x:%02x] err: %x", wIndex, bSubIndex, abortCode);
}

void main()
{
        UNS32 ret;
        UNS8 transType=0x255;
        UNS8 DataType=ACODT_UINT8;
        NODEID_t OtherNode=0x05, NodeId=0x02;
        HACONODE hSlaveNode = acoCreateNode(NULL);
        if (ACOERR_SUCCESS == (ret = acoNodeImportEDS(hSlaveNode, “slave.eds”, &NodeId)) )
        {
                if (ACOERR_SUCCESS == (ret = acoOpenCANPort(hSlaveNode, “CAN1:”, 1000)))
                {    // should be ImportEDS first than OpenCANPort
                        if (ACOERR_SUCCESS == (ret = acoStartNode(hSlaveNode)))
                        {
                                //*** access Object Dictionary: async method
                                ret = acoWriteSDO(hSlaveNode, OtherNode, 0x1800, 0x02,
                                                DataType, sizeof(transType), &transType, OnWriteSDOResult, NULL);
                                if (ACOOD_SUCCESSFUL == ret)
                                        printf ("acoWriteSDO succ.");
                                else
                                        printf ("acoWriteSDO failed: %X", ret);
                        }
                        acoStopNode(hSlaveNode);
                        acoCloseCANPort(hSlaveNode);
                }
                acoFreeNode(hSlaveNode);
                hSlaveNode = NULL;
        }
}
```

acoRegODDataChangedEvent

Register an event for the local object dictionary entry; event will be called while the data of entry [index, sub-index] is changed, pass the changed data of the entry to the application-defined function.

```
UNS32 acoRegODDataChangedEvent(
   HACONODE hNode,
   UNS16 wIndex,
   UNS8 SubIndex,
   TCOnODDataChangedEvent cb,
   LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to monitor.

cb

[in] Pointers to the application-defined function to execute by the library while the data of the entry [index, sub-index] of the object dictionary being changed.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Application can monitor the data of local object dictionary with specific index and sub-index through register a callback function without polling. If the data is changed without respect to by local or by remote SDO Client, application can get notify and to do something. But if register too many callback function for the data changed, it decreases the program efficiency.

Use acoUnRegODDataChangedEvent() to un-register the callback function.

Example

```
UNS32 OnODDataChanged(HACONODE hNode, UNS16 wIndex, UNS8 SubIndex,
                      UNS8 DataType, UNS32 DataLen, void * pData, LPVOID pvArg)
{
    //To do.
    Return 0;
}
void RegNodeEvents(void)
{
    acoRegODDataChangedEvent(hSlaveNode, 0x1800, 0x02, OnODDataChanged, NULL);
}
```

acoUnRegODDataChangedEvent

Un-register an event for the local object dictionary entry, do not get notify if the data is changed.

```
UNS32 acoUnRegODDataChangedEvent(
   HACONODE hNode,
   UNS16 wIndex,
   UNS8 SubIndex
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

wIndex, SubIndex

[in] Which index and subindex of the object dictionary

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
void UnRegNodeEvent()
{
    acoUnRegODDataChangedEvent(hSlaveNode, 0x1800, 0x02);
}
```

acoRegGeneralEvent

Register a callback function for general event that the node is in state changed, after transmitting or receiving SYNC message or after transmitting PDO message synchronously.

```
UNS32 acoRegGeneralEvent(
   HACONODE hNode,
   e_GeneralEventType et,
   TCOnGeneralEvent cb,
   LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

et

[in] which event type being trigger an callback function. The event type can be node in initialization state, preoperational state, and operational state, stop state, post SYNC and post PDO.

Notify event type	Description
et_STATE_INITIALIZE	the node is into NMT state Initialisation
et_STATE_PREOPERATIONAL	the node is into NMT state Pre-operational
et_STATE_OPERATIONAL	the node is into NMT state Operational
et_STATE_STOPPED	the node is into NMT state Stopped
et_STATE_SYNC	the node transmits or receives a sync frame
et_STATE_POST_PDO	the node transmits a PDO frame

cb

[in] Pointer to the application-defined function to be called by the library while specific the event is triggered.

typedef void (*TCOnGeneralEvent)(HACONODE hNode, LPVOID pvArg);

hNode argument is pointer to Master or Slave node object handle; indicates which the node enters this event.

pvArg argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Master or Slave node can be aware its current state by notify automatically, do not need to call acoGetNodeState(). In application-defined NMT state-changed event callback function, application can do something in the state. For example, Master can change transmission type of slave Transmit PDO as it's expects in state pre-operational callback function.

Example

```
extern HACONODE hSlaveNode;


void DoStateInitialization(HACONODE hNode, LPVOID pvArg)
{
        //To do.
}
void DoStatePreoperational(HACONODE hNode, LPVOID pvArg)
{
        //To do.
}
void DoStateOperational(HACONODE hNode, LPVOID pvArg)
{
        //To do.
}
void DoStateStop(HACONODE hNode, LPVOID pvArg)
{
        //To do.
}
void RegNodeEvents(void)
{
        acoRegGeneralEvent(hSlaveNode, et_STATE_INITIALIZE, DoStateInitialization, NULL);
        acoRegGeneralEvent(hSlaveNode, et_STATE_PREOPERATIONAL, DoStatePreoperational, NULL);
        acoRegGeneralEvent(hSlaveNode, et_STATE_OPERATIONAL, DoStateOperational, NULL);
        acoRegGeneralEvent(hSlaveNode, et_STATE_STOPPED, DoStateStop, NULL);
}
```

acoSendData

Transmit a raw message to other nodes.

```
UNS32 acoSendData(
    HACONODE hNode,
    UNS32 CobId,
    bool bRTR,
    UNS32 dataLen,
    void * pData
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

CobId

[in] Specified Cob-ID of the CAN message. The format of Cob-ID is defined by CiA DS-301 v4.02:

10			7	6						0
Function Code				Node-ID						

bRTR

[in] Identify the message is Remote Transmission Request or not

dataLen

[in] the length of the pData, maximum length is 8 in CAN message.

pData

[in] the value to be sent

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
// Send NMT Node Guarding to Node 2
acoSendData(hSlaveNode, 0x702, 0, 0, NULL);
```

acoSendEmcy

Transmit an emergency message from an emergency producer on the CANopen device. Emergency objects are triggered by occurrence of a CANopen device internal error situation, and the error register (1001h) are specified;

the messages shall contain the error field with pre-defined error and additional information.

```
UNS32 acoSendEmcy(
   HACONODE hNode,
   UNS16 errCode,
   UNS8 errRegMask,
   UNS16 addInfo
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

errCode

[in] emergency error codes

errRegMask

[in] This value is filled in at the location of object 1001h (error register). That means it must include object 1001h in EDS.

Bit	M/O	Meaning
0	M	Generic error
1	O	Current
2	O	Voltage
3	O	Temperature
4	O	Communication error (overrun, error state)
5	O	Device profile specific
6	O	Reserved (always 0)
7	O	Manufacturer-specific

addInfo

[in] Application-specific additional information is defined by other profile specifications.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

If CAN device internal triggers an error interrupt, acoapi library transmits an emergency message with FF00h device specific error in errCodes, 11h communication error in error register. May use acoRegODDataChangedEvent() to monitor object 1001h, 00h to understand whether emergency object is triggered and CAN device internal error occurs. The new error codes will be filled in at the top of the array of error codes (object 1003h) if supported.

After transmitting an emergency message, need to use acoResetEmcy() function to reset error, the CANopen device enters the error free state and transmit an 'reset error / no error' emergency object. For example, a node generates a temperature emergency (measured temperature exceeds the limits), only when the temperature has returned within limits the node transmit another emergency message, this time clearing/resetting the temperature emergency.

Use acoRegEmcyEvent() to register an event for the emergency consumer that the callback function will be called if receiving a EMCY message.

Example

```
// Send emergency message active: temperature
acoSendEmcy (hSlaveNode, 0x4200, 0x08, 0);
```

acoResetEmcy

Deletes specific error code and clears corresponding bits in Error register. If all errors are clear, the CANopen device enters the error Free State.

```
UNS32 acoResetEmcy(
   HACONODE hNode,
   UNS16 errCode
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

errCode

[in] emergency error codes

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

After transmitting an emergency message, need to use acoResetEmcy() function to reset error, the CANopen device enters the error free state and transmit an 'reset error / no error' emergency object.

Example

```
// clear error 0x4200: temperature error
acoResetEmcy (hSlaveNode, 0x4200);
```

acoRegEmcyEvent

One or more emergency consumers (master or slaves) register an event for trigger if receiving an emergency message from emergency producer (other nodes.)

```
UNS32 acoRegEmcyEvent(
   HACONODE hNode,
   TCOnPostEmcyEvent pf,
   LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

pf

[in] Pointer to the application-defined function of type TCOnPostEmcyEvent; represents the starting address of the function. The function would to be called by the library while the node receives an emergency message. TCOnPostEmcyEvent is a placeholder for the application-defined function name.

```
typedef void (*TCOnPostEmcyEvent)(HACONODE hNode, NODEID_t NodeID, UNS16 errCode, UNS8 errReg,
UNS8 errManufacField[5], LPVOID pvArg);
```

hNode argument [in] is the pointer to the node object handle is the same as *hNode* argument of acoRegEmcyEvent() as a emergency consumer. *NodeID* argument [in] is the node identifier that which the slave node producers the emergency message. *errCode*, *errReg* and *errManufacField* argument [in] is emergency error codes and error register that the slave node producers. *pvArg* argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
    void DoRecvEmcyEvent(HACONODE hNode, NODEID_t NodeID, UNS16 errCode, UNS8 errReg, UNS8 
errManufacField[5], LPVOID pvArg)
    {
        printf("Receives an emergency message from Node[%d]: %04x %02x\r\n"), NodeID, errCode, errReg);
    }

    acoRegEmcyEvent(mp_hSlaveNode, DoRecvEmcyEvent, NULL);
```

acoRegHeartbeatEvent

Heartbeat consumer (the node) registers an event for trigger if receiving a heartbeat message from heartbeat producer (remote slave nodes.)

```
UNS32 acoRegHeartbeatEvent (
  HACONODE hNode,
  TCOnHeartbeatEvent cb,
  LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

cb

[in] Pointer to the application-defined function of type TCOnHeartbeatEvent; represents the starting address of the function. The function would to be called by the library while master node receives a heartbeat message.

```
typedef void (*TCOnHeartbeatEvent)(HACONODE hNode, NODEID_t NodeId, e_acoNodeState currState, 
LPVOID pvArg);
```

hNode argument [in] is the pointer to the node object handle as a heartbeat consumer. *NodeID* argument [in] is the node identifier that which the remote slave node producers the heartbeat message, *currState* argument [in] is the current NMT state of the remote slave node. *pvArg* argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

A heartbeat producer transmits a heartbeat message cyclically; so the application-defined function would be called cyclically. Therefore, be carefully using this function that may cause the system decrease the efficiency.

Through heartbeat message, can know the remote node it's presently NMT state, generally used in NMT master node.

The heartbeat producer time and heartbeat consumer time can be configurable via object dictionary (EDS). The heartbeat protocol starts if the time value unequal to 0.

Example

```
extern HACONODE hMasterNode;

void DoHeartbeatEvent(HACONODE hNode, NODEID_t NodeId, e_acoNodeState currState, LPVOID pvArg)
{
        printf("The remote node [%02x] is in state %x.", NodeId, currState);
}
acoRegHeartbeatEvent(hMasterNode, DoHeartbeatEvent, NULL);
```

acoRegHeartbeatErrorEvent

Register an event for heartbeat error in network object; event will be called while not receive heartbeat message within the heartbeat consumer time.

```
UNS32 acoRegHeartbeatErrorEvent(
   HACONODE hNode,
   TCOnHeartbeatErrorEvent cb,
   LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

cb

[in] Pointer to the application-defined function of type TCOnHeartbeatErrorEvent; represents the starting address of the function. The function would to be called by the library while master node not receives a heartbeat message within heartbeat consumer time, timeout occurs.

typedef void (*TCOnHeartbeatErrorEvent)(HACONODE hNode, NODEID_t NodeId, LPVOID pvArg);

hNode argument [in] is the pointer to the node object handle as a heartbeat consumer. *NodeID* argument [in] indicates is the node identifier that does not receive a heartbeat message from the node within the heartbeat consumer time. *pvArg* argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

The configuration of about heartbeat consumer must be in object dictionary of object 1016h and sub-index 01h above, so that the node just could be a heartbeat consumer. The value of heartbeat consumer time indicates the expected heartbeat cycle times. If the heartbeat time is equal to 0, no heartbeat error event will be triggered. If the heartbeat time is unequal 0, a heartbeat error event will be generated if the heartbeat is not received within the heartbeat consumer time.

Example

```
extern HACONODE hMasterNode;

// CALLBACK function for Heartbeat error
void DoHeartbeatErrorEvent(HACONODE hNode, unsigned char id, LPVOID pvArg)
{
        printf("Node[%x] Heartbeat error!", id);
}

//Register callback function for Heartbeat error event
acoRegHeartbeatErrorEvent(hMasterNode, DoHeartbeatErrorEvent, NULL);
```

acoRegRecvPDOEvent

Register a event is for receiving a PDO message; This function should be called every time a PDO has received from foreign nodes.

```
UNS32 acoRegRecvPDOEvent(
   HACONODE hNode,
   TCOnRecvPDOEvent cb,
   LPVOID pvArg
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

cb

[in] Pointer to the application-defined function of type TCOnRecvPDOEvent; represents the starting address of the function. The function would to be called by the library while the node receives a PDO message.

typedef void (*TCOnRecvPDOEvent)(HACONODE hNode, int rpdoIndex, UNS32 cobId, UNS8 bRTR, UNS8 size, void *pData, LPVOID pvArg);

hNode argument [in] is the pointer to the node object handle. *rpdoIndex* argument [in] indicates Which RPDO offset [0..511] of the object dictionary [1400..15FF] to received. *cobId* argument [in] indicates the COB-ID of the RPDO message. *bRTR* argument [in] is the RTR flag of RPDO message. *size* argument [in] is the data length of pData in bytes. *pData* argument [in] is the data of RPDO message. *pvArg* argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Example

```
extern HACONODE hMasterNode;
```

```
        // CALLBACK function for RPDO
        void OnRecvPDOEvent(HACONODE hNode, int rpdoIndex, UNS32 cobId, UNS8 bRTR, UNS8 size, void *pData,
LPVOID pvArg)
        {
                int i, j;
                unsigned char *ptr = (unsigned char *)pData;
                printf("RPDO[%d]: %04X; ", rpdoIndex+1, cobId);
                for (i=0; i<size; i++)
                        printf("%02X ", ptr[i]);
        }

        //Register callback function for receiving PDO event
        acoRegRecvPDOEvent(hMasterNode, OnRecvPDOEvent, NULL);
```

acoSendPDOwithCOS

Send PDOs if any data has changed (Change-Of-State), only transmission type 255 (0xFF) is supported yet. This function should be called every time a PDO has changed and sent to foreign nodes immediately.

```
UNS32 acoSendPDOwithCOS(
   HACONODE hNode,
   int tpdoIndex
);
```

Parameters

hNode

[in] Pointer to the Master or Slave node object handle that returned by acoCreateNode().

tpdoIndex

[in] Which TPDO offset [0..511] of the object dictionary [1800..19FF] to be sent if the data has changed, or -1 for all TPDO.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

The change-of-state transmission method simply transmits a TPDO message if the process data in it changes and transmission type is 255 (0xFF). But it could be affected by inhibited time if it exists. Even the data has been changed but it may not send the TPDO until the inhibit time expired. So if the process data actually changes several times while the timer is running not all of these changes will be transmitted, that means potentially some process data is lost.

Example

```
// Send PDO immediately if transmission type is 255
int tpdoOffset = 0;
ret = acoSendPDOwithCOS(hSlaveNode, tpdoOffset);
```

acoCreateNetwork

Create a CANOpen network object in NMT master with the requested attributes.

```
HACONETWORK acoCreateNetwork(
   HACOMASTER hMasterNode
);
```

Parameters

hMasterNode

[in] Assign a Master node for this CANopen network. A node created by acoCreateNode().

Return Values

If the function succeeds, the return value is a network object handle. If the function fails, the return value is NULL. The generality of failed reason could be allocating memory failed.

Network object includes a Master node and remote node set. Master node should be ready first before creating network object. Use acoFreeNetwork() to free network object.

Example

```
void DoBootupEvent(HACONODE hNode, NODEID_t id);

HACONETWORK hNetworkObj;
HACOMASTER    hMasterNode;
NODEID_t          MasterNode=0x01;
hMasterNode = acoCreateNode(NULL);
if (hMasterNode == NULL)     return;
if (ACOERR_SUCCESS == acoNodeImportEDS(hMasterNode, "Master.eds", &MasterNode))
{
        hNetworkObj = acoCreateNetwork(hMasterNode);
        if (hNetworkObj)
        {
                // register some events
                acoRegBootupEvent(hNetworkObj, DoBootupEvent);
                if (ACOERR_SUCCESS == acoOpenCANPort(hMasterNode, "CAN1:", 1000))
                {
                        acoStartNetwork(hNetworkObj);
                        // do something…
                        acoStopNetwork(hNetworkObj);
                }
                acoFreeNetwork(hNetworkObj);
        }
}
acoFreeNode(hMasterNode);

void DoBootupEvent(HACONODE hNode, NODEID_t id)
{
        if (!acoRemoteNodeExist(hNetworkObj, id))
        { // not exist in list
                HACOREMOTE pRemoteNode = acoNetworkAddSlave(hNetworkObj, id, NULL);
                if (pRemoteNode)
                {
                        // change RxPDO setting to receive Slave's PDO
                        UINT32 cobid=0x180 + id;
                        acoSetODentry(hMasterNode, 0x1400, 0x01, ACODT_UINT32, sizeof(cobid), &cobid);
                }
        }
        // Start remote slave node into Operational state
        acoStartRemoteNode(pNetworkObj, id);
}
```

acoFreeNetwork

Free CANopen network object and its remote nodes list. Before release the network object, it will stop all remote nodes first by broadcasting NMT stop message and then stop the master node.

```
void acoFreeNetwork(
   HACONETWORK hNetworkObject
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that return by acoCreateNetwork().

Return Values

None

acoStartNetwork

Start the CANopen network to operation, start master node enter NMT state Operational and then start all remote nodes in remote list by broadcasting NMT start remote node message.

```
UNS32 acoStartNetwork(
   HACONETWORK hNetworkObject
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Start CANopen network, master firstly enter NMT state Operational, and continue to demand other remote slave nodes enter NMT state Operational if online. Offline remote slave nodes can be demanded if receiving the boot up message, reference acoRegBootupEvent(). Call acoStartRemoteNode() only start remote slave node.

All of nodes must be in NMT state Operational can transmit PDO object. In other words, any process data can be transferred the node must be in state Operational; else the process data can not be transferred.

Use acoStopNetwork() to stop CANopen network.

Table 12: NMT states and communication objects

State / Object	Pre-operational	Operational	Stopped
PDO		V	
SDO	V	V	
SYNC	V	V	
TIME	V	V	
EMCY	V	V	
NMT	V	V	V

Example

Reference to acoCreateNetwork() example.

acoStopNetwork

Stop the CANopen network, stop all remote nodes in remote list by broadcasting NMT stop message and then stop master node, all of nodes would be enter NMT state Stopped.

```
UNS32 acoStartNetwork(
   HACONETWORK hNetworkObject
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

Reference to acoCreateNetwork() example.

acoGetRemoteNodeCount

Return the number of remote nodes in network object of remote node list.

```
UNS32 acoGetRemoteNodeCount(
   HACONETWORK hNetworkObject
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

Return Values

Return the number of remote nodes in network object.

acoGetRemoteNode

Get remote node object handle by specified node identifier in remote node list.

```
HACOREMOTE acoGetRemoteNode(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

NodeId

[in] the node identifier of remote node.

Return Values

If the remote node with specified node identifier is exist in network object of remote node set, return value is remote node handle. If it is not existing, return value is NULL.

Remarks

According to remote node object handle, use acoGetRemoteNodeId() function to get node identifier or use acoGetRemoteNodeState() to get the current state of the remote node.

acoGetRemoteNodebyPos

Get remote node object at a specified position in remote node list.

```
HACOREMOTE acoGetRemoteNodebyPos(
   HACONETWORK hNetworkObject,
   size_t _Pos
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

_Pos

[in] the index number of the remote node. The base is 0.

Return Values

The return value specifies the remote node object in the remote node list of position. If the position is over the number of remote nodes, return value is NULL.

Example

```
extern HACONETWORK hNetworkObj;

void DoMasterInitialization(HACOMASTER hMaster)
{
        // Update Master Local Object Dictionary : RPDO 0x1400~0x1403
        // to Receive TPDO which node id allowed
        UNS32 CcobID[] = {0x200, 0x300, 0x400, 0x500};
        for (unsigned int i=0; i<min(4, acoGetRemoteNodeCount(hNetworkObj)); i++)
        {
                HACOREMOTE hRemote = acoGetRemoteNodebyPos(hNetworkObj, i);
                UNS32 cob=CcobID[i] + acoGetRemoteNodeId(hRemote);
                acoSetODentry(hMaster, 0x1400+i, 1, ACODT_UINT32, sizeof(cob), &cob);
        }
}
```

acoGetRemoteNodeId

Get node identifier of the remote node object handle.

```
NODEID_t acoGetRemoteNodeId(
   HACOREMOTE hRemoteNode
);
```

Parameters

hRemoteNode

[in] Pointer to remote node object handle.

Return Values

Return the node identifier of the remote node object handle that be set by acoNetworkAddSlave(). Return CINVALID_NODEID if remote node object is invalid.

Remarks

The node identifier is unique identifier for each CANopen device in the network, a value in the range of [1..127] specified by CiA DS-301.

Example

Reference to acoGetRemoteNodebyPos() example.

acoRemoteNodeExist

Check the node identifier whether exist in remote node list or not.

```
BOOL acoRemoteNodeExist(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

NodeId

[in] Specified a unique node identifier in a range of [1..127].

Return Values

Nonzero indicates the remote node with specified node identifier exists in remote node list of network object. Zero indicates the node identifier does not exist in remote node list.

Example

Reference to acoCreateNetwork() example.

acoNetworkAddSlave

NMT master creates a remote node object with the requested attributes and inserts it to the remote node set of the network object.

```
HACOREMOTE acoNetworkAddSlave(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId,
   char *pFileName
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

NodeId

[in] a node identifier.

pFileName

[in] Device profile path and file name of remote slave node, it can be NULL.

Return Values

If the function succeeds, the return value is a remote node object handle. If the function fails, the return value is NULL, Call GetLastError() to get error code.

Remarks

The node identifier should be a unique value in a range of [1..127] defined by CiA DS-301. So that the maximum count of the remote nodes in network object is 126, one node identifier is reserved for Master node. If the number is over the limit, get error code.

Inserting a node to remote node list, the remote node list in network object would re-sort by node identifier so that the memory location would be rearranged. If the application keeps the pointer of the remote node, that caused to pointer to different remote node. Therefore, recommends that call acoGetRemoteNode() or acoGetRemoteNodebyPos() each time to retrieve the current pointer of the remote node object from the list, do not keep the pointer of the remote node.

Example

Reference to acoCreateNetwork() example.

acoNetworkRemoveSlave

NMT master removes the remote node object identified by NodeId from the remote node list of the network object. NMT master demands the remote node would be enter the NMT state Stopped and then remove it from list.

```
UNS32 acoNetworkRemoveSlave(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

NodeId

[in] a node identifier in a range of [1..127].

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

Before removing the remote node from list in network object, firstly transmitting NMY message to the remote slave node entered NMT state Stopped and then remote it from list. The remote node list in network object would re-sort by node identifier so that the memory location would be rearranged. If the application keeps the pointer of the remote node, that caused to pointer to different remote node. Therefore, recommends that call acoGetRemoteNode() or acoGetRemoteNodebyPos() each time to retrieve the current pointer of the remote node object from the list, do not keep the pointer of the remote node.

Example

```
extern HACONETWORK hNetworkObj;
// remove node identifier 0x02 from list
acoNetworkRemoveSlave(hNetworkObj, 0x02);
```

acoNetworkSetState

NMT master change the remote slave node to new state by NMT message. Node identifier is 0 means broadcasting NMT message to all remote slave nodes to change its state.

```
UNS32 acoNetworkSetState(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId,
   e_acoNodeState newState
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

NodeId

[in] the remote slave node identifier in range [0..127], 0 means all remote slave nodes.

newState

[in] the new NMT state.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
      extern HACONETWORK hNetworkObj;
      // Set remote node 0x02 to NMT state Operational
      acoNetworkSetState(hNetworkObj, 0x02, ns_Operational);
```

acoGetRemoteNodeState

Get the NMT state of the specified remote node object handle that kept in NMT master side, not really remote slave node state.

```
e_acoNodeState acoGetRemoteNodeState(
   HACOREMOTE hRemoteNode
);
```

Parameters

hNetworkObject

[in] Pointer to network object handle that returned by acoCreateNetwork().

Return Values

Return the currently NMT state of the remote node. If return value is ns_Unknown_state, indicates the remote slave node may be offline that can not response any message or just request its node guard message and waiting its NMT state.

acoReqNodeGuard

NMT master request the remote slave node an extra a node guard message. If the slave replies the node guard message, the state of the remote node will be updated in network object.

```
UNS32 acoReqNodeGuard(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

NodeId

[in] the remote slave node identifier in range [0..127], 0 means all remote slave nodes.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

NMT master application transmit guarding requests single to remote slave node or all nodes, the state of asked remote nodes in network object would be into ns_Unknown_state state, update the newly state until the guarding is achieved if the slave node supports node guarding protocol. For a later a cycle of CAN message, Call acoGetRemoteNodeState() to get the latest state of remote slave node.

The state of remote slave nodes update cyclically if the device supports the heartbeat mechanism, does not need to request the node guarding alone. The heartbeat mechanism for a device is established by cyclically transmitting the heartbeat message by the heartbeat producer. If the heartbeat cycle fails for the heartbeat producer, the heartbeat consumer will be informed the event, reference acoRegHeartbeatErrorEvent().

Example

```
extern HACONETWORK hNetworkObj;
extern HACOREMOTE hRemoteNode;


//Set remote node to Operational
acoReqNodeGuard (hNetworkObj, 0x02);
// wait a moment for the guarding message responsed
Sleep(1000L);
e_acoNodeState NodeState = acoGetRemoteNodeState(hRemoteNode);
switch(NodeState)
{
    case ns_Initialisation:
            printf("Remote node state: Initialisation");
            break;
    case ns_Pre_operational:
            printf("Remote node state: Pre_operational");
            break;
    case ns_Operational:
            printf("Remote node state: Operational");
            break;
    case ns_Stopped:
            printf("Remote node state: Stopped");
            break;
    default:
            printf("Remote node state: invalid state");
            break;
}
```

acoStartRemoteNode

The NMT master uses the NMT service start remote node specified by node identifier to change the NMT state. The state of the remote slave node shall be entering NMT state Operational.

```
UNS32 acoStartRemoteNode(
  HACONETWORK hNetworkObject,
  NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

NodeId

[in] the remote slave node identifier in a range [0..127]. CBROADCAST_NODEID (0) means broadcast NMT start remote node message to all online remote nodes.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

By broadcasting NMT start remote node message, all remote slave nodes shall be enter NMT state Operational even if the remote node is not exist in the remote node list of network object. Therefore, application needs to maintain the remote node list in the network object by self. The service start remote node is unconfirmed, can not get really the NMT state after starting, application can use other method to know which state the remote node in, for example: heartbeat protocol.

Need to use acoStartNetwork() firstly to start CANopen network and master node. Use acoStopRemoteNode() to stop a remote node.

Example

```
extern HACONETWORK hNetworkObj;

//Start remote node 0x02
acoStartRemoteNode(hNetworkObj, 0x02);
```

acoStopRemoteNode

The NMT master uses the NMT service stop remote node specified node specified to change NMT state. The state of the remote slave node shall be entering NMT state Stopped.

```
UNS32 acoStopRemoteNode(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

NodeId

[in] the remote slave node identifier in a range [0..127]. CBROADCAST_NODEID (0) means broadcast NMT start remote node message to all online remote nodes.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Example

```
extern HACONETWORK hNetworkObj;

//Start remote node 0x02
acoStopRemoteNode(hNetworkObj, 0x02);
```

acoRegBootupEvent

Register an event for receiving Slave boot up message in network object; event will be called while master receiving a slave boot up message.

```
UNS32 acoRegBootupEvent(
   HACONETWORK hNetworkObject,
   TCOnBootupEvent cb,
   LPVOID pvArg
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

cb

[in] Pointer to the application-defined function of type TCOnBootupEvent; represents the starting address of the function. Library will call this function while the master receiving a slave node boot up frame.

typedef void (*TCOnBootupEvent)(HACONODE hNode, NODEID_t NodeId, LPVOID pvArg);

hNode argument [in] is the pointer to the master node object handle. *NodeID* argument [in] is the slave node with the node identifier transmits a boot-up message. *pvArg* argument [in] pointer to an argument that is passed through from the callback function.

pvArg

[in] Pointer to the data to be passed to the callback function or pointer to NULL; the type is a void that allows the application to declare, define, and initialize a structure or argument to hold any information.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

The NMT slave transmits a boot-up message to indicate the state transition occurred from the NMT state Initialisation to the NMT state Pre-operational. It is ready to CANopen network. In application-defined function, Master can change the object dictionary settings of remote slave node by SDO request, or request the remote slave node enter the NMT state Operational to start operation.

Example

Reference acoCreateNetwork() example.

acoNetworkReadSDO

Read an entry with specified index and sub-index from remote object dictionary of specified node identifier through SDO protocol. This function is an asynchronous method, it starts a task is returned immediately without waiting for a result. When the task finishes, the library notifies the application that the message was successfully processed and pass the result to the callback function.

```
UNS32 acoNetworkReadSDO(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 DataType,
   TCOnReadSDOResult pf
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

NodeId

[in] specified a SDO server node identifier, read its object dictionary

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to read.

DataType

[in] the data type of the [index, sub-index] value. See data type constant in acodef.h.

pf

[in] Pointer to the application-defined function of type TCOnReadSDOResult; represents the starting address of the function. The function will be executed if SDO request is responded or timeout occurs.

Return Values

If the function succeeds, the return value is ACOERR_SUCCESS. If the function fails, please reference acoapi error codes.

Remarks

This asynchronous function is for the master node asks other remote slave node the value of object dictionary with object index and sub-index. The application-defined function of type TCOnReadSDOResult will be called if receiving the responded value of specified wIndex and sub-index from the remote slave node as SDO server. Therefore, SDO parameters should be described in both master node and remote slave node at object dictionary of object 1280h to 12FFh SDO client parameter, and object 1200h to 127Fh SDO server parameter. Else there is not receiving any responded message from the remote slave node.

typedef void (*TCOnReadSDOResult)(HACONODEOBJ hNode, NODEID_t NodeId, UNS16 wIndex, UNS8 SubIndex, UNS8 dataType, UNS32 dataLen, void* pData, UNS32 abortCode);

hNode argument [in] is the pointer to the remote slave node object handle with node identifier is *NodeID*. *NodeID* argument [in] is the remote slave node identifier. *wIndex* and *SubIndex* arguments [in] indicate which object to be read. *dataType* argument [in] indicates the data type of the pData. *dataLen* argument [in] is the data length of pData. *pData* argument [in] is the data of object [index, sub-index]. *abortCode* argument is the return value, ACOOD_SUCCESSFUL means success, else get fails, reference acoapi error codes and SDO abort codes.

Reference acoReadSDO() and acoWriteSDO().

Example

```
extern HACONETWORK hNetworkObj;

void OnReadSDOResult(
            HACONODEOBJ hNode, NODEID_t NodeId,
            UNS16 wIndex, UNS8 bSubIndex, UNS8 dataType, UNS32 dataLen, void *pData, UNS32 abortCode)
{
    if (ACOOD_SUCCESSFUL == abortCode) {
```

```
                UNS8 *ptr = (UNS8 *)pData;
                printf("ReadSDOResult: succ: Remote node [%02x] object [%04x:%02x] DataLen=%d, Data=%02x"
                    , NodeId, wIndex, bSubIndex, dataLen, *ptr);
            } else
                printf("ReadSDOResult failed: %x", abortCode);
    }

    //*** access Object Dictionary: async method
    acoNetworkReadSDO(hNetworkObj, 0x02, 0x1800, 0x02, ACODT_UINT8, OnReadSDOResult);
```

acoNetworkWriteSDO

Write the value at the index and sub-index to remote object dictionary of the node whose node identifier is NodeId through SDO protocol. This function is an asynchronous method, it starts a task is returned immediately without waiting for a result. When the task finishes, the function notifies the application that the message was successfully processed and pass the result to the callback function.

```
UNS32 acoNetworkWriteSDO(
   HACONETWORK hNetworkObject,
   NODEID_t NodeId,
   UNS16 wIndex,
   UNS8 SubIndex,
   UNS8 DataType,
   UNS32 DataLen,
   void *pData,
   TCOnWriteSDOResult pf
);
```

Parameters

hNetworkObject

[in] Pointer to network handle that created by acoCreateNetwork().

NodeId

[in] specified a SDO server node identifier, write to its object dictionary

wIndex, SubIndex

[in] Which index (16bits) and sub-index (8bits) of entry in the object dictionary want to write.

DataType

[in] the data type of the pData value. See data type constant in acodef.h.

DataLen

[in] the data length of pData.

pData

[in] the data of [index, sub-index] value.

pf

[in] Pointer to the application-defined function of type TCOnWriteSDOResult; represents the starting address of the function. The function will be executed if SDO request is responded or timeout occurs.

Return Values

If the function succeeds, the return value is ACOOD_SUCCESSFUL. If the function fails, please reference acoapi error codes and SDO abort codes.

Remarks

This asynchronous function is for the master node writes the value of object dictionary with object index and sub-index to other remote slave node. The application-defined function of type TCOnWriteSDOResult will be called if receiving the responded from the remote slave node as SDO server. Therefore, SDO parameters should be described in both master node and remote slave node at object dictionary of object 1280h to 12FFh SDO client parameter, and object 1200h to 127Fh SDO server parameter. Else there is not receiving any responded message from the remote slave node.

typedef void (*TCOnWriteSDOResult)(HACONODEOBJ hNode, NODEID_t NodeId, UNS16 wIndex, UNS8 SubIndex, UNS32 abortCode);

hNode argument [in] is the pointer to the remote slave node object handle with node identifier is *NodeID*. *NodeID* argument [in] is the remote slave node identifier. *wIndex* and *SubIndex* arguments [in] indicate which object to be write. *abortCode* argument is the return value, ACOOD_SUCCESSFUL means success, else get fails, reference acoapi error codes and SDO abort codes.

Reference acoReadSDO() and acoWriteSDO().

Example

```
extern HACONETWORK hNetworkObj;

void OnWriteSDOResult(
                HACONODEOBJ hNode, UNS8 NodeId, UNS16 wIndex, UNS8 bSubIndex, UNS32 abortCode)
{
        if (ACOOD_SUCCESSFUL == abortCode) {
                printf("WriteSDOResult: succ: RemoteNode[%02x] object [%04x:%02x] \n\r" \
                        , NodeId, wIndex, bSubIndex);
        } else
                printf ("WriteSDOResult failed: %X", abortCode);
}


UNS8 data=255;
//*** access Object Dictionary: async method
ret = acoNetworkWriteSDO(hNetworkObj, 0x02, 0x1800, 2,
                                        ACODT_UINT8, sizeof(data), &data, OnWriteSDOResult);
```

3. Advantech CANopen Examples

3.1. Overview

Advantech CANopen provides C#, VB.NET, VC.NET, VC and BCB examples for your reference.

The C#, VB.NET, VC.NET examples are developed by Microsoft Visual Studio 2005.

The VC examples are developed by VC6.0.

The BCB examples are developed by Borland C++ Builder 6.0

There are tow examples for each language ：acoMaster and acoSlave.

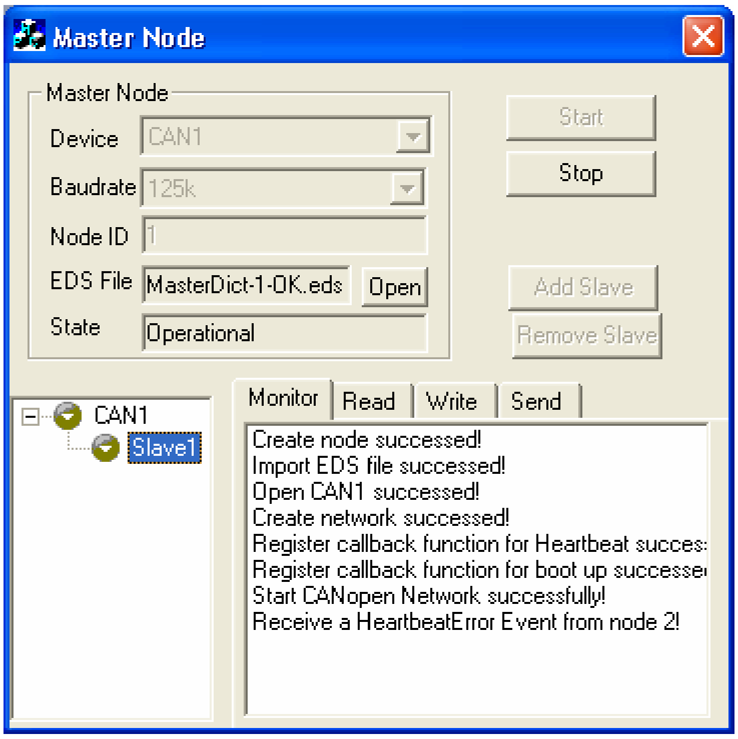
acoMaster example use MasterDict-1-ok.eds as its EDS file.

acoSlave example use SlaveDict-1-ok.eds as its EDS file.

3.2. Example usage

When user installs the advantech windows CANopen package, all examples will be installed in C:\Program Files\Advantech\Advantech CANopen by default.

Click the acoMaster to run the acoMaster example.



Master Node

Device: CAN1

Baudrate: 125k

Node ID: 1

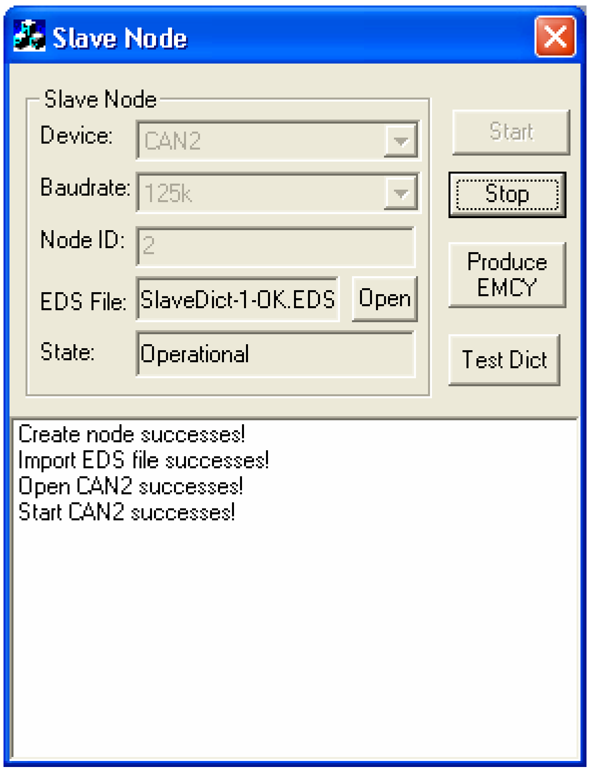
EDS File: MasterDict-1-OK.eds

State: Operational

```
Create node successed!
Import EDS file successed!
Open CAN1 successed!
Create network successed!
Register callback function for Heartbeat succes
Register callback function for boot up successe
Start CANopen Network successfully!
Receive a HeartbeatError Event from node 2!
```

Figure 5: Master Example Main Window

Click the acoSlave to run the acoSlave example.



Device: CAN2
Baudrate: 125k
Node ID: 2
EDS File: SlaveDict-1-OK.EDS
State: Operational

```
Create node successes!
Import EDS file successes!
Open CAN2 successes!
Start CAN2 successes!
```

Figure 6: Slave Example Main Window

4. Advantech CANopen Network Utility

4.1. Overview

The Advantech CANopen network utility is a utility to configure CANopen network node, to control and monitor the whole CANopen network. The main functions of Advantech CANopen Network Utility can be described as follows.

- An EDS file editor:configure network nodes, export EDS file that needed by CANopen library API when user develop application based on CANopen library.
- CANopen network manger: user can use the utility to create a CANopen network and do a simple test, it can also to control and monitor the CANopen network.

The following picture is the main window of this utility.

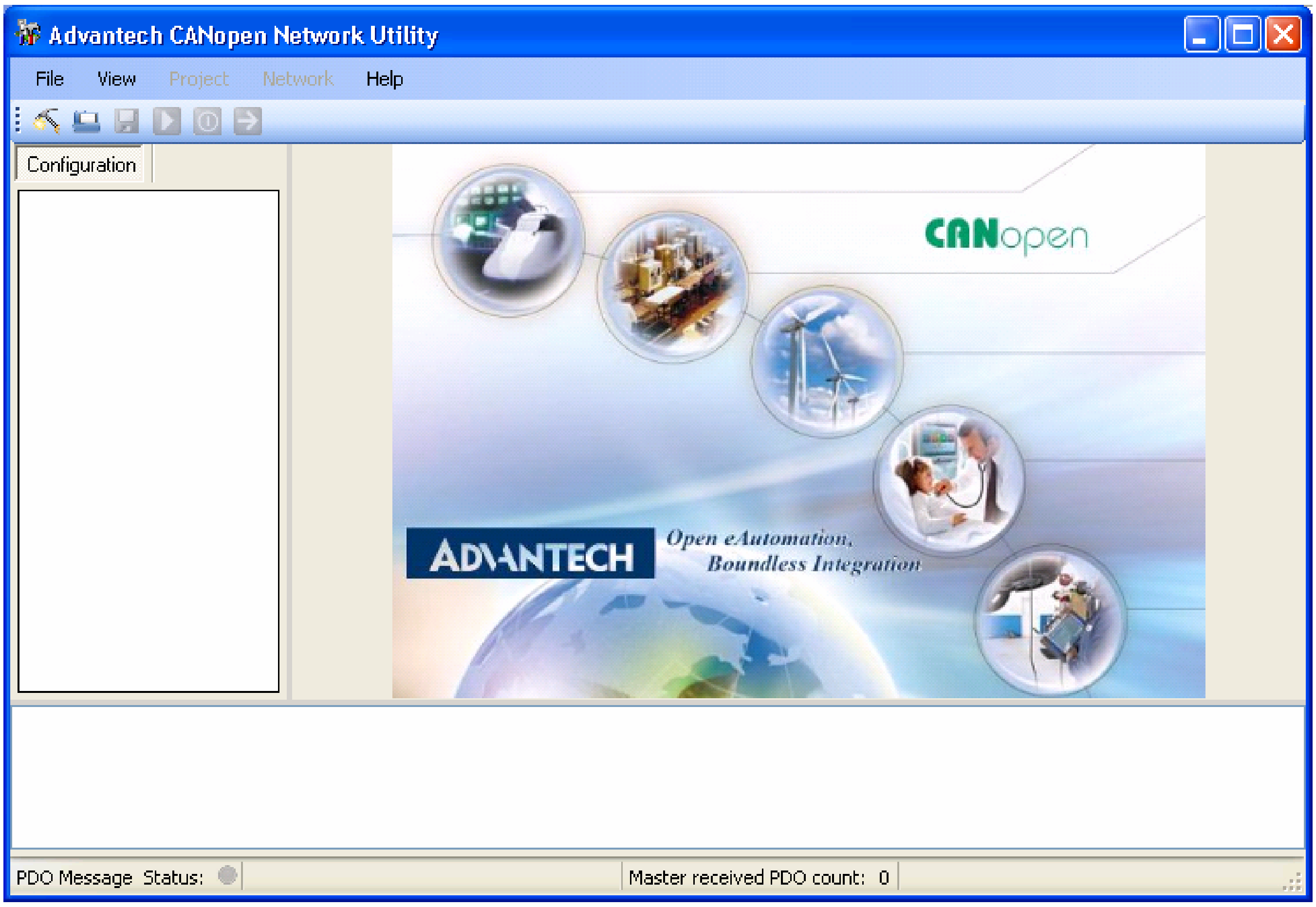
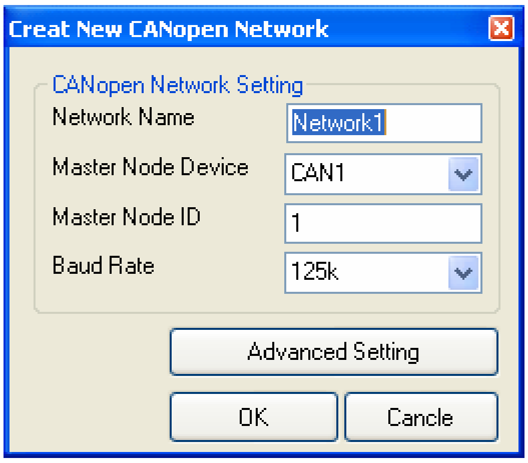

Advantech CANopen Network Utility
File
View
Project
Network
Help
Configuration
CANopen
ADVANTECH
Open eAutomation, Boundless Integration
PDO Message Status:
Master received PDO count: 0

Figure 7: Main frame of utility

4.2. Configure CANopen Network

4.2.1. Create CANopen Network

Firstly, Create the CANopen network, Click the "File->New CANopen NetWork" menu item, the utility will display a wizard to create network, user only need to input network name, master node device, master node id, network baud rate. These setting can also be changed after network is created.


Creat New CANopen Network

CANopen Network Setting

Network Name: Network1

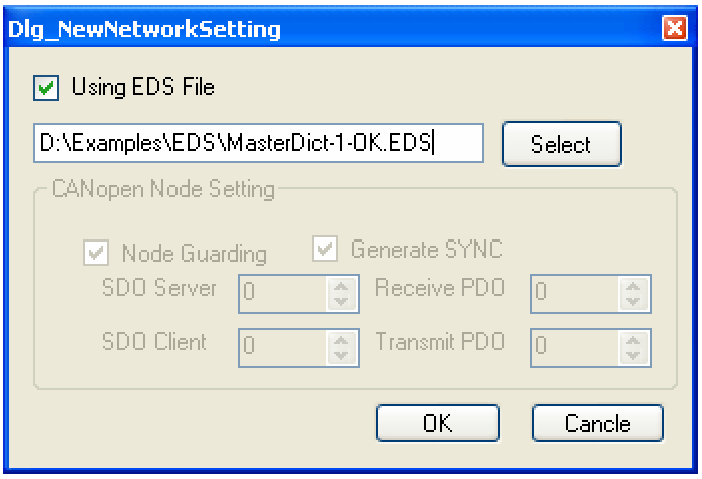
Master Node Device: CAN1

Master Node ID: 1

Baud Rate: 125k

Figure 8: Step 1 of Create Network Wizard

If user wants to set more about new network before create network, click the "Advanced Setting" button, advanced setting dialog support create network From EDS File, and user also can set the SDO, PDO number for master node, the setting dialog as follows.


Dlg_NewNetworkSetting

☑ Using EDS File

D:\Examples\EDS\MasterDict-1-OK.EDS

CANopen Node Setting

☑ Node Guarding ☑ Generate SYNC

SDO Server: 0 Receive PDO: 0

SDO Client: 0 Transmit PDO: 0

Figure 9: Step 2 of Create Network Wizard

Click "OK" button, a new CANopen network will be created according to the user's setting. The left view of the utility shows the structure of the new network as follows.

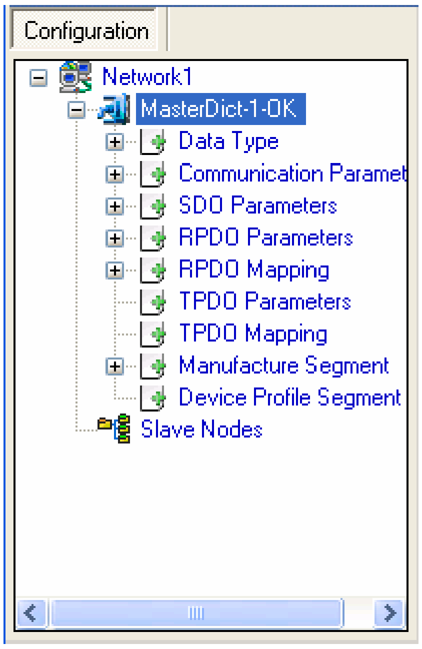


Figure 10: CANopen Network Tree View

When create CANopen network, only the master node is create in new network, user should to add slave node.

4.2.2.Create slave node

Select the "Slave Nodes" node, right click the mouse, a menu will display as follows.

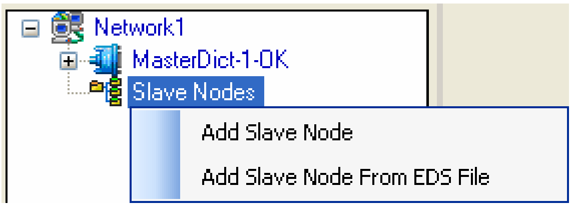
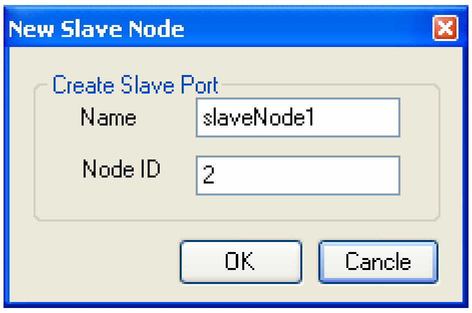


Figure 11: Add Slave Node Menu

There are two ways to create slave node, from EDS file or create empty slave node.

When clicking the "Add Slave Node", the following dialog will be displayed.


New Slave Node

Create Slave Port

Name: slaveNode1

Node ID: 2

Figure 12: Add Save Node Dialog

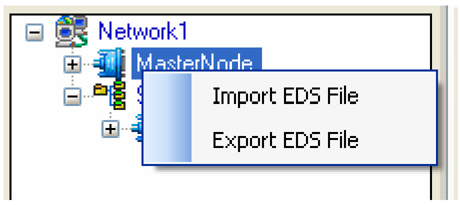
User input the name and node id, click "OK", and then the empty slave node will be created and add to network.

When clicking the "Add Slave Node from EDS File", user should give an EDS file path then the slave node will be created according to EDS file and add to network.

4.2.3.Configure Node

After create network and add slave node, user can configure them further, for example, create new data type, add communication parameters, PDO, SDO and so on, modify the setting, even import another EDS file for the master node or slave node. After finishing the configuration, the network configuration can be saved; each port configuration can export to EDS file.

4.2.4.Import EDS File



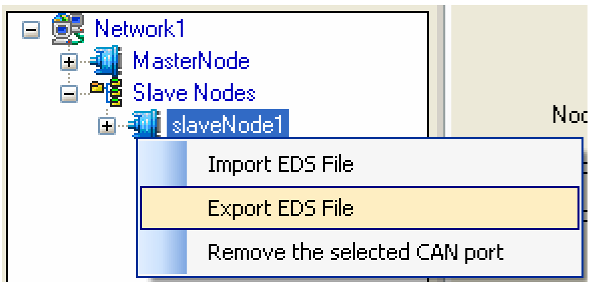


Figure 13: Import EDS File for Master Node and Slave Node

User can import EDS file to configure each CAN node. After import EDS file, the original configuration is replace by the new EDS file.

4.2.5.Create new data type

User can define new data type and use it when adding manufacture data, select the "Data Type" tree node of the CAN port, right click the mouse, and then click the "Create New Type" menu item.

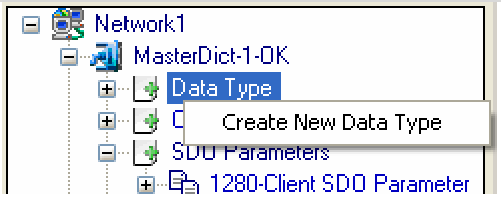


Figure 14: Create New Data Type Menu

User can input the new data type name and index, the index value must be between 0x40 and 0xFFF.For example, we create a new data type named “AI” in index 0x40.

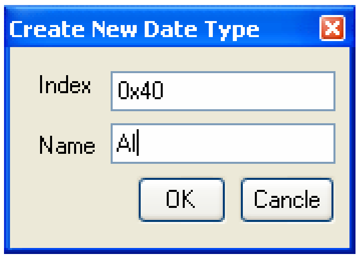


Figure 15: Create New Data Type

Click “OK” button，new data type is created.

Right click mouse and choose “Add New Variable” menu to add data member to the new data type. For example, add AI1, AI2, AI3, AI3 variables to AI.

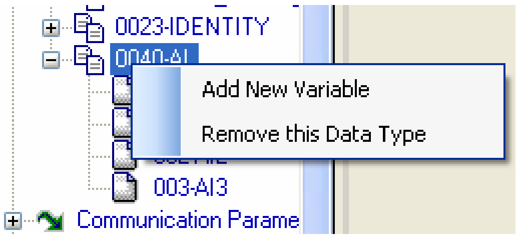


Figure 16: add data member to new data type

How to use the new data type will be introduced in later section.

4.2.6.Configure the communication parameters

Select the “Communication Parameters” tree node, right click the mouse and click “Add/Remove Communication” menu item.

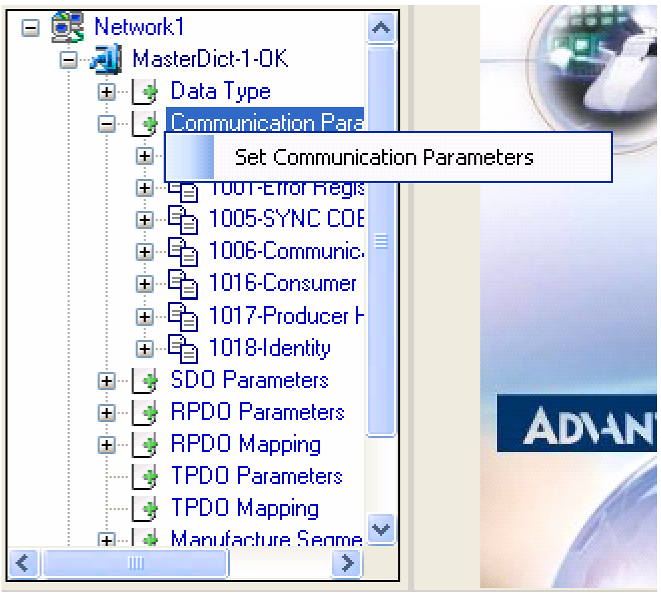
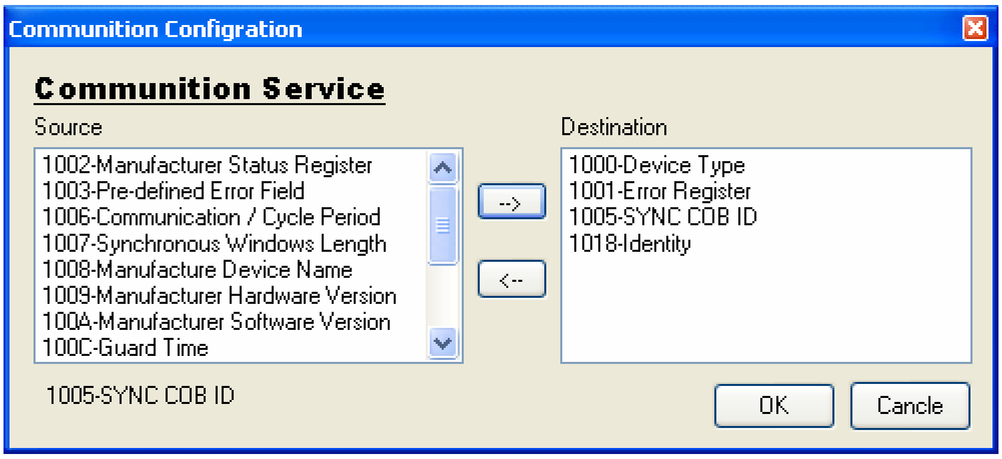


Figure 17: Add/Remove Communication Menu Item

Then a dialog displays and user can add or remove the communication parameters.



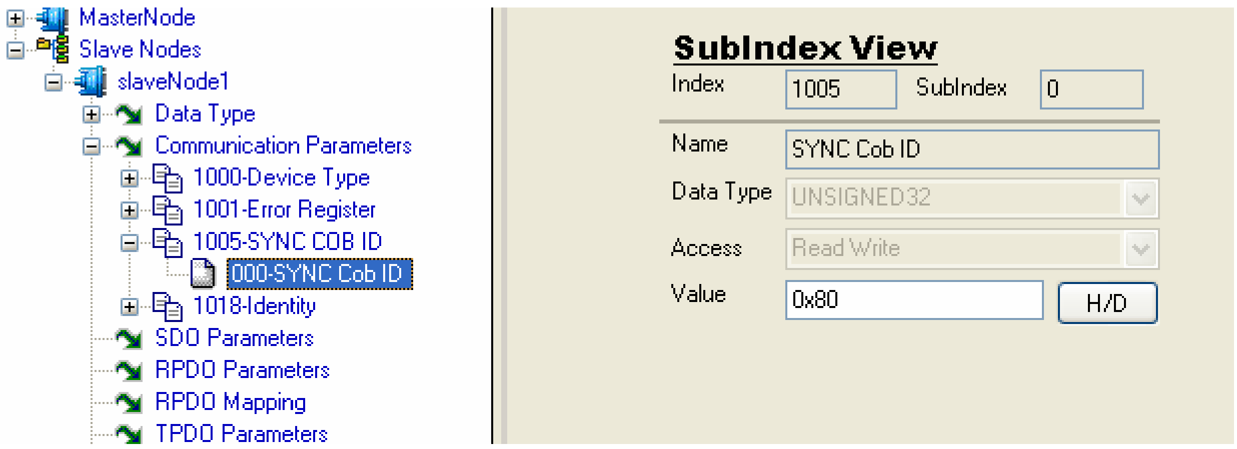
Communition Service

Source	Destination
1002-Manufacturer Status Register	1000-Device Type
1003-Pre-defined Error Field	1001-Error Register
1006-Communication / Cycle Period	1005-SYNC COB ID
1007-Synchronous Windows Length	1018-Identity
1008-Manufacture Device Name	
1009-Manufacturer Hardware Version	
100A-Manufacturer Software Version	
100C-Guard Time	

1005-SYNC COB ID

Figure 18: Add/Remove Communication Dialog

Click "OK", the select communication parameters will be added to CANopen network, user then can to configure them.



SubIndex View

Index 1005 SubIndex 0

Name: SYNC Cob ID

Data Type: UNSIGNED32

Access: Read Write

Value: 0x80

Figure 19: configure communication parameters

4.2.7.Create and configure SDO

User can add SDO server or SDO client by click “Add SDO Server” or “Add SDO Client”, and then user need to configure the new SDO Server and SDO Client.

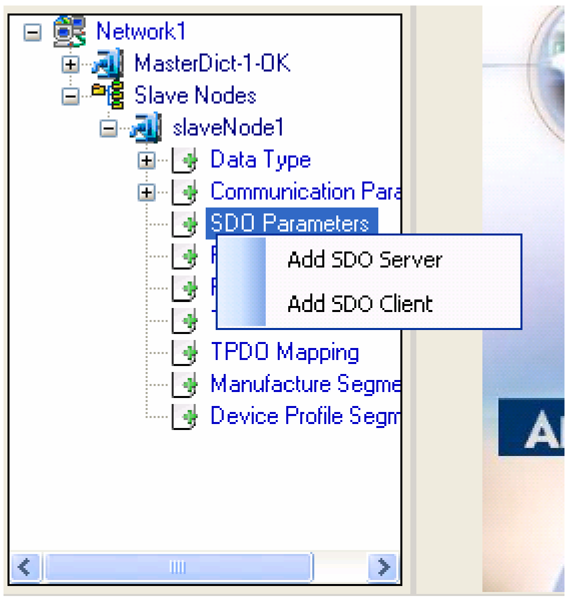
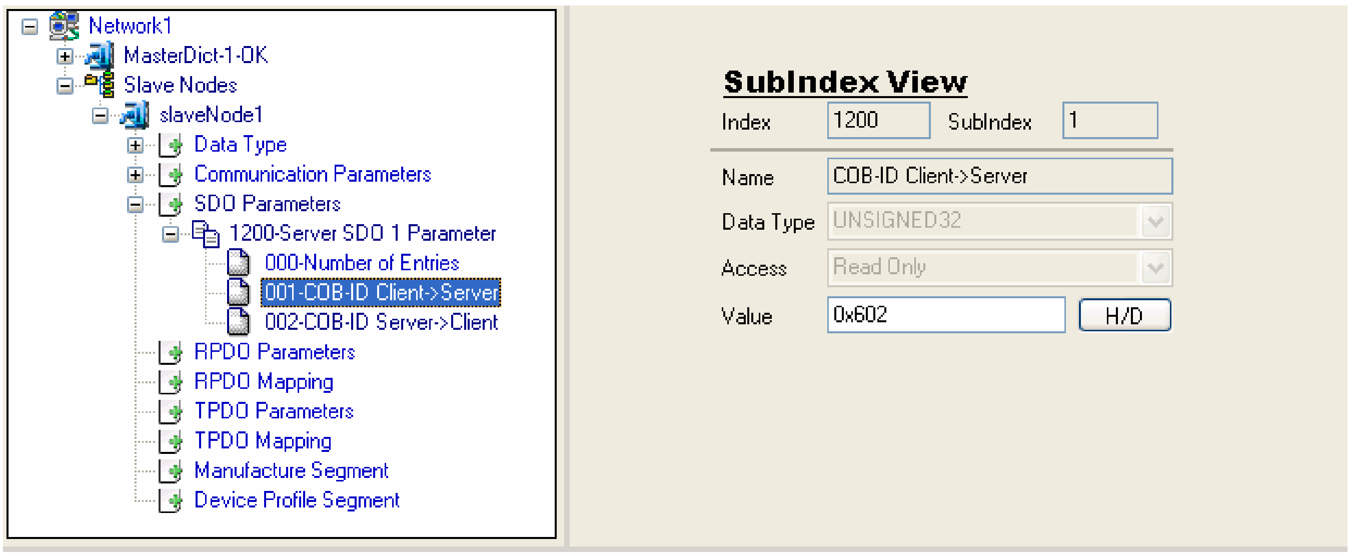


Figure 20: Add SDO Client and SDO Server Menu



SubIndex View

Index 1200 SubIndex 1

Name COB-ID Client->Server

Data Type UNSIGNED32

Access Read Only

Value 0x602

Figure 21: Configure SDO Parameter

4.2.8.Create and configure PDO

User can add transmit PDO or receive PDO by click “Add Receive PDO” or “Add Transmit PDO” menu item, and then user need to configure the new PDO to meet the user's requirement.

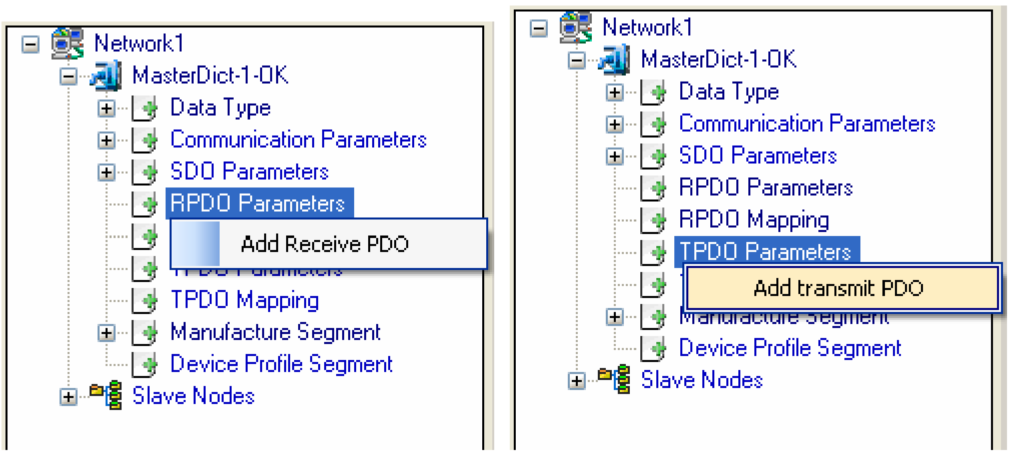
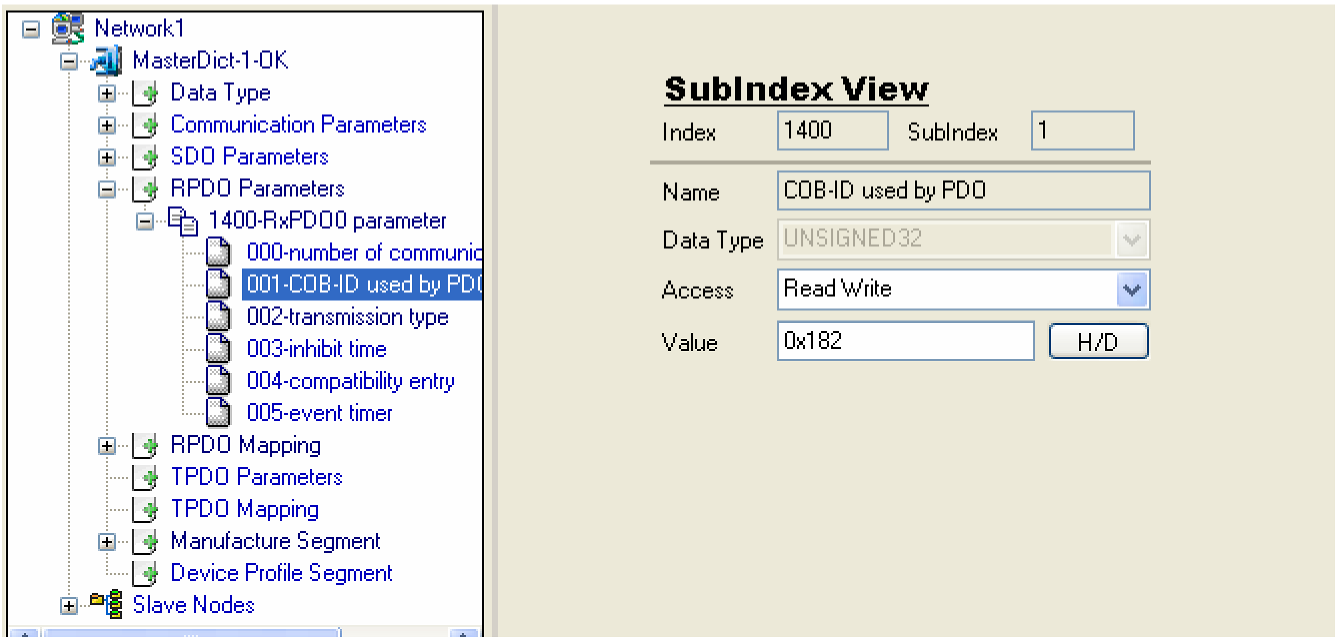


Figure 22: Add PDO Menu Item

The following picture is the PDO parameters configuration window.



SubIndex View

Index 1400 SubIndex 1

Name: COB-ID used by PDO

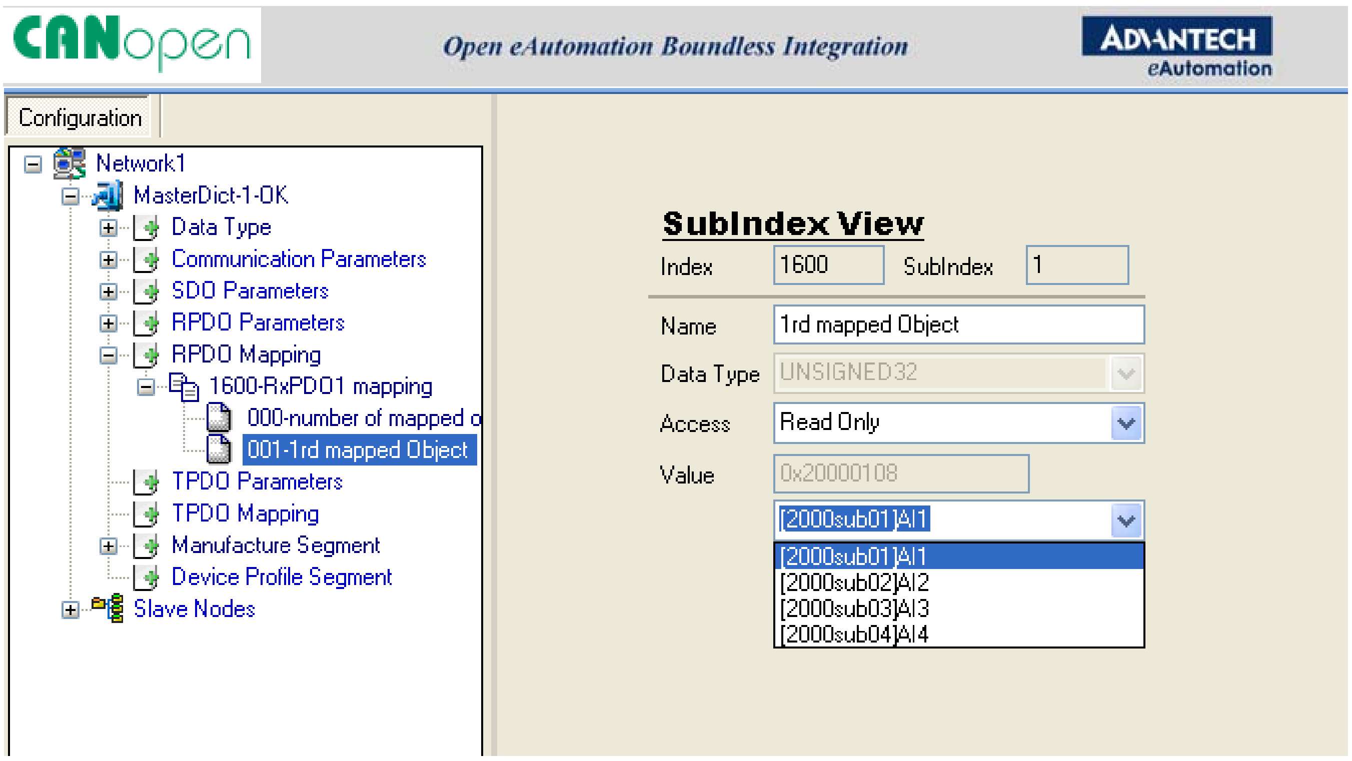
Data Type: UNSIGNED32

Access: Read Write

Value: 0x182

Figure 23: PDO Configuration

The following picture is the mapping data configuration window.



Index 1600 SubIndex 1

Name 1rd mapped Object

Data Type UNSIGNED32

Access Read Only

Value 0x20000108

[2000sub01]AI1

[2000sub01]AI1
[2000sub02]AI2
[2000sub03]AI3
[2000sub04]AI4

Figure 24: Add PDO Mapping Variable

User can remove PDO by click "Remove this PDO" menu item, and also can change PDO parameters number by click "PDO parameter Configuration".

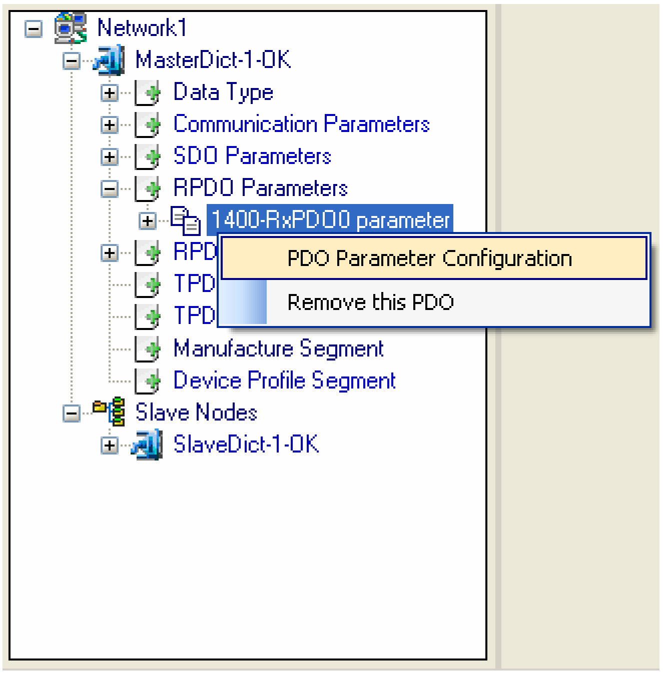
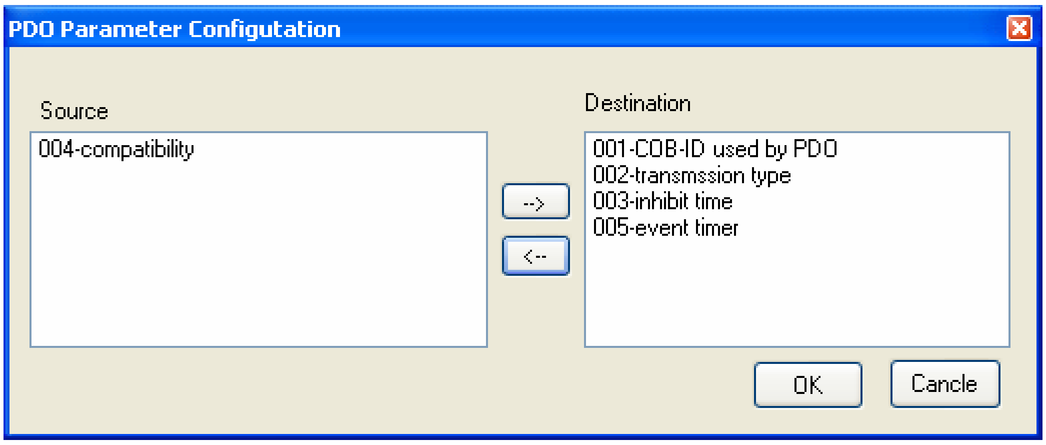


Figure 25: PDO Parameter Configuration Menu Item

The following picture is the dialog to add or remove the PDO parameters.


PDO Parameter Configutation

Source

004-compatibility

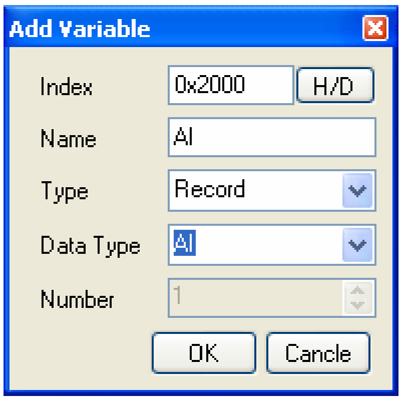
Destination

001-COB-ID used by PDO
002-transmssion type
003-inhibit time
005-event timer

Figure 26: PDO Parameter Configuration Dialog

4.2.9.Create manufacture variable

User can add manufacture data by "Add Mapping Variable" menu item. The following dialog will display.


Add Variable

Index	0x2000
Name	AI
Type	Record
Data Type	AI
Number	1

Figure 27: Add Manufacture Data variable

User can add variable, array and record data.

If add record, the new data type created by user can be used here.

Manufacture data's index must be between 0x2000 to 0x5FFF, they often will be used by PDO.

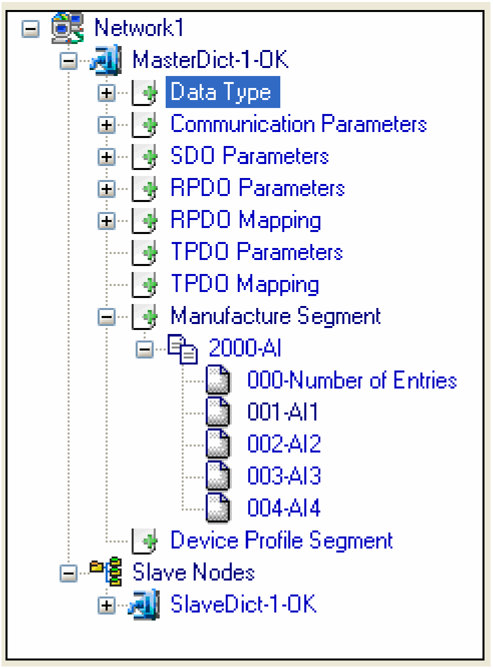


Figure 28: Add Manufacture Data Variable

4.2.10. Save the configuration

After finished configuration, user can save the whole network by "File->Save Project" or "File Save Project As" menu item. The saved project file can also be opened by utility for later use if need.
If user only needs to save one node setting, "Export EDS file" is the way. The EDS file is also need for CANopen application development based on acoapi library, so user can edit and export the EDS file in the utility for later use.
The export EDS file can be imported if needed.

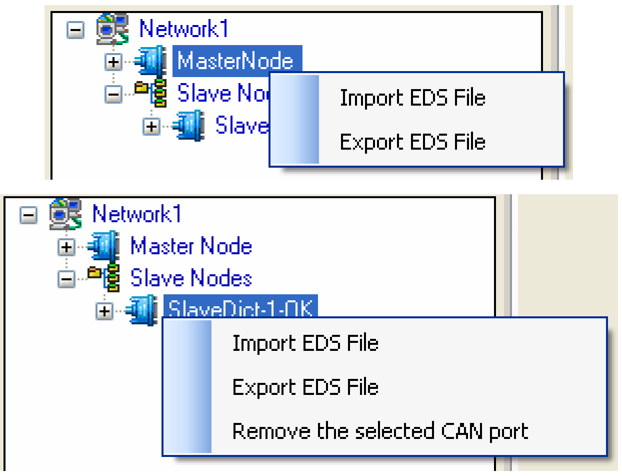


Figure 33: Import and Export EDS File

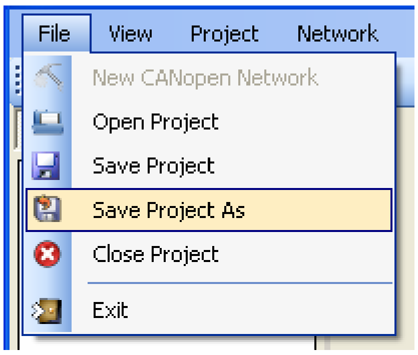


Figure 34: Save the Network Configuration

4.3. Manger CANopen network

4.3.1. Enter run mode

Utility can enter run mode to manger the CANopen network, it control and monitor the whole CANopen network by sending CANopen messages to slaves through Master node, that is to say, We can get slave node state, node id, PDO data value, read and write the local or remote object dictionary. Click "Run" menu item to enter the run mode and start the whole CANopen network after configuration.

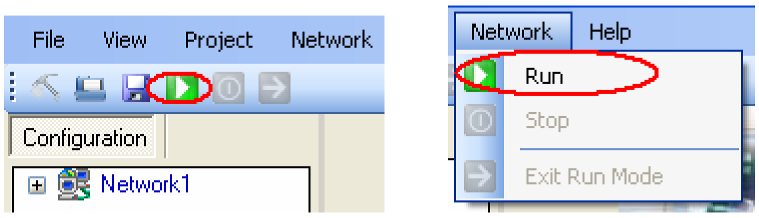
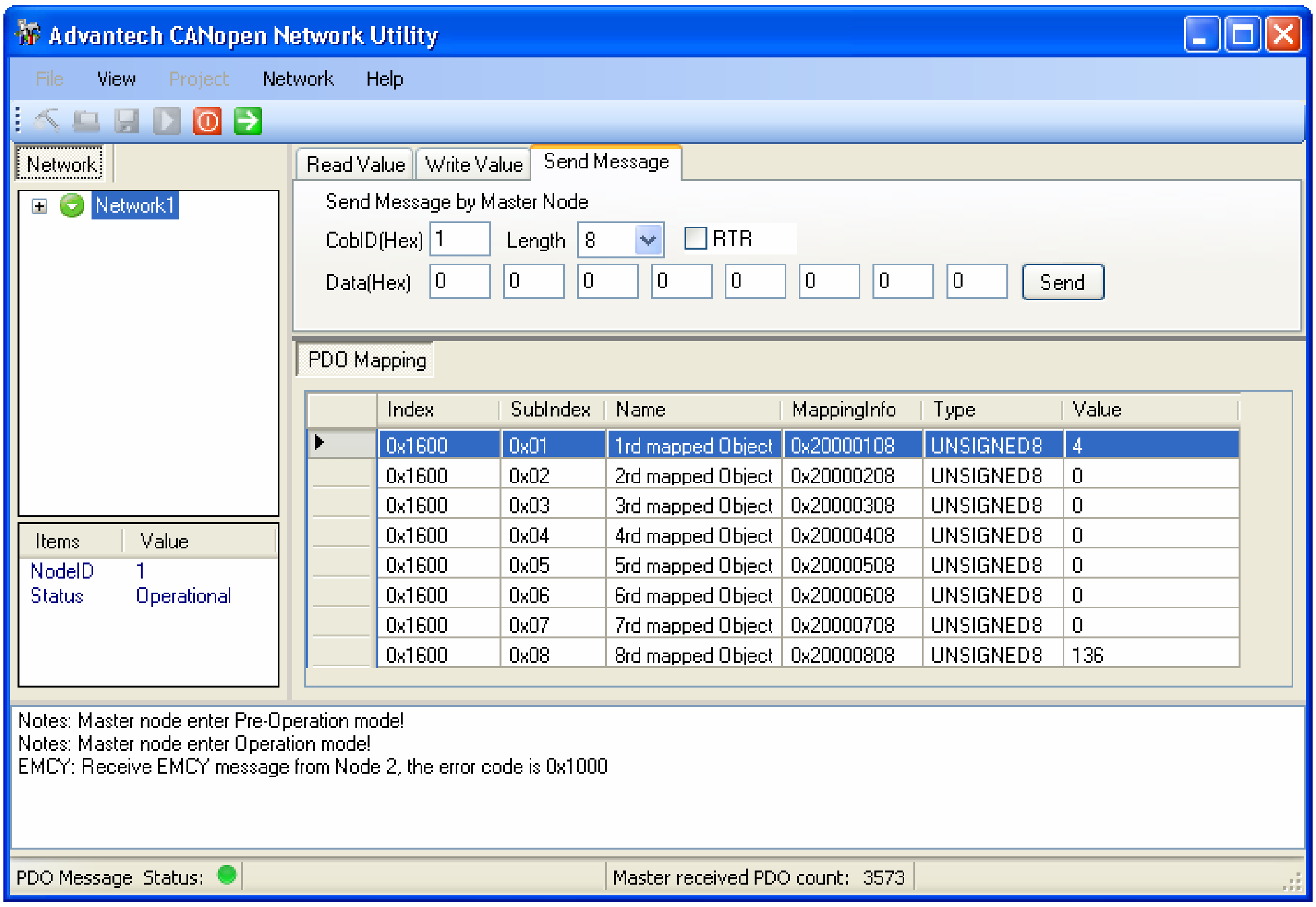


Figure 35: Enter Run Mode

The main window of running mode is as follows.


Advantech CANopen Network Utility
File View Project Network Help
Network
Network1
Read Value Write Value Send Message
PDO Mapping

Send Message by Master Node

CobID(Hex) 1 Length 8 RTR

Data(Hex) 0 0 0 0 0 0 0 0 Send

Items	Value
NodeID	1
Status	Operational

Index	SubIndex	Name	MappingInfo	Type	Value
0x1600	0x01	1rd mapped Object	0x20000108	UNSIGNED8	4
0x1600	0x02	2rd mapped Object	0x20000208	UNSIGNED8	0
0x1600	0x03	3rd mapped Object	0x20000308	UNSIGNED8	0
0x1600	0x04	4rd mapped Object	0x20000408	UNSIGNED8	0
0x1600	0x05	5rd mapped Object	0x20000508	UNSIGNED8	0
0x1600	0x06	6rd mapped Object	0x20000608	UNSIGNED8	0
0x1600	0x07	7rd mapped Object	0x20000708	UNSIGNED8	0
0x1600	0x08	8rd mapped Object	0x20000808	UNSIGNED8	136

```
Notes: Master node enter Pre-Operation mode!
Notes: Master node enter Operation mode!
EMCY: Receive EMCY message from Node 2, the error code is 0x1000
```

PDO Message Status: ● Master received PDO count: 3573

Figure 36: Run Mode Main Window

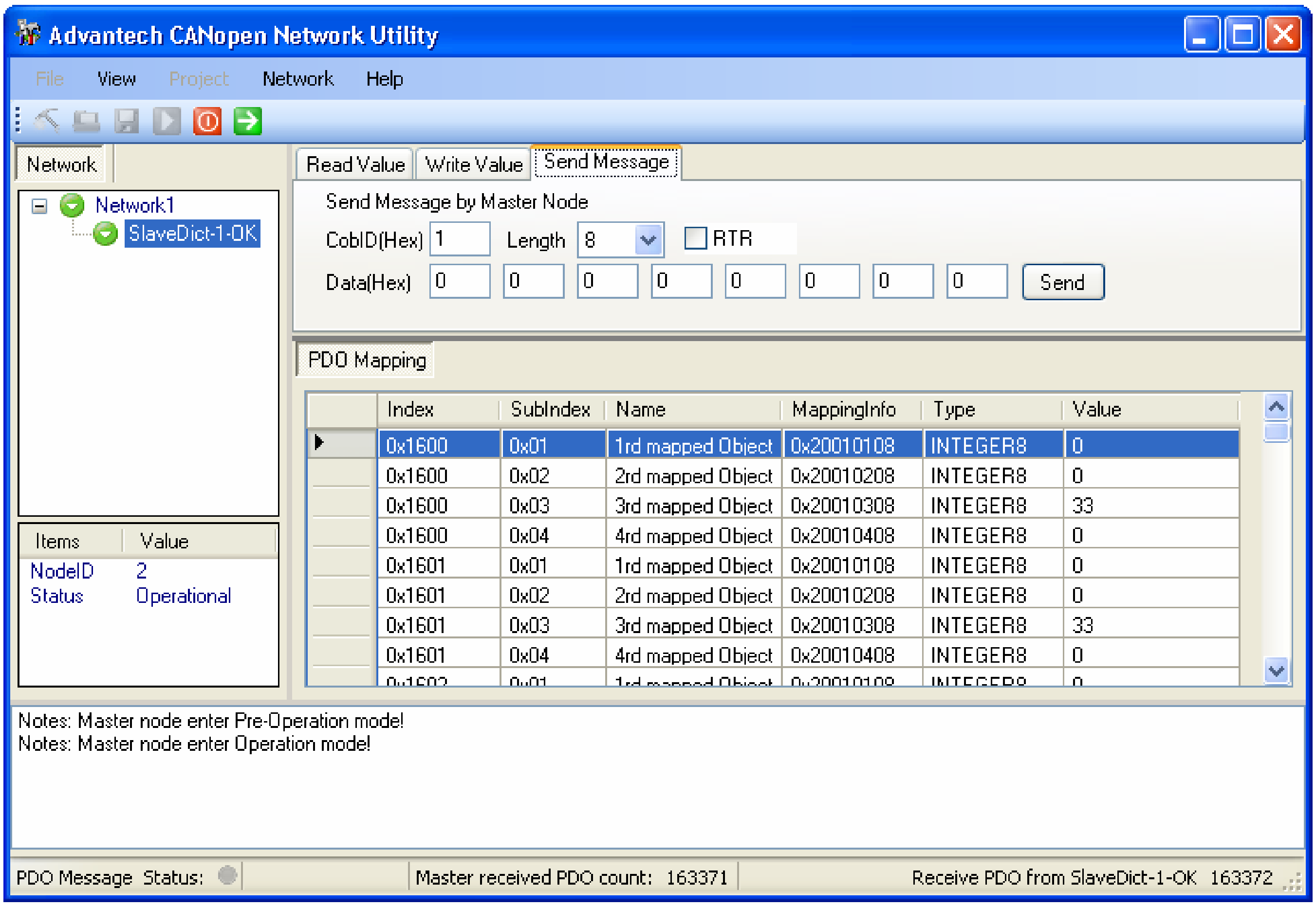
4.3.2.Monitor Node Data

Then user can select master node or slave node to monitor its data.

The following picture is the slave node monitor view. We can see its state and PDO data.

Please note that the utility can only control the network through master node, user should run slave node application to communicate with the utility if necessary, for example to run acoSlave example in the slave node, and then the utility can communication with slave node opened by acoSlave. Utility can also communicate with other manufacture CAN device base on CANopen protocol.

The following picture is slave node monitor view.


Advantech CANopen Network Utility
File View Project Network Help
Network
Network1
SlaveDict-1-OK
Read Value Write Value Send Message
Send Message by Master Node
RTR
Send
PDO Mapping

Items	Value
NodeID	2
Status	Operational

CobID(Hex) 1 Length 8
Data(Hex) 0 0 0 0 0 0 0 0

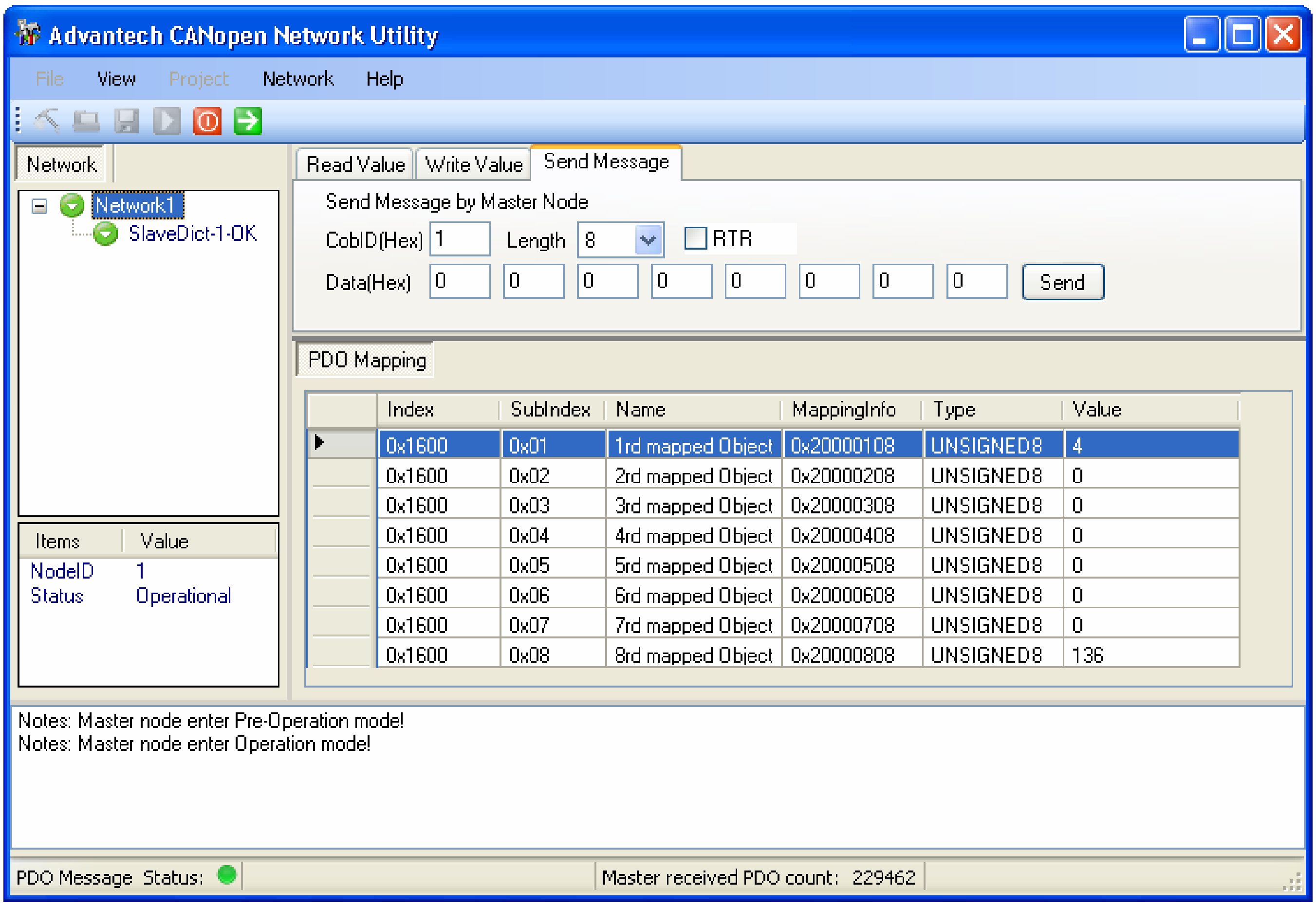
Index	SubIndex	Name	MappingInfo	Type	Value
0x1600	0x01	1rd mapped Object	0x20010108	INTEGER8	0
0x1600	0x02	2rd mapped Object	0x20010208	INTEGER8	0
0x1600	0x03	3rd mapped Object	0x20010308	INTEGER8	33
0x1600	0x04	4rd mapped Object	0x20010408	INTEGER8	0
0x1601	0x01	1rd mapped Object	0x20010108	INTEGER8	0
0x1601	0x02	2rd mapped Object	0x20010208	INTEGER8	0
0x1601	0x03	3rd mapped Object	0x20010308	INTEGER8	33
0x1601	0x04	4rd mapped Object	0x20010408	INTEGER8	0
0x1602	0x01	1rd mapped Object	0x20010108	INTEGER8	0

```
Notes: Master node enter Pre-Operation mode!
Notes: Master node enter Operation mode!
```

PDO Message Status: | Master received PDO count: 163371 | Receive PDO from SlaveDict-1-OK 163372

Figure 37: Run Mode Slave Node Window

The following picture is Master node monitor view.


Advantech CANopen Network Utility
File View Project Network Help
Network
Network1
SlaveDict-1-OK
Read Value Write Value Send Message
PDO Mapping
Send

Send Message by Master Node

CobID(Hex) 1 Length 8 RTR

Data(Hex) 0 0 0 0 0 0 0 0

Items	Value
NodeID	1
Status	Operational

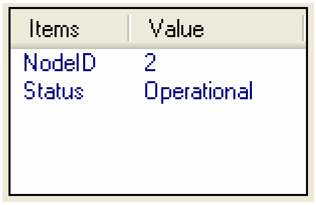
Index	SubIndex	Name	MappingInfo	Type	Value
0x1600	0x01	1rd mapped Object	0x20000108	UNSIGNED8	4
0x1600	0x02	2rd mapped Object	0x20000208	UNSIGNED8	0
0x1600	0x03	3rd mapped Object	0x20000308	UNSIGNED8	0
0x1600	0x04	4rd mapped Object	0x20000408	UNSIGNED8	0
0x1600	0x05	5rd mapped Object	0x20000508	UNSIGNED8	0
0x1600	0x06	6rd mapped Object	0x20000608	UNSIGNED8	0
0x1600	0x07	7rd mapped Object	0x20000708	UNSIGNED8	0
0x1600	0x08	8rd mapped Object	0x20000808	UNSIGNED8	136

```
Notes: Master node enter Pre-Operation mode!
Notes: Master node enter Operation mode!
```

PDO Message Status: ● Master received PDO count: 229462

Figure 38: Run Mode Master Node Window

From above Figure, we can see left-top view of the utility show the CANopen network topology structure, when use select one node, its node id and status will be shown as follows.



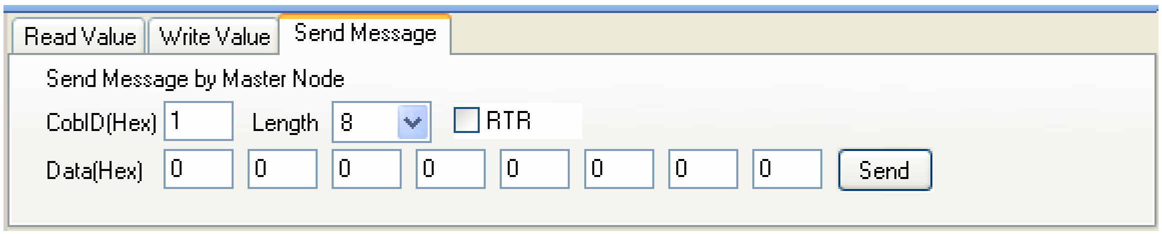
Items	Value
NodeID	2
Status	Operational

Figure 39: Node State View

The right view display the selected node PDO Mapping data value, if any data changed, this monitor view will update the data value in time.

4.3.3.Control Network

Master Node can send message to any node in the network by “send message” function tab. The entire node in the network (Master or slave) can read value from object dictionary by “Read Value” function in second tab and write value to node object dictionary by "Write Value" function in third tab page. The following picture is send message tab.



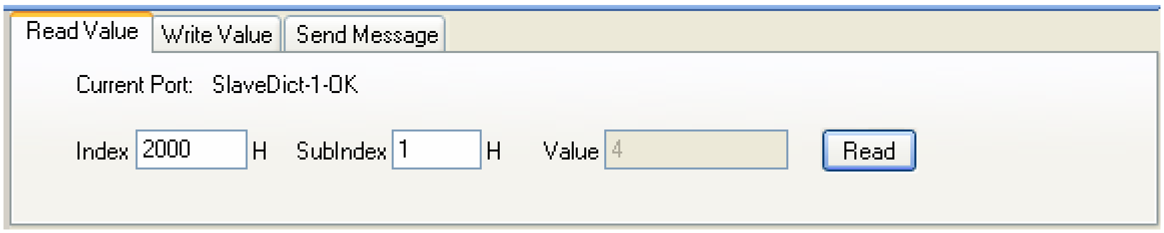
Send Message by Master Node

CobID(Hex) 1 Length 8 RTR

Data(Hex) 0 0 0 0 0 0 0 0

Figure 40: Send Message View

The following picture is read value tab. User can read the local or remote object dictionary as follows, only input the index and sub index then click the "Read" button.

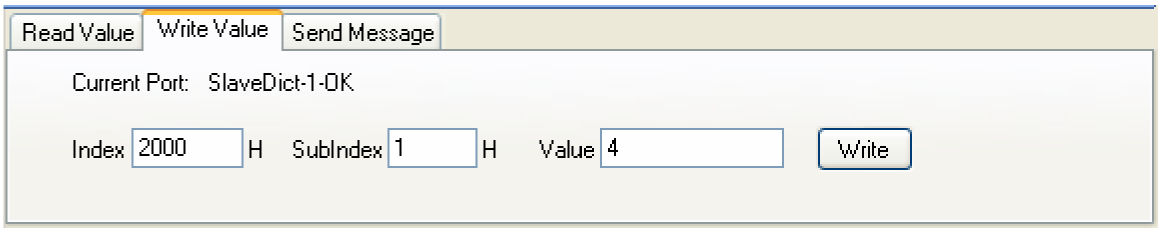


Current Port: SlaveDict-1-OK

Index 2000 H SubIndex 1 H Value 4

Figure 41: Read Message View

The following picture is write value tab. User can write the local or remote object dictionary as follows, only input the index and sub index then click the "Write" button.



Current Port: SlaveDict-1-OK

Index 2000 H SubIndex 1 H Value 4

Figure 42: Write Message View

User can use the menu to change the remote node state.

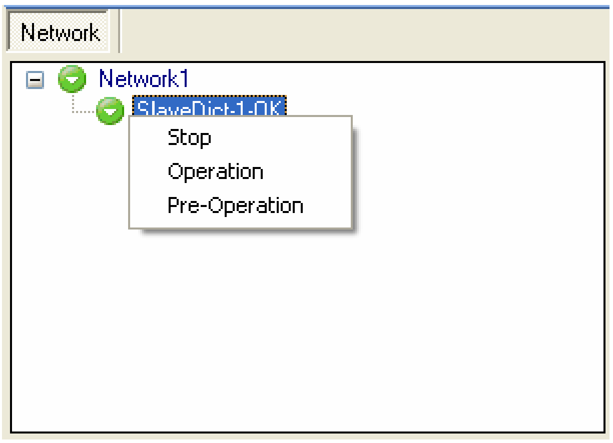


Figure 43: Change slave node state

4.3.4.Exit Run Mode

Click "Network->Exit Run Mode" to exit run mode, then it will return configuration mode, user can configure the network again.

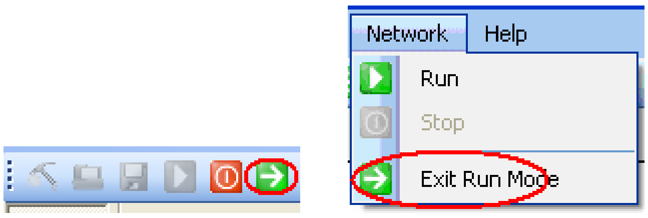


Figure 44: Exit Run Mode

APPENDIX

This appendix describes all the Objects and Data types supported by Advantech CANopen stack.

Data type support list

Data Type according to Object Dictionary Data Types of CANopen DS 301 v4.02 p.80.

Data Type	Value	Comment
BOOLEAN	0x01	Boolean，1-bit
INT8	0x02	a 8-bit integer
INT16	0x03	a 16-bit integer
INT32	0x04	a 32-bit integer
UINT8	0x05	a 8-bit unsigned integer
UINT16	0x06	a 16-bit unsigned integer
UINT32	0x07	a 32-bit unsigned integer
REAL32	0x08	a 32-bit real

COB-ID support list

COB-ID according to pre-defined connection of CANopen DS 301 v4.02 p.77.

Object	COB-IDs	Index
NMT	0	-
SYNC	128(80h)	1005h
EMERGENCY	129 (81h) – 255 (FFh)	1014h
TxPDO1	385 (181h) – 511 (1FFh)	1800h
RxPDO1	513 (201h) – 639 (27Fh)	1400h
TxPDO2	641 (281h) – 767 (2FFh)	1801h
RxPDO2	769 (301h) – 895 (37Fh)	1401h
TxPDO3	897 (381h) – 1023 (3FFh)	1802h
RxPDO3	1025 (401h) – 1151 (47Fh)	1402h
TxPDO4	1153 (481h) – 1279 (4FFh)	1803h
RxPDO4	1281 (501h) – 1407 (57Fh)	1403h
SDO Server	1409 (581h) – 1535 (5FFh)	1200h
SDO Client	1537 (601h) – 1663 (67Fh)	1200h
NMT Error Control	1793 (701h) – 1919 (77Fh)	-

Object Support list

Object according to Overview Object Dictionary Entries for Communication of CANopen DS 301 v4.02 p.84.

Object List	Default Value	Value Range
[1000] Device Type	0	ro
[1001] Error Register	0	ro
[100C] Guard time	0	ro
[100D] Life time factor	0	ro
[1018] Identity		
[1018sub0] number of entries	4	ro
[1018sub1] Vendor ID	0x251	ro
[1018sub2] Product Code	0xE2AE	ro
[1018sub3] Revision Number	0xA005	ro
[1018sub4] Serial Number	0	ro
[1003] Pre-defined Error Field		
[1003sub1] number of errors	0	rw, Read:0 –0xfe; Write:0
[1003sub1] Standard Error Field	NO	ro
[1005] SYNC COB ID	0x80	rw,0x80, 0x40000080
[1006] Communication / Cycle Period	0	rw, >1 millisecond
[1014] COB-ID EMCY	$NODEID+0x80	ro
[1016] Consumer Heartbeat Time		
[1016sub0] number of errors Time 1	NO	ro, 1 – 127
[1016sub1] To [1016sub7F]	0	rw,
[1017] Producer Heartbeat Time	0	rw
[1200] Server SDO Parameter		1 SDO Server
[1200sub0] Number of Entries	2	ro

[1200sub1] COBID Client to Server	0x600 + $NODEID	ro
[1200sub2] COBID Server to Client	0x580 + $NODEID	ro
[1280h - 12FFh] Client SDO Parameter		127 SDO Clients
[1280h - 12FFh sub0] number of entries	3	ro
[1280h - 12FFh sub１] COBID Client to Server	0x80000000	rw, 0x601-0x67F
[1280h - 12FFh sub２] COBID Server to Client	0x80000000	rw, 0x581-0x5FF
[1280h - 12FFh sub3] NODEID of the SDO Server	No	rw, 0x1-0x7F
[1400-15FF] RxPDO Parameter		512 RxPDO
[1400-15FF sub0] largest sub-index supported	5	ro
[1400-15FF sub1] COB-ID used by PDO	1400h: 200h + $Node-ID,	rw,1400h: 200h +$Node-ID, 80000200h+$Node-ID
	1401h: 300h + $Node-ID,	rw, 1401h: 300h + $Node-ID, 80000300h+$Node-ID
	1402h: 400h + $Node-ID,	rw, 1402h: 400h + $Node-ID, 80000400h+$Node-ID
	1403h: 500h +$ Node-ID	rw, 1403h: 500h + $Node-ID, 80000500h+$Node-ID
	……	……
[1400-15FF sub2] transmission type	NO	rw
[1400-15FF sub3] inhibit time	0	rw
[1400-15FF sub4] compatibility entry	0	rw
[1400-15FF03sub5] event timer	0	rw
[1600-17FF] RxPDO Mapping		512 RxPDO

[1600-17FF sub0] number of mapped application objects in PDO	NO	rw,1-40h
[1600-17FFsub1h－40h] PDO mapping for the nth application object to be mapped	NO	rw
[1800－19FF] TxPDO Parameter		512 TxPDO
[1800－19FF sub0] largest sub-index supported	5	ro
[1800－19FF sub1] COB-ID used by PDO	1800h: 180h + $Node-ID,	rw 1800h: 180h + $Node-ID, 80000180h+$Node-ID
	1801h: 280h + $Node-ID,	rw 1801h: 280h + $Node-ID, 80000280h+$Node-ID
	1802h: 380h + $Node-ID,	rw 1802h: 380h + $Node-ID, 80000380h+$Node-ID
	1803h: 480h + $Node-ID	rw 1803h: 480h + $Node-ID, 80000480h+$Node-ID
	……	……
[1800－19FF sub2] transmission type	NO	rw
[1800－19FF sub3] inhibit time	0	rw
[1800－19FF sub4] reserved	0	rw
[1800－19FF sub5] event timer	0	rw
[1A00-1BFF] transmit PDO mapping		512 TxPDO
[1A00-1BFF sub0]	NO	ro, 1-40h
[1A00-1BFF sub1h－40h]	NO	ro