CANopen Manager

Software Manual

esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
www.esd-electronics.com • Fax: 0511/37 29 8-68
Phone: 0511/37 29 80 • International: +49-5 11-37 29 80

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

USA / Canada:
esd electronics Inc.
525 Bernardston Road
Suite 1
Greenfield, MA 01301
USA

Phone: +1-800-732-8006
Fax: +1-800-732-8093
E-mail: us-sales@esd-electronics.com
Internet: www.esd-electronics.us

Manual File:	I:\texte\Doku\MANUALS\PROGRAM\CAN\CAL-COPN\CANOPEN\CANopen_Manager_21.en9
Date of Print:	2007-07-16

Described Software:	CANopen Manager
Revision:	2.1.x

Order no.:	CANopen manager: P.1405.80

Changes in the Software and/or Documentation

Chapter	Alterations in this manual versus previous version	Alterations in software	Alterations in documentation
all	With software revision 2.x the CANopen manager is completely rewritten to comply to CiA DS-302. The API is the same as for the CAL-/CANopen master for backward compatibility but the syntax of the initialization file has changed from a proprietary ASCII format to XML.	x	x
all	Document layout revised.	-	no changes in respect of content

Technical details are subject to change without notice.

This page is intentionally left blank.

Content

Page

1. Reference

/1/: CiA DS-301 V 4.02, CANopen-Application layer and communication profile, February 2002

/2/: CiA DSP-302 V3.3.0, Framework for CANopen Managers and Programmable Devices, October 2003

/3/: CiA DS-306 V1.2.0, Electronic Data Sheet Specification for CANopen, July 2004

2. Overview

2.1 Notes to this Manual

This manual describes the CANopen manager software package by esd. The CANopen manager is available for various hardware platforms and operating systems.
The first chapters of this manual describe the general functionality of the CANopen manager. The operating system specific features of the manager are listed in the appendix.
Knowledge of CANopen definitions and protocol descriptions is required.. References to CiA publications are accordingly cross-referenced (/1/.../3/).

2.2 Introduction

The CANopen protocol started as CAL based protocol using its communication and network management services and defined application and device specific objects which are stored in an object dictionary. Nowadays the CANopen protocol is developed further without any reference to CAL.

Each remote node in a CANopen network is assigned a module identifier (1-127) which has to be unique in the network. CANopen nodes are configured by the configuration manager part of the CANopen manager via access to the object dictionary based on Device Configuration Files (DCF). They are managed and monitored by the NMT master via CAN after the boot-up.

2.3 Features of the CANopen manager library

The CANopen manager library provides all features to configure, start and monitor CANopen slaves. The library can be used easily as stand-alone application to setup a CANopen network based on a manager configuration file in XML format and DCF files for the CANopen nodes but provides also a Application Programmer Interface (API) to control remote slaves by a custom application. All functionality specified by CANopen for a master to configure and manage remote nodes is covered by the library:

- Detection of CANopen nodes in the CAN network.
- Configuration of CANopen nodes with or without DCF files (Configuration manager).
- Management and monitoring of remote node states (NMT master).
- Monitoring of CANopen Emergency Objects.

Communication between the library and the application is realized with an event mechanism. In addition the libraries API gives the application the possibility to:

- Control the node state of remote slaves.
- Monitor the node state of remote slaves.
- Read and write object dictionary entries of remote slaves.

3. Initializing the CANopen Manager

3.1 Starting and Terminating the CANopen Manager

The CANopen manager operates in the background and if it is running faultlessly it does not require further intervention by the user after its start

A configuration file has to be created to describe the CANopen network which represents the project specific CANopen net.

3.2 Configuration of the CANopen Manager

The manager configuration file and the CANopen DCF files can be created with a standard text editor for ASCII files. The syntax of the CANopen DCF files is described in /3/. The manager configuration file follows the XML syntax. The keywords are described in following abstract.

3.2.1 Configuration file element relationship

Each XML configuration file for the CANopen manager has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<CANopen_Manager>
     <Manager_Instance>
          .
          .
          <Node_Instance>
               .
               .
          </Node_Instance>
          <Node_Instance>
               .
               .
          </Node_Instance>
     </Manager_Instance>
</CANopen_Manager>
```

`CANopen_Manager` is the root element of the configuration file. It contains child elements of `Manager_Instance` for every physical CAN network that should be controlled by this manager. Each of these `Manager_Instance` child elements contains child elements of `Node_Instance` for every instance of a remote slave that should be controlled in this network. Each of the elements mentioned above contain only element content. These elements are described in the following abstracts.

3.2.2 Element description

The description of all XML elements that are supported in the configuration file are made in a table with the following structure:

Keyword	Category	Data Type	Dictionary Entry
Value Range	Default	Obsolete syntax	
Description			

- **Keyword** is a XML element name.
- **Category** is either Mandatory or Optional.
- **Data Type** is one of the following:
  - UCHAR - 0..0xFF
  - USHORT - 0..0xFFFF
  - ULONG - 0..0xFFFFFFFF
  - BOOL - TRUE/FALSE
  - ASCII - ASCII string
  - ENUM - Element specific enumeration
- **Dictionary Entry** describes index, subindex and bit position for a keyword that is directly related to a DS-302 dictionary entry. If there is no relation this box is empty.
- **Value Range** is the valid range for this keyword. If a range isn't applicable this box is empty.
- **Default** is the value that is used if an optional keyword is missing.
- **Obsolete Syntax** is the keyword representation in the old style configuration syntax. If there is no such relation this box is empty. You can use this as a reference to port previous configuration files.
- **Description** is the purpose or effect of this keyword.

3.2.3 Elements of <Manager_Instance>

Net_No	Mandatory	UCHAR	
0 - 255		Net	
The element defines the logical net number the device driver has assigned to the CAN port that should be used by this instance of the manager.			

Baudrate	Mandatory	USHORT	
Valid CANopen baudrate		Baud	
The baudrate used for this CAN network in Kbit/s.			

Manager_Node_Id	Optional	UCHAR	
1 - 127	127		
According to DS-302 the manager itself is an active instance of a CANopen slave to offer configuration mechanisms via an object dictionary. For this purpose a Node-ID has to be assigned to the manager. This Node-ID must not conflict with the Node-ID of any remote CANopen slave in the network that is controlled by this manager.			

Config_File_Repository	Optional	ASCII	
OS dependent valid path		DCF_PATH	
Additional search path for the CANopen node's configuration files (DCF file or CCF file) to ease the logical organization of configuration files in the host file system. If opening a given configuration file in the working directory fails, this string is concatenated with the operating system specific path separator is prepended to the filename to open the configuration file from here.			

Heartbeat_Producer_Time	Optional	USHORT	0x1017
100 - 65535	0		
This parameter defines the cycle time of the manager heartbeat. The producer heartbeat time is 0 if not defined.			

Boot_Time	Optional	USHORT	0x1F89
100 - 65535	0		
Maximum time in ms the NMT master will wait for all mandatory slaves before signalling an error. If the time is 0 the master will wait endlessly. Attempt to boot optional slaves will always be repeated endlessly.			

NMT_Master	Optional	BOOL	0x1F80, Bit 0
TRUE, FALSE	TRUE		
If set to TRUE this is the NMT master. If set to FALSE this isn't the NMT master, the application is started in the slave only mode and no attempt is made to configure and start the CANopen network.			

NMT_Start_All	Optional	BOOL	0x1F80, Bit 1
TRUE, FALSE	FALSE	AUTOSTART	
This parameter defines the way the NMT master will start the remote slaves If set to TRUE the master will perform the service 'NMT Start Remote Nodes All Nodes'. If set to FALSE the master will start each remote node individually.			

NMT_Auto_Operational	Optional	BOOL	0x1F80, Bit 2
TRUE, FALSE	TRUE	AUTOSTART	
This parameter defines the behaviour of the NMT master after all mandatory slaves are booted successfully and all optional slaves are tried to be booted at least once. If set to TRUE the master will automatically enter the node state 'Operational' and continue to start the remote slaves. If set to FALSE the NMT master will wait to start the remote slaves until the application decides the master should enter the state 'Operational'.			

NMT_Auto_Start	Optional	BOOL	0x1F80, Bit 3
TRUE, FALSE	TRUE	AUTOSTART	
This parameter defines the behaviour of the NMT master after it has entered the state 'Operational'. If set to TRUE the master is allowed to start the remote slaves. If set to FALSE the application will start the remote slaves.			

NMT_Action_On_Error	Optional	ENUM	0x1F80, Bit 4/6
	RESET_SLAVE		
This parameter defines the way the NMT master handles error control events. If this parameter is set to RESET_SLAVE on Error Control Event of a mandatory the slave is treated individually. If this parameter is set to RESET_ALL on Error Control Event of a mandatory slave the NMT master performs a 'NMT Reset all Nodes' command (including self). If this parameter is set to STOP_ALL on Error Control Event of a mandatory slave the NMT master performs a 'Stop all Nodes' command (including self).			

Boot_Delay	Optional	USHORT	
0 - 65535	200		
This parameter defines the delay in ms the manager waits after the initial 'NMT Reset Communication 'before continuing with the 'Start Boot Slave Process'. You might adjust this parameter to be sure that all slaves finished their initialisation. See also the node specific parameter **Boot_Delay** in the next chapter.			

Wait_For_Startup	Optional	BOOL	
TRUE, FALSE	FALSE		
This parameter defines the behaviour of mcanOpenStart() that initiates the 'Start Boot Slave' process, which is performed in parallel for each managed slave node. If this parameter is set to FALSE the call will return immediately after the parallel processes are started. If this parameter is set to TRUE the call won't return before every process hasn't created at least the signal for the Boot Slave try.			

Preset_Emergency_Consumer	Optional	BOOL	
TRUE, FALSE	FALSE		
This parameter defines the behaviour of the CANopen manager es emergency consumer. If this parameter is set to FALSE only emergency messages of remote slaves that are defined in this configuration file are received. If this parameter is set to TRUE the manager is configured to receive emergency messages of all remote nodes according to the default connection set even if they are not defined in this configuration file.			

3.2.4 Elements of <Node_Instance>

Node_Id	Madatory	UCHAR	
1-127		NODEID	
Node-ID of this instance of the remote node that should be controlled by the master. The same Node-ID must not be assigned twice within one Manager_Instance. The Node-ID of the manager itself has to be unique, too.			

NMT_Slave	Optional	BOOL	0x1F81, Sub Node_Id, Bit 0
TRUE, FALSE	TRUE		
If set to TRUE the node with this Node-ID is a slave. After configuration (with Configuration Manager) the Node will be set to the state Operational by the NMT master. If set to FALSE the node with this Node-ID is not started by the NMT master			

NMT_Auto_Start_Slave	Optional	BOOL	0x1F81, Sub Node_Id, Bit 2
TRUE, FALSE	FALSE		
If set to FALSE on Error Control Event or detection of a booting slave the application is informed but the slave isn't automatically configure and started. If set to TRUE on Error Control Event or detection of a booting slave inform the application and do start the process "Start Boot Slave".			

NMT_Mandatory_Slave	Optional	BOOL	0x1F81, Sub Node_Id, Bit 3
TRUE, FALSE	FALSE		
If set to TRUE the node with this Node-ID is a mandatory slave and the network must not be started if this slave node could not be contacted during the boot slave procedure. If set to FALSE the node with this Node-ID is an optional slave network may be started even if this node could not be contacted			

Guard_Time	Optional	USHORT	0x1F81, Sub Node_Id, Byte 2-3
0-65535	0	GUARD	
This parameter determines the interval in ms a guard RTR is send. If the answer is missing (Retry_Factor-1) the master will create a guard error event. Guarding will be performed only if Guard_Time and Retry_Factor are both different from 0. The guard time in ms configured for the NMT master should be smaller than the guard time configured for the remote slave by the configuration manager to prevent frequent timeout errors.			

Life_Time_Factor	Optional	UCHAR	0x1F81, Sub Node_Id, Byte 1
0-255	0	**LIFE**	
This parameter determines the factor the guard RTR is resend before the NMT master will create a guard error event. Guarding will be performed only if Guard_Time and Retry_Factor are both different from 0.			

NMT_Check_Nodestate	Optional	BOOL	0x1F81, Sub Node_Id, Bit 4
TRUE, FALSE	FALSE		
If set to FALSE the slave node may be reset with 'NMT Reset Communication' command independent of its state. Hence, no checking of its state has to be executed prior to 'NMT Reset Communication' command. If set to TRUE the NMT Master must not send 'NMT Reset Communication' for this node if it notices the slave to be in 'Operational' state. This is noticed by waiting for the heartbeat message or sending RTR for the node guard message.			

NMT_Verify_Software	Optional	BOOL	0x1F81, Sub Node_Id, Bit 5
TRUE, FALSE	FALSE		
If set to FALSE the application software version verification for this node is not required. If set to TRUE application software version verification for this node is required.			

NMT_Allow_Software_Update	Optional	BOOL	0x1F81, Sub Node_Id, Bit 6
TRUE, FALSE	FALSE		
If set to FALSE the automatic application software update (download) is not allowed. If set to TRUE the automatic application software update (download) is allowed.			

Consumer_Heartbeat_Time	Optional	USHORT	0x1016, Sub Node_Id, Byte 0-1
0..65535	0		
Defines the consumer heartbeat time in ms. If within this interval no heartbeat message is received from the remote slave the NMT master will create a error monitoring event. The consumer heartbeat time in ms configured for the NMT master should be configured to a higher value than the producer heartbeat time configured for the remote slave by the configuration manager to prevent frequent error monitoring events. If heartbeat monitoring is configured for this remote node all configuration for guarding is ignored.			

Device_Type_Identification	Optional	ULONG	0x1F84, Sub Node_Id
0x0..0xFFFFFFFF	0		
This element allows to enter values for expected device types. On Boot-Up the Master reads object 0x1000 of each assigned slave. If this value is 0, this read access only gives information about the principle existence of a device with this Node-ID. If the value is not 0, it is compared against the value read from the device and the boot-up for that device is only continued on exact equality.			

Vendor_Identification	Optional	ULONG	0x1F85, Sub Node_Id
0x0..0xFFFFFFFF	0		
This element allows to enter values for expected vendor id. If Vendor_Identification is not 0 the master will read object 0x1018, subindex 1 of the remote node and will compare this value against the value read from the device. The boot-up for that device is only continued on exact equality.			

Product_Code	Optional	ULONG	0x1F86, Sub Node_Id
0x0..0xFFFFFFFF	0		
This element allows to enter values for expected vendor id. If Product_Code is not 0 the master will read object 0x1018, subindex 2 of the remote node and will compare this value against the value read from the device. The boot-up for that device is only continued on exact equality.			

Revision_Number	Optional	ULONG	0x1F87, Sub Node_Id
0x0..0xFFFFFFFF	0		
This element allows to enter values for expected vendor id. If Revision_Number is not 0 the master will read object 0x1018, subindex 3 of the remote node and will compare this value against the value read from the device. The boot-up for that device is only continued on exact equality.			

Serial_Number	Optional	ULONG	0x1F88, Sub Node_Id
0x0..0xFFFFFFFF	0		
This element allows to enter values for expected vendor id. If Serial_Number is not 0 the master will read object 0x1018, subindex 4 of the remote node and will compare this value against the value read from the device. The boot-up for that device is only continued on exact equality.			

DCF_File	Optional	ASCII	
		DCF	
DCF file according to DS-306 that is used by the Configuration Manager to configure the remote node. This file is translated internally into the 'Concise Configuration Format' (CCF) defined in DS-302. For one Node_Instance either a DCF file or a CCF file can be defined. If no DCF file or CCF file is assigned to the node the Configuration Manager can't configure the remote slave. Error control is only possible if guarding is configured and the remote slave supports passive guarding or if the remote slave has a pre-configured heartbeat time that matches the consumer heartbeat time configured for the NMT master.			

CCF_File	Optional	ASCII	
CCF file according to DS-302 that is used by the Configuration Manager to configure the remote node. A CCF file is already in a (binary) format that describes the data stream the Configuration Manager sends to the remote node for configuration. CCF files have to be derived from DCF files in an offline configuration step. For one Node_Instance either a DCF file or a CCF file can be defined. If no DCF file or CCF file is assigned to the node the Configuration Manager can't configure the remote slave. Error control is only possible if guarding is configured and the remote slave supports passive guarding or if the remote slave has a pre-configured heartbeat time that matches the consumer heartbeat time configured for the NMT master.			

Old_Config_Style	Optional	BOOL	
TRUE, FALSE	FALSE		
In DS-301 Rev. 4.x the configuration process for PDOs was described in detail the first time. To configure the PDO mapping the Configuration Manager has to set the number of mapping entries to 0 before changing the PDO mapping. Some older CANopen devices may have problems with this behaviour as they expect the number of mapping entries is configured to the correct value before PDO Mapping entries are changed. If this parameter is set to FALSE the number of mapping entries is set to 0 before the mapping entries itself are changed. The number of mapping entries is set to the correct value afterwards to validate the configuration. If this parameter is set to TRUE the correct value is written before the mapping entries itself are changed. This entry only has an effect to DCF files. As CCF files are already in a format that describes the data stream the Configuration Manager sends, changing configuration style has to be indicated during offline conversion from DCF files into CCF files.			

Boot_Delay	Optional	USHORT	
0 - 65535	200		
This parameter defines the delay in ms the manager waits after a 'NMT Reset Communication' or 'NMT Reset Application' for this node before continuing with the 'Start Boot Slave Process'. It is applied if the manager tries to restart the node as result of an error monitoring event, if the application forces the reset or the node is configured with the 'Keeping alive' option (see parameter **NMT_Check_Nodestate**). In comparison to the network specific parameter **Boot_Delay** this time is part of the **Boot_Time** configured for this network.			

SDO_Timeout	Optional	USHORT	
0 - 65535	200	SDO_RX_TIMEOUT	
If the object dictionary of the remote slave is read or written by the CANopen manager during bootup and configuration the operation fails after a timeout without an reply of the slave. This parameter defines the timeout value in ms.			

SDO_Segmented_Size_Indication	Optional	BOOL	
TRUE, FALSE	TRUE		
According to DS-301 a segmented SDO download request may indicate the data set size or not in the SDO Download Initiate. If this parameter is set to TRUE, the download size is indicated to the remote CANopen node, which is the sensible default for most CANopen devices on the market. If this parameter is set to FALSE the size isn't indicated to the remote CANopen node during SDO Download Initiate, instead the last downloaded segment is marked to indicate the end of transmission.			

3.2.5 Example configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<CANopen_Manager>
	<Manager_Instance>
		<Net_No>0</Net_No>
		<Baudrate>500</Baudrate>
		<Manager_Node_Id>127</Manager_Node_Id>
		<Config_File_Path>./dcf</Config_File_Path>
		<Boot_Time>20000</Boot_Time>
		<Heartbeat_Producer_Time>5000</Heartbeat_Producer_Time>
		<NMT_Master>TRUE</NMT_Master>
		<NMT_Start_All>TRUE</NMT_Start_All>
		<NMT_Auto_Operational>TRUE</NMT_Auto_Operational>
		<NMT_Action_On_Error>STOP_ALL</NMT_Action_On_Error>
		<Node_Instance>
			<Node_Id>1</Node_Id>
			<NMT_Slave>TRUE</NMT_Slave>
			<NMT_Mandatory_Slave>FALSE</NMT_Mandatory_Slave>
			<Guard_Time>0</Guard_Time>
			<Life_Time_Factor>0</Life_Time_Factor>
			<Consumer_Heartbeat_Time>3000</Consumer_Heartbeat_Time>
			<Device_Type_Identification>0x00000191</Device_Type_Identification>
			<Vendor_Identification>0x17</Vendor_Identification>
			<NMT_Check_Nodestate>FALSE</NMT_Check_Nodestate>
			<DCF_File>asyncio.dcf</DCF_File>
		</Node_Instance>
		<Node_Instance>
			<Node_Id>2</Node_Id>
			<NMT_Slave>TRUE</NMT_Slave>
			<NMT_Mandatory_Slave>FALSE</NMT_Mandatory_Slave>
			<Guard_Time>0</Guard_Time>
			<Life_Time_Factor>0</Life_Time_Factor>
			<Consumer_Heartbeat_Time>3000</Consumer_Heartbeat_Time>
			<Device_Type_Identification>0x00000191</Device_Type_Identification>
			<Vendor_Identification>0x17</Vendor_Identification>
			<NMT_Check_Nodestate>FALSE</NMT_Check_Nodestate>
			<CCF_File>asyncio.ccf</CCF_File>
		</Node_Instance>
		<Node_Instance>
			<Node_Id>3</Node_Id>
			<NMT_Slave>TRUE</NMT_Slave>
			<NMT_Mandatory_Slave>FALSE</NMT_Mandatory_Slave>
			<Guard_Time>2000</Guard_Time>
			<Life_Time_Factor>2</Life_Time_Factor>
			<Consumer_Heartbeat_Time>0</Consumer_Heartbeat_Time>
			<Device_Type_Identification>0x00000191</Device_Type_Identification>
			<Vendor_Identification>0x17</Vendor_Identification>
			<NMT_Check_Nodestate>FALSE</NMT_Check_Nodestate>
			<DCF_File>asyncio2.dcf</DCF_File>
		</Node_Instance>
	</Manager_Instance>
</CANopen_Manager>
```

4. Manager API

4.1 Overview

The API consists of three groups of functions:

- Basic functions that have to be called to setup the manager.
- Optional functions that can be called by the application to get more detailed information about the managed nodes..
- Functions to access the remote nodes object dictionary or change their node state.

4.2 Basic manager initialization functions

The following procedure is necessary to setup and start the manager.

1. Initializing the manager library with *mcanOpenInitialize()*.
2. Reading the configuration by calling *mcanOpenConfig()*.
3. Starting the manager by calling *mcanOpenStart()*.

The following procedure is necessary to stop the manager.

1. Terminating the master by calling *mcanOpenShutDown()*.
2. Cleanup the library with *mcanOpenCleanup()*.

mcanOpenInitialize()

Name: **mcanOpenInitialize()** - Initialize the manager library

Call: **int mcanOpenInitialize(void);**

Description: This function initializes the manager library. The function has to be called once before any other API call.

Return: Success or error code described in the appendix.

---

mcanOpenCleanup()

Name: **mcanOpenCleanup()** - Cleanup the manager library

Call: **int mcanOpenCleanup(void);**

Description: This function returns all resources that are allocated by the manager library to the operating system. The function has to be called once at the end of the library usage.

Return: Success or error code described in the appendix.

mcanOpenConfig()

Name: **mcanOpenConfig()** - Configuring the manager database.

Call:
```
int    mcanOpenConfig
       (
       char     *cfgFile
       );
```

Description: This function reads the manager configuration file *cfgFile* and all associated DCF or CCF files for the project specific CANopen network setup. The CAN controller will be initialized and all necessary resources will be allocated.

Return: Success or error code described in the appendix.

---

mcanOpenStart()

Name: **mcanOpenStart()** - Start the manager.

Call:
```
void  mcanOpenStart(void);
```

Description: The function starts the manager daemon tasks which setup and start all nodes in the CANopen network according to the configuration file given with *mcanOpenConfig()*.

Return: N/A.

---

mcanOpenShutDown()

Name: **mcanOpenShutDown()** - Terminate the manager.

Call:
```
int    mcanOpenShutDown(void);
```

Description: This function terminates the CANopen manager and frees all dynamically allocated resources.

Return: Success or error code described in the appendix.

mcanOpenGetVersion()

Name: **mcanOpenGetVersion()** - Return the software-version number

Call:

```
int    mcanOpenGetVersion
       (
       unsigned short          *version
       );
```

Description: In parameter *version the current software-version number is returned.

Value range:

Bit 12...15 : level
Bit 8..11 : revision
Bit 0..7 : change

Return: Success or error code described in the appendix.

4.3 Optional node control functions

The following functions are optional node control functions that can be called by an application to monitor the state of the remote nodes.

---

mcanOpenEvent()

Name: **mcanOpenEvent()** - Receive events from the manager

Call:

```
int     mcanOpenEvent
        (
        EVENT_T     *event,
        int         wait
        );
```

Description: This functions has to be called by the application to receive errors, warnings and indications from the manager daemons. The application can either block until the manager indicates an event to the application by setting parameter *wait* to 1 which is typically done with a separate task or can poll for events by setting parameter *wait* to 0. In the structure `EVENT_T` which is initialized by the manager the event type and additional parameter is stored:

```
typedef struct {
                    long event;
                        int net;
                        int node;
                        union {
                                char                c;
                                unsigned char       uc;
                                short               s;
                                unsigned short      us;
                                int                 i;
                                unsigned int        ui;
                                long                l;
                                unsigned long       ul;
                                struct {
                                            unsigned short  index;
                                            unsigned short  subindex;
                                            ERROR_T         err;
                                } write;
                                char            str[64];
                    } para;
                } EVENT_T;
```

This structure has to be interpreted in the following way:

The MSB of EVENT_T.event is set, if EVENT_T.net is invalid, the next is then accordingly set, if EVENT_T.node is invalid.

The following 22 bits mark the valid parameter of the Union EVENT_T.para. The least significant byte contains the message number of the message field corresponding to the parameter type. Flags and messages needed to evaluate 'event' are to be taken from 'mcanopen.h' and 'paraInfo.h'.

Example:

```
EVENT_T.event   = 0x00001022
EVENT_T.net     = 1
EVENT_T.node    = 5
EVENT_T.para.i  = 2

NET_NOT_VAL = 0x80000000
NOD_NOT_VAL = 0x40000000
PARA        = 0x3FFFFF00

0x00001022 & NET_NOT_VAL == 0       ==>   EVENT_T.net is valid
0x00001022 & NOD_NOT_VAL == 0       ==>   EVENT_T.node is valid
0x00001022 & PARA == 0x00001000     ==>   EVENT_T.para is valid
0x00001000 == PARA_I                ==>   EVENT_T.para.i is valid
0x00001022 & 0xFF == 0x22           ==>   lsb is 0x22, i.e. the error is
                                          the 34th error with integer
                                          parameter

Result: EVENT_T.net, Event_T.node, iParaInfo[0x22], EVENT_T.para.i
                          |      |              |                |
Meaning:          "Net: 1 Node: 5   guarding toggle error no    2"
```

Return: Success or error code described in the appendix.

mcanOpenGetNodeState()

Name: **mcanOpenGetNodeState()** - Return the current CANopen node state.

Call:

```
int    mcanOpenGetNodeState
       (
       int         net,
       int         nodeID,
       int         *state
       );
```

Description: This function returns the current CANopen node state of the node *nodeId* on logical CANopen net *net*. The state is returned in the parameter state and can have the following numerical values:

Numerical value	Node state
0x00	Offline
0x04	Stopped
0x05	Operational
0x7F	Pre-Operational

Return: Success or error code described in the appendix.

4.4 Extended Functions

The extended functions can be used to read/write the object dictionary of CANopen nodes and to force a change of node state.

---

mcanOpenReadSDO()

Name: **mcanOpenReadSDO()** - Reading an object-directory entry

Call:

```
int     mcanOpenReadSDO
        (
        unsigned short    net,
        unsigned short    modID,
        int               index,
        int               subindex,
        void              *buffer
        int               *len,
        ERROR_T           *error
        );
```

Description: As SDO client this function transfers the data of the object dictionary entry defined by *index* and *subindex* from the remote module *modId* into the memory of the caller given by the parameter *buffer.* The available buffer size has to be indicated by the caller in the parameter *len*. If the operation was successfully completed *buffer* contains the data and *len* contains the number of bytes received.

If the remote node given by *modId* is managed by the master the internal SDO handle is used for SDO transfer otherwise a new handle is opened for this operation and closed afterwards. The receive and transmit timeout for the internal handle can be configured individually for each node with the parameters SDO_TX_TIMEOUT and SDO_RX_TIMEOUT in the manager configuration file. The default RX/TX timeout for nodes not managed by the master is 200 ms.

If the function returned with the error code MCAL_ABORT_DOMAIN the following structure that has to be provided by the caller with the parameter *error* which is used to return the abort codes defined in DS-301.

```
typedef struct{
        unsigned char eclass;
        unsigned char code;
        unsigned char spec;
        unsigned char global;
      }  ERROR_T;
```

Return: Success or an error code listed in the appendix.

mcanOpenWriteSDO()

Name: **mcanOpenWriteSDO()** - Writing an object-directory entry

Call:

```
int     mcanOpenWriteSDO
        (
        unsigned short    net,
        unsigned short    modID,
        int               index,
        int               subindex,
        int               type,
        void              *buffer
        int               len,
        ERROR_T           *error
        );
```

Description: As SDO client this function writes the content of the data buffer *buffer* into the object directory of *modID* into the entry defined by *index* and *subindex*.

The parameter *type* specifies the data type. The following table contains the list of supported data types and their default transfer size.

Type descriptor	Transfer size in bytes
TYP_CHAR	1
TYP_INT8	1
TYP_INT16	2
TYP_INT32	4
TYP_UINT8	1
TYP_UINT16	2
TYP_UINT32	4
TYP_FLOAT	4
TYP_VIS_STRING	N/A
TYP_OCT_STRING	N/A
TYP_DATE	6
TYP_TIME_OF_DAY	6
TYP_TIME_DIFFERENCE	6
TYP_DOMAIN	N/A

If you want to transmit data with an arbitrary size you have to provide the length of the buffer with the parameter *len*.

If the transfer size of the buffer is less than 5 bytes the expedited SDO transfer mode instead of the segmented SDO transfer mode is used. In addition the parameter *type* can be logically or'd with TYP_INICATE_SIZE or TYP_INDICATE_NO_SIZE to determine that the data transfer size is indicated to the SDO server during initialization of the SDO transfer or not. The default behaviour is *length indication* for expedited transfer and *no length indication* for segmented transfer.

If the remote node given by *modId* is managed by the master the internal SDO handle is used for SDO transfer otherwise a new handle is opened for this operation and closed afterwards. The receive and transmit timeout for the internal handle can be configured individually for each node with the parameters SDO_TX_TIMEOUT and SDO_RX_TIMEOUT in the master configuration file (see page ?). The default RX/TX timeout for nodes not managed by the master is 2000 ms.

If the function returned with the error code MCAL_ABORT_DOMAIN the following structure that has to be provided by the caller with the parameter *error* which is used to return the abort codes defined in DS-301.

```
typedef struct{
        unsigned char eclass;
        unsigned char code;
        unsigned char spec;
        unsigned char global;
      }  ERROR_T;
```

Return: Success or an error code listed in the appendix.

mcanOpenNodeControl()

Name: **mcanOpenNodeControl()** - Requesting the control over a remote node

Call:

```
int    mcanOpenNodeControl
       (
       int    net,
       int    nodeID,
       long   flags
       );
```

Description: Obsolete function, present for API backward compatibility.

Return: Always success.

mcanOpenStartNode()

Name: **mcanOpenStartNode()** - Start a remote node

Call:
```
int    mcanOpenStartNode
       (
       int    net,
       int    nodeID
       );
```

Description: This functions starts the remote node *nodeId* on CANopen network *net*.

Return: Success or an error code listed in the appendix.

---

mcanOpenResetNode()

Name: **mcanOpenResetNode()** - Reset a remote node

Call:
```
int    mcanOpenResetNode
       (
       int    net,
       int    nodeID
       );
```

Description: This functions resets the remote node *nodeId* on CAN network *net*.

Return: Success or an error code listed in the appendix.

---

mcanOpenEnterPreop()

Name: **mcanOpenEnterPreop()** - Make remote node enter Pre-Operational

Call:
```
int    mcanOpenEnterPreop
       (
       int    net,
       int    nodeID
       );
```

Description: This functions makes the remote node *nodeId* on CAN network *net* enter the node state Pre-Operational.

Return: Success or an error code listed in the appendix.

mcanOpenStopNode()

Name: **mcanOpenStopNode()** - Stop the remote node

Call:

```
int    mcanOpenEnterPreop
       (
       int    net,
       int    nodeID
       );
```

Description: This functions makes the remote node *nodeId* on CAN network *net* enter the node state Stopped.

Return: Success or an error code listed in the appendix.

5. Appendix

5.1 Error Codes

The following tables list the possible error codes that can be returned by the manager library API calls. They are divided into three groups:

- Error codes of the driver.
- Error codes returned by reading the DCF files.
- General master error codes.

When evaluating return values you should never use the numerical values but should always use the constants defined for this error codes.

MCANOPEN_OK

Success (no warning or error).

Severity	Success
Description	The operation was executed without any errors.
Function	All functions

MCAL_CAN_RX_TIMEOUT

RX timeout for CAN receive operation

Severity	Application specific
Description	The application performed an operation to receive data from a remote node but the receive operation timed out. It is application specific if this is an expected behaviour or an error.
Solutions	If this behaviour is not expected: • Check remote node. • Check cables. • Check baudrate.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()*

MCAL_CAN_TX_TIMEOUT

TX timeout for CAN transmit operation

Severity	Error / Warning
Description	The application performed an operation to transmit data to a remote node but the transmit operation timed out. The reason for this behaviour could be a very high CAN bus load caused by CAN messages with a higher priority or a communication error.
Solutions	• Check cables. • Check baudrate.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()* *mcanopenStopNode()* *mcanopenStartNode()* *mcanopenResetNode()*

MCAL_CAN_TX_ERROR

TX error in CAN transmit operation

Severity	Error
Description	The application performed an operation to transmit data to a remote node but the transmit operation timed could not be performed. The reason for this behaviour is a communication error.
Solutions	• Check cables. • Check baudrate.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()* *mcanopenStopNode()* *mcanopenStartNode()* *mcanopenResetNode()*

MCAL_CAN_CONTR_OFF_BUS

Error in CAN transmit operation

Severity	Error
Description	The application performed an operation to transmit data to a remote node but the transmit operation could not be performed as the CAN controller is in BUS OFF state due to too many received error frames.
Solutions	Try again later as driver will try an automatic recovery. If an recovery is not possible you must check your cable and if all remote nodes using the same baudrate.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()* *mcanopenStopNode()* *mcanopenStartNode()* *mcanopenResetNode()*

MCAL_CAN_CONTR_WARN

Error in CAN transmit operation

Severity	Error
Description	The application performed an operation to transmit data to a remote node but the transmit operation could not be performed as the controller is in warning state due to too many received error frames.
Solutions	Try again later as driver will try an automatic recovery. If an recovery is not possible you must check your cable and if all remote nodes using the same baudrate.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()* *mcanopenStopNode()* *mcanopenStartNode()* *mcanopenResetNode()*

MCAL_CAN_CONTR_BUSY

Error in CAN transmit operation

Severity	Error / Warning
Description	The application performed an operation to transmit data to a remote node but this transmit operation could not be performed as the last transmit operation for this Id was not completed.
Solutions	• Decrease update frequency in your application. • Use flag TX_DONE_PDO in PDO definition.
Function	*mcanOpenWriteSDO()* *mcanOpenReadSDO()*

CANOPEN_DCF_INIT_ERROR

Error during initialization

Severity	Error
Description	A DCF file could not be opened.
Solutions	• Check paths and filename in master configuration file. • Check access rights on host.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_OUT_OF_MEMORY

Error during initialization

Severity	Error
Description	Not enough memory to process a DCF file.
Solutions	Increase memory size that can be utilized by the CANopen master.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_WRONG_NODE_ID

Error during initialization

Severity	Error
Description	The node id given in the master configuration file and given in the DCF file in section [Device Commissioning] differ.
Solutions	Use same module id's.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_WRONG_BAUDRATE

Error during initialization

Severity	Error
Description	The baudrate given in the master configuration file and given in the DCF file in section [Device Commissioning] differ or the baudrate is not a valid CANopen baudrate.
Solutions	• Use same baudrates in configuration and DCF file. • Use standard baudrate.
Function	calMStart() mcanopenConfig()

CANOPEN_DCF_MISSING_SECTION

Error during initialization.

Severity	Error
Description	A required section is missing in the DCF file.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_MISSING_ENTRY

Error during initialization

Severity	Error
Description	A required keyword is missing in a section of the DCF file.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_WRONG_PARAMETER

Error during initialization

Severity	Error
Description	A parameter of a certain keyword is invalid or missing.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_OUT_OF_RANGE

Error during initialization

Severity	Error
Description	The value of a parameter is out of range according to its data type. E.g. the value of an unsigned char parameter is greater than 256.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_OUT_OF_LIMITS

Error during initialization

Severity	Error
Description	The value of a parameter is out of range according to the given limits with keyword *HighLimit* and *LowLimit*.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_DOUBLE_SECTION

Error during initialization

Severity	Error
Description	A section name is not unique within the DCF file.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_DOUBLE_KEYWORD

Error during initialization

Severity	Error
Description	A keyword name is not unique within the scope of a section in the DCF file.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_UNKNOWN_KEYWORD

Error during initialization

Severity	Error
Description	An unknown keyword was found in the DCF file.
Solutions	Check and correct DCF file. Use the tool *dcftest* for more details.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_WRONG_NET_NO

Error during initialization

Severity	Error
Description	The static tables of the master are too small to configure a node on this net number.
Solutions	Choose a lower number for this network.
Function	*calMStart()* *mcanopenConfig()*

CANOPEN_DCF_WRONG_MODULE_NO

Error during initialization

Severity	Error
Description	The module number is out of the valid range 0 < modId < 128.
Solutions	Choose a correct module number.
Function	*calMStart()* *mcanopenConfig()*

MCAL_PARA_ERROR

Invalid parameter

Severity	Error
Description	One or more parameter of a function call were invalid.
Solutions	Compare parameter value with ranges given in the manual.
Function	All functions

MCAL_SYS_ERROR

Internal error

Severity	Error
Description	An internal data structure that is necessary for operation was not initialized or could not be initialized.
Solutions	• Compare order of initialization with manual. • If an initialization file is given check if the file exist and users access rights for the file are sufficient for the operation.
Function	All functions

MCAL_NO_NET

Net number invalid

Severity	Error
Description	The CANopen master is not initialized at all or a certain net is not initialized.
Solutions	• Compare order of initialization with manual. • Check if the net is registered in the master configuration file.
Function	All functions

MCAL_OBJ_NO_MEMORY

Error allocating a resource

Severity	Error
Description	Allocating a resource like memory or a synchronization object that is necessary to complete the operation failed.
Solutions	Increase the available memory for the CANopen master process.
Function	*calMStart()* *mcanopenConfig()* *mcanopenStart()*

MCAL_DOWNLOAD_ABORT_SLAVE

A slave aborted a CAL configuration

Severity	Error
Description	A CAL slave aborted the configuration download.
Solutions	Check CAL slave manual.
Function	*calMStart()* *mcanopenConfig()* *mcanopenStart()*

MCAL_WRONG_SLAVE_ANSWER

Error during CAL configuration

Severity	Error
Description	A CAL slave produced a protocol error during CAL configuration.
Solutions	Check CAL slave manual.
Function	*calMStart()* *mcanopenConfig()* *mcanopenStart()*

MCAL_WRONG_SERVER_ANSWER

Protocol error

Severity	Error
Description	An answer of a SDO server was wrong with respect to the SDO protocol.
Solutions	Check manual of remote CAL/CANopen slave.
Function	*mcanopenReadSDO()*

MCAL_TOGGLE_ERROR

Protocol error

Severity	Error
Description	An answer of a SDO server was wrong with respect to the SDO protocol.
Solutions	Check manual of remote CAL/CANopen slave.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()*

MCAL_BUFFER_OVERFLOW

Insufficient memory

Severity	Error
Description	The memory that was provided by the caller was insufficient to keep all the date that was received by the SDO server.
Solutions	Increase memory to store data.
Function	*mcanopenReadSDO()*

MCAL_ABORT_DOMAIN

Protocol error

Severity	Error
Description	The SDO server of a remote CANopen node aborted the transfer.
Solutions	Check error code returned by the call and the manual of the remote CANopen slave to find out the reason of the error condition.
Function	*mcanopenReadSDO()* *mcanopenWriteSDO()*

MCAL_NODE_NOT_EXISTEND
Unknown node

Severity	Error
Description	A remote node with a given module id is not managed by the master for the given net.
Solutions	Check if node is registered in the master configuration file and is correctly configured during startup.
Function	*mcanopenGetNodeInfo()* *mcanopenGetNodeState()* *mcanopenNodeControl()* *mcanopenResetNode()* *mcanopenStartNode()* *mcanopenStopNode()*

MCAL_WRONG_GUARD_ID
Guard identifier out of range

Severity	Error
Description	The identifier for node guarding is out of range with respect to the CAL or CANopen draft standard.
Solutions	Compare valid range in standard with given value in configuration file.
Function	*calMStart()* *mcanopenConfig()*

MCAL_NEW_GUARD_PARA
Guard identifier mismatch

Severity	Warning
Description	The identifier of a CANopen slave for node guarding given in configuration file and given in DCF file differ.
Solutions	Use the same identifier or use only DCF file to configure guarding information.
Function	*calMStart()* *mcanopenConfig()*

MCAL_NODE_NOT_FREE
Getting node control failed

Severity	Warning / Error
Description	Taking over the node control for a given node on a given net was not possible as the CANopen master itself is accessing the node at the moment.
Solutions	Try taking over the node control later.
Function	*mcanopenGetNodeControl()*

MCAL_NO_EVENT
Event queue empty

Severity	Warning
Description	If the master event queue is polled this is an indication that the event queue is empty at the moment.
Solutions	N/A
Function	*mcanopenEvent()*

MCAL_UNSUPPORTED
Unsupported operation

Severity	Warning / Error
Description	The operation could not be performed as this service is unsupported in this version of the CAL/CANopen master.
Solutions	N/A
Function	N/A