CAN in Automation e. V.

CANopen
Communication Profile for Industrial Systems

Based on CAL

CiA Draft Standard 301

Revision 3.0

Date: 30.10.96

History

Date	Changes
Oct 96	Document completely revised; Summary of major changes: • Object Dictionary structure extended (downwards compatible) • Mapping of Emergency Telegram clarified • Boot-Up procedure clarified • Minimum capability device enhanced • Default Identifiers for Node guarding added • Transmission types enhanced • PDO Mapping clarified • Communication Parameters of supported PDOs made mandatory • SDO parameters included in Object Dictionary • Storing/Restoring of parameters clarified / Objects added • Object Dictionary structure accessible • Chapter „Physical Layer“ included • Electronic Data Sheet specification included

Errata Sheet for CiA DS-301 V 3.0 22.05.98
CANopen Communication Profile

changes are printed in *italics*

Chapter 8.1, Top of page 8-5:
Add: Power on values are the last stored parameters. If storing is not supported or has not been executed *or if the reset was preceded by a restore_default command (object 1011H),* the power on values are the default values according...
Chapter 10.1, Bottom of page 10-2:
Add: The range 6000H-*9*FFFH contains the standardised device profile parameters. *The range A000H-FFFFH is reserved for future use.*
Chapter 10.1.1, Bottom of page 10-4
Add:.. ...it enables one to upload the entire structure of the object dictionary. *This entry FFh is not counted in the number of entries in sub-index 0*.
Chapter 10.1.3, Top of page 10-8
Replace:.. first an “8” is written to sub-index *0* , thus setting ....
Chapter 10.1.4, Table 10-11, page 10-9 + 10-10:
Index 100C (guard time): Replace Unsigned 32 by *Unsigned16*
Index 100D (life time factor): Replace Unsigned32 by *Unsigned8*
Index 12FF: Replace Server SDO parameter by *Client* SDO parameter
Chapter 10.3, Page 10-29, Object 1014H COB-ID Emergency Message
Replace: The structure of this object is shown in Figure 10-8 and Table 10-15, it is similar to the entry 1005h (COB-ID SYNC message).
By: *The structure of this object is shown in Figure 10-10 and Table 10-15, it is similar to the entry 100Eh (Node Guarding Identifier).*
Chapter 10.3, Top of page 10-30
Replace:.. All other server SDOs are invalid by default (invalid bit - see table 10-*10*).
Chapter 10.3, Top of page 10-34:
Change: object code for objects 1600H-17FFH is *ARRAY,* value range is *Unsigned8*
Chapter 10.3, Top of page 10-37:
Change: object code for objects 1A00H-1BFFH is *ARRAY*
Chapter 12.5.1, Top of page 12-5:
Add: ...ANSI character set. *The keywords are not case-sensitive.*
Chapter 12.5.2, Top of page 12-6:
Add: ..for a particular device class. *The EDS consists of sections designated by a section name. Each section contains entries designated by keywords.*
Chapter 12.5.2.1, Page 12-6:
Replace: characters in format „mm-dd-yy“; By: characters in format „mm-dd-yyyy“
Chapter 12.5.2.3.3, Page 12-10:
Add: The entries are numbered *decimal* beginning with number 1.
Add:.. SubNumber - number of sub-indices available at this index, n*ot counting sub-index FFh (255dez).*
Add: It is not necessary to use all keywords in every section. *ParameterName, ObjectType, DataType, AccessType however must be present in every object section. The absence of the following keywords means*
LowLimit - No
HighLimit - No
DefaultValue - No
All other keywords are handled in the way that keywords for numeric values are evaluated as 0 and for string values as empty strings.
The absence of a keyword is equal to the statement
<keyword>= without a value.
Chapter 12.5.2.3.4, Middle of page 12-13:
Add between the description of the structure and the example: *The list of object links is numbered decimal beginning with number 1.*
Chapter 12.5.2.4, Top of page 12-14:
Add between the description of the structure and the example: *The line number is decimal beginning with number 1.*

Table of Contents

1 Scope

Devices which have to communicate can be found in nearly every automated system. Unfortunately there is no standardised product definition with respect to the control techniques of functional elements. Therefore, each system integrator of automated systems creates his own standards.

The resulting control and information flow concepts differ between each integrator and in most cases the employed control devices are not even compatible.

This document contains the description of the CANopen Communication Profile for industrial applications using CAN as their installation bus. It is based on the CAN Application Layer (CAL) (see /4, .., 16/) specification from the CAN in Automation international users and manufacturers group (CiA e.V.).

The CANopen Communication Profile forms the interface between the CAN Application Layer and the CANopen Device Profiles. It includes the real-time communication model and the protocols which are common to all devices in the network. Device Profiles, on the other hand, are a common means to describe device specific functionality.

An introduction to CANopen Device Profiles is given in this document as well, however, the specification of full device profiles does not fall within the scope of this document.

2 References

/1/: ISO 7498, 1984, Information Processing Systems - Open Systems Interconnection - Basic Reference Model

/2/: ISO 11898, 1993, Road Vehicles, Interchange of Digital Information - Controller Area Network (CAN) for high-speed Communication

/3/: Robert Bosch GmbH, CAN Specification 2.0 Part A+B, September 1991

/4/: CiA DS-102, CAN Physical Layer for Industrial Applications, April 1994

/5/: CiA DS-201, CAN Reference Model, February 1996

/6/: CiA DS-202/1, CMS Service Specification, February 1996

/7/: CiA DS-202/2, CMS Protocol Specification, February 1996

/8/: CiA DS-202/3, CMS Data Types and Encoding Rules, February 1996

/9/: CiA DS-203/1, NMT Service Specification, February 1996

/10/: CiA DS-203/2, NMT Protocol Specification, February 1996

/11/: CiA DS-204/1, DBT Service Specification, February 1996

/12/: CiA DS-204/2, DBT Protocol Specification, February 1996

/13/: CiA DS-207, Application Layer Naming Specification, February 1996

/14/: CiA DS-205/1, LMT Service Specification, February 1996

/15/: CiA DS-205/2, LMT Protocol Specification, February 1996

3 Definitions

CAN	Controller Area Network
CiA	CAN in Automation international users and manufacturers group e.V.
CMS	CAN Message Specification. One of the service elements of the CAN Application Layer in the CAN Reference Model.
COB	Communication Object. (CAN Message) A unit of transportation in a CAN Network. Data must be sent across a Network inside a COB.
COB-ID	COB-Identifier. Identifies a COB uniquely in a Network. The identifier determines the priority of that COB in the MAC sub-layer too.
DBT	Distributor. One of the service elements of the CAN Application Layer in the CAN Reference Model. Its the responsibility of the DBT to distribute COB-ID's to the COB's that are used by CMS.
DIN	Deutsches Institut für Normung
ISO	International Standardisation Organisation
LMT	Layer Management. One of the service elements of the CAN Application Layer in the CAN Reference Model. It serves to configure parameters of each layer in the CAN Reference Model.
NMT	Network Management. One of the service elements of the CAN Application Layer in the CAN Reference Model. It performs initialisation, configuration and error handling in a CAN network.
OSI	Open Systems Interconnection
PDO	Process Data Object
SDO	Service Data Object

4 Introduction

4.1 CANopen Communication Reference Model

The communication concept can be described similar to the ISO-OSI Reference Model.

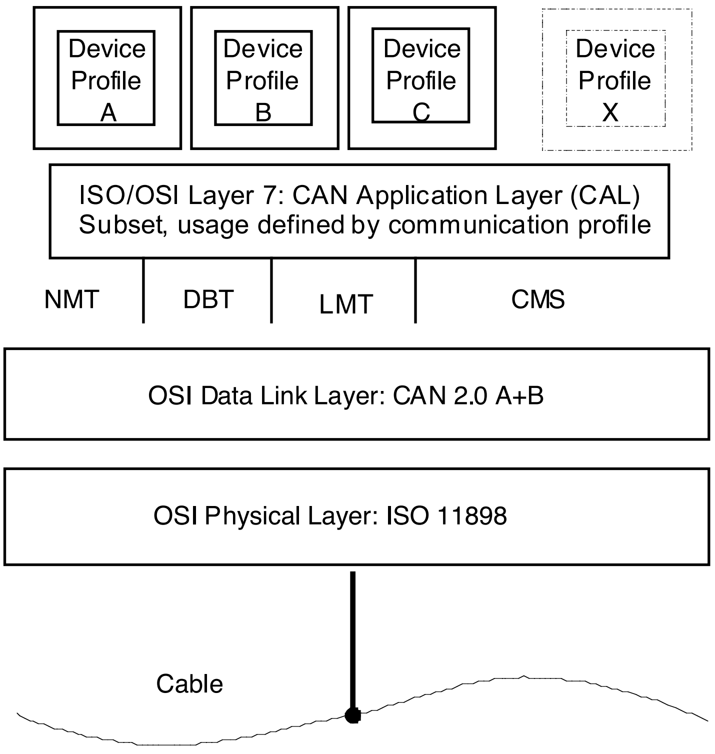

Device Profile A
Device Profile B
Device Profile C
Device Profile X
ISO/OSI Layer 7: CAN Application Layer (CAL)
Subset, usage defined by communication profile
NMT
DBT
LMT
CMS
OSI Data Link Layer: CAN 2.0 A+B
OSI Physical Layer: ISO 11898
Cable

Figure 4-1: The CANopen Communication Reference Model

OSI-Layer 1-7:
The layered structure of the communication model is briefly described in /5/.

CANopen Communication Profile:

The CANopen Communication Profile comprises a concept to configure and communicate real-time-data as well as the mechanisms for synchronisation between devices. Basically the CANopen Communication Profile describes how a subset of CAN Application Layer (CAL) services is used by devices.

CANopen Device Profiles:
The device profile describes the functionality of a particular device.

4.2 Standardisation Via Profiling

The two principal advantages of the profile approach to device specification are in the areas of system integration and device standardisation. If two independent device manufacturers are to design products which are to communicate with each other then each manufacturer must be provided with a specification of the other manufacturers device. This specification could take many forms if left to individual manufacturers to produce. The concept of device profiling provides a standard for producing such specifications. By adopting this approach all manufacturers will specify their devices in a similar fashion which greatly reduces the effort involved in system integration.

The other clear advantage of the profile approach to device specification is that it can be used to guide manufacturers into producing standardised devices. The advantages of standardised devices are numerous. Perhaps most importantly the idea of a standardised device de-couples a system integrator from a specific supplier. If one supplier cannot meet product demand, for example, the integrator can use devices from another supplier without having to re-configure network software. On the other hand the supplier is not forced any more to implement a private protocol for each customer.

A device profile defines a *standard device*. This standard device specifies a basic functionality which every device within a class must exhibit. This mandatory functionality is necessary to ensure at least simple non-manufacturer-specific operation of a device is possible[1].

The concept of device standardisation is extended by the notion of optional functionality defined within the standard device profiles. Such optional functionality does not have to be implemented by all manufacturers. However, if a manufacturer wishes to implement such functionality he must do so in the manner defined for the standard device.

The concept of optional functionality provides a very powerful mechanism to ensure all manufacturers implementing particular functionality do so in a defined fashion[2].

The device profiles provide a mechanism by which manufacturers wishing to implement truly manufacturer specific functionality can do so. This is clearly necessary since it would be impossible to anticipate all possible device functionality and define this in the optional category of each device class. This approach guarantees that the standard device profiles are 'future-proof'.

By defining mandatory device characteristics basic network operation is guaranteed. By defining optional device features a degree of defined flexibility can be built in. By leaving 'hooks' for manufacturer specific functionality manufacturers will not be constrained to an out-of-date standard.

---

1 For example the standard drive unit provides a 'HALT' function to stop a drive from moving. This function is defined as mandatory such that any drive unit supporting the drive profile can be halted using the same message.

2 For example, the standard digital I/O module may define optional functionality to cater for units with up to 64 I/O channels (This is specified in the device profile). Whilst many units will not use anything like this number of I/O the definition ensures that 64-channel I/O modules developed by independent manufacturers will be largely interchangeable.

4.3 The Object Dictionary

The most important part of a device profile is the object dictionary description. The object dictionary is essentially a grouping of objects accessible via the network in an ordered pre-defined fashion. Each object within the dictionary is addressed using a 16-bit index.

The overall layout of the standard object dictionary is shown below. This layout closely conforms with other industrial Fieldbus concepts:

Index (hex)	**Object**
0000	not used
0001-001F	Static Data Types
0020-003F	Complex Data Types
0040-005F	Manufacturer Specific Data Types
0060-007F	Device Profile Specific Static Data Types
0080-009F	Device Profile Specific Complex Data Types
00A0-0FFF	Reserved for further use
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardised Device Profile Area
A000-FFFF	Reserved for further use

Table 4-1: Object Dictionary Structure

The Standard Object Dictionary may contain a maximum of 65536 entries which are addressed through a 16bit index.

The Static Data Types at indices 0001h through 001Fh contain type definitions for standard data types like Boolean, integer, floating point, string, etc. These entries are included for reference only, they cannot be read or written.

Complex Data Types at indices 0020h through 003Fh are pre-defined structures that are composed of standard data types and are common to all devices.

Manufacturer Specific Data Types at indices 0040h through 005Fh are also structures composed of standard data types but are specific to a particular device.

Device Profiles may define additional data types specific to their device type. The static data types defined by the device profile are listed at indices 0060h - 007Fh, the complex ones at indices 0080h - 009Fh.

A device may optionally provide the structure of the supported complex data types (indices 0020h - 005Fh and 0080h - 009Fh) at read access to the corresponding index. Sub-index 0 then provides the number of entries at this index, and the following sub-indices contain the data type encoded as Unsigned16 according to table 10-4.

The Communication Profile Area at indices 1000 through 1FFF contains the communication specific parameters for the CAN network. These entries are common to all devices.

The Standardised Device Profile Area at indices 6000h through 9FFFh contains all data objects common to a class of devices that can be read or written via the network. The object dictionary for each device type has a range of mandatory entries. These entries ensure that all devices of a particular type behave in a defined manner (at least from a basic functionality viewpoint).

The object dictionary concept caters for optional device features which means a manufacturer does not have to provide certain extended functionality on his devices but if he wishes to do so he must do it in a pre-defined fashion[3].

By defining object dictionary entries for anticipated increased functionality in an optional category manufacturers wishing to implement enhanced functionality will all do so in the same way[4].

4.3.1 Index and Sub-Index Usage

A 16-bit index is used to address all entries within the object dictionary. In case of a simple variable this references the value of this variable directly. In case of records and arrays however, the index addresses the whole data structure.

To allow individual elements of structures of data to be accessed via the network a sub-index has been defined. For single object dictionary entries such as an unsigned8, Boolean, integer32 etc. the value for the sub-index is always zero. For complex object dictionary entries such as arrays or records with multiple data fields the sub-index references fields within a data-structure pointed to by the main index. For example on a single channel RS232 interface module there may exist a data-structure at index 6092h which defines the communication parameters for that module. This structure may contain fields for baud-rate, number of data bits, number of stop bits and parity type. The sub-index concept can be used to access these individual fields[5] as shown below:

Main Index	**Sub Index**	**Variable Accessed**	**Data Type**
6092	0	Number of Entries	Unsigned8
	1	Baud Rate	Unsigned16
	2	Number Of Data Bits	Unsigned8
	3	Number Of Stop Bits	Unsigned8
	4	Parity	Unsigned8

Table 4-2: Use of Index and Sub-Index

A sub-index can also be used to reference a similar data field for more than one channel. For example a digital output module may have the capability to configure individual output signals for different polarity. The object dictionary entry at index 6022h may define the polarity information. A manufacturer could define his object dictionary such that the polarity of channel 1 was set by writing data to index 6022 sub-index 1. Similarly the polarity for channel 3 could be configured by writing the polarity information to index 6022 sub-index 3. This is allowed since the sub-index is meant to index elements of a data structure. Multiple channels of the same type can be viewed as an array of channels [6] of the same type - which is a structure.

[3] For example the mandatory part of the object dictionary for a digital output module could define how to access a minimum number of outputs (8 for example). Manufacturers wishing to implement devices with eight outputs would merely conform with the defined standard. However manufacturers wishing to make modules with a greater number of outputs would have no standard to operate within. They would be free to define the communication with the other output signals as they wished. This could lead to module incompatibility problems.

[4] Space is left in the object dictionary at indices 2000h through 5FFFh for truly manufacturer specific functionality.

[5] The fields accessed by the sub-index can be of differing data types.

[6] Channel numbering must begin at sub-index 1, be consecutive, and the entry for sub-index 0 must reflect the number of channels implemented on a device (Unsigned8).

5 Communication Model

The CANopen communication model specifies the different communication objects and services and the available modes of message transmission triggering.

The communication model supports the transmission of synchronous and asynchronous messages. By means of synchronous message transmission a network wide co-ordinated data acquisition and actuation is possible. The synchronous transmission of messages is supported by pre-defined communication objects (Sync message, time stamp message). Synchronous messages are transmitted with respect to a pre-defined synchronisation message, asynchronous message may be transmitted at any time.

With respect to their functionality, four types of messages (objects) are distinguished:

- Administrative Messages (Layer Management, Network Management and Identifier Distribution Messages)
- Service Data Messages
- Process Data Messages
- Pre-defined Messages (Synchronisation-, Time Stamp-, Emergency Messages)

By means of **Administrative Messages** the setting up of layer specific parameters (Layer Management services), the initialisation, configuration (see chapter 8.2) and supervision of the network is performed. Services and protocols of these functions are according to the LMT (Layer Management), NMT (Network Management) and DBT (Distributor) service entities of the CAL specification (/9/, /10/, /11/, /12/, /14/, /15/).

Service Data Messages are used for read and write access to all entries of the object dictionary of a device. The main usage of this facility is device configuration.

By means of **Process Data Messages** the real-time data transmission is performed.

The main features of Service and Process Data Objects are shown in table 5-1.

Process Data Object	Service Data Object
used for real-time data exchange	access to a device object dictionary entry by index and sub-index
typically high priority messages	low priority messages
synchronous and asynchronous message transmission[7]	transmitted asynchronously
cyclic and acyclic transmission[8]	
data content configurable via SDOs	usage of data field determined by CMS Multiplexed Domain protocol

Table 5-1: Main Features of Process Data and Service Data Objects

By means of **Pre-defined Messages** node synchronisation, time stamping and emergency notification is supported optionally. The corresponding objects are described in chapter 7.

[7] see chapter 5.2

[8] see chapter 5.2

5.1 The Service Data Object

5.1.1 SDO usage

With "Service Data Objects (SDOs)" the access to entries of a device object dictionary is provided. A SDO is represented by a CMS object of type "Multiplexed Domain" according to the CAL specification /6, 7/. By means of a SDO a peer-to-peer communication channel between two devices may be established. The owner of the accessed object dictionary is the server of the SDO. A device may support more than one SDO. One supported SDO is the default case.

5.1.2 Services

According to the CAL specification /6/ the following services can be applied onto a SDO depending on the application requirements:

- Define Domain
- Domain Download
- Domain Upload
- Initiate Domain Download
- Download Domain Segment
- Initiate Domain Upload
- Upload Domain Segment
- Abort Domain Transfer

The following attributes specify a SDO domain object:

• name:	according to CMS naming conventions with <application-specific name> = "SDO_xxx“ with „xxx“ as the number of the SDO, starting with 001. xxx = 001 - 128 for Server SDOs, 129 - 256 for Client SDOs.
• user type:	client or server (owner of accessed object dictionary = server)
• class:	Multiplexed Domain
• priority:	application specific, suggested between [6,7]
• mux data type:	STRUCTURE OF UNSIGNED (16) index, UNSIGNED (8) sub-index with "index" specifying an entry of the device object dictionary and "sub-index" specifying a component of a device object dictionary entry

5.1.3 Protocols

The specification of the Multiplexed Domain protocol is according to /7/.

For transmission of messages with data length less than 5 bytes an "expedited" transfer may be performed by means of the "Initiate Domain Download/Upload" protocols. With these protocols the data transmission is performed within the initiate protocol. The transfer of messages of more than 4 data bytes has to be initiated by the "Initiate Domain Down- or Upload"-protocol.

It shall be pointed out that the CMS domain protocols indicate a failure response by initiating the Abort-Protocol. For the CANopen Communication Profile the reason of an abort is coded within a 4 byte "application error code" as shown in Table 5-2 and Table 5-3.

The application error codes are a small subset of the error classes of the PROFIBUS-specification (EN 50170) represented by a 4 byte value composed of an "error class", "error code" and "additional code" field (Figure 5-1).

In addition to the protocol specification (see /7/) the CANopen Communication Profile defines the application error codes carried by the abort domain transfer protocol[9].

The appl-error-codes field is a 32bit value composed of the following elements:

• Error-Class:	1 Octet
• Error-Code:	1 Octet
• Additional Code:	2 Octets

Byte:	4	6	7	8
	Additional Code	Error-Code	Error-Class	

Figure 5-1: Structure of the Application Error Codes

The Additional Code is also broken up into the following fields:

Bit:	15 8	7 4	3 0
	Reserved	Global-Code	Specific-Code

Figure 5-2: Structure of the Additional Code

The combination of the Error Class and the Error Code (see Table 5-2) explain the error which has been occurred. The Additional Code (see Table 5-3) is necessary with some error types to give further details of the fault.

[9] The Index and Sub-Index parameters in the Abort Transfer protocol are optional. Since there can be only one Domain Protocol outstanding at any time which is already identified by the COB-ID, the transmission of Index/Sub-index parameters is redundant. If Index/Sub-index fields are not valid, they must be set to 0.

Error Class Value	Error Code Value Meaning	Example
5 Service error	3 Parameter Inconsistent	Toggle bit not alternated.
	4 Illegal Parameter	Time out value reached.
6 Access error	1 Object Access Unsupported	Attempt to write a read-only or to read a write-only parameter.
	2 Object non-existent	The object does not exist in the dictionary.
	6 Hardware fault	Access failed because of an hardware error.
	7 Type Conflict	Data type does not match.
	9 Object attribute inconsistent	The sub-index does not exist.
8 Other error	0	Transfer was aborted by user.

Table 5-2: Error Class and Code

The error classes/codes not listed here are reserved.

Additional Code	Meaning
0	No precise details for the reason for the error
10H	Service parameter with an invalid value
11H	Sub-Index does not exist
12H	Length of service parameter too high
13H	Length of service parameter too low
20H	Service cannot currently be executed
21H	because of local control
22H	because of the present device state
30H	Value range of parameter exceeded
31H	Value of parameter written too high
32H	Value of parameter written too low
36H	Maximum value is less than minimum value
40H	Incompatibility with other values
41H	data cannot be mapped to the PDO
42H	PDO length exceeded
43H	General parameter incompatibility reason
47H	General internal incompatibility in the device

Table 5-3: Additional Code (Global Code and Specific Code)

The additional codes not listed here are reserved.

5.2 The Process Data Object

5.2.1 PDO Usage

The real-time data transfer is performed by means of "Process Data Objects (PDO)". PDOs are represented by CMS objects of type "Stored-Event" according to the CAL specification /6, 7/. Hence the transfer of PDOs is performed with no protocol overhead.

The PDOs correspond to entries in the device Object Dictionary and provide the interface to the application objects. Data type and mapping of application objects into a PDO is determined by a corresponding default PDO mapping structure within the Device Object Dictionary. If variable PDO-mapping is supported the number of PDOs and the mapping of application objects into a PDO may be transmitted to a device during the device configuration process (see chapter 8.2) by applying the corresponding SDO services.

Number and length of PDOs of a device is application specific and have to be specified within the device profile.

5.2.1.1 Transmission Modes

The following PDO transmission modes are distinguished:

- Synchronous Transmission
- Asynchronous Transmission

In order to synchronise devices a synchronisation object (SYNC object) is transmitted periodically by a synchronisation application. The SYNC object is represented by a pre-defined communication object (see chapter 7.1). In Figure 5-3 the principle of synchronous and asynchronous transmission is shown. Synchronous PDOs are transmitted within a pre-defined time-window immediately after the SYNC object. The principle of synchronous transmission is described in more detail in chapter 6.

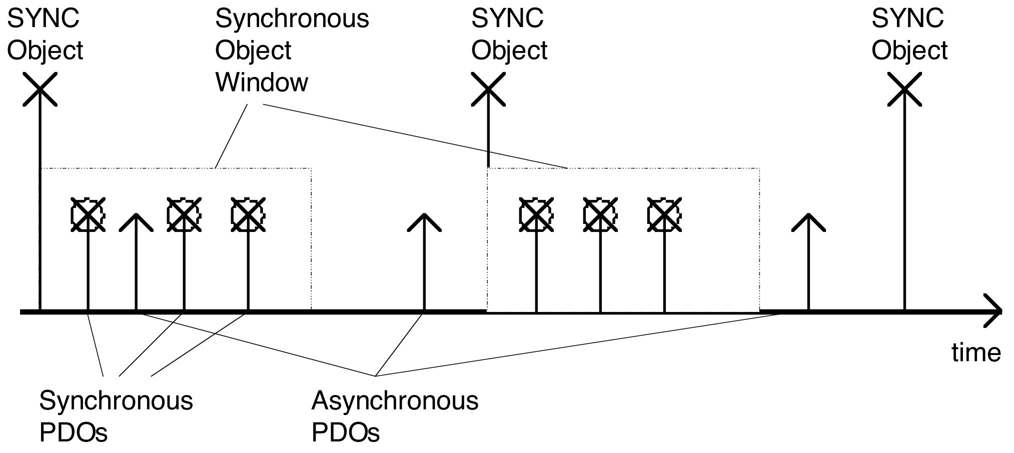


Figure 5-3: Synchronous and Asynchronous Transmission

The transmission rate (see also: Table 10-7) of a synchronous PDO is specified within the entry "Transmission Type" in the Device Object Dictionary in form of a factor (PDO-rate) of the basic SYNC-object transmission period. A transmission type of zero means that the message shall be transmitted synchronously with the SYNC object but acyclic (not periodically).

Asynchronous PDOs may be transmitted without any relation to a SYNC (PDO-rate = 254, 255).

In Figure 5-4 the classification of synchronous and asynchronous PDOs is shown.

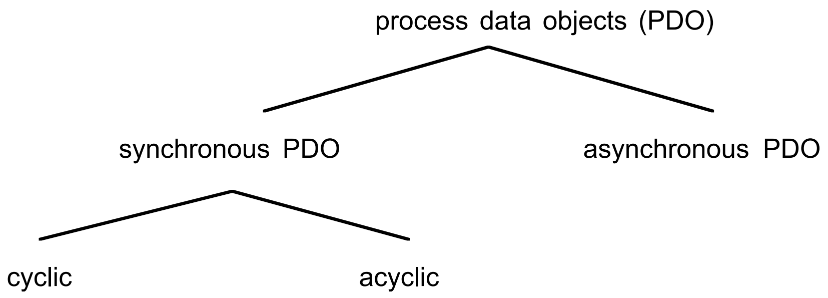


Figure 5-4: Synchronous and Asynchronous PDOs

5.2.1.2 Triggering Modes

The CANopen Communication Profile distinguishes two message triggering modes:

- **Event Driven**
  Message transmission is triggered by the occurrence of an object specific event. For synchronous PDOs this is the expiration of the specified transmission period, synchronised by the reception of the SYNC object.

  For acyclically transmitted synchronous PDOs and asynchronous PDOs the triggering of a message transmission is an application specific event specified in the device profile.

- **Remotely requested**
  The transmission of asynchronous PDOs may be initiated on receipt of a remote request initiated by another device.

5.2.2 PDO Services

The specified PDO triggering modes of the CANopen Communication Profile are modelled by services onto the CMS object type "Stored-Events" according to /6/. CMS objects of type Stored-Event provide event-driven and remotely-requested transmission.

According to CMS the following services on Events are relevant:

- Define Event
- Store (and immediately Notify) Event
- Read Event

The following CMS event object attributes are specified for event PDOs:

• name:	according to the CMS naming conventions with <application-specific-name> = "xPDOyyy" with „yyy“ as the number of the PDO, starting with 001, and with x = {„T“|“R“} for a receive or transmit PDO.
• user type:	depending on the role-in-service of the device; transmitter of event data: server receiver or requester of event data: client
• class:	Stored Event
• priority:	application specific, suggested between [2, 5]
• data type:	determined by corresponding PDO-mapping entry of the PDO
• inhibit time:	application specific

5.2.3 PDO Protocol

The Stored-Event protocols are according to /7/.

6 Synchronisation by the SYNC Master

6.1 Transmission of Synchronous PDO Messages

Synchronous transmission of a message means that the transmission of the message is fixed in time with respect to the transmission of the SYNC message. The synchronous message is transmitted within a given time window with respect to the SYNC transmission, and at most once for every period of the SYNC.

In general the fixing of the transmission time of synchronous PDO messages coupled with the periodicity of transmission of the SYNC message guarantees that sensor devices may arrange to sample process variables and that actuator devices may apply their actuation in a co-ordinated fashion.

Assuming a structure where a master device controls a number of slave devices by issuing a COMMAND message and receiving the ACTUAL messages from the slave devices. Synchronous COMMAND messages transmitted by the master device will occur in a definite time period with respect to the transmission of the SYNC message.

In general the COMMAND cyclic message will be transmitted before a SYNC and the device will actuate based on this COMMAND at the next SYNC. Simple devices operating in the cyclic mode may therefore use the SYNC message as a trigger to output or actuate based upon their previous COMMAND.

Depending upon its capabilities, a device may also be parameterised with the time period *synchronous window length* after the SYNC at which it is guaranteed that its COMMAND has arrived. It may therefore perform any processing on the COMMAND data which is required in order to actuate at the next SYNC message.

The arrival of a SYNC will also prompt a device operating in the cyclic mode to sample its feedback data and transmit an ACTUAL back to the master device as soon as possible afterwards.

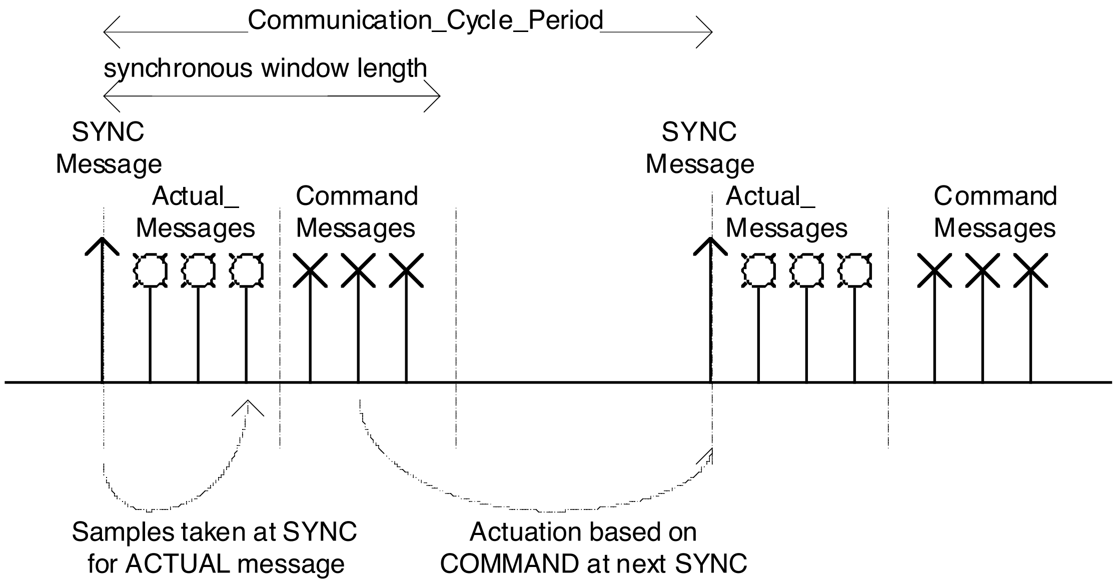

Communication_Cycle_Period
synchronous window length
SYNC Message
Actual_ Messages
Command Messages
SYNC Message
Actual_ Messages
Command Messages
Samples taken at SYNC for ACTUAL message
Actuation based on COMMAND at next SYNC

Figure 6-1: Bus Synchronisation and Sampling/Actuation

6.2 Optional High Resolution Synchronisation Protocol

The standard synchronisation mechanism provides for a lean implementation. The synchronisation message carries no data and is easy to generate. However, the jitter of this SYNC depends on the bit rate of the bus as even the very high priority SYNC has to wait for the current message on the bus to be transmitted before it gains bus access.

Some time critical applications especially in large networks with reduced transmission rates require more accurate synchronisation; it may be necessary to synchronise the local clocks with an accuracy in the order of microseconds. This is achieved by using the optional high resolution synchronisation protocol which employs a special form of time stamp message (see Figure 6.2) to adjust the inevitable drift of the local clocks.

The synchronisation master time-stamps the interrupt generated at t1 by the successful transmission of the SYNC message (this takes until t2). After that (at t4) he sends a time-stamp message containing the corrected time-stamp (t1) for the SYNC transmission success indication. The slaves that have taken local time-stamps (t3) on the reception (t1) of the SYNC can now compare their corrected time-stamp (t1) with the one received in the time-stamp message from the master. The difference between these values determines the amount of time to adjust the local clock. With this protocol only the local latencies (t2-t1 on the master and t3-t1 on the slave) are time critical. These latencies depend on local parameters (like interrupt processing times and hardware delays) on the nodes which have to be determined once. The accuracy of this determination is implementation specific, it forms the limiting factor of the synchronisation (or clock adjustment) accuracy. Note that each node only has to know its own latency time as the time-stamp message contains the corrected value t1 and not t2.

The time-stamp is encoded as unsigned32 with a resolution of 1 microsecond which means that the time counter restarts every 72 minutes. It is configured by mapping the high resolution time-stamp (Object 1013h) into a PDO.

It is reasonable to repeat the clock adjustment only when the maximum drift of the local clock exceeds the synchronisation accuracy. For most implementations this means that it is sufficient to add this time-stamp message to the standard SYNC once every second.

This principle enables the best accuracy that can be achieved with bus-based synchronisation, especially when implemented on CAN controllers that support time-stamping. Note that the accuracy is widely independent of the transmission rate. Further improvement requires separate hardware (e.g. wiring).

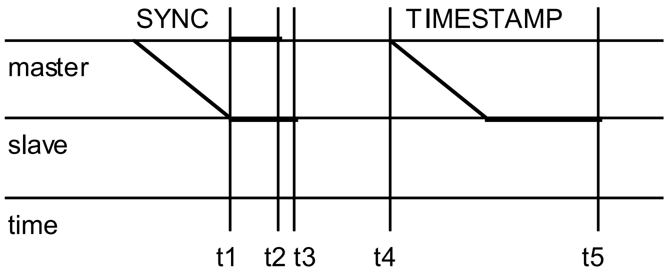


Figure 6-2: Optional High Resolution Synchronisation Protocol

6.3 Other Synchronisation

The synchronisation mechanisms described in the preceding sections are mainly optimised for a synchronisation between a master and other devices. For special applications however, further synchronisation between (slave-) devices may be required as well. For instance, in order to allow multiple drive units to operate as an electronic gear, it may be necessary to configure one drive as the gearing master which broadcasts a master angle message to all slave drives at much shorter time intervals than the Communication Cycle. For these purposes, further synchronisation messages may be defined between any devices on the network. However, care should be taken not to increase the jitter of the SYNC when broadcast messages are to be transmitted at higher frequencies than the SYNC.

7 Pre-defined Communication Objects

There are three pre-defined Communication Objects. The implementation of these objects is optional.

7.1 The SYNC Object

7.1.1 SYNC Usage

The SYNC object is broadcasted periodically by the synchronisation device to all application devices. This SYNC object provides the basic network clock. The time period between the SYNC objects is specified by the standard parameter **communication cycle period**, which may be written by a configuration tool to the application devices during the boot-up process. There can be a time jitter in transmission by the SYNC master corresponding approximately to the latency due to some other message being transmitted just before the SYNC.

In order to guarantee timely access to the CAN bus the SYNC object is given a very high priority identifier [10]. Devices which operate synchronously may use the SYNC object to synchronise their own timing with that of the synchronisation device. The details of this synchronisation are device dependent and do not fall within the scope of this document. Devices which require a more accurate common time base may use the time stamp synchronisation mechanism described in chapter 6.2.

7.1.2 SYNC Services

The SYNC object is implemented as CMS object of type "Basic Variable" with the synchronisation device acting as the client of that object.

According to CMS the following services on variables are relevant:

- Define Variable
- Write Variable

The following CMS variable object attributes are specified for the SYNC object:

• name:	according to the CMS naming conventions with <application-specific-name> = "SYNC000"
• user type:	synchronisation master device: client receiving device: server
• class:	Basic Variable
• priority:	0 (suggested)
• data type:	NIL
• access type:	Write_Only
• inhibit time:	<communication period>

[10] For the SYNC message identifier 128 in priority group 0 is suggested

7.1.3 SYNC Object Protocols

See specification of Basic Variable protocols /7/.

7.2 The Time Stamp Object

7.2.1 Time Stamp Object Usage

By means of the Time-Stamp-Object a common time frame reference is provided to application devices[11].

7.2.2 Time Stamp Object Services

The Time-Stamp object is implemented as CMS object of type Stored Event with the transmitting device acting as the server of the event.

According to CMS the following services on Stored Events are relevant:

- Define Event
- Notify Event
- Read Event

The following CMS event object attributes are specified for the Time-Stamp object:

- name: according to the CMS naming conventions with
  <application-specific-name> = "TIME000"
- user type: Time stamp provider: server
  Time stamp consumers: clients
- class: Stored Event
- priority: 1 (suggested)
- data type: TIME-of-DAY [12]
- inhibit time: application specific

7.2.3 Time Stamp Object Protocols

See specification of the Stored Event protocols /7/.

[11] For the Time Stamp message identifier 256 in priority group 1 is suggested.

[12] Additional Time Stamp messages with different data types (e.g. DATE) can be defined by mapping the appropriate data types into standard PDOs.

7.3 The Emergency Object

7.3.1 Emergency Object Usage

Emergency messages are triggered by the occurrence of a device internal fatal error situation and are transmitted from the concerned application device to the other devices with highest priority. This makes them suitable for interrupt type error alerts[13].

By means of CANopen Communication Profile defined emergency error codes (Table 7-1), the error register (Table 10-14) and device specific additional information, specified in the device profile the emergency condition is specified.

Error Code (hex)	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
90xx	External Error
F0xx	Additional Functions
FFxx	Device specific

Table 7-1: Emergency Error Code

[13] An Emergency Telegram may be sent only once per 'error event', i.e. the emergency messages must not be repeated. As long as no new errors occur on a device no further emergency messages must be sent.

The emergency object is optional. If a device supports the emergency object, it has to support at least the two error codes 00xx and 10xx. All other error codes are optional.

A device may be in one of two emergency states (Figure 7-1). Dependent on the transitions emergency objects will be transmitted.

1. After initialisation the device enters the error free state if no error is detected. No error message is sent.
2. The device detects an internal error indicated in the first three bytes of the emergency message (error code and error register). The device enters the error state. An emergency object with the appropriate error code and error register is transmitted[14]. The error code is filled in at the location of object 1003H (pre-defined error field).
3. One, but not all error reasons are gone. An emergency message containing error code 0000 (Error reset) may be transmitted together with the remaining errors in the error register and in the manufacturer specific error field.
4. A new error occurs on the device. The device remains in error state and transmits an emergency object with the appropriate error code. The new error code is filled in at the top of the array of error codes. If one error is repaired the appropriate error code is erased from the array. It has to be guaranteed that the error codes are sorted in a timely manner (oldest error - highest sub-index, see Object 1003H).
5. All errors are repaired. The device enters the error free state and transmits an emergency object with the error code ‘reset error / no error'.

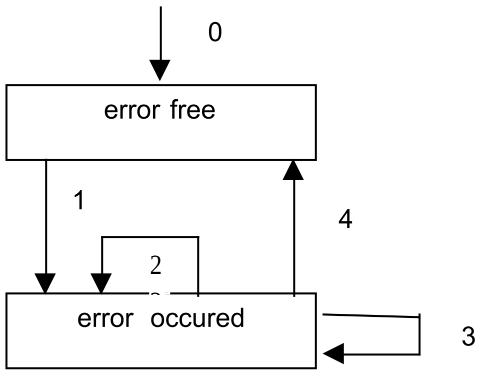


Figure 7-1: Emergency State Transition Diagram

7.3.2 Emergency Object Mapping

The Emergency Telegram consists of 8 bytes with the mapping as shown in Figure 7-2.

Byte	0	1	2	3	4	5	6	7
Content	Emergency Error Code (see Table 7-1)		Error register (Object 1001H)	Manufacturer specific Error Field				

Figure 7-2: Mapping Emergency Object

[14] If the error is indicated only in the manufacturer specific part (last five bytes) of the emergency message the emergency message may be transmitted

7.3.3 Emergency Object Services

The Emergency object is implemented as a CMS object of type Stored Event with the notifying device acting as server of the event.

According to CMS the following services on Stored Events are relevant:

- Define Event
- Notify Event
- Read Event

The following CMS event object attributes are specified for the Emergency object:

• name:	according to the CMS naming conventions with <application-specific-name> = "EMCY000"
• user type:	notifying device: server other devices: clients
• class:	Stored Event
• priority:	0 or 1 suggested
• data type:	STRUCTURE OF UNSIGNED(16) emergency_error_code, UNSIGNED(8) error_register, ARRAY (5) of UNSIGNED(8) manufacturer_specific_error_field
• inhibit time:	application specific

7.3.4 Protocols

See specification of the Stored Event protocols /7/.

8 Network Management and Identifier Distribution

8.1 Services and Protocols

NMT services (/9/, /10/) provide a means for identification and monitoring of all nodes in the network. NMT requires one device in the network which fulfils the function of the NMT Master. NMT offers the following functionality groups:

- Module Control Services for initialisation of NMT Slaves that want to take part in the distributed application
-
- Error Control Services for supervision of the nodes and networks communication status
-
- Configuration Control Services for up- and downloading of configuration data from respectively to a module of the network.

A NMT Slave represents that part of a node which is responsible for the node’s NMT functionality. A NMT Slave is uniquely identified by its Module ID or Name. Module ID (node identification number) and Module Name are configurable by LMT (/14/, /15/) services and protocols or other local means (e.g. DIP-Switch).

The NMT Master can identify NMT Slaves, setting up of NMT parameters, download configuration data if necessary, allow the DBT Slave to request COB-IDs via DBT services and protocols (/11/, /12/) and enable or disable the CMS services at a certain node or simultaneously at all nodes.

Error Control services are achieved principally through periodically transmitting of life guarding messages by the NMT Master. If a NMT Slave doesn’t respond within a defined span of time or if the NMT Slave’s communication status has changed, the NMT Master informs its NMT Master Application about that event. Also if a NMT Slave is not guarded within the node’s life time, the NMT Slave informs its local Application about that event.

The node initialisation process is controlled by a NMT Master Application during the network boot-up process via NMT services according to /9/.

In addition to the node states of the state diagram specified in /9/, the CANopen Communication Profile introduces the two status's "Pre-Operational" and "initialising" according to Figure 8-1. In Table 8-1 the state transitions of the NMT Slave state diagram according to the CAL specification are described in detail, in Table 8-2 the additional, profile specific transitions are described in detail.

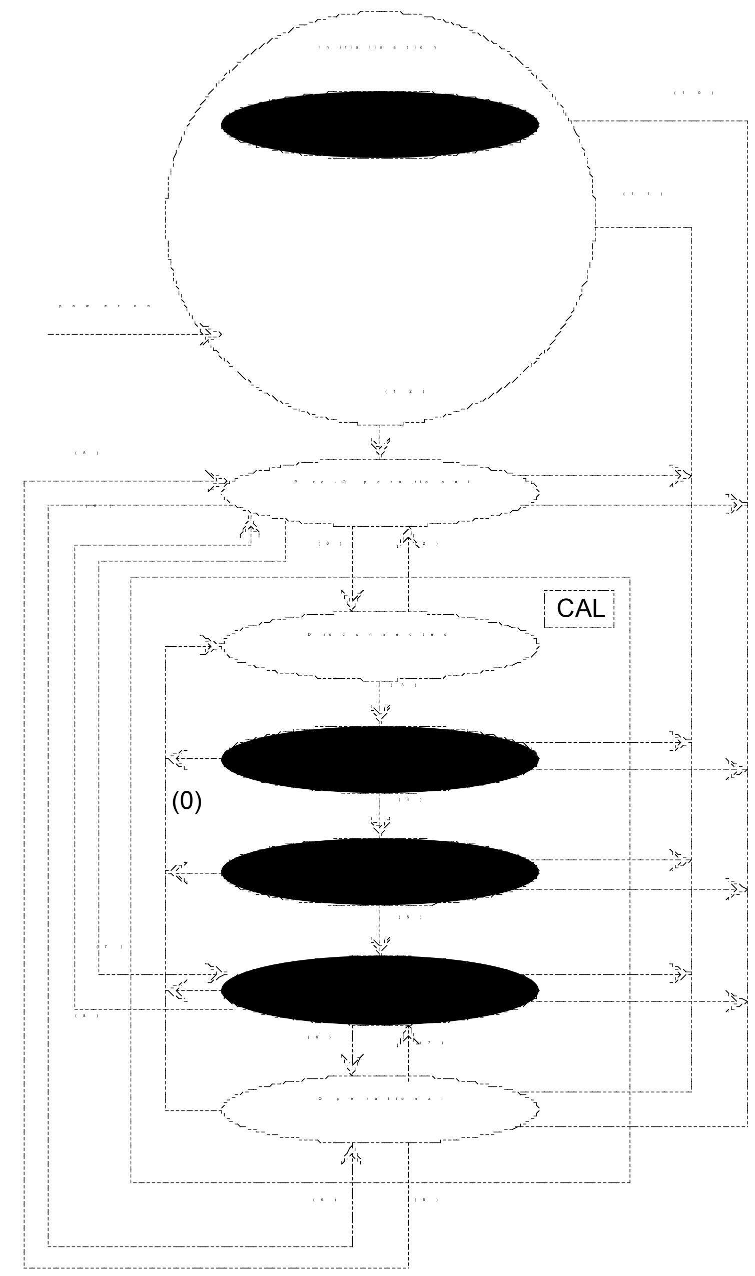


Figure 8-1: Extended NMT Node State Diagram

(0)	Disconnect_Remote_Node indication
	Disconnect_Node request
	Error response
(2)	Delete_Node request (local service)
(3)	Connect_Node request
(4)	Connect_Remote_Node response
(5)	Prepare_Remote_Node response
(6)	Start_Remote_Node indication
(7)	Stop_Remote_Node indication
(8)	Enter_Pre-Operational_State indication
(10)	Reset_Node indication
(11)	Reset_Communication indication
(12)	Initialisation finished - enter Pre-Operational automatically

NMT Master	Direction of telegram	NMT Slave
		(0) Disconnect_Node request NMT local service request to set node state to DISCONNECTED
(0) Disconnect_Remote_Node request NMT service request to set the state of the remote node object and remote node to DISCONNECTED	→	**(0) Disconnect_Remote_Node indication** NMT service indication to set node state to DISCONNECTED
(0) Error confirmation NMT service confirmation informing NMT Master Application of an error (in response to Connect_Remote_Node or Prepare_Remote_Node request)	←	**(0) Error response** NMT service response forcing node to set its state to DISCONNECTED in response to Connect_Remote_Node or Prepare_Remote_Node indication from NMT Master
(1) Add_Remote_Node request NMT local service request to create remote node object		**(1) Create_Node request** NMT local service request to create node object
(2) Remove_Remote_Node request NMT local service request to delete remote node object		**(2) Delete_Node request** NMT local service request to delete node object
		(3) Connect_Node request NMT local service request to set node state to CONNECTING
(3) Connect_Remote_Node confirmation Confirmation to the NMT Master Application that node state is CONNECTED (in response to a NMT Master Connect_Remote_Node request)	←	**(4) Connect_Remote_Node response** NMT service response confirming node state as CONNECTING in response to a NMT Master Connect_Remote_Node indication. After successful transmission of the response the NMT slave enters the node state PREPARING
(4) Prepare_Remote_Node confirmation NMT service confirmation informing NMT Master Application about successful remote node preparing	←	**(5) Prepare_Remote_Node response** NMT service response confirming the end of node preparing in response to NMT Master Prepare_Remote_Node indication
(5) Start_Remote_Node request NMT service request to enable communication via PDOs and SDOs	→	**(6) Start_Remote_Node indication** NMT service indication informing node that communication via PDOs and SDOs is enabled
(6) Stop_Remote_Node request NMT service request to disable communication	→	**(7) Stop_Remote_Node indication** NMT service indication informing node to disable communication

Table 8-1: State Transitions of the extended NMT Slave State Diagram according to the CAL specification

The right hand part of Table 8-1 describes state transitions of NMT Slaves corresponding to the state diagram of Figure 8-1.

NMT Master	Direction of telegram	NMT Slave
(8) Enter_Pre-Operational_State request NMT service allowing NMT Master to enable SDOs	→	**(8) Enter_Pre-Operational_State indication** NMT service informing NMT Slave that SDOs are enabled
(9) Prepare_Remote_Node request NMT service requesting NMT Slave to request COB-Identifiers for not allocated PDOs	→	**(9) Prepare_Remote_Node indication** NMT service informing NMT Slave to request COB-IDs for not allocated PDOs
(10) Reset_Node request NMT service allowing NMT Master to reset node applications	→	**(10) Reset_Node indication** NMT service informing the NMT Slave(s) to reset local application
(11) Reset_Communication request NMT service allowing NMT Master to reset communication parameters at the slave node(s)	→	**(11) Reset_Communication indication** NMT service informing the NMT Slave to reset the communication parameters in the object dictionary to default values
		(12) Initialisation finished The NMT Slave enters this state automatically after finishing the initialisation tasks. This means that every reset request leads to the state Pre-Operational.
		(13, 14) Application or Communication Reset performed These state transitions are performed automatically.

Table 8-2: Profile Specific State Transitions of the extended NMT Slave State Diagram

After its initialisation, the device enters the „Pre-Operational“ state autonomously. In this state, only communication via SDOs is operational, using the default COB-IDs derived from the module-ID. Configuration of PDOs, device parameters and also the allocation of application objects (PDO-mapping) may be performed by a configuration application.

Then the extended boot-up is started by disconnecting the node (Disconnect_Node request). Alternatively, the node may be switched into the operational stage directly by sending a Start_Remote_Node request (Minimum Boot-Up).

If a node is guarded via the node guarding protocol the state transferred from the slave back to the NMT Master should be 127 (see /10/) if the NMT Slave is in Pre-Operational state.

With the introduction of the "initialisation" state a complete or a partial reset of a node is possible via the network. The „initialisation“ state is divided into three sub-states (see Figure 8-2).

1. Reset_Application: In this state the parameters of the manufacturer specific profile area and in the standardised device profile area are set to their default values. After setting of the power-on values the state Reset_Communication is entered.
2. Reset_Communication: In this state the parameters of the communication profile area are set to their power-on values. After this the state Initialising is entered.
3. Initialising: This is the first sub-state the device enters after power-on. After performing the basic node initialisation in this state the state Pre-Operational is entered automatically.

Power-on values are the last stored parameters. If storing is not supported or has not been executed, the power-on values are the default values according to the CANopen communication and device profiles.

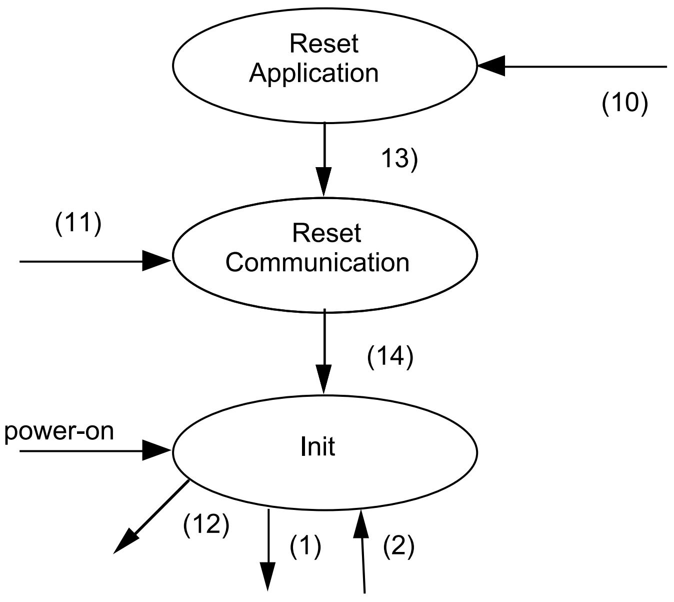


Figure 8-2: Sub-states of the initialisation state

(1)	Create_Node request
(2)	Delete Node request
(10)	Reset_Node indication
(11)	Reset_Communication indication
(12)	Initialisation finished - enter Pre-Operational automatically
(13)	Application Reset performed
(14)	Communication Reset performed

To control the different state transitions three additional services are introduced.

8.1.1 Enter_Pre-Operational_State Service and Protocol

With this service a NMT Slave may be forced into the Pre-Operational state.

Parameter	Request/Indication
Argument	**mandatory**
Node-ID	selection
All	selection

The service will only be executed for the selected remote node objects whose state is "prepared" or "operational". Through this service the NMT Master sets the state of the selected NMT Slave(s) from "prepared" or "operational" to "Pre-Operational". In this state a node can communicate only via its SDOs. Therefore the service may also be used to switch off PDOs with a transition from "operational" to "Pre-Operational" state.

The service is unconfirmed and mandatory.

In Figure 8-3 the protocol of the Enter_Pre-Operational_State service is shown[15].

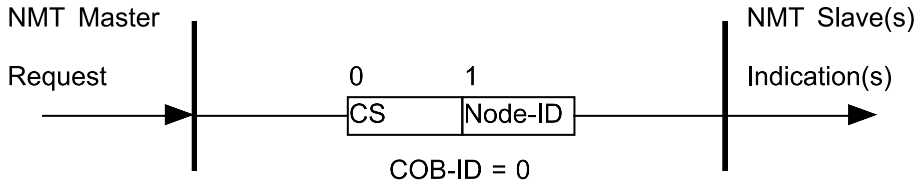


Figure 8-3: Enter_Pre-Operational_State, Reset_Node and Reset_Communication protocols

- CS = 128: Enter_Pre-Operational_State Service
  CS = 129: Reset_Node Service
  CS = 130: Reset_Communication Service
- Node ID: Node-ID of the NMT Slave as assigned by the NMT Master in the Node connect protocol or zero. If zero, the protocol addresses all NMT Slaves.

[15] Due to the introduction of the new services the reservation of command specifiers of value 128, 129 and 130 is necessary in the CAL Standard.

8.1.2 Reset_Node Service and Protocol

With this service the NMT Master may force a complete reset of a distinct or of all nodes.

Parameter	Request/Indication
Argument	**mandatory**
Node-ID	selection
All	selection

The service will only be executed for the selected remote node objects independent of the state of the object. Through this service the NMT Master sets the state of the selected NMT Slave(s) from any state except DISCONNECTED to the "reset application" state. In this state a reset of the node application will be performed. The parameters of the entire object dictionary are set to their power-on values.

The service is unconfirmed and mandatory for all devices.

In Figure 8-3 the protocol of the Reset_Node-service is shown.

8.1.3 Reset_Communication Service and Protocol

With this service the NMT Master may force a reset of the communication parameters of a distinct or of all nodes.

Parameter	Request/Indication
Argument	**mandatory**
Node-ID	selection
All	selection

The service will only be executed for the selected remote node objects independent of the state of the object. Through this service the NMT Master sets the state of the selected NMT Slave(s) from any state except DISCONNECTED to the "reset communication" state. In this state the parameters of the communication area in the object dictionary are set to their power-on values.

The service is unconfirmed and mandatory for all devices.

In Figure 8-3 the protocol of the Reset_Communication service is shown.

8.2 Network Initialisation Process (Bootup-Process)

In Figure 8-4 the general flow chart of the extended network initialisation process, controlled by a NMT Master Application or Configuration Application is shown.

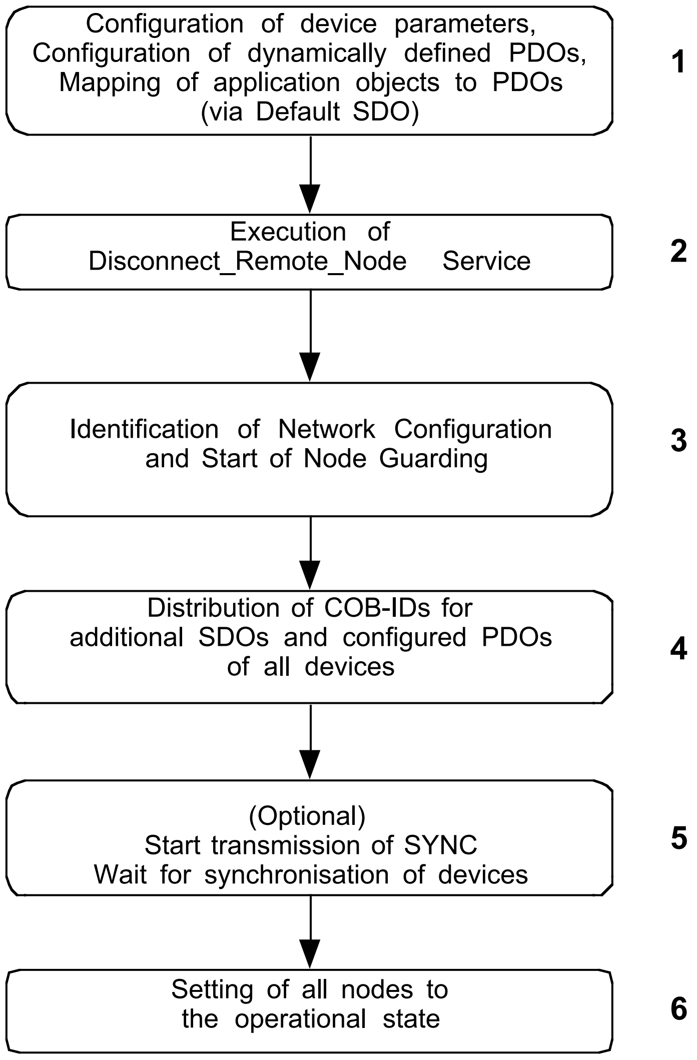

Configuration of device parameters,
Configuration of dynamically defined PDOs,
Mapping of application objects to PDOs
(via Default SDO)
1
Execution of
Disconnect_Remote_Node Service
2
Identification of Network Configuration
and Start of Node Guarding
3
Distribution of COB-IDs for
additional SDOs and configured PDOs
of all devices
4
(Optional)
Start transmission of SYNC
Wait for synchronisation of devices
5
Setting of all nodes to
the operational state
6

Figure 8-4: Flow Chart of the Extended Network Initialisation Process

In step 1 the devices are in the node state Pre-Operational which is entered automatically after power-on. In this state the devices are accessible via their Default-SDO using identifiers that have been assigned according to the Predefined Connection Set. In this step the configuration of device parameters takes place on all nodes which support parameter configuration.

This is done from a Configuration Application (which can reside on the same node as the NMT Master Application or from a Configuration Tool via the Default-SDO. For devices that support these features the selection and/or configuration of PDOs, the mapping of application objects (PDO mapping), the configuration of additional SDOs and optionally the setting of COB-IDs may be performed via the Default-SDO objects.

In step 2 the Disconnect_Remote_Node service is executed. By receiving the service indication the devices enter the node state Disconnected. After entering this state (and optionally performing an additional local initialisation depending on the configured parameters) the devices have to execute the local Connect_Node service for entering the node state Connecting.

During step 3 the NMT Master Application performs the identification of all nodes in the network by repeated execution of the Connect_Remote_Node service. Within this service also the node guarding parameters are negotiated between NMT Master and NMT Slaves and node guarding is started.

In step 4 the assignment of COB-IDs to the configured PDOs and additional SDOs is performed by the DBT (if COB-ID distribution shall be performed). The COB-IDs for the default-SDOs according to the Predefined Connection Set should not be changed.

Step 5 is an optional step. It can be used to ensure that all nodes are synchronised by the SYNC object before entering the node state Operational in step 6. The synchronisation is done by starting the cyclic transmission of the SYNC object and waiting a sufficient span of time to allow all nodes to synchronise.

With step 6 all nodes are enabled to communicate via their PDO objects.

If the extended network initialisation process is not supported by a device (e.g. Minimum Capability Device, see 8.3) or if e.g. the devices have already been configured completely in a previous extended boot-up and have stored their configuration, the minimum boot-up shown in Figure 8-5 can be applied.

The minimum boot-up has to be supported by all CANopen devices.

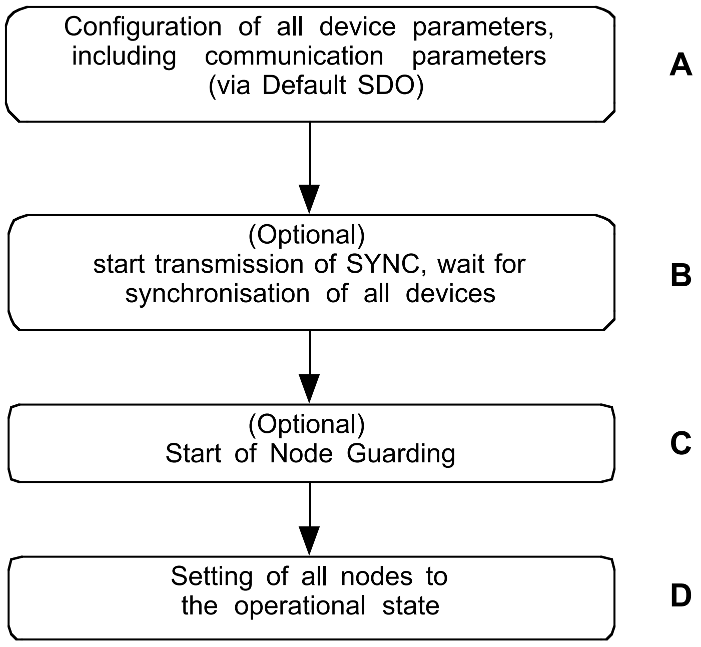

Configuration of all device parameters, including communication parameters (via Default SDO)
A
(Optional) start transmission of SYNC, wait for synchronisation of all devices
B
(Optional) Start of Node Guarding
C
Setting of all nodes to the operational state
D

Figure 8-5: Flow Chart of the Minimum Network Initialisation Process

Step A corresponds to step 1 of the extended network initialisation process.

Step B corresponds to step 5 of the extended network initialisation process.

In Step C Node guarding can be activated (if supported) using the guarding parameters configured in step A.

With step D all nodes are enabled to communicate via their PDO objects.

8.3 Minimum Capability Device

To enable devices which do not support the complete DBT- and NMT Slave functionality to co-operate with full capability devices, the following minimal capabilities are required:

- Module-ID,
- Object dictionary, depending on the device functionality,
- One SDO, supporting the mandatory entries (read only)
- Support of the following services as NMT Slave: Reset_Node, Enter_Pre-Operational_State, Start_Remote_Node, Stop_Remote_Node, Reset_Communication
- Default profile ID-allocation scheme

In Figure 8-6 the state diagram of a node with minimum capability is shown. Minimum capability devices enter the Pre-Operational state directly after finishing the device initialisation. During this state device parameterisation and ID-allocation via SDO (e.g. using a configuration tool) is possible. Then the nodes can be switched directly into the Operational state. By switching a device into the Prepared[16] state it is forced to stop the communication altogether (except node guarding, if active). Furthermore, this state can be used to achieve certain application behaviour. The definition of this behaviour falls into the scope of device profiles.

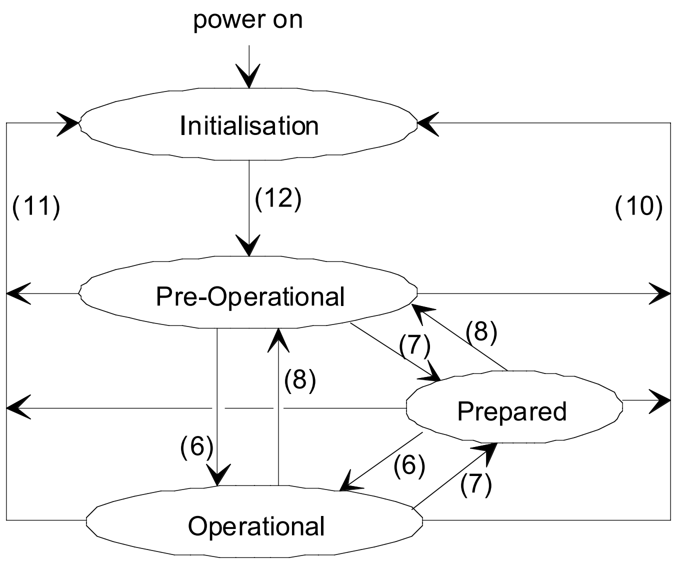


Figure 8-6: State Diagram of a Minimum Capability Device

(6)	Start_Remote_Node indication
(7)	Stop_Remote_Node indication
(8)	Enter_Pre-Operational_State indication
(10)	Reset_Node indication
(11)	Reset_Communication indication
(12)	Initialisation finished - enter Pre-Operational automatically

[16] Note that the NMT master does not have to use the Prepared state during boot-up. Minimal Boot-Up consists of one CAN message: a broadcast Start_Remote_Node indication.

8.4 Allocation of COB-ID's and Inhibit-Times

As CAN is a communication object (COB-) based network where each communication object has an associated identifier which specifies its priority implicitly, the allocation of identifiers to the COB's is an essential issue in the system design. Inhibit-times are defined by CMS to prevent high priority messages to flood the bus to such an extent, that lower priority messages are denied bus access at all times. The inhibit-time specifies the time period during which a certain COB must not be transmitted again.

The communication object identifiers (COB-ID's) and inhibit-times may be distributed to the devices either statically or dynamically.

Static distribution means that the identifiers and inhibit-times are fixed by the module suppliers and may be changed by the system integrator through module specific means such as setting dip-switches, adapting firmware, etc.

Dynamic distribution means that the identifiers and inhibit-times are distributed via the CAN network through standard DBT services and protocols or via SDO. The dynamic distribution is the preferred method in the CANopen Communication Profile.

Some identifiers (1-7, 1740-1760dec) have been reserved by CANopen for future enhancements of the communication profile. They may still be used in applications employing dynamic distribution on all devices, but it is not recommended to use them.

8.4.1 Predefined Connection Set

In order to reduce configuration effort for simple networks a mandatory default identifier allocation scheme is defined. These identifiers are available in the Pre-Operational state directly after initialisation (if no modifications have been stored) and may be modified by means of dynamic distribution. A device has to provide the corresponding identifiers only for the supported communication objects.

The default profile ID-allocation scheme (Table 8-3 + 8-4) consists of a functional part, which determines the object priority and a module-ID-part, which allows to distinguish between devices of the same functionality. The ID-allocation scheme corresponds to a pre-defined master/slave connection set and allows a peer-to-peer communication between a single master device and up to 127 slave devices. It also supports the broadcasting of non-confirmed NMT-services, SYNC- and Time-Stamp-objects and node guarding. Broadcasting is indicated by a module-Id of zero.

The pre-defined master/slave connection set supports one emergency object, one SDO, at maximum 2 Receive-PDOs and 2 Transmit-PDOs and the node guarding object.

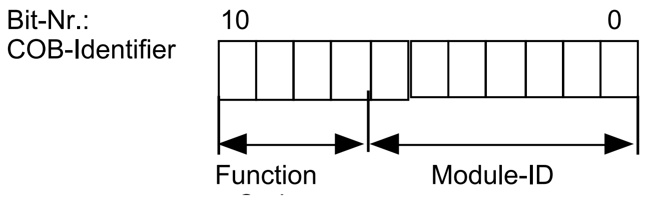


Figure 8-7: Identifier allocation scheme for the pre-defined master/slave connection set

Table 8-3 and Table 8-4 show the supported objects and their allocated COB-IDs.

object	function code (binary)	resulting COB-ID	Communication Parameters at Index	CMS priority group
NMT	0000	0	-	0
SYNC	0001	128	(1005h)	0
TIME STAMP	0010	256	-	1

Table 8-3: Broadcast Objects of the Pre-defined Master/Slave Connection Set

object	function code (binary)	resulting COB-IDs	Communication Parameters at Index	CMS priority group
EMERGENCY	0001	129 - 255	-	0 , 1
PDO1 (tx)	0011	385 - 511	1800h	1 , 2
PDO1 (rx)	0100	513 - 639	1400h	2
PDO 2 (tx)	0101	641 - 767	1801h	2 , 3
PDO2 (rx)	0110	769 - 895	1401h	3 , 4
SDO (tx)	1011	1409 - 1535		6
SDO (rx)	1100	1537 - 1663		6 , 7
Nodeguard	1110	1793-1919	(100Eh)	-

Table 8-4: Peer-to-Peer Objects of the Pre-defined Master/Slave Connection Set[17]

If devices with and without DBT capability shall operate in the same network, it is necessary to reserve the corresponding default-COB-IDs in the DBT data base by predefinitions during the boot-up process. If the default ID-allocation is overwritten by a configuration tool, the corresponding predefinitions should be deleted in the COB data base.

8.4.2 Dynamic Distribution

CAN Application Layer (CAL) uses a CMS priority level model (see /11/). It describes 8 priority levels with 220 identifiers each and comprises the identifiers 1 to 1760. The remaining identifiers (0, 1761-2031) are reserved for network control by NMT, DBT and LMT.

The first step in the system design is to allocate the appropriate priority levels to the various communication objects. As CANopen devices feature a default identifier allocation for a basic set of communication objects, one can start from there and modify the allocation if necessary using either SDO services or DBT (if the extended boot-up is supported).

In order to guarantee a minimum jitter the synchronisation message (SYNC) has the highest priority of all messages that occur in normal operation.

For transmitting emergency messages each node in the network must be able to use messages with the highest available priority. For these exceptional messages a number of identifiers in priority level 0 and 1 have been reserved.

The time stamp messages have been allocated to priority level 1, with lower priority than the emergency messages . The remaining identifiers above the PDOs are reserved for other synchronisation purposes.

[17] The table has to be seen from the devices point of view.

The CANopen Communication Profile describes two different PDO modes (see chapter 5). One is used for synchronous data exchange and the other one for asynchronous message transfer. To guarantee the cyclic behaviour of the system data exchange a higher priority (suggested: level 2) is allocated to the synchronous PDO's than to the asynchronous PDO's (level 3 - 5). Priority level 6 - 7 has been suggested for the SDO's.

A special feature of the Network Management (NMT) is the life guarding/node guarding capability. For that purpose a range of 255 identifiers have been reserved starting at 1761.

CMS priority level	message / object
0 (Id 1 - 220)	- Synchronisation - Emergency (fault)
1 (Id 221 - 440)	- Emergency (fault) - Network timing (SYNC, time stamp) - other high priority sync. Messages
2 (Id 441 - 660)	- PDO (synchronous)
3 (Id 661 - 880) 4 (Id 881 - 1100) 5 (Id 1101 - 1320)	- PDO (asynchronous)
6 (Id 1321 - 1540)	- SDO's
7 (Id 1541 - 1760)	reserved for SDO's

Table 8-5: Suggested Message Priorities

8.4.3 Naming Conventions

CMS Naming Conventions

The CMS Naming Conventions of the CAN Application Layer define a syntax to create the names of CMS objects and COB's. The dynamic distribution of COB-ID's is based on CMS object names.

CMS Object Name Syntax: <prof-id> <appl-spec-name> <node-ID>

<prof-id>:

3 numeric characters for a device profile number registered by the CiA[18].

<appl-spec-name>:
The first four (1 - 4) characters should identify what part of the CANopen Communication Profile is modelled by this particular CMS object.

Abbreviation	part of communication profile
SYNC	SYNC message
TIME	time stamp message
EMCY	emergency message
SDO_	service data object (SDO)
TPDO or RPDO	process data object (PDO)

The remaining three characters (5 - 7) should be interpreted as three ASCII-numbers.

<node-ID> :

3 numerical characters represent the node-ID of the module where the CMS object is used. The node-ID is defined by the NMT.

The CMS names can be automatically generated by using these naming conventions. This is useful if the number of used PDO's is not fixed. Names will only be generated for the needed PDO's.

[18] For instance '401' is used for the I/O modules' profile.

9 Error Handling

All types of errors detectable by the devices in an industrial system can be grouped into the five classes shown below:

- Bus errors - usually electrical and wiring faults.
- Communications or protocol errors - for example, a specific bit in a telegram is not set as expected.
- Life guarding errors - life guarding telegrams have not been received by a device within a specified time-out period.
- Applications errors - such as accessing a device which is not available.
- Device failures - such as hardware failures on a device.

Bus Errors - Low level communications errors are catered for implicitly by the CAN specification. Only under complete bus failure would a node be aware of such errors. The error response then would be to go 'bus-off' and attempt to achieve a 'safe' hardware state. The NMT master is aware of the devices active on the bus via the node guarding mechanism.

Protocol Errors - Such errors can occur for example during domain downloads. A typical response to this kind of error would be to signal failure via the appropriate protocol (e.g. initiating the abort of a domain transfer) and ignoring the contents of the partially received information. The device would still maintain its current state of readiness i.e. remain operational.

Life guarding Errors - are dependent upon what type of life guarding is supported by a device. If the device possesses full NMT life guarding the error mechanism depends on whether the NMT Master detects a time out receiving a response from the device to a life guarding telegram or whether the module itself detects a time out because no life guarding telegram has been received from the NMT Master within its defined time out period. In the first case, it should be tried to shut-down all devices in the network and then perform a warm-start. In the second case, the device would assume that the NMT Master is not operational and must indicate this to its application[19].

Application Errors - Such errors could occur, for example, in a Programmable Logic Controller (PLC) if an application program tried to access an output signal which did not exist. A typical response would be to indicate to the application that an error had occurred and send an emergency telegram.

Device Failures - Such errors could occur, for example, in a Drive Unit when the temperature of power components exceeds a critical limit. To allow the response to such cases to be application dependent such an error would result in the transmission of an emergency telegram. A time-out period would be initiated and if the device did not get any response within the device dependent time limit the device would take its own safe action e.g. put all outputs into a default 'safe' state.

[19] The definition of device reactions to life guarding errors falls in the scope of the device profile.

9.1 Node Guarding / Life Guarding

The communication interface of a particular device is assumed to be in a certain state in order to be able to process these messages correctly. However there could occur local events on a device that force it to a different state than was assumed by the transmitter of a message. Especially, it could happen that due to an error a particular device goes to the *Disconnected State* and does no longer take part on the bus communication.

To detect these errors with devices that do not transmit PDO's regularly, the NMT Master manages a network database where, besides other information, the expected states of all connected devices are recorded. The NMT Master regularly retrieves the actual states of all devices on the network and compares them to the states recorded in the network database. Mismatches are indicated first locally on the NMT Master through the *Network Event* service. Consequently the application must take appropriate actions to ensure that all devices on the bus will go to a save state. Generally, the application will first force all devices to the *Disconnected State* and then perform a new boot-up sequence.

Node Guarding is optional. It starts for the slave when the first remote-transmit-request for its guarding identifier is received. This may be during the node-connect phase (extended boot-up) or later. The slave uses the guard time and life-time factor from its object dictionary (either default values or modified at any stage). If guard time and life time factor are zero (default values), the NMT Slave does not guard the NMT Master (lifeguarding).

If the extended boot-up is used, it is recommended that the NMT Master distributes node-guarding identifiers starting from 1793dec, as this enables him to include very simple devices in the network without readjustments.

With devices which are configured to use a PDO on a cyclic basis the application may be able to detect fatal errors by means of missing telegrams. These devices do not have to support guarding services. Whether or not a device needs to be guarded can be indicated to the NMT Master during the boot-up phase of the network.

9.2 Emergency Telegram

The Life Guarding mechanism provides a means for detecting errors in the network interfaces of devices. However, it cannot detect failures within the device itself. For instance, the temperature of some power components on a device could exceed a critical limit whereas the network interface could still be in good working condition. For notification of this type of errors emergency telegrams have been introduced (see chapter 7.3).

10 Dictionary Structure and Entries

This section details the Object Dictionary structure and entries which are common to all devices. The format of the object dictionary entries is shown below:

Index (hex)	**Object (Symbolic Name)**	**Name**	**Type**	**Attrib.**	**M/ O**

Table 10-1: Format of Object Dictionary Headings

The complete object dictionary consists of the six columns shown above. The **Index** column denotes the objects position within the object dictionary. This acts as a kind of address to reference the desired data field. The sub-index is not specified here. The sub-index is used to reference data fields within a complex object such as an array or record.

The **Object** column contains the Object Name according to the table below and is used to denote what kind of object is at that particular index within the object dictionary. The following definitions are used:

Object Name	Comments	Object Code
NULL	A dictionary entry with no data fields	0
DOMAIN	Large variable amount of data e.g. executable program code	2
DEFTYPE	Denotes a type definition such as a Boolean, unsigned16, float and so on	5
DEFSTRUCT	Defines a new record type e.g. the PDOMapping structure at 21 Hex	6
VAR	A single value such as an unsigned8, Boolean, float, integer16, visible string etc.	7
ARRAY	A multiple data field object where each data field is a simple variable of the SAME basic data type e.g. array of unsigned16 etc.	8
RECORD	A multiple data field object where the data fields may be any combination of simple variables	9

Table 10-2: Object Dictionary Object Definitions

The **name** column provides a simple textual description of the function of that particular object. The **type** column gives information as to the type of the object [20]. These include the following pre-defined types: Boolean, floating point number, Unsigned Integer, Signed Integer, visible/octett string, date, time-of-day, time-difference and domain (see /8/). It also includes the pre-defined complex data type PDOMapping and may also include others which are either manufacturer or device specific. The **Attribute** column defines the access rights for a particular object.

[20] It is not possible to define records of records, arrays of records or records with arrays as fields of that record. In the case where an object is an array or a record the sub-index is used to reference one data field within the object. Note that sub-index 0 contains the number of elements and therefore itself is not part of the array.

It can be of the following:

Attribute	Description
rw	read and write access
wo	write only access
ro	read only access
const	read only access, value is constant

Table 10-3: Access Attributes for Data Objects

The **M/O** column defines whether the object is **M**andatory or **O**ptional. A mandatory object must be implemented on a device. An optional object need not be implemented on a device.

10.1 DictionaryComponents

The overall layout of the object dictionary is shown in Table 4-1.

Index 01h-1Fh contain the standard data types, index 20h - 22h contain predefined complex data types. The range of indices from 23-3FH is not defined yet but reserved for future standard data structures.

The range of indices from 40-5FH is free for manufacturers to define custom data types. The range 60-0FFFH is reserved for possible future enhancements of the CANopen Communication Profile. The range 1000H-1FFFH contains the communication specific object dictionary entries defined by the CANopen Communication Profile.

These parameters are called *communication entries*, their specification is common to all device types, regardless of the device profile they use. The objects in range 1000H-1FFFH not specified by this communication profile are reserved for further use. The range 2000H-5FFFH is free for manufacturer specific profile definition.

The range 6000H-FFFH contains the standardised device profile parameters.

10.1.1 Data Types

The static data types are placed in the object dictionary for definition purposes only. However, indices in the range 1-7 can be mapped as well in order to define the appropriate space in the PDO as not being used by this device (don’t care). An example mapping can be found in the appendix.

The order of the data types is as follows:

Index	Object	Name
0001	DEFTYPE	Boolean
0002	DEFTYPE	Integer8
0003	DEFTYPE	Integer16
0004	DEFTYPE	Integer32
0005	DEFTYPE	Unsigned8
0006	DEFTYPE	Unsigned16
0007	DEFTYPE	Unsigned32
0008	DEFTYPE	Floating Point (Float)
0009	DEFTYPE	Visible String
000A	DEFTYPE	Octet String
000B	DEFTYPE	Date
000C	DEFTYPE	Time Of Day
000D	DEFTYPE	Time Difference
000E	DEFTYPE	Bit String
000F	DEFTYPE	Domain
0010-001F	Null Objects	(reserved)
0020	DEFSTRUCT	PDO CommPar
0021	DEFSTRUCT	PDO Mapping
0022	DEFSTRUCT	SDO Parameter
0023-003F	Null Objects	(reserved)
0040-005F	DEFTYPE	Manufacturer Specific Data Types
0060-007F	DEFTYPE	Device Profile Specific Standard Data Types
0080-009F	DEFSTRUCT	Device Profile Specific Complex Data Types

Table 10-4: Object Dictionary Data Types

The data type representations used are detailed in /8/ . Every device does not need to support all the defined data types. A device only has to support the data-types it uses with the objects in the range 1000H-9FFFH.

The predefined complex data-types are placed after the standard data-types. Three types are defined at present, the ***PDO Communication Parameter (PDO CommPar)*** the ***PDO Mapping*** record and the **SDO Parameter record (SDO_Par)**. They are placed at index 20H, 21H and 22H.

A device may optionally provide the length of the standard data types encoded as Unsigned32 at read access to the index that refers to the data type. E.g. index 000Bh (Date) contains the value 38h=56dec as the data type „Date“ is encoded using 56 bits (see /8/). If the length is variable (e.g. 000Fh = Domain), the entry contains 0h.

For the supported complex data types a device may optionally provide the structure of that data type at read access to the corresponding index. Sub-index 0 then provides the number of entries at this index (not counting sub-indices 0 and 255), and the following sub-indices contain the data type according to table 10-4 encoded as Unsigned8. The entry at Index 20h describing the structure of the PDO Communication Parameter then looks as follows (see also 10.1.2):

Subindex	Value	(Description)
0h	04h	(4 subindices follow)
1h	07h	(Unsigned32)
2h	05h	(Unsigned8)
3h	06h	(Unsigned16)
4h	05h	(Unsigned8)

Standard (simple) and complex manufacturer specific data types can be distinguished by attempting to read sub-index 1h: At a complex data type the device returns a value and sub-index 0h contains the number of sub-indices that follow, at a standard data type the device aborts the domain transfer as no sub-index 1h available. In that case sub-index 0h contains the length of the simple data type in bits.

Note that some entries of data type Unsigned32 have the character of a structure (e.g. PDO COB-ID entry, see Figure 10-1).

If an object dictionary entry (index) contains several sub-indices, then sub-index 0 describes the number of entries (sub-indices) that follow (not counting sub-indices 0 and FFh). The number of entries is encoded as Unsigned8.

Sub-index FFh (255dec) describes the structure of the entry by providing the data type and the object type of the entry. It is encoded as Unsigned32 and organised as follows:

	MSB		LSB
bits	31-16	15-8	7-0
value	reserved (value: 00 00h)	Data Type (see Table 10-4)	Object (see Table 10-2)
encoded as	-	Unsigned8	Unsigned8

It is optional to support Sub-Index FFh. If it is supported throughout the object dictionary and the structure of the complex data tapes is provided as well, it enables one to upload the entire structure of the object dictionary.

As Sub-index has the same structure and meaning throughout the object dictionary, it is not described at each detailed object description (Chapter 10.3).

10.1.2 PDO Communication Parameter

The format of the structure is as follows:

Index	Sub-Index	Field In PDO Communication Record	Data Type
0020H	0H	number of supported entries in the record	Unsigned8
	1H	COB-ID used by PDO	Unsigned32
	2H	transmission type	Unsigned8
	3H	inhibit time	Unsigned16
	4H	CMS priority group	Unsigned8

Table 10-5: PDO Communication Parameter Record

If the device supports the identifier distribution via DBT the value on sub-index 0H is 4 otherwise it is 2 (inhibit time not supported) or 3. The COB-ID at Index 20H, Sub-Index 1H is defined using data type Unsigned32 in order to cater for 11-bit CAN Identifiers (CAN 2.0A) as well as for 29-bit CAN identifiers (CAN 2.0B). The entry has to be interpreted as defined in Figure 10-1 and Table 10-6.

Unsigned32

MSB LSB

bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0/1	1	29-bit Identifier	

Figure 10-1: Structure of PDO COB-ID entry

bit number	value	meaning
31 (MSB)	0	PDO valid
	1	PDO not valid
30	0	RTR allowed on this PDO
	1	no RTR allowed on this PDO
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28 - 11	0	if bit 29=0
	X	if bit 29=1: bits 28-11 of 29-bit-COB-ID
10-0 (LSB)	X	bits 10-0 of COB-ID

Table 10-6: Description of PDO COB-ID entry

The PDO valid/not valid allows to select which PDOs are used in the operational stage. There may be PDOs fully configured (e.g. by default) but not used, and therefore set to "not valid". Bit 29 and 30 may be static (not changeable), e.g. due to hardware restrictions. In that case no error is signalled on the attempt to change them.

The transmission type (sub-index 2) defines the transmission character of the PDO (see section 5.2). Table 10-7 describes the usage of this entry.

transmission type	PDO transmission				
	cycli c	acycli c	synchronou s	asynchronous	RTR only
0		X	X		
1-240	X		X		
241-251	- reserved -				
252			X		X
253				X	X
254[21]				X	
255[22]				X	

Table 10-7: Description of transmission type

Synchronous (transmission types 0-240 and 252) means that the transmission of the PDO shall be related to the SYNC object as described in chapter 6.1. Preferably the devices use the SYNC as a trigger to output or actuate based on the previous synchronous Receive-PDO respectively to update the data transmitted at the following synchronous Transmit-PDO. Details of this mechanism depend on the device type and are defined in the device profile if applicable.

Asynchronous means that the transmission of the PDO is not related to the SYNC object.

A transmission type of zero means that the message shall be transmitted synchronously with the SYNC object but not periodically.

A value between 1 and 240 means that the PDO is transmitted synchronously and cyclically, the transmission type indicating the number of SYNC objects between two PDO transmissions.

The transmission types 252 and 253 mean that the PDO is an event without immediate notification and is only transmitted on remote transmission request. At transmission type 252, the data is updated (but not sent) immediately after reception of the SYNC object. At transmission type 253 the data is updated at the reception of the remote transmission request (hardware and software restrictions may apply).

Transmission type 254 means, the application event is manufacturer specific (manufacturer specific part of the object dictionary), transmission type 255 means, the application event is defined in the device profile.

Sub-index 3H contains the inhibit time as specified in /11/. If the inhibit time feature is not supported, the entry at sub-index 0H is set to 2.

Sub-index 4H contains the CMS priority group as specified in /11/ that the device requests for this PDO if DBT services are executed. Devices that do not support DBT do not need to support this entry (value at sub-index 0H is set to 3H). The value range is 0...7.

[21] The transmission of this PDO is initiated by an event on the device. The event is manufacturer specific.

[22] The transmission of this PDO is initiated by an event on the device. This event must be defined in the device profile.

10.1.3 PDO Mapping

The PDOMapping entry describes the PDO content by listing the sequence and length of the mapped object dictionary entries (see Figure 10-2).

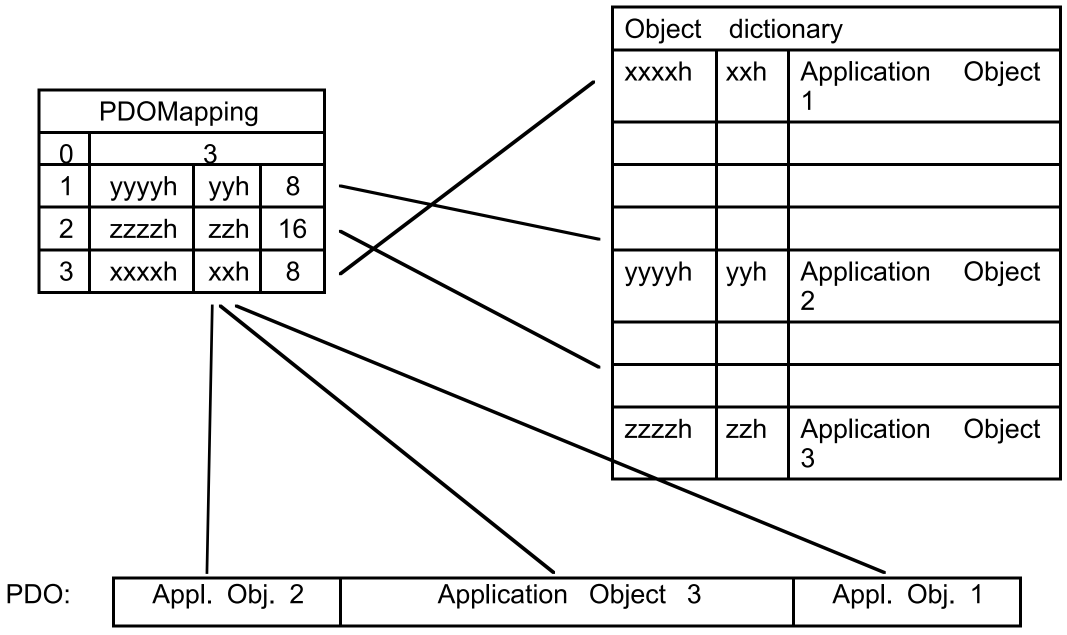


Figure 10-2 : Principle of PDOMapping

The PDOMapping is structured as follows:

Index	Sub-Index	Field in PDOMapping Record	Data Type
0021H	0H	number of mapped objects in PDO	Unsigned8
	1H	1st object to be mapped	Unsigned32
	2H	2nd object to be mapped	Unsigned32
: : : : : :	: : : : : :	: : : : : : : :	: : : : : :
	40H	64th object to be mapped	Unsigned32

Table 10-8: PDO Mapping

The structure of the entries from sub-index 1H - 40H is as follows:

Byte: MSB		LSB
index (16 bit)	sub-index (8 bit)	object length (8 bit)

Figure 10-3: Structure of PDO Mapping Entry

The object length is encoded as unsigned8 with a value range of 1...64. This parameter can be used to verify the overall mapping length. It is mandatory.

If the change of the PDO mapping cannot be executed (e.g. the PDO length is exceeded or the SDO client attempts to map an object that cannot be mapped) the device responds with an Abort_Domain_Transfer service.

Writing to sub-index 0 determines the valid number of objects that have been mapped. E.g. if 64 objects of data type Boolean in the mapping structure are to be replaced by 8 objects of data type Unsigned8, first an „8“ is written to sub-index 8, thus setting valid only the first 8 objects. These can then be re-mapped without immediately exceeding the length of the PDO.

If data types (Index 1-7h) are mapped they serve as „dummy entries“. The corresponding data in the PDO is not evaluated by the device. This optional feature is useful e.g. to transmit data to several devices using one PDO, each device only utilising a part of the PDO.

A device that supports dynamic mapping of PDOs must support this during the Pre-Operational state. If dynamic mapping during the Operational state is supported, the SDO client is responsible for data consistency.

10.1.4 SDO Parameter

In order to describe the SDOs used on a device the data type *SDO parameter object* is introduced. The data type has the index 22h in the object dictionary. The object is structured as follows:

Index	Sub-Index	Field in SDOParameter Record	Data Type
0022H	0H	number of supported entries	Unsigned8
	1H	COB-ID client -> server	Unsigned32
	2H	COB-ID server -> client	Unsigned32
	3H	node ID of SDO’s client resp. server	Unsigned8

Table 10-9: SDO Parameter Record

The number of supported entries in the SDO object record is specified at sub-index 0H. The values at 1H and 2H specify the COB-ID for this SDO. The usage of extended identifiers is possible (if DBT is not used). Subindex 3 gives the server of the SDO in case the record describes an SDO for which the device is client and gives the client of the SDO if the record describes an SDO for which the device is server.

Devices which are clients for the SDO must support the entry 3H (they must know the node id of the SDO server). If the device is the server of the SDO the sub-index 3H is optional.

Unsigned32

MSB LSB

bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0	1	29-bit Identifier	

Figure 10-4: Structure of SDO COB-ID entry

bit number	value	meaning
31 (MSB)	0 1	SDO valid SDO not valid
30	0	reserved (always 0)
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)

28 - 11	0
	X
		if bit 29=0
		if bit 29=1: bits 28-11 of 29-bit-COB-ID

10-0 (LSB)	X	bits 10-0 of COB-ID

Table 10-10: Description of SDO COB-ID entry

An SDO is only valid if booth SDO-valid-bits are 0. Bit 29 may be static (not changeable) e.g. due to hardware restrictions.

Overview Communication Profile Object Dictionary Entries

Table 10-11 gives an overview over the object dictionary entries defined by the communication profile:

Index (hex)	**Object (Symbolic Name)**	**Name**	**Type**	**Acc.**[23]	**M/O**
1000	VAR	device type	Unsigned32	ro	M
1001	VAR	error register	Unsigned8	ro	M
1002	VAR	manufacturer status register	Unsigned32	ro	O
1003	ARRAY	pre-defined error field	Unsigned32	ro	O
1004	ARRAY	number of PDOs supported	Unsigned32	ro	O
1005	VAR	COB-ID SYNC-message	Unsigned32	rw	O
1006	VAR	communication cycle period	Unsigned32	rw	O
1007	VAR	synchronous window length	Unsigned32	rw	O
1008	VAR	manufacturer device name	Vis-String	ro	O
1009	VAR	manufacturer hardware version	Vis-String	ro	O
100A	VAR	manufacturer software version	Vis-String	ro	O
100B	VAR	Node-ID	Unsigned32	ro	O
100C	VAR	guard time	Unsigned32	rw	O
100D	VAR	life time factor	Unsigned32	rw	O
100E	VAR	COB-ID guarding protocol	Unsigned32	rw	O
100F	VAR	number of SDOs supported	Unsigned32	ro	O
1010	VAR	store parameters	Unsigned32	rw	O
1011	VAR	restore default parameters	Unsigned32	rw	O

[23] Access type listed here may vary for certain sub-indices. See detailed object specification.

1012	VAR	COB-ID time stamp	Unsigned32	rw	O
1013	VAR	high resolution time stamp	Unsigned32	rw	O
1014	VAR	COB-ID Emergency	Unsigned32	rw	O
1015		reserved			
:::::	:::::	:::::	:::::	:::::	:::::
11FF		reserved			
Server SDO Parameter (22H)					
1200	RECORD	1st Server SDO parameter	SDOParameter	ro	O
1201	RECORD	2nd Server SDO parameter	SDOParameter	rw	O
:::::	:::::	:::::	:::::	:::::	:::::
127F	RECORD	128th Server SDO parameter	SDOParameter	rw	O
Client SDO Parameter (22H)					
1280	RECORD	1st Client SDO parameter	SDOParameter	rw	O
1281	RECORD	2nd Client SDO parameter	SDOParameter	rw	O
:::::	:::::	:::::	:::::	:::::	:::::
12FF	RECORD	128th Server SDO parameter	SDOParameter	rw	O
1300		reserved			
:::::	:::::	:::::	:::::	:::::	:::::
13FF		reserved			
Receive PDO Communication Parameter (20H)					
1400	RECORD	1st receive PDO Parameter	PDOCommPar	rw	M/O*
1401	RECORD	2nd receive PDO Parameter	PDOCommPar		M/O*
:::::	:::::	:::::	:::::	:::::	:::::
15FF	RECORD	512th receive PDO Parameter	PDOCommPar	rw	M/O*
Receive PDO Mapping Parameter (21H)					
1600	ARRAY	1st receive PDO mapping	PDOMapping	rw	M/O*
1601	ARRAY	2nd receive PDO mapping	PDOMapping	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
17FF	ARRAY	512th receive PDO mapping	PDOMapping	rw	M/O*
Transmit PDO Communication Parameter (20H)					
1800	RECORD	1st transmit PDO Parameter	PDOCommPar	rw	M/O*
1801	RECORD	2nd transmit PDO Parameter	PDOCommPar	rw	M/O*
:::::	:::::	:::::	:::::	:::::	::::
19FF	RECORD	512th transmit PDO Parameter	PDOCommPar	rw	M/O O
Transmit PDO Mapping Parameter (21H)					
1A00	ARRAY	1st transmit PDO mapping	PDOMapping	rw	M/O*
1A01	ARRAY	2nd transmit PDO mapping	PDOMapping	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
1BFF	ARRAY	512th transmit PDO mapping	PDOMapping	rw	M/O*

* If a device supports PDOs, the according PDO communication parameter and PDO mapping entries in the object dictionary are mandatory. These may be read_only.

Table 10-11: Standard Objects

10.2 Detailed Object Specification

So far, parts of the object dictionary have been illustrated in tabular form. All device profiles should contain a full object dictionary definition in this format. However this provides only an overall description of a module's capabilities. Each object dictionary entry needs to be specified precisely.

Every object dictionary entry must be described in the following manner:

OBJECT DESCRIPTION

INDEX	Profile Index Number e.g. 6059H
Name	Name of parameter e.g. 'Frequency-Motor-Min-Max'
Object Code	Variable classification e.g. 8H (=Array)
Number Of Elements	not counting sub-indices 0h (contains no. of elements) and FFh (contains data type), e.g. 4H = 4 elements (this field is used only if object is not a simple variable)
Data Type	Profile type number e.g. 7H => Integer32

Table 10-12: Format of an Object Description

The Object Code must be one of those defined in Table 10-2 above. For better readability, the Object Description should contain the symbolic Object Name rather than the numeric Object Code, e.g. ARRAY instead of 8H.

VALUE DESCRIPTION

Sub-Index	number of sub-index being described (field only used for arrays & records)
Description	Description of the functionality of the entry.
Object Class	Optional Or Mandatory
Access	Read Only (ro) or Read/Write (rw) or Write Only (wo)
PDO Mapping	Optional/Default/No - can this object be mapped to a PDO. Description: Optional: Object may be mapped into a PDO Default: Object is part of the default mapping (see device profile) No: Object must not be mapped into a PDO
Value Range	size of data e.g. Unsigned32
Mandatory Range	(No) not applicable or range defined as a min & max. value e.g. -50 to +200
Default Value	(No) not applicable or default value of an object after device initialisation

Table 10-13: Object Value Description Format

For simple variables the value description appears once without the sub-index field. For arrays or records the value description must be defined for each element (sub-index).

10.3 Detailed Specification of Communication Profile specific Objects

Object 1000H: Device Type

Contains information about the device type. The object at index 1000H describes the type of device and its functionality. It is composed of a 16bit field which describes the device profile that is used and a second 16bit field which gives additional information about optional functionality of the device. The Additional Information parameter may be device specific. Its specification does not fall within the scope of this document, it is defined in the appropriate device profile.

OBJECT DESCRIPTION

INDEX	1000H
Name	device type
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Mandatory
Access	ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Byte:	MSB	LSB
	Additional Information	Device Profile Number

Figure 10-5: Structure of the Device Type Parameter

Object 1001H: Error Register

This object is an error register for the device. The device can map internal errors in this byte.

OBJECT DESCRIPTION

INDEX	1001H
Name	error register
Object Code	VAR
Data Type	Unsigned8

VALUE DESCRIPTION

Object Class	Mandatory
Access	ro
PDO Mapping	Optional
Value Range	Unsigned8
Mandatory Range	No
Default Value	No

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
7	O	manufacturer specific

Table 10-14: Structure of the Error Register

If a bit is set to 1 the specified error has occurred. The only mandatory error that has to be signalled is the generic error.

Object 1002H: Manufacturer Status Register

This object is a common status register for manufacturer specific purposes. In this document only the size and the location of this object is defined.

OBJECT DESCRIPTION

INDEX	1002H
Name	manufacturer status register
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	Optional
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Object 1003H: Pre-defined Error Field

The object at index 1003H holds the errors that have occurred on the device and have been signalled via the Emergency Object. In doing so it provides an error history.

1. The entry at sub-index 0 contains the number of actual errors that are recorded in the array starting at sub-index 1.
2. Every new error is stored at sub-index 1, the older ones move down the list.
3. Writing a „0“ to sub-index 0 deletes the entire error history (empties the array)
4. The error numbers are of type Unsigned32 and are composed of a 16bit error code (see Table 7-1) and a 16bit additional error information field which may be device specific. The error code is contained in the lower 2 bytes (LSB) and the additional information is included in the upper 2 bytes (MSB). If this object is supported it must consist of two entries at least. The length entry on sub-index 0H and at least one error entry at sub-index 1H.

Byte:	MSB	LSB
	Additional Information	Error code

Figure 10-6: Structure of the pre-defined error field

OBJECT DESCRIPTION

INDEX	1003H
Name	pre-defined error field
Object Code	ARRAY
Number of Elements	1(Mandatory), 2-254 (Optional)
Data Type	Unsigned32

VALUE DESCRIPTION

Sub-Index	0H
Description	number of errors
Object Class	Mandatory
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	1
Default Value	No

Sub-Index	1H
Description	standard error field
Object Class	Mandatory
Access	ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Sub-Index	2H
Description	standard error field
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

to

Sub-Index	FEH
Description	standard error field
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	No

Object 1004H: Number of PDOs supported

Index 1004H contains information about the maximum number of PDOs supported by the device. It is distinguished between input and output PDOs and between synchronous and asynchronous transmission. Sub-index 0 describes the overall number of PDOs supported (Synchronous and Asynchronous)[24]. Sub-index 1 describes the number of synchronous PDOs that is supported by the device, sub-index 2 the number of asynchronous PDOs that is supported. Each of the values is of type Unsigned16. Figure 10-7 shows the structure of this entry.

OBJECT DESCRIPTION

INDEX	1004H
Name	number of PDOs supported
Object Code	ARRAY
Number of Elements	2
Data Type	Unsigned32

[24] A device may support a fixed number of PDOs, each of which may either be of synchronous and asynchronous transmission type.

VALUE DESCRIPTION

Sub-Index	0H
Description	number of PDOs supported
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	unsigned32
Mandatory Range	0H - 1FF01FFH
Default Value	No

Sub-Index	1H
Description	number of synchronous PDOs
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	unsigned32
Mandatory Range	0H - 1FF01FFH
Default Value	No

Sub-Index	2H
Description	number of asynchronous PDOs
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	unsigned32
Mandatory Range	0H - 1FF01FFH
Default Value	No

Sub-index	MSB	LSB
0	No. of Receive PDOs supported	No. of Transmit PDOs supported
1	No. of synchronous Receive PDOs	No. of synchronous Transmit PDOs
2	No. of asynchronous Receive PDOs	No. of asynchronous Transmit PDOs

Figure 10-7: Structure of entry 1004H

Object 1005H: COB-ID SYNC message

Index 1005H defines the COB-ID of the Synchronisation Object (SYNC). Further, it defines whether the device consumes the SYNC or whether the device generates the SYNC. The structure of this object is shown in Figure 10-8 and Table 10-15. The default value for the SNYC COB-ID is given in chapter 7.1.

OBJECT DESCRIPTION

INDEX	1005H
Name	COB-ID SYNC message
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	80h

Unsigned32

MSB LSB

bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0/1	1	29 -bit Identifier	

Figure 10-8: Structure of SYNC COB-ID entry

bit number	value	meaning
31 (MSB)	0	Device does not consume SYNC message
	1	Device consumes SYNC message
30	0	Device does not generate SYNC message
	1	Device generates SYNC message
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28 - 11	0	if bit 29=0
	X	if bit 29=1: bits 28-11 of 29-bit-SYNC-COB-ID
10-0 (LSB)	X	bits 10-0 of SYNC-COB-ID

Table 10-15: Description of SYNC COB-ID entry

Bit 29 may be static (not changeable) e.g. due to hardware restrictions.

Object 1006H: Communication Cycle Period

This object defines the communication cycle period in µsec. 0 if not used.

OBJECT DESCRIPTION

INDEX	1006H
Name	communication cycle period
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	00’00’00’00h

Object 1007H: Synchronous Window Length

Contains the length of the time window for synchronous messages in µsec. 0 if not used.

OBJECT DESCRIPTION

INDEX	1007H
Name	synchronous window length
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	00’00’00’00h

Object 1008H: Manufacturer Device Name

Contains the manufacturer device name.

OBJECT DESCRIPTION

INDEX	1008H
Name	manufacturer device name
Object Code	VAR
Data Type	Visible String

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Object 1009H: Manufacturer Hardware Version

Contains the manufacturer hardware version description.

OBJECT DESCRIPTION

INDEX	1009H
Name	manufacturer hardware version
Object Code	VAR
Data Type	Visible String

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Object 100AH: Manufacturer Software Version

Contains the manufacturer software version description.

OBJECT DESCRIPTION

INDEX	100AH
Name	manufacturer software version
Object Code	VAR
Data Type	Visible String

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	No
Mandatory Range	No
Default Value	No

Object 100BH: Node-ID

The object at Index 100BH contains the Node-ID. The node ID entry has the access type "read only" as it can not be changed using SDO services. However, it is feasible to change it via LMT (see /14/), via hardware (e.g. dip-switch).

OBJECT DESCRIPTION

INDEX	100BH
Name	Node-ID
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	1 - 256
Mandatory Range	1 - 127
Default Value	No

MSB LSB

Reserved	Reserved	Reserved	Node -ID

Figure 10-9: Structure of the Node-ID Parameter

Object 100CH: Guard Time

The objects at index 100CH and 100DH include the guard time in milli-seconds and the life time factor. The life time factor multiplied with the guard time gives the life time for the Node Guarding Protocol (see /10/).

OBJECT DESCRIPTION

INDEX	100CH
Name	guard time
Object Code	VAR
Data Type	Unsigned16

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned16
Mandatory Range	No
Default Value	0h

Object 100DH: Life Time Factor

The life time factor multiplied with the guard time gives the life time for the node guarding protocol (see /10/).

OBJECT DESCRIPTION

INDEX	100DH
Name	life time factor
Object Code	VAR
Data Type	Unsigned8

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	0h

Object 100EH: Node Guarding Identifier

The identifier used for node guarding and life guarding procedure (see chapter 9.1)

OBJECT DESCRIPTION

INDEX	100EH
Name	node guarding identifier
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	700h + node-ID

Unsigned32

MSB LSB

bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	reserved		0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	reserved		1	29 -bit Identifier	

Figure 10-10: Structure of the Node Guarding Identifier entry

Bit 29 may be static (not changeable) e.g. due to hardware restrictions.

Object 100FH: Number of SDOs Supported

If a device supports more than the default SDO the number of supported SDOs is described in this object. This entry shows all available SDOs including the default SDO. The object is composed of a 16bit field which describes the number of Client SDOs and a 16bit field which describes the number of Server SDOs.

OBJECT DESCRIPTION

INDEX	100FH
Name	number of SDOs supported
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	1h-80h (server), 0-80h (client)
Default Value	No

Byte:	MSB		LSB
	number of client SDOs		number of server SDOs

Figure 10-11: Structure of the Number of SDOs entry

Object 1010H: Store parameters

This Object supports the saving of parameters in non volatile memory. By read access the device provides information about its saving capabilities. Several parameter groups are distinguished:

Sub-Index 0 contains the largest Sub-Index that is supported.

Sub-Index 1 refers to all parameters that can be stored on the device.

Sub-Index 2 refers to communication related parameters (Index 1000h - 1FFFh manufacturer specific communication parameters).

Sub-Index 3 refers to application related parameters (Index 6000h - 9FFFh manufacturer specific application parameters).

At Sub-Index 4 - 127 manufacturers may store their choice of parameters individually.

Sub-Index 128 - 254 are reserved for future use.

In order to avoid storage of parameters by mistake, storage is only executed when a specific signature is written to the appropriate Sub-Index. The signature is „save“.

Signature	MSB			LSB
ASCII	e	v	a	s
hex	65h	76h	61h	73h

Figure 10-12: Storage write access signature

On reception of the correct signature in the appropriate Sub-Index the device stores the parameter and then confirms the SDO transmission (initiate download response). If the storing failed, the device responds with abort domain transfer, error class 6h, error code 6h (hardware fault).

If a wrong signature is written, the device refuses to store and responds with abort domain transfer, error class 8h, error code 0h (other error).

On read access to the appropriate Sub-Index the device provides information about its storage functionality with the following format:

Unsigned32

bits	MSB 31-2	1	LSB 0
	reserved (=0)	0/1	0/1

Figure 10-13: Storage read access structure

bit number	value	meaning
31-2	0	*reserved*
1	0	Device does not save parameters autonomously
	1	Device saves parameters autonomously
0	0	Device does not save parameters on command
	1	Device saves parameters on command

Autonomous saving means that a device stores the storable parameters in a non-volatile manner without user request.

OBJECT DESCRIPTION

INDEX	1010H
Name	store parameters
Object Code	ARRAY
Number of Elements	1H - 7FH
Data Type	Unsigned32

VALUE DESCRIPTION

Sub-Index	0H
Description	largest supported Sub-Index
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	No

Sub-Index	1H
Description	save all parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-12+13)
Mandatory Range	No

Default Value	No

Sub-Index	2H
Description	save communication parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-12+13)
Mandatory Range	No
Default Value	No

Sub-Index	3H
Description	save application parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-12+13)
Mandatory Range	No
Default Value	No

Sub-Index	4H - 7FH
Description	save manufacturer defined parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-12+13)
Mandatory Range	No
Default Value	No

Object 1011H: Restore default parameters

With this object the default values of parameters according to the communication or device profile are restored. By read access the device provides information about its capabilities to restore these values. Several parameter groups are distinguished:

Sub-Index 0 contains the largest Sub-Index that is supported.

Sub-Index 1 refers to all parameters that can be restored.

Sub-Index 2 refers to communication related parameters (Index 1000h - 1FFFh manufacturer specific communication parameters).

Sub-Index 3 refers to application related parameters (Index 6000h - 9FFFh manufacturer specific application parameters).

At Sub-Index 4 - 127 manufacturers may restore their individual choice of parameters.

Sub-Index 128 - 254 are reserved for future use.

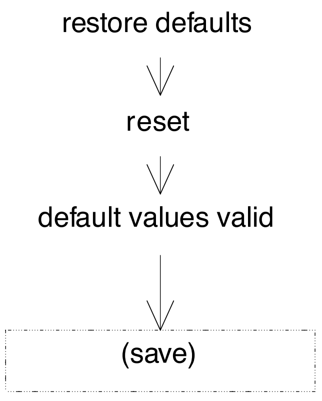
In order to avoid the restoring of default parameters by mistake, restoring is only executed when a specific signature is written to the appropriate Sub-Index. The signature is „load“.

Signature	MSB			LSB
ASCII	d	a	o	l
hex	64h	61h	6Fh	6Ch

Figure 10-14: Restoring write access signature

On reception of the correct signature in the appropriate Sub-Index the device restores the default parameters and then confirms the SDO transmission (initiate download response). If the restoring failed, the device responds with abort domain transfer, error class 6h, error code 6h (hardware fault). If a wrong signature is written, the device refuses to restore the defaults and responds with abort domain transfer, error class 8h, error code 0h (other error).

The default values are set valid after the device is reset (reset node for subindex 1h - 127h, reset communication for subindex 2h). If the device requires storing on command (see 1010h), the appropriate command has to be executed after the reset if the default parameters are to be stored permanently.



On read access to the appropriate Sub-Index the device provides information about its default parameter restoring capability with the following format:

Unsigned32

	MSB LSB	
bits	*31-1*	*0*
	reserved (=0)	0/1

Figure 10-15: Restoring default values read access structure

bit number	value	meaning
31-1	0	*reserved*
0	0	Device does not restore default parameters

	1	Device restores parameters

OBJECT DESCRIPTION

INDEX	1011H
Name	restore default parameters
Object Code	ARRAY
Number of Elements	1H - 7FH
Data Type	Unsigned32

VALUE DESCRIPTION

Sub-Index	0H
Description	largest supported Sub-Index
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	No

Sub-Index	1H
Description	restore all default parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-14+15)
Mandatory Range	No
Default Value	No

Sub-Index	2H
Description	restore communication default parameters
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-14+15)
Mandatory Range	No
Default Value	No

Sub-Index	3H
Description	restore application default parameters
Object Class	Optional
Access	rw
PDO Mapping	No

Value Range	Unsigned32 (see Fig. 10-14+15)
Mandatory Range	No
Default Value	No

Sub-Index	4H - 7FH
Description	restore manufacturer defined default parameters
Object Class	Optional
PDO Mapping	No
Value Range	Unsigned32 (see Fig. 10-14+15)
Mandatory Range	No

Object 1012H: COB-ID Time-stamp message

Index 1012H defines the COB-ID of the Time-Stamp Object (TIME). Further, it defines whether the device consumes the TIME or whether the device generates the TIME. The structure of this object is shown in Figure 10-8 and Table 10-15, it is similar to the entry 1005h (COB-ID SYNC message).

OBJECT DESCRIPTION

INDEX	1012H
Name	COB-ID time stamp message
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	100h

Object 1013H: High Resolution Time Stamp

This object contains a time stamp with a resolution of 1 µsec. It can be mapped into a PDO in order to define a high resolution time stamp message. (Note that the data type of the standard time stamp message (TIME) is fixed). Further application specific use is encouraged.

OBJECT DESCRIPTION

INDEX	1013H
Name	high resolution time stamp
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	Optional
Value Range	Unsigned32
Mandatory Range	No
Default Value	00’00’00’00h

Object 1014H: COB-ID Emergency Message

Index 1014H defines the COB-ID of the Emergency Object (EMCY). The structure of this object is shown in Figure 10-8 and Table 10-15, it is similar to the entry 1005h (COB-ID SYNC message).

OBJECT DESCRIPTION

INDEX	1014H
Name	COB-ID Emergency message
Object Code	VAR
Data Type	Unsigned32

VALUE DESCRIPTION

Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	80h + node-ID

Object 1200H - 127FH: Server SDO Parameter

These objects contain the parameters for the SDOs for which the device is the server. If a device handles more than one server SDO the default SDO must be located at index 1200H as the first server SDO. This entry is read only[25]. All additional server SDOs are invalid by default (invalid bit - see table 10-6).

The description of the Client of the SDO (Sub-index 3h) is optional. It is not available for the default SDO (no Sub-index 3h at Index 1200H), as this entry is read_only.

OBJECT DESCRIPTION

INDEX	1200H - 127FH
Name	Server SDO parameter
Object Code	RECORD
Number of Elements	3H
Data Type	SDOPar

VALUE DESCRIPTION

Sub-Index	0H
Description	number of entries
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2
Default Value	No

Sub-Index	1H
Description	COB-ID Client->Server (rx)
Object Class	Optional
Access	Index 1200h: ro, Index 1201h-127Fh: rw
PDO Mapping	No
Value Range	Unsigned32 (figure 10-4)
Mandatory Range	No
Default Value	Index 1200h: 600h+Node-ID, Index 1201h-127Fh: No

25 It must be ensured that the COB-IDs of the default SDO can not be manipulated by writing to the object dictionary.

Sub-Index	2H
Description	COB-ID Server -> Client (tx)
Object Class	Optional
Access	Index 1200h: ro Index 1201-127Fh: rw
PDO Mapping	No
Value Range	Unsigned32 (figure 10-4)
Mandatory Range	No
Default Value	Index 1200h: 580h+Node-ID, Index 1201h-127Fh: No

Sub-Index	3H
Description	node ID of the SDO client
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	No

Object 1280H - 12FFH: Client SDO Parameter
These objects contain the parameters for the SDOs for which the device is the client. If the entry is supported, all sub-indices must be available.

OBJECT DESCRIPTION

INDEX	1280H - 12FFH
Name	Client SDO parameter
Object Code	RECORD
Number of Elements	3H
Data Type	SDOPar

VALUE DESCRIPTION

Sub-Index	0H
Description	number of entries
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	3
Default Value	3

Sub-Index	1H
Description	COB-ID Client->Server (tx)
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (figure 10-4)
Mandatory Range	No
Default Value	No

Sub-Index	2H
Description	COB-ID Server -> Client (rx)
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (figure 10-4)
Mandatory Range	No
Default Value	No

Sub-Index	3H
Description	node ID of the SDO server
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	No
Default Value	No

Object 1400H - 15FFH: Receive PDO Communication Parameter

Contains the communication parameters for the PDOs the device is able to receive.

OBJECT DESCRIPTION

INDEX	1400H - 15FFH
Name	receive PDO parameter
Object Code	RECORD
Number of Elements	2 (Mandatory), 3-4 (Optional)
Data Type	PDOCommPar

VALUE DESCRIPTION

Sub-Index	0H
Description	number of entries
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2 - 4

Sub-Index	1H
Description	COB-ID used by PDO
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (Figure 10-6)
Mandatory Range	No
Default Value	Index 1400h: 200h + Node-ID, Index 1401h: 300h + Node-ID, Index 1402h - 15FFh: No

Sub-Index	2H
Description	transmission type
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8 (Table 10-7)
Mandatory Range	No
Default Value	(Device Profile dependent)

Sub-Index	3H
Description	inhibit time
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned16
Mandatory Range	No
Default Value	No

Sub-Index	4H
Description	CMS priority group
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	0 - 7
Default Value	No

Object 1600H - 17FFH: Receive PDO Mapping Parameter

Contains the mapping for the PDOs the device is able to receive.

OBJECT DESCRIPTION

INDEX	1600H - 17FFH
Name	receive PDO mapping
Object Code	RECORD
Number of Elements	1H-64H
Data Type	PDOMapping

VALUE DESCRIPTION

Sub-Index	0H
Description	number of mapped application objects in PDO
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	1 - 64
Default Value	(Device Profile dependent)

Sub-Index	1H - 40H
Description	PDO mapping for the nth application object to be mapped
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32
Mandatory Range	No
Default Value	(Device Profile dependent)

Object 1800H - 19FFH: Transmit PDO Communication Parameter

Contains the communication parameters for the PDOs the device is able to transmit.

OBJECT DESCRIPTION

INDEX	1800H - 19FFH
Name	transmit PDO parameter
Object Code	RECORD
Number of Elements	2 (Mandatory), 3-4 (Optional)
Data Type	PDOCommPar

VALUE DESCRIPTION

Sub-Index	0H
Description	number of entries
Object Class	Optional
Access	ro
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	2 - 4

Sub-Index	1H
Description	COB-ID used by PDO
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned32 (Figure 10-6)
Mandatory Range	No
Default Value	Index 1800h: 180h + Node-ID, Index 1801h: 280h + Node-ID, Index 1802h - 18FFh: No

Sub-Index	2H
Description	transmission type
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8 (Table 10-7)
Mandatory Range	No
Default Value	(Device Profile dependent)

Sub-Index	3H
Description	inhibit time
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned16
Mandatory Range	No
Default Value	No

Sub-Index	4H
Description	CMS priority group
Object Class	Optional
Access	rw
PDO Mapping	No
Value Range	Unsigned8
Mandatory Range	0 - 7
Default Value	No

Object 1A00H - 1BFFH: Transmit PDO Mapping Parameter

Contains the mapping for the PDOs the device is able to transmit.

OBJECT DESCRIPTION

INDEX	1A00H - 1BFFH
Name	transmit PDO mapping
Object Code	RECORD
Number of Elements	1H-64H
Data Type	PDOMapping

VALUE DESCRIPTION

See Value Description of objects 1600H - 17FFH.

11 Physical Layer

The physical medium for CANopen devices is a differentially driven two-wire bus line with common return according to ISO 11898 /2/.

11.1 Physical medium Specification

See /4/.

11.2 Transceiver

See /2/. The maximum rating for V_{CAN_H} and V_{CAN_L} is +16V. Galvanic isolation between bus nodes is optional.

11.3 Bit Rates, Bit Timing

Every module has to support a bit rate of 20 kbit/s. The recommended bit rates and corresponding bit timing recommendations are listed in table 11-1.

Bit rate Bus length (1)	Nominal bit time t_b	Number of time quanta per bit	Length of time quantum t_q	Location of sample point
1 Mbit/s 25 m	1 µs	8	125 ns	6 t_q (750 ns)
800 kbit/s 50 m	1.25 µs	10	125 ns	8 t_q (1 µs)
500 kbit/s 100 m	2 µs	16	125 ns	14 t_q (1.75 µs)
250 kbit/s 250 m (2)	4 µs	16	250 ns	14 t_q (3.5 µs)
125 kbit/s 500 m (2)	8 µs	16	500 ns	14 t_q (7 µs)
50 kbit/s 1000 m (3)	20 µs	16	1.25 µs	14 t_q (17.5 µs)
20 kbit/s 2500 m (3)	50 µs	16	3.125 µs	14 t_q (43.75 µs)
10 kbit/s 5000 m (3)	100 µs	16	6.25 µs	14 t_q (87.5 µs)

Table 11-1: Recommended Bit Timing Settings

Oscillator frequency	16 MHz +/-0.1% (1000 ppm)	
Sampling mode	Single sampling	SAM = 0
Synchronisation mode	Recessive to dominant edges only	SYNC = 0
Synchronisation jump width	1 * t_q	SJW = 0
Phase Segment 2	2 * t_q	TSEG2 = 1

Note 1: Rounded bus length estimation (worst case) on basis 5 ns/m propagation delay and a total effective ECU-internal (Elec. Control Unit) in-out delay as follows: 1M-800 kbit/s: 210ns

500 - 250 kbit/s:	300 ns (includes 2 * 40 ns for optocouplers)
124 kbit/s:	450 ns (includes 2 * 100 ns for optocouplers)
50 - 10 kbit/s:	1.5 t_q; Effective delay = delay recessive to dominant plus dominant to recessive divided by two.

Note 2: For bus length greater than about 200 m the use of optocouplers is recommended. If optocouplers are placed between CAN controller and transceiver this affects the maximum bus length depending upon the propagation delay of the optocouplers i.e. -4m per 10 ns propagation delay of employed optocoupler type.

Note 3: For bus length greater than about 1 km bridge or repeater devices may be needed.

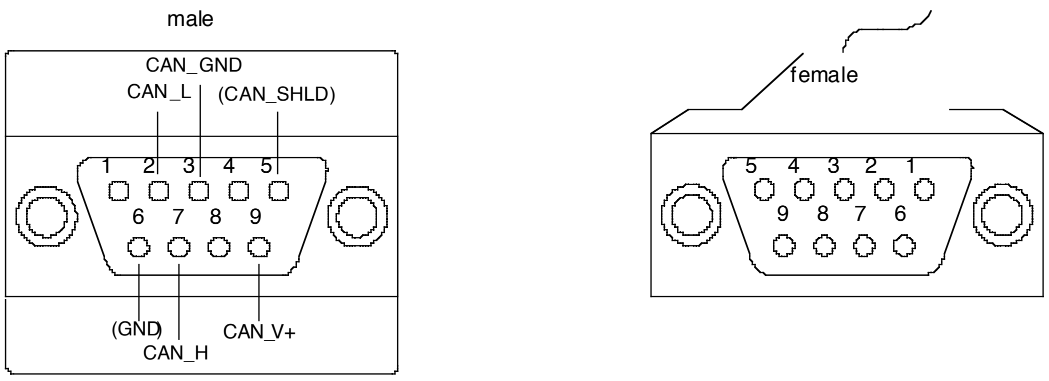
A module shall support as many of the recommended bit rates as possible. It is not required, that a module has to support all recommended bit rates.

11.4 External Power Supply

The recommended output voltage at the optional external power supply is +18VDC < V+ < +30VDC in order to enable the use of standard industrial power supplies (24VDC).

11.5 Bus Connector

11.5.1 9-pin D-Sub Connector



It is recommended to use a 9-pin D-Sub connector (DIN 41652 or corresponding international standard) with the pinning according to CiA DS 102 /4/. For convenience the pinning is repeated here:

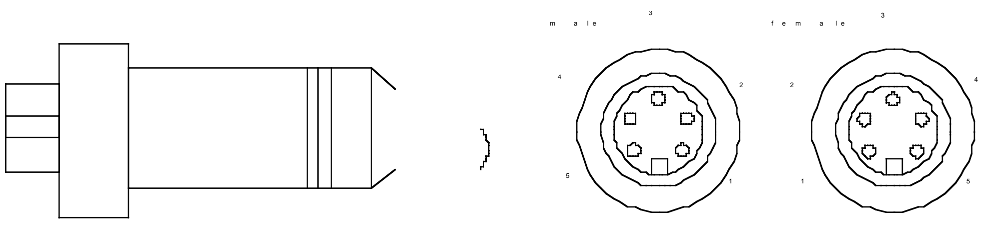
Pin	Signal	Description
1	-	Reserved
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_GND	CAN Ground
4	-	Reserved
5	(CAN_SHLD)	Optional CAN Shield
6	(GND)	Optional Ground
7	CAN_H	CAN_H bus line (dominant high)
8	-	Reserved
9	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and optocouplers, if galvanic isolation of the bus node applies)

If 9-pin D-Sub connector is supported, a male connector meeting the above specification has to be provided by the bus node. Within the modules, pin 3 and pin 6 have to be interconnected. Inside of such modules providing two bus connections, and inside the T-connectors, all the pins (including the reserved ones) have to be connected. The intention is, that there shall be no interruption of any of the wires in the bus cable, assuming a future specification of the use of the reserved pins.

By using the pin V+ for supplying transceivers in case of galvanic isolation, the necessity of an extra local power isolation (e.g. DC/DC-converter) is avoided.

If an error line is needed within a system, then pin 8 shall be used for this purpose.

11.5.2 5-pin Mini Style Connector

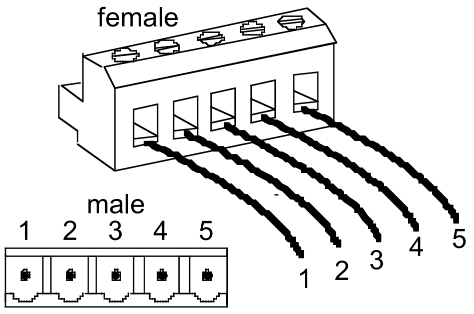


If 5-pin Mini Style Connectors are used the following pinning applies:

Pin	Signal	Description
1	(CAN_SHLD)	Optional CAN Shield
2	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and optocouplers, if galvanic isolation of the bus node applies)
3	CAN_GND	Ground / 0V / V-
4	CAN_H	CAN_H bus line (dominant high)
5	CAN_L	CAN_L bus line (dominant low)

The bus node provides the male pins of the connector.

11.5.3 Open Style Connector

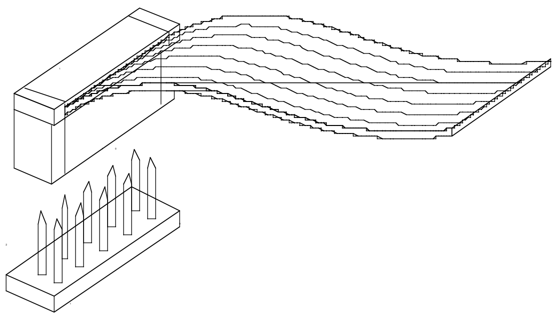


If Open Style Connectors are used the following pinning is recommended:

Pin	Signal	Description
1	CAN_GND	Ground / 0 V / V-
2	CAN_L	CAN_L bus line (dominant low)
3	(CAN_SHLD)	Optional CAN Shield
4	CAN_H	CAN_H bus line (dominant high)
5	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and optocouplers, if galvanic isolation of the bus node applies)

4-pin Open Style Connectors either use pins 1-4 (Version A) or pins 2-5 (Version B). 3-pin Open Style Connectors use pins 2-4. The bus node provides the male pins of the connector.

11.5.4 Multipole Connector



If (5 x 2) multipole connectors are used (e.g. inside EMI protected housings) the following pinning is recommended, as it supports direct connection of the flat cables to 9-pin D-sub connectors:

Pin	Signal	Description
1	reserved	
2	(GND)	Optional Ground
3	CAN_L	CAN_L bus line (dominant low)
4	CAN_H	CAN_H bus line (dominant high)
5	CAN_GND	CAN Ground
6	reserved	
7	reserved	
8	(CAN_V+)	Optional CAN external positive supply
9	reserved	
10	reserved	

11.5.5 Other Connectors

If different connectors are used, the pins have to be named (either in the accompanying manual or directly on the device) using the following terminology:

Signal description	notation
CAN_L bus line (dominant low)	CAN_L or CAN_{low} or CAN-
CAN_H bus line (dominant high)	CAN_H or CAN_{high} or CAN+
CAN Ground	CAN_GND or CAN_{GND} or Ground or GND
Optional CAN Shield	CAN_SHLD or CAN_{SHIELD} or Shield or SHLD
Optional CAN external positive supply	CAN_V+ or CAN_{V+} or V+ or UC or U_{CAN}
Optional Ground	OPT_GND or GND_{opt} or V- or 0V

12 APPENDIX

12.1 ExamplePDOMapping:

Consider a device which has the following object dictionary entries:

Index	Sub-Index	Object Name	Data Type
6060	-	variable_1	Unsigned16
6091	-	variable_2	Unsigned32
6092	-	variable_3	Unsigned32
60AE	-	variable_4	Unsigned16
60D0	0	rec_elem_0	Unsigned8
	1	rec_elem_1	Unsigned8
	2	rec_elem_2	Unsigned8
	3	rec_elem_3	Unsigned8

Table A.1: Example of Device Profile Description

If we wish to configure the receive PDO defined on index 1600h to have the following structure:

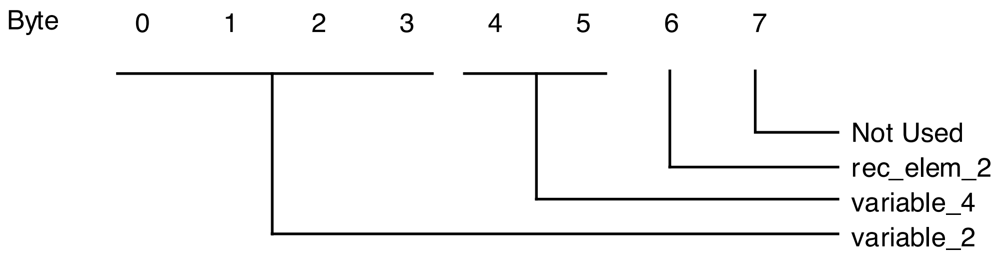


Figure A.1: Desired Input PDO Configuration

Then we would have to write the following values to the corresponding PDOMapping structure at index 1600h in the object dictionary:

Sub-Index (hex)	Value (hex)	Comment
0	3	Number of objects in input = 3
1	60 91 00 20	First object mapped is at index 6091.0 (32 bit length)
2	60 AE 00 10	Second object mapped is at index 60AE.0 (16 bit length)
3	60 D0 02 08	Third object mapped is at index 60D0.2 (8 bit length)

Table A.2: Mapping of receive PDO

If we want this PDO to be reveived synchronously every communication cycle with the COB-ID 300 we have to write the following values in the communication parameter located at 1400H.

Sub-Index (hex)	Value (hex)	Comment
0	4	number of supported entries
1	12C	COB-ID = 300
2	1	transmission type = 1 (cyclic, synchronous)
3	0	no inhibit time
4	0	no priority group (ID is distributed statically)

Table A.3: Communication parameters for receive PDO

If we wish to configure the transmit PDO defined at index 1A00h to have the following structure:

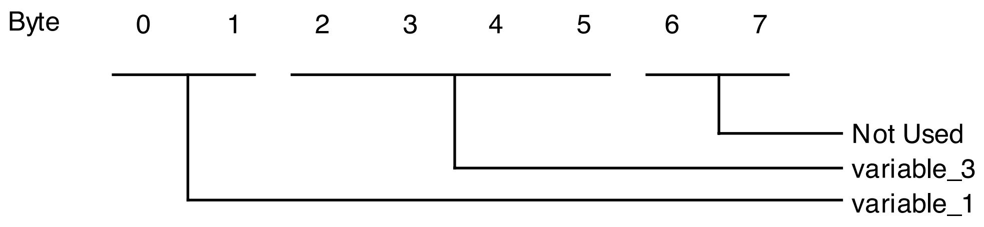


Figure A.2: Desired transmit PDO Configuration

Then we would have to write the following values to the corresponding PDOMapping structure at index 1A00h in the object dictionary:

Sub-Index (hex)	Value (hex)	Comment
0	2	Number of objects in transmit PDO = 2
1	60 60 00 10	First object mapped is at index 6060.0 (16 bit length)
2	60 92 00 20	Second object mapped is at index 6092.0 (32 bit length)

Table A.4: Mapping of transmit PDO

If we want this PDO to be transmitted asynchronously on an device specific event specified in the device profile with the COB-ID 400 we have to write the following values in the communication parameter located at 1800H.

Sub-Index (hex)	Value (hex)	Comment
0	4	number of supported entries
1	190	COB-ID = 400
2	FF	transmission type = 255 (asynchronous, profile specific)
3	0	no inhibit time
4	0	no priority group (ID is distributed statically)

Table A.5: Communication parameters for transmit PDO

12.2 Example for Emergency Message

A Temperature Error is signaled as follows (if detailed error source supported):

Error Code: 40xx (xx = additional information, specified by the device profile)

Error Register:

Bit	7	6	5	4	3	2	1	0
Content	0	0	0	0	1	0	0	1

= 09H

Emergency Message: xx 40 09 yy yy yy yy yy (yy = manufacturer specific)

12.3 Example for Naming Conventions

A device needs identifiers for two synchronous PDO (COMMAND and ACTUAL) and two asynchronous PDO's (device = server). The table lists the COB names used by the device in order to request the identifiers via DBT. The device's node-ID is 1. In „xxx“ the device profile number used for this particular device is coded (e.g. 401 for I/O device profile).

COB name	Communication Parameter at Index	PDOMapping at Index
xxxSYNC000001X	1005h (COB-ID only)	-
xxxSDO_001001C		-
xxxSDO_001001S		-
xxxRPDO001001X	1400h (1st receive PDO)	1600H (1st receive PDO)
xxxTPDO001001X	1800h (1st transmit PDO)	1A00H (1st transmit PDO)
xxxRPDO002001X	1401h (2nd receive PDO)	1601H (2nd receive PDO)
xxxTPDO002001X	1801h (2nd transmit PDO)	1A01H (2nd transmit PDO)
xxxEMCY000001X	-	-

12.4 Example for Device Profile: Object 6C05H of Analogue I/O Device

12.4.1 Object 6C05H

Writes the value to the output channel 'n' (unconverted). Value is 32 bits wide or less.

OBJECT DESCRIPTION

INDEX	6C05H
Name	Write_Analogue_Output_32
Object Code	RECORD
Number Of Elements	0H(Mandatory) 1H-20H(Optional)
Object Class	Mandatory
PDO Mapping	0H(Mandatory) 1H-20H(Optional)

VALUE DESCRIPTION

Sub-Index	0H
Description	Number_Analogue_Outputs
Data Type	Unsigned8
Length	1
Object Class	Mandatory
PDO Mapping	NO
Value Range	0H -20H
Mandatory Range	NO

Sub-Index	1H
Description	Output_1H
Data Type	Unsigned32
Length	2
Object Class	Optional
PDO Mapping	Possible
Value Range	Unsigned32
Mandatory Range	NO

Sub-Index	2H
Description	Output_2H
Data Type	Unsigned32
Length	2
Object Class	Optional
PDO Mapping	Possible
Value Range	Unsigned32
Mandatory Range	NO

to

Sub-Index	20H
Description	Output_20H
Data Type	Unsigned32
Length	2
Object Class	Optional
PDO Mapping	Possible
Value Range	Unsigned32
Mandatory Range	NO

12.5 Electronic Data Sheet Specification

12.5.1 Introduction

In order to give the user of a CANopen device more support the device's description should be available in a standardised way. This gives the opportunity to create standardised tools for:

- configuration of CANopen devices,
- designing networks with CANopen devices,
- managing project information on different platforms.

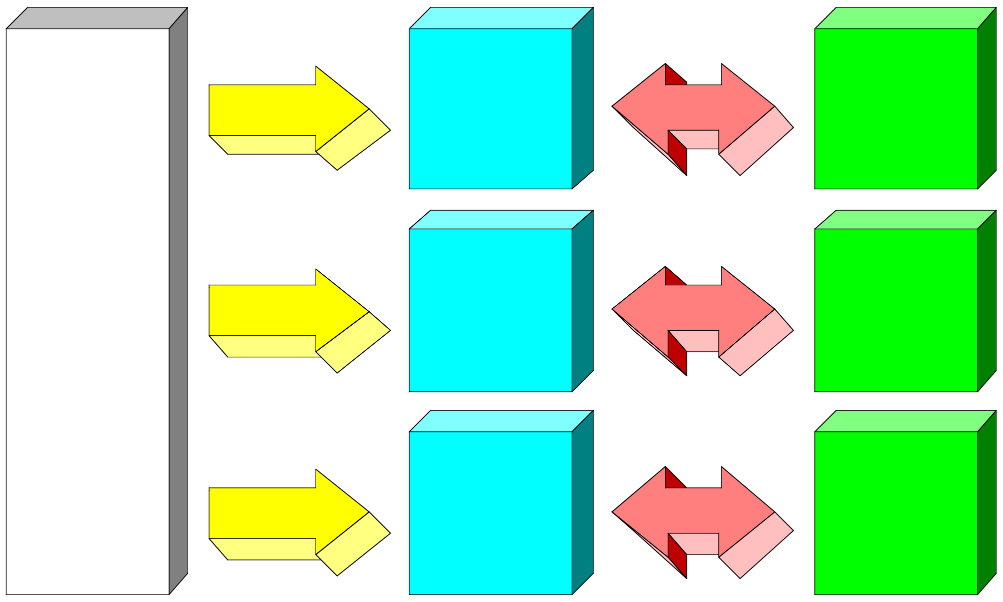
Therefore two types of files are introduced to define a CANopen device with electronically means. The files are ASCII-coded and it is recommended to use the ANSI character set.

12.5.1.1 Electronic Data Sheets (EDS) and Device Configuration Files (DCF)

An EDS can be used to describe the:

- Communication functionality and objects as defined in the CANopen CAL-based Communication Profile DS-301,
- Device specific objects as defined in the device profiles DS-4XX.

The EDS is the template for a device „XY" of the vendor „UV". The DCF describes the incarnation of a device not only with the objects but also with the values of the objects. Furthermore a value for the baudrate of a device and for the module-id are added.



12.5.2 Structure of an EDS (Electronic Data Sheet)

An EDS should be supplied by the vendor of a particular device. If a vendor has no EDS available for his CANopen devices a default EDS can be used. The default EDS comprises all entries of a device profile for a particular device class.

An EDS can be divided into three parts:
- informations regarding the EDS file (creation time, version etc.),
- general device information (name, serial number, hardware revision etc.),
- object dictionary with default values.

12.5.2.1 File Information

Information regarding
- file description,
- creation date and time
- modification date and time
- version management

can be found in the section `FileInfo`.

The following keywords are used:

`FileName`	- file name (according to DOS restrictions),
`FileVersion`	- actual file version (Unsigned8),
`FileRevision`	- actual file revision (Unsigned8),
`Description`	- file description (255 characters),
`CreationTime`	- file creation time (characters in format „hh:mm(AM|PM)“),
`CreationDate`	- date of file creation (characters in format „mm-dd-yy“),
`CreatedBy`	- name or description of file creator (255 characters),
`ModificationTime`	- time of last modification (characters in format „hh:mm(AM|PM)“),
`ModificationDate`	- date of last file modification (characters in format „mm-dd-yy“),
`ModifiedBy`	- name or description of the modification (255 characters).

Example:

```
[FileInfo]
FileName=vendor1.eds
FileVersion=1
FileRevision=2
Description=EDS for simple I/O-device
CreationTime=09:45AM
CreationDate=05-15-95
CreatedBy=Zaphod Beeblebrox
ModificationTime=11:30PM
ModificationDate=08-21-95
ModifiedBy=Zaphod Beeblebrox
```

12.5.2.2 General Device Information

The device specific information

- vendor name,
- vendor code,
- device name,
- device code,
- version information,
- LMT-information (parts of the LMT-address),
- device abilities.

can be found in the section `DeviceInfo`.

The following keywords are used:

`VendorName`	- vendor name (255 characters)
`VendorNumber`	- vendor code (Unsigned32)
`ProductName`	- product name (255 characters)
`ProductNumber`	- product number (Unsigned32)
`ProductVersion`	- product version (Unsigned8)
`ProductRevision`	- product revision (Unsigned8)
`OrderCode`	- order code for this product (255 characters)
`LMT_ManufacturerName`	- manufacturer name part of LMT-address (7 characters)
`LMT_ProductName`	- product name part of LMT-address (7 characters)
`BaudRate_10`	- supported baud rates (Boolean, 0 = not supported, 1 = supported)
`BaudRate_20`	
`BaudRate_50`	
`BaudRate_100`	
`BaudRate_125`	
`BaudRate_250`	
`BaudRate_500`	
`BaudRate_800`	
`BaudRate_1000`	
`SimpleBootUpMaster`	- simple boot-up master functionality (Boolean, 0 = not supported, 1 = supported),
`SimpleBootUpSlave`	- simple boot-up slave functionality (Boolean, 0 = not supported, 1 = supported),
`ExtendedBootUpMaster`	- extended boot-up master functionality (Boolean, 0 = not supported, 1 = supported),
`ExtendedBootUpSlave`	- simple boot-up slave functionality (Boolean, 0 = not supported, 1 = supported),
`Granularity`	- This value gives you the granularity allowed for the mapping on this device - most of the existing devices supports a granularity of 8 (Unsigned8; 0 - mapping not modifiable, 1-64 granularity)

Example:

```
[DeviceInfo]
VendorName=Nepp Ltd.
VendorNumber=156678
ProductName=E/A 64
ProductNumber=45570
ProductVersion=4
ProductRevision=1
OrderCode=BUY ME - 177/65/0815
LMT_ManufacturerName=NEPPLTD
LMT_ProductName=E/AXY64
BaudRate_50=1
BaudRate_250=1
BaudRate_500=1
BaudRate_1000=1
SimpleBootUpSlave=1
ExtendedBootUpMaster=0
SimpleBootUpMaster=0
ExtendedBootUpSlave=0
```

12.5.2.3 Object Dictionary

In this section of the EDS the following information can be found:
1. which object of the object dictionary is supported,
2. limit values for parameters,
3. default values.

The description of the objects take place in four separate sections corresponding to:
- data types,
- mandatory objects,
- optional objects,
- manufacturer specific objects.

12.5.2.3.1 Data type section

The section `StandardDataTypes` lists all standard data types available on the device. Data types are listed according to their index (table 11.1-1 in DS301). Additional the complex standard data types PDO Communication Parameter Record and PDO Mapping Record are known.

The entries follow this scheme:

```
<data type index>={0|1}
```

Example:

```
[StandardDataTypes]
0x0001=1
0x0002=1
0x0003=1
0x0004=1
0x0005=1
0x0006=1
0x0007=1
0x0008=0
0x0009=1
0x000A=0
0x000B=0
0x000C=0
0x000D=0
0x000E=0
0x000F=0
0x0020=1
0x0021=1
0x0022=0
```

12.5.2.3.2 Mapping of dummy entries

Sometimes it is required to leave holes in the mapping of a device. This means that e.g. a device only evaluates the last two data bytes of a PDO of 8 bytes length. The first six bytes should be ignored (perhaps they are evaluated by another device). In this case the mapping of this device must be created by using dummy entries for these first six bytes.

The indices from the data type area of the object dictionary are used for this purpose. The user of a device has to know what data type can be used for creating dummy entries and what not (indeed only the length of the dummy object is important).

The section `DummyUsage` is used for describing dummies. The entries follow this scheme:

```
Dummy<data type index (without 0x-prefix)>={0|1}
```

Example:

```
[DummyUsage]
Dummy0001=0
Dummy0002=1
Dummy0002=1
Dummy0003=1
Dummy0004=1
Dummy0005=1
Dummy0006=1
Dummy0007=1
```

This means that the device will tolerate the mapping of the data types Integer8/16/32 and Unsigned8/16/32.

12.5.2.3.3 Object sections

The section `MandatoryObjects` describes the mandatory objects of the object dictionary. There exists only the two entries `DeviceType` and `ErrorRegister`. In the section `MandatoryObjects` the following entries are possible:

`SupportedObjects` - number of entries in the section (Unsigned16)
The entries are numbered beginning with number 1. This way the last entry gives the number of available entries. In these sections the entries have the same values for all devices because there are no optional entries.

```
1=0x1000
2=0x1001
```

The entries of the particular objects of the object dictionary are all constructed following the same scheme. The section name is constructed according to the following:

```
[<Index>(sub<Sub-Index>)]
```

using the hexadecimal values for Index and Sub-index without the leading „`0x`“.

In a section the following keywords may exist:

`SubNumber`	- number of sub-indices available at this index (Unsigned8)
`ParameterName`	- parameter name (255 characters)
`ObjectType`	- This is the object code (DS301 table 11-2).
`DataType`	- This is the index of the data type of the object in the object dictionary (DS301 table 11.1-1).
`LowLimit`	- Lowest limit of object value (only if applicable).
`HighLimit`	- Upper limit of object value (only if applicable).
`AccessType`	- Access type for this object (String „ro“ - read only, „wo“ - write only, „rw“ - read/write, „rwr“ - read/write on process input, „rww“ - read/write on process output, „const“ - constant value,)
`DefaultValue`	- default value for this object,
`PDOMapping`	- default value for this object (Boolean, 0 = not mappable, 1 = mappable).

It is not necessary to use all keywords in every section.

case 1: An object can be accessed directly by an index. There are no sub-indices for this object

Used keywords:

```
[1000]
SubNumber=0
ParameterName=Device Type
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=
PDOMapping=0
```

Case 2: Parts of an object can be accessed via sub-indices. The entry in the EDS can be realised like the following:

```
[1003]
SubNumber=2
ParameterName=Pre-defined Error Field
```

The sub-index entries are defined in this manner:

```
[1003sub0]
ParameterName=Number of Errors
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0x1
PDOMapping=0

[1003sub1]
ParameterName=Standard Error Field
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=0x0
PDOMapping=0
```

Example - mandatory section:

```
;------------------------------------
[MandatoryObjects]
SupportedObjects=0x02
1=0x1000
2=0x1001

[1000]
SubNumber=0
ParameterName=Device Type
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=
PDOMapping=0

[1001]
SubNumber=0
ParameterName=Error Register
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0x0
PDOMapping=1
```

The section `OptionalObjects` describes the optional objects supported by the device.

```
;-----------------------------------
[OptionalObjects]
SupportedObjects=0x7
1=0x1003
2=0x1004
3=0x1005
4=0x1008
5=0x1009
6=0x100A
7=0x100B

[1003]
SubNumber=2
ParameterName=Pre-defined Error Field

[1003sub0]
ParameterName=Number of Errors
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=
PDOMapping=0

[1003sub1]
ParameterName=Standard Error Field
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=
PDOMapping=0

; value for object 1006 (communication cycle period)
; no sub-index available
[1006]
ParameterName=Communication Cycle Period
ObjectType=0x7
DataType=0x0007
LowLimit=1000
HighLimit=100000
AccessType=rw
DefaultValue=20000
PDOMapping=0

:
```

The section `ManufacturerObjects` describes the manufacturer specific entries in the object dictionary.

```
;-----------------------------------
[ManufacturerObjects]
SupportedObjects=0
```

12.5.2.3.4 Object Links

In order to ease the implementation of a configuration tool it is possible to group related objects together via the keyword `ObjectLinks`.

An object link has the following structure:

```
[<index>ObjectLinks]
ObjectLinks=<number of available links>
1=<index of 1st linked object>
2=<index of 2nd linked object>
3=<index of 3rd linked object>
4=<index of 4th linked object>
5=<index of 5th linked object>
:
```

Example:

```
; assuming we describe closed loop
; this is the object „factor‟
[5800ObjectLinks]
ObjectLinks=0x3
; gain
1=0x5801
; zero
2=0x5802
; pole
3=0x5803
```

12.5.2.4 Comments

Comments can be added to the EDS by using the `Comments` section. This section has only entries determining the line number and the line contents.

`Lines` - number of commentlines (Unsigned16)
`Line<line number>` - one line of comment (character 255)

Example:

```
[Comments]
Lines=3
Line1=|-------------|
Line2=| Don't panic |
Line3=|-------------|
```

12.5.3 Structure of a DCF (Device Configuration File)

The device configuration file comprises all objects for a configured device. The device configuration file has the same structure as the EDS for this device. There are some additional entries in order to describe e configured device.

12.5.3.1 Additional Entries

12.5.3.1.1 File Information Section

`LastEDS` - file name of the EDS file used as template for this DCF

12.5.3.1.2 Object Sections

`ParameterValue` - object value (as defined by `ObjectType` and `DataType`)

Example:

```
; value for object 1006 (communication cycle period)
[1006]
SubNumber=0
ParameterName=Communication Cycle Period
ObjectType=0x7
DataType=0x0007
LowLimit=1000
HighLimit=100000
DefaultValue=20000
AccessType=ro
ParameterValue=15000
PDOMapping=0
```

If the `ObjectType` is Domain (0x2) the value of the object can be stored in a file.

`UploadFile:`	if a read access is performed for this object, the data are stored in this file (character 255)
`DownloadFile:`	if a write access is performed for this object, the data to be written can be taken from this file (character 255)

Example:

```
; manufacturer specific object 5600 (downloadable program)
[5600]
SubNumber=0
ParameterName=Real Good Program (RGP)
ObjectType=0x2
DataType=0x000F
AccessType=wo
DownloadFile=C:\FAST\PROGRAMS\FIRST.HEX

; manufacturer specific object 5700 (core dump)
 [5700]
SubNumber=0
ParameterName=Core Dump (CD)
ObjectType=0x2
DataType=0x000F
AccessType=ro
UploadFile=C:\FAST\DEBUG\DUMPALL.HEX
```

12.5.3.1.3 Device Commissioning

There is an additional section in the DCF named `DeviceComissioning`:

`NodeID` - device's address (Unsigned8)

`NodeName` - node name part of NMT-address (7 characters)

`Baudrate` - device's baudrate (Unsigned16)

`NetNumber` - number of the network (Unsigned32)

`NetworkName` - name of the network (character 255)

`LMT_SerialNumber` - serial number part of LMT-address (14 characters [0-9])

Example:

```
[DeviceComissioning]
NodeID=2
NodeName=DEVICE2
Baudrate=1000
NetNumber=42
NetworkName=very important subnet in a big network
LMT_SerialNumber=33678909231354
```

Request for Comments
for CiA DS-301 V 3.0 CANopen Communication Profile

Introduction

Version 3.0 of the CANopen Communication Profile CiA DS-301 was frozen in October 1996 in order to provide a stable base for industrial implementations. It is not intended to change or enhance the standard in the near future.
However, practical experience with CANopen products encouraged the Interest Group CANopen within CiA to issue some recommendations for future implementations of the CANopen standard. As these recommendations are not part of the standard, their implementation is not verified in conformance testing. On the other hand, conformance testing will not fail if the recommendations are implemented.
Depending on the comments received, the recommendations may be included in future versions of the standard. Please address your comments to the

Interest Group CANopen,
c/o CiA Headquarters
Am Weichselgarten 26
D-91058 Erlangen
Fax +49-9131-69086-79
email headquarters@can-cia.de

There is a Fax Form for your feedback at the end of this document.

RfC 1: Mandatory Life Guarding

Question
The support of the guarding message in CANopen is not mandatory. This is no drawback if there is a device which communicates in a cyclic way. A cyclic PDO could be taken as some kind of heart beat for the device.
But how should a network deal with devices which ”only” support acyclic PDOs? The PDO may be sent very seldom - for instance only in the case of emergency (e.g. a limit switch).
To implement a network without consistency check is very insecure. Of course there is always a workaround possible but CANopen devices should provide at least the simplest security mechanisms.
Recommendation
Support of node guarding and life guarding is recommended for all devices, especially for those supporting other PDO transmission types than 1-240.

RfC 2: Boot-Up Message

Question
Assuming a network with node guarding. A slave is guarded by the NMT Master. Now the slave is resetted by a local power fail. A slave will continue the guarding with the toggle bit set to 0 after the power is stable enough and initialization is finished. The NMT master connected to the system will not recognize the failure under the following circumstances:
- the slave was in the state PRE-OPERATIONAL (default state after boot-up),
- the last toggle bit sent from the slave was set to 1,
- the boot-up time of the slave is shorter than the guard time.
Recommendation
Every slave should send a boot-up message after power-on. As the default emergency identifier is available on most devices, the boot-up message uses this identifier and has no data bytes. The boot up message is sent after initialization. This allows one to retrieve the sending node directly from the used COB-ID.

RfC 3: Mandatory sections Electronic Data Sheet

Question
Which sections and keywords in the EDS description are optional?
Answer
There are no optional keywords and sections in the EDS description. This means that every keyword must be listed in the right section. If a keyword is not used the value is NONE (CR/LF after EQUAL sign).
Exception:
For objects with a data type > 0x0008 it is not necessary to support
- DefaultValue,
- HighLimit,
- LowLimit.

Reason
It is necessary to have one format with a basic set of entries in order to support the development of standardized tools. Furthermore it is very important for a certification procedure to rely on a fixed format.

RfC 4: Manufacturer and Profile Specific Keywords in the EDS

The EDS is often used for vendor specific entries. This means that a lot of vendors have added their own sections with their own entries. It is not allowed to add manufacturer specific entries in the sections specified in the DS-301. But it is allowed to create manufacturer or device specific sections with specific entries in it. The manufacturer or the SIG must ensure that his keywords don't collide with the keywords listed is the DS-301.

The section `ManufacturerSections` defines the keywords used for manufacturer specific section entries.

```
[ManufacturerSections]
SectionNumber=4
1=FirstManufactSection
2=SecondManufactSection
3=ThirdManufactSection
4=FourthManufactSection
```

The section `ManufacturerEntries` defines the keywords used for manufacturer specific entries in the manufacturer's sections.

```
[ManufacturerEntries]
EntriesNumber=3
1=ThisEntry
2=ThatEntry
3=OnlyEntry
```

In the same way it is possible to define device specific EDS entries. These entries have to be defined by the Special Interest Groups (SIG).

The section `DeviceSections` defines the keywords used for device specific section entries.

```
[DeviceSections]
SectionNumber=4
1=FirstDevSection
2=SecondDevSection
3=ThirdDevSection
4=FourthDevSection
```

The section `DeviceEntries` defines the keywords used for device specific entries in the device's sections.

```
[DeviceEntries]
```

```
EntriesNumber=3
1=ThisEntry
2=ThatEntry
3=OnlyEntry
```

The numbers in the sections are decimal coded.

RfC 5: Restore Defaults

Question
Does a device have to support the ”restore default parameters”-feature (Object 1011h)?
Recommendation
The support of this object is optional. However, if a device supports storing of parameters in non-volatile memory (object 1010h) it is strongly recommended that it supports ”restore default parameters” as well. For test purposes it is necessary to set a device in a ”default setting state”. Otherwise it is not possible to perform a static test for a device.

RfC 6: Gaps in arrays

Question: Is it allowed to leave gaps in arrays?
Functional related objects of devices often are gathered in arrays. Is it allowed to implement only the required objects? Consider for example the object 1010H ,,Store Parameters". If a device shall implement Sub-Index 3H ”Save Application Parameters”, but is not able to store all parameters of Sub-Index 1H and 2H, what has to be done ?

Answer:
It is allowed to leave the Sub-Indexes IH and 2H of that example empty. This reduces effort and module resources. Another example is the establishment of dynamic variables in programmable devices (refer to WD-302 framework for Programmable Device) The mechanisms used there would waste many Kbytes of memory and some 100 Kbytes (!) of DCF file size, if gaps were not allowed.

The length element on Sub-index OH has to store the highest index implemented. In the example 1010H above this would be 3H. In the EDS/DCF the entry SubNumber has to store the number or sub-object - in the example this is 2 (Sub-Index 0H and Sub-Index 3H).

RfC 7: Error codes in the abort domain protocol (SDO)

As described in the DS-301 the application error code returned by the abort domain service is a 32bit value. Unfortunately there are not enough error reasons supported. Furthermore the description of the error codes is very confusing. The codes are derived from the Profibus document. But some of the Profibus ”features” are not present at CANopen devices. Generally the error codes deal only with object dictionary specific errors. The following table shows all the known error codes as a 32bit integer value (hexadecimal coded). This 32bit value is transmitted according to the encoding rules. Differences with respect to the DS-301 are marked with *.

application error codes	description
0x05030000	Toggle bit not alternated.
0x05040000	Time out value reached.
0x05040001	Client/server command specifier not valid or unknown.*
0x06010000	Attempt to read a write only object.
0x06010001	Attempt to write a read only object.
0x06020000	Object does not exist in the object dictionary.
0x06040000	The index is reserved for further use (index values from 00A0h-0FFFh and A000h-FFFFh).
0x06040041	Object cannot be mapped to the PDO. *
0x06040042	The number and length of the objects to be mapped would exceed PDO length. *
0x06040043	General parameter incompatibility reason.
0x06040047	General internal incompatibility in the device.
0x06060000	Access failed because of an hardware error.
0x06070010	Data type does not match, length of service parameter does not match
0x06070012	Data type does not match, length of service parameter too high
0x06070013	Data type does not match, length of service parameter too low
0x06090011	Sub-index does not exist.
0x06090030	Value range of parameter exceeded (only for write access).
0x06090031	Value of parameter written too high.
0x06090032	Value of parameter written too low.
0x06090036	Maximum value is less than minimum value.
0x08000020	Data cannot be transferred or stored to the application.
0x08000021	Data cannot be transferred or stored to the application because of local control.

0x08000022	Data cannot be transferred or stored to the application because of the present device state.
0x08000023	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error). *

Fax
to CiA Interest Group CANopen
c/o CAN in Automation, International Headquarters, D-91058 Erlangen
Fax +49-9131-69086-79

Feedback regarding Requests for Comments for CiA DS 301 V 3.0

from:

Name ______

Company ______

Address ______

______

Phone ______

Fax ______

Comment regarding RfC No.__: