CHIEFTEK PRECISION CO., LTD.

cpc TC1/TC1-B Servo Drive

CiA 301 CANopen

Communication Manual

Revision 1.0

Table of Contents

1. About This Manual

1.1. Revision History

Revision	Date	Description	Remarks
1.0	March, 2018	Initial release	--

1.2. Disclaimer

1. Information furnished by cpc is believed to be accurate and reliable. However, no responsibility is assumed by cpc for its use, nor for any infringements of patents or other rights of third parties which may result from its use. cpc doesn't grant any license under its patent rights, nor the rights of others.
2. In addition, cpc assumes no responsibility for any errors that may appear in this document and for any claims or damages arising from information contained in this document.
3. The product specified in this document has been developed, produced, tested and documented in accordance with the relevant standards. cpc is not responsible for damages, accidents, or injuries caused by any deviation from the configuration and installation described in this guide;
4. Furthermore, cpc is not responsible for the performance of new measurements or ensuring that regulatory requirements are met.
5. The product specified in this document is not assumed to be used in critical application including, but not limited to, medical equipment, transportation, aerospace and nuclear instruments, undersea equipment, power plant equipment, as well as disaster prevention and crime prevention equipment.
6. We reserve the right to modify our products, including its hardware and software design, in order to improve its design and/or performance. The information in this document is subject to change without notice and does not represent a commitment by cpc.
7. Specifications are subject to change without notice.
8. Performance specification beyond those specified by safety regulations are guaranteed by design and not subject to production test.
9. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.
10. cpc assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using cpc products.

1.3. Contact Us

Headquarters

Chieftek Precision Co., Ltd.

NO.3, Dali 1st Rd., Xinshi Dist., Southern Taiwan Science Park,

Tainan City. 741-45, Taiwan (R.O.C.) 3

TEL: +886-6-505-5858

FAX: +886-6-505-5959

Email : service@mail.chieftek.com

China

Chieftek Machinery Kunshan Co., Ltd. ()

No.1188, Hongqiao Rd, Kunshan, Jiangsu, P.R. China 1186

Tel : +86-512-55252831

Fax : +86-512-55252851

Email : cn.service@mail.chieftek.com

Europe

cpc Europa GmbH

Industriepark 314, D-78244 Gottmadingen, Germany

Tel : +49-7731-59130-38

Fax : +49-7731-59130-28

Email : info@cpc-europa.de

USA

Chieftek Precision USA Co., Ltd.

4881 Murietta Street. Chino, CA. 91710

TEL: +1-909-628-9300

FAX: +1-909-628-7171

Email : info@usa.chieftek.com

2. Introduction

This manual repeats certain CiA standards and includes cpc-specific information. It explains how to implement CiA 301 communication with cpc's TC1/TC1-B servo drive. Most of the cpc drive communication functionality is standard, based on CiA 301 CANopen, version 4.2.0.

CAN communication:
CAN is message based, prioritizing messages is hence necessary. CAN allows any node to receive all transmitted messages, decide relevance and act (or ignore) accordingly. The CAN devices send/receive data across the CAN network in packets called "frames".

- **CAN Frame**:
  - An entire CAN data transmission (also referred to as a CAN message) consisting of CAN-ID, RTR, data length, and the actual data.
  - Comes in two formats according to CiA 301:
    - Standard addressing, using an 11-bit CAN-ID;
    - Extended addressing, using a 29-bit CAN-ID.

      Note:

      Extended addressing is not implemented.

- **CAN-ID**:
  - Consists of a 4-bit function code indicating the message's priority (the lower the value, the higher the priority), and of a 7-bit Node-ID.
  - It is necessary that users (or master device) assign unique Node-ID for each device respectively in the network to enable data transmission.

  See further details about CAN-ID in chapter 3.2.

To set up Node-ID please refer to **chapter 13** (Initial CAN Communication Setup);
To download EDS please refer to **chapter 11** (The EDS).

Note:

The listed functionalities below are **not** implemented in the cpc drives.

- RTR (Remote Transmission Request)
- Service SDO block transfer
- Service node guarding
- Service life guarding
- Multiplex PDO (MPDO)

and

- The following objects:
  - Object 0x1007: Synchronous window length
  - Object 0x100C: Guard time
  - Object 0x100D: Life time factor
  - Object 0x1012: COB-ID time stamp object
  - Object 0x1020: Verify configuration
  - Object 0x1023: OS command
  - Object 0x1024: OS command mode
  - Object 0x1025: OS debugger interface
  - Object 0x1026: OS prompt
  - Object 0x1027: Module list
  - Object 0x1028: Emergency consumer object
  - Object 0x1029: Error behavior object
  - Object 0x1280 to 0x12FF: SDO client parameter
  - Object 0x1FA0 to 0x1FCF: Object scanner list
  - Object 0x1FD0 to 0x1FFF: Object dispatching list

2.1. Abbreviations and Terms

Abbreviations	Definition
CAN	Controller area network
CAN base frame	message that contains up to 8 byte and is identified by 11 bits as defined in /ISO11898-1/
CAN extended frame	message that contains up to 8 byte and is identified by 29 bits as defined in /ISO11898-1/
CAN-ID	CAN identifier – identifier for CAN data.
COB	Communication object – identifier that contains the CAN-ID and additional control bits.
COB-ID	COB identifier
CSDO	Client-SDO
EMCY	(referring to Emergency Object)
Entity	particular thing, such as a person, place, process, concept, association, or event.
FSA	Finite state automaton – model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state; computation begins in the start state with an input string; it changes to new states depending on the transition function.
Logical device	representation of a field device in terms of its objects and behavior according to a field device model that describes the device's data and behavior as viewed through a network
LSB	Least significant bit/byte
MPDO	Multiplexed-PDO
MSB	Most significant bit/byte
NMT	Network management

Abbreviations	Definition
Node-ID	Node identifier – network-wide unique identifier for each CANopen device.
Object	entity with a well-defined boundary and identity that encapsulates state and behavior
PDO	Process data object
PLS	Physical layer signaling
PMA	Physical medium attachment
RPDO	Receive-PDO
RTR	Remote transmission request
SDO	Service data object
SSDO	Server-SDO
SYNC	Synchronization object
TPDO	Transmit-PDO

3. CANopen Basics

This chapter explains, in general, those CANopen communication features most related to the cpc driver.

3.1. Physical Layer

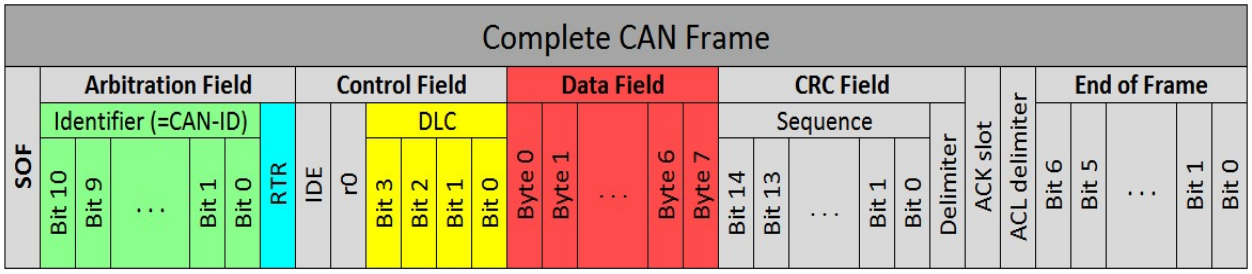
CAN is a serial communication standard in which the transferred data is coded as electrical pulses on a two-wire communication line.

3.2. CAN-ID

3.2.1. CANopen (Base) Frame Format

The format of a CANopen message is shown as below. The most relevant fields to users are: Identifier (i.e. CAN-ID), RTR, Data length code, and Data field.

<CANopen Frame in Format>



Explanation:

Field name	Length (bits)	Purpose
Start-of-frame (SOF)	1	Denotes the start of frame transmission
Identifier (= CAN-ID)	11	A unique identifier which also represents the message priority
Remote transmission request (RTR)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames.
Identifier extension bit (IDE)	1	Must be dominant (0) for base frame format with 11-bit identifiers
Reserved bit (r0)	1	Reserved bit. Must be dominant (0), but accepted as either dominant or recessive.
Data length code (DLC)	4	Number of bytes of data (0–8 bytes)
Data field	0–64 (0-8 bytes)	Data to be transmitted (length in bytes dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

Note:

- In CANopen, the 11-bit CAN-ID is split into two parts:
  - a 4-bit **function code**, and
  - a 7-bit CANopen **node ID**. The 7-bit size limits the number of devices on a CANopen network to 127 nodes.
- The **COB-ID** is composed of the CAN-ID bits and additional control bits.

3.2.2. Generic pre-defined connection set

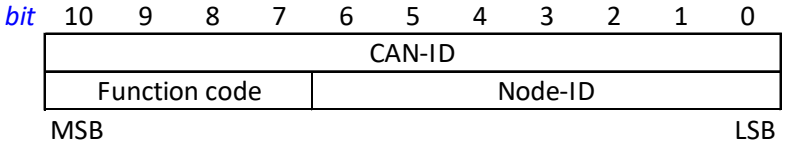
In order to reduce configuration effort for simple networks a CAN-ID allocation scheme is defined. These CAN-IDs are available in the NMT state Pre-operational directly after the NMT state Initialisation (if no modifications have been stored).
The objects SYNC, TIME, EMCY write and PDO can be deleted and re-created with new CAN-IDs when a master device designates new CAN-IDs to them.

The **CAN-ID-allocation scheme**:

- Consists of
  - a 4-bit **functional code**, which determines the object **priority**, and
  - a 7-bit **Node-ID**, which allows to distinguish between CANopen devices of the same functionality.

  This scheme allows **a peer-to-peer communication between a single master CANopen device and up to 127 NMT slave CANopen devices**.

<CAN-ID-allocation scheme for the generic pre-defined connection>



- Supports the broadcasting of non-confirmed NMT, SYNC and TIME messages. **Broadcasting** is indicated by a **Node-ID of zero**.

- The generic pre-defined connection set supports: one EMCY object, one SDO, maximum 4 RPDOs and 4 TPDOs, and the NMT objects.

The two tables below show the supported objects and their allocated CAN-IDs.

<Broadcast objects of the generic pre-defined connection set>

COB Type	Function code (bit 7-10 of CAN-ID)	resulting CAN-ID
NMT	0000_b	0 (000_h)
SYNC	0001_b	128 (080_h)
TIME	0010_b	256 (100_h)

<Peer-to-peer objects of the generic pre-defined connection set>

COB Type	Function code (bit 7-10 of CAN-ID)	resulting CAN-IDs
EMCY	0000_b	129 (081h) – 255 (0FFh)
PDO1 (tx)	0011_b	385 (181h) – 511 (1FFh)
PDO1 (rx)	0100_b	513 (201h) – 639 (27Fh)
PDO2 (tx)	0101_b	641 (281h) – 767 (2FFh)
PDO2 (rx)	0110_b	769 (301h) – 895 (37Fh)
PDO3 (tx)	0111_b	897 (381h) – 1023 (3FFh)
PDO3 (rx)	1000_b	1025 (401h) – 1151 (47Fh)
PDO4 (tx)	1001_b	1153 (481h) – 1279 (4FFh)
PDO4 (rx)	1010_b	1281 (501h) – 1407 (57Fh)
SDO (tx)	1011_b	1409 (581h) – 1535 (5FFh)
SDO (rx)	1100_b	1537 (601h) – 1663 (67Fh)
NMT error control	1110_b	1793 (701h) – 1919 (77Fh)

3.2.3. Restricted CAN-IDs

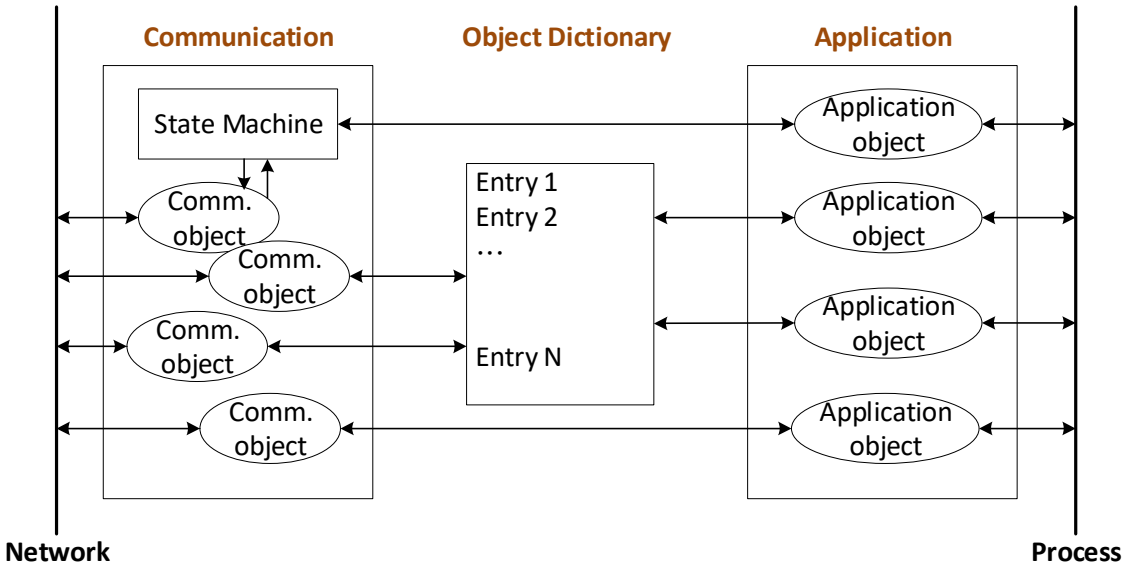
Any CAN-ID listed below is of restricted use and cannot be used by configurable communication objects, neither by SYNC, TIME, EMCY, and SDO.

<Restricted CAN-IDs>

CAN-IDs	used by COB
0 (000h)	NMT
1 (001h) – 127 (07Fh)	reserved
257 (101h) – 384 (180h)	reserved
1409 (581h) – 1535 (5FFh)	default SDO (tx)
1537 (601h) – 1663 (67Fh)	default SDO (rx)
1760 (6E0h) – 1791 (6FFh)	reserved
1793 (701h) – 1919 (77Fh)	NMT Error Control
2020 (780h) – 2047 (7FFh)	reserved

3.3. CANopen Device Model

A CANopen device is structured as the diagram below.



- **Communication** – This function unit provides the communication objects and the proper functionality to **transport data items** via the underlying network structure.
- **Object dictionary** – The object dictionary is **a collection of all the data** items which have an influence on the behavior of the application objects, the communication objects and the state machine used on this device.
- **Application** – The application comprises **the functionality of the device** with respect to the **interaction** with the process environment.

3.4. CANopen Application Layer

The application layer is an **abstraction layer**.

- describes a concept to configure and communicate real-time data as well as the mechanisms for synchronization between CANopen devices.
- specifies the shared communications protocols and interface methods used by hosts in a communications network.

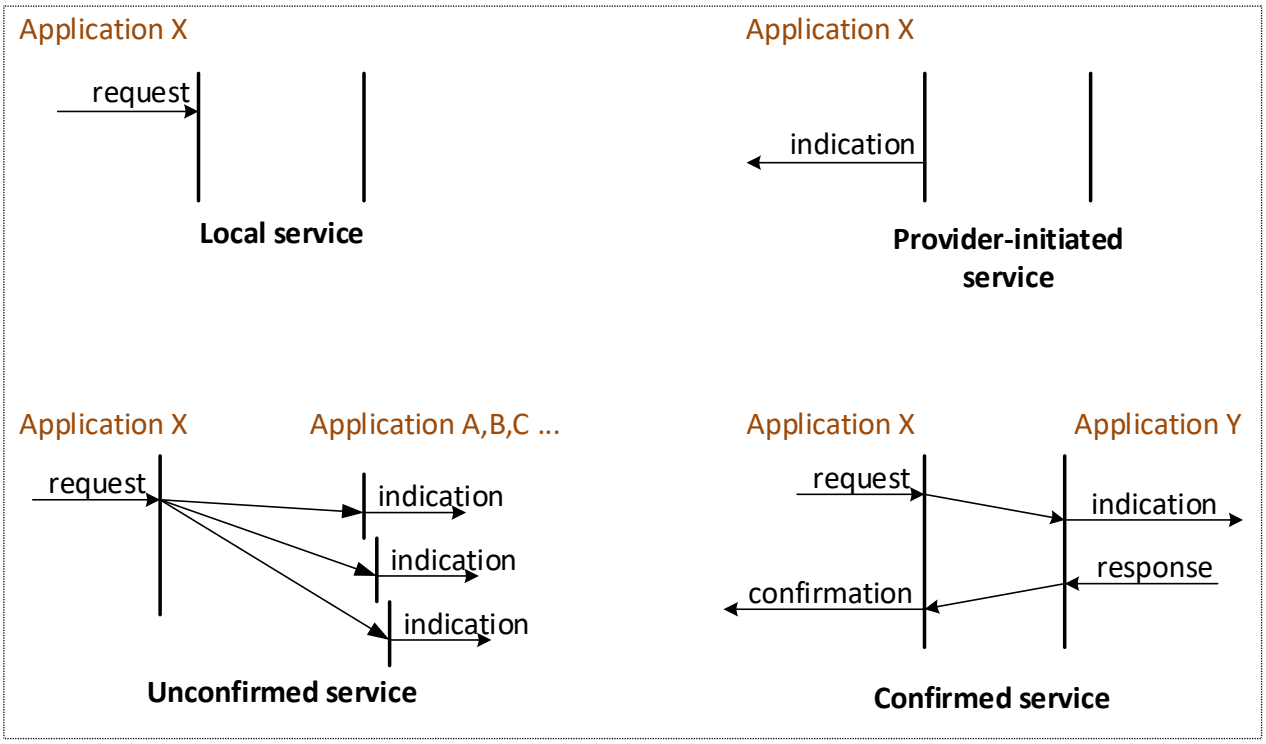
3.4.1. Service Primitives

Service primitives are **the methods by which the application and the application layer interact**. There are 4 types:

Service Type	issued by:	to the:	Behavior
Request	application	application layer	To request a service
Indication	application layer	Application	To **report** an internal event detected by the application layer, OR **to indicate that a service is requested.**
Response	application	application layer	To respond to a previously received indication.
Confirmation	application layer	application	To **report the result** of a previously issued request.

3.4.2. Application Layer Services

<Application Layer Services>



An application layer service defines the primitives that are exchanged between the application layer and the co-operating applications for a particular service of a service object.

The application layer services supported by CANopen are shown as the picture above. See explanations on next page.

- A local service involves only the local service object. The application issues a request to its local service object that executes the requested service without communicating with (a) peer service object(s).
- A provider-initiated service involves only the local service object. The service object (being the service provider) detects an event not solicited by a requested service. This event is then indicated to the application.
- An unconfirmed service involves one or more peer service objects. The application issues a request to its local service object. This request is transferred to the peer service object(s) that each passes it to their application as an indication. The result is not confirmed back.
- A confirmed service involves **only one peer** service object. The application issues a request to its local service object. This request is transferred to the peer service object that passes it to the other application as an indication. The other application issues a response that is transferred to the originating service object that passes it as a confirmation to the requesting application.

Note:
Unconfirmed and confirmed services are collectively called **remote services**.

3.5. Communication Protocol Sequences

3.5.1. General

The communication protocol defines:

- The principles of each communication protocol, and
- The available modes of message transmission triggering.

➢ There are 2 types of communication protocol models:

- **Producer/Consumer protocol** (see chapter 3.5.2.)
  And subcategory Master/Slave protocol
- **Client/Server protocol** (see sub-clause 3.5.3.)

➢ The CANopen communication protocol supports the transmission of

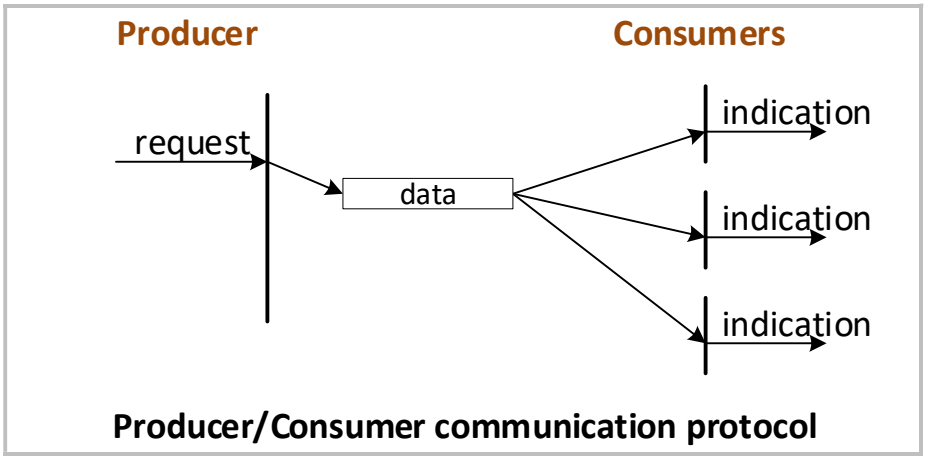
- synchronous message – transmitted at the pre-defined timing
- event-driven message – transmitted at any time.
  (See **Chapters 7.1 and 7.2**).

Through the synchronous message transmission, it is possible to acquire and activate data via network wide coordination

3.5.2. Producer/consumer protocol

The producer/consumer protocol involves one producer and **several consumers.**

This model is featured with an **unconfirmed** protocol requested by the producer. **Any CANopen device can be the request producer, the rest devices (consumers) can choose to enable/disable the message.** For example, services using this producer/consumer protocol: PDO and EMCY.



Producer/Consumer communication protocol

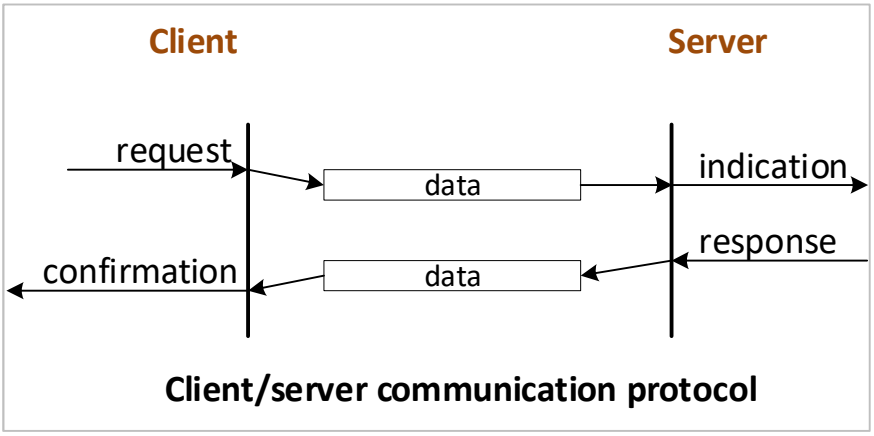
There is a subcategory under the producer/consumer category – **Master/slave protocol:** At any time, there is exactly **one** CANopen device serving as a master. All other CANopen devices in the network are considered as slaves. **The master initiates** a request and the salves will respond (if the protocol requires this behavior).

Note:
Some services can be initiated **only by Master device**, such as SYNC, TIME, and NMT state services.

3.5.3. Client/server protocol

This is a communication protocol used between **a single client and a single server.**

A client issues a request (upload/download) hence triggering the server to execute a certain task. After completing the task, the server answers the request.



Client/server communication protocol

3.6. Communication Objects

The data-byte units transported through a CAN network are called communication objects (COBs). The cpc TC1/TC1-B servo drive uses the following COB types:

COB Type	Description
Service Data Object (SDO)	SDO messages are used to manipulate OD objects according to their IDs. The server receives the SDO, which specifies in its message which object is to be dealt with. Large data set can be transferred using a chain of segmented SDO messages.
Process Data Object (PDO)	PDO messages are used to manipulate OD objects without explicit reference to the object identifier, which is possible if there is an a-priori convention concerning the OD item referenced. Such conventions are called "PDO mappings"; these are actually OD objects themselves and may be defined and manipulated using an SDO.
Emergency (EMCY)	Emergency messages are used by the servo drives to warn of an exception. The EMCY is the only COB type that a servo drive may transmit without first being explicitly asked. EMCY objects are similar to servo drive "interrupts": they eliminate the need to poll the servo drive continuously for the emergency status.
Network Management (NMT)	NMT objects are used by CAN clients to initialize a servo drive as a server.

3.7. Object Dictionary

The object dictionary is a collection of objects accessible via the network in an ordered predefined fashion. Each object within the object dictionary is addressed using a 16-bit index and an 8-bit sub-index. See further details in **chapter 4.**

3.8. RTR – Remote Transmission Request

The RTR function is not implemented in cpc drives.

3.9. Inhibit Time

The inhibit time for a given message type is the minimum time that must elapse from the message is first transmitted until the time that it is transmitted again.

The purpose of inhibit time is to prevent that the communication objects with lower priority are cut out from net source by those with higher priority. Configuring inhibit time avoids the possibility that the lower-priority communication objects being not able to transmit messages.

3.10. Data Types

To be able to exchange meaningful data across the network, it is necessary that the **format** of this data and its meaning is known by the producer and consumer(s). This specification models this by the concept of data types.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. **Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (bytes)**. For numerical data types the encoding is little endian.

There are also some general “extended” data types such as “Visible String” or “Time of Day” for example (see **chapter 3.10.4.** and **chapter 3.10.5.**).

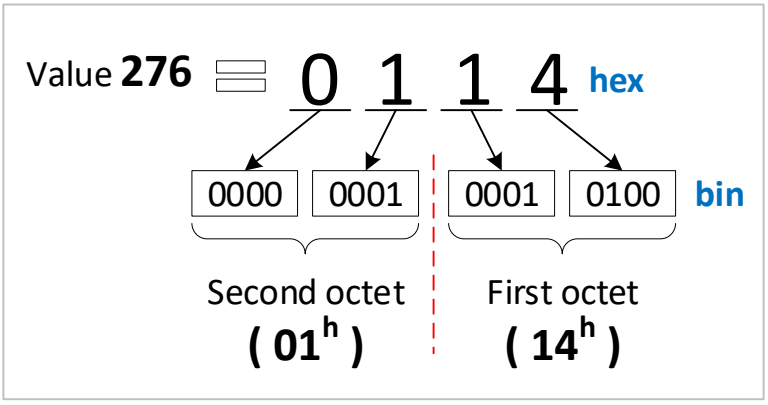
In the following sections, we will describe Unsigned integer, Signed integer, Floating-point numbers, Visible string, and Time of day.

3.10.1. Unsigned Integer

Data of UNSIGNEDn has values in the non-negative integers. The value range is 0, …, 2^{n-1}. The data is shown as bit sequences of length n.

For example:

The value 276 = 114^h (supposed with UNSIGNED16 data type format), 114^h is transferred in two octets across the bus, first 14^h and then 01^h.



<Transfer syntax for data type UNSIGNEDn>

octet number	1	2	3	4	5	6	7	8
UNSINGED8	b7…b0							
UNSINGED16	b7…b0	b15…b8						
UNSINGED32	b7…b0	b15…b8	b23…b16	b31…b24				
UNSINGED64	b7…b0	b15…b8	b23…b16	b31…b24	b39…b32	b47…b40	b55…b48	b63…b56

3.10.2. Signed Integer

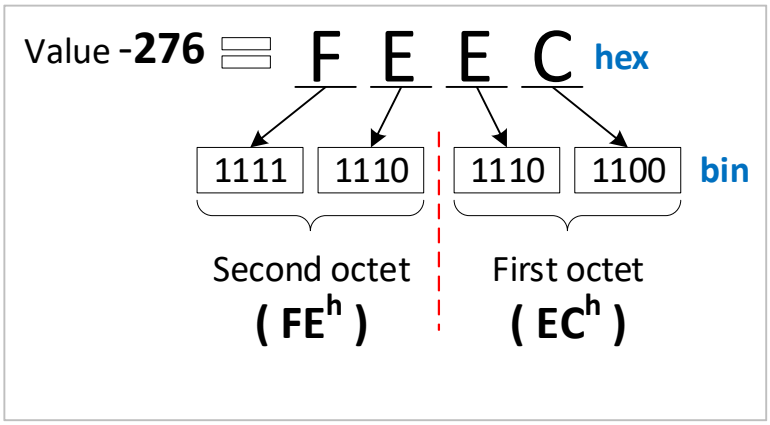
Data of INTEGERn has values in the integers.

The value range is from -2^{n-1} to $2^{n-1}-1$, including 0.

The data is represented as bit sequences of length n.

For example:

The value -276 = FEEC^h (supposed with SIGNED16 data type format), FEEC^h is transferred in two octets across the bus, first EC^h and then FE^h.



<Transfer syntax for data type INTEGERn>

octet number	1	2	3	4	5	6	7	8
UNSINGED8	b7…b0							
UNSINGED16	b7…b0	b15…b8						
UNSINGED32	b7…b0	b15…b8	b23…b16	b31…b24				
UNSINGED64	b7…b0	b15…b8	b23…b16	b31…b24	b39…b32	b47…b40	b55…b48	b63…b56

3.10.3. Floating-Point Numbers

Data of basic data types REAL32 and REAL64 have values in the real numbers.
The data type REAL32 is represented as bit sequence of length 32. The encoding of values follows /IEEE754/.
The data type REAL64 is represented as bit sequence of length 64. The encoding of values follows /IEEE754/.

3.10.4. Visible String

The data type VISIBLE_STRINGlength is defined below. The admissible values of data of type VISIBLE_CHAR are 0h and the range from 20h to 7Eh. The data are interpreted as ISO 646-1973(E) 7-bit coded characters. length is the length of the visible string.

UNSIGNED8	VISIBLE_CHAR
ARRAY [length] OF VISIBLE_CHAR	VISIBLE_STRINGlength

There is no 0h necessary to terminate the string.

3.10.5. Time of Day

The data type TIME_OF_DAY represents absolute time.
It follows from the definition and the encoding rules that TIME_OF_DAY is represented as bit sequence of length 48.
Component **ms** is the time in milliseconds after midnight. Component days is the number of days since January 1, 1984.

```
STRUCT OF
    UNSIGNED28 ms,
    VOID4 reserved,
    UNSIGNED16 days
TIME_OF_DAY
```

3.10.6. Domain

Domains are used to transfer an arbitrary large block of data from a client to a server and vice versa.

4. The Object Dictionary

4.1. Dictionary General Structure

The overall layout of the standard object dictionary is shown in the table below.

<Object Dictionary Structure>

Index	Object
0x1000 - 0x1FFF	Communication profile area - contains the communication specific parameters. - common to all CANopen devices.
0x2000 – 0x5FFF	Manufacturer-specific profile area. (manufacturer-specific functionality.)
0x6000 – 0x67FF	Standardized profile area 1st logical device

4.2. Object Code Usage

"Object code" represents what kind of object is at that specific index within the object dictionary.

Object code	Object kind	Description
00^h	NULL	An object with no data fields.
02^h	DOMAIN	Large variable amount of data; e.g. executable program code
05^h	DEFTYPE	Denotes a data type; e.g. INTERGER8, DOMAIN, and FLOAT.
07^h	VAR	A single value; e.g. an UNSIGNED8, a FLOAT, or an INTERGER8.
08^h	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. • Its sub-index 0 is of UNSIGNED8 and therefore not part of the ARRAY data.
09^h	RECORD	A multiple data field object where the data fields may be any combination of simple variables. • Its sub-index 0 is of UNSIGNED8 and sub-index 255 is of UNSIGNED32 and therefore not part of the RECORD data.

4.3. Access Usage

The “attribute” in the table below defines the access rights of a particular object. The view point is **from network into CANopen device**.

Note:
Even with READ or WRITE access right, some objects cannot be configured during motor on.

<Access attributes for data objects>

Attribute	Description
rw	read and write access
wo	write only access
ro	read only access
const	read only access, value is constant. The value may change in NMT state Initialisation. The value won’t change in the NMT states pre-operation, operational and stopped.

4.4. Format of Object and Entry

The formats of object description and entry description are as follows.

<Format of Object Description>

Object description

Index	Profile index number
Name	Name of object
Object code	Variable classification (i.e., NULL, DOMAIN, DEFTYPE, VAR, ARRAY, and RECORD)
Data type	Data type classification

<Format of Entry Description>

Entry description

Sub-index	Number of the sub- being described
Description	Descriptive name of the sub-index (field only used for arrays, records and structures)
Access	Read only (ro) or read/write (rw) or write only (wo) or const (c)
PDO mapping	defines if this object should be mapped to a PDO. Description: Default: Object is part of the default mapping (see device profile or application profile). TPDO: Object can be mapped into a TPDO and cannot be mapped into a RPDO. RPDO: Object can be mapped into a RPDO and cannot not be mapped into a TPDO. No: Object should not be mapped into a PDO.
Value range	range of possible values, or name of data type for full range
Default value	No: No default value applicable. Profile-specific: Default value of an object should be defined in a profile. Value: default value of an object after CANopen device initialisation

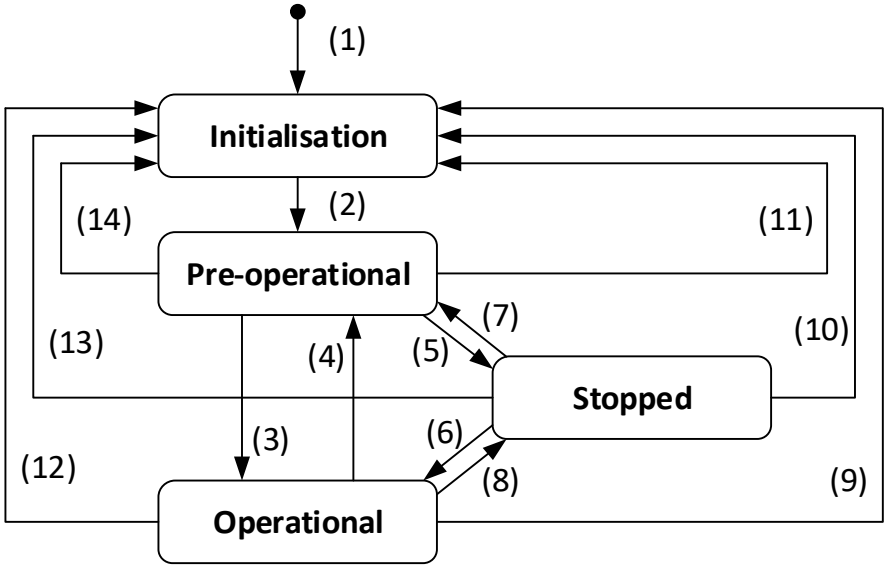
5. Network Management (NMT)

The network management (NMT) is CANopen device oriented and follows a master-slave structure. All CANopen devices are regarded as NMT slaves.
An NMT slave is uniquely identified in the network by its Node-ID, a value in the range of [1 to 127].

5.1. NMT State Machine

The diagram below shows the mechanism of NMT state machine. See the table below for further description of transitions.

<NMT state diagram of a CANopen device>



(1)	At Power on the NMT state initialization is entered
(2)	NMT state Initialization is finished, then enter NMT state Pre-operational automatically
(3)	NMT service start remote node indication or by local control
(4), (7)	NMT service enter pre-operational indication
(5), (8)	NMT service stop remote node indication
(6)	NMT service start remote node indication
(9), (10), (11)	NMT service reset node indication
(12), (13), (14)	NMT service reset communication indication

5.1.1. NMT State Transitions

NMT state transitions are caused by:

- Receipt of an NMT service used for node control services.
- Hardware reset, or
- node control services locally initiated by application events, defined by device profiles and application profiles

5.1.2. NMT States

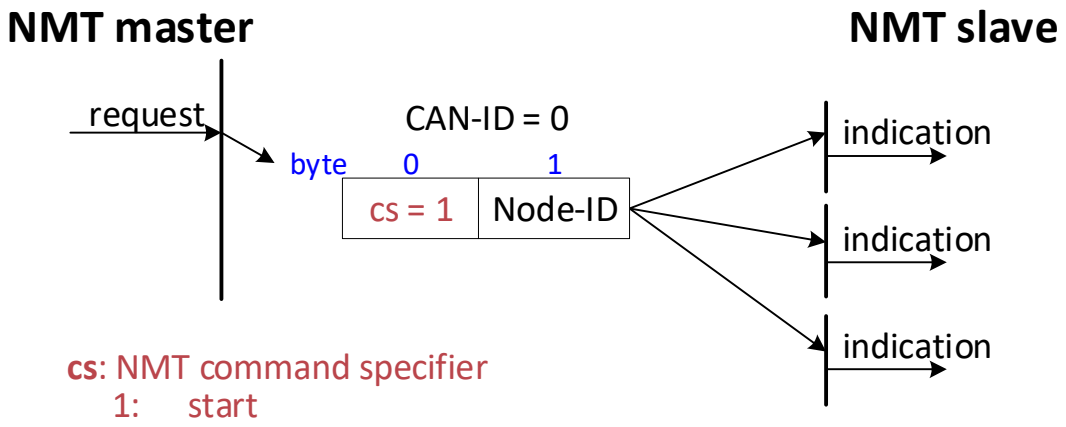
State	Description
Initialization	• Servo drive is not ready, or it is booting. Drive will not respond to communication and will not transmit anything.
Pre-operational	• CANopen devices enter this NMT state Pre-operational directly after finishing the CANopen devices initialization. • PDOs do NOT exist; PDO communication is NOT allowed. • Communication via SDOs is allowed. • Via SDOs, CANopen device is parameterized and CAN-IDs are allocated.
Operational	• All communication objects are active.
Stopped	• By switching a CANopen device into this NMT state Stopped, it is forced to **stop the communication altogether (except heartbeat, if active)**. • If there are EMCY messages triggered in this NMT state, they will be pending. The most recent active EMCY reason will be transmitted after the CANopen device transits into another NMT state. **Note**: The error history can be read by accessing the object 1003h.

5.2. NMT protocols

5.2.1. Node Control Protocols

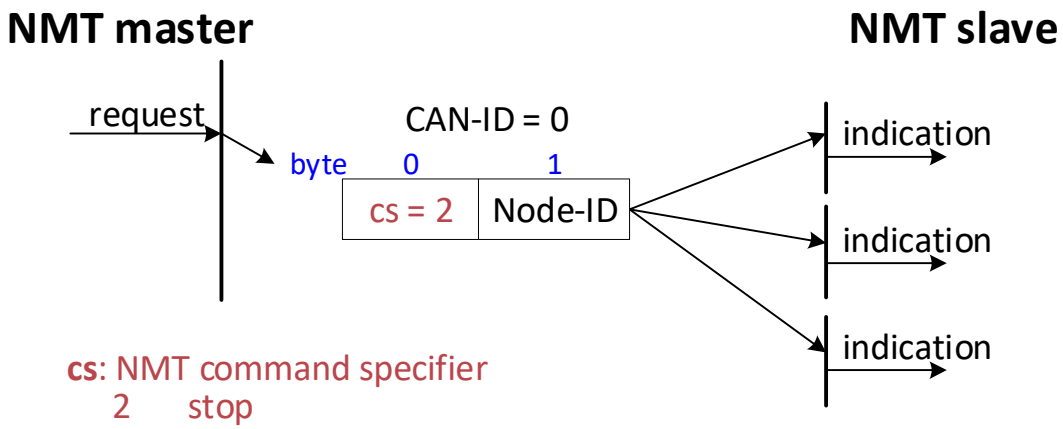
5.2.1.1. Protocol – start remote node

The protocol as depicted in the figure below is used to implement the NMT service – start remote node.



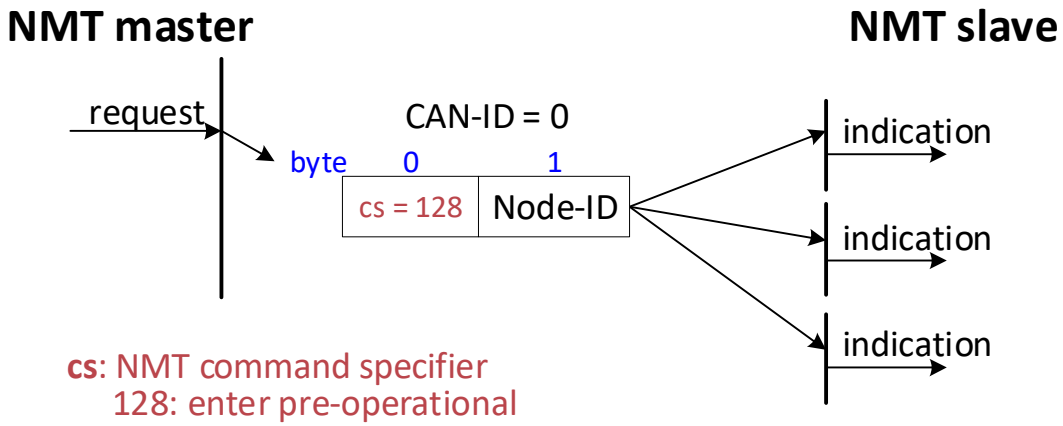
5.2.1.2. Protocol – stop remote node

The protocol as depicted in the figure below is used to implement the NMT service – stop remote node.



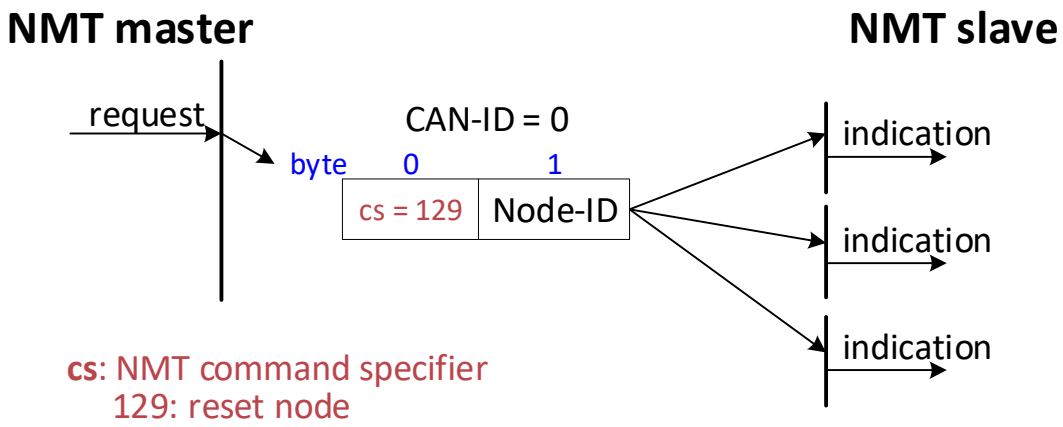
5.2.1.3. Protocol – enter pre-operational

The protocol as depicted in the figure below is used to implement the NMT service – enter pre-operational.



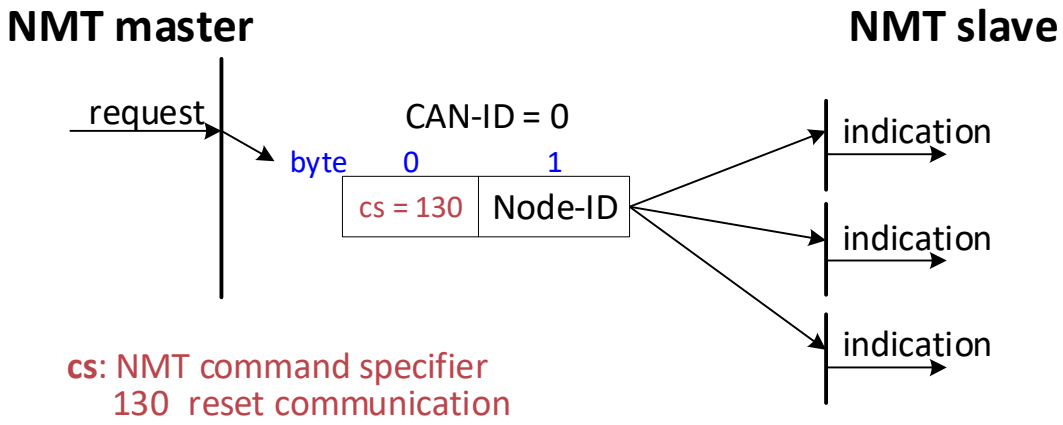
5.2.1.4. Protocol – reset node

The protocol as depicted in the figure below is used to implement the NMT service – reset node.



5.2.1.5. Protocol – reset communication

The protocol as depicted in the figure below is used to implement the NMT service – reset communication.



5.2.2. Error Control Protocol

The cpc TC1/TC1-B drives implement the heartbeat protocol for error control.

5.2.2.1. Protocol – heartbeat

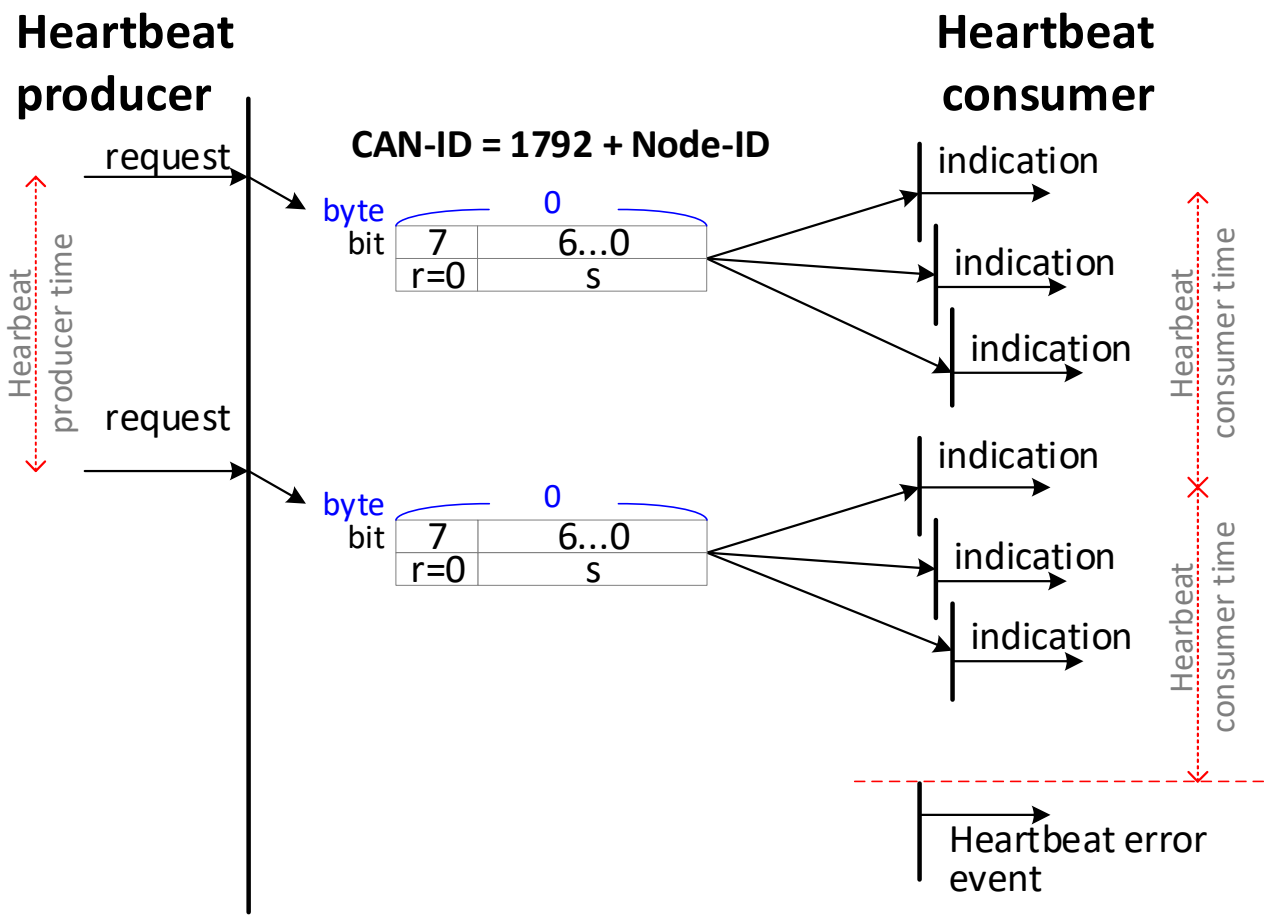
A heartbeat producer transmits a heartbeat message cyclically. One or more heartbeat consumer receives the indication. The relationship between producer and consumer is configurable via the object dictionary.

- Heartbeat consumer time
  The heartbeat consumer guards the reception of the heartbeat within the heartbeat consumer time.
  If the heartbeat is not received within the heartbeat consumer time, a **heartbeat error event** will be generated.

- Heartbeat producer time
  If the heartbeat producer time is configured on a CANopen device the heartbeat protocol begins immediately.
  If a CANopen device starts with a value for the heartbeat producer time unequal to 0, the heartbeat protocol starts on the transition from the NMT state Initialization to the NMT state Preoperational. **In this case the boot-up message is regarded as first heartbeat message.**

See protocol as depicted on next page.

<Heartbeat protocol>



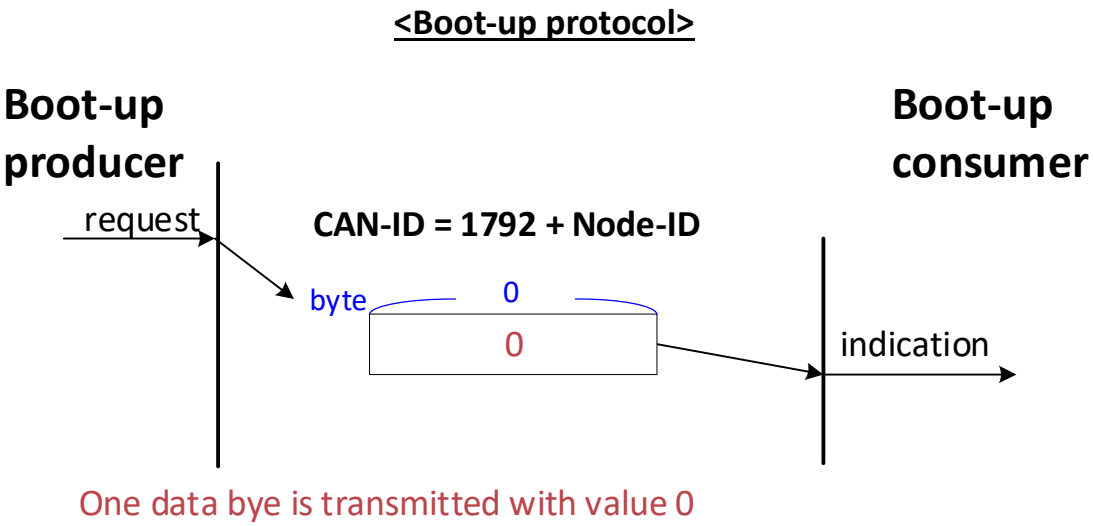
r: reserved (always 0)

S: the state of the heartbeat producer

0:	Boot-up
4:	Stopped
5:	Operational
127:	Pre-operational

5.2.3. Protocol Boot-up

The protocol Boot-up depicted in the figure below is used to signify that **a NMT slave has entered the NMT state Pre-operational** after the NMT state Initialization. This protocol uses the same CAN-ID as the heartbeat protocol.



One data bye is transmitted with value 0

5.3. NMT States and Communication Object Relation

The table below indicates the relation between NMT states and communication objects.

The services of the communication objects can only be executed if the CANopen devices are in the suitable NMT states. See the table below:

<NMT states and executable communication objects>

Object / State	Pre-operational	Operational	Stopped
PDO	-	V	-
SDO	V	V	-
SYNC	V	V	-
TIME	V	V	-
EMCY	V	V	-
Node control and error control	V	V	V

*V: executable

6. Service Data Objects (SDOs)

A SDO provides direct access to object entries of a CANopen device's object dictionary.

As these object entries may contain data of arbitrary size and data type. SDOs can be used to **transfer multiple data sets** (each containing an arbitrary large amount of data) from a client to a server and vice versa.

Basically, **an SDO is transferred as** a sequence of **segments**.
Prior to transferring the segments, there is an **initialization phase** where client and server prepare themselves for transferring the segments. For SDOs, it is also possible to transfer a data set of up to four bytes during the initialization phase. This mechanism is called SDO **expedited transfer**.

- Always the **client initiates an SDO transfer** for any type of transfer.
- The owner of the accessed object dictionary is the server of the SDO.
- Either the client or the server may take the initiative to abort the transfer of an SDO.

6.1. SDO Services and Protocols

The model for the SDO communication is the Client/Server model as described in **chapter 3.5.3**.

Features:

- SDO number	SDO number [1] for every user type on the local device.
- User type	“client” or “server”.
- Mux data type	Multiplexer contains index and sub-index of type STRUCTURE OF UNSIGNED16, UNSIGNED8; The index specifies an object of the CANopen device's object dictionary, the sub-index specifies a component of a CANopen device's object dictionary object.
- Transfer type	According to the length of data to transfer: · Expedited. · Normal (segmented) for up to 4 bytes. · Normal (segmented) for more than 4 data bytes.
- Data type	According to the referenced index and sub-index

The following services can be applied onto an SDO depending on the application requirements:

- SDO download, which is subdivided into
  - SDO download initiate
  - SDO download segment
- SDO upload, which is subdivided into
  - SDO upload initiate
  - SDO upload segment
- SDO abort transfer

SDO expedited transfer is supported.
SDO segmented transfer is supported since objects larger than 4 Bytes are supported.

6.1.1. SDO Download

The client is using the service SDO download for transferring data **from the client to the server** (owner of the object dictionary). The data, the multiplexer (index and sub-index) of the data set, and its size are indicated to the server.

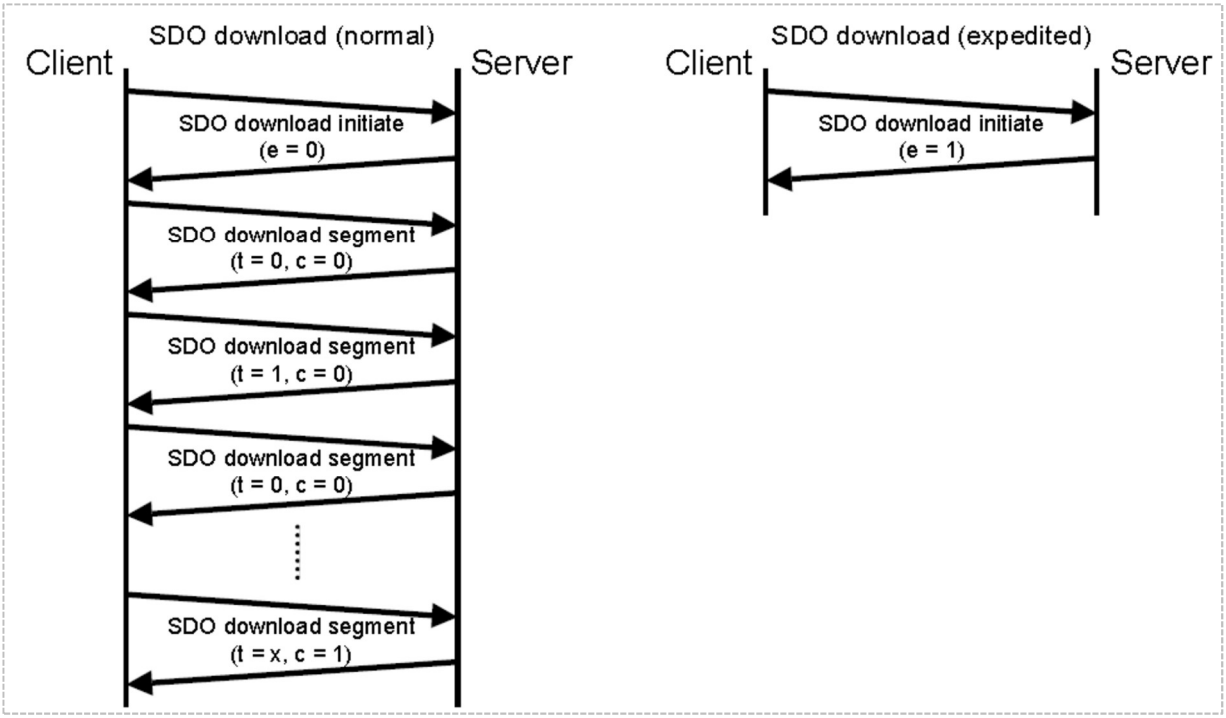
The SDO download contains

- the **SDO download initiate** service, and
- the **SDO download segment** services, including:
  - normal download (data length > 4 bytes)
  - expedited download (data length within 4 byes)

SDOs are downloaded as a sequence of one or several "SDO download segment" services preceded by an "SDO download initiate" service.

The protocol depicted on next page is used to implement the "SDO download" service.

<Protocol - SDO download (normal / expedited)>



e – download type

0: normal

1: expedited

t – toggle bit (only used by SDO normal download)

c – completion (only used by SDO normal download)

0: downloading

1: successful completion of download

The SDO download sequence will be terminated by following occasions:

a. An SDO download initiate request/indication with the **e-bit set to 1** followed by an SDO download initiate response/confirm, indicating the successful completion of an **expedited** download sequence.

b. An SDO download segment response/confirm with the **c-bit set to 1**, indicating the successful completion of a **normal** download sequence.

c. An SDO abort transfer request/indication, indicating the unsuccessful completion of the download sequence.

d. A new SDO download initiate request/indication, indicating the unsuccessful completion of the download sequence and the start of a new SDO download sequence.

Toggle bit failure

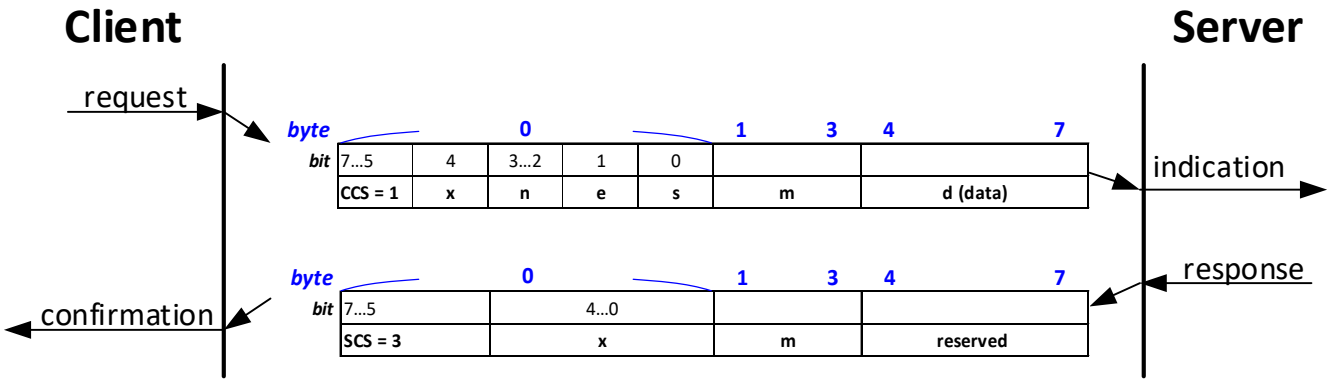
If in the download of two consecutive segments, the toggle bit does not alter, the content of the last segment will be ignored.

If such an error occurs to the application (client or server), the application will decide to abort the download – if it being the case, the user will need to send the message all over (from the beginning of message) again.

6.1.2. SDO Download Initiate

The client requests the server (i.e., owner of the object dictionary) to prepare downloading of data by using the "SDO download initiate" service.
Besides, optionally the number of data bytes is indicated to the server. See the protocol explanations below.

< Protocol - SDO download initiate >



where:

ccs	**Client command specifier** 1: initiate download request.
scs	**Server command specifier** 3: initiate download response
n	Number of bytes in [d] that do not contain data. Bytes [8-n, 7] do not contain data. n = 0 if no segment size is indicated.
e	**Transfer type** 0: normal transfer 1: expedited transfer
s	**Size indicator** 0: Data set size is not indicated 1: Data set size is indicated

m	**Multiplexor**. Represents index/sub-index of the data to be transferred by SDO.
d	**Data.** · **e** = 0, **s** = 0: d is reserved for future use. · **e** = 0, **s** = 1: d contains number of bytes to be downloaded. Byte 4 contains LSB and byte 7 contains MSB. · **e** = 1, **s** = 1: d contains data of length 4-n to be downloaded. The encoding depends on the type of data referenced by index and sub-index. · **e** = 1, **s** = 0: d contains an unspecified number of bytes to be downloaded.
x	Not used; always 0.
reserved	Reserved for future use; always 0.

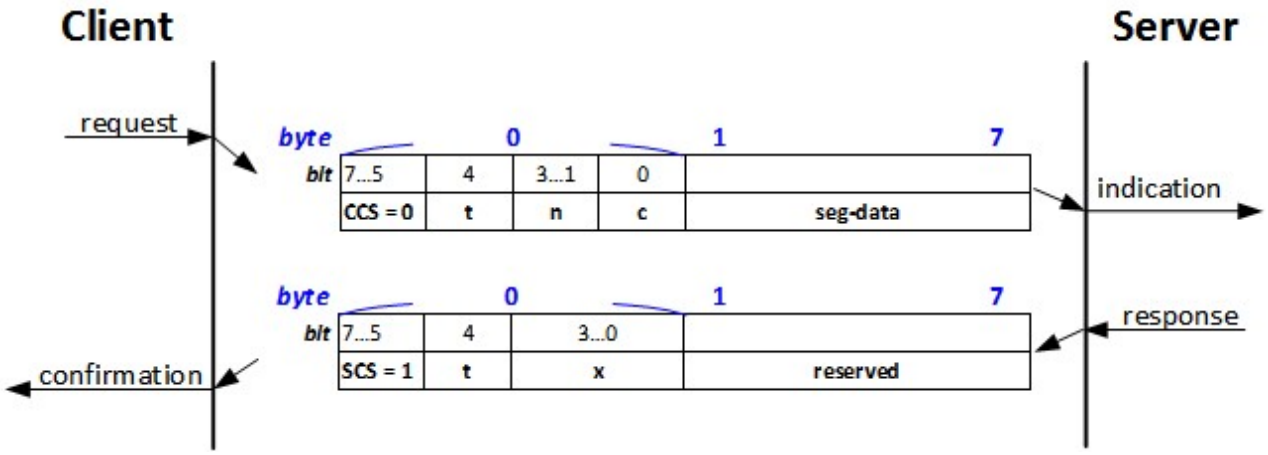
6.1.3. SDO Download Segment

This protocol is used to implement the Download SDO Segment service.

The client transfers the segmented data to the server by using the SDO download service. The segment data and optionally its size are indicated to the server.

The "continue parameter (c)" indicates the server whether there are still more segments to be downloaded or that this is the last segment to be downloaded.

< Protocol - SDO download segment >



Where:

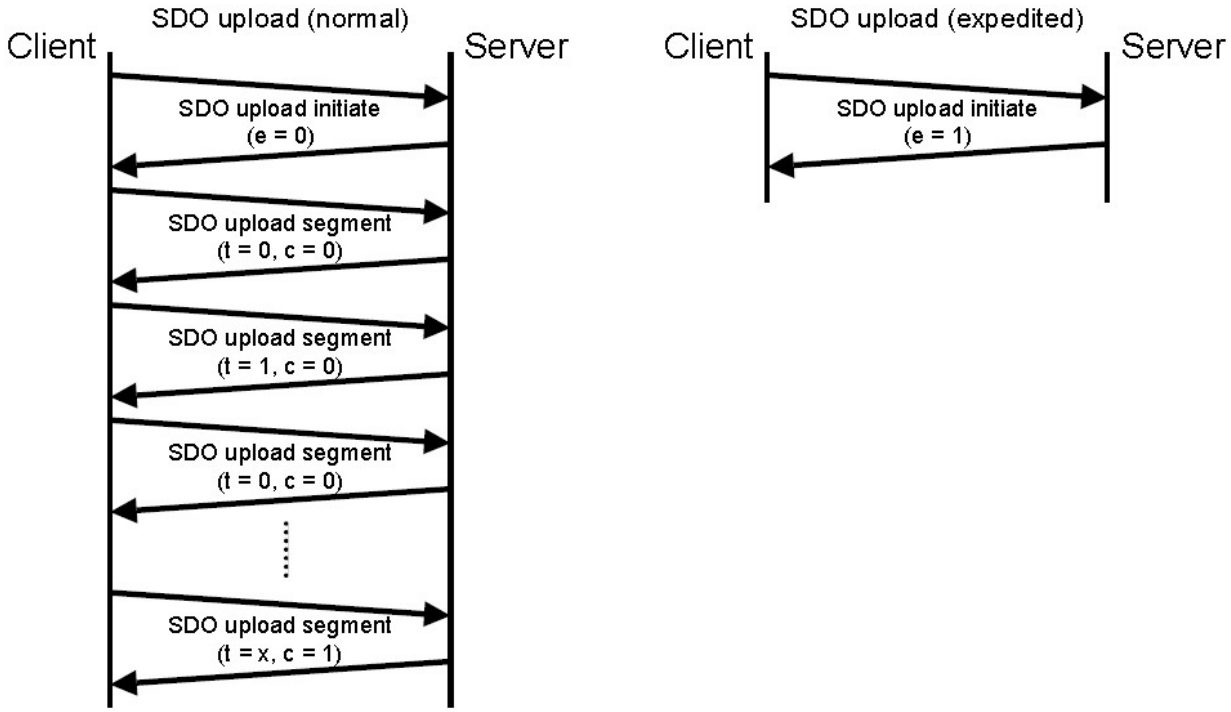
ccs	**Client command specifier** 0: download segment request.
scs	**Server command specifier** 1: download segment response.
seg-data	Maximum 7 bytes of segment data can be downloaded. The encoding depends on the type of the data referenced by index and sub-index.
n	Number of bytes in [d] that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
c	Whether there are more segments to be downloaded: 0: more segments to be downloaded. 1: no more segments to be downloaded.

t	**Toggle bit**, which alternates for each subsequent segment that is downloaded. The first segment has the toggle-bit set to 0. The toggle bit is equal for the request and the response message.
x	Not used; always 0.
reserved	Reserved for future use; always 0.

6.1.4. SDO Upload

The client is using the service SDO upload for transferring the data **from the server** (owner of the object dictionary) **to the client**. The multiplexer (index and sub-index) of the data set is indicated to the server. The SDO upload consists of at least the SDO upload initiate service and optional of SDO upload segment services (data length > 4 bytes).

< Protocol – SDO upload >



e – upload type

0: normal

1: expedited

t – toggle bit (only used by SDO normal upload)

c – completion (only used by SDO normal upload)

0: downloading

1: successful completion of upload

This protocol (specified in the diagram above) is used to implement the **SDO upload service**. **SDOs are uploaded as a sequence of several SDO upload segment services** preceded by an “SDO upload initiate” service.

The SDO upload sequence will be terminated by following occasions:

a. An SDO upload initiate response/confirm with the **e-bit set to 1**, indicating the successful completion of an **expedited** upload sequence.
b. An SDO upload segment response/confirm with the c-bit set to 1, indicating the successful completion of a **normal** upload sequence.
c. An SDO abort transfer request/indication, indicating the unsuccessful completion of the upload sequence.
d. A new SDO upload initiate request/indication, indicating the unsuccessful completion of the upload sequence and the start of a new sequence.

Toggle bit failure:

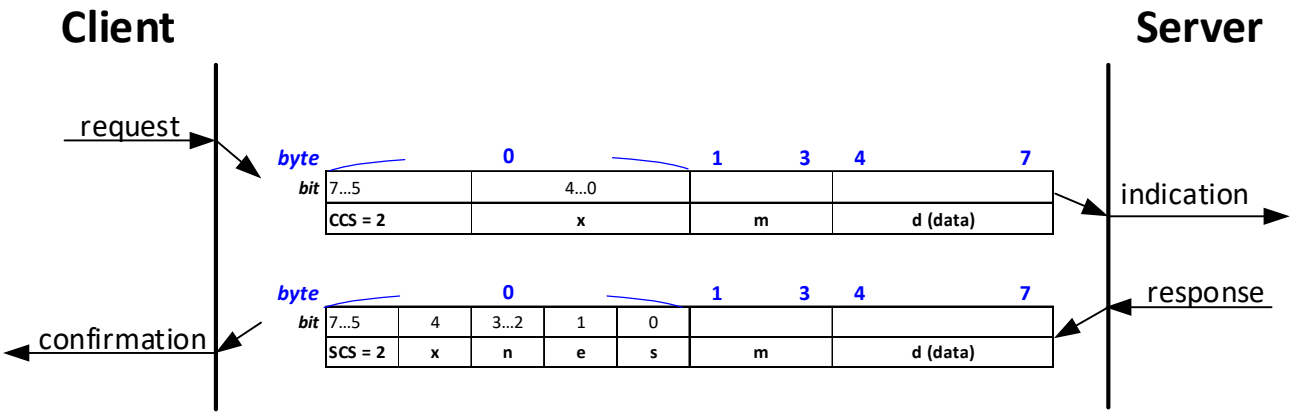
If in the upload of two consecutive segments the toggle bit does not alter, the content of the last segment will be ignored.

If such an error is reported to the application (server to client), the application will decide to abort the upload.

6.1.5. SDO Upload Initiate

The client requests the server to prepare the data for uploading by using the SDO upload initiate service. The multiplexer (index and sub-index) of the data set whose upload is initiated is indicated to the server.

The protocol as depicted in the diagram below is used to implement the "**SDO upload initiate**" service.



Where:

ccs	**Client command specifier** 2: initiate upload request.
scs	**Server command specifier** 2: initiate upload response
n	Number of bytes in [d] that do not contain data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
e	**Transfer type** 0: normal transfer 1: expedited transfer
s	**Size indicator** 0: data set size is not indicated 1: data set size is indicated

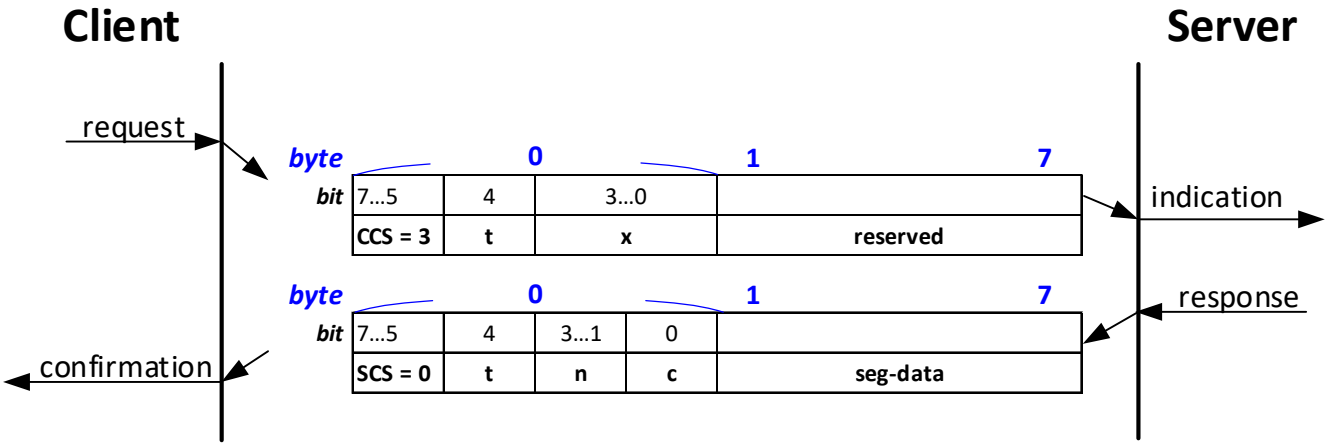
m	**Multiplexer –** representing the index/sub-index of the data to be transferred by the SDO.
d	**Data.** · **e** = 0, **s** = 0: d is reserved for future use. · **e** = 0, **s** = 1: d contains the number of bytes to be uploaded. Byte 4 contains LSB and byte 7 contains MSB. · **e** = 1, **s** = 1: d contains data of length 4-n to be downloaded. The encoding depends on the type of data referenced by index and sub-index. · **e** = 1, **s** = 0: d contains an unspecified number of bytes to be uploaded.
x	Not used, always 0.
Reserved	Reserved for future use, always 0.

6.1.6. SDO Upload Segment

The client requests the server to supply the data of the next segment by using the SDO upload segment service. The continue parameter ([c]) indicates the client whether there are still more segments to be uploaded or that this was the last segment to be uploaded.

Note:
One SDO can only have at most one SDO upload segment service.

The protocol as depicted in the diagram below is used to implement the "**SDO upload segment**" service.



Where:

ccs	**Client command specifier** 3: upload segment request
scs	**Server command specifier** 0: upload segment response
t	**Toggle bit**, which alternates for each subsequent segment that is downloaded. The first segment has the toggle-bit set to 0. The toggle bit is equal for the request and the response message.

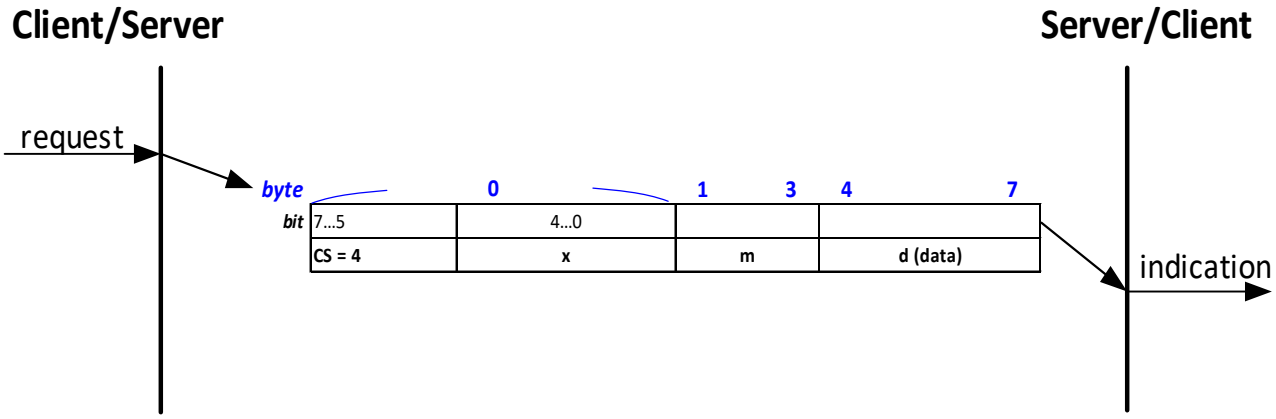
c	**Transfer type** 0: more segments to be uploaded 1: no more segments to be uploaded
seg-data	Maximum seven bytes of segment data uploaded. Encoding depends on type of data referenced by index and sub-index.
n	Number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
x	Not used, always 0.
reserved	Reserved for future use, always 0.

6.1.7. SDO Abort Transfer

The SDO abort transfer service aborts the SDO upload service or SDO download service of an SDO referenced by its number. The reason is indicated (in byte 4 to 7, see Table <SDO Abort Codes> on next page).

The service is unconfirmed. **Both** the client and the server of an SDO may execute the service at any time. If the client of an SDO has a confirmed service outstanding, the indication of the abort is taken to be the confirmation of that service.

The protocol as depicted in the diagram below is used to implement the "**SDO abort transfer**" service.



Where:

cs	**Command specifier** 4: abort transfer request
X	Not used, always 0.
m	**Multiplexer –** representing the index and sub-index of the SDO.
d	Contains a 4-byte abort code about the reason for the abort.

<SDO Abort Codes>

Abort code	Description
0x0503 0000	Toggle bit not alternated
0x0504 0000	SDO protocol timed out
0x0504 0001	Client/server command specifier not valid or unknown
0x0504 0002	Invalid block size (block mode only)
0x0504 0003	Invalid sequence number (block mode only)
0x0504 0005	Out of memory
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write only object
0x0601 0002	Attempt to write a read only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Object cannot be mapped to the PDO
0x0604 0042	The number and length of the objects to be mapped would exceed PDO length
0x0604 0043	General parameter incompatibility reason
0x0604 0047	General internal incompatibility in the device
0x0606 0000	Access failed due to a hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter too high
0x0607 0013	Data type does not match, length of service parameter too low
0x0609 0011	Sub-index does not exist
0x0609 0030	Invalid value for parameter (download only)
0x0609 0031	Value of parameter written too high (download only)
0x0609 0032	Value of parameter written too low (download only)
0x0609 0036	Maximum value is less than minimum value
0x060A 0023	Resource not available: SDO connection
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored to the application
0x0800 0021	Data cannot be transferred or stored to the application because of local control

Abort code	Description
0x0800 0022	Data cannot be transferred or stored to the application because of the present device state
0x0800 0023	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error)
0x0800 0024	No data available

7. Process Data Objects (PDOs)

The real-time data transfer is performed by means of "Process Data Objects (PDO)". The transfer of PDO is performed with no protocol overhead.

The PDO corresponds to objects in the 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 object dictionary.

There are two kinds of use for PDO: data transmission "**Transmit-PDO (TPDO)**" and the data reception "**Receive-PDO (RPDO)**".

- CANopen devices supporting TPDO are called PDO **producer;**
  CANopen devices supporting RPDO are called PDO **consumer**.

- PDO are described by these two parameters:
  - **PDO communication parameter** - describes the communication capabilities of the PDO.
  - **PDO mapping parameter** - contains information about the contents of the PDO.

  The structure of PDO communication as well as mapping parameter are explained in **chapter 12** – Objects 0x1400, 0x1600, 0x1800, and 0x1A00.

- For each PDO the pair of **communication** and **mapping parameter is mandatory.**

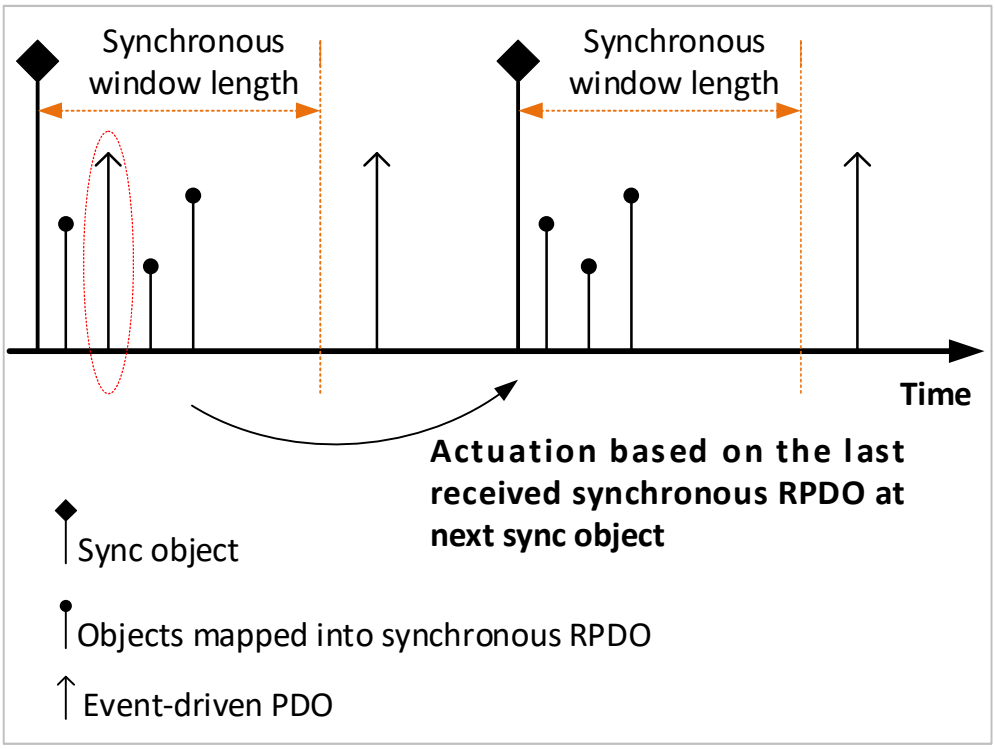
7.1. Transmission Modes

The following transmission modes are distinguished:

- **Synchronous transmission**
- **Event-driven transmission** (also “Value-change” for TPDOs, see **chapter 7.2.2.**)

In order to synchronize CANopen devices, a synchronization object (SYNC) is transmitted periodically by a synchronization application. The SYNC object is represented by a pre-defined communication object (see chapter 9). The diagram below shows the principle of synchronous and event-driven transmission.

<Synchronous and event-driven transmission>



- **Synchronous PDOs** are transmitted **within a pre-defined time-window** immediately after the SYNC object.
- **Event-driven PDOs** are transmitted **at any time**.

The transmission type parameter of a PDO specifies the transmission mode as well as the triggering mode.

- **TPDO:**
  - For synchronous TPDOs the transmission type also specifies the **transmission rate** in form of a factor **based on the basic SYNC object transmission period.**

    The transmission type 1 means that the message is transmitted with every SYNC object. A transmission type of *n* means that the message is transmitted with every *n-th* SYNC object. (See chapter 12 – Object 0x1800, sub-index 0x02).
  - Event-driven TPDOs are transmitted **without any relation to the SYNC object**.
- **RPDO**:

  The data of synchronous RPDOs received after the occurrence of the SYNC object is passed to the application with the occurrence of **the following SYNC**, **independent of the transmission rate specified by the transmission type.**

  The data of event-driven RPDOs is passed directly to the application.

7.2. Triggering Modes

7.2.1. Synchronously Triggered

Message transmission is triggered by the occurrence of the SYNC object.

The trigger condition is **the number of Sync** and optionally **an internal event**.

7.2.2. Event-and timer-driven

Message transmission is either triggered by:

- The occurrence of an application-specific event specified in the device profile, application profile or manufacturer-specific, or
- A specified time (event-time) which has elapsed without occurrence of an event.

8. Emergency (EMCY)

Emergency objects are **triggered by the occurrence of a CANopen device internal error situation** and are transmitted from an emergency producer on the CANopen device. Emergency objects are suitable for interrupt type error alerts.

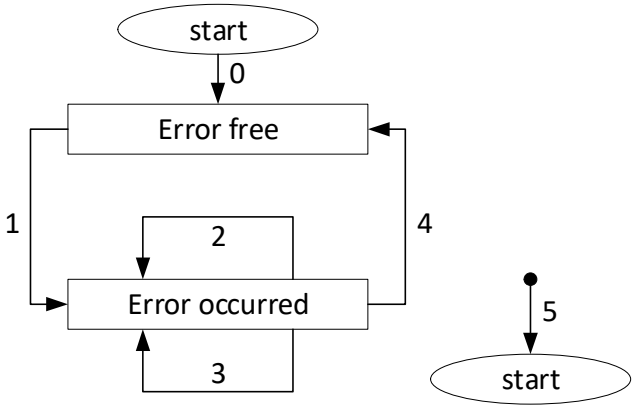
An emergency object is transmitted **only once per 'error event'**. No further emergency object is transmitted as long as no new errors occur on a CANopen device.

Through this specification, error code register is specified as described in chapter 12 – Object 0x1001 (Error register). Meanwhile, cpc emergency error codes are defined in the document: **cpc TC1/TC1-B Series Installation Guide,** see section of Maintenance/Troubleshoot.

8.1. Emergency state transition

A CANopen device is in either “error free” or “error occurred” state. (See the figure below).

<Emergency state transition diagram>

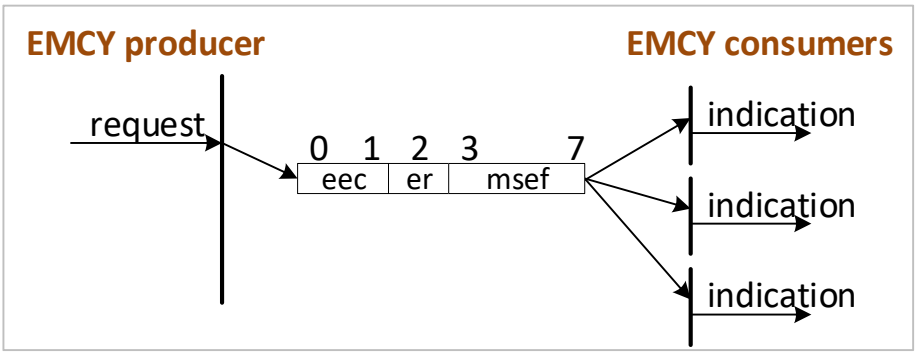


0. After initialization, **if no error is detected** then the CANopen device enters the **error free state**. No error message is sent.
1. The CANopen device **detects an internal error** indicated in the first three bytes of the emergency message (error code and error register) ➔ The CANopen device enters the **error state**.

   An emergency object with the appropriate error code and error register **is transmitted.** The error code is filled in at the location of **object 1003h** (pre-defined error field).
2. **One, but not all, error reasons is gone** ➔ An emergency message containing **error code 0000h (Error reset) is transmitted** (meaning the ‘gone’ error reason is cleared) together with the remaining errors in the **error register** and in the manufacturer-specific error field.
3. **A new error occurs** on the CANopen device ➔ The CANopen 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 (1003h). (oldest error - highest sub-index, see object 1003h).
4. **All errors are repaired** ➔ The CANopen device enters the **error free state** and transmits an **emergency object** with the error code **‘reset error / no error'.**
5. Reset or power-off.

8.2. Emergency Object Service and Protocol

The emergency object follows the producer/consumer as described in chapter 3.5.2.
The service is unconfirmed; there are zero or more consumers for EMCY.
The EMCY object has exactly one producer.



- eec: Emergency error code (see TC1/TC1-B Series Installation Guide)
- er: Error register (see object 0x1001)
- msef: Manufacturer-specific error code (see TC1/TC1-B Series Installation Guide)

9. SYNC

This SYNC provides the basic network synchronization mechanism.
The SYNC producer broadcasts the synchronization object periodically.

- The time period between the SYNCs is specified by the standard parameter "**communication cycle period**" (see **chapter 12 – Object 0x1006**), which may be written by a configuration tool to the CANopen devices during the boot-up process. There may be a time jitter in transmission by the SYNC producer corresponding approximately to the latency due to some other message being transmitted just before the SYNC.
- The optional parameter **counter** is used to define an explicit relationship between the current SYNC cycle and PDO transmission (see PDO communication parameter SYNC start value at **chapter 12 – Object 0x1800, sub-index 0x02**).
- In order to guarantee timely access to the network, the SYNC is given a very high priority CAN-ID (see **chapter 12 – Object 0x1005**: COB-ID SYNC message). CANopen devices that operate synchronously may use the SYNC object to synchronize their own timing with that of the synchronization object producer. The details of this synchronization are application-specific and do not fall within the scope of this specification.

9.1. SYNC Service and Protocol

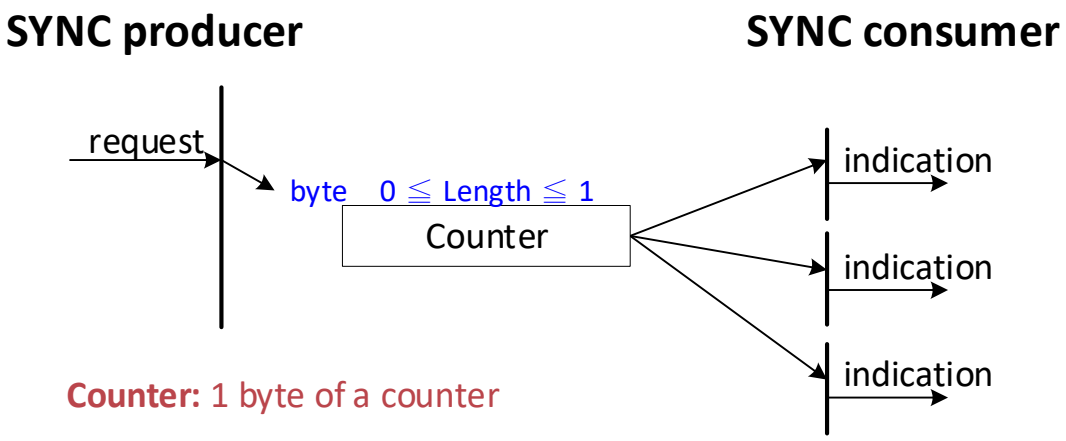
The SYNC transmission follows the producer/consumer push model.
There is exactly one producer of SYNC and zero or more consumers of SYNC.
The service is unconfirmed. Data type is UNSIGNED8.

The optional parameter counter

The optional parameter counter is incremented by 1 with every transmission. The maximum counter value is the current value as defined in the "synchronous counter overflow value" (see **chapte**r 12 – Object 0x1019). Once the maximum value is reached, the counter will be set to 1 with the next transmission.
The initial value of the counter after the NMT service boot-up **is 1**. The value of the counter will be reset to 1 if the CANopen device transits from the NMT state stopped into the NMT state pre-operational or into the NMT state operational.

The protocol depicted as below is used to implement the service of SYNC write.

< Protocol – SYNC write>



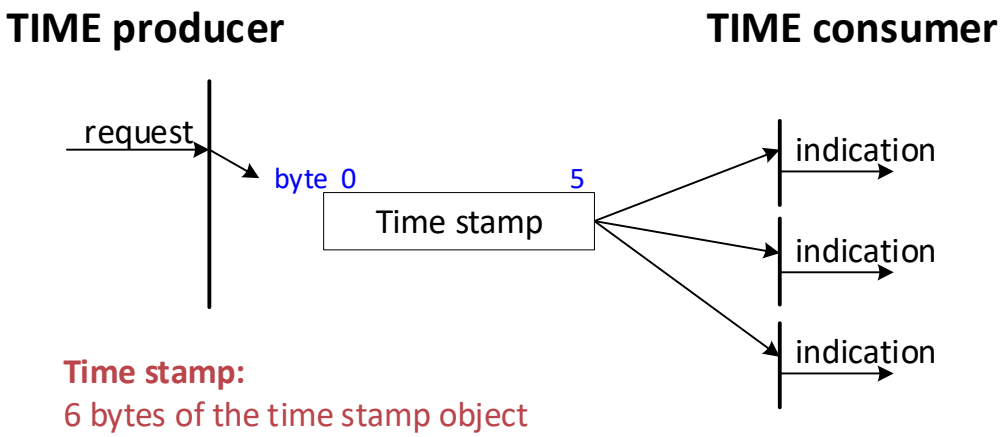
10. Time Stamp (TIME)

The TIME producer broadcasts the time stamp object. This TIME provides the simple network clock. There may be a time jitter in transmission by the TIME producer corresponding approximately to the latency due to some other message being transmitted just before the TIME. In order to guarantee timely access to the network the TIME is given a very high priority CAN-ID. CANopen devices that operate a local clock may use the TIME object to adjust their own time base to that of the time stamp object producer. The details of this mechanism are implementation specific and do not fall within the scope of this specification.

10.1. TIME Service and Protocol

The time stamp object transmission follows the producer/consumer model.
There is exactly one producer of a TIME and zero or more consumers of a TIME.
The service is unconfirmed. Data type is TIME_OF_DAT
The protocol depicted as below is used to implement the service of TIME write.

< Protocol –TIME write>



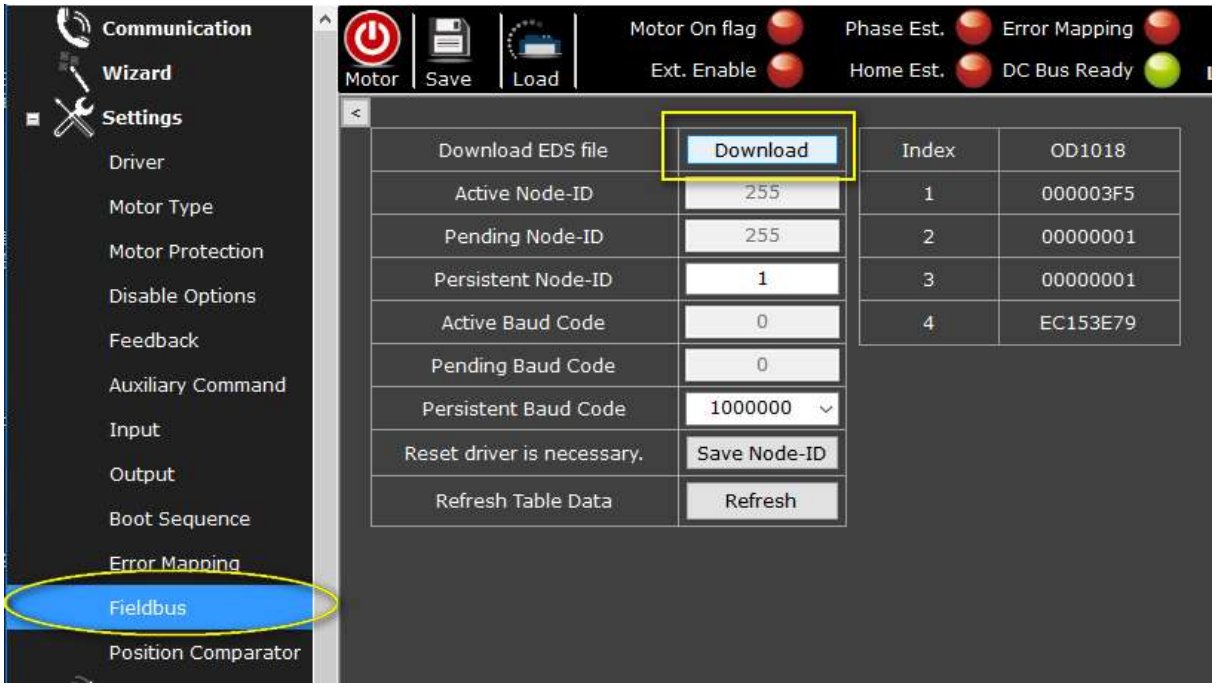
Time stamp:
6 bytes of the time stamp object

11. The EDS

The EDS can be downloaded via SDO segmented transfer. The device which requests the file reads the Object 0x1021 (Store EDS; see chapter 12) to make the drive keep transferring data until the drive responds with a message saying no more data for transfer; in this case the EDS is successfully downloaded to the requesting device and can be saved to a file.

To download EDS via cpc GUI software, please follow the steps below:

1. Go to Settings > Fieldbus section in the GUI.
2. Click “download” button beside the field Download EDS file.

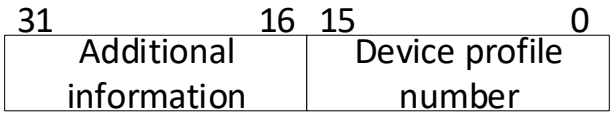


3. A window will then pop out asking users to allocate a place in the computer to save the downloaded EDS.

12. Communication Profile

Object 0x1000: Device type

This object contains information about the device type and functionality.
It is comprised of a 16-bit field that describes the device profile used, and a second 16-bit field that gives additional information about optional functionality of the device.



- Object description

Index	1000
Name	Device type
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	C
PDO mapping	No
Value range	Not applicable. (0x00420192)
Default value	0x00420192
Units	No

Object 0x1001: Error register

This object **provides error information.** The CANopen device maps internal errors into this object. It is a part of an emergency object.

- Bit definition (M for mandatory, O for optional)

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

If a bit is set to one, the specific corresponding error has occurred. The only mandatory bit is bit 0 (generic error) which is signals at any kind of error.

- Object description

Index	1001
Name	Error register
Object code	Variable
Data type	UINT8

- Entry description

Sub-index	0x00
Access	RO
PDO mapping	No
Value range	0 ~ 7
Default value	No
Units	No

Object 0x1002: Manufacturer status register

This object provides a common status register for manufacturer-specific purposes. In this specification, only the size and the location of this object are defined.

- Value definition

Bit#	Meaning
0~15	Same as object 0x6041 of DS402
16	External Enable digital input state
17	Motor On flag state
18	Phase Find operation completed successfully
19	Home position has been established
20	Error Mapping function is active
21	Auxiliary Command is added to internal commands
22~23	Reserved
24	Motor is stationary, functionally equivalent to “Speed is equal to 0” Statusword bit during Profile Velocity mode
25~31	Reserved

- Object description

Index	1002
Name	Manufacturer status register
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	RO
PDO mapping	No
Value range	See value definition
Default value	No
Units	No

Object 0x1003: Pre-defined error field

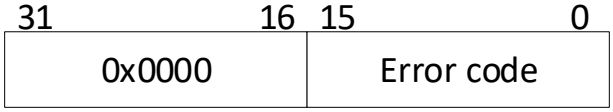
This object shows the errors that have occurred on the CANopen device and have been signaled via the Emergency object. In doing so, it provides an **error history**.

- Value definition
  - The object entry at sub-index 0x00 contains the **actual number** of the errors that are recorded in this array starting at sub-index 0x01.
    Note:
    If no error is present, the value of sub-index 0x00 is 0 and a read access to sub-index 0x01 will be responded with an SDO abort message (abort code: 0800 0024_h).

  - Every NEW error is stored at sub-index 0x01;
    older errors will be moved to the next higher sub-index.

  - Writing value 0 to sub-index 0x00 will delete the entire history (empties this array). In this case, other values than 0 are not allowed and will result in an abort message (error code: 0609 0031_h).

  - The error numbers are of type Usigned32 and are composed of a 16-bit error code which is contained in the lower 2 bytes (LSB).



- Object description

Index	1003
Name	Pre-defined error field
Object code	Array
Data type	UINT32

- Entry description

Sub-index	0x00
Description	Number of error
Access	RW
PDO mapping	No
Value range	UINT8
Default value	No
Units	No

Sub-index	0x01, 0x02, … 0x0F, and 0x10.
Description	Standard error field
Access	RO
PDO mapping	No
Value range	UINT32
Default value	No
Units	No

Object 0x1005: COB-ID SYNC message

This object defined the configured C0B-ID of the SYNC object.
Also, it defines **whether the CANopen device generates the SYNC**.

- Value definition

< Structure of SYNC COB-ID >

<table>
<tr><td>31</td><td>30</td><td>29</td><td colspan="2">28 11 10 0</td></tr>
<tr><td rowspan="2">X</td><td rowspan="2">gen.
(0/1)</td><td rowspan="2">frame
(0)</td><td>0x00000</td><td>11-bit CAN-ID</td></tr>
<tr><td colspan="2">29-bit CAN-ID of the extended frame</td></tr>
</table>

< Description of SYNC COB-ID >

Bit(s)	Value	Description
X	X	Do not care
gen.	0 1	CANopen device does NOT generate SYNC message. CANopen device generates SYNC message.
frame	0* 1	11-bit CAN-ID valid (CAN base frame) ***This bit is always 0.** 29-bit CAN-ID valid (CAN extended frame)
11-bit CAN-ID	X	11-bit CAN-ID of the CAN base frame

- Bit 29 (frame) is not changeable, it's always 0.
- gen.:

  Enable SYNC function.

  If a CANopen device is not able to generate SYNC message, an attempt to set bit 30 (gen) to 1 will be responded with the SDO abort transfer service (abort code: 0x0609 0030).

- The first transmission of SYNC object starts within 1 sync cycle after setting bit 30 to 1.

  By setting bit 30 to 1_b while the synchronous counter overflow value is greater than 0 the first SYNC message starts with the counter reset to 1 (i.e. counting from 1). While the object exists (bit 30 = 1_b) it is not allowed to change the bits 0 to 29.

- Object description

Index	1005
Name	COB-ID SYNC message
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	No
Value range	0 ~ 0x800007FF
Default value	0x80
Units	No

Object 0x1006: Communication cycle period

This object shows the communication cycle period. This object defines the **time length of the SYNC cycle**.

- Value definition

  The value is given in multiple of μ sec.

  If the value is set to 0x0000 0000 the transmission of SYNC messages is disabled.

  By changing the value from 0000 0000h and the synchronous counter overflow value is greater than 0 the first SYNC message starts with the counter value reset to 1.

  The transmission of SYNC messages starts within one communication cycle period (as given by the value) after object 0x1006 is set to the new value.

- Object description

Index	1006
Name	Communication cycle period
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	No
Value range	0 ~ 240
Default value	0x0
Units	μ sec

Object 0x1008: Manufacturer device name

This object shows the device name given by cpc.

- Value definition

  For example, the values may show as TC1-8/230, TC1-B9/230-E, and so on.

- Object description

Index	1008
Name	Manufacturer device name
Object code	Variable
Data type	Visible string

- Entry description

Sub-index	0x00
Access	C
PDO mapping	No
Value range	Visible string
Default value	Same as the name string shown in cpc UI > Setting > Information > Model.
Units	No

Object 0x1009: Manufacturer hardware version

This object provides the description of cpc hardware version, such as “00.100”.

- Object description

Index	1009
Name	Manufacturer hardware version
Object code	Variable
Data type	Visible string

- Entry description

Sub-index	0x00
Access	C
PDO mapping	No
Value range	Visible string
Default value	Same as the name string shown in cpc UI > Setting > Information > Hardware Revision
Units	No

Object 0x100A: Manufacturer software version

This object provides the cpc software version description.

- Object description

Index	100A
Name	Manufacturer software version
Object code	Variable
Data type	Visible string

- Entry description

Sub-index	0x00
Access	C
PDO mapping	No
Value range	Visible string
Default value	Same as the name string shown in cpc UI > Setting > Information > Firmware Revision.
Units	No

Object 0x1010: Store parameters

This object controls the saving function of parameter in non-volatile memory.

- Value definition

  By read access, this object provides information about its saving capacity, using:

  - Sub-index 0x00: The largest supported sub-index.
  - Sub-index 0x01: Save **all** parameters.
  - Sub-index 0x02: Save **Communication** related parameters (index from 0x1000 to 0x1FFF).
  - Sub-index 0x03: Save **Application** related parameters (index from 0x6000 to 0x9FFF).
  - Sub-index 0x04: Save **RS232** parameters.
  - Sub-index 0x05: Save **Virtual machine** program.
  - Sub-index 0x06: Save **Error-mapping** data.

  In order to avoid unexpected storage (of parameters), storage is only executed when a specific signature – save – is written to the appropriate sub-index.

  < Signature of storage write access >

Signature	MSB			LSB
	e	**v**	**a**	**s**
hex	0x65	0x76	0x61	0x73

  - Once receiving the correct signature in the appropriate sub-index:
    The CANopen device **saves the parameter** and **confirms the SDO transmission** (SDO download initiate response).

    For example:

    By writing “save” to sub-Index 0x02, the communication parameters (Object 1000h to 1FFFh) are stored.

If storing the parameter failed, the CANopen device responds with the SDO abort transfer service (abort code: 0x0606 0000).

- Once receiving a wrong signature:
  The CANopen device **refuses to store** and **responds with the SDO abort transfer service** (abort code: 0x0800 0022).

- On **read access** to the appropriate sub-index, the CANopen device provides information about its storage functionality using the following format:

< Structure of storage and read access >

31 … 2	1	0
reserved (0x0000 0000)	**auto** (0)	**cmd** (1)

auto = 0:
cpc CANopen drive does not automatically save parameters.

cmd = 1:
cpc CANopen drive can save parameters on command.

For example:
By reading data of an object entry sub-index 0x02, the cpc driver provides the value “0x0000 0001” as its information of saving functionality

- Object description

Index	1010
Name	Store parameters
Object code	Array
Data type	UINT32

- Entry description (0x00 ~ 0x06)

Sub-index	0x00
Description	Largest supported sub-index
Access	C
PDO mapping	No
Value range	0x01 ~ 0x7F
Default value	6 (as of firmware version 0.7.15)

Sub-index	0x01
Description	Save all parameters
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	1. See *<Structure of storage and read access>*

Sub-index	0x02
Description	Save communication parameters
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	*1. See <Structure of storage and read access>*

Sub-index	0x03
Description	Save application related parameters
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	*1. See <Structure of storage and read access>*

Sub-index	0x04
Description	Save RS232 parameters
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	*1. See <Structure of storage and read access>*

Sub-index	0x05
Description	Save virtual machine program
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	*1. See <Structure of storage and read access>*

Sub-index	0x06
Description	Save error mapping data
Access	RW
PDO mapping	No
Value range	See *<Signature of storage write access>*
Default value	*1. See <Structure of storage and read access>*

Object 0x1011: Restore default parameters

With this object, the default values cab be restored.

- Value definition

  By read access, this object provides information about its capability to restore these values, using:

  - Sub-index 0x00: The largest supported sub-index.
  - Sub-index 0x01: Restore **all** parameters.
  - Sub-index 0x02: Restore **Communication** related parameters (index from 0x1000 to 0x1FFF).
  - Sub-index 0x03: Restore **Application** related parameters (index from 0x6000 to 0x9FFF).
  - Sub-index 0x04: Restore **RS232** parameters.
  - Sub-index 0x05: Restore **Virtual machine** program.
  - Sub-index 0x06: Restore **Error-mapping** data.

  In order to avoid unexpected restoring (of parameter default values), restoring is only executed when a specific signature – **load** – is written to the appropriate sub-index.

  < Signature of restore default write access >

Signature	MSB			LSB
	s	**a**	**o**	**L**
hex	0x64	0x61	0x6F	0x6C

  - Once receiving the correct signature in the appropriate sub-index:
    The CANopen device **restores the parameter** and **confirms the SDO transmission** (SDO download initiate response).

    For example:
    By writing “load” to sub-Index 0x02, the default values of communication parameters (Object 1000h to 1FFFh) are restored.

If restoring the default values failed, the CANopen device responds with the SDO abort transfer service (abort code: 0x0606 0000).

- Once receiving a wrong signature:
  The CANopen device **refuses to restore** and **responds with the SDO abort transfer service** (abort code: 0x0800 0022).

- The default values are set as valid after the CANopen device is reset or power cycled.

- On **read access** to the appropriate sub-index, the CANopen device provides information about its default value restoring capability using the following format:

< Structure of restore default read access >

31 … 1	0
reserved (0x0000 0000)	**cmd** (1)

cmd = 1:

cpc CANopen drive restores parameters.

For example:

By reading data of an object entry sub-index 0x02, the cpc driver provides the value “0x0000 0001” as its information of restoring capability.

- Object description

Index	1011
Name	Restore default parameters
Object code	Array
Data type	UINT32

- Entry description (0x00 ~ 0x06)

Sub-index	0x00
Description	Largest supported sub-index
Access	C
PDO mapping	No
Value range	0x01 ~ 0x7F
Default value	6 (as of firmware version 0.7.15)

Sub-index	0x01
Description	Restore all parameters' default values
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Sub-index	0x02
Description	Restore communication parameters' default values
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Sub-index	0x03
Description	Restore application parameters' default values
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Sub-index	0x04
Description	Restore RS232 parameters' default values
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Sub-index	0x05
Description	Restore virtual machine program
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Sub-index	0x06
Description	Restore error mapping data
Access	RW
PDO mapping	No
Value range	See *<Signature of restore default write access>*
Default value	1. See *<Structure of restore default read access>*

Object 0x1013: High resolution time stamp

This object contains a time stamp with a 1-microsecond (1 μ sec) resolution.

- Object description

Index	1013
Name	High resolution time stamp
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	Yes
Value range	UINT32
Default value	0
Units	1 μ sec

Object 0x1014: COB-ID EMCY

This object shows the COB-ID for the EMCY write service.

- Value definition

< Structure of the EMCY COB-ID >

31	30	29	28 ~ 11	10 ~ 0
valid (0/1)	0	**frame** (0)	0x0 0000	**11-bit CAN-ID**

< Description of the EMCY COB-ID >

Bit(s)	Value	Description
valid	0	EMCY exists / is valid
	1	EMCY does not exist / is not valid
30	0	Reserved (always 0)
frame	0*	11-bit CAN-ID valid (CAN base frame) ***This bit is always 0.**
	1	29-bit CAN-ID valid (CAN extended frame)
11-bit CAN-ID	X	11-bit CAN-ID of the CAN base frame

- Bits 29 and 30 are not changeable, they are always 0.
- While the object exists (bit 31 = 1_b), it is not allowed to change the bits 0~29.

- Object description

Index	1014
Name	COB-ID EMCY
Object code	Variable
Data type	UINT32

- Entry description

Sub-index	0x00
Access	RO
PDO mapping	No
Value range	NODE-ID+0x80 ~ NODE-ID+0x80000080
Default value	NODE-ID+0x80

Object 0x1015: Inhibit time EMCY

This object indicates the configured inhibit time for the EMCY message.
An error will result in an EMCY message; if several errors occur within a short period of time this object is used to be the interval time between each EMCY message transmission.

- Value definition

  This object is given in multiples of 100 um; value 0 will disable the inhibit time.

- Object description

Index	1015
Name	Inhibit time EMCY
Object code	Variable
Data type	UINT16

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	No
Value range	UINT16
Default value	0
Units	100 μ sec

Object 0x1016: Consumer heartbeat time

The **consumer** heartbeat time defines the expected heartbeat cycle time therefore **has to be greater than** the corresponding **producer** heartbeat cycle time (Object 0x1017) configured on the device producing this heartbeat.

Note:

- Monitoring of the producer heartbeat will start after the first heartbeat is received.
- Before receipt of the first heartbeat, the status of the heartbeat producer is unknown.

- Value definition

31 24	23 16	15 0
Reserved (0)	Node-ID	Heartbeat time

  - If the heartbeat time is 0 or the Node-ID is 0 or greater than 127 the corresponding object entry is not used.
  - The heartbeat time duration is given in **multiples of 1ms.**
  - Trying to configure several heartbeat times unequal 0 for the same Node-ID the CANopen device will respond with the SDO abort transfer service (abort code: 0604 0043h).

- Object description

Index	1016
Name	Consumer heartbeat time
Object code	Array
Data type	UINT32

- Entry description

Sub-index	0x00
Description	Largest supported sub-index
Access	C
PDO mapping	No
Value range	0x01 ~ 0x7F
Default value	4 (as of firmware version 0.7.15)

Sub-index	0x01 ~ 0x04
Description	Consumer heartbeat time 1, 2,3 and 4
Access	RW
PDO mapping	No
Value range	UINT32
Default value	0

Object 0x1017: Producer heartbeat time

The producer heartbeat time defines the configured cycle time (duration) of the heartbeat.

- Value definition

  This object is given in multiples of 1 ms.

  Value 0 will disable the producer heartbeat.

- Object description

Index	1017
Name	Producer heartbeat time
Object code	Variable
Data type	UINT16

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	No
Value range	UINT16
Default value	0
Units	1 ms

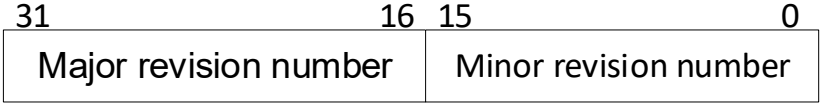
Object 0x1018: Identity object

This object indicates the general identification information of CANopen device.

The master device needs this object to differentiate CANopen slave devices so as to later on assign a NODE-ID for each slave.

- Value definition
  - Sub-index 0x01 contains the unique value that is assigned by CiA to each manufacturer of a CANopen device. cpc's vendor ID value is 0x000003F5.
  - Sub-index 0x02 contains the unique value that identifies a specific type of CANopen devices.
  - Sub-index 03h contains the major revision number and the minor revision number of the revision of the CANopen device (see <Structure of revision number>).

<Structure of revision number>



- The major revision number identifies a specific CANopen behavior. That means if the CANopen functionality is different, the major revision number will be incremented.
- The minor revision number identifies different versions of CANopen device with the same CANopen behavior.

- Object description

Index	1018
Name	Identity object
Object code	Record
Data type	Identity

- Entry description

Sub-index	0x00
Description	Number of entries
Access	RO
PDO mapping	No
Value range	4
Default value	4

Sub-index	0x01
Description	Vendor-ID
Access	RO
PDO mapping	No
Value range	UINT32
Default value	0x000003F5

Sub-index	0x02
Description	Product code
Access	RO
PDO mapping	No
Value range	UINT32
Default value	1

Sub-index	0x03
Description	Revision number
Access	RO
PDO mapping	No
Value range	UINT32
Default value	1

Sub-index	0x04
Description	Serial number
Access	RO
PDO mapping	No
Value range	UINT32
Default value	Unique to each drive

Object 0x1019: Synchronous counter overflow value

This object indicates the configured highest value the synchronous counter supports.

- This object needs to be implemented by the producer and the consumer.
- If the value is greater than 1, the SYNC message data length is **1 byte**. (See *value definition chart*)
- The SYNC consumer ignores the value of SYNC message.
- The SYNC consumer cares only about:
  (1) **Receipt** of SYNC message, and
  (2) **Data length** of a received SYNC message
  If isn't 1 byte, an EMCY message (error code: 0x8240 - unexpected SYNC data length) will be transmitted by a SYNC consumer.

- Value definition

Value	Description
0	The SYNC message is transmitted as a CAN message which carries no data (**data length = 0**).
1	Reserved (it's meaningless to set value to 1)
2 ~ 240	The SYNC message is transmitted as a CAN message of **data length 1.**
241 ~ 255	Reserved

The counter value keeps incremented up to the overflow value and then wraps to value 1.
The overflow value used has to be the Least Common Multiple of all the TPDO transmission types (1 < n < 240) used.

- Object description

Index	1019
Name	Synchronous counter overflow value
Object code	VAR
Data type	UINT8

- Entry description

Sub-index	0x00
Access	RW
PDO mapping	No
Value range	UINT8
Default value	0

Object 0x1021: Store EDS

This object indicates the downloaded EDS file itself.
When this object is read, the drive's firmware uploads the EDS file via network to the device requesting for EDS file.

The Electronic Data Sheet (EDS) is an **object directory** that contains numerous objects describing the functions and properties of a CANopen device. It assists CANopen configuration personnel in determining which objects a CAN slave supports. Also, it is used by network configuration tools to help you identify devices and easily command them on a network.
The EDS file format is standardized and specified by CiA, enabling all participants of CANopen systems to read and process the complete and correct device descriptions.

- Object description

Index	1021
Name	Store EDS
Object code	VAR
Data type	Domain

- Entry description

Sub-index	0x00
Access	RO
PDO mapping	No
Value range	A text file
Default value	No

Object 0x1022: Store format

Object 0x1022 defines whether the EDS can be compressed or not, which must be 0, for No compression.

- Value definition

  This object is given in multiples of 100 um; value 0 will disable the inhibit time.

<Values for EDS store format>

Value	Description
0x00	/ISO10646/, not compressed

- Object description

Index	1022
Name	Store format
Object code	Variable
Data type	UINT8

- Entry description

Sub-index	0x00
Access	RO
PDO mapping	No
Value range	UINT8
Default value	0

Object 0x1200: SDO server parameter

This object describes the properties of **SDO server**.
The cpc driver supports **one** server only.

- Value definition

  The **number** of supported sub-index entries in the SDO object record is specified by sub-index 0x00.
  The values of sub-index 0x01 and 0x02 specify the COB-ID for this SDO.

< Structure of SDO Server COB-ID >

31	30	29	28 ... 11	10 ... 0
valid (0)	dyn (0)	frame (0)	0x0 0000	**11-bit CAN-ID**
			29-bit CAN-ID of the extended frame	

< Description of SDO Server COB-ID >

Bit(s)	Value	Description
valid	0	SDO exists / is valid / is enabled. ***This bit is always 0.**
dyn	0	Value is assigned statically. ***This bit is always 0.**
	1	Value is assigned dynamically.
frame	0*	11-bit CAN-ID valid (CAN base frame) ***This bit is always 0.**
	1	29-bit CAN-ID valid (CAN extended frame)
29-bit CAN-ID	X	29-bit CAN-ID of the CAN extended frame
11-bit CAN-ID	X	11-bit CAN-ID of the CAN base frame

An SDO exists only if the bit *valid* (bit 31) of BOTH sub-index 0x01 and 0x02 is set to 0.

The bit *dyn* (bit 30) of sub-index 0x01 and 0x02 in cpc driver is always 0, meaning that the values of all sub-indexes of this object are stored in non-volatile memory.

- Object description

Index	1200
Name	SDO server parameter
Object code	Variable
Data type	SDO parameter record

- Entry description

Sub-index	0x00
Description	Largest supported sub-index
Access	C
PDO mapping	No
Value range	2
Default value	2

Sub-index	0x01
Description	COB-ID client to server (rx)
Access	C
PDO mapping	No
Value range	See value definition
Default value	CAN-ID: 0x0600 + Node-ID frame: 0 dyn: 0 valid: 0

Sub-index	0x02
Description	COB-ID client to server (tx)
Access	RO
PDO mapping	No
Value range	See value definition
Default value	CAN-ID: 0x0580 + Node-ID frame: 0 dyn: 0 valid: 0

Object 0x1400~0x1403: RPDO communication parameter

This object describes the communication parameters (i.e., to receive or not) of the PDOs the CANopen device is able to receive.

- Value definition
  - **Sub-index 0x00** indicates the number of valid object entries within the record; the minimum value is 2.
  - **Sub-index 0x01** indicated the COB-ID of the RPDO.

<Structure of RPDO COB-ID>

31	30	29	28 ~ 11	10 ~ 0
Valid (0/1)	reserved (x)	frame (0)	0x0 0000	11-bit CAN-ID
			29-bit CAN-ID of the extended frame	

<Description of RPDO COB-ID>

Bit(s)	Value	Description
valid	0	PDO exists / is valid / is enabled.
	1	PDO doesn't exist / is not valid.
reserved	X	Do not care
frame	0*	11-bit CAN-ID valid (CAN base frame) ***This bit is always 0.**
	1	29-bit CAN-ID valid (CAN extended frame)
29-bit CAN-ID	X	29-bit CAN-ID of the CAN extended frame
11-bit CAN-ID	X	11-bit CAN-ID of the CAN base frame

➢ **Sub-index 0x02** defines the reception character of the RPDO (see <Description of RPDO transmission type>). An SDO abort transfer service (abort code: 0x0609 0030) will be issued if trying to change the value of the transmission type to a not supported value.

<Description of RPDO transmission type>

Value	Description
0x00 ~ 0xF0	Synchronous
0xF1 ~ 0xFD	Reserved
0xFE ~ 0xFF	Event-driven

- *Synchronous*:

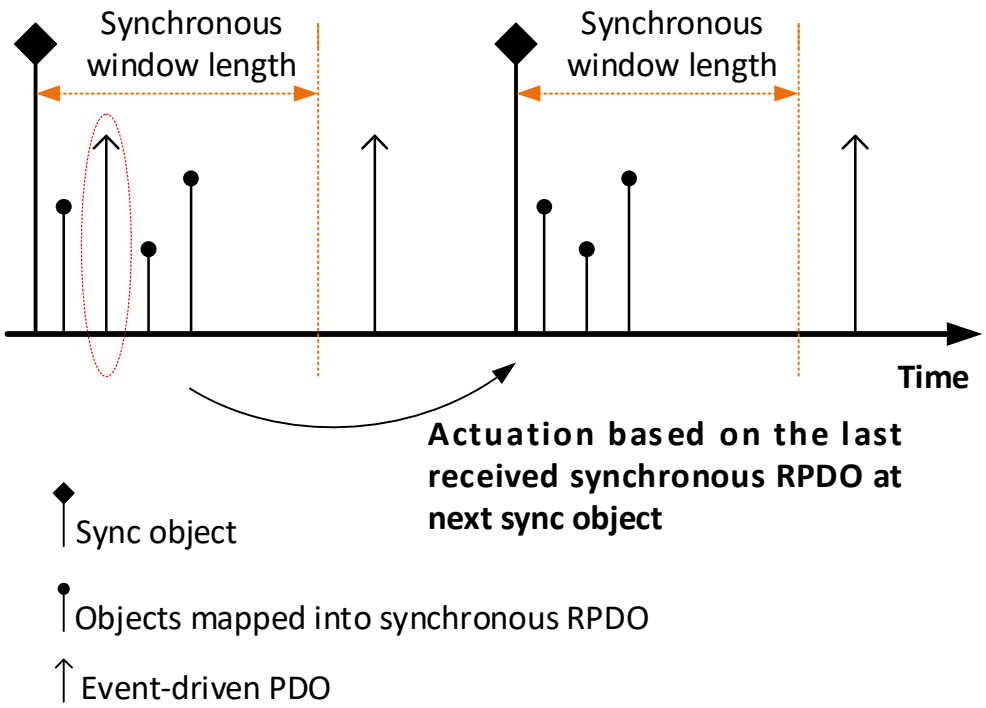
  The CANopen device activates the received data at the receipt of next SYNC (see figure below)

- *Event-driven*:

  The PDO may be received at any time.

  The CANopen device activates the received data immediately.

<Bus synchronization and actuation>



- Object description

Index	1400 ~ 1403
Name	RPDO communication parameter
Object code	Record
Data type	PDO communication parameter record

- Entry description

	0x1400	**0x1401**	**0x1402**	**0x1403**
Sub-index	**0x00**			
Description	Largest supported sub-index			
Access	C			
PDO mapping	No			
Value range	2			
Default value	2			
Sub-index	**0x01**			
Description	COB-ID used by RPDO			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	Node-ID + 0x00000200	Node-ID + 0x80000300	Node-ID + 0x80000400	Node-ID + 0x80000500
Sub-index	**0x02**			
Description	Transmission type			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	FF			

Object 0x1600~0x1603: RPDO mapping parameter

This object defines the contents (i.e., what to receive) of the PDOs that the CANopen device is able to receive.

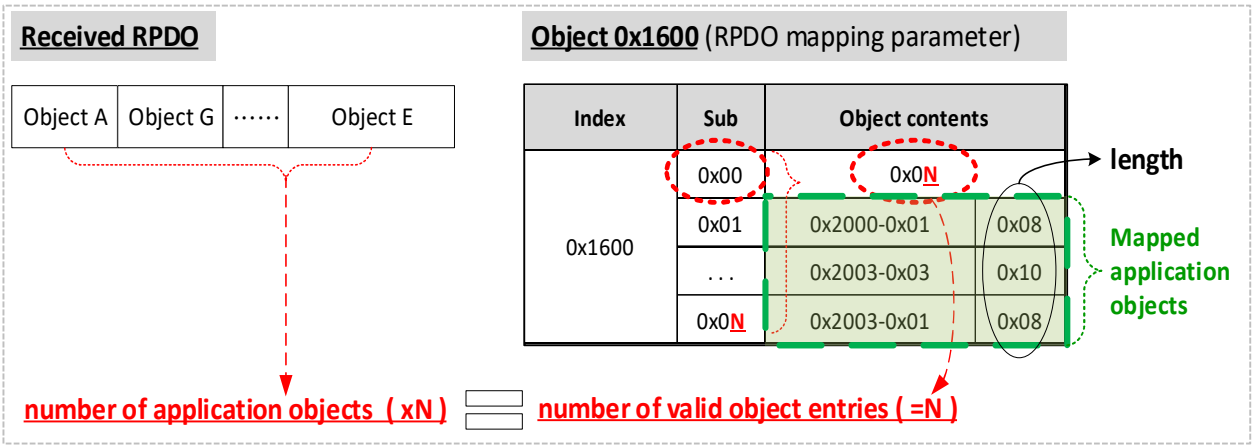
- Value definition

  Sub-index 0x00 contains the number of valid object entries within the mapping record.

<Value of RPDO mapping sub-index 0x00>

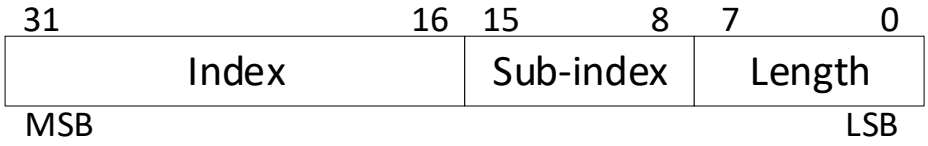
Value	Meaning
0x00	Mapping disabled
0x01	Sub-index 0x01 valid
0x02	Sub-index 0x01 and 0x02 valid
0x03	Sub-index 0x01 to 0x03 valid
0x04	Sub-index 0x01 to 0x04 valid
0x05~ 0xFF	Reserved

The number of valid object entries is the number of the application object that is received with the corresponding RPDO.



Sub-indexes 0x01 to 0x04 contain the information of the mapped application objects. **The mapped application objects** describe the content of the PDO by their index, sub-index, and length (See <The structure of RPDO mapping> and <The Principle of RPDO mapping>).
The length indicates the length of the application object in bits.

<The structure of RPDO mapping>



Trying to **change the value of an object entry** to a not-supported value will be responded with the SDO abort transfer service. Possible causes for a not-supported value would be:

- Mapping (index and sub-index) of a non-existing application object.
- Wrong length of the mapped application object.
- Wrong length of the PDO.

Dummy mapping:
When there is a need to fill up the length of a RPDO (so as to fit the according TPDO) or there is no proper application object, the index and sub-index will refer to a simple data type as shown in the chart below.

<Data type for dummy mapping>

Index	Object	Name
0x0002	DEFTYPE	Integer8
0x0003	DEFTYPE	Integer16
0x0004	DEFTYPE	Integer32
0x0005	DEFTYPE	Unsigned8
0x0006	DEFTYPE	Unsigned16
0x0007	DEFTYPE	Unsigned32
0x0008	DEFTYPE	Real32

Remapping procedure:

Used when users need to configure a mapping different from the default This procedure can take place during the NMT state Pre-operational and during the NMT state Operational:

1. Destroy RPDO by setting bit valid to 1_b of sub-index 0x01 of the according RPDO communication parameter (i.e., Object 0x1400~0x1403).
2. Disable mapping by setting sub-index 0x00 of Object 0x1600 to 0.
3. Modify mapping by changing the values of the corresponding sub-indices.
4. Enable mapping by setting sub-index 0x00 of Object 0x1600 to the number of mapped objects.
5. Create RPDO by setting bit valid to 0_b of sub-index 0x01 of the according RPDO communication parameter.

If during step 3 the CANopen device detects that index and sub-index of the mapped object does not exist or the object cannot be mapped, the CANopen device will respond with the SDO abort transfer service (abort code: 0604 0041h).

If during step 4 the CANopen device detects that the RPDO mapping is not valid or not possible, the CANopen device will respond with the SDO abort transfer service (abort code: 0604 0041h).

If the CANopen device receives a PDO which is with more data bytes than the mapped data is (length), then the CANopen device will use the first data bytes up to the length.

<The Principle of RPDO mapping>

Received RPDO

Object A	Object G	Object E	Object D

Object Dictionary

Index	Sub	Object contents	
0x1600	0x00	0x04	
0x1600	0x01	0x2006-0x01	0x08
0x1600	0x02	0x2011-0x03	0x10
0x1600	0x03	0x2011-0x01	0x04
0x1600	0x04	0x2010-0x00	0x04
0x2006	0x00	0x02	
0x2006	0x01	Object A	
0x2006	0x02	Object B	
0x2007	0x00	Object C	
0x2010	0x00	Object D	
0x2011	0x00	0x03	
0x2011	0x01	Object E	
0x2011	0x02	Object F	
0x2011	0x03	Object G	

(See Object description and Entry description on subsequent pages)

- Object description

Index	1600 ~ 1603
Name	RPDO mapping parameter
Object code	Record
Data type	RPDO mapping parameter record

- Entry description

	0x1600	**0x1601**	**0x1602**	**0x1603**
Sub-index	**0x00**			
Description	Number of mapped application objects in PDO			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	1	2	2	0
Sub-index	**0x01**			
Description	1st application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0x60400010	0x60400010	0x60400010	0
Sub-index	**0x02**			
Description	2nd application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0x607A0020	0x60FF0020	0
Sub-index	**0x03**			
Description	3rd application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0	0	0
Sub-index	**0x04**			
Description	4th application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0	0	0

Object 0x1800~0x1803: TPDO communication parameter

This object describes the communication parameters of the PDOs the CANopen device is able to transmit.

- Value definition

  Sub-index 0x00 indicates the number of valid object entries within the record. Its value is **5** since inhibit time and event timer are supported.

<Structure of TPDO COB-ID>

31	30	29	28 ~ 11	10 ~ 0
valid (0/1)	RTR (1)	frame (0)	0x0 0000	11-bit CAN-ID
			29-bit CAN-ID	

<Description of TPDO COB-ID>

Bit(s)	Value	Description
valid	0	PDO exists / is valid / is enabled.
	1	PDO doesn't exist / is not valid.
RTR	1	No RTR allowed on this PDO. ***This bit is always 1.**
frame	0*	11-bit CAN-ID valid (CAN base frame) ***This bit is always 0.**
	1	29-bit CAN-ID valid (CAN extended frame)
29-bit CAN-ID	X	29-bit CAN-ID of the CAN extended frame
11-bit CAN-ID	X	11-bit CAN-ID of the CAN base frame

The bit valid (bit 31) is to enable/disable a TPDO in the NMT state Operational.

The cpc driver supports the CAN base frame type only and do not support RTRs; an attempt to set bit 29 (frame) to 1_b or bit 30 (RTR) to 0_b is responded with the SDO abort transfer service (abort code: 0609 0030h). It is not allowed to change bit from 0 to 29 (frame) while the PDO exists and is valid (bit 31 = 0_b).

CANopen devices supporting the enabling (bit 31 = 0_b) and disabling (bit 31 = 1b) of a TPDO only shall respond with the SDO abort transfer service (abort code: 0x0601 0000) on an attempt to change the values from bit 0 to bit 30.

Sub-index 0x02 defines the transmission character of the TPDO (see Table <Description of TPDO transmission type>). An SDO abort transfer service (abort code: 0x0609 0030) will be issued if trying to change the value of the transmission type to a not supported value.

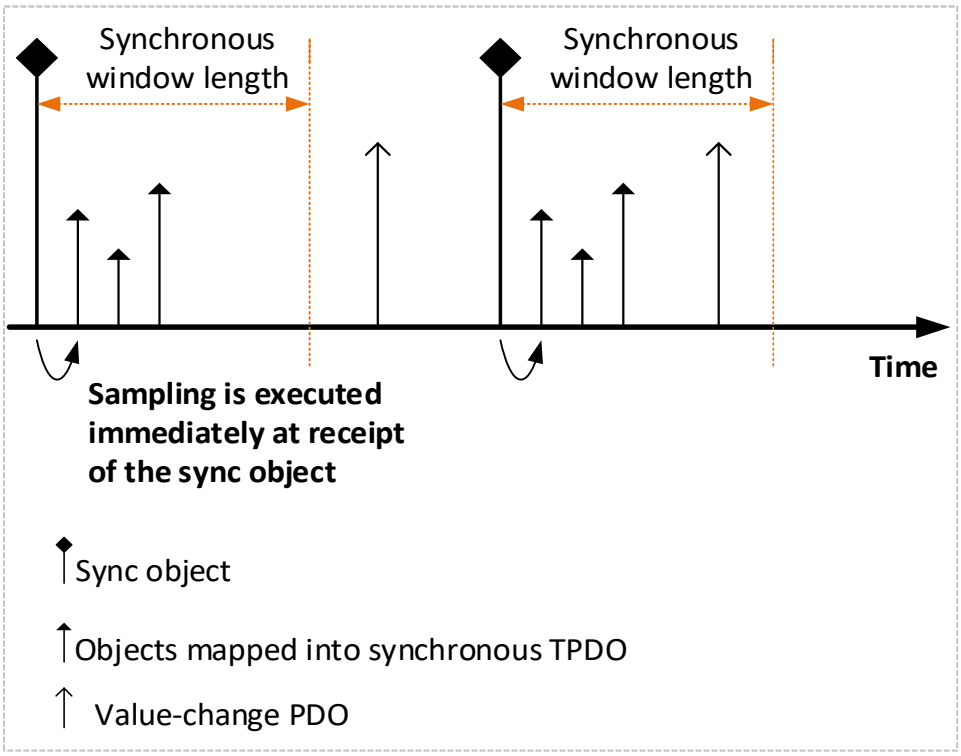
<Description of TPDO transmission type>

Value	Description
0x00	Synchronous
0x01	Synchronous (cyclic every sync)
0x02	Synchronous (cyclic every 2nd sync)
0x03	Synchronous (cyclic every 3rd sync)
……	……
0xF0	Synchronous (cyclic every 240th sync)
0x00F1 ~ 0x00FD	Reserved
0x00FE ~ 0x00FF	Value-change

- *Synchronous*:
  - The PDO is transmitted after the SYNC.
  - The CANopen device **starts sampling data at receipt of SYNC**.
  - Value 0x00 and 0x01: transmit PDO after receipt of SYNC.

- *Value-change*:

  The PDO is transmitted at any time when there is change of value in CANopen device.



Sub-index 0x03 defines the Inhibit Time.

The time is the minimum interval time for PDO transmission if the transmission type is set to 0xFE and 0xFF.

The value is given in multiple of 100 us. Value 0 will disable inhibit time.

When PDO exists (bit 31 of sub-index 0x01 is set to 0), the value of inhibit time cannot be changed.

Sub-index 0x04 is reserved. It should not be used.

Any read or write access to this entry will lead to SDO abort transfer service (abort code: 0x0609 0011).

Sub-index 0x05 contains the event-timer.

The time is the maximum interval time for PDO transmission if the transmission type is set to 0xFE and 0xFF. The value is given in multiple of 1 ms. Value 0 will disable the event-timer.

- Object description

Index	1800 ~ 1803
Name	TPDO communication parameter
Object code	Record
Data type	PDO communication parameter record

- Entry description

	0x1800	0x1801	0x1802	0x1803
Sub-index	**0x00**			
Description	Largest supported sub-index			
Access	C			
PDO mapping	No			
Value range	2 ~ 5			
Default value	5			
Sub-index	**0x01**			
Description	COB-ID used by TPDO			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	Node-ID +	Node-ID +	Node-ID +	Node-ID +
	0x40000180	0x40000280	0x40000380	0x40000480
Sub-index	**0x02**			
Description	Transmission type			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0x00FF			

	0x1800	0x1801	0x1802	0x1803
Sub-index	**0x03**			
Description	Inhibit time			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0			
Sub-index	**0x04**			
Description	Compatibility entry			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0			
Sub-index	**0x05**			
Description	Event timer			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0x0064			

Object 0x1A00: TPDO mapping parameter

This object contains the contents (i.e., what to transmit) of the PDOs that the CANopen device is able to transmit.

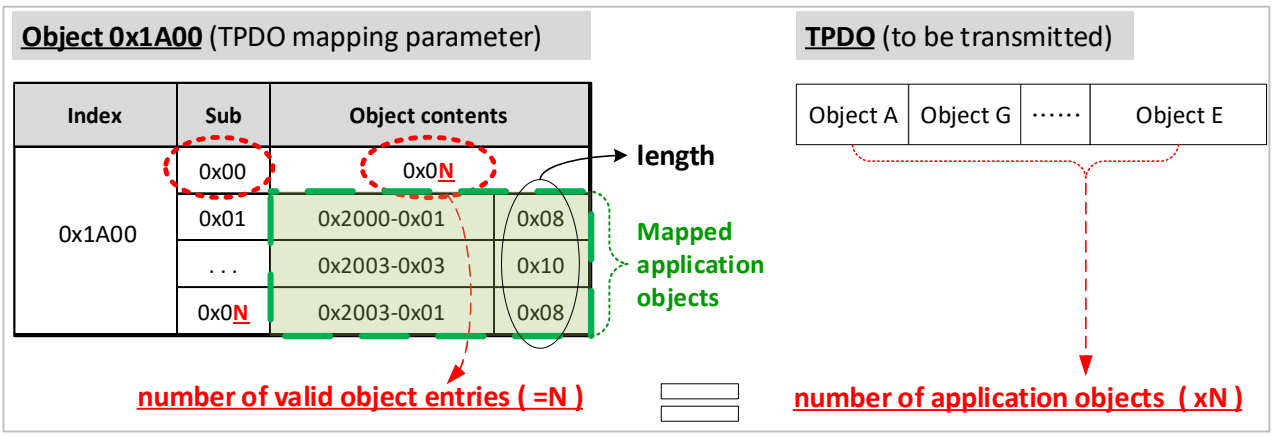
- Value definition

  Sub-index 0x00 contains the number of valid object entries within the mapping record.

<Value of RPDO mapping sub-index 0x00>

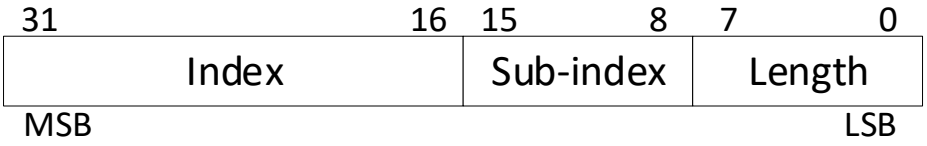
Value	Meaning
0x00	Mapping disabled
0x01	Sub-index 0x01 valid
0x02	Sub-index 0x01 and 0x02 valid
0x03	Sub-index 0x01 to 0x03 valid
0x04	Sub-index 0x01 to 0x04 valid
0x05~ 0xFF	Reserved

The number of valid object entries is the number of the application object that is transmitted with the corresponding TPDO.



Sub-indexes 0x01 to 0x04 contain the information of the mapped application objects. **The mapped application objects** describe the content of the PDO by their index, sub-index, and length. (See <The structure of TPDO mapping> and <The Principle of TPDO mapping>).
The length indicates the length of the application object in bits.

<The structure of RPDO mapping>



Trying to **change the value of an object entry** to a not-supported value will be responded with the SDO abort transfer service. Possible causes for a not-supported value would be:

- Mapping (index and sub-index) of a non-existing application object.
- Wrong length of the mapped application object.
- Wrong length of the PDO.

Remapping procedure:

Used when users need to configure a mapping different from the default. This procedure can take place during the NMT state Pre-operational and during the NMT state Operational:

1. Destroy TPDO by setting bit valid to 1_b of sub-index 0x01 of the according TPDO communication parameter (i.e., Object 0x1800~0x1803).
2. Disable mapping by setting sub-index 0x00 of Object 0x1A00 to 0.
3. Modify mapping by changing the values of the corresponding sub-indices.
4. Enable mapping by setting sub-index 0x00 of Object 0x1A00 to the number of mapped objects.
5. Create TPDO by setting bit valid to 0_b of sub-index 0x01 of the according TPDO communication parameter.

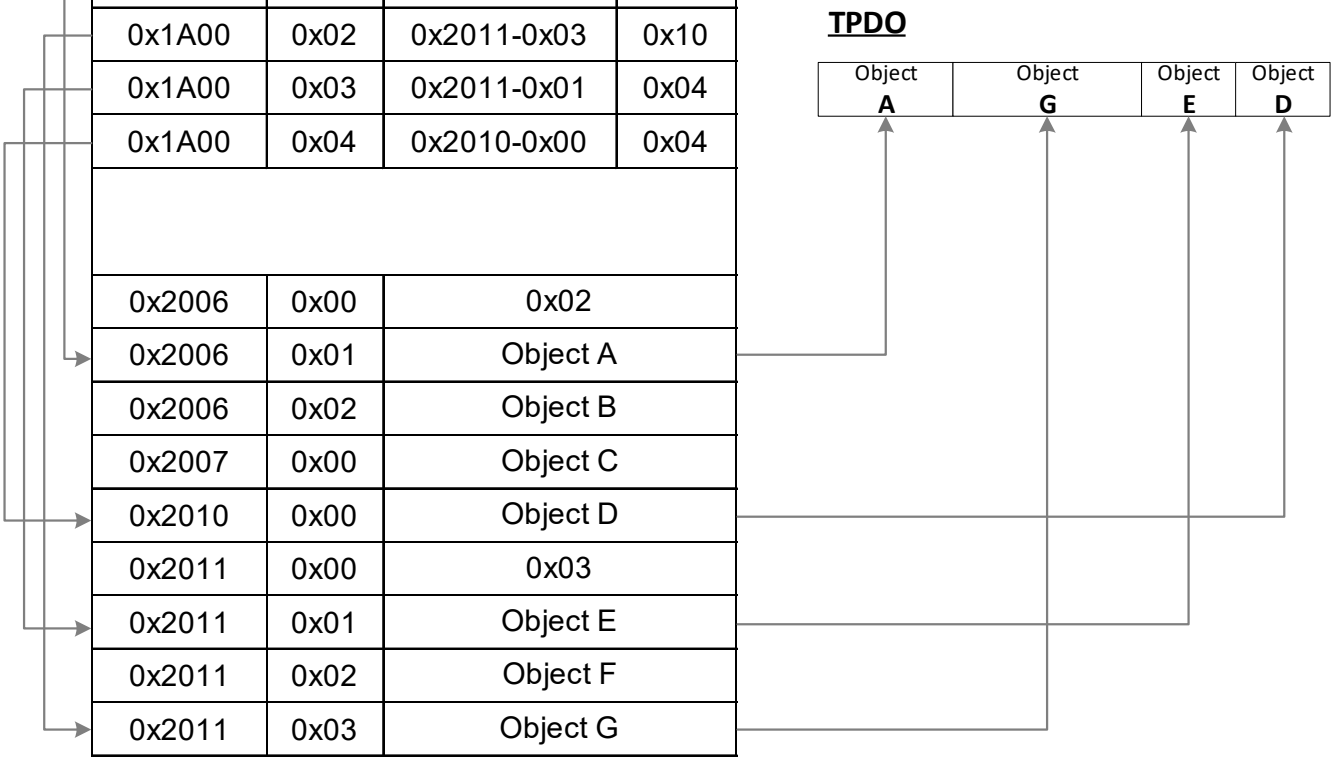
If during step 3 the CANopen device detects that index and sub-index of the mapped object does not exist or the object cannot be mapped, the CANopen device will respond with the SDO abort transfer service (abort code: 0604 0041h).

If during step 4 the CANopen device detects that the RPDO mapping is not valid or not possible, the CANopen device will respond with the SDO abort transfer service (abort code: 0604 0041h).

<The Principle of TPDO mapping>>

Object Dictionary

Index	Sub	Object contents	
0x1A00	0x00	0x04	
0x1A00	0x01	0x2006-0x01	0x08
0x1A00	0x02	0x2011-0x03	0x10
0x1A00	0x03	0x2011-0x01	0x04
0x1A00	0x04	0x2010-0x00	0x04
0x2006	0x00	0x02	
0x2006	0x01	Object A	
0x2006	0x02	Object B	
0x2007	0x00	Object C	
0x2010	0x00	Object D	
0x2011	0x00	0x03	
0x2011	0x01	Object E	
0x2011	0x02	Object F	
0x2011	0x03	Object G	



(See Object description and Entry description on subsequent pages)

- Object description

Index	1A00 ~ 1A03
Name	TPDO mapping parameter
Object code	Record
Data type	PDO mapping parameter record

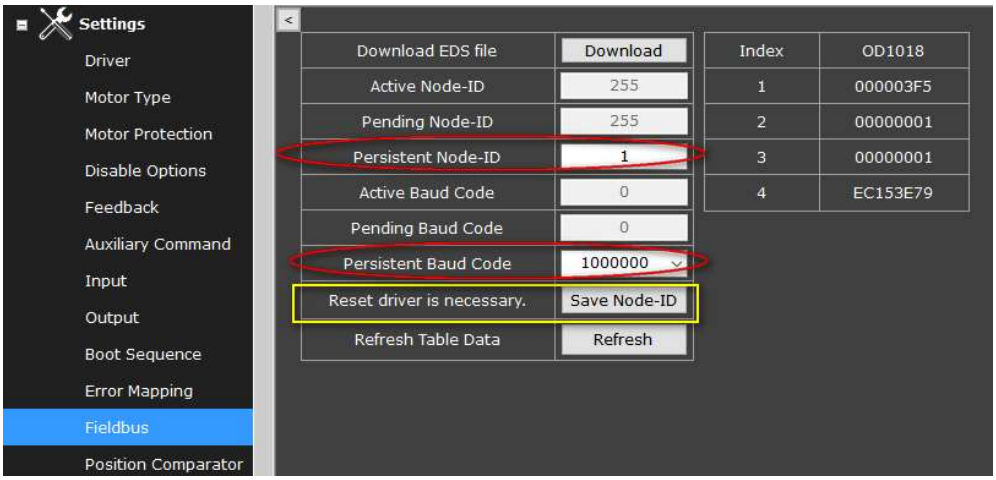
- Entry description

	0x1A00	0x1A01	0x1A02	0x1A03
Sub-index	**0x00**			
Description	Number of mapped application objects in TPDO			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	1	2	2	2
Sub-index	**0x01**			
Description	1st application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0x60410010	0x60410010	0x60410010	0x60410010
Sub-index	**0x02**			
Description	2nd application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0x60640020	0x606C0020	0x606C0020
Sub-index	**0x03**			
Description	3rd application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0	0	0
Sub-index	**0x04**			
Description	4th application object			
Access	RW			
PDO mapping	No			
Value range	See value definition			
Default value	0	0	0	0

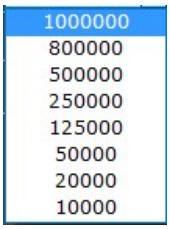
13. Initial CAN Communication Setup

In order to program the communication parameters, communication must first be established with the servo drive (*note), so that the RS-232 communication channel, which is always active, can be used. After RS-232 channel is set up, use the cpc GUI (graphic user interface) software to configure baud rate and Node-ID as described as below.

1. Follow the manual of cpc GUI user guide to install the software.
2. Go to the Settings > Fieldbus section in the GUI and configure the parameters of “Persistent Node-ID” and “Persistent Baud Code”.



- Persistent Node-ID: ranges from 1 to 127.
- Persistent Baud Code: select the preferred desired baud rate.



3. Click “Save Node-ID” button to save settings.

The panel on the right side shows the content of Object 0x1018. The master device needs this information to differentiate from salve devices.

***Note**: **Users can also set up baud rate and Node-ID according to the methods stated in CiA DS305.**

End of Document