Introducing the industry standard

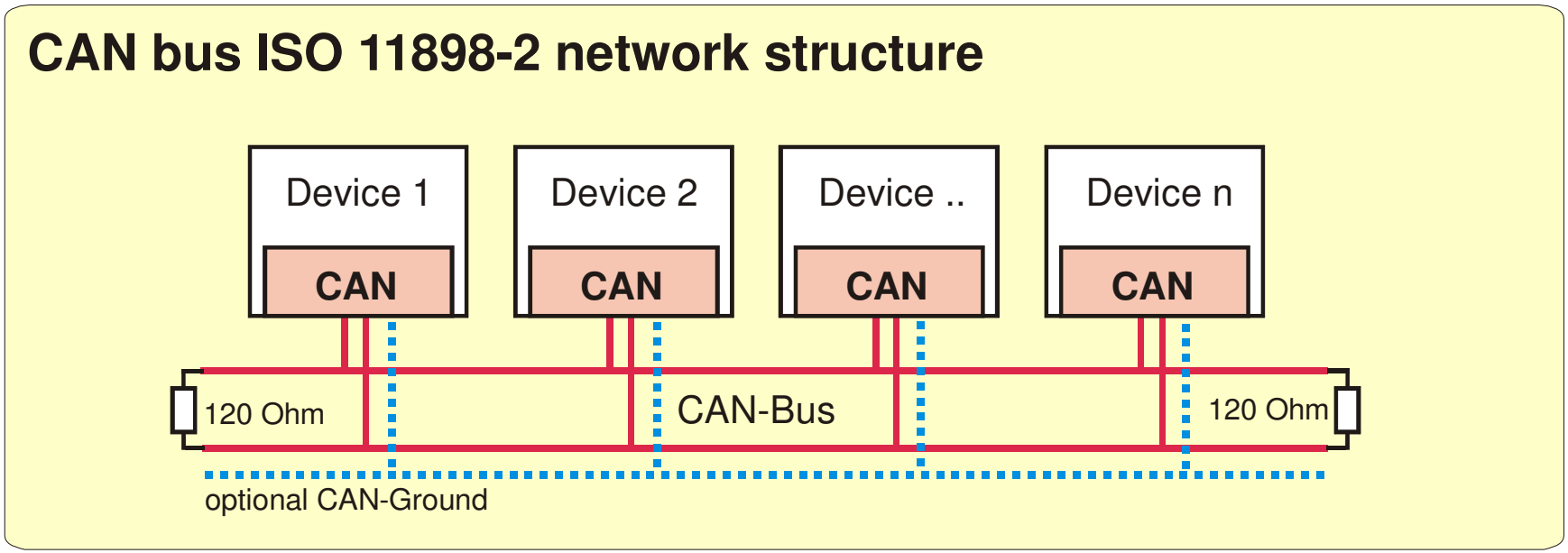
CANopen

1. Introduction to CAN

CAN is short for ‚Controller Area Network' and was originally developed by the companies Bosch and Intel as a bus system for vehicles. In the meantime, CAN is a very popular field bus for the scope area of industrial automation.

The CAN bus is a serial bus system. CAN uses a bus structure with a differential pair of bus lines for data and an optional CAN-Ground. The bus lines must be terminated with 120 Ohms at both ends. This bus structure is standardized in ISO11898-2.

All devices, also called bus nodes, are connected in parallel, so each data telegram, that was sent to the bus, is received by all devices.



1.1 CAN frames

The telegrams for data transmission are also named CAN frames. The number of bits of a telegram depends on the size of the data field. The CANopen protocol only modifies the arbitration field, that holds the CAN identifier, and the data field. All other bits of a CAN frame are modified by the hardware of a CAN module.

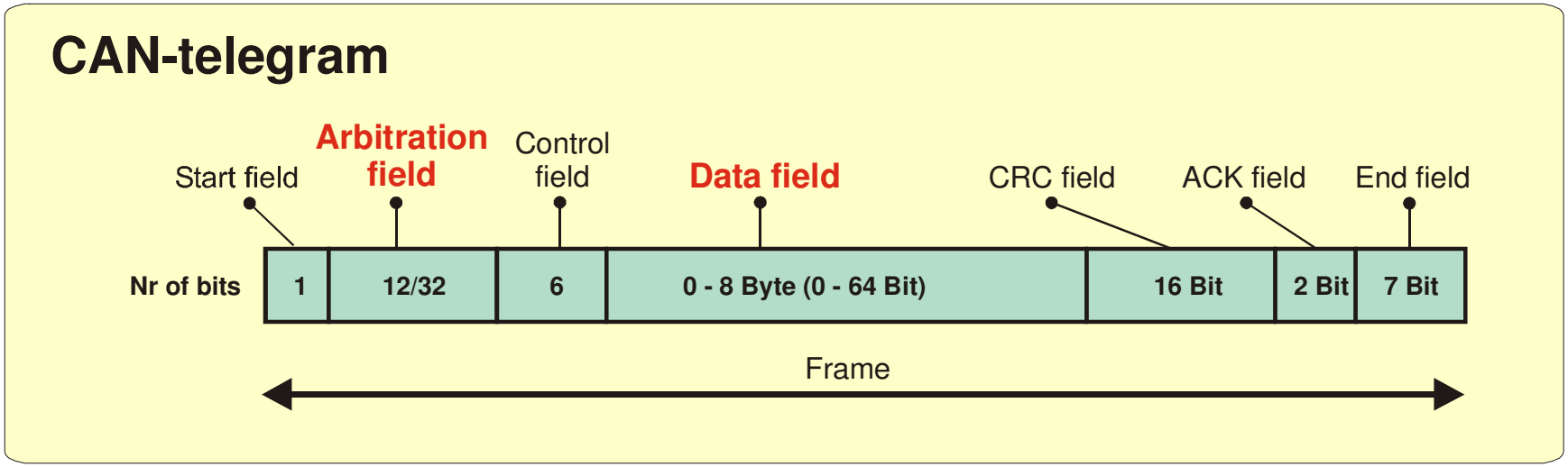


Figure 1: CAN-telegram (bit-stream)

In the ISO11898 standard, two telegram formats are defined. There is the standard format with a 11 bit identifier (used in CANopen networks) and the extended format with a 29-bit identifier (for example used in J1939). The architecture of the telegram allows the use of both types in one network.

There are two levels defined on the bus line:
Recessive bus level : is generated from the CAN bus transceiver, for transmitting bit value 1.
Dominant bus level : is generated from the CAN bus transceiver, for transmitting bit value 0.
If a dominant and a recessive bit level is sent from different nodes at the same time, the dominant level overwrites the recessive level.

1.1.1 CAN identifier and control field

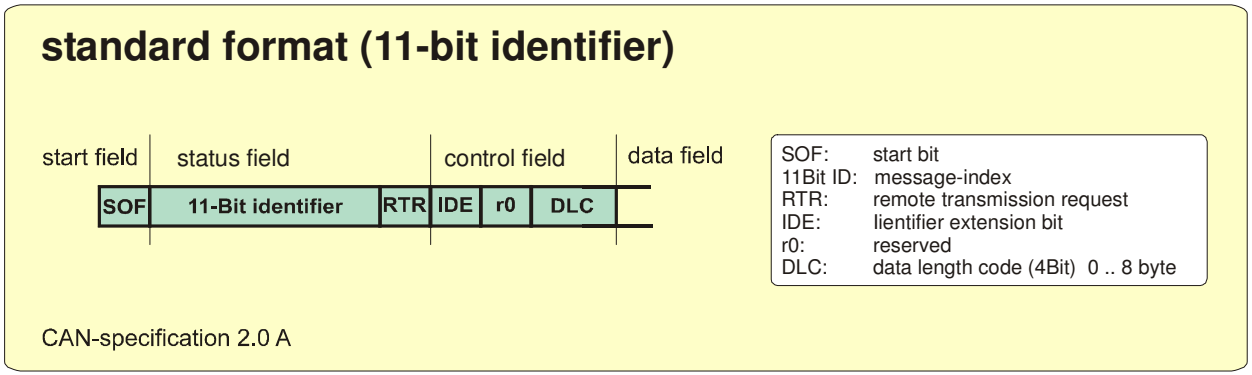


Figure 2: CAN-telegram in standard format 11-bit ID

For telegrams using 11 bit identifiers, the minimum telegram length is 44 bit at 0 data byte information

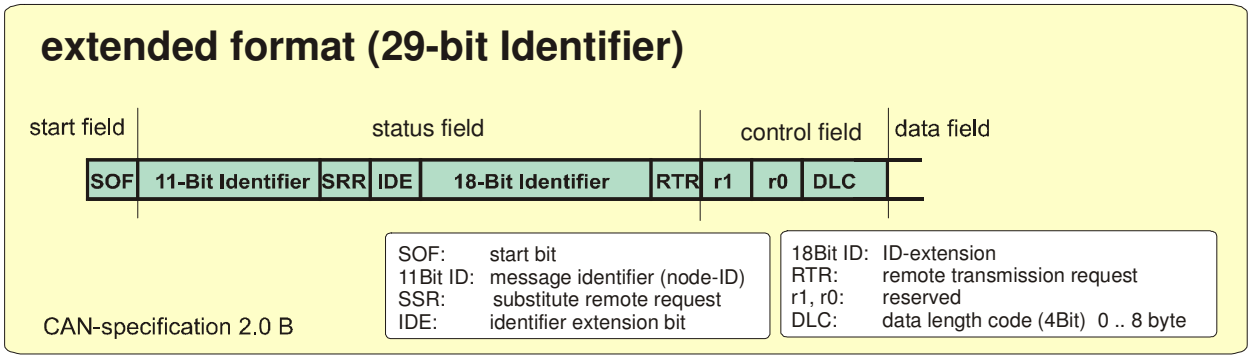


Figure 3: CAN-telegram in standard format 29-bit ID

For telegrams using 29 bit identifiers, the minimum telegram length is 62 bit at 0 data byte information

1.2 Arbitration in CAN networks

The CAN bus is a network with multimaster functionality. All devices have same rights. The CAN bus protocol allows simultaneous bus access from different nodes. If more than one node is accessing the bus at the same time, a non-destructive, bit-wise arbitration method is used, to grant the bus to the device that transmits the message with highest priority.

When the bus is in Idle state, several nodes may start transmission of a frame. Every node reads back the bit stream, from the bus during complete message transmission and compares the transmitted bit value with the received bit value. Per definition the bits with dominant values overwrite those with recessive values. If a transmitter wants to transmit a recessive bit to the bus line, but reads back a dominant bus level, the transmitter will immediately stop sending its own frame, so transmission of the frame with the first dominant bit position will be completed.

Note:

This arbitration method some preconditions must be guaranteed:

1) Arbitration must be done within the arbitration field:
   Any CAN identifier must be sent only from one node.
   It is forbidden for the devices to use same CAN identifiers for transmitting data frames.
   The same identifier however may be received by multiple nodes.

2) The prioritization is always processed within a single bit. Bus line extension must be limited in that way, that transmission and reading back of a bit over the complete the network must be possible within the time of 1 bit time (accurate ¾).
   This is a limitation of bus length and CAN baud rates.

1.3 Bus length

The following bus length table approximately result from the requirements of the arbitration method.

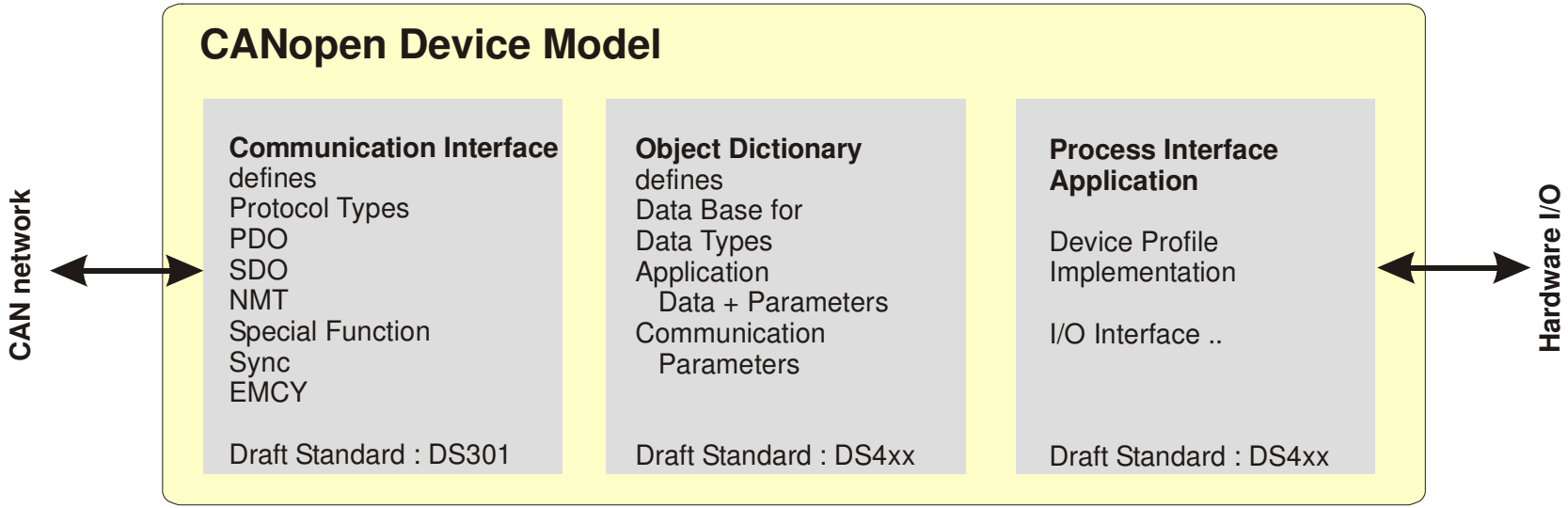
Bit rate	bus length	nominal bit time
1 Mbit/sec	30 m	1 usec
800 kBit/sec	50 m	1,25 usec
500 kBit/sec	100 m	2 usec
250 kBit/sec	250 m	4 usec
125 kBit/sec	500 m	8 usec
50 kBit/sec	1000 m	20 usec
20 kBit/sec	2500 m	50 usec
10 kBit/sec	5000 m	100 usec

The shown bus lengths are approximations for use of ISO11898 compliant drivers, standard bus conductors without regard of opto couplers.

2 CANopen

CANopen is the open protocol standard for CAN in the field of automation technology and has been standardized by the association „CAN in Automation“ (CiA). CANopen normally uses a master slave concept. Each CANopen device (also called node) has a unique node ID, that is used to identify a node and to generate CAN identifiers for various messages.

CANopen defines various communication protocols for data exchange and also device profiles in order to specify the functional behavior for various application types such as I/O modules, drives, encoders, sensors etc. The complete process data and parameter set is held within an object dictionary, that is also defined in the device profiles.



A CANopen device can be divided into three parts:

1) The communication interface and protocol stack enables access to the object dictionary of the device.
2) The object dictionary is working as data base and holds all process and configuration data of the device.
   The object dictionary is working as an interface from CAN bus line to application
3) The process interface and application program is running the application itself and processes the hardware interface.

2.1 Draft Standards and Device Profiles

CANopen defines the application layer (OSI-layer 7) as a communication profile, which was standardised by the CiA in the Draft Standards DS30x for all applications. The Draft Standard DS301 describes all object dictionary entries for the communication layer. This standard is unique for all devices.

The Draft Standards DS4xx specify the generic and specific device profiles with the functional behaviour of a logical device. Application profiles describe a set of virtual device interfaces. The functionality and process image is represented within the object dictionary. The object dictionary is a data set holding all process data elements and parameters.

Draft Standard (CiA)	for the device types
DS 301	Communication profile
DS 401	Device profile for digital and analog I/O, joystick applications, etc
DS 402	Device profile for drives
DS 404	Device profile for sensors / regulators
DS 405	Device profile for programmable devices such as PLC systems
DS 406	Device profile for encoders
DS …	(further device types)

Example of device profiles

2.2 Object directory

The object directory is the data set of all process data, variables and parameters (called objects) of a CANopen device. The data shows the process image (for example state of digital input channels) and with the parameters the functional behavior of a CANopen device can be influenced (for example inverting the digital input channels).

The object dictionary is the interface from communication layer to the application layer of a CANopen device. For example the application layer reads the digital input bitmap from hardware pins to the object dictionary and the communication layer transmits these information to the CAN bus line using a PDO (Process Data Object).

All objects are addressed using an index and for complex data types such as arrays and records (structures) a sub index.

object index (hex)	object
0000	not used
0001 - 001F	static data types
0020 - 003F	complex data types
0040 - 005F	manufacturer specific complex data types
0060 - 007F	device profile specific static data types
0080 - 009F	device profile specific complex data types
00A0 - 0FFF	reserved for further use
1000 - 1FFF	communication profile area
2000 - 5FFF	manufacturer specific profile area
6000 - 9FFF	standardized device profile area
A000 - FFFF	reserved for further use

Structure of a CANopen object directory

Example for an object directory:

Part of object dictionary for CANopen chip CO4011. The CO4011 is a standard I/O module, it uses the draft standard DS401, that defines the structure of the object dictionary.

index	*sub-index*	*Name*	*Functionality*	*Access*	*PDO-mapping*
0005	*-*	*dummy 8*		*ro*	*Yes*
0006	*-*	*dummy 16*		*ro*	*Yes*
100C	*-*	*guard time*	*These two objects are used to configure the node- and life-guarding functionality of the device*	*rw*	*-*
100D	*-*	*life time factor*		*rw*	*-*
100E	*-*	*COB-ID guard*		*rw*	*-*
1014	*-*	*COB ID emergency*		*rw*	*-*
1015	*-*	*inhibit time emergency*		*rw*	*-*
1017	*-*	*producer heartbeat time*		*rw*	*-*
1018		*identity object*			
	0	*(no. of subentries)*		*ro*	*-*
	1	*vendor ID*		*ro*	*-*
	2	*product code*		*ro*	*-*
	3	*revision number*		*ro*	*-*
2000	*-*	*device manufacturer*		*ro*	*-*
2101	*-*	*system configuration*		*ro*	*-*
6000	*0 to n*	*read digital input 8-bit*	*This object holds the bitmap of the hardware input pins*	*ro*	*Yes*
6002	*0 to n*	*polarity input 8-bit*	*With this object, an optional input pin inverter may be activated*	*rw*	*-*
6005		*global interrupt enable*		*rw*	*-*
6006	*0 to n*	*Interrupt mask: any change*		*rw*	*-*
6007	*0 to n*	*Interrupt mask rising edge*		*rw*	*-*
6200	*0 to n*	*Digital output*	*This object is written from CAN bus line in order to set the digital output pins of a CANopen device.*		

2.3 CANopen communication

The CANopen protocol standard defines several CAN message types for data exchange, network management and reporting device errors. All message types for data exchange are accessing the object dictionary of a CANopen device.

Message Type	Description	Default CAN-Identifiers
NMT	**Network Management Telegram** These telegrams are sent from the master to the slave nodes in order to control the network state of the slaves. - Highest priority CAN identifier - Broadcast message from master to all slaves Possible states of a CANopen device: stopped, preoperational, operational.	0x00
SDO	**Service Data Object** These telegram type is used to exchange configuration data. - used in device states preoperational and operational - Lower priority CAN identifier - Mainly used during bus start up. - Each telegram is initiated from master node. - Each telegram is answered, so transfer is slowly. - There is only one object (data from object dictionary) that can be exchanged - Data is addressed using index and sub index.	0x600 + Node-ID 0x580 + Node-ID
PDO	**Process Data Object** These telegram type is used to transfer process data (for example digital input bitmap) - High priority CAN identifier - PDOs may only be transmitted in operational device state. - Predefined data content of max. 8 bytes. No addressing using index and sub index. - Data transfer may be initiated from each node. - Data transfer is not answered	0x180 + Node-ID … 0x480 + Node-ID 0x200 + Node-ID … 0x500 + Node-ID

Message Type	Description	Default CAN-Identifiers
EMCY	**Emergency Message** These telegrams are sent in order to indicate an error condition of the device. - High priority CAN identifier	0x80 + Node-ID
SYNC	**Synchronization Message** This telegram is sent from master to all slaves, in order to synchronize exchanging of process data with hardware and in order to cause transmission of PDOs - High priority CAN identifier - Message without any data content	0x80
Boot-Up	**Boot Up Message** This telegram type is used to indicate, that a node has performed a reset and is no ready to take part within the network communication. - Low priority CAN identifier - Sent only once after node has performed reset procedure.	0x700 + Node-ID
Error-Control	**Error-Control Protocol** These telegram types are used to monitor the device state. The protocols are used to detect breakdowns of slaves or the master in order to enter fail safe condition states. There are two types of Error-Control Protocols: - Node-Guarding / Life Guarding The master polls each slave individually. The slave sends an answer protocol if still alive. - Heartbeat Each node periodically transmits its NMT state.	0x700 + Node-ID

2.4 Default Identifier Allocation

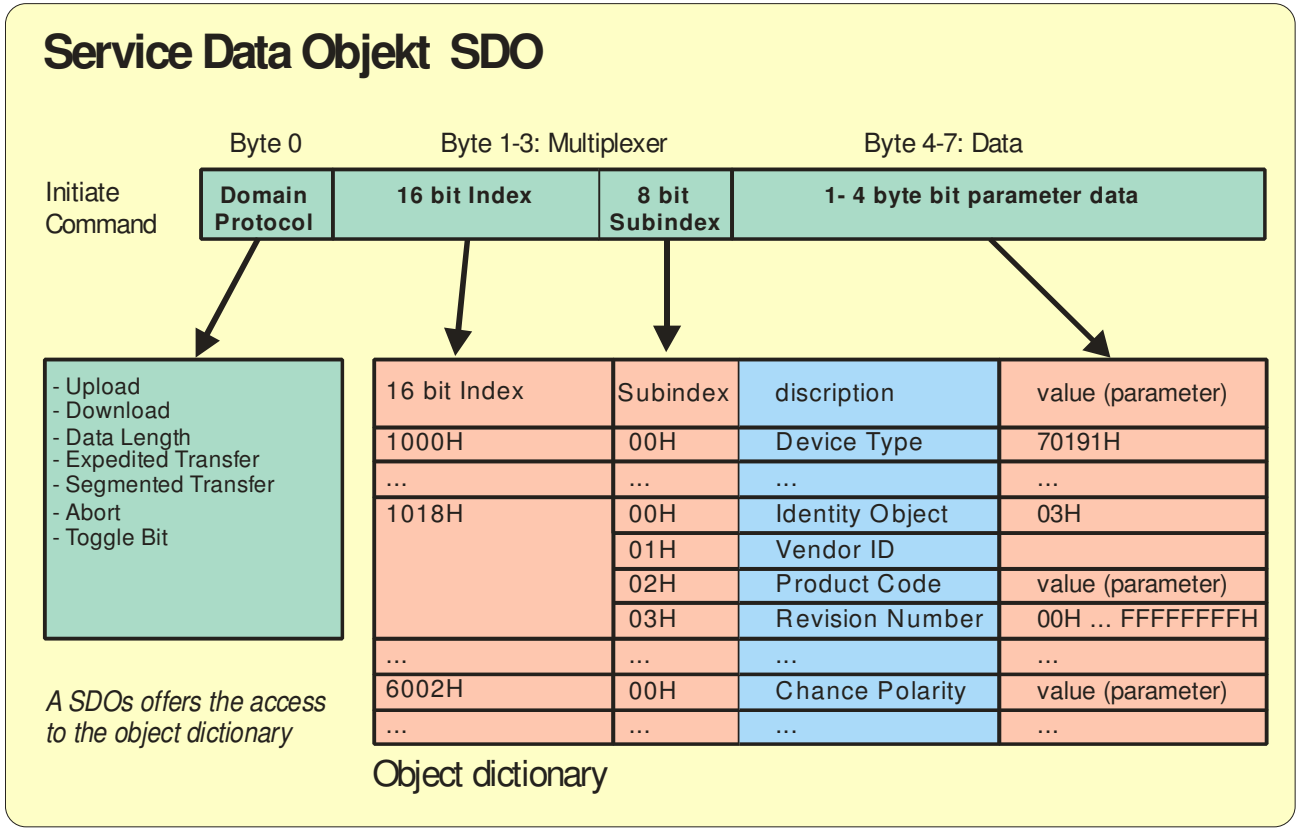
A default identifier allocation saves configuration effort. Thereby the CAN identifier is based on the devices node number.

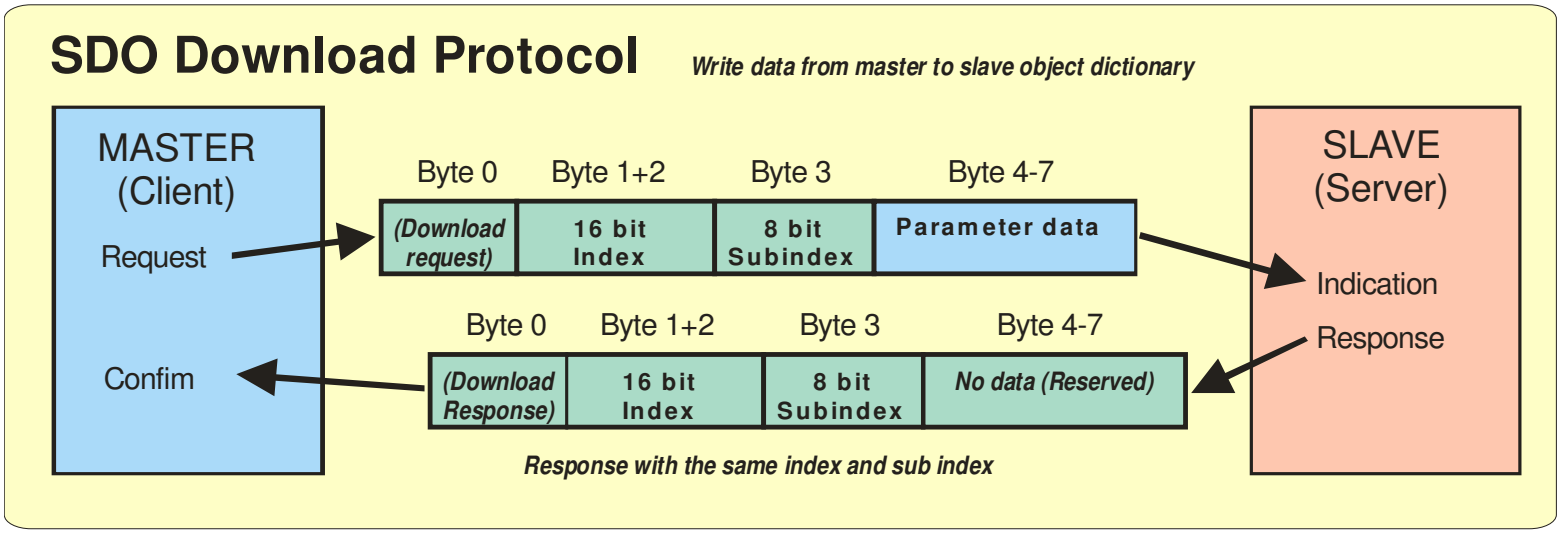
The default identifier allocation is defined in CAN open as followed:

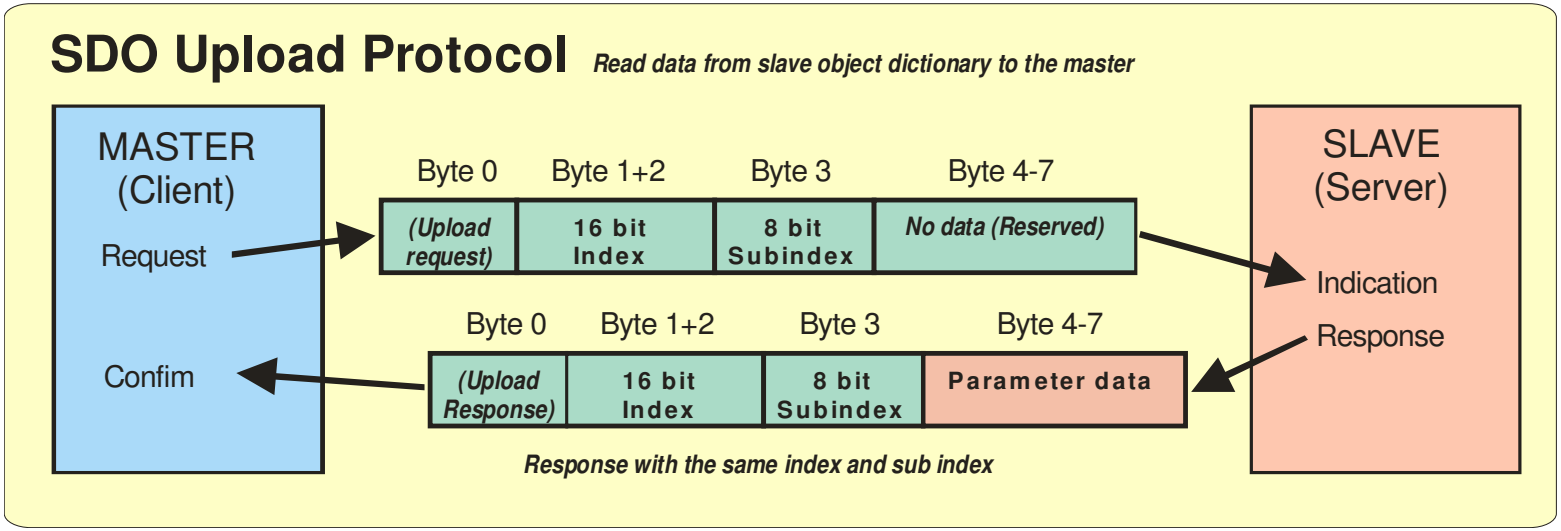
Identifier 11-bit (binary)	Identifier (decimal)	Identifier (hex)	Telegram Type / Function
00000000000	0	0	NMT : Network Management Telegram
00010000000	128	80h	SYNC : Synchronization Message
0001xxxxxxxx	129 - 255	81h - FFh	EMCY : Emergency
0011xxxxxxxx	385 - 511	181h - 1FFh	PDO1 (tx)
0100xxxxxxxx	513 - 639	201h - 27Fh	PDO1 (rx)
0101xxxxxxxx	641 - 767	281h - 2FFh	PDO2 (tx)
0110xxxxxxxx	769 - 895	301h - 37Fh	PDO2 (rx)
0111xxxxxxxx	897 - 1023	381h - 3FFh	PDO3 (tx)
1000xxxxxxxx	1025 - 1151	401h - 47Fh	PDO3 (rx)
1001xxxxxxxx	1153 - 1279	481h - 4FFh	PDO4 (tx)
1010xxxxxxxx	1281 - 1407	501h - 57Fh	PDO4 (rx)
1011xxxxxxxx	1409 - 1535	581h - 5FFh	SDO send
1100xxxxxxxx	1537 - 1663	601h - 67Fh	SDO receive
1110xxxxxxxx	1793 - 1919	701h - 77Fh	Boot Up and Error Control Protocol
xxxxxxxx = node number 1 - 127			

2.5 SDO : Service Data Object

A SDO is an access of exactly one object of the object dictionary using the objects index and sub index as an address (also named multiplexer). It always uses 8 data bytes of the CAN message. SDOs are always initiated from the CANopen bus master, and are answered from the slave node.
The principle structure of SDOs shows the usage of the CAN telegram data bytes.




SDO Download Protocol
Write data from master to slave object dictionary
MASTER (Client)
Request
Confim
SLAVE (Server)
Indication
Response
Byte 0
Byte 1+2
Byte 3
Byte 4-7
(Download request)
16 bit Index
8 bit Subindex
Parameter data
Byte 0
Byte 1+2
Byte 3
Byte 4-7
(Download Response)
16 bit Index
8 bit Subindex
No data (Reserved)
Response with the same index and sub index


SDO Upload Protocol
Read data from slave object dictionary to the master
MASTER (Client)
Request
Confim
SLAVE (Server)
Indication
Response
Byte 0
Byte 1+2
Byte 3
Byte 4-7
(Upload request)
16 bit Index
8 bit Subindex
No data (Reserved)
Byte 0
Byte 1+2
Byte 3
Byte 4-7
(Upload Response)
16 bit Index
8 bit Subindex
Parameter data
Response with the same index and sub index

2.6 The process data object PDO

The process data exchange via CANopen is done with standard CAN messages without protocol overhead.

For each PDO, there is a configuration object in order to specify the transmission mode of a PDO and additionally there is a mapping object, that predefines the data content of a PDO. Up to 8 byte of data can be transmitted within a PDO message.

Transmission of a PDO may be initiated in several transmission modes:
- asynchronous PDO transmission is initiated on an event. (For example change of mapped data)
- synchronous PDO transmission is initiated by the SYNC message
- periodically PDO transmission is initiated from an event timer, that triggers transmission periodically
- polling PDO transmission is initiated, if the node has received a remote frame with same CAN identifier.

2.6.1 PDO Communication Parameters

For each PDO there is a setting object, that defines the communication parameters just like CAN identifier and the transmission mode for the related PDO. The communication parameter objects are using data type record.

Object Index	Sub Index	PDO	Parameter	Description
14xx		RPDO	Communication Parameters for Receive-PDO xx	
18xx		TPDO	Communication Parameters for Transmit-PDO xx	
14xx/18xx	00		Nr of Sub	Number of valid sub index for this object
	01		COB-ID	CAN message identifier, that is used for this PDO
	02		Transmission Type	Transmission Type for the PDO.
	03		Inhibit Time	Minimum time interval from one PDO transmission to the next
	04		-	Implemented because of compatibility reasons without any function
	05		Event Time	Time interval to trigger PDO transmission periodically

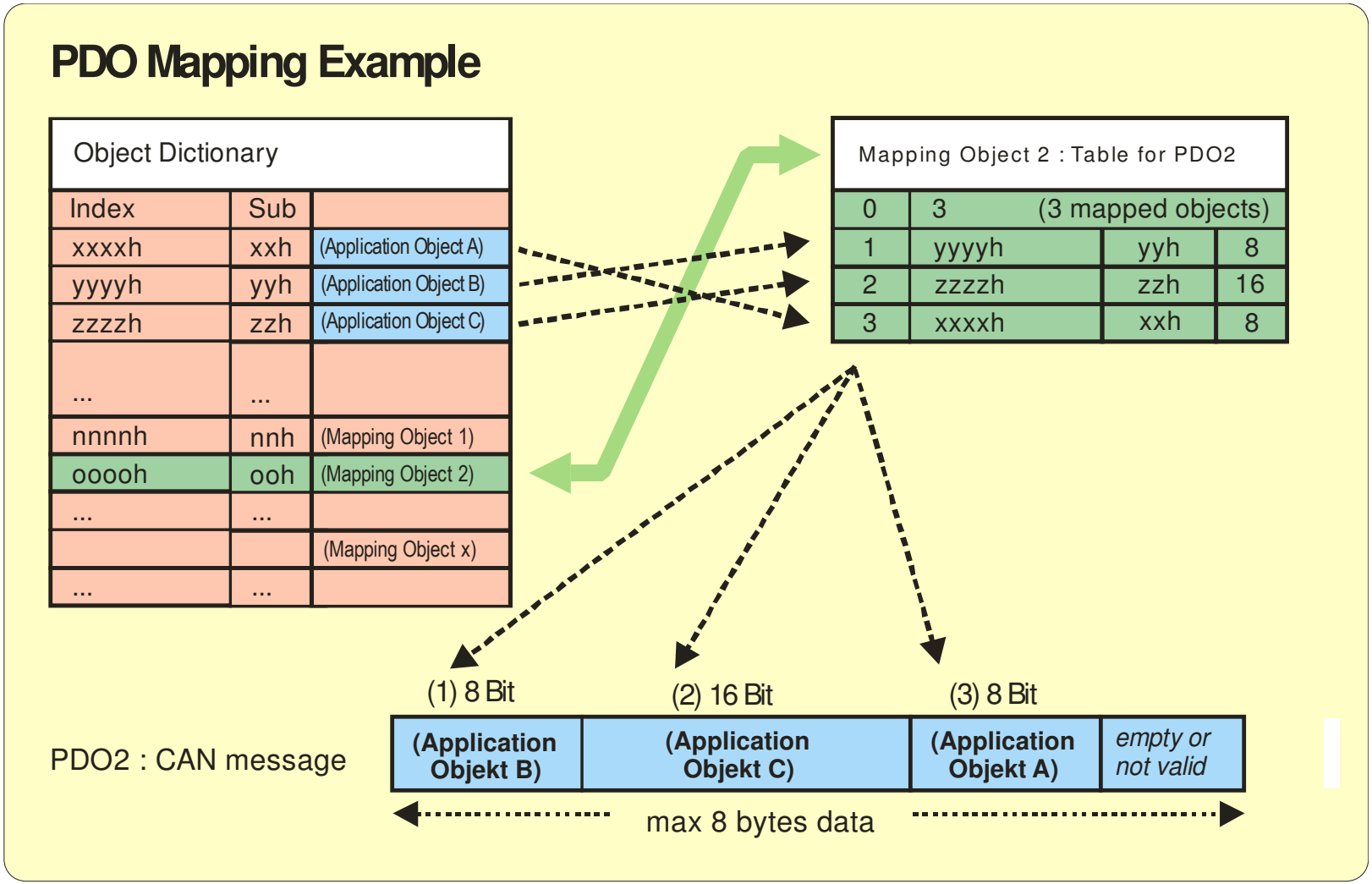
2.6.2 PDO Mapping Parameters

With the PDO mapping the data content of a PDO is preconfigured within a mapping table. This mapping table holds the index, sub index and data size of each object, that is transmitted within the PDO message.

The mapping table for each PDO is represented as an object within the devices object dictionary. The mapping table is realized as an array of mapped objects.

Object Index	Sub Index	PDO	Parameter	Description
16xx		RPDO	Mapping Parameters for Receive-PDO xx	
1Axx		TPDO	Mapping Parameters for Transmit-PDO xx	
16xx/1Axx	00		Nr of Sub	Shows the number of mapped objects within one PDO message
	01 .. n		Mapping Entry	Shows index, sub index and data size given in bits, of a mapped object, that is transferred within the PDO message

PDO Mapping Example



2.7 NMT : Network Management

For controlling the network status of all slave nodes, the CANopen master uses several NMT messages. The NMT messages are highest priority messages with CAN identifier 0 and are always using two data bytes.

	Identifier	Data Byte 0	Data Byte 1
NMT telegram	0x000	NMT command	Slave Node Id

The first data byte holds the NMT command that switches the Network State, and the second data byte holds the Node-ID of the CANopen slaves, that are addressed. If the Slave Node ID is 0, the NMT has to be processed by all connected slave nodes.

NMT Command Byte-Value	**Command**	**Description**
0x01	Operational	Switches to Operational state enables PDO transfer. Typically the operational state with PDO transfer is the network state for running systems after the master has finished initialization of all slaves.
0x02	Stop Node	In status stopped, no PDO and no SDO communication is allowed.
0x80	PreOperational	Switches to Operational state. In Preoperational state no PDO, but only SDO transfer is allowed. After device start up, a CANopen node switches to PreOperational state automatically. This is the typical network state during initialization procedure.
0x81	Reset Node	Performs a complete reset of the CANopen device
0x82	Reset Communication	Resets all objects from object dictionary that hold communication relevant parameters to default values

2.8 EMCY : Emergency Message

With emergency messages, a CANopen device reports any change of an error condition to the bus. An emergency message is always 8 data bytes long.

Identifier	Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	Data Byte 4	Data Byte 5	Data Byte 6	Data Byte 7
0x80 + Node-ID	EMCY Code		ErrReg	Manufacturer specific error code				

EMCY Code Emergency error code. Codes are defined in Draft Standard DS301.
ErrReg Error Register. The error register is implemented as object 1001 within the devices object dictionary. This object is also inserted into emergency messages.

A CANopen device transmits an emergency message, if setting or resetting of any error condition is detected.

2.9 SYNC : Synchronization Telegram

The SYNC message is sent as a broadcast message from the CANopen master to all slaves. It is used to enable synchronous data exchange from object dictionary to application and vice versa. For example reading of digital input pins might be triggered for all CANopen devices at the same time, using a SYNC message. The SYNC message normally uses CAN identifier 0x00 an no data bytes. The SYNC telegram works only on the PDOs of a slave device, that are configured for synchronous transmission within the PDO communication parameter objects.

Identifier
0x80

2.10 Error Control Protocols and Boot-Up Message

Error control protocols are implemented, in order to enable monitoring of the state of nodes. These protocols are important to switch to uncritical system states in case of crashes of single nodes or bus lines. For example, a drive should stop motion in case of a PLC crash.

The Error Control Protocols uses a CAN telegram with CAN identifier 0x700 + Node-Id and one data byte, that reports the NMT state of the node.

Identifier	Data Byte 0
0x700 + Node-ID	NMT-State

The NMT State is reported as follows:

0x00 Boot-Up Message
0x05 Operational
0x7F Preoperational

There are three types of error control protocol.

2.10.1 Boot Up Message

As mentioned above, the boot up message is a special version of error control protocol with NMT state 0x00. This message is sent for exactly one time after a node is started or has performed a reset sequence in order to indicate the start up to all nodes of the CANopen network.

2.10.2 Node Guarding

When in Node-Guarding mode, the bus master is polling the CANopen device using a remote frame. The slave responds with the error control protocol, but adding a toggle bit (most significant bit) to the NMT state, that is reported in the data byte. The toggle bit is toggled after each successful transmission of the error control protocol.

With node guarding, network monitoring is possible for both directions.

The slaves waits for an incoming polling message that must be sent from CANopen master. If the slave does not receive the polling message, it switches the application to a fail safe state. For example a drive will perform an emergency stop, an I/O node will switch of all digital output pins.

For configuration of the Node Guarding protocol, each CANopen device has two objects implemented within the object dictionary. These objects must be initialized from a master node during network start up procedure, in order to activate the node guarding features.

Object Index	Name	Description
0x100C	Guard Time	Time period for polling the slave node.
0x100D	Life Time Factor	Multiplication factor for the Guard Time. Life Time = Guard Time * Life Time Factor If a slave receives no polling telegram within the life time (product of Guard Time and Life Time Factor), the node enters the fail safe state.

2.10.3 Heart Beat

If Heart Beat protocol is used, the node transmits its error control message cyclically to the bus. There is no polling etc. With the Heart Beat protocol, only the receiver can monitor the Heart Beat producing node.

For configuration of the Heart Beat protocol, each CANopen device has one object implemented within the object dictionary for each direction. These objects must be initialized from a master node during network start up procedure, in order to activate sending or monitoring of the Heart Beat messages.

Object Index	Name	Description
0x1016	Consumer Heart Beat Time	Configures the CAN identifier and time period for monitoring incoming heart beat messages.
0x1017	Producer Heart Beat Time	Configures the time period for transmitting heart beat messages.

Please Note:

For one CANopen node, only one type of error control either Node Guarding or Heart Beat is allowed at the same time.

2.11 EDS File

For the user of a CANopen device, the object directory is saved as EDS-file (electronic data sheet). The EDS file shows the complete device information. All objects are saved with index, sub index, name, data type, default value, minimum, maximum and access rights (read/write, transfer only via SDO or even via PDO etc.). Thereby, using the EDS-file, the whole functionality of a CANopen device is described.

For building a CANopen network, normally the EDS files for all nodes are read into a network configuration tool.

3 Example of a simple CANopen Network Start Up

For this example we use a CANopen master with node ID 1 and a simple digital I/O module as a slave node with node ID 3
All values are given in hex. <> brackets are used to indicate a value as CAN message identifier

Messages from Master	Messages from Slave	Description
…		
	<703> 00	Boot Up message from slave node
<603> 40 00 10 00 00 00 00 00		SDO request from Master in order to read object 1000
	<583> 42 00 10 00 91 01 03 00	SDO answer from Slave with data content of object 1000
<603> 22 17 10 00 F4 01 00 00		SDO Master writes 0x1F4 (500dec) to object 1017 This activates the Heart Beat protocol
	<583> 60 17 10 00 00 00 00 00	SDO write acknowledge from slave
	<703> 7F	Heart Beat message from slave indicating NMT state PREOPERATIONAL
<000> 01 03		NMT : start node message from master switches slave with Node ID 3 to OPERATIONAL state
	<183> 00 00	TPDO1 from slave with digital input bitmap
	<703> 05	Heart Beat message from slave indicating NMT state OPERATIONAL
	<183> 02 81	TPDO1 from slave with digital input bitmap, 3 input pins are scanned high
<203> 00 01		RPDO1 transmitted from master to slave in order to set one digital output