

Setup and configuration of a CANopen sub-system

Summary

This is a short introduction to the basics of CAN communication.

Chapter CAN-Overview (page 2) summarizes the basics of CAN communication.

Chapter CANopen-overview (page 10) presents the services defined in a CANopen environment. The typical sequence of CAN messages during start and operation of a CANopen environment is shown.

The startup of a CANopen sub-system is explained starting from page 25.

The last chapter – robust configuration (page 29) gives some advice about the robust configuration of CANopen and a CANopen servo-drive.

Applies To

All FAULHABER MotionControllers and MotionControl Systems having a CANopen interface.

Glossary

CAN-controller	The message frame is fixed and is generated by dedicated CAN controllers, which meanwhile are typically integral part of a μ Controller. In the early days of CAN stand-alone CAN-controllers like the SJA 1000 have been combined with general-purpose controllers.
CANopen master	In a CANopen system, there is typically exactly one CANopen master and several CANopen slaves. The main task of the CANopen master is to monitor the network, configure the slaves if necessary and step through the different stages of the initialization.
Application master	While the CANopen master is responsible for the communication according to the CANopen protocol the application master will be the one to really use the data and start any behavior of the slaves
node	An otherwise unspecified device connected to a CAN. Could be a master or slave
SOF	S tart O f F rame of a CAN message
Component	A device within a machine. Could be a drive, a sensor, ...
PLC	The programmable logic controller. A component, which typically includes the application master and the CANopen master.

Description

CAN Overview

BOSCH introduced **C**ontroller **A**rea **N**etwork as an automotive network back in the middle of the eighties.

Messages are exchanged using a single pair of wires analog to a RS485 interface. The transmission of the bits is differential using the lines CAN_H and CAN_L. If there is no message present, the voltage levels on both lines should be around 2.5V. An active bit will result in a differential signal of typically 1V.

The physical interface of a CAN is a voltage signal. Therefore – as in a RS485 environment – a common ground (GND) is also required. The GND levels within a CAN must not exceed a level of ±2V.

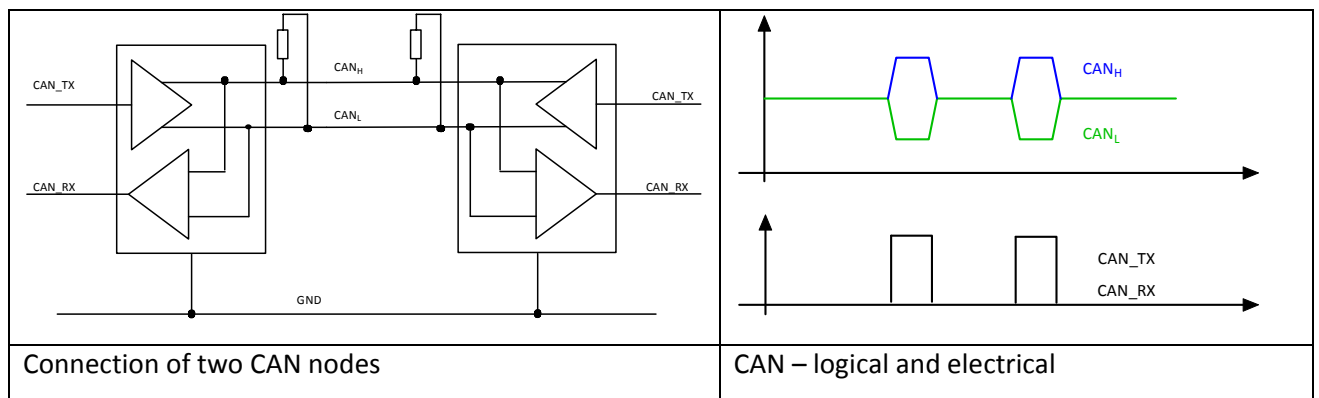


Figure 1

CAN Frame

The message frame is fixed and is generated by dedicated CAN controllers, which meanwhile are typically integral part of a μ Controller. In the early days of CAN stand-alone CAN-controllers like the SJA 1000 have been combined with general-purpose controllers.

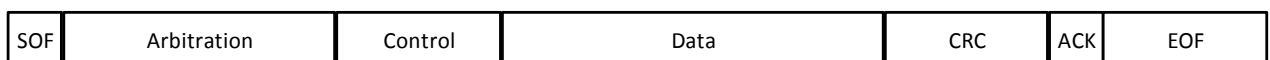


Figure 2 CAN-Frame

Arbitration field

As all the participants are connected in parallel, frames might collide if more than one controller starts a transmission after the minimum idle time of the bus.

CAN uses a specific bus access mechanism, called carrier-sense multiple access with collision resolution (CSMA/CR). The access to the bus is granted based on the identifier of a message which is coded in the arbitration area of the CAN-frame.

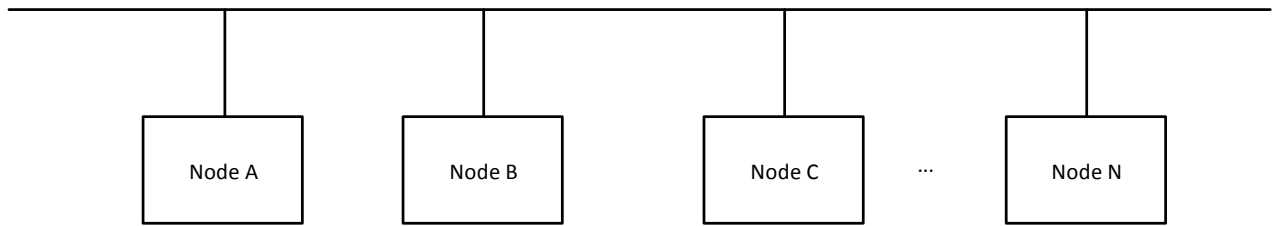


Figure 3 CAN as a network of equals

The identifier field carries either 11 bits or 29 bits. In a CAN environment without any additional protocol definition messages are identified by their identifiers only. In CANopen environments 11 bits identifiers are used. This allows for 2048 different messages to be used. The identifier is used by the nodes to identify the message and by the CAN-controllers to gain the bus access. The message having the lowest identifier will win the arbitration. During the arbitration several CAN controllers could write their message identifier to the bus. The message with the lowest identifier will be dominant; the other nodes will stop their access and will listen only. To avoid collisions every message, identified by a specific identifier, should be assigned to a single sender.

Data and Control

A CAN frame can have a payload of 0 to 8 bytes of data. The actual length is coded in the Control field. Of course, there has to be an agreement between the interacting nodes how to interpret the contents of the data part. This is one of the responsibilities of the CANopen protocol.

Remote Transfer

CAN offers a special request – response mechanism even at the very low level. There is a so called **Remote Transfer Request (RTR)** bit within the control field. It is used to send a request for a specific message. The message is identified by its message id. If this feature is used, the requesting node will send a message with the expected Id having the RTR bit set. The data length is 0. The node responsible for the message having this Id shall react by sending the requested message – and of course in case of the response there will be some data. The response itself is however outside the responsibility of the CAN controller itself and will typically be prepared by some kind of driver firmware.

CRC and ACK

A CAN message is protected by a CRC. Every CAN-controller in a system will receive the frame and check the CRC. If the frame is received successfully the receiving nodes will write a dominant bit into the ACK field. Otherwise – if an error has been detected – the receiving controller will destroy the frame by sending dominant bits. These frames are called error frames. Monitoring tools will typically indicate the number of observed error frames. Standard CAN controllers will discard error frames automatically but will count their number.

A sending controller can therefore check, whether the frame has been transmitted successfully by reading the ACK field. If there is no dominant ACK it will re-send the frame until it is acknowledged.

This is a very powerful mechanism. As all the nodes connected to a CAN sub-system will listen and receive all the messages either all or no node will receive a message. If it has been disturbed, there will be no

acknowledge and the sender will resend it. So – even without an explicit handshake the successful transmission can be assumed as soon as the local CAN controller signals the message to be sent.

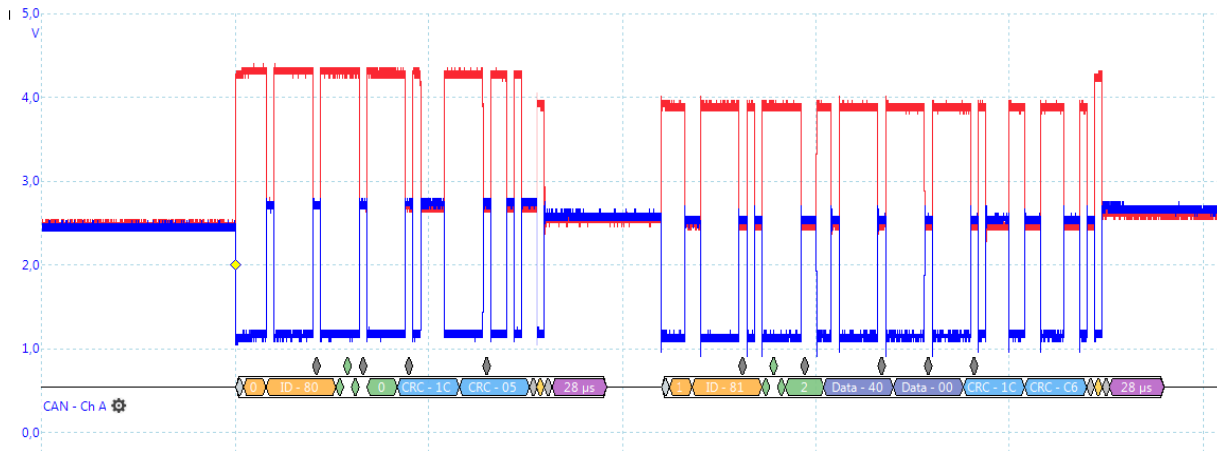



Figure 4 CAN frames

Termination Resistors

Due to this synchronization down to the ACK bit, the different CAN-controllers must be more or less perfectly synched. This is done by a re-synchronization during the SOF. To allow for this synchronization reflections at the end of the network have to be avoided. Therefore, termination resistors are mandatory. In a linear CAN system this would be a 120Ω resistor at each end of the network. In non-linear CANs the placement and value of the termination resistors might have to be optimized manually.



As a first test of a CAN system you might check the resistance measured between CAN_H and CAN_L. If both termination resistors are present, the measured resistance should be approximately 60Ω.

This is possible only when the system is electrically switched off.

Baud rates and bus topology


All nodes in a CAN sub-system have to use the same preconfigured baud-rate. Supported baud-rates of FAULHABER MotionControllers are:

Baud-rate	MC V2.5	MC V3.0
1000 kbit/s	✓	✓
800 kbit/s	✓	-
500 kbit/s	✓	✓
250 kbit/s	✓	✓
125 kbit/s	✓	✓
50 kbit/s	✓	✓
20 kbit/s	✓	✓
10 kBit/s	✓	✓

Table 1 baud-rates supported by FAULHABER CANopen controllers

Due to the bit-length, the allowable length of a CAN sub-system depends on the baud-rate. At the highest speed the maximum length of the CAN is 30m.

The theoretical topology is a linear system with all the nodes connected in parallel. Practical systems are more complex. There are branches and tree structures. In these more complex systems, it might therefore be more challenging to find an appropriate termination. Generally, the rate of error frames has to be checked during commissioning. Ideally there should not be error frames at all.

	<p>In a simple CAN sub-system all nodes are connected to the same GND. There are more complex situations when different parts of a machine have to be connected or a modular system has to be designed.</p> <p>In these cases CAN repeaters can be used to connect different, electrical separated CAN sub-systems to a single CAN. These repeaters can provide a galvanic decoupling as well. Some of these devices are even offering limited routing functionality.</p> <p>There is no transport layer defined for a CAN system.</p>
---	--

Connectors and wiring

Different devices might offer different CAN connector types, depending on their sizes and housing. The CANopen standard CiA 102 [1] specifies a 9 pin D-Sub connector which is typically used for monitoring tools. The minimum wiring here is:

Pin	Signal	Description
2	CAN _L	CAN _L bus line (dominant low)
3	GND	CAN ground
7	CAN _H	CAN _H bus line (dominant high)

Table 2 Default pin-out of a CAN (D-sub 9)

The physical layer of the CAN uses a differential transmission to increase the robustness of the transmission. The signal lines within a CAN cable therefore have to be in a twisted pair configuration to preserve the symmetry. CAN ground is also necessary to have a common GND potential for the transceivers.

Monitoring

All CAN nodes in a sub-system are electrically connected in parallel. Therefore, it is always possible to add a monitoring node, typically a PC connected to the CAN using a standard CAN-to-USB converter.

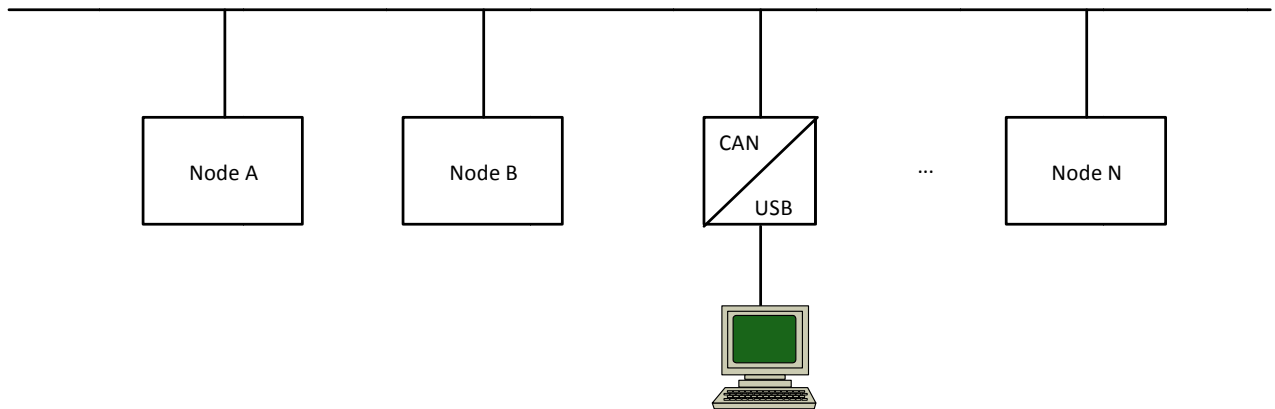



Figure 5 Adding a monitor node to the CAN

There is a variety of CAN monitoring and analyzing tools available. Typically, there is simple monitor software delivered together with the CAN-to-USB converter. Required functions are:

- Logging the traffic & save the log to a text based file
- Sending a CAN message manually
- Filtering the traffic to allow for a more focused analysis of a single controller

	<p>The FAULHABER Motion Manager can be connected to a CAN using a CAN to USB converter too.</p> <p>The supported solutions are:</p> <ul style="list-style-type: none"> • IXXAT • Peak • ESD • EMS
---	---

Even if it is possible to add a PC via a CAN-to-USB converter, some pre-requisites might be necessary. It generally is a good idea to have access to a pre-configured CAN cable having different 9-pin D-Sub connectors in parallel. These cables are handy if you need to connect your PC or any additional node into an existing system.

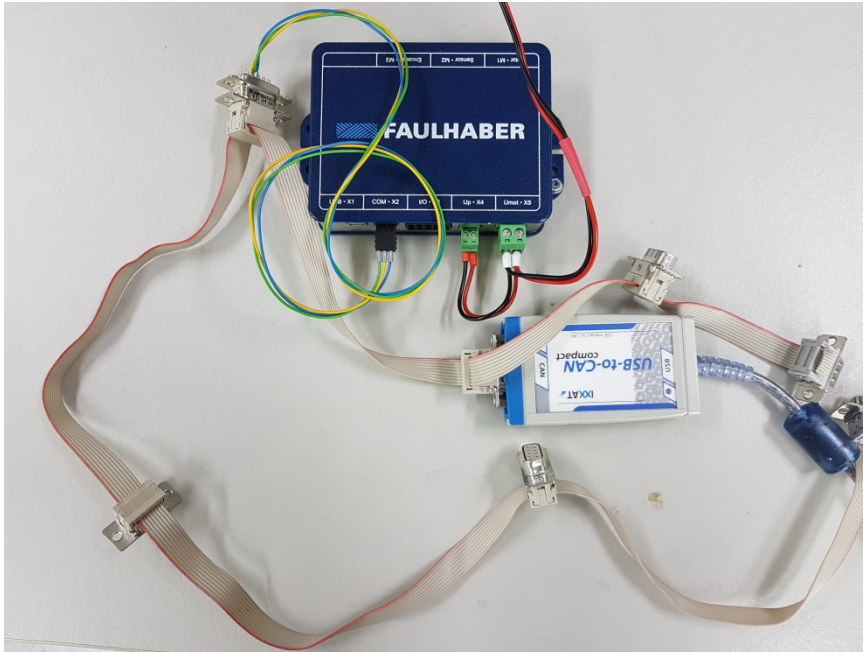


Figure 6 Test equipment for a CAN environment

Communication Patterns

While CAN itself does only describe the physical layer and the coding and handling of the messages itself, higher layer protocols such as CANopen use additional patterns to implement their services.

Client – Server

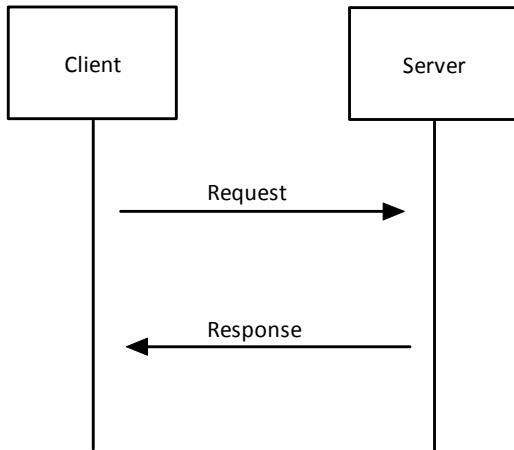


Figure 7

In a client-server pattern there is one server which does offer services (such as controlling a motor) or data (such as an actual position) and there is a client which wants to use the services, read the data or change any settings of the server.

The communication is initiated by the client, which sends a request to the server. The server might respond in different ways – moving something, But at least the request is acknowledged by a response. The response might be on the quality of a simple "ok" or might transmit a requested value.

In the CANopen environment the client server pattern is used to read and write any of the drives parameters – the SDO service (see page 14).

Publisher - Subscriber

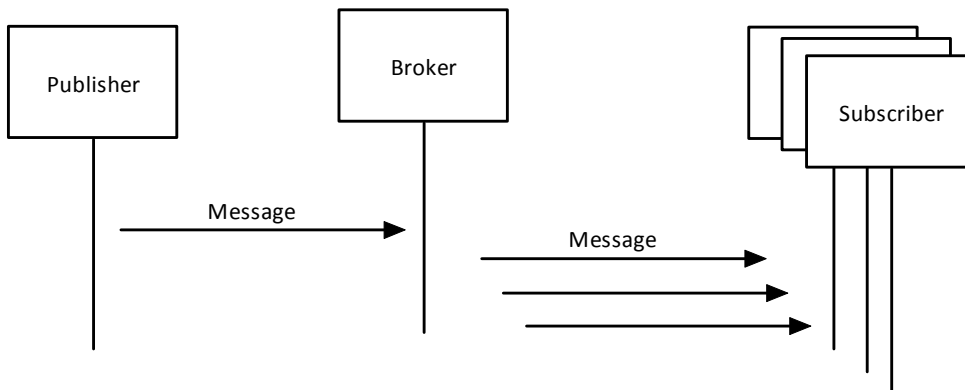


Figure 8

In a publisher –subscriber pattern the publisher provides some data, cyclically or in reaction to a specific event. In an IoT environment the publisher itself would be logged into a broker and send the data to the broker, where any number of subscribers could be logged in and read the data. Additional measures might be necessary to ensure the successful transmission.

Producer - Consumer

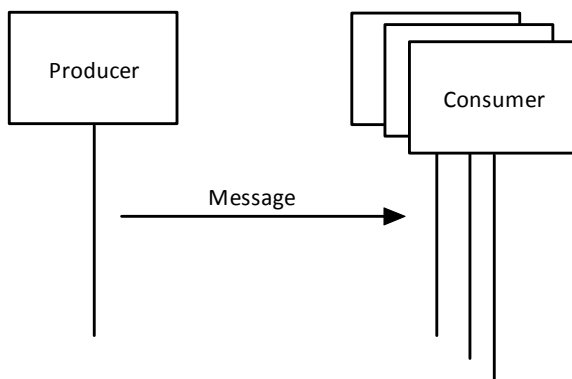


Figure 9

In a CAN system publishing this is much simpler. A producer can send a set of values packed into a message. The quality of transmission is guaranteed by the basic CAN mechanisms. Due to the structure of the CAN, all the connected nodes can decide, whether to use it or to ignore it.

The only prerequisite then is, all the listening nodes – the consumers - have to be aware of the structure of the data coded into the message.

In a CANopen environment the producer – consumer pattern is used during the active operation of the system to exchange the process image using the PDO service (see page 16). The process image is the set of predefined parameters to be used to control the application.

CANopen Overview

Master and Slaves

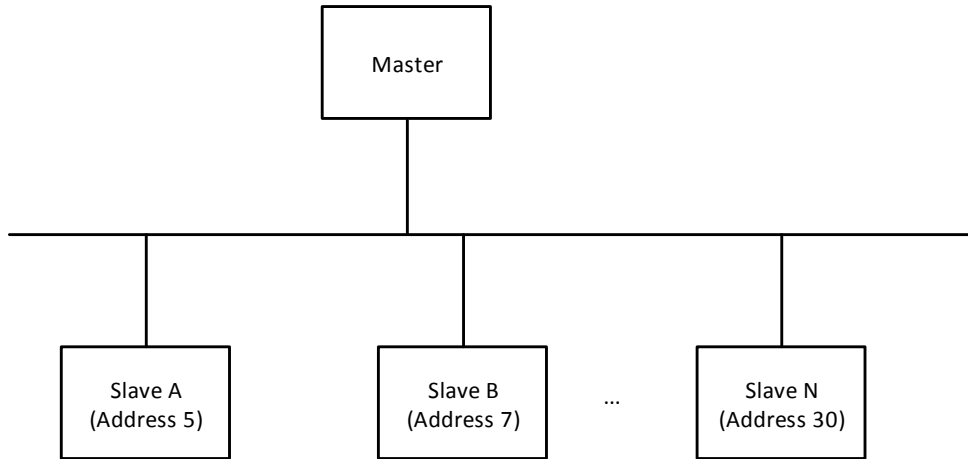


Figure 10

CAN is a peer-to-peer solution – no roles are defined. CANopen is an industrial application of the CAN. Industry comes with hierarchy. Here there are defined roles – one of the nodes is the master.

In a CANopen system there is typically exactly one CANopen master and several CANopen slaves (Figure 10). The slaves are identified by a node address within a range of 1...127. The node address 255 indicates an unconfigured slave which will only start its services, when a valid node address is assigned.

The main task of the **CANopen master** is to monitor the network, configure the slaves if necessary and step through the different stages of the initialization.

After the initialization, the master will typically be the node to generate the cyclic SYNCH signal which starts the repeating communication cycles.

Additionally the master might also incorporate an **application specific master** – which in most cases is a PLC. While the CANopen master is responsible for the communication according to the CANopen protocol the application master will be the one to really use the data and start any behavior of the slaves.

The slaves might be different devices – here motor controllers – but could also be sensor-nodes or general I/Os. Before these slaves start their operation, their communication functions have to be started by the CANopen master. The services of the Network Management (NMT) are used here.

Communication Services

The CANopen communication services and mechanisms are defined in the CiA 301 standard [2].

The common functions and parameters of a servo-drive are defined in the CiA 402 standard [4], while a general purpose I/O module is defined in the CiA 401 standard [3].

Within a CANopen environment the CAN-message-Ids of certain services are referred as COB-Ids.

NMT – Start the communication and supervise the stability

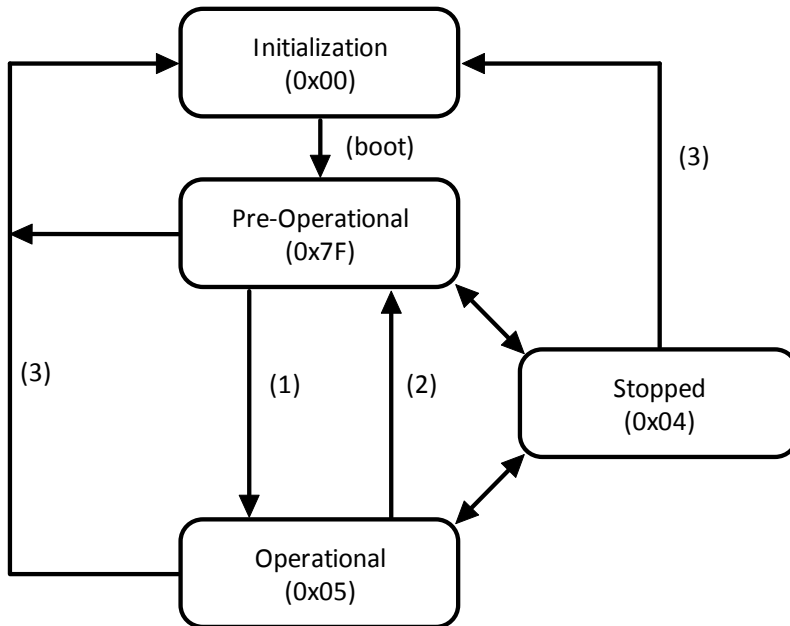


Figure 11 NMT state machine

The NMT services are used to detect newly started slaves, to start the communication of a slave and to supervise their state.

Initialization (0x00)

During the reset or boot of a slave the slave will internally be in the Initialization state. After the basic initialization is finished it will change into the pre-operational state of the communication.

A boot message is sent by the slave during this transition.

Pre-Operational (0x7F)

A slave being in the pre-operational state is ready to be configured. SDO service to read and write parameters is supported. The drive function of a CANopen operated servo drive will however refuse to be enabled unless the communication state is switched to the operational state.

Operational (0x05)

When in operational state the PDO mechanism to exchange the process image – the actual- and target-values is enabled. The drive function of a CANopen operated servo drive can now be enabled.

Stopped (0x04)

In stopped state of the communication no access to the parameters is possible. No SDO service is available in this state.

NMT Identifiers and commands

Basically there are two CAN messages or message ranges used by the NMT.

COB-Id	Description		
0x000	<p>Used by the CANopen master to switch the slaves between their communication states.</p> <p>The NMT message is a CAN message having a length of two data bytes:</p> <table border="1" style="margin-left: 40px;"> <tr> <td style="width: 100px;">command</td> <td>address</td> </tr> </table> <p>If an address in the range of 1 ... 127 is used, a single node is addressed. An address of 0 will be executed by all nodes.</p> <p>The complete set of commands can be found in the FAULHABER CANopen manual. The main commands found in CANopen logs are:</p> <p>0x01: start node (1) 0x80: switch to pre-operational (2) 0x81: reset node (3)</p>	command	address
command	address		
0x700 + Node-Id	<p>Used by the slaves to indicate their communication state</p> <p>This is a message having a length of one data byte where the actual communication state is transmitted. It is used either as a boot message, during node guarding or within the heartbeat protocol.</p> <table border="1" style="margin-left: 40px;"> <tr> <td>NMT state</td> </tr> </table>	NMT state	
NMT state			

Table 3 NMT messages and commands

Guarding or Heartbeat

There are two alternative mechanisms to supervise the communication state of the slaves.

Node-Guarding is triggered by the CANopen master. It's using a client-server pattern.

Parameters at each slave are the guarding time and the life-time-factor.

The master will send a guarding request after the configured guarding time, which the slave has to answer with its actual communication state as a payload.

The details are a little more complex as the request is done using a remote-transfer-request. A request and response share the same identifier. Additionally the MSB of the node state has to be toggled at each response.

In a log this would be:

```

→ 701 RTR
← 701 05           // state is operational
...
→ 701 RTR
← 701 85           //still operational but with a toggled MSB
  
```

→ 701 RTR

← 701 05 //still operational but again with a toggled MSB

...

If there is no response to the request, the master will send a next request after the next guarding period. The life-time factor defines the maximum number of retries before the communication is considered to be lost.

Heartbeat messages are sent by the master or the slave as a simple indication – no request necessary. The used identifier is the same as for the node guarding. So it's either node guarding or heartbeat.

The heartbeat cycle can be configured for both directions. If no heartbeat message is received within the defined period either the master or the slave will consider the communication to be lost and might stop the application (slave side) or try to restart the slave (master side).

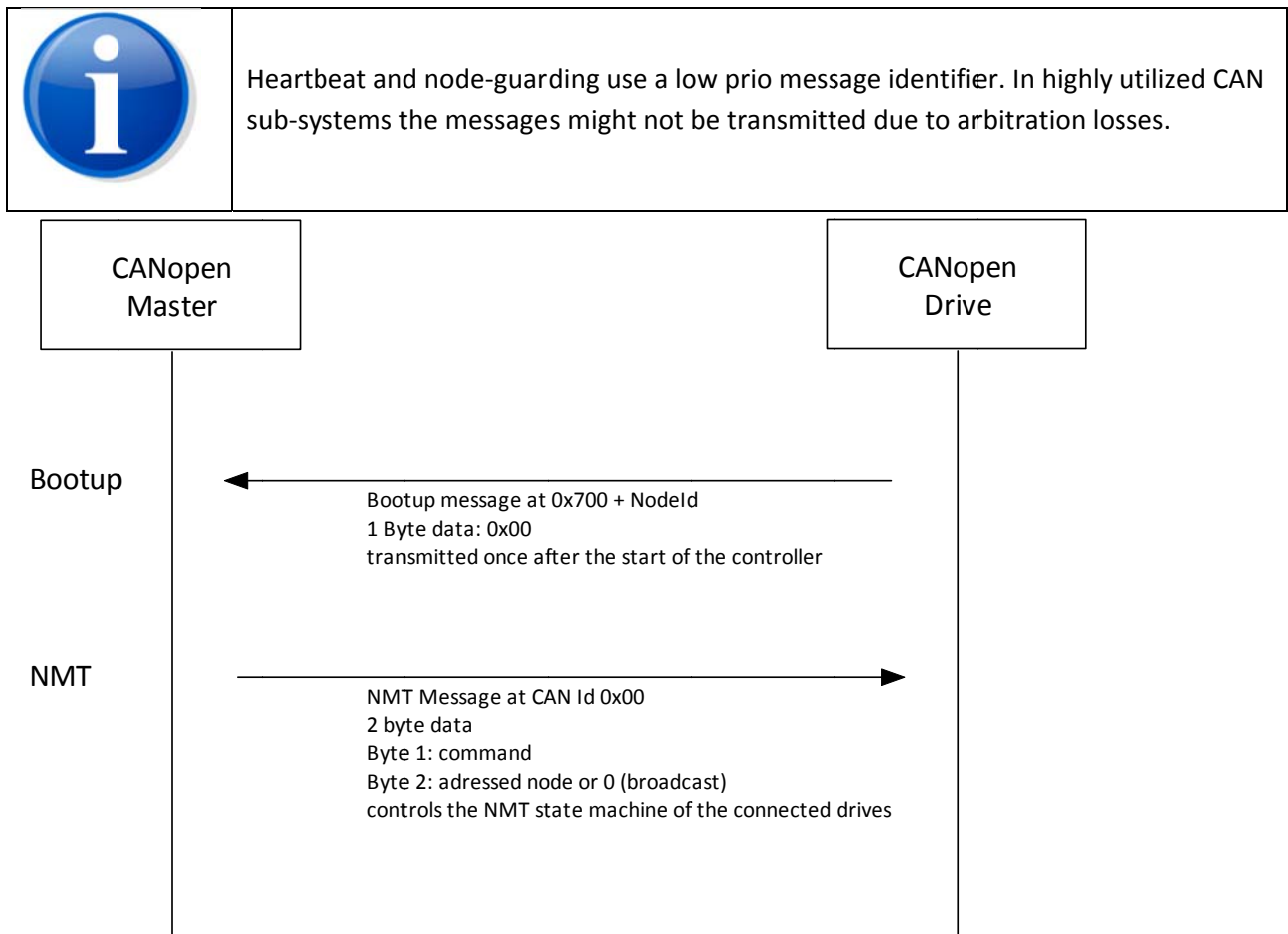


Figure 12

SDO – Read and write parameters of a CANopen slave

In a CANopen servo-drive all the parameters of the system are collected in a so called object dictionary (OD). Whatever sub-function is addressed; they all have their parameters collected there. So the OD is the collection of all parameters, references, control inputs and actual values of the drive.

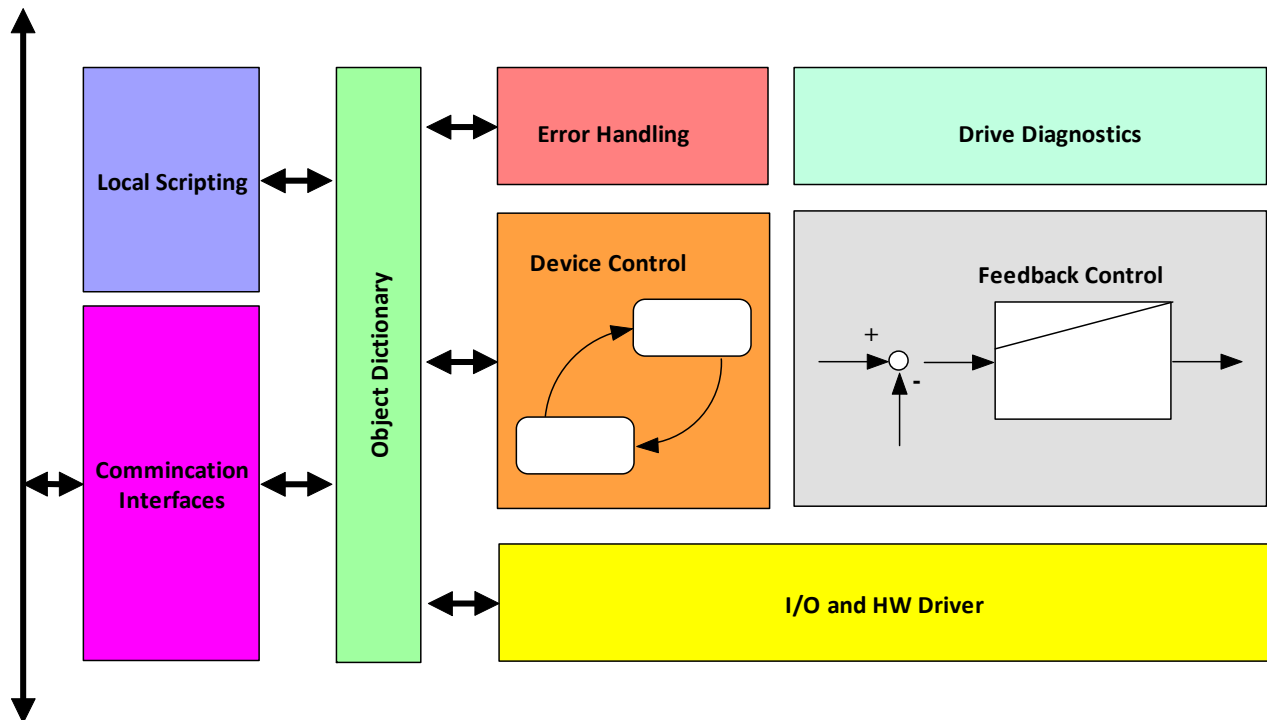


Figure 13 Overview over the major software parts of a FAULHABER MotionController

Any access to the application is routed via one of the parameters collected in the OD. Consequently the parameters are referred as objects. The basic operation here is a read- or write-access to the parameters collected here – read object and write object. So basically to interact with such a drive will result in a sequence of read and write commands such as:

- Set Target Velocity to xxxx
- Read Actual Velocity
- ...

All parameters/objects are identified by a 16 bit index – the parameter number and a 8 bit sub-index, allowing for structured parameters.

Simple Parameter			Structured Parameter		
Idx	Sub	Parameter	Idx	Sub	Parameter
0x607A	00	Target Position	0x2311	00	Digital I/O entries
				01	Logical Input State
				02	Physical Input State
				03	Output State
0x60FF	00	Target Velocity			

Table 4¹

The mechanism used for this simple sequential access is the **Service-Data-Object (SDO)**. SDO is a client server type of communication where the master – PC or PLC – is sending a request to read or write a single parameter to the client. The drive is the server. It will execute this read or write and will answer with a response.

There is one dedicated COB-Id per node for the request (0x600 + NodeId) and one for the response of the drive (0x580 + NodeId).

So the benefit of the SDO transfer is, after having received the response the automation master will have a confirmation of the executed parameter access. The drawback is, it will take some time. So if for example a target value will have to be updated, it's a little more complex than writing the updated value to a direct output of the PLC.

The SDO mechanism is typically used during the preoperational state of the NMT to configure a drive or adjust the drive's communication settings.

¹ The CiA defined ranges for the parameters of a CANopen node. The range 0x1xxx is reserved for communication related parameters out of the CiA 301 standard. The parameters starting from 0x6000 are reserved for the device profile. CiA 402 defines the servo-drive profile and its parameters. The parameter range in between (0x2000 ... 0x5FFF) is reserved for manufacturer specific parameters like I/O configuration or control parameters.

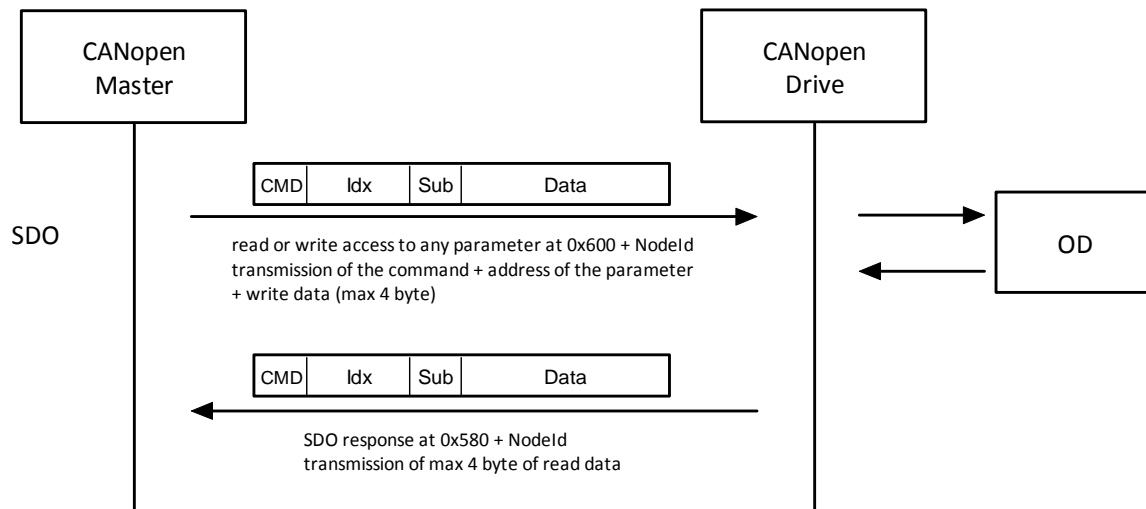


Figure 14

Industrial CANopen PLCs or gateways used as the master of a local CANopen sub-system will typically offer to write the communication settings of all services (PDO, SYNCH, Node-Guarding) before starting the node into the operational state.

PDO - exchange the process image

The **Process Data Object (PDO)** service is used to exchange the process image after the initial configuration is done and the slaves have been switched into the operational state of the communication. The PDO service is available in operational state only.

Where the SDO can address a random but single parameter with each access, a message used as a PDO can combine different values into a single message – provided the data-length will fit into the 8 byte limit of a single CAN message.

So it is possible to combine the parameter controlword (16-bit), used to control the main behavior of a CANopen servo-drive the selected mode of operation (8-bit) and any of the target values like the target-position (32-bit) or the target-speed (32-bit) in a single message.

PDOs are distinguished between the one to be received by a node (RxPDOs) and the ones to be sent by a node (TxPDOs).

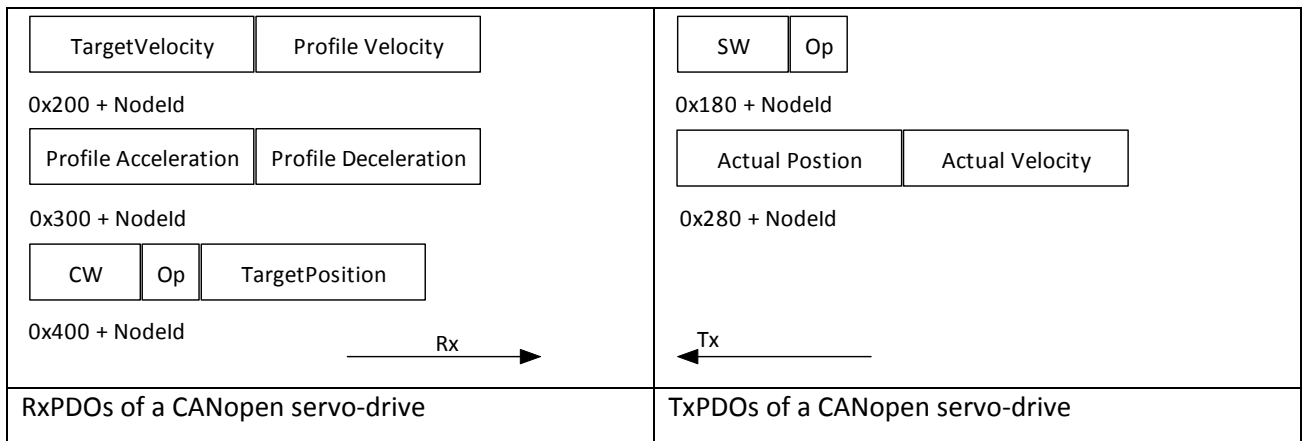


Figure 15


PDOs are implemented using the Producer-Consumer pattern. So each node sends its PDOs, the others can be consumers and receive and process the data. There is no explicit handshake implemented by the communication service itself. However, being in a CAN environment, we can at least assume the message to be received as soon as we were able to send it. A PDO is therefore very close to the original intention of raw CAN. A challenge of course is: as there is no information within the CAN message itself, which parameters are combined, the producer and the consumers need to have a sort of agreement, which combination is to be expected at which message identifier. Every PDO has its own message Id by which it will be identified by the consumers.


The agreement about the payload is called PDO mapping. This is a set of parameters stored at each CAN-node which describes the mapping of parameters into the supported PDOs per direction.


In the example above the mapping would be:

<p>RxPDO 1 (0x200 + NodeId)</p> <ul style="list-style-type: none"> • 0x60FF.00 – 32 Bit • 0x6081.00 – 32 Bit 	<p>TxPDO 1 (0x180 + NodeId)</p> <ul style="list-style-type: none"> • 0x6041.00 – 16 bit • 0x6061.00 – 8 bit
<p>RxPDO2 (0x300 + NodeId)</p> <ul style="list-style-type: none"> • 0x6083.00 – 32 bit • 0x6084.00 – 32 bit 	<p>TxPDO2 (0x280 + NodeId)</p> <ul style="list-style-type: none"> • 0x6064.00 – 32 bit • 0x606C.00 – 32 bit
<p>RxPDO3 (0x400 + NodeId)</p> <ul style="list-style-type: none"> • 0x6040.00 – 16 bit • 0x6060.00 – 8 bit • 0x607A.00 – 32 bit 	<p>TxPDO3 (0x380 + NodeId)</p> <ul style="list-style-type: none"> • Not used
<p>RxPDO4 (0x500 + NodeId)</p> <ul style="list-style-type: none"> • Not used 	<p>TxPDO4 (0x480 + NodeId)</p> <ul style="list-style-type: none"> • Not used

Figure 16 Example mapping of a CANopen servo-drive operated in Profile Position Mode and alternatively in Profile Velocity Mode

	<p>The length of a PDO depends on the number of bytes used in the data section. So the actual length of a PDO can be in between 1 and 8 bytes.</p> <p>Receivers will usually check for the correct length of a PDO.</p>
---	---

	<p>The mapping information is a part of the communication related parameters stored in the 0x1xxx parameter range.</p>
---	--

	<p>The mapping information of a PDO must not be changed during the operational state of the communication. Otherwise the producer and the consumers will at least temporarily no longer have a common understanding of the coded information.</p> <p>If a PDO has to be remapped it has to be set into an invalid state first which prevents further updates. Only after the mapping information of the producer and all consumers is update it may be activated again.</p> <p>Typically PDO mapping is static a soon as the pre-operational state is left.</p>
---	---

As every node has its own set of PDOs and COB-Ids the typical consumer of a TxPDO in such a CAN sub-system is the automation master. The peer nodes will not process the PDOs of the other nodes, as their

RxPDOs are configured for a different COB-Id. The same applies for TxPDOs. They will be sent by the automation master via the CANopen master (Figure 17). So it is usually a virtual 1:1 connection to the master which then connects the different parts of information together to create a synchronized behavior.

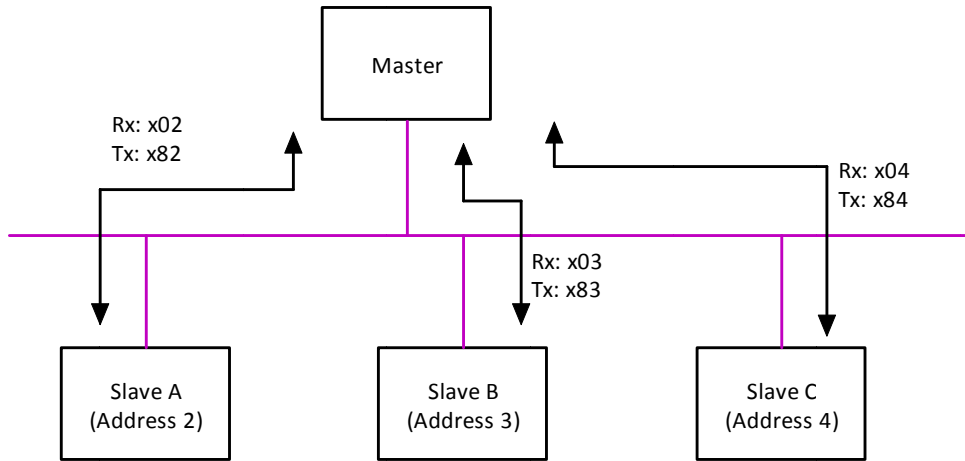
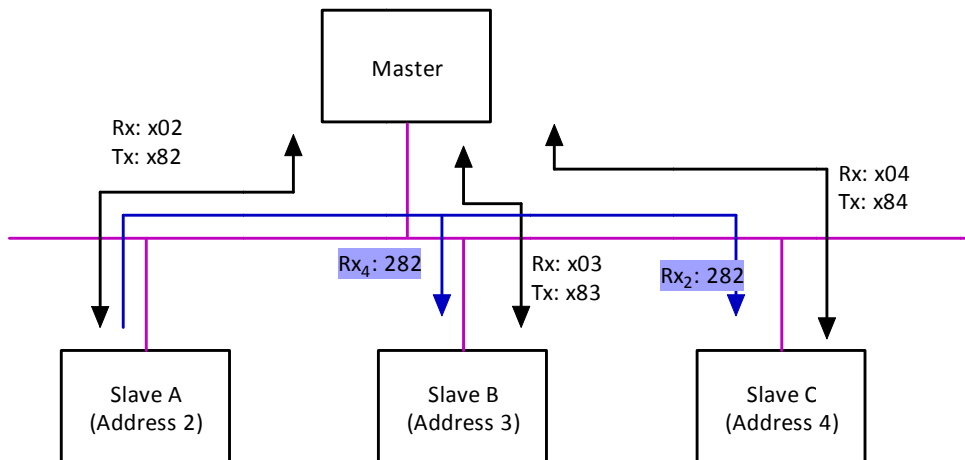



Figure 17 Default PDO COB-Id within a CANopen subsystem.

If a direct peer-to-peer communication is to be used, the default COB-Ids of some of the PDOs have to be modified, to be able to directly process a message sent by a peer-node ().



Therefore, even if a peer-to-peer exchange would be possible it typically is not used and requires manual configuration.

	<p>If a peer-to-peer exchange shall be used, the identifiers used for the PDOs have to be adjusted. So if the nodes 3 and 4 shall receive the 2nd TxPDO of node 2 the configuration could be :</p> <p>Node 2: default identifiers – here TxPDO2 @ 0x282</p> <p>Node 3: one of the RxPDOs has to listen to Node 2: e.g. RxPDO4. Therefore, the identifier assigned to RxPDO4 would no longer be 0x503 but 0x282.</p> <p>Node 4: one of the RxPDOs has to listen to Node 2: e.g. RxPDO2. Therefore, the identifier assigned to RxPDO2 would no longer be 0x204 but 0x282.</p>
---	--

SYNCH – trigger the exchange

The typical data exchange within a CANopen sub-system is cyclic one. The CANopen master will create a zero length synch message (default Id: 0x80) which will trigger the exchange cycles.


The SYNCH interval has to be configured at the CANopen master. Typical values are 10ms ... 200ms depending on the number of nodes.

PDOs can be configured to be transferred either cyclically or in case of changed contents. The setting is called the transmission type having values of 0, 1 ... 240, 253 or 255.

Type	Description
0	PDO will be transmitted after a SYNCH, but only if changed
1 ... 240	PDO will be transmitted every 1 ... 240 cycle
253	The PDO will only be sent, if a related remote transfer request has been received
255	PDO will be transmitted only on a change – depending on the device profile.

Table 5 PDO transmission types

The transmission type of each PDO is a next parameter that needs to be configured for each PDO.

	<p>Transmission type 255: asynch – device profile dependent – will trigger a transmission of the PDO only if the status word (0x6041.00) is mapped to the PDO and has changed.</p> <p>Masters on the other hand will typically send their asynch PDOs at every change of the contents.</p>
--	--

Event-Timer – an alternative method of cyclic exchange

The event-timer of a PDO can be used to cyclically send a PDO which is configured to a transmission type 255 – asynchronous transfer. Using the event-timer the update-rate of the PDOs can be configured individually for each TxPDO.

A PDO having a transmission-type of 255 and a event-timer > 0 will be updated at changes of the status-word and then cyclically at every event-timer cycle period.

EMCY – in case of a problem

The EMCY Object is a simple method to indicate errors occurred within the device. It is a single message at a high prio COB-Id (0x80 + NodeId). So every node has its own EMCY message.

The length of an EMCY is 8 bytes. The contents is mixed between standard error words, defined in the CANopen standards and a FAULHABER specific error word.

For a FAULHABER MotionController when or whether to send an EMCY message can be configured within the CANopen devices. By default the MotionContoller will trigger an EMCY at every detected error. If the

error is transient – like an over temperature - a first EMCY will signal the error and a second empty EMCY will be sent, if the error is gone again².



Within a PLC environment there is usually no direct access to the contents of EMCY messages. The PLC program usually has no direct awareness of the CANopen sub-system and its specific services. So if the device state has to be monitored out of a PLC program, mapping the internal error register to one of the PDOs might be easier.

Pre-defined connection set

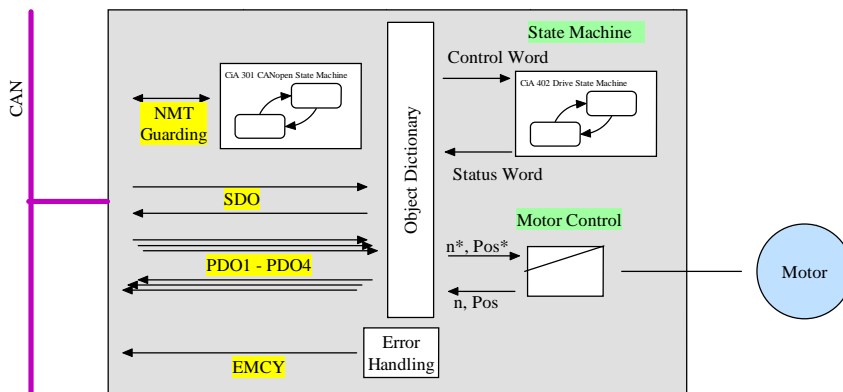


Figure 18 A CANopen drive. All the services on the left of the OD, the drive behavior on the right.

Some of the services above do have a fixed COB-Id assigned to it, some are located in an Id range, combining a basic Id and the NodeId. The Ids are assigned in a way which allows for hardware (HW) filtering of the messages to be received by a CAN controller.

This assignment of COB-Ids is called predefined connection set.

² Whether an error shall trigger a EMCY or not can be configured. By default, all errors are activated for EMCY transfer.

Service	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
NMT (0)	0	0	0	0				0			
SYNCH (0x80)	0	0	0	1				0			
EMCY (0x80 + Id)	0	0	0	1				NodeId			
TxPDO1 (0x180 + Id)	0	0	1	1				NodeId			
RxPDO1 (0x200 + Id)	0	1	0	0				NodeId			
TxPDO2 (0x280 + Id)	0	1	0	1				NodeId			
RxPDO2 (0x300 + Id)	0	1	1	0				NodeId			
TxPDO3 (0x380 + Id)	0	1	0	1				NodeId			
RxPDO3 (0x400 + Id)	0	1	1	0				NodeId			
TxPDO4 (0x480 + Id)	0	1	0	1				NodeId			
RxPDO4 (0x500 + Id)	0	1	1	0				NodeId			
SDO Request (0x600 + Id)	1	1	0	0				NodeId			
SDO Response (0x580 + Id)	1	0	1	1				NodeId			
Guarding (0x700 + Id)	1	1	1	0				NodeId			
LSS Request	1	1	1					0xE5			
LSS Response	1	1	1					0xE4			

Table 6 Ranges for the COB-Ids according to the predefined connection set

Handling of COB-Ids when NodeId is changed

Some of the message-Ids used for the CANopen services can be configured freely – others are fixed. The COB-Id of the configurable Ids is stored in the 0x1xxx parameter range of the OD. E.g. 0x183 for the TxPDO1 of node 3.

When the NodeId of a CANopen slave is changed, the handling of the adjustable COB-Ids could either be:

- Adjust all the COB-Ids to fit to the new Node-Id
- Don't adjust the COB-Ids

This could either be done by a master (MotionManager) or by the slave itself.

FAULHABER MotionControllers will adjust the COB-Id by firmware, when the NodeId is changed from 255 – unconfigured to any valid NodeId 1 ... 127.

The FAULHABER MotionManager will offer to change the COB-Ids of the PDOs, when the NodeId of a controller is changed within the range of valid Ids.


LSS – configure the NodeId and baud-rate

Layer Setting Services (LSS) can be used to initially configure the NodeId and the baud-rate of either a previously unconfigured node or to change these settings. FAULHABER MotionControllers do not use any HW-switches to adjust these settings.

LSS can be used within a sub-system. Even if there are several unconfigured nodes the one to be configured can be identified by its identification settings and ultimately its serial number. This allows to add a completely unconfigured device into an existing sub-system. The configuration could then be automated by the master to create a copy of the previous one.

It might be easier to configure the NodeId and baud-rate in a 1:1 configuration though without having to deal with the serial number of the device.

To use the LSS please refer to [5] for implementation details or use one of the PC-based tools to change the settings.

	The FAULHABER Motion Manager can be used to set the NodeId and baud-rate of a controller to the required values.
---	--

EDS-File

The capabilities of a CANopen device are summarized in an Electronic DataSheet (.eds) file which usually is distributed together with the device. The latest .eds file of FAULHABER MotionController is distributed together with the FAULHABER Motion Manager. The .eds file contains a complete list of all supported parameters as well as their properties and – if applicable – their default values.

```
[2311sub1]
ParameterName=Digital input logical state
ObjectType=7
DataType=0x0005
AccessType=RO
PDOMapping=1
```

Table 7 Example of an entry in a .eds file - here object 0x2311.01

The .eds file can be imported into CANopen masters, which will use the information to assist the configuration. Without an imported .eds file the master will not be able to display the parameter names or check the value ranges.

CANopen communication cycles

After the configuration of the nodes the CANopen master will switch the nodes to their operational state using the NMT. The SYNCH service will be started and driven by the SYNCH the cyclic exchange of the process image will start.

The type of communication has to match the requirements of the application. In a CAN sub-system of servo-drives the drives are usually operated using the **Profile Position Mode (PP)** which will move from A to B autonomously after having received the start command. In such an environment the master will send a new target position only if required and will start the movement with a single access to the drives controlword. As such a move might take several 10ms to multiple 100ms the commands from the master can be sent on demand – asynchronously.

If **Cyclic Synchronous Position mode (CSP)** is used, the target position should be updated in short cycles, so most likely the target will be sent synchronously. While CSP is active there is no interaction with the controlword – so a PDO having the controlword only, might stay in asynchronous configuration.

The actual values of the drives however are often used to monitor the drive behavior and are typically updated cyclically every communication cycle.

So the master uses PDOs configured for asynchronous transmission while the drives will send their PDOs cyclically. The synch interval length depends on the number of nodes connected to the sub-system, the baud-rate and the number of messages that have to be exchanged.

Typical values are 10ms (a few nodes only) to 50ms or more.

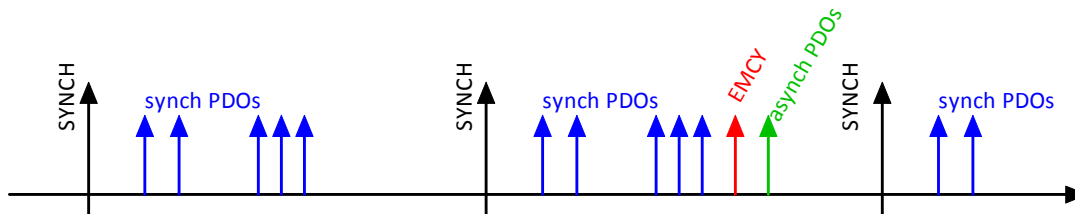


Figure 19 Communication cycles of a CANopen sub-system

In addition to the process data, there might be occasional requests for a SDO service when a specific parameter of one of the slaves has to be changed.

The guarding- or heartbeat-interval is usually a multiple of the synch interval. This is again depending on the needs of the application. A failed guarding might for example disable a drive, as the connection to the master is lost. So the selection of the guarding-interval might be related to the physical risk of the movement.

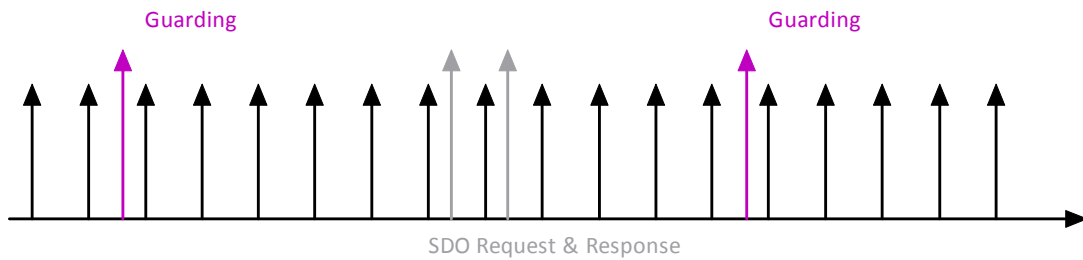


Figure 20 Mix of PDO-cycles, SDO and node-guarding

Startup of a CANopen sub-system

If a CAN node or a sub-system does not behave as expected a check of the communication might be necessary. Different phases can be identified. During the following description, we assume all nodes are configured and have a unique NodeId and their baud-rate is configured to be identical to the one of the master.

Boot-Msg

After a node is powered up it will send its boot-msg. This will happen independently from the state of the complete sub-system.

Example: node 3 is sending its boot-msg:

Id	Data	Description
...		
703	00	The 7xx indicates the message as one of out of the guarding messages. The 0 denotes the boot-msg.

Here we do see a message within the range of the guarding and boot messages. These messages do have a single data byte, which contains the state of the communication stack. The boot-message is sent when entering the pre-operational state. The state (init) is coded by the 0.

Configuration via SDO

After a new drive is identified by its boot-message, the CANopen master will configure the slave. This is an optional step. Typical parameters are at least the communication settings like guarding interval and the PDO settings (mappings & transmission types).

Additionally applications specific configurations can be added for many industrial masters. Some implementations will even keep a copy of the complete drive configuration within the master. In such a case even an exchange of a component will be supported.

Example: node 3 is configured via SDO after having sent its boot-message:

Id	Data	Description
...		
703	00	The boot-message
603	xx 00 14 00 dd dd dd dd	xx contains the SDO request command 00 14 00 is the OD entry 0x1400.00 dd contains the data to be written
583	yy 00 14 00 rr rr rr rr	yy contains the SDO response code

		00 14 00 is the OD entry 0x1400.00 rr contains the response data
603	xx ii ii ss dd dd dd dd	will be repeated several times until the complete configuration is written. ii ii is the index of the accessed object, ss the sub-index
583	yy ii ii ss rr rr rr rr	

Start via NMT

After the node is configured – that is after the CANopen master successfully wrote the entire stored configuration via SDO – it will start the cyclic data exchange. This could be done for the complete network, usually it is done on a per node base. For the node 3 the message would be

Id	Data	Description
...		
603	xx ii ii ss dd dd dd dd	will be repeated several times until the complete configuration is written. ii ii is the index of the accessed object, ss the sub-index
583	yy ii ii ss rr rr rr rr	
0	01 03	NMT command to start node 3

Cyclic data exchange

After at least a first node is switched into the operational state the communication cycles as in Figure 19 are started by the master by sending a first SYNCH message. When receiving the first SYNCH, all the nodes will send their PDOs according to their configuration. At this start of the communication all the asynch PDOs are sent too, to create an initial valid process image.


In a sub-system having the nodes 2, 3 and 4 with a configuration according to Figure 16 we might see something like:


Id	Data	Description
...		
0	01 02	NMT command to start node 2
0	01 03	NMT command to start node 3
0	01 04	NMT command to start node 24
80		SYNCH
182	40 00 01	
183	40 00 01	Disabled drives OpMode = PP
184	40 00 01	
202	00 00 00 00 B8 0B 00 00	TargetSpeed = 0 ProfileSpeed = 3000
203	00 00 00 00 B8 0B 00 00	TargetSpeed = 0 ProfileSpeed = 3000
204	00 00 00 00 B8 0B 00 00	TargetSpeed = 0 ProfileSpeed = 3000
282	34 12 00 00 00 00 00 00	ActPos = 0x1234 ActSpeed = 0
283	78 56 00 00 FF FF FF FF	ActPos = 0x5678 ActSpeed = -1
284	FC FF FF FF 00 00 00 00	ActPos = -4 ActSpeed = 0
302	4D 1C 00 00 C4 09 00 00	Acceleration = 7500 Deceleration = 2500
303	4D 1C 00 00 C4 09 00 00	Acceleration = 7500 Deceleration = 2500
304	4D 1C 00 00 C4 09 00 00	Acceleration = 7500 Deceleration = 2500

402	00 00 01 00 00 00 00			TargetPos = 0
403	00 00 01 00 00 00 00	Shutdown command	OpMode = PP	TargetPos = 0
404	00 00 01 00 00 00 00			TargetPos = 0
80	SYNCH			
182	40 00 01			
183	40 00 01	Disabled drives	OpMode = PP	
184	40 00 01			
282	30 12 00 00 FE FF FF FF	ActPos = 0x1230		ActSpeed = -2
283	78 56 00 00 FF FF FF FF	ActPos = 0x5678		ActSpeed = 0
284	02 00 00 00 02 00 00 00	ActPos = 0x0002		ActSpeed = 2
83	EC EC ER FF FF 00 00 00	EMCY message having the 16-bit error code (EC), the 8 bit error register (ER) and the FAULHABER error register (FF).		
80	SYNCH			
182	40 00 01			
183	40 00 01	Disabled drives	OpMode = PP	
184	40 00 01			
602	xx ii ii ss dd dd dd dd	SDO request for node 2		
282	34 12 00 00 00 00 00 00	ActPos = 0x1234		ActSpeed = 0
582	yy ii ii ss rr rr rr rr	SDO response for node 2		
283	78 56 00 00 FF FF FF FF	ActPos = 0x5678		ActSpeed = -1
284	FC FF FF FF 00 00 00 00	ActPos = -4		ActSpeed = 0
...				

Therefore, besides the SYNCH and the PDOs we will see an occasional EMCY depending on the drive state and there might be additional SDO services – started by the application master via the CANopen master.

A first check might be whether all the expected PDOs have been transmitted and whether the logging tool reports any error frames.

	<p>If not all the expected PDOs are found, check their configured transmission type. The CANopen master might report missing PDOs too.</p> <p>Alternatively check whether the communication cycle is long enough for all PDOs to be transferred.</p>
---	--

	<p>It's very likely to see some error frames, when the system is started up. Later on the rate of error messages should be very small compared to the undisturbed messages. In a limited network there should not be any error frames if the wiring and termination is correct.</p> <p>If there are different components connected to a sub-system a certain rate of error frames is sometimes observed. In this case the timing of the different components might differ slightly. This can be tolerated, if the rate of error frames is still very low and no avalanches of errors occur.</p>
---	---

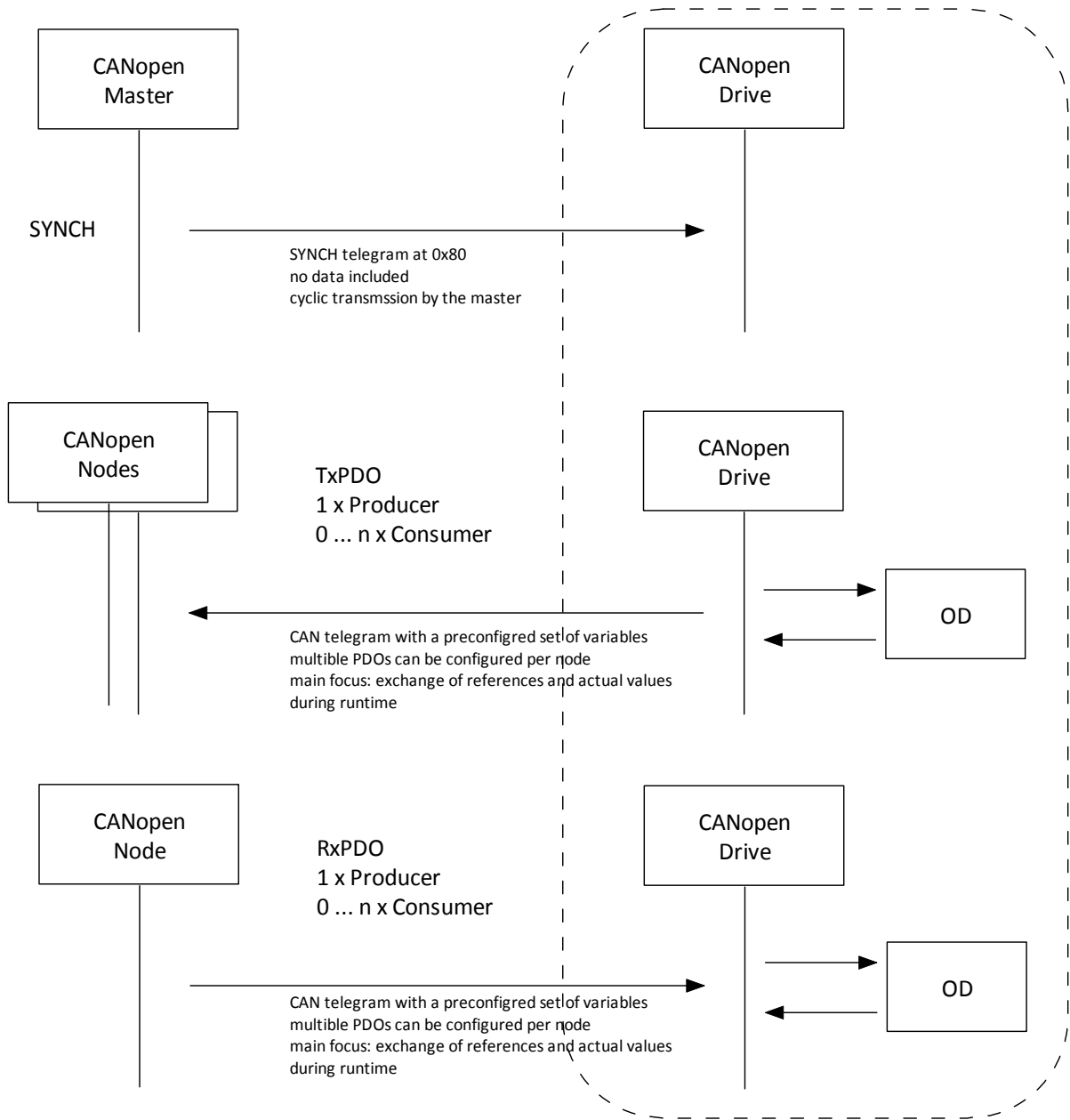


Figure 21 Services in operational state of the communication

Robust configuration

After having a basic understanding of the services of a CANopen sub-system there are some hints on which services to use and how to use them.

The necessary areas of configuration are:

- The common settings of the communication.
These are the baud-rate, the SYNCH interval and typically the type of node-guarding to be used.
- The individual configuration per node:
These are the PDO settings like mapping and transmission type and the reaction of a node to a communication error.

Common settings

Baud-rate

The baud-rate typically is either 125kBit/s, 250kBit/s, 500 kBit/s or 1 MBit/s. There is no harm in using the fast rates. The total length of the sub-system is limited to around 30m in case of the 1 Mbit/s. But that's no concern in many systems. If 30m is not sufficient use 500 kBit/s. 250 kBit/s is for really large systems only.

A higher baud-rate will allow for shorter communication cycles. Therefore, if you need to have a synchronized behavior of several axes, 1 Mbit/s again might be the choice.

The baud-rate setting of the nodes should not be left at the initial auto-baud detection. In an application it's very unlikely to change the baud-rate on the fly. A node being configured for auto-baud detection will not be able to send its boot-message until the real baud-rate is detected. A decision of either having the correct rate or not can only be done when a message is received. So not only do nodes configured for auto-baud detection require others to be talking to be able to synchronize into the speed, they can as well be misled in their detection by occasional error frames.



Using a fixed baud-rate in all nodes results in a more robust system start behavior compared to auto-baud detection.

Communication Cycle

The communication cycle is defined by the interval between two SYNCH messages. This synch interval has to be long enough for all nodes to send their PDOs within the cycle. At 1 Mbit/s our experience is to have a cycle with at least one ms per node and PDO. So if we do have 10 nodes connected and 1 RxPDO + 1 TxPDO per node the minimum synch cycle would be 10ms. Numbers however depend on the capabilities of the nodes to handle the traffic.

Node-Guarding vs. Heartbeat

CANopen originally defined the node-guarding, where the master requests an updated state information from a slave by sending a remote transfer request of exactly this message.

Remote transfer is very specific for a CAN environment and therefore the heartbeat had been added and is the preferred solution in the CANopen literature.


All participants send heartbeat messages cyclically. Using the same guarding interval the heartbeat will reduce the traffic slightly, as no RTR frames are required.

Drawback of the heartbeat however is its lack of robustness.

Consumer and a producer heartbeat-interval are usually configured parameters of the slaves. There is one common producer cycle for the master. Usually the master then sets these cycles to identical values during the preoperational configuration. Compared to the node-guarding there is no tolerance.

If we compare a typical configuration of a node-guarding, having a guarding cycle of 100ms and a life-time-factor of 3 the master will request a guarding message every 100ms. If there is no response e.g. because the node is temporarily overloaded, the master will consider the slave to be offline only after the third consecutive missing response.

Heartbeat on the other hand does not offer such a tolerance. If any of the slaves tries to send its heartbeat during a situation where a lot of higher prio message such as PDOs are exchanged, it might at least be delayed. We did observe masters handling this delay as a missing node, restarting the drive via the CAN. As the cycle times are usually configured by the CANopen master – identical for master and slave – there is no explicit tolerance against the traffic jam here.

	<p>Even if heartbeat is the meanwhile recommended version of life-state checking, node-guarding seems to be the more robust solution and should be preferred.</p>
---	---

Node-individual settings

PDO Mapping

The PDO mapping is individual for each application. Some restrictions should be noted though.

The CiA 402 Servo Drive Profile includes standard PDO mapping for the RxPDOs 1 – 3 and the TxPDOs 1-3.

<p>RxPDO1</p> <ul style="list-style-type: none"> • 0x6040.00 Controlword 	<p>TxPDO1</p> <ul style="list-style-type: none"> • 0x6041.00 Statusword
<p>RxPDO2</p> <ul style="list-style-type: none"> • 0x6040.00 Controlword • 0x607a.00 TargetPosition 	<p>TxPDO1</p> <ul style="list-style-type: none"> • 0x6041.00 Statusword • 0x6064.00 PositionActualValue
<p>RxPDO2</p> <ul style="list-style-type: none"> • 0x6040.00 Controlword • 0x60FF.00 TargetVelocity 	<p>TxPDO1</p> <ul style="list-style-type: none"> • 0x6041.00 Statusword • 0x606C.00 VelocityActualValue

Table 8 Default mappings of a CANopen servo-drive

Each of the RxPDOs contains the controlword, each of the TxPDOs contains the statusword. That's done to ease the commissioning. Even without any changes in PDO mapping a master will be able to operate a drive in either Profile Position Mode (PP) or Profile Velocity Mode (PV). But that's true only, if only the related PDO is sent by the master.


There is trap here. In PP mode as well as in Homing mode the start of a movement of the drive is triggered by a rising edge in bit 4 of the controlword – it's a start bit.

In PP mode the drive will start to move using the last target position received before the rising edge of the start bit. Unfortunately, that is difficult to control out of a PLC environment. In such an environment the variables mapped into the PDOs are directly mapped to global variables of the PLC runtime environment. So in a PLC application the code could like

```
TargetPos := Pos A;
ControlWord.Bit4 := true;
```

Unfortunately there are no guarantees about when and in which sequence the PDOs are sent. Usually the PLC task time will be shorter than the CAN communication cycle. So if all three PDOs are mapped to the PLC application they will all be updated after this code sequence – at least the controlword should have changed here. But – depending on the timing, it might even be RxPDO3 that is the first to be sent after the change of the variable ControlWord. In such a case the drive will start its movement, but will still use the last target position – which very likely is the actual one. Therefore, no movement will occur at all.

To avoid this, the controlword and the target position should be mapped to the same RxPDO and the controlword must not be mapped to any other active PDO.

	<p>If Profile Position Mode (PP) is used the controlword and the target position have to be mapped to the same RxPDO. The controlword must not be mapped to a second active PDO!</p>
---	--

The drive will flag the new position with a set bit 12 – setpoint acknowledge. Only after this response is received it's safe to clear the start-bit again.

```
If (StatusWord.Bit12) then
    ControlWord.Bit4:= 0;
End IF
```

As soon as the drive leaves the target window it will also reset the target reached bit (bit 10) in the statusword. This bit will be set again, if the actual position reached the target window again and remained within the window for the position window time.

If the OpMode has to be changed during the operation of an application, the same rule applies for the Modes of Operation (0x6060.00). During a switch from whatever OpMode to PP the drive will reset the target position to the actual value and will hold the actual position until a new move is started.

Now if the idea is to:


```
OpMode := 1;
```

```
TargetPos := Pos A;
ControlWord.Bit4 := true;
```

The same problem arises. Of course, the Modes of Operation has to be added to one of the RxPDOs³, but it also has to be assured, it's received together with the controlword and the target position⁴.

This can be done by mapping controlword, modes of operation and target position into the same RxPDO.


It is no problem to locate a target velocity or a target torque in any of the other RxPDOs as there is no handshake here. So the resulting mapping might look similar to what is given in Figure 16.

	<p>If the TxPDOs are transferred synchronously it isn't necessary to have the statusword as a part of each TxPDO. A single occurrence is sufficient.</p>
---	--

PDO transmission types

The default transmission types of a FAULHABER servo drive is 255 – asynchronously. That is, the master is expected to send it's update whenever a value changed, the drive will send its PDOs only if the statusword is mapped into it and the contents of the statusword changed. Changes of other values like position or velocity do not trigger a re-transmission as this would flood the CAN. The CiA 401 I/O node profile defines a corridor around analog values to handle the problem of jitter at actual values. That's not the case for the CiA 402 Servo Drive Profile.

So if a master wants to trace the movement of a slave it is best practice to change the transmission type to 1 – cyclic every SYNCH cycle.

	<p>In many applications the transmission type of the RxPDOs is 255 – asynchronously. The transmission of actual values back to the master is done synchronously - transmission type 1.</p>
---	--

There might be one exception to the rule. That is the case where a relatively long communication cycle is used (multiples of 10ms). In such a case whenever a new position command is received, the drive will of course flag the reaction more or less immediately. But as the cycle is a long one, the PLC will not receive the answer immediately. It might even occur, that the new position is reached, before the response has

³ Of course the Modes of Operation could alternatively be changed using the SDO service. This might be an option to reduce the traffic, if the change is necessary only after the initial homing.

If frequent changes between PV and PP are used during the operational state of the machine, it might be better to have the Modes of Operation mapped to a PDO.

⁴ It is possible to safely change from one OpMode to another one by reading the Modes of Operation Display (0x6061.00) back. If the new position is sent only after the Modes of Operation signals the PP being active, the new target position will no longer be canceled by an initial internal target position.

been sent. In such a case the response might contain the setpoint acknowledge flag as well as the target reached flag. Depending of the implementation of a step-sequence of commands at the master, this step sequence might be misled due to the target being reached more or less immediately.

In such a case, it might be better to have a single TxPDO occupied by the statusword only and having the transmission type of this PDO being 255. So it's updated immediately after every change of the statusword. The PDOs carrying the actual values – if used at all – will still have to be transferred synched.

Error handling (0x6007)

By default a CANopen servo drive will not be switched out of the switch on disabled state of the drive state machine of CiA 402, unless the communication is in operational state⁵. The Abort connection option code (0x6007.00) determines the reaction of the drive function, when the communication is lost after the drive has been started.

Options are:

- Don't care
- Switch into the fault state of the drive state machine, which will usually stop the drive
- Switch off the power
- switch into the quick-stop state of the drive state machine

Don't care might not be a safe selection. If CAN is the active interface and the communication is disturbed, the drive should be stopped. Which one of the stops is selected depends on the application. The reaction of the drive when switched into the quick-stop state or the fault state is configured using the quick-stop option code or the fault reaction option code.

Literature

[1] CiA 102 Physical Layer for industrial applications

[2] CiA 301 CANopen application layer and communication profile

[3] CiA 401 CANopen device profile for generic I/O modules

[4] CiA 402 CANopen device profile for drives and motion control

[5] CiA 305 Layer Setting Services

⁵ Drives having several interfaces might have a different behavior. For a FAULHABER MotionController generation V3.0 the behavior is configurable.

Rechtliche Hinweise

Urheberrechte. Alle Rechte vorbehalten. Ohne vorherige ausdrückliche schriftliche Zustimmung der Dr. Fritz Faulhaber & Co. KG darf diese Application Note oder Teile dieser unabhängig von dem Zweck insbesondere nicht vervielfältigt, reproduziert, gespeichert (z.B. in einem Informationssystem) oder be- oder verarbeitet werden.

Gewerbliche Schutzrechte. Mit der Veröffentlichung, Übergabe/Übersendung oder sonstigen Zur-Verfügung-Stellung dieser Application Note werden weder ausdrücklich noch konkludent Rechte an gewerblichen Schutzrechten, übertragen noch Nutzungsrechte oder sonstige Rechte an diesen eingeräumt. Dies gilt insbesondere für gewerbliche Schutzrechte, die mittelbar oder unmittelbar den beschriebenen Anwendungen und/oder Funktionen dieser Application Note zugrunde liegen oder mit diesen in Zusammenhang stehen.

Kein Vertragsbestandteil; Unverbindlichkeit der Application Note. Die Application Note ist nicht Vertragsbestandteil von Verträgen, die die Dr. Fritz Faulhaber GmbH & Co. KG abschließt, und der Inhalt der Application Note stellt auch keine Beschaffenheitsangabe für Vertragsprodukte dar, soweit in den jeweiligen Verträgen nicht ausdrücklich etwas anderes vereinbart ist. Die Application Note beschreibt unverbindlich ein mögliches Anwendungsbeispiel. Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt insbesondere keine Gewährleistung oder Garantie dafür und steht auch insbesondere nicht dafür ein, dass die in der Application Note illustrierten Abläufe und Funktionen stets wie beschrieben aus- und durchgeführt werden können und dass die in der Application Note beschriebenen Abläufe und Funktionen in anderen Zusammenhängen und Umgebungen ohne zusätzliche Tests oder Modifikationen mit demselben Ergebnis umgesetzt werden können. Der Kunde und ein sonstiger Anwender müssen sich jeweils im Einzelfall vor Vertragsabschluss informieren, ob die Abläufe und Funktionen in ihrem Bereich anwendbar und umsetzbar sind.

Keine Haftung. Die Dr. Fritz Faulhaber GmbH & Co. KG weist darauf hin, dass aufgrund der Unverbindlichkeit der Application Note keine Haftung für Schäden übernommen wird, die auf die Application Note und deren Anwendung durch den Kunden oder sonstigen Anwender zurückgehen. Insbesondere können aus dieser Application Note und deren Anwendung keine Ansprüche aufgrund von Verletzungen von Schutzrechten Dritter, aufgrund von Mängeln oder sonstigen Problemen gegenüber der Dr. Fritz Faulhaber GmbH & Co. KG hergeleitet werden.

Änderungen der Application Note. Änderungen der Application Note sind vorbehalten. Die jeweils aktuelle Version dieser Application Note erhalten Sie von Dr. Fritz Faulhaber GmbH & Co. KG unter der Telefonnummer +49 7031 638 688 oder per Mail von mcsupport@faulhaber.de.

Legal notices

Copyrights. All rights reserved. This Application Note and parts thereof may in particular not be copied, reproduced, saved (e.g. in an information system), altered or processed in any way irrespective of the purpose without the express prior written consent of Dr. Fritz Faulhaber & Co. KG.

Industrial property rights. In publishing, handing over/dispatching or otherwise making available this Application Note Dr. Fritz Faulhaber & Co. KG does not expressly or implicitly grant any rights in industrial property rights nor does it transfer rights of use or other rights in such industrial property rights. This applies in particular to industrial property rights on which the applications and/or functions of this Application Note are directly or indirectly based or with which they are connected.

No part of contract; non-binding character of the Application Note. The Application Note is not a constituent part of contracts concluded by Dr. Fritz Faulhaber & Co. KG and the content of the Application Note does not constitute any contractual quality statement for products, unless expressly set out otherwise in the respective contracts. The Appli-

cation Note is a non-binding description of a possible application. In particular Dr. Fritz Faulhaber & Co. KG does not warrant or guarantee and also makes no representation that the processes and functions illustrated in the Application Note can always be executed and implemented as described and that they can be used in other contexts and environments with the same result without additional tests or modifications. The customer and any user must inform themselves in each case before concluding a contract concerning a product whether the processes and functions are applicable and can be implemented in their scope and environment.

No liability. Owing to the non-binding character of the Application Note Dr. Fritz Faulhaber & Co. KG will not accept any liability for losses arising from its application by customers and other users. In particular, this Application Note and its use cannot give rise to any claims based on infringements of industrial property rights of third parties, due to defects or other problems as against Dr. Fritz Faulhaber GmbH & Co. KG.

Amendments to the Application Note. Dr. Fritz Faulhaber & Co. KG reserves the right to amend Application Notes. The current version of this Application Note may be obtained from Dr. Fritz Faulhaber & Co. KG by calling +49 7031 638 688 or sending an e-mail to mcsupport@faulhaber.de.