item

CANopen Manual

CANopen

item Servo Positioning Controller C Series

item Industrietechnik GmbH
Friedenstraße 107-109
42699 Solingen
Germany

Phone: +49-(0)212-6580-0
Fax: +49-(0)212-6580-310
E-mail: info@item24.com

Fehler! Verweisquelle konnte nicht gefunden werden.

Table of contents

Table of Figures

1 General Terms

1.1 Documentation

This manual describes how to parametrize and control the using the standardised protocol CANopen. The adjustment of the physical parameters, the activation of the CANopen protocol, the embedding into a CAN network and the communication with the controller will be explained. It is intended for persons who are already well versed with the servo positioning controller series.

It contains safety notes that have to be noticed.

For more information, please refer to the following manuals of the :

- **Software Manual "item Servo Positioning Controller C Series":** Description of the device functionality and the software functions of the firmware including RS232 communication. Description of the parameterisation program ™ with instructions on the commissioning of the .
- **Product Manual "item Servo Positioning Controller C 1-Series":** Description of the technical specifications and the device functionality as well as notes on the installation and the operation of the .
- **Product manual "item Servo Positioning Controller C 3-Series":** Description of the technical data and the device functionality plus notes concerning the installation and operation of the .
- **CANopen Manual "item Servo Positioning Controller C 3-Series":** Description of the implemented CANopen protocol as per DSP402.
- **PROFIBUS Manual "item Servo Positioning Controller C 3-Series":** Description of the implemented PROFIBUS-DP protocol.

1.2 CANopen

CANopen is a standard established by the association "CAN in Automation". A great number of device manufacturers are organised in this association. This standard has replaced most of all manufacturer-specific CAN protocols. So a manufacturer independent communication interface is available for the user:

CiA Draft Standard 201...207: In these standards the general network administration and the transfer of objects are determined. This book is rather comprehensive. The relevant aspects are treated in the CANopen manual in hand so that it is not necessary in general to acquire the DS201..207.

CiA Draft Standard 301: In this standard the basic structure of the object dictionary of a CANopen device and the access to this directory are described. Besides this the statements made in the DS201..207 are described in detail. The elements needed for the of the object directory and the access methods which belong to them are described in the present manual. It is advisable to acquire the DS301 but not necessary.

CiA Draft Standard 402: This standard describes the concrete implementation of CANopen in servo controllers. Though all implemented objects are also briefly documented and described in this CANopen manual the user should own this book.

Order address

CAN in Automation (CiA) International Headquarter
Am Weichselgarten 26
D-91058 Erlangen
Tel. +49-09131-601091
Fax: +49-09131-601092
www.can-cia.de

The underlying standards are used by the implementation of the CANopen:

[1]	CiA Draft Standard 301,	Version 4.02,	13. Februar	2002
[2]	CiA Draft Standard Proposal 402,	Version 2.0,	26. Juli	2002

2 Safety Notes for electrical drives and controls

2.1 Symbols and signs

ℹ	Information Important informations and notes.
⚠	Caution! The nonobservance can result in high property damage.
✋	DANGER! The nonobservance can result in property **damages** and in **injuries to persons**.
⚡	Caution! High voltage. The note on safety contains a reference to a possibly occurring life dangerous voltage.

?! The parts of this document marked with this sign should give examples to make it easier to understand the use of single objecs and parameters.

2.2 General notes

In case of damage resulting from non-compliance with the safety notes in this manual, will not assume any liability.

> Prior to the initial use you must read the chapters Safety Notes for electrical drives and controls *starting on page 14*

If the documentation in the language at hand is not understood accurately, please contact and inform your supplier.

Sound and safe operation of the servo drive controller requires proper and professional transportation, storage, assembly and installation as well as proper operation and maintenance. Only trained and qualified personnel may handle electrical devices:

TRAINED AND QUALIFIED PERSONNEL

in the sense of this product manual or the safety notes on the product itself are persons who are sufficiently familiar with the setup, assembly, commissioning and operation of the product as well as all warnings and precautions as per the instructions in this manual and who are sufficiently qualified in their field of expertise:

- Education and instruction or authorisation to switch devices/systems on and off and to ground them as per the standards of safety engineering and to efficiently label them as per the job demands.
- Education and instruction as per the standards of safety engineering regarding the maintenance and use of adequate safety equipment.
- First aid training.

The following notes must be read prior to the initial operation of the system to prevent personal injuries and/or property damages:

These safety notes must be complied with at all times.

Do not try to install or commission the servo drive controller before carefully reading all safety notes for electrical drives and controllers contained in this document. These safety instructions and all other user notes must be read prior to any work with the servo drive controller.

In case you do not have any user notes for the servo positioning controller, please contact your sales representative. Immediately demand these documents to be sent to the person responsible for the safe operation of the servo drive controller.

If you sell, rent and/or otherwise make this device available to others, these safety notes must also be included.

The user must not open the servo drive controller for safety and warranty reasons.

Professional control process design is a prerequisite for sound functioning of the servo drive controller!

DANGER!

Inappropriate handling of the servo drive controller and non-compliance of the warnings as well as inappropriate intervention in the safety features may result in property damage, personal injuries, electric shock or in extreme cases even death.

2.3 Danger resulting from misuse

DANGER!

High electrical voltages and high load currents!

Danger to life or serious personal injury from electrical shock!

DANGER!

High electrical voltage caused by wrong connections!

Danger to life or serious personal injury from electrical shock!

DANGER!

Surfaces of device housing may be hot!

Risk of injury! Risk of burning!

DANGER!

Dangerous movements!

Danger to life, serious personal injury or property damage due to unintentional movements of the motors!

2.4 Safety notes

2.4.1 General safety notes

The servo drive controller corresponds to IP20 class of protection as well as pollution level 1. Make sure that the environment corresponds to this class of protection and pollution level.

Only use replacements parts and accessories approved by the manufacturer.

The devices must be connected to the mains supply as per EN regulations, so that they can be cut off the mains supply by means of corresponding separation devices (e.g. main switch, contactor, power switch).

The servo drive controller may be protected using an AC/DC sensitive 300mA fault current protection switch (RCD = Residual Current protective Device).

Gold contacts or contacts with a high contact pressure should be used to switch the control contacts.

Preventive interference rejection measures should be taken for control panels, such as connecting contactors and relays using RC elements or diodes.

The safety rules and regulations of the country in which the device will be operated must be complied with.

The environment conditions defined in the product documentation must be kept. Safety-critical applications are not allowed, unless specifically approved by the manufacturer.

For notes on installation corresponding to EMC, please refer to Product Manual . The compliance with the limits required by national regulations is the responsibility of the manufacturer of the machine or system.

The technical data and the connection and installation conditions for the servo drive controller are to be found in this product manual and must be met.

DANGER!

The general setup and safety regulations for work on power installations (e.g. DIN, VDE, EN, IEC or other national and international regulations) must be complied with.

Non-compliance may result in death, personal injury or serious property damages.

Without claiming completeness, the following regulations and others apply:

VDE 0100	Regulations for the installation of high voltage (up to 1000 V) devices
EN 60204	Electrical equipment of machines
EN 50178	Electronic equipment for use in power installations

2.4.2 Safety notes for assembly and maintenance

The appropriate DIN, VDE, EN and IEC regulations as well as all national and local safety regulations and rules for the prevention of accidents apply for the assembly and maintenance of the system. The plant engineer or the operator is responsible for compliance with these regulations:

The servo drive controller must only be operated, maintained and/or repaired by personnel trained and qualified for working on or with electrical devices.

Prevention of accidents, injuries and/or damages:

Additionally secure vertical axes against falling down or lowering after the motor has been switched off, e.g. by means of:

- Mechanical locking of the vertical axle,
- External braking, catching or clamping devices or
- Sufficient balancing of the axle.

The motor holding brake supplied by default or an external motor holding brake driven by the drive controller alone is not suitable for personal protection!

Render the electrical equipment voltage-free using the main switch and protect it from being switched on again until the DC bus circuit is discharged, in the case of:

- Maintenance and repair work
- Cleaning
- long machine shutdowns

Prior to carrying out maintenance work make sure that the power supply has been turned off, locked and the DC bus circuit is discharged.

The external or internal brake resistor carries dangerous DC bus voltages during operation of the servo drive controller and up to 5 minutes thereafter. Contact may result in death or serious personal injury.

Be careful during the assembly. During the assembly and also later during operation of the drive, make sure to prevent drill chips, metal dust or assembly parts (screws, nuts, cable sections) from falling into the device.

Also make sure that the external power supply of the controller (24V) is switched off.

The DC bus circuit or the mains supply must always be switched off prior to switching off the 24V controller supply.

Carry out work in the machine area only, if AC and/or DC supplies are switched off. Switched off output stages or controller enablings are no suitable means of locking. In the case of a malfunction the drive may accidentally be put into action.

Initial operation must be carried out with idle motors, to prevent mechanical damages e.g. due to the wrong direction of rotation.

Electronic devices are never fail-safe. It is the user's responsibility, in the case an electrical device fails, to make sure the system is transferred into a secure state.

The servo drive controller and in particular the brake resistor, externally or internally, can assume high temperatures, which may cause serious burns.

2.4.3 Protection against contact with electrical parts

This section only concerns devices and drive components carrying voltages exceeding 50 V. Contact with parts carrying voltages of more than 50 V can be dangerous for people and may cause electrical shock. During operation of electrical devices some parts of these devices will inevitably carry dangerous voltages.

DANGER!

High electrical voltage!

Danger to life, danger due to electrical shock or serious personal injury!

The appropriate DIN, VDE, EN and IEC regulations as well as all national and local safety regulations and rules for the prevention of accidents apply for the assembly and maintenance of the system. The plant engineer or the operator is responsible for compliance with these regulations:

Before switching on the device, install the appropriate covers and protections against accidental contact. Rack-mounted devices must be protected against accidental contact by means of a housing, e.g. a switch cabinet. The regulations VBG 4 must be complied with!

Always connect the ground conductor of the electrical equipment and devices securely to the mains supply. Due to the integrated line filter the leakage current exceeds 3.5 mA!

Comply with the minimum copper cross-section for the ground conductor over its entire length as per EN60617!

Prior to the initial operation, even for short measuring or testing purposes, always connect the ground conductor of all electrical devices as per the terminal diagram or connect it to the ground wire. Otherwise the housing may carry high voltages which can cause electrical shock.

Do not touch electrical connections of the components when switched on.

Prior to accessing electrical parts carrying voltages exceeding 50 Volts, disconnect the device from the mains or power supply. Protect it from being switched on again.

For the installation the amount of DC bus voltage must be considered, particularly regarding insulation and protective measures. Ensure proper grounding, wire dimensioning and corresponding short-circuit protection.

The device comprises a rapid discharge circuit for the DC bus as per EN60204 section 6.2.4. In certain device constellations, however, mostly in the case of parallel connection of several servo drive controllers in the DC bus or in the case of an unconnected brake resistor, this rapid discharge may be rendered ineffective. The servo drive controllers can carry voltage until up to 5 minutes after being switched off (residual capacitor charge).

2.4.4 Protection against electrical shock by means of protective extra-low voltage (PELV)

All connections and terminals with voltages between 5 and 50 Volts at the servo drive controller are protective extra-low voltage, which are designed safe from contact in correspondence with the following standards:

International: IEC 60364-4-41

European countries within the EU: EN 50178/1998, section 5.2.8.1.

DANGER!

High electrical voltages due to wrong connections!

Danger to life, risk of injury due to electrical shock!

Only devices and electrical components and wires with a protective extra low voltage (PELV) may be connected to connectors and terminals with voltages between 0 to 50 Volts.

Only connect voltages and circuits with protection against dangerous voltages. Such protection may be achieved by means of isolation transformers, safe optocouplers or battery operation.

2.4.5 Protection against dangerous movements

Dangerous movements can be caused by faulty control of connected motors, for different reasons:

- Improper or faulty wiring or cabling
- Error in handling of components
- Error in sensor or transducer
- Defective or non-EMC-compliant components
- Error in software in superordinated control system

These errors can occur directly after switching on the device or after an indeterminate time of operation.

The monitors in the drive components for the most part rule out malfunctions in the connected drives. In view of personal protection, particularly the danger of personal injury and/or property damage, this may not be relied on exclusively. Until the built-in monitors come into effect, faulty drive movements must be taken into account; their magnitude depends on the type of control and on the operating state.

DANGER!

Dangerous movements!

Danger to life, risk of injury, serious personal injuries or property damage!

For the reasons mentioned above, personal protection must be ensured by means of monitoring or superordinated measures on the device. These are installed in accordance with the specific data of the system and a danger and error analysis by the manufacturer. The safety regulations applying to the system are also taken into consideration. Random movements or other malfunctions may be caused by switching the safety installations off, by bypassing them or by not activating them.

2.4.6 Protection against contact with hot parts

DANGER!

Housing surfaces may be hot!

Risk of injury! Risk of burning!

Do not touch housing surfaces in the vicinity of heat sources! Danger of burning!

Before accessing devices let them cool down for 10 minutes after switching them off.

Touching hot parts of the equipment such as the housing, which contain heat sinks and resistors, may cause burns!

2.4.7 Protection during handling and assembly

Handling and assembly of certain parts and components in an unsuitable manner may under adverse conditions cause injuries.

DANGER!

Risk of injury due to improper handling!

Personal injury due to pinching, shearing, cutting, crushing!

The following general safety notes apply:

Comply with the general setup and safety regulations on handling and assembly.

Use suitable assembly and transportation devices.

Prevent incarcerations and contusions by means of suitable protective measures.

Use suitable tools only. If specified, use special tools.

Use lifting devices and tools appropriately.

If necessary, use suitable protective equipment (e.g. goggles, protective footwear, protective gloves).

Do not stand underneath hanging loads.

Remove leaking liquids on the floor immediately to prevent slipping.

3 Cabling and pin assignment

3.1 Pin assignment

At the the CAN interface is already integrated in the device and therefore always available.

According to the CANopen specification a 9-pin DSUB-plug (male) is integrated in the device.

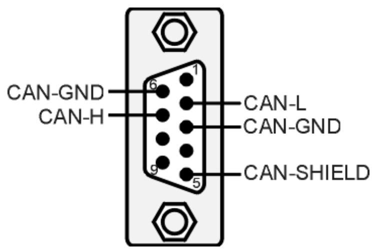


Figure 3.1: CAN connector for

CAN bus cabling

Please respect carefully the following information and notes for the cabling of the controller to get a stable and undisturbed communication system. A non professional cabling can cause malfunctions of the CAN bus which hence the controller to shutdown with an error.

120Ω Termination resistor

No termination resistor is integrated in the item servo positioning controller C 1-Series

3.2 Cabling hints

The CAN bus offers an easy and safe way to connect all parts of a plant. As condition all following instructions have to be respected carefully.

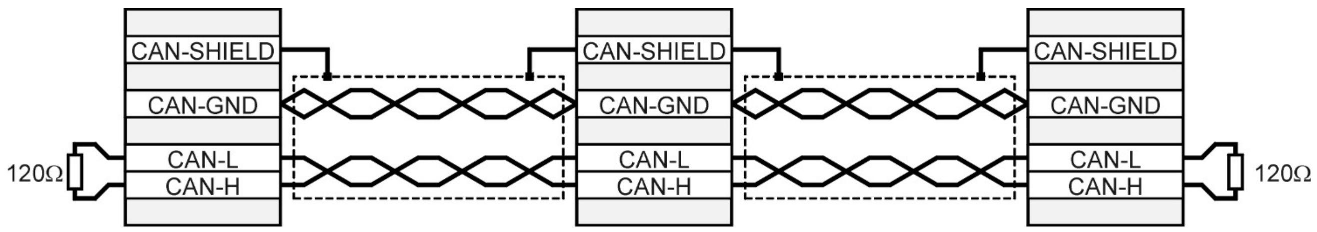


Figure 3.2: Cabling (schematically)

- All nodes of a network are principally connected in series, so that the CAN cable is looped through all controllers (see **Figure 3.2**).
- The two ends of the CAN cable have to be terminated by a resistor of 120**Ω** +/- 5%. Please note that such a resistor is often already installed in CAN cards or the PLC.
- For cabling **shielded** cable with exactly two **twisted** pairs have to be used.
  - One twisted pair is used for CAN-H and CAN-L.
  - One twisted pair is used commonly for CAN-GND.
  - The shield of the cable is connected to CAN-SHIELD at all nodes.

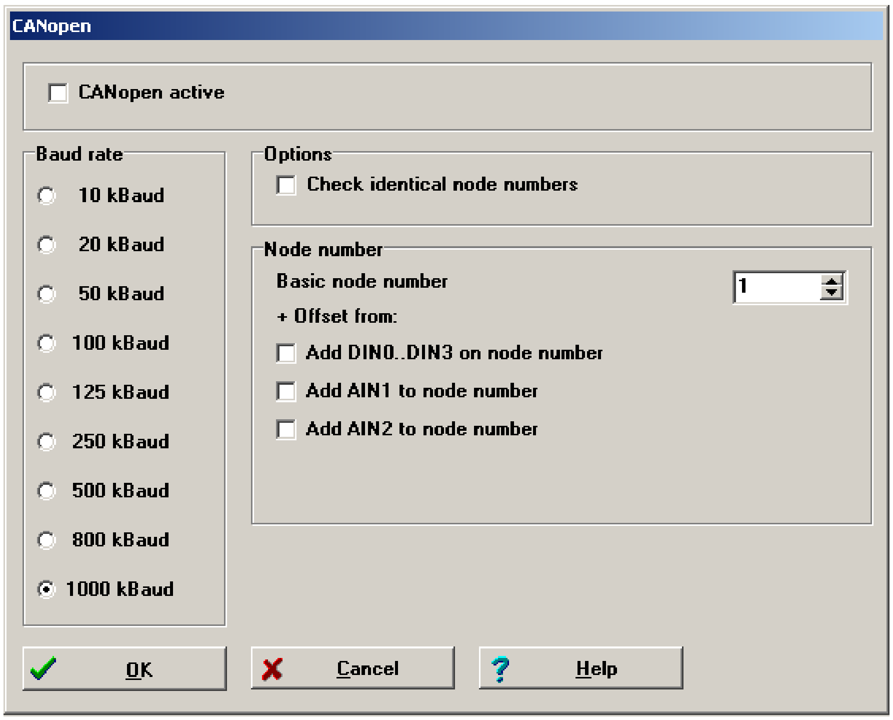
  A table with technical data of suitable cables can be found at the end of this chapter. Recommended cables can be found in the product manual.
- We dissuade from using connectors in between the CAN bus line. If it is still necessary to use connectors, assure that the connection of the shield is done by using metallic cases.
- For less noise injection principally
  - Never place motorcables parallel to signalcables.
  - Use only motorcables specified by .
  - Shield and earth motorcables correctly.
- For further informations refer to the Controller Area Network protocol specification, Ver. 2.0, Robert Bosch GmbH, 1991.
- Technical data CAN bus cable:

2 twisted pairs, d ≥ 0,22 mm^2	loop resistance < 0,2 Ω/m
shielded	char. impedance 100-120 Ω

4 Activation of CANopen

4.1 Survey

The activation of CANopen is done one-time using the serial interface of the servo controller. The CAN protocoll can be activated in the window “CANopen” of the ™.



CANopen

☐ CANopen active

Baud rate
- ○ 10 kBaud
- ○ 20 kBaud
- ○ 50 kBaud
- ○ 100 kBaud
- ○ 125 kBaud
- ○ 250 kBaud
- ○ 500 kBaud
- ○ 800 kBaud
- ● 1000 kBaud

Options
- ☐ Check identical node numbers

Node number
- Basic node number: 1
- + Offset from:
- ☐ Add DIN0..DIN3 on node number
- ☐ Add AIN1 to node number
- ☐ Add AIN2 to node number

There have to be set three different parameters:

- Basic Node Number

  For unmistakable identification each user within the network has to have an unique node number. The node number is used to address the device.

  As an option it is possible to calculate the node number dependent of the plug-in location of the device. Therefore once after reset the combination of digital inputs (DIN0...DIN3) or analogue inputs AIN1 and AIN2 is added to the basic node number. AIN1 will be added with a valence of 32 and AIN2 with a valence of 64, if the particular input is connected to

  Vref = 10V.

- Baudrate

  This parameter determines the used baudrate in kBaud. Please note that high baudrates can only be achieved with short cable length.

- Options

  All CANopen nodes send a bootup message containing their own node number. If the servo positioning controller receives such a message containing its own node number, the error 12-0 will be raised.

Finally the CANopen protocoll can be activated. Please take into account that the parameters mentioned above can only be changed when the protocoll is deactivated.

Please note that the activation of CANopen will only be available after a reset if the parameter set has been saved.

5 Access methods

5.1 Survey

CANopen offers an easy and standardised way to access all parameters of the servo controller (e.g. the maximum current). To achieve a clear arrangement a unique *index* and *subindex* is assigned to every parameter (*CAN object*). The parameters altogether form the so called *object dictionary*.

The object dictionary can be accessed via CAN bus in primarily two ways: A confirmed access with so called SDOs and a unconfirmed access using so called PDOs with no hand-shake.

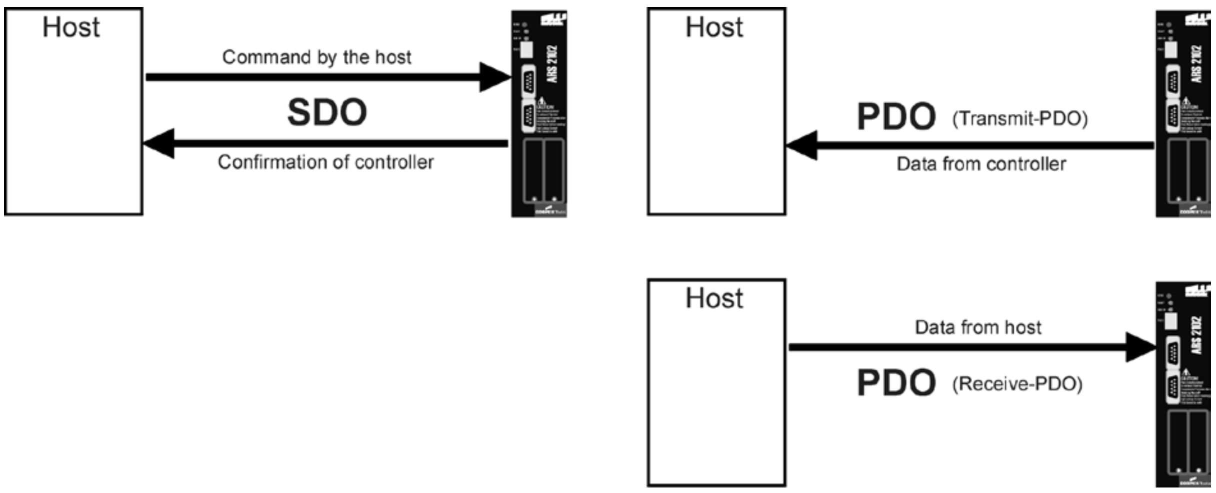


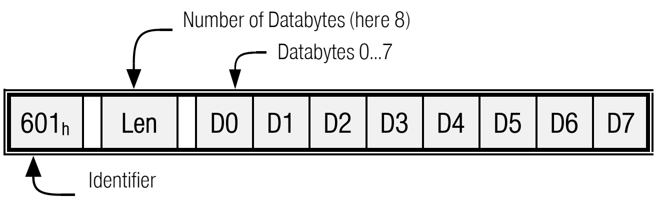
Figure 5.3: Access methods

As a rule the servo controller will be configured and controlled by SDOs. Additional types of messages (so called Communication Objects, COB) are defined for special applications. They will be sent either by the superimposed control or the servo controller:

SDO	**S**ervice **D**ata **O**bject	Used for normal parametrization of the servo controller
PDO	**P**rocess **D**ata **O**bject	Fast exchange of process data (e.g. velocity actual value) possible.
SYNC	**Sync**hronization Message	Synchronisation of several CAN nodes.

EMCY	Emergency Message	Used to transmit error messages of the servo controller.
NMT	Network Management	Used for network services. For example the user can act on all controllers at the same time via this object type.
HEARTBEAT	Error Control Protocol	Used for observing all nodes by cyclic messages.

Every message sent via CAN bus contains an address to identify the node the message is meant for. This address is called *Identifier*. The lower the identifier, the higher the priority. Each communication object mentioned above has a specific identifier.The following figure shows the schematic structure of a CANopen message:



5.2 Access by SDO

The object dictionary can be accessed with **S**ervice **D**ata **O**bjects (SDO). This access is particularly easy and clear. Therefore it is recommended to base the application on SDOs first and later adapt some accesses to the certainly faster but more complicated **P**rocess **D**ata **O**bjects (PDOs).
SDO accesses always start from the superimposed control (host). The host sends a write request to change a parameter or a read request to get a parameter from the servo controller. Every request will be answered by the servo controller either sending the requested parameter or confirming the write request. Every command has to be sent with a definite identifier so that the servo controller knows what command is intended for it.
This identifier is composed of the base 600_h + node number of the corresponding servo controller. The servo controller answers with identifier 580_h + node number.
The structure of the writing and reading sequences depends on the data type as 1, 2 or 4 data bytes have to be sent or received. The following data types will be supported:

UINT8	8 bit value, unsigned	0	...	255
INT8	8 bit value, signed	-128	...	127
UINT16	16 bit value, unsigned	0	...	65535

INT16	16 bit value, signed	-32768	...	32767
UINT32	32 bit value, unsigned	0	...	$(2^{32}-1)$
INT32	32 bit value, signed	$-(2^{31})$	...	$(2^{31}-1)$

5.2.1 SDO sequences to read or write parameters

Following sequences have to be used to read or write can objects of mentioned type. Commands to write a value into the servo controller start with a different token depending on the parameters data type, whereas the first token of the answer is always the same. For commands to read parameters it is vice versa: They always start with the same token, whereas the answer of the servo controller starts with a token depending on the parameters data type. For all numerical values the hexadecimal notation is used.

Read commands / Write commands

Low-Byte of main index (hex)

High-Byte of main index (hex)

Subindex (hex)

UINT8 / INT8

Token for 8 Bit

	Read commands					Write commands				
Command	40_h	IX0	IX1	SU		$2F_h$	IX0	IX1	SU	D0
Answer	$4F_h$	IX0	IX1	SU	D0	60_h	IX0	IX1	SU	

Token for 8 Bit

Token for 16 Bit

UINT16 / INT16

	Read commands						Write commands					
Command	40_h	IX0	IX1	SU			$2B_h$	IX0	IX1	SU	D0	D1
Answer	$4B_h$	IX0	IX1	SU	D0	D1	60_h	IX0	IX1	SU		

Token for 16 Bit

Token for 32 Bit

UINT32 / INT32

	Read commands								Write commands							
Command	40_h	IX0	IX1	SU					23_h	IX0	IX1	SU	D0	D1	D2	D3
Answer	43_h	IX0	IX1	SU	D0	D1	D2	D3	60_h	IX0	IX1	SU				

Token for 32 Bit

EXAMPLE

UINT8 / INT8

Reading of Obj. 6061_00_h
Returning data: 01_h

Writing of Obj. 1401_02_h
Data: EF_h

Command:	40_h	61_h	60_h	00_h		$2F_h$	01_h	14_h	02_h	EF_h
Answer:	$4F_h$	61_h	60_h	00_h	01_h	60_h	01_h	14_h	02_h	

UINT16 / INT16

Reading of Obj. 6041_00_h
Returning data: 1234_h

Writing of Obj. 6040_00_h
Data: $03E8_h$

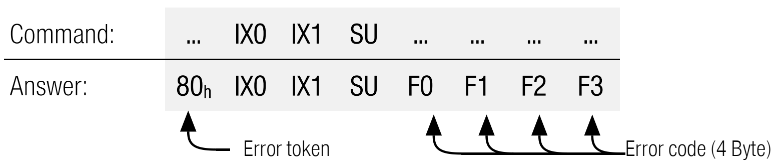
Command:	40_h	41_h	60_h	00_h			$2B_h$	40_h	60_h	00_h	$E8_h$	03_h

		Reading	Writing
	Answer:	4B$_h$ 41$_h$ 60$_h$ 00$_h$ 34$_h$ 12$_h$	60$_h$ 40$_h$ 60$_h$ 00$_h$
UINT32 / INT32		Reading of Obj. 6093_01$_h$ Returning data: 12345678$_h$	Writing of Obj. 6093_01$_h$ Data: 12345678$_h$
	Command:	40$_h$ 93$_h$ 60$_h$ 01$_h$	23$_h$ 93$_h$ 60$_h$ 01$_h$ 78$_h$ 56$_h$ 34$_h$ 12$_h$
	Answer:	43$_h$ 93$_h$ 60$_h$ 01$_h$ 78$_h$ 56$_h$ 34$_h$ 12$_h$	60$_h$ 93$_h$ 60$_h$ 01$_h$

Always wait for the acknowledge of the controller!
Only if a request has been acknowledged by the controller it is allowed to send the next request.

5.2.2 SDO-error messages

If an error occurs while reading or writing an object (e.g. because the value is out of range) the servo controller answers with an error message instead of the normal answer:



Error code F3 F2 F1 F0	Description
05 03 00 00$_h$	Toggle bit not alternated
05 04 00 01$_h$	Client / server command specifier not valid or unknown
06 01 00 00$_h$	Unsupported access to an object
06 01 00 01$_h$	Attempt to read a write only object
06 01 00 02$_h$	Attempt to write a read only object
06 02 00 00$_h$	Object does not exist in the object dictionary
06 04 00 41$_h$	Object cannot be mapped to the PDO
06 04 00 42$_h$	The number and length of the objects to be mapped would exceed PDO length
06 04 00 47$_h$	General internal incompatibility in the device
06 07 00 10$_h$	Data type does not match, length of service parameter does not match
06 07 00 12$_h$	Data type does not match, length of service parameter too high
06 07 00 13$_h$	Data type does not match, length of service parameter too low
06 09 00 11$_h$	Sub-index does not exist
06 04 00 43$_h$	General parameter incompatibility
06 06 00 00$_h$	Access failed due to an hardware error *1)
06 09 00 30$_h$	Value range of parameter exceeded
06 09 00 31$_h$	Value of parameter written too high
06 09 00 32$_h$	Value of parameter written too low
06 09 00 36$_h$	Maximum value is less than minimum value
08 00 00 20$_h$	Data cannot be transferred or stored to the application *1)
08 00 00 21$_h$	Data cannot be transferred or stored to the application because of local control
08 00 00 22$_h$	Data cannot be transferred or stored to the application because of the present device state *3)

Error code F3 F2 F1 F0	Description
08 00 00 23h	No Object Dictionary is present *2)

*1) According to DS301 used on invalid access to store_parameters / restore_parameters

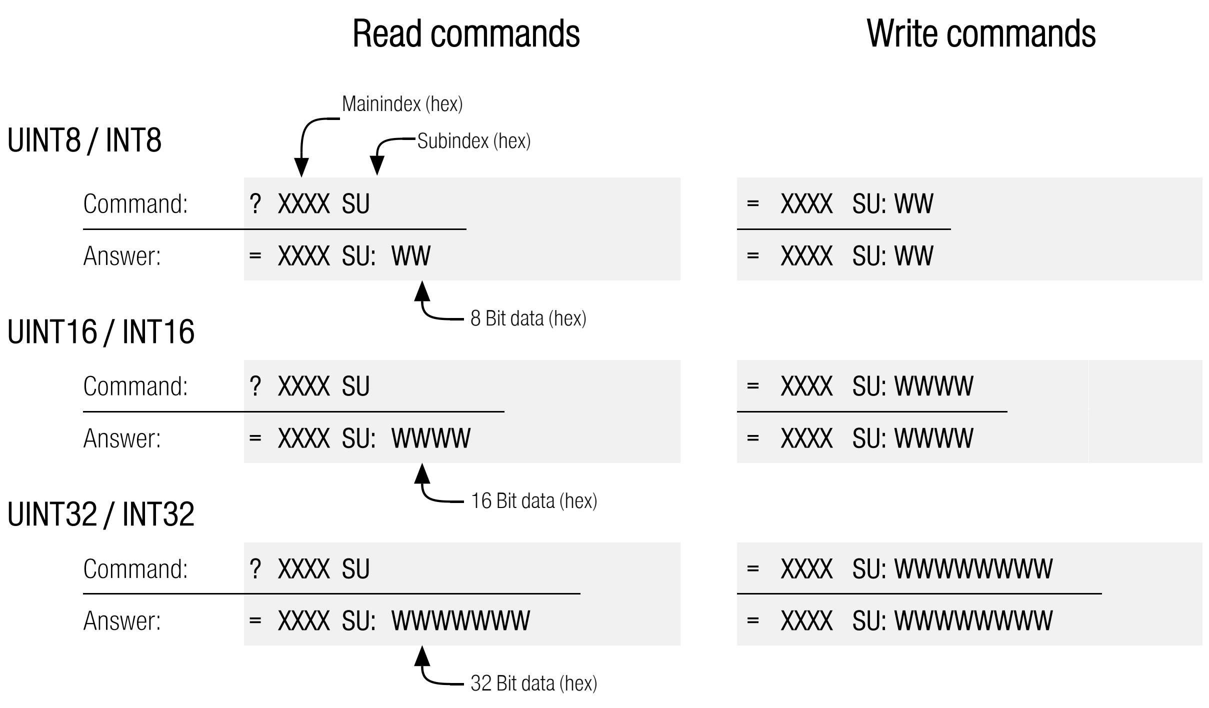
*2) This abort codes signals that another fieldbus controls the servo or the access to the parameter is not allowed.

*3) "Device State" is used generally: This can be a wrong mode of operation as well as a missing technology module.

5.2.3 Simulation of SDO accesses via RS232

The firmware of the servo controller offers the option to simulate SDO accesses via the serial port. Consequently it is possible to check objects written to the controller via CAN bus by using the serial port. Particularly using the transfer window of ™ (see **File | Transfer**) will simplify the building of applications.

The following syntax has to be used:



```
                     Read commands                    Write commands

UINT8 / INT8
     Command:   ?  XXXX  SU                        =  XXXX  SU: WW
     Answer:    =  XXXX  SU:  WW                   =  XXXX  SU: WW

UINT16 / INT16
     Command:   ?  XXXX  SU                        =  XXXX  SU: WWWW
     Answer:    =  XXXX  SU:  WWWW                 =  XXXX  SU: WWWW

UINT32 / INT32
     Command:   ?  XXXX  SU                        =  XXXX  SU: WWWWWWWW
     Answer:    =  XXXX  SU:  WWWWWWWW             =  XXXX  SU: WWWWWWWW
```

Read error Write error

Command:	? XXXX SU	= XXXX SU: WWWWWWWW [1)]
Answer:	! FFFFFFFF	! FFFFFFFF
	32 Bit — Error code *F3 F2 F1 F0* accord. Chap. 5.2.2	32 Bit — Error code *F3 F2 F1 F0* accord. Chap. 5.2.2

1) The answer is build similarly for each 3 commands (8, 16, 32 Bit).

Please note that the commands are composed out of chars without spaces.

Never use this access mechanism in real applications!

The access via RS232 is only implemented for checking your application. The protocol is inapplicable for real time access to the controller and may cause errors.

In addition the syntax of this protocol may change without notice.

5.3 PDO-Message

Process Data Objects (PDOs) are suitable to transmit data event-controlled, whereat the PDO contains one or more predefined parameters. In contrast to SDOs no hand-shake is used. So the receiver has to be able to handle an arriving PDO at any time. In most cases this requires a great deal of software in the host computer. This disadvantage is in contrast to the advantage that the host computer does not need cyclically inquiry of the objects embedded in a PDO, which means a strong reduction of bus load.

EXAMPLE

The host computer wants to know when the servo controller has reached the target position at a positioning from A to B.

If SDOs are used the host constantly has to poll the object **statusword**, e.g. every millisecond, thus loading the bus capacity more or less depending on the request cycle time.

If PDOs are used the servo controller is configured at the start of an application in such a way that a PDO including the **statusword** is send on each modification of the **statusword**.

So the host computer does not need to poll the statusword all the time. Instead a message is send to the host automatically if the specified event occurs.

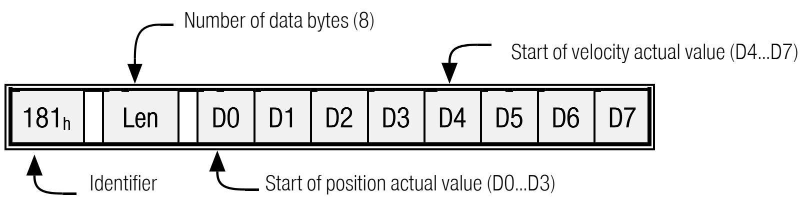
Following types of PDOs can be differenced:

Transmit-PDO	**(T-PDO)**	Servo → Host	Servo controller sends PDO if a certain event occurs
Receive-PDO	**(R-PDO)**	Host → Servo	Servo controller evaluates PDO if a certain event occurs

The servo controller disposes of four Transmit- and four Receive-PDOs.

Almost all parameters can be embedded (mapped) into a PDO, i.e. the PDO is for example composed of the velocity actual value, the position actual value or the like.

Before a PDO can be used the servo controller has to know, what data shall be transmitted, because a PDO only contains useful data and no information about the kind of parameter. In the following example the PDO contains the position actual value in the data byte D0...D3 and the velocity actual value in the data bytes D4...D7.


Number of data bytes (8)
Start of velocity actual value (D4...D7)
181h
Len
D0
D1
D2
D3
D4
D5
D6
D7
Identifier
Start of position actual value (D0...D3)

Almost any desired data frame can be built this way. The following chapter shows how to parametrize the servo controller for that purpose:

5.3.1 Description of objects

Identifier of PDOs

COB_ID_used_by_PDO

In the object COB_ID_used_by_PDO the desired identifier has to be entered. The PDO will be sent with this identifier. If bit 31 is set the associated PDO will be deactivated. This is the default setting.

It is prohibited to change the COB-ID if the PDO is not deactivated (= bit 31 set). Therefore a different identifier as the current one may only be written, if bit 31 is set.

A set bit 30 at reading the identifier indicates that it is not possible to request a PDO by a remote frame. This bit will be ignored at writing and is always set at reading.

Number of objects to be transmitted

number_of_mapped_objects

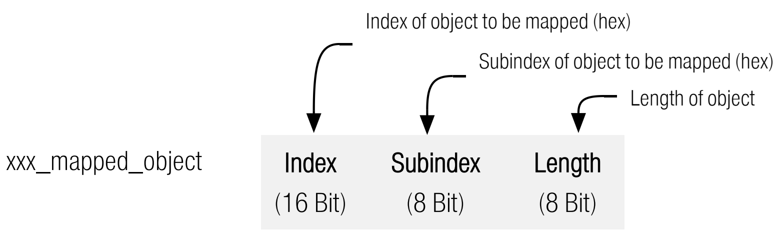
The object determines how many objects are mapped into the specific PDO. Following restrictions has to be respected:

- A maximum of 4 objects can be mapped into a PDO.
- The total length of a PDO must not exceed 64 bit.

Objects to be transmitted

first_mapped_object ... fourth_mapped_object

The host has to parametrize the index, the subindex and the length of each object that should be transmitted by the PDO. The length has to match with the length stored in the Object Dictionary.
Parts of an object cannot be mapped.
The following format has to be used:



To simplify the mapping the following sequence has to be used:

1.) The number_of_mapped_objects is set to 0.

2.) The parameter first_mapped_object...fourth_mapped_object can be parameterised (The length of all objects will not be considered at this time).

3.) The number_of_mapped_objects is set to a value between 1...4: The length of all mapped objects may not exceed 64 bit now.

Transmission type

transmission_type und **inhibit_time**

For each PDO it can be configured which event results in sending (Transmit-PDO) resp. evaluating (Receive-PDO) the PDO:

Value	Description	Allowed with
01_h –$F0_h$	SYNC message The value determines how many SYNC messages have to be received before the PDO will be - sent (T-PDO) resp. - evaluated (R-PDO).	TPDOs RPDOs
FE_h	Cyclic A Transfer-PDO will be updated and sent cyclic. The period is determined by the object **inhibit_time**. Receive-PDOs will be evaluated immediately after receipt.	TPDOs (RPDOs)
FF_h	On change The Transfer-PDO will be sent, if at least one bit of the PDO data has changed. Therefore the object **inhibit_time** determines the minimal period between two PDOs in multiples of 100µs.	TPDOs

The use of any other value for this parameter is inhibited.

Mask

transmit_mask_high and **transmit_mask_low**

Using the **transmission_type** "On change" the TPDO will always be sent if at least one bit has changed. Sometimes it is useful to send the TPDO only if a defined bit has changed. Therefore it is possible to mask the TPDO. Thereby only TPDO bits with an "1" in the corresponding bit of the mask will take effect to determine if the PDO has changed. This function is manufacturer specific and deactivated by default, i.e. all bits of the mask are set by default.

EXAMPLE

Following objects should be transmitted in a PDO:

Name of the object	Index_subindex	Meaning
statusword	$6041_h\_00_h$	Device Control
modes_of_operation_display	$6061_h\_00_h$	Operating mode
digital_inputs	$60FD_h\_00_h$	Digital inputs

The 1st Transmit-PDO (TPDO 1) should be used and should always be sent if a digital input changes but with a minimum repetition time of 10 ms. The PDO should use identifier 187_h.

1.) Deactivate a PDO

If the PDO is active, it must be deactivated at first.

Write identifier with set bit 31 (PDO wil be deactivated) ⇒ cob_id_used_by_pdo = $C0000187_h$

2.) Set number of mapped objects to 0

To enable the mapping, the number of mapped objects has to be zero. ⇒ number_of_mapped_objects = 0

3.) Parametrize objects to be mapped:

The above mentioned objects have to be assembled to a 32 bit value:

Index =6041_h	Subin. = 00_h	Length = 10_h	⇒ first_mapped_object	= 60410010_h
Index =6061_h	Subin. = 00_h	Length = 08_h	⇒ second_mapped_object	= 60610008_h
Index =$60FD_h$	Subin. = 00_h	Length = 20_h	⇒ third_mapped_object	= $60FD0020_h$

4.) Set number of mapped objects:

The PDO contains 3 objects ⇒ number_of_mapped_objects = 3_h

5.) Parametrize transmission type

The PDO should be sent if a digital input changes.	⇒ transmission_type =	FF_h
The PDO have to be masked in order to restrict the condition for a transmission of the PDO to a change of the digital inputs.	⇒ transmit_mask_high =	$00FFFF00_h$
	⇒ transmit_mask_low =	00000000_h
The PDO should be sent at most every 10 ms (100×100µs).	⇒ inhibit_time =	64_h

6.) Parametrize the identifier

The PDO should be transmitted with the identifier 187h.

Writing the new identifier and activating the PDO by resetting bit 31: ⇒ cob_id_used_by_pdo = 40000187_h

Please note that it is only allowed to change the settings of the PDOs if the Network state (NMT) is not **operational**. See also chapter 5.6.

5.3.2 Objects for parameterising PDOs

The servo positioning controller of the contain 4 Transmit- and 4 Receive-PDOs. The objects for parameterising these PDOs are equal for each 4 TPDOs and each 4 RPDOs. Therefore only the description for the first TPDO is stated below. It can be taken analogous for all the other PDOs, listed in a table thereafter.

Index	1800_h
Name	transmit_pdo_parameter_tpdo1
Object Code	RECORD
No. of Elements	3

Sub-Index	01_h
Description	cob_id_used_by_pdo_tpdo1
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	181_h...$1FF_h$, Bit 31 may be set
Default Value	$C0000181_h$

Sub-Index	02_h
Description	transmission_type_tpdo1
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0...$8C_h$, FE_h, FF_h
Default Value	FF_h

Sub-Index	03_h
Description	inhibit_time_tpdo1
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	100µs (i.e. 10 = 1ms)
Value Range	--

Default Value	0

Index	1A00$_h$
Name	transmit_pdo_mapping_tpdo1
Object Code	RECORD
No. of Elements	2

Sub-Index	00$_h$
Description	number_of_mapped_objects_tpdo1
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0...4
Default Value	0

Sub-Index	01$_h$
Description	first_mapped_object_tpdo1
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	--
Default Value	see Table

Sub-Index	02$_h$
Description	second_mapped_object_tpdo1
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	--
Default Value	see Table

Sub-Index	03$_h$
Description	third_mapped_object_tpdo1
Data Type	UINT32
Access	rw
PDO Mapping	no

Units	--
Value Range	--
Default Value	see Table

Sub-Index	04_h
Description	fourth_mapped_object_tpdo1
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	--
Default Value	see Table

Please note that the object records **transmit_pdo_parameter_xxx** and **transmit_pdo_mapping_xxx** can only be written, if the PDO is deactivated (Bit 31 in **cob_id_used_by_pdo_xxx** is set)

1. Transmit-PDO

Index	Comment	Type	Acc.	Default Value
$1800_h\_00_h$	number of entries	UINT8	ro	03_h
$1800_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000181_h$
$1800_h\_02_h$	transmission type	UINT8	rw	FF_h
$1800_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A00_h\_00_h$	number of mapped objects	UINT8	rw	01_h
$1A00_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A00_h\_02_h$	second mapped object	UINT32	rw	00000000_h
$1A00_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1A00_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

2. Transmit-PDO

Index	Comment	Type	Acc.	Default Value
$1801_h\_00_h$	number of entries	UINT8	ro	03_h
$1801_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000281_h$
$1801_h\_02_h$	transmission type	UINT8	rw	FF_h
$1801_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A01_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1A01_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A01_h\_02_h$	second mapped object	UINT32	rw	60610008_h
$1A01_h\_03_h$	third mapped object	UINT32	rw	00000000_h

$1A01_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

3. Transmit-PDO

Index	Comment	Type	Acc.	Default Value
$1802_h\_00_h$	number of entries	UINT8	ro	03_h
$1802_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000381_h$
$1802_h\_02_h$	transmission type	UINT8	rw	FF_h
$1802_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A02_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1A02_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A02_h\_02_h$	second mapped object	UINT32	rw	60640020_h
$1A02_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1A02_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

4. Transmit-PDO

Index	Comment	Type	Acc.	Default Value
$1803_h\_00_h$	number of entries	UINT8	ro	03_h
$1803_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000481_h$
$1803_h\_02_h$	transmission type	UINT8	rw	FF_h
$1803_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A03_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1A03_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A03_h\_02_h$	second mapped object	UINT32	rw	$606C0020_h$
$1A03_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1A03_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

tpdo_1_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2014_h\_00_h$	number of entries	UINT8	ro	02_h
$2014_h\_01_h$	tpdo_1_transmit_mask_low	UINT32	rw	$FFFFFFFF_h$
$2014_h\_02_h$	tpdo_1_transmit_mask_high	UINT32	rw	$FFFFFFFF_h$

tpdo_2_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2015_h\_00_h$	number of entries	UINT8	ro	02_h
$2015_h\_01_h$	tpdo_2_transmit_mask_low	UINT32	rw	$FFFFFFFF_h$
$2015_h\_02_h$	tpdo_2_transmit_mask_high	UINT32	rw	$FFFFFFFF_h$

tpdo_3_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2016_h\_00_h$	number of entries	UINT8	ro	02_h
$2016_h\_01_h$	tpdo_3_transmit_mask_low	UINT32	rw	$FFFFFFFF_h$
$2016_h\_02_h$	tpdo_3_transmit_mask_high	UINT32	rw	$FFFFFFFF_h$

tpdo_4_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2017_h\_00_h$	number of entries	UINT8	ro	02_h
$2017_h\_01_h$	tpdo_4_transmit_mask_low	UINT32	rw	$FFFFFFFF_h$
$2017_h\_02_h$	tpdo_4_transmit_mask_high	UINT32	rw	$FFFFFFFF_h$

1. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1400_h\_00_h$	number of entries	UINT8	ro	02_h
$1400_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000201_h$
$1400_h\_02_h$	transmission type	UINT8	rw	FF_h
$1600_h\_00_h$	number of mapped objects	UINT8	rw	01_h
$1600_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1600_h\_02_h$	second mapped object	UINT32	rw	00000000_h
$1600_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1600_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

2. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1401_h\_00_h$	number of entries	UINT8	ro	02_h
$1401_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000301_h$
$1401_h\_02_h$	transmission type	UINT8	rw	FF_h
$1601_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1601_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1601_h\_02_h$	second mapped object	UINT32	rw	60600008_h
$1601_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1601_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

3. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1402_h\_00_h$	number of entries	UINT8	ro	02_h
$1402_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000401_h$
$1402_h\_02_h$	transmission type	UINT8	rw	FF_h
$1602_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1602_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1602_h\_02_h$	second mapped object	UINT32	rw	$607A0020_h$
$1602_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1602_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

4. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1403_h\_00_h$	number of entries	UINT8	ro	02_h
$1403_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000501_h$
$1403_h\_02_h$	transmission type	UINT8	rw	FF_h
$1603_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1603_h\_01_h$	first mapped object	UINT32	rw	60400010_h

$1603_h\_02_h$	second mapped object	UINT32	rw	$60FF0020_h$
$1603_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1603_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

5.3.3 Activation of PDOs

The following points have to be fulfilled for the activation of a PDO:

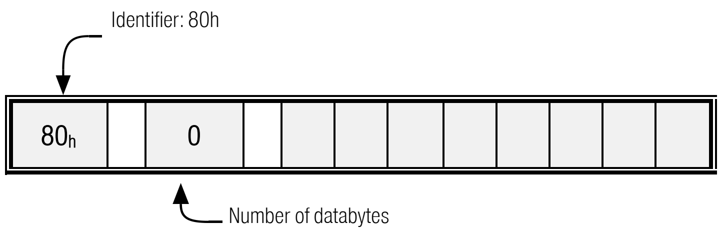
- The object **number_of_mapped_objects** has to be different from zero
- The bit 32 has to be deleted in the object **cob_id_used_for_pdos**
- The communication status of the servo has to be **operational** (see chapter 5.6, Network management)

The following points have to be fullfilled to parametrize a PDO

- The communication status of the servo must not be **operational**

5.4 SYNC-Message

Several devices of a plant can be synchronised with each other. To that purpose one of the devices (in most cases the superimposed control) periodically sends synchronisation messages. All connected servo controllers receive these messages and use them for the treatment of the PDOs (see chapter 5.3).



The identifier on which the servo controller receives SYNC messages is fixed to 080_h. The identifier can be read via the object **cob_id_sync**.

Index	**1005_h**
Name	**cob_id_sync**
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	no
Units	--
Value Range	80000080_h, 00000080_h
Default Value	00000080_h

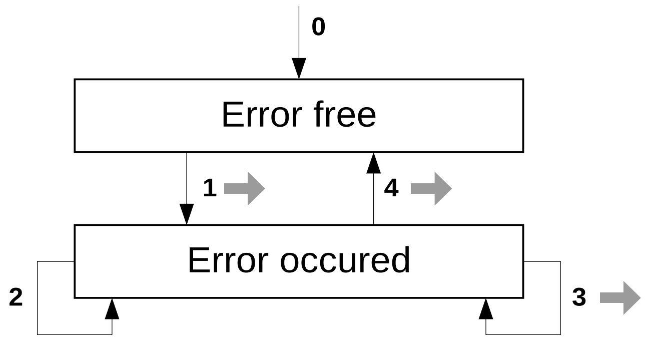
5.5 EMERGENCY-Message

The servo controller monitors the functions of its essential units. The power supply, the power stage, the angle encoder input, and the technology module belong to these units. Besides this the motor (temperature, angle encoder) and the limit switches are constantly controlled. Bad parameters could also result in error messages (division by zero etc.).

If an error occurs, the error number is displayed by the servo controller. If several error messages occur at the same time the message which has the highest priority (the least number) is displayed.

5.5.1 Survey

If an error occurs or an error-reset has been carried out the servo controller sends an EMERGENCY message. The identifier of this message is 80_h plus node number.

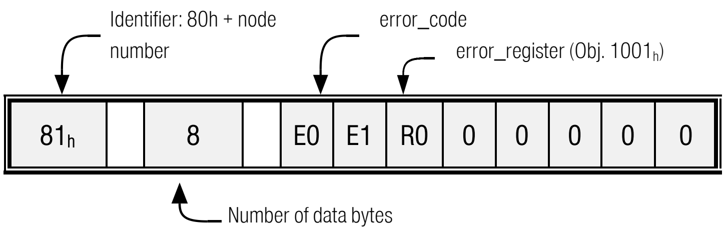


After Reset the state of the servo controller will be *Error free* (if there is an error right from the start, it immediately moves to *Error occured*). The following state transitions are possible:

No.	Cause	Meaning
0	Initialisation completed	
1	Error occurs	No error was present and a new error occurs: An EMERGENCY- Telegram with the error code of the newly occurred error will be sent.
2	Error-reset	An error-reset (see Chap. 7.1.3.1) will be executed, but not all error reasons are removed.
3	Error occurs	An error was still present and yet another error occurs: An EMERGENCY- Telegram with the error code of the newly occurred error will be sent.
4	Error-reset	An error-reset will be executed and all error reasons are removed. An EMERGENCY- Telegram with the error code 0000 will be sent.

5.5.2 Structure of an EMERGENCY message

The EMERGENCY message consists of eight data bytes with the **error code** in the first two bytes.These **error_codes** are described in the following table. There is a further error code (object 1001_h) in the third byte. The other five bytes contain zeros.



The following **error_codes** can occur

error_code (hex)	Anzeige	Bedeutung
0000	–	Error free
6180	E 01 0	Stack overflow
3220	E 02 0	Undervoltage of DC-bus
4310	E 03 x	Overtemperature motor
4210	E 04 0	Overtemperature of the power stage
4280	E 04 1	Overtemperature in the DC-bus
5114	E 05 0	Internal undervoltage supply 1
5115	E 05 1	Internal undervoltage supply 2
5116	E 05 2	Driver voltage fault
5410	E 05 3	Undervoltage dig. I/O
5410	E 05 4	Overcurrent dig. I/O
2320	E 06 x	Short circuit in the power stage
3210	E 07 0	Overvoltage
7380	E 08 0	Angle encoder error resolver
7382	E 08 2	Error of track signals Z0 incremental encoder
7383	E 08 3	Error of track signals Z1 incremental encoder
7384	E 08 4	Error of track signals of digital incremental encoder
7385	E 08 5	Error of Hall signals of incremental encoder
7386	E 08 6	Communication error angle encoder
7387	E 08 7	Signal amplitude: erroneous incremental track
7388	E 08 8	Internal angle encoder error
7389	E 08 9	Angle encoder on X2B is not supported
73A1	E 09 0	Angle encoder parameter set: type item C Series
73A2	E 09 1	Angle encoder parameter set cannot be decoded
73A3	E 09 2	Angle encoder parameter set: unknown version
73A4	E 09 3	Angle encoder parameter set: a damaged data structure
73A5	E 09 7	EEPROM angle encoder is read-only
73A6	E 09 9	EEPROM angle encoder is too small
8A80	E 11 0	Error at start of homing run
8A81	E 11 1	Error during homing run
8A82	E 11 2	Homing: Erroneous index pulse
8A83	E 11 3	Homing: timeout
8A84	E 11 4	Homing: wrong / invalid limit switch
8A85	E 11 5	Homing: I^2t / following error
8A86	E 11 6	Homing: end of the homing distance
8180	E 12 0	CAN bus: Duplicate node number
8120	E 12 1	CAN bus: Communication error: BUS OFF
8181	E 12 2	CAN bus: Communication error: Transmit error
8182	E 12 3	CAN bus: Communication error: Receive error
6185	E 15 0	Division by 0
6186	E 15 1	Range overflow
6181	E 16 0	Errorneous program execution
6182	E 16 1	Illegal interrupt
6187	E 16 2	Initalisation error
6183	E 16 3	Unexpected state
8611	E 17 x	Max. following error exceeded
5280	E 21 1	Error 1 current measurement U
5281	E 21 1	Error 1 current measurement V
5282	E 21 2	Error 2 current measurement U
5283	E 21 3	Error 2 current measurement V
6080	E 25 0	Invalid device type
6081	E 25 1	Device type is not supported
6082	E 25 2	HW revision is not supported
6083	E 25 3	Limited device function

error_code (hex)	Anzeige	Bedeutung
5580	E 26 0	Missing user parameter set
5581	E 26 1	Checksum error
5582	E 26 2	Flash: Error during write operation
5583	E 26 3	Flash: Error during read operation
5584	E 26 4	Flash: Error in internal flash
5585	E 26 5	Missing calibration data
5586	E 26 6	Missing user parameter sets
8611	E 27 0	Following error warning level
FF01	E 28 0	Counter hours of operation is missing
FF02	E 28 1	Counter hours of operation: write error
FF03	E 28 2	Counter hours of operation corrected
FF04	E 28 3	Counter hours of operation converted
6380	E 30 0	Internal conversion error
2312	E 31 0	I^2T – motor
2311	E 31 1	I^2T – servo controller
2313	E 31 2	I^2T – PFC
2314	E 31 3	I^2T – brake chopper
3280	E 32 0	Loading period DC-bus exceeded
3281	E 32 1	Undervoltage for active PFC
3282	E 32 5	Brake chopper overload
3283	E 32 6	Discharging period DC-bus exceeded
3284	E 32 7	Missing power supply for controller enable
3285	E 32 8	Power supply failure at controller enable
3286	E 32 9	Phase failure
8A87	E 33 0	Following error encoder emulation
8780	E 34 0	No synchronization via fieldbus
8781	E 34 1	Synchronization error fieldbus
8480	E 35 0	Overspeed protection linear motor
6320	E 36 x	Parameter has been limited
8612	E 40 x	SW limit switch
8680	E 42 0	Positioning: Drive stopped. Missing following position.
8681	E 42 1	Positioning: Drive stopped. Inversion of the direction is not allowed.
8682	E 42 2	Positioning: Inversion of the direction is not allowed after “Halt”
8081	E 43 0	Limit switch: Negative set point inhibited
8082	E 43 1	Limit switch: Positive set point inhibited
8083	E 43 2	Limit switch: Positioning suppressed
8084	E 45 0	Drivers supply cannot be switched off.
8085	E 45 1	Drivers supply cannot be activated.
8086	E 45 2	Drivers supply has been activated.
7580	E 60 0	Ethernet I
7581	E 61 0	Ethernet II
F080	E 80 0	Overflow current controller - IRQ
F081	E 80 1	Overflow speed controller - IRQ
F082	E 80 2	Overflow positioning controller - IRQ
F083	E 80 3	Overflow interpolator- IRQ
F084	E 81 4	Overflow low-level - IRQ
F085	E 81 5	Overflow MDC- IRQ
5080	E 90 x	Hardware error
6000	E 91 0	Internal initialisation error

5.5.3 Description of Objects

5.5.3.1 Object 1003$_h$: pre_defined_error_field

The **error_codes** of the error messages are recorded in a four-stage error memory. This memory is structured like a shift register so that always the last error is stored in the object **1003$_h$_01$_h$** (**standard_error_field_0)**. By a read access to the object **1003$_h$_00$_h$** (**pre_defined_error_field**) you can find out how many error messages are recorded in the error memory at the moment. The error memory is deleted by writing the value 00$_h$ into the object **1003$_h$_00$_h$** (**pre_defined_error_field**). In addition an **error reset** (see chapter 7.1: state transition 15) has to be executed to reactivate the power stage of the servo controller after an error.

Index	1003_h
Name	pre_defined_error_field
Object Code	ARRAY
No. of Elements	4
Data Type	UINT32

Sub-Index	01_h
Description	standard_error_field_0
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

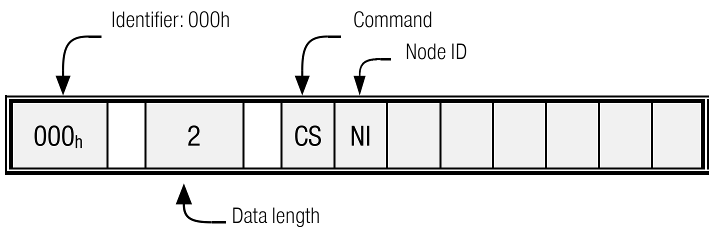
Sub-Index	02_h
Description	standard_error_field_1
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

Sub-Index	03_h
Description	standard_error_field_2
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

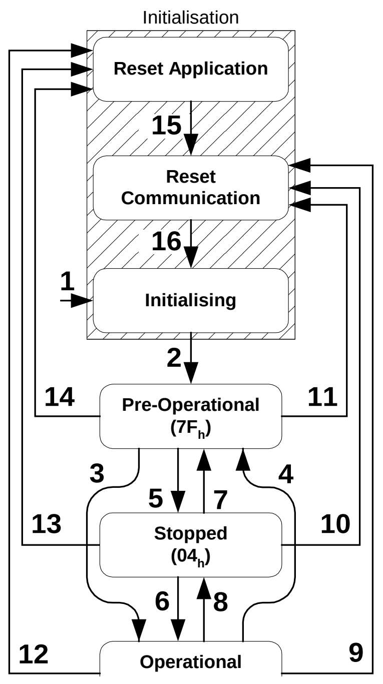
Sub-Index	04_h
Description	standard_error_field_3
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

5.6 Network management (NMT service)

All CANopen devices can be triggered via the network management. A special identifier (000h) is reserved for that.
Commands can be sent to one or all servo controller via this identifier. Each command consists of two bytes. The first byte contains the command code and the second byte the node address of the addressed servo controller. All nodes which are in the network can be addressed via the node address zero simultaneously. So it is possible, for example, to make a reset in all devices at the same time. The servo controller does not quit the NMT-commands. It is only indirectly possible to decide if a reset was successful (e. g. through the Bootup message after a reset).
Structure of the message:



The NMT states of a CANopen device are determined in a state diagram. With the byte **CS** of the NMT message state transitions can be initiated. They are mostly determined by the target state.



	Meaning	CS	Target state	
2	Bootup	–	Pre-Operational	$7F_h$
3	Start Remote Node	01_h	Operational	05_h
4	Enter Pre-Operational	80_h	Pre-Operational	$7F_h$
5	Stop Remote Node	02_h	Stopped	04_h
6	Start Remote Node	01_h	Operational	05_h
7	Enter Pre-Operational	80_h	Pre-Operational	$7F_h$
8	Stop Remote Node	02_h	Stopped	04_h
9	Reset Communication	82_h	Reset Communication *1)	
10	Reset Communication	82_h	Reset Communication *1)	
11	Reset Communication	82_h	Reset Communication *1)	
12	Reset Application	81_h	Reset Application *1)	

13	Reset Application	81_h	Reset Application *1)
14	Reset Application	81_h	Reset Application *1)

*1) The final target state is **Pre-Operational** ($7F_h$), because the transitions 15, 16 and 2 are done automatically by the controller.

Figure 5.4: NMT-State machine

With the following commands the NMT state can be changed:

CS	Meaning	Transition	Target state
01_h	Start Remote Node	3, 6	Operational (05_h)
02_h	Stop Remote Node	5, 8	Stopped (04_h)
80_h	Enter Pre-Operational	4, 7	Pre-Operational ($7F_h$)
81_h	Reset Application	12, 13, 14	Reset Application *1)
82_h	Reset Communication	9, 10, 11	Reset Communication *1)

All remaining transitions will be executed automatically by the servo controller, e.g. if initialising has been finished.

The parameter **NI** contains the node number of the servo controller or zero, if all nodes within the network will be addressed. Depending on the NMT state several communication objects can not be used. For example it is necessary to set the NMT state to **operational** to enable sending and receiving PDOs.

Name	Meaning	SDO	PDO	NMT
Reset Application	No communication. All CAN objects are set to their reset values (application parameter set).	-	-	-
Reset Communication	No communication. The CAN controller will be re-initialised.	-	-	-
Initialising	State after Hardware Reset. Reset of the CAN node, sending of the Bootup message	-	-	-
Pre-Operational	Communication via SDOs possible. PDOs inactive (No sending / receiving)	X	-	X
Operational	Communication via SDOs possible. PDOs active (sending / receiving)	X	X	X
Stopped	No communication except heartbeat + NMT	-	-	X

NMT telegrams should not be sent in a burst (back-to-back).
The double cycle time of the position controller must be present between two back-to-back NMT messages on the bus (also for several nodes!), so that the controller can correctly proccess the

NMT messages.

The NMT command „Reset Application“ will be delayed while a save process to the flash (save_all_parameters) is running, as otherwise the save process may be uncompleted leading to a faulty parameter set.

The delay can last several seconds.

The communication status has to be set to **operational** to allow the servo to send and receive PDOs

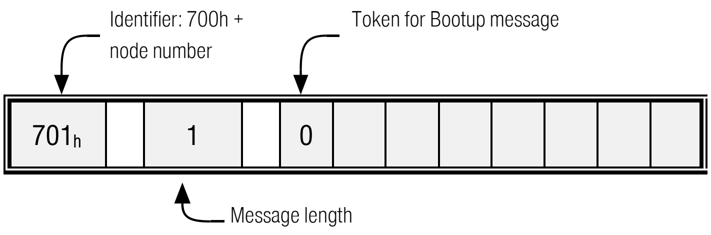
5.7 Bootup (Error Control Protocol)

5.7.1 Survey

After power-on or after reset, the servo positioning controller reports through a Bootup message that the initialising has been finished. The servo is afterwards in the NMT state **preoperational** (see Chapter 5.6, Network management)

5.7.2 Structure of the Bootup message

The Bootup message is nearly identical with the following Heartbeat message. Only instead of the NMT state zero will be sent.



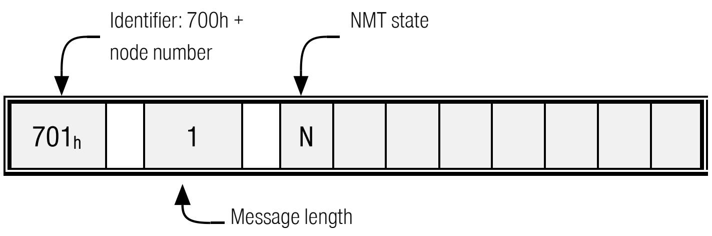
5.8 Heartbeat (Error Control Protocol)

5.8.1 Survey

To monitor the communication between slave (servo) and master the Heartbeat protocol is implemented. The servo cyclically sends a message to the master. The master can check if it cyclically receives the Heartbeat and initiate appropriate reactions if not. As the Heartbeat telegram as well as the Nodeguarding telegram use identifier **700_h + node number** it is not possible to use both simultaneously. If both protocols are active simultaneously only Heartbeat will be available.

5.8.2 Structure of the Heartbeat message

The Heartbeat message will be sent with identifier **700$_h$ + node number**. It is only composed of 1 Byte, containing the NMT state of the servo (see Chapter 5.6, Network management).



N	Description
04$_h$	Stopped
05$_h$	Operational
7F$_h$	Pre-Operational

5.8.3 Objects

5.8.3.1 Object 1017$_h$: producer_heartbeat_time

The time between two Heartbeat messages can be determined by the object **producer_heartbeat_time**.

Index	1017$_h$
Name	producer_heartbeat_time
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	no
Units	ms
Value Range	0...65535
Default Value	0

The **producer_heartbeat_time** can be saved in the parameter set. If the servo starts with a **producer_heart-beat_time** unequal zero, the Bootup message is seen as the first heatbeat.

The controller can be used only as so-called Heartbeat Producer. Thus the object 1016$_h$ (consumer_heartbeat_time) is implemented only because of compatibility reasons and always returns 0.

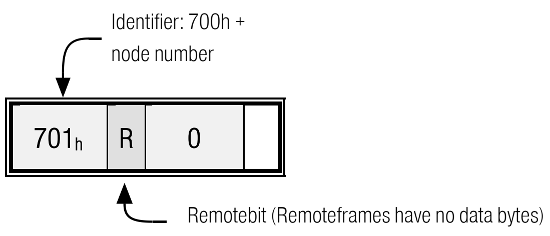
5.9 Nodeguarding (Error Control Protocol)

5.9.1 Survey

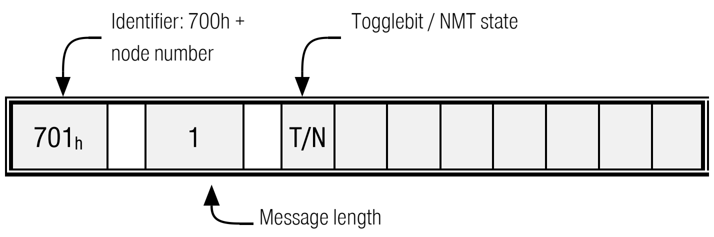
Nodeguarding can be used as well as the Heartbeat protocol to monitor the communication between Slave (servo controller) and Master. In opposite to the Heartbeat protocol it is possible that master and slave monitors each other:
The master asks the slave cyclically for its NMT state. In each answer of the slave one special bit will be toggled. If the slave doesn't answer or the toggle bit doesn't change, the master can react appropriate.
In the same way the slave monitors the master: If no Nodeguarding request arrives within a definite period of time, error 12-4 will be raised.
As the Nodeguarding telegram as well as the Heartbeat telegram use identifier **700_h + node number** it is not possible to use both simultaneously. If both protocols are active simultaneously only Heartbeat will be available. Nodeguarding will be available as of firmware 3.5.x.1.1.

5.9.2 Structure of the Nodeguarding message

The request of the master has to be sent by a Remoteframe with Identifier **700_h + node number.** A Remoteframe is a special CAN telegram where the remote bit is set. Remoteframes in principle have no data bytes.


Identifier: 700h + node number
701h
R
0
Remotebit (Remoteframes have no data bytes)

The response of the slave is similar to the Heartbeat message. It is also composed of just 1 Byte, containing the toggle bit and the NMT state of the servo (see Chapter 5.6, Network management).



The first data byte (**T/N**) is composed as follows:

Bit	Value	Name	Meaning
7	80_h	toggle_bit	Changes with every response
0...6	$7F_h$	nmt_state	04_h Stopped 05_h Operational $7F_h$ Pre- Operational

The guarding time (monitoring time) can be configured. The slave starts monitoring with the first received Remoteframe from the master. From this time all Remoteframes have to arrive before the expiration of the guard time, as otherwise error 12-4 will occur.

The toggle bit will be deleted by the NMT command **Reset Communication**. Therefore it is reset in the first response of the servo controller.

5.9.3 Description of Objects

5.9.3.1 Object $100C_h$: guard_time

For activating the Nodeguarding the maximum time between two remote requests of the master will be configured. As this time will be calculated as product of **guard_time** ($100C_h$) and **life_time_factor** ($100D_h$), it is recommended to set the **life_time_factor** to 1 and write the **guard_time** directly in milliseconds.

Index	$100C_h$
Name	guard_time
Object Code	VAR
Data Type	UINT16

As of Firmware 3.5.x.1.1

Access	rw
PDO Mapping	no
Units	ms
Value Range	0...65535
Default Value	0

5.9.3.2 Objekt $100D_h$: life_time_factor

The **life_time_factor** should be set to 1 to write the **guard_time** directly in milliseconds.

Index	$100D_h$
Name	life_time_factor
Object Code	VAR
Data Type	UINT8

As of Firmware 3.5.x.1.1

Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

5.10 Table of identifiers

The following table gives a survey of the used identifiers.

Object-Type	Identifier (hexadecimal)	Remark
SDO (Host to Servo)	600_h+node id	
SDO (Servo to Host)	580_h + node id	
TPDO1	181_h	Standard values. Can be changed on demand.
TPDO2	281_h	
TPDO3	381_h	
TPDO4	481_h	
RPDO1	201_h	
RPDO2	301_h	
RPDO3	401_h	
RPDO4	501_h	
SYNC	080_h	
EMCY	080_h + node number	
HEARTBEAT	700_h + node number	
NODEGUARDING	700_h + node number	
BOOTUP	700_h + node number	
NMT	000_h	

6 Adjustment of parameters

Before a certain task (e.g. torque or velocity control) can be managed by the servo controller several parameters have to be adjusted according to the used motor and the specific application. Therefore the chronological order suggested by the following chapters should be abided.

After explaining the parameter adjustment the device control and the several modes of operation will be presented.

An "A" will be displayed by the servo controller if it is not properly commissioned yet. If the complete adjustment of parameter should be done via the CAN bus, object **6510$_h$_C0$_h$** have to be configured to suppress the "A". (see chapter 6.17.1.12 Object 6510h_C0h: commissioning_state).

Beside the parameters, described in details here, there are more parameters in the controller's object dictionary, which must be implemented according to the CANopen specification. They normally contain no information, which can be meaningful used for configuring an application with the . If necessary the specification of such objects can be found in [1] and [2] (see page 13).

6.1 Load and save set of parameters

6.1.1 Survey

The servo controller has three parameter sets:

- Current parameter set

  This parameter set is in the transient memory (RAM) of the servo controller. It can be read and written optionally via the parameter set-up program ™ or via the CAN bus. When the servo controller is switched on the **application parameter set** is copied into the **current parameter set**.

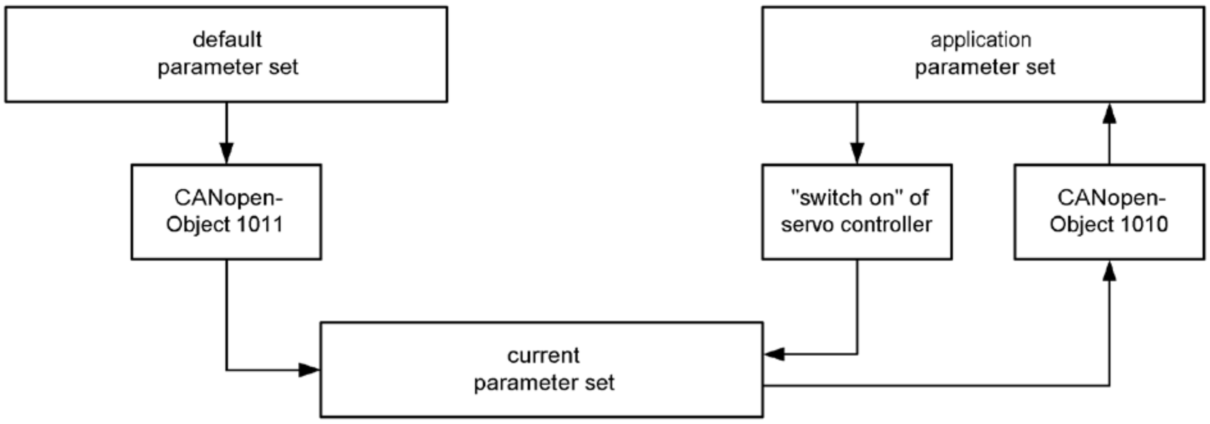
- Default parameter set

  This is the unmodifiable **default parameter set** of the servo controller given by the manufacturer. The **default parameter set** can be copied to the current parameter set through a write process into the CANopen object **1011$_h$_01$_h$** (**restore_all_default_parameters**). This copy process is only possible while the output power stage is switched off.

- Application parameter set

  The **current parameter set** can be saved into the non-transient flash memory. This saving process is enabled by a write access to the CANopen object **1010_h_01_h** (**save_all_parameters**). When the servo controller is switched on the **application parameter set** is copied to the **current parameter set**.

The following graphic illustrates the coherence between the respective parameter sets.



Two different methods are possible concerning the parameter set administration:

1. The parameter set is made up with the parameter set-up program ™ and also transferred to the single servo controller by the parameter set-up program ™. With this method only those objects which can be accessed via CANopen exclusively have to be adjusted via the CAN bus.
   This method has the disadvantage that the parameter set-up software is needed for every start of a new machine or in case of repair (exchange of servo controller). Therefore this method only makes sense for individual units.

2. This method is based on the fact that most application specific parameter sets only vary in few parameters from the **default parameter set**. Thus it is possible to set up the **current parameter set** after every reset via the CAN bus. To that purpose the **default parameter set** is first loaded by the superimposed control (call of the CANopen object **1011_h_01_h (restore_all_default_parameters)**). Afterwards only those objects are transferred which vary. The complete process only lasts about 0,3 seconds per drive. It is advantageous that this method also works for non-configured servo controllers and the parameter set-up software ™ is not necessary for this. It is urgently recommended to use method 2.

Before switching on the power stage for the first time, assure that the servo controller contains the desired parameters.

An incorrect parameter set-up may cause uncontrolled behaviour of the motor and thereby personal or material damage may occur.

6.1.2 Description of Objects

6.1.2.1 Object 1011_h: restore_default_parameters

Index	1011_h
Name	restore_parameters
Object Code	ARRAY
No. of Elements	1
Data Type	UINT32

Sub-Index	01_h
Description	restore_all_default_parameters
Access	rw
PDO Mapping	no
Units	--
Value Range	$64616F6C_h$ („load“)
Default Value	1 (read access)

Through the object 1011_h_01_h (**restore_all_default_parameters**) it is possible to put the **current parameter set** into a defined state. For that purpose the **default parameter set** is copied to the **current parameter set**. The copy process is enabled by a write access to this object and the string "load" is to be passed as data set in hexadecimal form.
This command is only executed while the output power stage is deactivated. Otherwise the SDO error "The controller is in wrong operation mode for this kind of operation" is generated.
The parameters for the CAN communication (node number, baudrate and mode) and several parameters for configuring the encoder inputs (partly needing a reset for becoming valid) remain unchanged.

6.1.2.2 Object 1010_h: store_parameters

Index	1010_h
Name	store_parameters
Object Code	ARRAY
No. of Elements	1
Data Type	UINT32

Sub-Index	01_h
Description	save_all_parameters
Access	rw
PDO Mapping	no
Units	--
Value Range	65766173_h („save“)
Default Value	1

To store the **default parameter set** as **application parameter set**, the object **1010_h_01_h** (**save_all_parameters**) must be used additionally.

The default behaviour for this object being written by an SDO access is an immediate SDO response. So the response does not mirror the end of the “saving parameters” process. This behaviour can be changed by object **6510_h_$F0_h$** (**compatibility_control**).

6.2 Compatibility settings

6.2.1 Survey

In order to be compatible to earlier CANopen implementations (also to other device series for example) on the one hand and to achieve changes and patches compared to the DSP402 and the DS301 on the other hand, the object **compatibility_control** has been introduced. In the default parameter set this object returns 0, i.e. compatibility to earlier versions. For new applications we recommend to set the defined bits, to facilitate as high as possible conformance with the mentioned standards.

6.2.2 Description of Objects

6.2.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
6510$_h$_F0$_h$	VAR	compatibility_control	UINT16	rw

6.2.2.2 Object 6510$_h$_F0$_h$: compatibility_control

Sub-Index	F0$_h$
Description	compatibility_control
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0...1FF$_h$, see table
Default Value	0

As of Firmware 3.2.0.1.1

Bit	Value	Name	
0	0001_h	homing_method_scheme*	
1	0002_h	reserved	
2	0004_h	homing_method_scheme	
3	0008_h	reserved	
4	0010_h	response_after_save	As of Firmware 3.4.0.1.1
5	0020_h	reserved	As of Firmware 3.5.0.1.1
6	0040_h	homing_to_zero	As of Firmware 3.5.0.1.1
7	0080_h	device_control	As of Firmware 3.5.0.1.1
8	0100_h	reserved	As of Firmware 3.5.0.1.1

Bit 0	homing_method_scheme*
	This bit is compatible with bit 2. If this bit will be set, bit 2 will be set as well and vice versa.
Bit 1	reserved
	This bit is reserved. It must not be set.
Bit 2	homing_method_scheme
	If this bit is set the numeration of the homing methods 32...35 corresponds to the DSP 402. If it is reset the numeration is compatible to previous implementations. (See also Chap. 8.2.3) If this bit will be set, bit 0 will be set as well and vice versa.
Bit 3	reserved
	This bit is reserved. It must not be set.
Bit 4	response_after_save
	If this bit is set, the sdo response to save_all_parameters will not be sent before the saving has been completed, which can take several seconds. This may lead to a timeout of the master. If this bit is reset, the response will be sent immediately, without waiting for the end of the saving process.

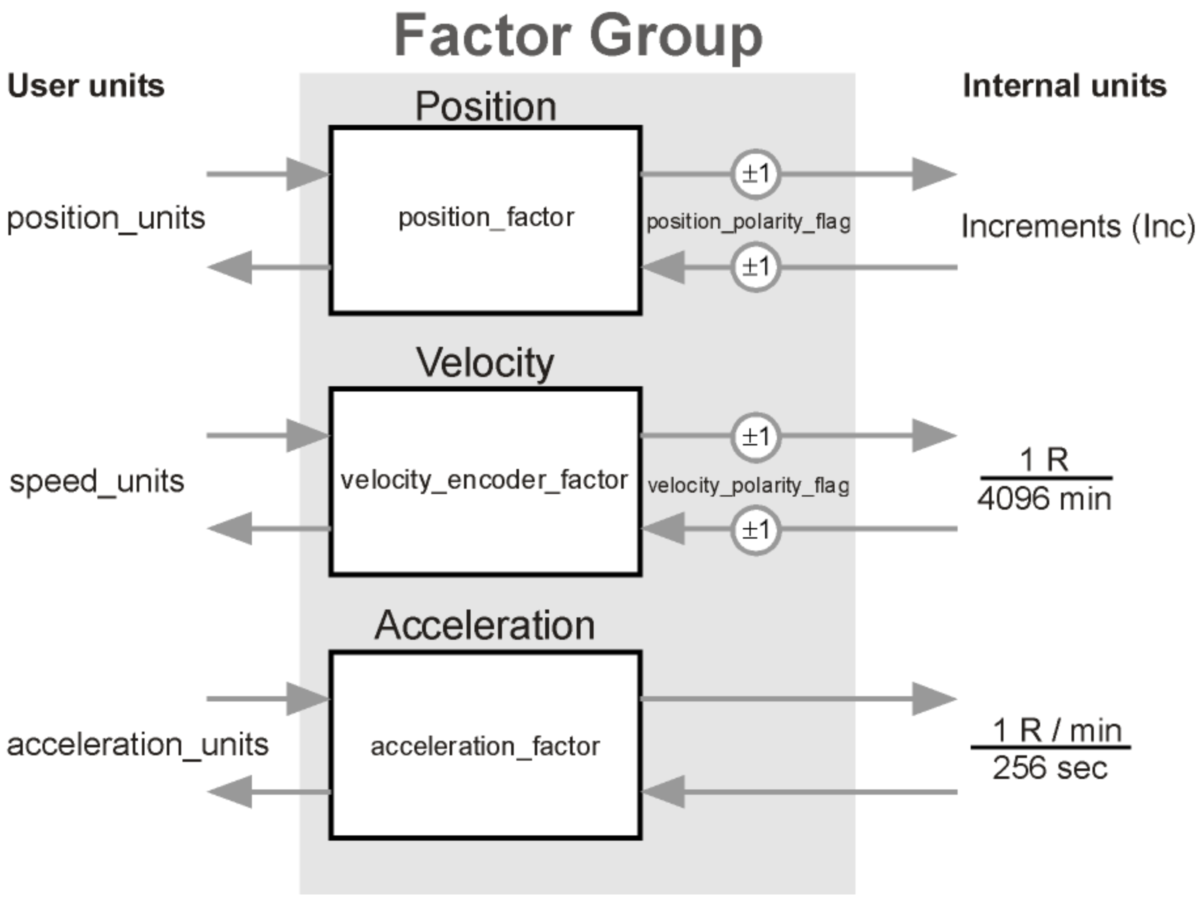
Bit 5	reserved	
		This bit is reserved. It must not be set.
Bit 6	homing_to_zero	
		A CANopen homing operation consists of two phases (Search for switch an Search for zero) until now: The drive does not move to the zero position (which can be different to the found reference position due to the **homing_offset** for example). If this bit is set, this behaviour will be changed and the servo controller always moves to the zero position.
Bit 7	device_control	
		If this bit is set, bit 4 of the **statusword** (**voltage_enabled**) complies to DSP 402 v2.0 Furthermore the state **FAULT_REACTION_ ACTIVE** is distinguishable from the state **FAULT**. See also Chapter 7
Bit 8	reserved	
		This bit is reserved. It must not be set.

6.3 Conversion factors (Factor Group)

6.3.1 Survey

Servo controllers will be used in a huge number of applications: As direct drive, with gear or for linear drives. To allow an easy parametrization for all kinds of applications, the servo controller can be configured in such a way that all values like the demand velocity refer to the driven side of the plant. The necessary calculation is done by the servo controller.

Consequently it is possible to enter values directly in e.g. millimetre per second if a linear drive is used. The conversion is done by the servo controller using the Factor Group. For each physical value (position, velocity and acceleration) exists a specific conversion factor to adapt the unit to the own application. In general the user specific units defined by the Factor Group are called **position_units**, **speed_units** and **acceleration_units**. The following Figure shows the function of the Factor Group:



Principally all parameters will be stored in its internal units and converted while reading or writing a parameter.

Therefore the Factor Group should be adjusted once before commissioning the servo controller and not to be changed during parametrization.

The default setting of the Factor Group is as follows:

Value	Name	Unit	Remark
Length	position_units	**Increments**	65536 Increments per revolution
Velocity	speed_units	**min^{-1}**	Revolution per minute (RPM)
Acceleration	acceleration_units	**min^{-1}/s**	Increase of velocity per second (RPM / s)

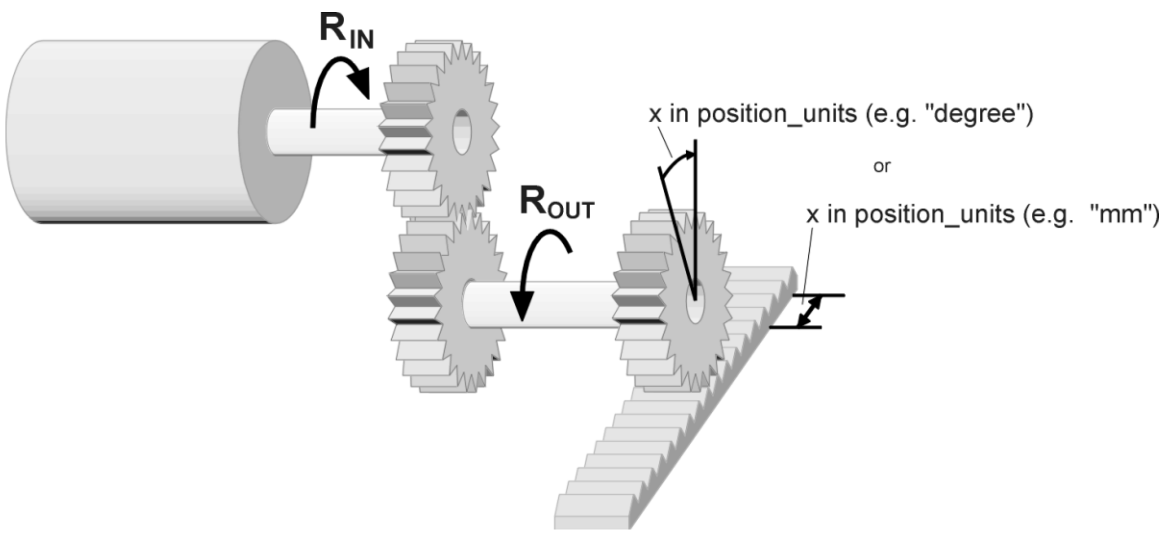
6.3.2 Description of Objects

6.3.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
6093_h	ARRAY	position_factor	UINT32	rw
6094_h	ARRAY	velocity_encoder_factor	UINT32	rw
6097_h	ARRAY	acceleration_factor	UINT32	rw
$607E_h$	VAR	polarity	UINT8	rw

6.3.2.2 Object 6093_h: position_factor

The object **position_factor** converts all values of length of the application from **position_units** into the internal unit **increments** (65536 Increments equals 1 Revolution). It consists of numerator and divisor:



R_{IN}
R_{OUT}
x in position_units (e.g. "degree")
or
x in position_units (e.g. "mm")

Figure 6.5: Survey: Factor Group

Index	6093$_h$
Name	**position_factor**
Object Code	ARRAY
No. of Elements	2
Data Type	UINT32

Sub-Index	01$_h$
Description	**numerator**
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	1

Sub-Index	02$_h$
Description	**divisor**
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	1

To calculate the **position_factor** the following values are necessary:

gear_ratio	Ratio between revolutions on the driving side (R_{IN}) and revolutions on the driven side (R_{OUT}).
feed_constant	Ratio between revolutions on the driven side (R_{OUT}) and equivalent motion in **position_units** (e.g. 1 rev = 360°)

The calculation of the **position_factor** is done with the following equation:

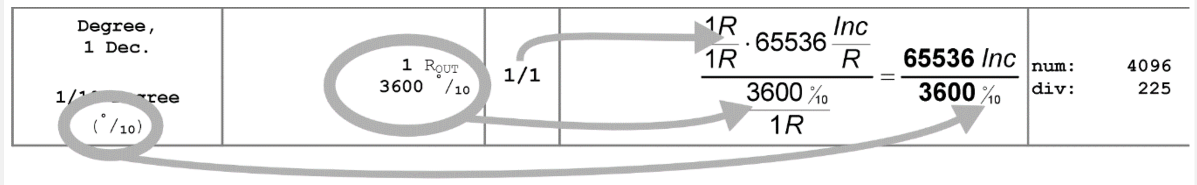
$$\text{position\_factor} = \frac{\text{numerator}}{\text{divisor}} = \frac{\text{gear\_ratio} \cdot 65536}{\text{feed\_constant}}$$

Numerator and divisor of the **position_factor** has to be entered separately. Therefore it may be necessary to extend the fraction to generate integers.

The **position_factor** must not exeed 2^{24}.

EXAMPLE

At first the desired unit (column 1) and the number of decimals (dec) must be specified, as well as the gear ratio and if necessary the feed constant of the application must be determined. This feed constant will be then represented in the desired position units (column 2). Finally all the values can be inserted in the formula and the fraction can be calculated.


Degree,
1 Dec.
1/10 Degree
(°/10)
1 R_OUT
3600 °/10
1/1
$\frac{\frac{1R}{1R}\cdot 65536\frac{Inc}{R}}{\frac{3600\,{}^{\circ}\!/_{10}}{1R}} = \frac{65536\,Inc}{3600\,{}^{\circ}\!/_{10}}$
num: 4096
div: 225

1. Desired unit on the driven side (*position_units*)
2. *feed_constant*: How many *position_units* are 1 Revolution (R_{OUT})
3. Gear_ratio: R_{IN} per R_{OUT}
4. Calculate equation

1.	2.	3.	4.	RESULT Shortened
Increments, 0 Dec. Inc	1 R_{OUT} = 65536 Inc	1/1	$\frac{\frac{1R}{1R}\cdot 65536\frac{Inc}{R}}{\frac{65536\,Inc}{1R}} = \frac{\mathbf{1}\,Inc}{\mathbf{1}\,Inc}$	num: 1 div: 1
Degree, 1 Dec. 1/10 Degree (°/10)	1 R_{OUT} = 3600 °/10	1/1	$\frac{\frac{1R}{1R}\cdot 65536\frac{Inc}{R}}{\frac{3600\,{}^{\circ}\!/_{10}}{1R}} = \frac{\mathbf{65536}\,Inc}{\mathbf{3600}\,{}^{\circ}\!/_{10}}$	num: 4096 div: 225
Revolutions, 2 Dec. 1/100 R (R/100)	1 R_{OUT} = 100 R/100	1/1	$\frac{\frac{1R}{1R}\cdot 65536\frac{Inc}{R}}{\frac{100\,{}^{R}\!/_{100}}{1R}} = \frac{\mathbf{65536}\,Inc}{\mathbf{100}\,{}^{R}\!/_{100}}$	num: 16384 div: 25
		2/3	$\frac{\frac{2R}{3R}\cdot 65536\frac{Inc}{R}}{\frac{100\,{}^{R}\!/_{100}}{1R}} = \frac{\mathbf{131072}\,Inc}{\mathbf{300}\,{}^{R}\!/_{100}}$	num: 32768 div: 75
mm, 1 Dec. 1/10 mm (mm/10)	63.15 mm/R ⇒ 1 R_{OUT} = 631.5 mm/10	4/5	$\frac{\frac{4R}{5R}\cdot 65536\frac{Inc}{R}}{\frac{631.5\,{}^{mm}\!/_{10}}{1R}} = \frac{\mathbf{2621440}\,Inc}{\mathbf{31575}\,{}^{mm}\!/_{10}}$	num: 524288 div: 6315

6.3.2.3 Object 6094_h: velocity_encoder_factor

The object **velocity_encoder_factor** converts all speed values of the application from **speed_units** into the internal unit **revolutions per 4096 minutes**. It consists of numerator and divisor:

Index	6094_h
Name	velocity_encoder_factor
Object Code	ARRAY
No. of Elements	2
Data Type	UINT32

Sub-Index	01_h
Description	numerator
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	1000_h

Sub-Index	02_h
Description	divisor
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	1

In principle the calculation of the **velocity_encoder_factor** is composed of two parts: A conversion factor from internal units of length into **position_units** and a conversion factor from internal time units into user defined time units (e.g. from seconds to minutes). The first part equals the calculation of the **position_factor**. For the second part another factor is necessary for the calculation:

time_factor_v	Ratio between internal and user defined time units. (z.B. 1 *min* = $^1/_{4096}$ *4096 min*)
gear_ratio	Ratio between revolutions on the driving side (R_{IN}) and revolutions on the driven side (R_{OUT}).
feed_constant	Ratio between revolutions on the driven side (R_{OUT}) and equivalent

motion in position_units (e.g. 1 R = 360°)

The calculation of the **velocity_encoder_factor** is done with the following equation:

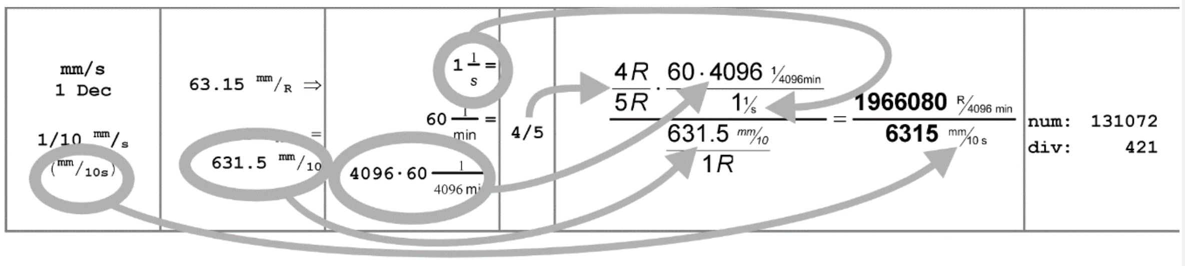
$$\textbf{velocity\_encoder\_factor} = \frac{\textbf{numerator}}{\textbf{divisor}} = \frac{\textbf{gear\_ratio} \cdot \textbf{time\_factor\_v}}{\textbf{feed\_constant}}$$

Numerator and divisor of the **velocity_encoder_factor** has to be entered separately. Therefore it may be necessary to extend the fraction to generate integers:

EXAMPLE

At first the desired unit (column 1) and the number of decimals (dec) must be specified, as well as the gear ratio and if necessary the feed constant of the application must be determined. This feed constant will be then represented in the desired position units (column 2). Subsequently the desired time unit has to be converted into the servo's time unit (column 3).

Finally all the values can be inserted in the formula and the fraction can be calculated.



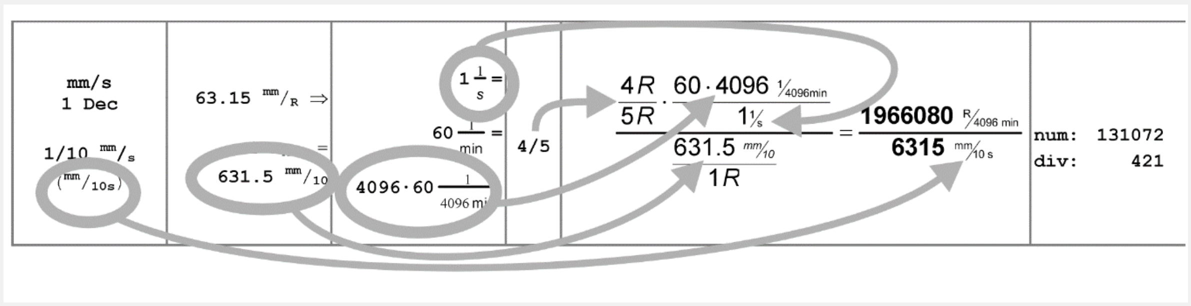
1. Desired unit on the driven side (*position_units*)
2. *feed_constant*: How many *position_units* are 1 Revolution (R_{OUT})
3. *time_factor_v*: Convert time_unit to internal unit
4. Gear_ratio: R_{IN} per R_{OUT}
5. Calculate equation

1.	2.	3.	4.	5.	RESULT Shortened
R/min 0 Dec. $^{R}/_{min}$	1 R_{OUT} = 1 R_{OUT}	$1\ \frac{1}{min}$ = $4096\ \frac{1}{4096\ min}$	1/1	$\dfrac{\frac{1R}{1R} \cdot \frac{4096\ ^{1}/_{4096min}}{1\ ^{1}/_{min}}}{\frac{1R}{1R}} = \dfrac{\mathbf{4096}\ ^{R}/_{4096min}}{\mathbf{1}\ ^{R}/_{min}}$	num: 4096 div: 1
R/min 2 Dec. 1/100 $^{R}/_{min}$ ($^{R}/_{100\ min}$)	1 R_{OUT} = 100 $^{R}/_{100}$	$1\ \frac{1}{min}$ = $4096\ \frac{1}{4096\ min}$	2/3	$\dfrac{\frac{2R}{3R} \cdot \frac{4096\ ^{1}/_{4096min}}{1\ ^{1}/_{min}}}{\frac{100\ ^{R}/_{100}}{1R}} = \dfrac{\mathbf{8192}\ ^{R}/_{4096min}}{\mathbf{300}\ ^{R}/_{100min}}$	num: 2048 div: 75
°/s 1 Dec.	1 R_{OUT} = 3600 $^{\circ}/_{10}$	$1\ \frac{1}{s}$ =	1/1	$\dfrac{\frac{1R}{1R} \cdot \frac{60 \cdot 4096\ ^{1}/_{4096min}}{1\ ^{1}/_{min}}}{3600\ ^{\circ}/_{10}} = \dfrac{\mathbf{245760}\ ^{R}/_{4096min}}{\mathbf{3600}\ ^{\circ}/_{10s}}$	num: 1024

EXAMPLE

At first the desired unit (column 1) and the number of decimals (dec) must be specified, as well as the gear ratio and if necessary the feed constant of the application must be determined. This feed constant will be then represented in the desired position units (column 2). Subsequently the desired time unit has to be converted into the servo's time unit (column 3).

Finally all the values can be inserted in the formula and the fraction can be calculated.



mm/s 1 Dec 1/10 $^{mm}/_{s}$ ($^{mm}/_{10s}$)	63.15 $^{mm}/_{R}$ ⇒ 631.5 $^{mm}/_{10}$	$1\frac{1}{s}=$ $60\frac{1}{min}=$ $4096\cdot 60\frac{1}{4096 min}$	4/5	$\dfrac{\dfrac{4R}{5R}\cdot\dfrac{60\cdot 4096\ ^{1}/_{4096min}}{1^{1}/_{s}}}{\dfrac{631.5\ ^{mm}/_{10}}{1R}}=\dfrac{1966080\ ^{R}/_{4096\ min}}{6315\ ^{mm}/_{10\ s}}$	num: 131072 div: 421

1. Desired unit on the driven side (*position_units*)
2. *feed_constant*: How many *position_units* are 1 Revolution (R_{OUT})
3. *time_factor_v*: Convert time_unit to internal unit
4. Gear_ratio: R_{IN} per R_{OUT}
5. Calculate equation

1.	2.	3.	4.	5.	RESULT Shortened
1/10 $^{\circ}/_{s}$ ($^{\circ}/_{10s}$)		$60\frac{1}{min}=60\cdot 4096\frac{1}{4096\ min}$			div: 15
mm/s 1 Dec. 1/10 $^{mm}/_{s}$ ($^{mm}/_{10s}$)	63.15 $^{mm}/_{R}$ ⇒ 1 R_{OUT} = 631.5 $^{mm}/_{10}$	$1\frac{1}{s}=$ $60\frac{1}{min}=60\cdot 4096\frac{1}{4096\ min}$	4/5	$\dfrac{\dfrac{4R}{5R}\cdot\dfrac{60\cdot 4096\ ^{1}/_{4096min}}{1^{1}/_{s}}}{\dfrac{631.5\ ^{mm}/_{10}}{1R}}=\dfrac{1966080\ ^{R}/_{4096\ min}}{6315\ ^{mm}/_{10\ s}}$	num: 131072 div: 421

6.3.2.4 Object 6097_h: acceleration_factor

The object **acceleration_factor** converts all acceleration values of the application from **acceleration_units** into the internal unit **RPM per 256 minutes.** It consists of numerator and divisor:

Index	6097_h
Name	acceleration_factor
Object Code	ARRAY
No. of Elements	2
Data Type	UINT32

Sub-Index	01_h
Description	numerator
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	100_h

Sub-Index	02_h
Description	divisor
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	1

The calculation of the **acceleration_factor** is also composed of two parts: A conversion factor from internal units of length into **position_units** and a conversion factor from internal time units squared into user defined time units squared (e.g. from seconds^2 to minutes^2). The first part equals the calculation of the **position_factor.** For the second part another factor is necessary for the calculation:

time_factor_a	Ratio between internal time units squared and user defined time units squared (z.B. $1\ min^2 = 1\ min \cdot 1\ min = 60\ s \cdot 1\ min = {}^{60}/_{256}\ 256\ min \cdot s$)
gear_ratio	Ratio between revolutions on the driving side (R_{IN}) and revolutions on the driven side (R_{OUT}).
feed_constant	Ratio between revolutions on the driven side (R_{OUT}) and equivalent

motion in position_units (e.g. 1 R = 360°)

The calculation of the **acceleration_factor** is done with the following equation:

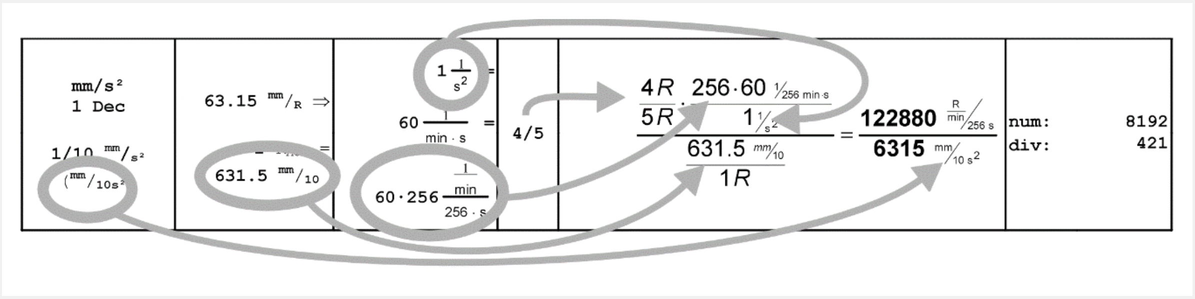
$$\textbf{acceleration\_factor} = \frac{\textbf{numerator}}{\textbf{divisor}} = \frac{\textbf{gear\_ratio} \cdot \textbf{time\_factor\_a}}{\textbf{feed\_constant}}$$

Numerator and divisor of the **acceleration_factor** has to be entered separately. Therefore it may be necessary to extend the fraction to generate integers.

EXAMPLE

At first the desired unit (column 1) and the number of decimals (dec) must be specified, as well as the gear ratio and if necessary the feed constant of the application must be determined. This feed constant will be then represented in the desired position units (column 2). Subsequently the desired time unit squared has to be converted into the servo's time unit squared (column 3).

Finally all the values can be inserted in the formula and the fraction can be calculated.



1. Desired unit on the driven side (*acceleration_units*)
2. *feed_constant*: How many *position_units* are 1 Revolution (R_{OUT})
3. *time_factor_v*: Convert (time_unit)^2 to internal unit^2
4. Gear_ratio: R_{IN} per R_{OUT}
5. Calculate equation

1.	2.	3.	4.	5.	RESULT Shortened
R/min/s 0 Dec. $^{R}/_{\text{min s}}$	1 R_{OUT} = 1 R_{OUT}	$1\,\frac{1}{\text{min}\cdot\text{s}} =$ $256\,\frac{\frac{1}{\text{min}}}{256\cdot\text{s}}$	1/1	$\frac{\frac{1R}{1R}\cdot\frac{256\,^{1}/_{256\,min\,s}}{1\,^{1}/_{min\cdot s}}}{\frac{1R}{1R}} = \frac{\mathbf{256}\,^{\frac{R}{min}}/_{256\cdot s}}{\mathbf{1}\,^{\frac{R}{min}}/_{s}}$	num: 256 div: 1
°/s² 1 Dec. 1/10 $^{\circ}/_{s^2}$ ($^{\circ}/_{10s^2}$)	1 R_{OUT} = 3600 $^{\circ}/_{10}$	$1\,\frac{1}{s^2} =$ $60\,\frac{1}{\text{min}\cdot\text{s}} =$ $60\cdot256\,\frac{\frac{1}{\text{min}}}{256\cdot\text{s}}$	1/1	$\frac{\frac{1R}{1R}\cdot\frac{60\cdot256\,^{1}/_{256\,min\cdot s}}{1\,^{1}/_{s^2}}}{\frac{3600\,^{\circ}/_{10}}{1R}} = \frac{\mathbf{15360}\,^{\frac{R}{min}}/_{256\cdot s}}{\mathbf{3600}\,^{\circ}/_{10\,s^2}}$	num: 64 div: 15
R/min² 2 Dec. 1/100 $^{R}/_{min^2}$ ($^{R}/_{100\,min^2}$)	1 R_{OUT} = 100 $^{R}/_{100}$	$1\,\frac{1}{\text{min}^2} =$ $\frac{1}{60}\,\frac{\frac{1}{\text{min}}}{\text{s}} =$ $\frac{256}{60}\,\frac{\frac{1}{\text{min}}}{256\cdot\text{s}}$	2/3	$\frac{\frac{2R}{3R}\cdot\frac{256\,^{1}/_{256\,min\cdot s}}{60\,^{1}/_{min^2}}}{\frac{100\,^{R}/_{100}}{1R}} = \frac{\mathbf{512}\,^{\frac{R}{min}}/_{256\,s}}{\mathbf{18000}\,^{R}/_{100\,min^2}}$	num: 32 div: 1125
mm/s² 1 Dec. 1/10 $^{mm}/_{s^2}$ ($^{mm}/_{10s^2}$)	63.15 $^{mm}/_{R}$ ⇒ 1 R_{OUT} = 631.5 $^{mm}/_{10}$	$1\,\frac{1}{s^2} =$ $60\,\frac{1}{\text{min}\cdot\text{s}} =$ $60\cdot256\,\frac{\frac{1}{\text{min}}}{256\cdot\text{s}}$	4/5	$\frac{\frac{4R}{5R}\cdot\frac{256\cdot60\,^{1}/_{256\,min\cdot s}}{1\,^{1}/_{s^2}}}{\frac{631.5\,^{mm}/_{10}}{1R}} = \frac{\mathbf{122880}\,^{\frac{R}{min}}/_{256\,s}}{\mathbf{6315}\,^{mm}/_{10\,s^2}}$	num: 8192 div: 421

6.3.2.5 Object 607E$_h$: polarity

The signs of the position and velocity values of the servo controller can be adjusted via the corresponding polarity flag. This flag can be used to invert the direction of rotation of the motor keeping the same desired values. In most applications it makes sense to set the **position_polarity_flag** and the **velocity_polarity_flag** to the same value. The conversion factors will be used when reading or writing a position or velocity value. Stored parameters will not be affected.

Index	607E$_h$
Name	polarity
Object Code	VAR
Data Type	UINT8

Access	rw
PDO Mapping	yes
Units	--
Value Range	40$_h$, 80$_h$, C0$_h$
Default Value	0

Bit	Value	Name	Description
6	40$_h$	velocity_polarity_flag	0: multiply by 1 (default) 1: multiply by –1 (invers)
7	80$_h$	position_polarity_flag	0: multiply by 1 (default) 1: multiply by –1 (invers)

6.4 Power stage parameters

6.4.1 Survey

The intermediate circuit is charged by the main supply voltage using a precharge control. Thereby the current is limited and the loading process is controlled. The precharge control will be bridged if the intermediate circuit is loaded completely. Before this it is not possible to enable the controller.

The rectified supply voltage is smoothened by the capacitors of the intermediate circuit. The motor is fed from the intermediate circuit via the IGBTs. The power stage contains a number of security functions which can be configured in part:

- Controller enable logic (software and hardware enabling)
- Overcurrent control
- Over- and undervoltage control of the intermediate circuit
- Power stage control

6.4.2 Description of Objects

Index	Object	Name	Typ	Attr.
6510$_h$	VAR	drive_data		

6.4.2.1 Object 6510$_h$_10$_h$: enable_logic

The digital inputs **enable power stage** and **enable controller** have to be set so that the power stage of the servo controller can be activated. The input **enable power stage** directly acts on the trigger signals of the power transistors and would also be able to interrupt them in case of a defective microprocessor. Therefore the clearing of the signal **enable power stage** during the motor is rotating causes the effect that the motor coasts down without being braked or is only stopped by a possibly existing holding brake. The signal of the input **enable controller** is processed by the microcontroller of the servo controller. Depending on the mode of operation the servo controller reacts differently after clearing this signal:

- **Profile Position Mode** and **Profile Velocity Mode**

  The motor is decelerated using the defined brake ramp after clearing the signal. The power stage is switched off if the motor speed is below 10 rpm and a possibly existing holding brake is locked.

- **Torque Mode**

  The power stage is switched off immediately after the signal has been cleared. At the same time a possibly existing holding brake is locked. Therefore the motor coasts down without being braked or is only stopped by a stop brake which might exists.

CAUTION!
Both signals do not ensure that the motor is de-energised,
although the power stage has been switched off.

If the servo controller is operated via the CAN bus, it is possible to connect the digital inputs **enable power stage** and **enable controller** together at 24 V and to control the enabling via the CAN bus. To that object **6510$_h$_10$_h$** (**enable_logic**) has to be set to 2. For safety reasons this is done automatically after activation of CANopen (also after a reset).

Index	6510$_h$
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	10$_h$
Description	enable_logic
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0...2
Default Value	2

Value	Description
0	Digital inputs enable power stage + enable controller.
1	Digital inputs enable power stage + enable controller + RS232
2	Digital inputs enable power stage + enable controller + CAN

6.4.2.2 Object 6510$_h$_30$_h$: pwm_frequency

The switching losses of the power stage are proportional to the switching frequency of the power transistors. A little more power can be taken from some devices of the by dividing the PWM frequency by two.The disadvantage is an increasing current ripple. Switch over is only possible while the power stage is switched off.

Sub-Index	30_h
Description	pwm_frequency
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

Value	Description
0	Nominal PWM frequency
1	Half of nominal PWM frequency

6.4.2.3 Object 6510_h_$3A_h$: enable_enhanced_modulation

With the object **enable_enhanced_modulation** the enhanced modulation can be activated. The DC bus voltage will be used more effective thereby increasing the possible velocity by up to 14%. In contrast the control behaviour and the smooth running properties at low speed diminish a little. This object can only be written if the power stage is switched off.

Sub-Index	$3A_h$
Description	enable_enhanced_modulation
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

Value	Description
0	Enhanced sinus modulation switched OFF
1	Enhanced sinus modulation switched ON

The Activation of the enhanced modulation will need a Reset to become valid. Therefore the parameter set must be saved (**save_all_parameters**) and after it a Reset must be performed.

6.4.2.4 Object 6510_h_31_h: power_stage_temperature

The temperature of the power stage can be read via the object **power_stage_temperature**. If the temperature specified in the object **6510_h_32_h** (**max_power_stage_temperature**) is exceeded the power stage is switched off and an error message is sent.

Sub-Index	31_h
Description	power_stage_temperature
Data Type	INT16
Access	ro
PDO Mapping	yes
Units	°C
Value Range	--
Default Value	--

6.4.2.5 Object 6510$_h$_32$_h$: max_power_stage_temperature

The temperature of the power stage can be read via the object **6510$_h$_31$_h$** (**power_stage_temperature**). If the temperature specified in the object **max_power_stage_temperature** is exceeded the power stage is switched off and an error message is sent.

Sub-Index	32$_h$
Description	max_power_stage_temperature
Data Type	INT16
Access	ro
PDO Mapping	no
Units	°C
Value Range	--
Default Value	Device specific

Device Type	Value
C 1-02	100°C
C 1-05	80°C
C 3-05	80°C
C 3-10	80°C

6.4.2.6 Object 6510$_h$_33$_h$: nominal_dc_link_circuit_voltage

The nominal device voltage in mV can be read via the object **nominal_dc_link_circuit_voltage**.

Sub-Index	33$_h$
Description	nominal_dc_link_circuit_voltage
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	mV
Value Range	--
Default Value	Device specific

Device Type	Value
C 1-02	360000

C 1-05	360000
C 3-05	560000
C 3-10	560000

6.4.2.7 Object 6510$_h$_34$_h$: actual_dc_link_circuit_voltage

The current voltage in mV of the intermediate circuit can be read via the object actual_dc_link_circuit_voltage.

Sub-Index	34$_h$
Description	actual_dc_link_circuit_voltage
Data Type	UINT32
Access	ro
PDO Mapping	yes
Units	mV
Value Range	--
Default Value	--

6.4.2.8 Object 6510$_h$_35$_h$: max_dc_link_circuit_voltage

The max_dc_link_circuit_voltage indicates above which voltage in the intermediate circuit the power stage is immediately switched off for reasons of safety. An error message is sent, too.

Sub-Index	35$_h$
Description	max_dc_link_circuit_voltage
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	mV
Value Range	--
Default Value	Device specific

Device Type	Value
C 1-02	460000
C 1-05	460000
C 3-05	800000
C 3-10	800000

6.4.2.9 Object 6510$_h$_36$_h$: min_dc_link_circuit_voltage

The servo controller has an undervoltage control. This control can be activated via the object **6510$_h$_37$_h$ (enable_dc_link_undervoltage_error)**. The object **6510$_h$_36$_h$ (min_dc_link_circuit_voltage)** determines the lower voltage range. Below this voltage the servo controller decelerates and switches off the power stage afterwards. Furthermore the error E 02 0, “Undervoltage in DC bus” will be activated.

Sub-Index	36$_h$
Description	min_dc_link_circuit_voltage
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	mV
Value Range	0...1000000
Default Value	0

6.4.2.10 Object 6510$_h$_37$_h$: enable_dc_link_undervoltage_error

The undervoltage control can be activated via the object **enable_dc_link_undervoltage_error**. The voltage of the intermidiate circuit above wich the servo should work correctly has to be placed in the object **6510$_h$_36$_h$** (**min_dc_link_circuit_voltage**).

Sub-Index	37$_h$
Description	enable_dc_link_undervoltage_error
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

Value	Description
0	Undervoltage error switched OFF (Reaction: Warning)
1	Undervoltage error switched ON (Reaction: Disable servo controller)

The enabling of the error 02-0 is done by changing the error reaction. Reactions leading to a stop of the motor will be returned as ON, all the rest as OFF. On writing a 0 the reaction „Warning“ will be set, on writing a 1 the reaction „Disable servo controller“. See also chapter 6.18 (Error management).

6.4.2.11 Object 6510$_h$_40$_h$: nominal_current

The nominal current of the device can be read via this object. It is also the upper limit for the object 6075$_h$ (motor_rated_current).

Sub-Index	40$_h$
Description	nominal_current
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	mA
Value Range	--
Default Value	Device specific

Device type	Value
C 1-02	2500
C 1-05	5000
C 3-05	2500
C 3-10	5000

Depending on the controller's cycle time and the frequency of the power stage different values could be shown due to power derating.

6.4.2.12 Object 6510_h_41_h: peak_current

The devices peak current can be read via this object. It is also the upper limit for the object 6073_h (max_current).

Sub-Index	41_h
Description	peak_current
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	mA
Value Range	--
Default Value	Device specific

Device type	Value
C 1-02	5000
C 1-05	10000
C 3-05	7500
C 3-10	15000

Depending on the controller's cycle time and the frequency of the power stage different values could be shown due to power derating.

6.5 Current control and motor adaptation

Caution!

Incorrect setting of current control parameters and the current limits may possibly destroy the **motor** and even the **servo controller** immediately!

6.5.1 Survey

The parameter set of the servo controller has to be adapted to the connected motor and the used cable set. The following parameters are concerned:

▪ Nominal current	Depending on motor
▪ Overload	Depending on motor
▪ Pairs of poles	Depending on motor
▪ Current controller	Depending on motor
▪ Direction of rotation	Depending on motor and the phase sequence in the motor cable and the resolver cable
▪ Offset angle	Depending on motor and the phase sequence in the motor cable and the resolver cable

These data have to be determined by the program ™ when a motor type is used for the first time. You may obtain elaborate parameter sets for a number of motors from your dealer. Please remember that direction of rotation and offset angle also depend on the used cable set. Therefore the parameter sets only work correctly if wiring is identical.

Permuted phase order in the motor or the resolver cable may result in a positive feedback so the velocity in the motor cannot be controlled. The motor will rotate uncontrolled!

6.5.2 Description of Objects

Index	Object	Name	Type	Attr.
6075_h	VAR	motor_rated_current	UINT32	rw
6073_h	VAR	max_current	UINT16	rw
$604D_h$	VAR	pole_number	UINT8	rw
6410_h	RECORD	motor_data		rw
$60F6_h$	RECORD	torque_control_parameters		rw

6.5.2.1 Object 6075_h: motor_rated_current

This value can be read on the motor plate and is specified in mA (effective value, RMS). The upper limit is determined by the object **6510_h_40_h: nominal_current**.

Index	6075_h
Name	motor_rated_current
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	mA
Value Range	0...nominal_current
Default Value	296

If a new value is written into the object **6075_h** (**motor_rated_current**) also object **6073_h** (**max_current**) has to be rewritten.

6.5.2.2 Object 6073$_h$: max_current

Servo motors may be overloaded for a certain period of time. The maximum permissible motor current is set via this object. It refers to the nominal motor current (object 6075$_h$: motor_rated_current) and is set in thousandths. The upper limit for this object is determined by the object 6510$_h$_41$_h$ (peak_current). Many motors may be overloaded by the factor 2 for a short while. In this case the value 2000 has to be written into this object.

Before writing object 6073$_h$ (max_current) the object 6075$_h$ (motor_rated_current) must have a valid value.

Index	6073$_h$
Name	max_current
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	per thousands of rated current
Value Range	--
Default Value	2023

6.5.2.3 Object 604D$_h$: pole_number

The number of poles of the motor can be read in the datasheet of the motor or the parameter set-up program ™. The number of poles is always an integer value. Often the number of pole pairs is specified instead of the number of poles. In this case the number of poles equals the number of pole pairs multiplied with two.
This parameter will not be changed by restore_default_parameters.

Index	604D$_h$
Name	pole_number
Object Code	VAR
Data Type	UINT8

Access	rw
PDO Mapping	yes
Units	--
Value Range	2... 254
Default Value	4 (after *INIT!*)

6.5.2.4 Object 6410_h_03_h: iit_time_motor

Servo motors may be overloaded for a certain period of time. This object indicates how long the motor may receive a current specified in the object **6073_h** (**max_current**). After the expiry of the I²t-time the current is automatically limited to the value specified in the object **6075_h** (**motor_rated_current**) in order to protect the motor. The default adjustment is 2 seconds and can be used for most motors.

Index	**6410_h**
Name	**motor_data**
Object Code	RECORD
No. of Elements	5

Sub-Index	**03_h**
Description	**iit_time_motor**
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	ms
Value Range	0…10000
Default Value	2000

6.5.2.5 Object 6410_h_04_h: iit_ratio_motor

The actual value of iit can be read via the object **iit_ratio_motor**.

Sub-Index	**04_h**
Description	**iit_ratio_motor**
Data Type	UINT16
Access	ro
PDO Mapping	no
Units	per mille
Value Range	--
Default Value	--

6.5.2.6 Object 6510$_h$_38$_h$: iit_error_enable

The object **iit_error_enable** determines the behaviour when reaching the iit limit: Either this will only be displayed in the **statusword** or Error 31 0 will be generated

Index	6510$_h$
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	38$_h$
Description	iit_error_enable
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

Value	Description	
0	Iit-error switched OFF	(Reaction: Warning)
1	Iit-error switched On	(Reaction: Disable servo controller)

The enabling of the error 31-0 is done by changing the error reaction. Reactions leading to a stop of the motor will be returned as ON, all the rest as OFF. On writing a 0 the reaction „Warning“ will be set, on writing a 1 the reaction „Disable servo controller“. See also chapter 6.18 (Error management).

6.5.2.7 Object 6410$_h$_10$_h$: phase_order

With the object **phase_order** it is possible to consider permutations of motor- or resolver cable. This value can be taken from ™. A zero means “right”, a one means “left”.

Sub-Index	10_h
Description	phase_order
Data Type	INT16
Access	rw
PDO Mapping	yes
Units	--
Value Range	0, 1
Default Value	0

Value	Description
0	Right
1	Left

6.5.2.8 Object 6410_h_11_h: encoder_offset_angle

In case of the used servo motors permanent magnets are on the rotor. These magnets generate a magnetic field whose orientation to the stator depends on the rotor position. For the electronic commutation the controller always has to position the electromagnetic field of the stator in the correct angle towards this permanent magnetic field. For that purpose it permanently determines the rotor position with an angle encoder (resolver etc.).
The orientation of the angle encoder to the magnetic field has to be written to the object **resolver_offset_angle**. This angle can be determined by the parameter set-up program ™. The angle determined by the parameter set-up program ™ is in the range of +/-180°. It has to be converted as follows to be written into the object **resolver_offset_angle**:

$$\text{encoder\_offset\_angle} = \text{„Offset of encoder“} \times \frac{32767}{180°}$$

This parameter will not be changed by **restore_default_parameters**.

Index	6410_h
Name	motor_data
Object Code	RECORD
No. of Elements	5

Sub-Index	11_h
Description	encoder_offset_angle
Data Type	INT16
Access	rw
PDO Mapping	yes
Units	--
Value Range	-32767...32767
Default Value	$E000_h$ (-45°) (after *INIT*)

6.5.2.9 Object 6410_h_14_h: motor_temperature_sensor_polarity

The polarity of the motor temperature sensor can be configured by this object. For B-contacts (normally closed) zero has to be entered, for A-contacts (normally opened) one.

Sub-Index	14_h
Description	motor_temperatur_sensor_polarity
Data Type	INT16
Access	rw
PDO Mapping	yes
Units	--
Value Range	0, 1
Default Value	0

As of Firmware 3.2.0.1.1

Value	Meaning
0	B-contact (normally closed)
1	A-contact (normally opened)

6.5.2.10 Objekt 6510_h_$2E_h$: motor_temperature

The current motor temperature can be read by this object if a suitable (analogue) motor temperature sensor is connected to the servo. Otherwise the object is undefined.

Index	6510_h
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	$2E_h$
Description	motor_temperature
Data Type	INT16
Access	ro
PDO Mapping	yes
Units	°C
Value Range	--
Default Value	--

As of Firmware 3.5.x.1.1

6.5.2.11 Objekt 6510_h_$2F_h$: max_motor_temperature

With this object the maximum motor temperature can be set: If the motor_temperature exceeds this value, a reaction according to the error management (Error 3-0, Overtemperature motor analog) will be initiated. If this reaction stops the movement („Error“) an emergency telegram will be sent. See Chapter 6.18 (error management) for more details.

Sub-Index	$2F_h$
Description	max_motor_temperature
Data Type	INT16
Access	rw
PDO Mapping	no
Units	°C
Value Range	20...300
Default Value	100

As of Firmware 3.5.x.1.1

6.5.2.12 Object 60F6$_h$: torque_control_parameters

The data of the current controller has to be taken from the parameter set-up program ™. The following conversions have to be noticed:

The gain of the current controller has to be multiplied by 256. In case of a gain of 1.5 in the parameter set-up program ™ the value 384 = 180$_h$ has to be written into the object **torque_control_gain**.

The time constant of the current controller is specified in milliseconds in the parameter set-up program ™. This time constant has to be converted to microseconds before it can be transferred into the object **torque_control_time**. In case of a specified time of 0.6 milliseconds a value of 600 has to be entered into the object **torque_control_time**.

Index	60F6$_h$
Name	torque_control_parameters
Object Code	RECORD
No. of Elements	2

Sub-Index	01$_h$
Description	torque_control_gain
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	256 = „1“
Value Range	0...32*256
Default Value	3*256 (768)

Sub-Index	02$_h$
Description	torque_control_time
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	µs
Value Range	104... 64401
Default Value	1020

6.6 Velocity controller

6.6.1 Survey

The parameter set of the servo controller has to be adapted to the specific application. In particular the gain strongly depends on the masses coupled to the motor. So the data have to be determined by means of the program ™ when the plant is set into operation.

Incorrect setting of the velocity control parameters may lead to strong vibrations and destroy parts of the plant!

6.6.2 Description of Objects

Index	Object	Name	Type	Attr.
60F9$_h$	RECORD	velocity_control_parameters		rw
2073$_h$	VAR	velocity_display_filter_time	UINT32	rw

6.6.2.1 Object 60F9$_h$: velocity_control_parameters

The data of the velocity controller can be taken from the parameter set-up program ™. Note the following conversions:

The gain of the velocity controller has to be multiplied by 256. In case of a gain of 1.5 in ™ the value 384 has to be written into the object **velocity_control_gain**.

The time constant of the velocity controller is specified in milliseconds in ™. This time constant has to be converted to microseconds before it can be transferred into the object **velocity_control_time**. In case of a specified time of 2.0 milliseconds a value of 2000 has to be written into the object **velocity_control_time**.

Index	60F9$_h$
Name	velocity_control_parameter_set
Object Code	RECORD
No. of Elements	3

Sub-Index	01$_h$
Description	velocity_control_gain
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	256 = Gain 1
Value Range	20...64*256 (16384)
Default Value	256

Sub-Index	02$_h$
Description	velocity_control_time
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	µs
Value Range	1...32000
Default Value	2000

Sub-Index	04$_h$
Description	velocity_control_filter_time
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	µs
Value Range	1...32000
Default Value	400

6.6.2.2 Objekt 2073$_h$: velocity_display_filter_time

The filter time of the filter for the actual display velocity (**velocity_actual_value_filtered**) can be configured by this object. This velocity value should only be used for display purposes.

As of Firmware 3.5.x.1.1

Index	2073$_h$
Name	**velocity_display_filter_time**
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	no
Units	µs
Value Range	1000..50000
Default Value	20000

⚠ The **velocity_actual_value_filtered** will be used for overspeed protection. If the **velocity_display_filter_time** is set to a great value, an overspeed error will be detected delayed.

6.7 Position Control Function

6.7.1 Survey

This chapter describes all parameters which are required for the position controller. The desired position value (**position_demand_value**) of the trajectory generator is the input of the position controller. Besides this the actual position value (**position_actual_value**) is supplied by the angle encoder (resolver, incremental encoder, etc.). The behaviour of the position controller can be influenced by parameters. It is possible to limit the output quantity (**control_effort**) in order to keep the position control system stable. The output quantity is supplied to the speed controller as desired speed value. In the **Factor Group** all input and output quantities are converted from the application-specific units to the respective internal units of the controller.

The following subfunctions are defined in this chapter:

1. Trailing error (Following Error)

The deviation of the actual position value (**position_actual_value**) from the desired position value (**position_demand_value)** is named trailing error. If for a certain period of time this trailing error is bigger than specified in the trailing error window (**following_error_window**) bit 13 (**following_error**) of the object **statusword** will be set. The permissible time can be defined via the object **following_error_time_out**.

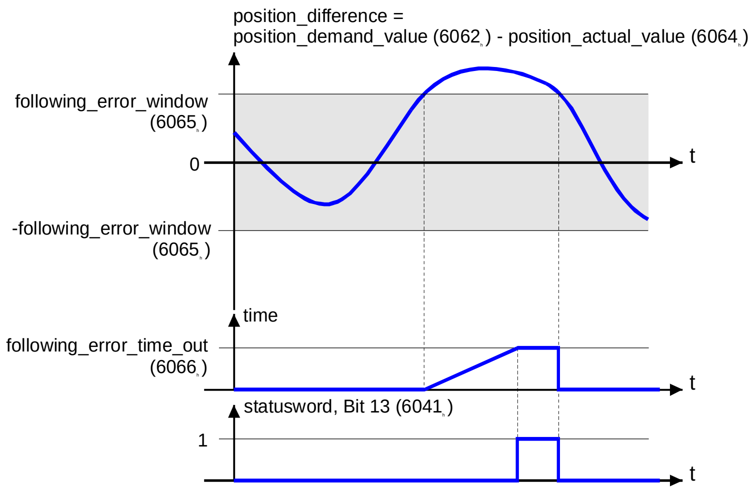


Figure 6.6: Trailing error (Following Error) – Function Survey

Figure 6.7 shows how the window function is defined for the message "following error". The range between x_i-x_0 and x_i+x_0 is defined symmetrically around the desired position (**position_demand_value**) x_i. For example the positions x_{t2} and x_{t3} are outside this window (**following_error_window**). If the drive leaves this window and does not return to the window within the time defined in the object **following_error_time_out** then bit 13 (**following_error**) in the **statusword** will be set.

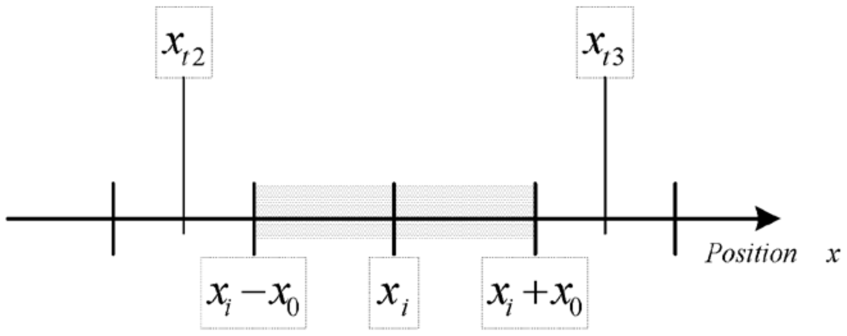


Figure 6.7: Trailing error (following error)

2. Position Reached

This function offers the chance to define a position window around the target position (**target_position**). If the actual position of the drive is within this range for a certain period of time – the **position_window_time** – bit 10 (**target_reached**) will be set in the **statusword.**

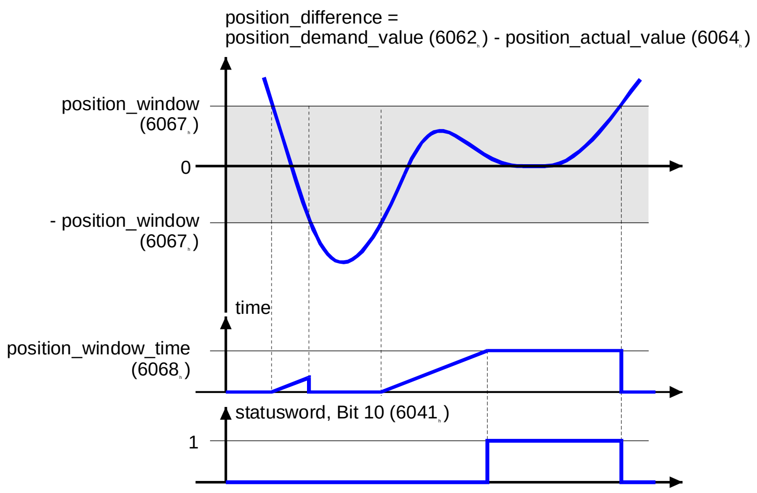


Figure 6.8: Position Reached – Function Survey

Figure 6.9 shows how the window function is defined for the message ”position reached”. The position range between x_i-x_0 and x_i+x_0 is defined symmetrically around the target position (**target_position**) x_i. For example the positions x_{t0} and x_{t1} are inside this position window (**position_window**). If the drive is within this window a timer is started. If this timer reaches the time defined in the object **position_window_time** and the drive uninterruptedly was within the valid range between x_i-x_0 and x_i+x_0, bit 10 (**target_reached**) will be set in the **statusword**. As far as the drive leaves the permissible range, bit 10 is cleared and the timer is set to zero.

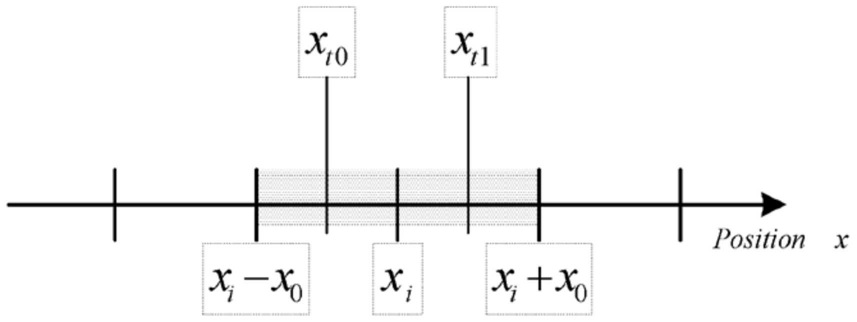


Figure 6.9: Position reached

6.7.2 Description of Objects

6.7.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$202D_h$	VAR	position_demand_sync_value	INT32	ro
6062_h	VAR	position_demand_value	INT32	ro
6063_h	VAR	position_actual_value*	INT32	ro
6064_h	VAR	position_actual_value	INT32	ro
6065_h	VAR	following_error_window	UINT32	rw
6066_h	VAR	following_error_time_out	UINT16	rw
6067_h	VAR	position_window	UINT32	rw
6068_h	VAR	position_window_time	UINT16	rw
$607B_h$	ARRAY	position_range_limit	INT32	rw
$60FA_h$	VAR	control_effort	INT32	ro
$60FB_h$	RECORD	position_control_parameter_set		rw
$60FC_h$	VAR	position_demand_value*	INT32	ro
6510_h_20_h	VAR	position_range_limit_enable	UINT16	rw
6510_h_22_h	VAR	position_error_switch_off_limit	UINT32	rw

6.7.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
$607A_h$	VAR	target_position	INT32	8.3 Operating Mode »Profile Position Mode«
$607C_h$	VAR	home_offset	INT32	8.2 Operating Mode »Homing mode«
$607D_h$	VAR	software_position_limit	INT32	8.3 Operating Mode »Profile Position Mode«
$607E_h$	VAR	polarity	UINT8	6.3 Conversion factors (Factor Group)
6093_h	VAR	position_factor	UINT32	6.3 Conversion factors (Factor Group)
6094_h	ARRAY	velocity_encoder_factor	UINT32	6.3 Conversion factors (Factor Group)
6096_h	ARRAY	acceleration_factor	UINT32	6.3 Conversion factors (Factor Group)
6040_h	VAR	controlword	INT16	7 Device Control
6041_h	VAR	statusword	UINT16	7 Device Control

6.7.2.3 Object $60FB_h$: position_control_parameter_set

All parameters of the servo controller have to be adapted to the specific application. Therefore the position control parameters have to be determined optimal by means of the parameter set-up program ™.

Incorrect setting of the position control parameters may lead to strong vibrations and so destroy parts of the plant!

The position controller compares the desired position with the actual position and forms a correction speed (Object **$60FA_h$: control_effort**). This correction speed is supplied to the speed controller. The position controller is relatively slow compared to the current controller and speed controller. Therefore the controller internally works with feed forward so that the correction work for the position controller is minimised reaching a fast settling time.
Usually a proportional control unit is sufficient as position controller. The gain of the position controller has to be multiplied by 256. In case of a gain of 1.5 in the menu **Position controller** of the parameter set-up program ™ the value 384 = 180_h has to be written into the object **position_control_gain**.
Normally the position controller can work without an integrator. In this case 0 has to be written into the object **position_control_time**. Otherwise the time constant of the position controller has to be converted to microseconds. So the value 4000 has to be written into the object **position_control_time** in case of a time of 4.0 milliseconds.

As the position controller even transforms smallest deviations into a considerable correction speed, very high correction speeds may occur in case of a short disturbance (e. g. short blocking). This can be avoided if the output of the position controller is adequately limited (e.g. 500 rpm) via the object position_control_v_max.
The object position_error_tolerance_window determines the maximum control deviation without reaction of the position controller. Therewith it is possible to even out backlash within the plant.

Index	60FB$_h$
Name	position_control_parameter_set
Object Code	RECORD
No. of Elements	4

Sub-Index	01$_h$
Description	position_control_gain
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	256 = „1“
Value Range	0...64*256 (16384)
Default Value	102

Sub-Index	02$_h$
Description	position_control_time
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	µs
Value Range	0
Default Value	0

Sub-Index	04$_h$
Description	position_control_v_max
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	speed units
Value Range	0...131072 min^{-1}
Default Value	500 min^{-1}

Sub-Index	05_h
Description	position_error_tolerance_window
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	position units
Value Range	1...65536 (1 R)
Default Value	2 (1 / 32768 R)

6.7.2.4 Object 6062_h: position_demand_value

The current position demand value can be read by this object. This position is fed into the position controller by the trajectory generator.

Index	6062_h
Name	position_demand_value
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

6.7.2.5 Object $202D_h$: position_demand_sync_value

The position setpoint of the synchronization encoder can be read by this object. This position is defined by the object 2022_h synchronization_encoder_select (chap.6.11). This object is specified in user-defined units.

Index	202D$_h$
Name	position_demand_sync_value
Object Code	VAR
Data Type	INT32

As of Firmware 3.2.0.1.1

Access	ro
PDO Mapping	no
Units	position units
Value Range	--
Default Value	--

6.7.2.6 Object 6064$_h$: position_actual_value

The actual position can be read by this object. This value is given to the position controller by the angle encoder. This object is specified in user-defined units.

Index	6064$_h$
Name	position_actual_value
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

6.7.2.7 Object 6065$_h$: following_error_window

The object **following_error_window** (trailing error window) defines a symmetrical range around the desired position value (**position_demand_value**). If the actual position (**position_actual_value**) is outside the trailing error window (**following_ error_window**) a trailing error occurs and bit 13 in the object **statusword** will be set.

The following reasons may cause a trailing error:

- A drive is locked
- The positioning speed is too high
- The accelerations are too high
- The object **following_error_window** configured too small
- The position controller is not configured correctly.

Index	6065_h
Name	following_error_window
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	position units
Value Range	$0...7FFFFFFF_h$
Default Value	9101 (9101 / 65536 R = 50°)

6.7.2.8 Object 6066$_h$: following_error_time_out

If a trailing error occurs longer than defined in this object bit 13 (following_error) will be set in the statusword.

Index	6066$_h$
Name	following_error_time_out
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	ms
Value Range	0...27314
Default Value	0

6.7.2.9 Object 60FA$_h$: control_effort

The output quantity of the position controller can be read via this object. This value is supplied internally to the speed controller as desired value.

Index	60FA$_h$
Name	control_effort
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	--

6.7.2.10 Object 6067$_h$: position_window

A symmetrical range around the target position (**target_position)** is defined by the object **position_window**. If the actual position value (**position_actual_value)** is within this range the target position (**target_position)** is regarded as reached.

Index	6067$_h$
Name	position_window
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	1820 (1820 / 65536 R = 10°)

6.7.2.11 Object 6068$_h$: position_window_time

If the actual position of the drive is within the positioning window (**position_window**) as long as defined in this object bit 10 (**target_reached**) will be set in the **statusword**.

Index	6068$_h$
Name	position_window_time
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	ms
Value Range	0...65536
Default Value	0

6.7.2.12 Object 6510$_h$_22$_h$: position_error_switch_off_limit

In the object **position_error_switch_off_limit** the maximum acceptable deviation between the position setpoint and the position actual value can be entered. In contrast to the following error message mentioned above the power stage will be immediately disabled, whenever the deviation is exceeded, and an error will occur. The motor rotates non-braked, except if a holding brake is available.

Index	6510$_h$
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	22$_h$
Description	position_error_switch_off_limit
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	position units
Value Range	0...2^{32}-1
Default Value	0

As of Firmware 3.2.0.1.1

Value	Meaning	
0	Limit value following error OFF	(Reaction: NONE)
> 0	Limit value following error ON	(Reaction: DISABLE POWER STAGE IMMEDIATELY)

The enabling of the error 17-0 is done by changing the error reaction. The reaction DISABLE POWER STAGE IMMEDIATELY is returned as **ON** and all others are returned as **OFF**. If the value is set to 0, the error reaction will be set to NONE. If a value greater as 0 is set the error reaction will be set to DISABLE POWER STAGE IMMEDIATELY. See also chapter 6.18 (Error management).

6.7.2.13 Object 607B$_h$: position_range_limit

The object group **position_range_limit** contains two subparameters, which limit the range of the position values. If one of these limits is exceeded, the position value automatically jumps to the respectively other limit. This allows the parameterisation of so-called circular axes. For this purpose the limits must be specified, which should equal the same position (e.g. 0° and 360°).
So that the limits take effect, a circular axis mode must be selected via the object **6510$_h$_20$_h$** (**position_range_limit_enable**).

Index	607B$_h$
Name	position_range_limit
Object Code	ARRAY
No. of Elements	2
Data Type	INT32

As of Firmware 3.3.x.1.1

Sub-Index	01$_h$
Description	min_position_range_limit
Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

As of Firmware 3.3.x.1.1

Sub-Index	02$_h$
Description	max_position_range_limit
Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

As of Firmware 3.3.x.1.1

6.7.2.14 Object 6510$_h$_20$_h$: position_range_limit_enable

The range limits, defined by the object **607B$_h$**, can be activated via the object **position_range_limit_enable.** Diverse modes are possible.

If the mode "shortest way" is selected, the positioning jobs are always executed using the physical shortest distance to the target. The drive adjusts the sign of the running speed. In the both modes "fixed rotating direction" the positioning job occurs basically in the direction, specified by the appropriate mode.

Index	6510$_h$
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	20$_h$
Description	position_range_limit_enable
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0...5
Default Value	0

As of Firmware 3.3.x.1.1

Value	Description
0	Off
1	Shortest way (because of compatibility reasons)
2	Shortest way
3	Reserved
4	Fixed rotating direction „positive“
5	Fixed rotating direction „negative“

6.7.2.15 Object 2030$_h$: set_position_absolute

All readable actual position values can be set to a defined position without changing the physical position. No movement will be executed. If an absolute encoder system is connected, the position offset will be written into the encoder if possible. So in this case this position offset will remain after a Reset. The saving process runs in background independantly from this object. All further parameters which are assigned to the encoder memory are also saved with their current value.

Index	2030$_h$
Name	set_position_absolute
Object Code	VAR
Data Type	INT32

As of Firmware 3.5.x.1.1

Access	wo
PDO Mapping	no
Units	position units
Value Range	--
Default Value	--

6.8 Setpoint limitation

6.8.1 Description of objects

6.8.1.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
2415$_h$	RECORD	current_limitation		rw
2416$_h$	RECORD	speed_limitation		rw

6.8.1.2 Object 2415$_h$: current_limitation

The record **current_limitation** allows a limitation of the maximum current in the operating modes profile_position_mode, interpolated_position_mode, homing_mode and velocity_mode, whereby a torque limited speed control mode is possible. With the object **limit_current_input_channel** the source of the limit torque can be chosen. Possible sources are “fixed value” or an analogue input. Depending on the chosen source, the object **limit_current** determines the limit torque (source = fixed value) or the scaling factor for the analogue input (source = analogue input). In the first case the limit current in mA can be entered directly, in the latter case the current in mA, corresponding to an input value of 10V, has to be entered.

Index	2415_h
Name	current_limitation
Object Code	RECORD
No. of Elements	2

Sub-Index	01_h
Description	limit_current_input_channel
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0...4
Default Value	0

Sub-Index	02_h
Description	limit_current
Data Type	INT32
Access	rw
PDO Mapping	no
Units	mA
Value Range	--
Default Value	0

Value	Description
0	No limitation
1	AIN0
2	AIN1
3	AIN2
4	Field bus (Field bus selector 2)

6.8.1.3 Object 2416_h: speed_limitation

The record **speed_limitation** allows a limitation of the maximum motor speed in the operating mode profile_torque_mode, whereby a speed limited torque control is possible. With the object **limit_speed_input_channel** the source of the limit speed can be chosen. Possible sources are "fixed value" or an analogue input. Depending on the chosen source the object **limit_speed** determines either the limit speed (source = fixed value) or the scaling factor for the analogue input (source = analogue input). In the first case the limit speed can be entered directly, in the latter case the speed, corresponding to an input value of 10V, has to be entered.

Index	2416_h
Name	speed_limitation
Object Code	RECORD
No. of Elements	2

As of Firmware 3.3.0.1.1

Sub-Index	01_h
Description	limit_speed_input_channel
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0...4
Default Value	0

As of Firmware 3.3.0.1.1

Sub-Index	02_h
Description	limit_speed
Data Type	INT32
Access	rw
PDO Mapping	no
Units	speed units
Value Range	--
Default Value	0

As of Firmware 3.3.0.1.1

Value	Description
0	No limitation

1	AIN0
2	AIN1
3	AIN2
4	Field bus (Field bus selector 2)

6.9 Encoder settings

6.9.1 Survey

This chapter describes the configuration of the angle encoders X2A, X2B and the incremental input X10.

Caution!

Wrong angle encoder settings may lead to uncontrolled behaviour of the drive and maybe destroy parts of the system.

6.9.2 Description of Objects

6.9.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
2024_h	RECORD	encoder_x2a_data_field		ro
2024_h_01_h	VAR	encoder_x2a_resolution	UINT32	ro
2024_h_02_h	VAR	encoder_x2a_numerator	INT16	rw
2024_h_03_h	VAR	encoder_x2a_divisor	INT16	rw
2025_h	RECORD	encoder_x10_data_field		ro
2025_h_01_h	VAR	encoder_x10_resolution	UINT32	rw
2025_h_02_h	VAR	encoder_x10_numerator	INT16	rw
2025_h_03_h	VAR	encoder_x10_divisor	INT16	rw
2025_h_04_h	VAR	encoder_x10_counter	UINT32	ro
2026_h	RECORD	encoder_x2b_data_field		ro
2026_h_01_h	VAR	encoder_x2b_resolution	UINT32	rw
2026_h_04_h	VAR	encoder_x2b_counter	UINT32	ro

6.9.2.2 Object 2024_h: encoder_x2a_data_field

The record **encoder_x2a_data_field** summarises the parameters, which are necessary for the operation of the angle encoder connected on X2A.
Because numerous angle encoder settings are activated only after a reset, the selection and the settings of the encoders should be made with the ™.
The following settings can be read resp. changed via CANopen.
The object **encoder_x2a_resolution** specifies how many increments per revolution or per unit length an encoder produces. The value 65536 is returned always by this object, because only resolvers evaluated with 16 bit can be connected on the input X2A.
The objects **encoder_x2a_numerator** and **encoder_X2a_divisor** specify the gear (signed or unsigned) **between the motor shaft and the encoder**.

Index	2024h
Name	encoder_x2a_data_field
Object Code	RECORD
No. of Elements	3

As of Firmware 3.2.0.1.1

Sub-Index	01h
Description	encoder_x2a_resolution
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	increments (4 * line count)
Value Range	--
Default Value	65536

As of Firmware 3.2.0.1.1

Sub-Index	02h
Description	encoder_x2a_numerator
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	-32768 ... 32767 (except 0)
Default Value	1

As of Firmware 3.2.0.1.1

Sub-Index	03h
Description	encoder_x2a_divisor
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	1 ... 32767
Default Value	1

As of Firmware 3.2.0.1.1

6.9.2.3 Object 2026$_h$: encoder_x2b_data_field

The record **encoder_x2b_data_field** summarises the parameters, which are necessary for the operation of the angle encoder connected on X2B.

The object **encoder_x2b_resolution** specifies how many increments per revolution an encoder produces (for incremental encoders the resolution equals the fourfold line count resp. periods per revolution). The object **encoder_x2b_counter** returns the actual counted number of increments. It returns values between 0 and the number of increments – 1.

An own gear cannot be specified for X2B. The objects **encoder_x2a_numerator** and **encoder_x2a_divisor** can be used instead to set the gear for X2B. The objects **encoder_x2a_numerator** and **encoder_X2a_divisor** specify the gear (signed or unsigned) **between the motor shaft and the encoder connected to X2b**.

Index	2026$_h$
Name	encoder_x2b_data_field
Object Code	RECORD
No. of Elements	4

As of Firmware 3.2.0.1.1

Sub-Index	01$_h$
Description	encoder_x2b_resolution
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	increments (4 * line count)
Value Range	depends on the used encoder
Default Value	depends on the used encoder

As of Firmware 3.2.0.1.1

Sub-Index	02$_h$
Description	encoder_x2b_numerator
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	-32768 ... 32767
Default Value	1

As of Firmware 3.3.0.1.1

Sub-Index	03_h
Description	encoder_x2b_divisor
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	1 ... 32767
Default Value	1

As of Firmware 3.3.0.1.1

Sub-Index	04_h
Description	encoder_x2b_counter
Data Type	UINT32
Access	ro
PDO Mapping	yes
Units	incremente (4 * line count)
Value Range	0 ... (encoder_x2b_resolution – 1)
Default Value	--

As of Firmware 3.2.0.1.1

6.9.2.4 Object 2025$_h$: encoder_x10_data_field

The record encoder_x10_data_field summarises the parameters, which are necessary for the operation of the incremental input X10. A digital incremental encoder or emulated incremental signals of another can be connected to the incremental input. The input signals via X10 can be used as setpoint or as actual value. You can find more details in chapter 6.11.
The object encoder_x10_resolution specifies how many increments per revolution an encoder produces (this equals the fourfold line count). The object encoder_ x10_counter returns the actual counted number of increments (between 0 and the adjusted number of increments – 1). The objects encoder_x10_numerator and encoder_x10_divisor specify the gear ratio (signed or unsigned).
When using the X10 signals as an actual value, the gear represents the gear between the motor and actual value encoder connected to X10, which is mounted on the output. When using the X10 signals as a setpoint, a gear ratio between the master and the slave can be realised.

Index	2025$_h$
Name	encoder_x10_data_field
Object Code	RECORD
No. of Elements	4

As of Firmware 3.2.0.1.1

Sub-Index	01$_h$
Description	encoder_x10_resolution
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	increments (4 * line count)
Value Range	depends on the used encoder
Default Value	depends on the used encoder

As of Firmware 3.2.0.1.1

Sub-Index	02$_h$
Description	encoder_x10_numerator
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	-32768 ... 32767 (except 0)
Default Value	1

As of Firmware 3.2.0.1.1

Sub-Index	03$_h$
Description	encoder_x10_divisor
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	1 ... 32767
Default Value	1

As of Firmware 3.2.0.1.1

Sub-Index	04_h
Description	encoder_x10_counter
Data Type	UINT32
Access	ro
PDO Mapping	yes
Units	increments (4 * line count)
Value Range	0 ... (encoder_x10_resolution – 1)
Default Value	--

As of Firmware 3.2.0.1.1

6.10 Incremental encoder emulation

6.10.1 Survey

This object group makes it possible to adjust the incremental output X11. Thus master-slave application, by which the master's output X11 is connected to the slave's input X10, can be parametrised via CANopen.

6.10.2 Description of Objects

6.10.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
2028_h	VAR	encoder_emulation_resolution	INT32	rw
$201A_h$	RECORD	encoder_emulation_data		ro
$201A_h\_01_h$	VAR	encoder_emulation_resolution	INT32	rw
$201A_h\_02_h$	VAR	encoder_emulation_offset	INT16	rw

6.10.2.2 Object 2028_h: encoder_emulation_resolution

The object record **encoder_emulation_data** encapsulates all settings for the incremental output X11:

Via the object **encoder_emulation_resolution** the output number of increments (= fourfold line count) can be freely set as a multiple of 4. To achieve a ratio of 1:1 in a master-slave application, this must equal the slave's **encoder_X10_resolution**.

With the object **encoder_emulation_offset** the position of the output zero pulse can be shifted based on the zero position of the actual value encoder.

Index	$201A_h$
Name	encoder_emulation_data
Object Code	RECORD
No. of Elements	2

As of Firmware 3.2.0.1.1

Sub-Index	01_h
Description	encoder_emulation_resolution
Data Type	INT32
Access	rw
PDO Mapping	no
Units	increments (4 * line count)
Value Range	4 * (1...8192)
Default Value	4096

As of Firmware 3.2.0.1.1

Sub-Index	02_h
Description	encoder_emulation_offset
Data Type	INT16
Access	rw
PDO Mapping	no
Units	32767 = 180°
Value Range	-32768...32767
Default Value	0

As of Firmware 3.2.0.1.1

6.10.2.3 Object 2028_h: encoder_emulation_resolution

The object **encoder_emulation_resolution** is only avalilable because of compatibility reasons. It complies with the object **$201A_h$_01_h**.

Index	2028_h
Name	encoder_emulation_resolution
Object Code	VAR
Data Type	INT32

As of Firmware 3.2.0.1.1

Access	rw
PDO Mapping	no
Units	see $201A_h$_01_h
Value Range	see $201A_h$_01_h
Default Value	see $201A_h$_01_h

6.11 Sources for demand / actual value

6.11.1 Survey

The source for the setpoint and the source for the actual value can be changed with the following objects. As standard the controller uses the input of the motor encoder X2A resp. X2B as an actual value for position control. When using an external position encoder, e.g. behind a gear, the supplied position value via X10 can be selected as an actual value for the position controller. Furthermore it is possible to select the X10 input signals (e.g. of a second controller) as an additional setpoint, whereby synchronous operation modes are possible.

6.11.2 Description of Objects

6.11.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$201F_h$	VAR	commutation_encoder_select	INT16	rw
2021_h	VAR	position_encoder_selection	INT16	rw
2022_h	VAR	synchronisation_encoder_selection	INT16	rw
2023_h	VAR	synchronisation_filter_time	UINT32	rw
$20F0_h$	RECORD	synchronisation_selector_data		ro
$20F0_h$_07_h	VAR	synchronisation_main	UINT16	rw

6.11.2.2 Object $201F_h$: commutation_encoder_select

The object **commutation_encoder_select** specifies the encoder input, which is used as commutating encoder. Because this value will be activated only after a reset, the setting of the commutating encoder should be made with the ™.

Index	201F$_h$
Name	commutation_encoder_select
Object Code	VAR
Data Type	INT16

As of Firmware 3.2.0.1.1

Access	rw
PDO Mapping	no
Units	--
Value Range	0, 2 (see table)
Default Value	0

Value	Description
0	X2A
2	X2B

6.11.2.3 Object 2021$_h$: position_encoder_selection

The object **position_encoder_selection** specifies the encoder input, which is used for the determination of the actual position (actual value encoder). This value can be changed in order to switch to position control via an external (connected to the output) encoder. Thereby it can be switched between the X10 and the encoder input (X2A, X2B), selected as commutating encoder input. If one of the encoders X2A / X2B is selected as the actual value encoder, then the one, used as the commutating encoder, should be used. If the respectively other one encoder is selected, then it will be automatically switched to the commutating encoder.

As of Firmware 3.2.0.1.1

Index	2021$_h$
Name	position_encoder_selection
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	0...2 (see table)
Default Value	0

Value	Description
0	X2A
1	X2B
2	X10

It can be chosen only between the encoder input X10 and the respective commutating encoder X2A or X2B as the actual value encoder. The configuration X2A as the commutating encoder and X2B as the actual value encoder or otherwise is not possible.

6.11.2.4 Object 2022$_h$: synchronisation_encoder_selection

The object synchronisation_encoder_selection specifies the encoder input, which is used as a synchronisation setpoint. Depending on the operation mode this corresponds to a position setpoint (Profile Position Mode) or to a speed setpoint (Profile Velocity Mode).

Only X10 can be used as a synchronisation input. Thus only X10 or No input can be selected. The input choosen as synchronisation setpoint must not be equal to the position encoder selection.

As of Firmware 3.2.0.1.1

Index	2022$_h$
Name	synchronisation_encoder_selection
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	-1, 2 (see table)
Default Value	2

Value	Description
-1	No encoder / undefined
2	X10

6.11.2.5 Object $202F_h$: synchronisation_selector_data

By the object **synchronisation_main** the enabling of a synchronisation setpoint can be set. To calculate the synchronous setpoint at all, bit 0 must be set. Bit 1 makes it possible in newer firmware versions to switchg on the synchronous position by starting a positioning. At present only 0 is parametrisable, so that the synchronous position is always switched on. Bit 8 specifies if the homing is made without adding the synchronous position, in order to reference the master and the slave separately.

Index	$202F_h$
Name	synchronisation_selector_data
Object Code	RECORD
No. of Elements	1

As of Firmware 3.2.0.1.1

Sub-Index	07_h
Description	synchronisation_main
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	see table
Default Value	--

As of Firmware 3.2.0.1.1

Bit	Value	Meaning
0	0001_h	0: synchronisation disabled 1: synchronisation enabled
1	0002_h	"flying saw" not possible
8	0100_h	0: synchronous position is added while homing 1: synchronous position is not added while homing

6.11.2.6 Object 2023$_h$: synchronisation_filter_time

The object **synchronisation_filter_time** specifies the filter time constant of a PT1 filter by which the synchronisation speed will be smoothed. This may be necessary particularly for low line counts, because small changes of the input value here correspond to high speeds. On the other hand the drive may not be able to follow the input signal as fast as necessary due to high filter times.

Index	2023$_h$
Name	synchronisation_filter_time
Object Code	VAR
Data Type	UINT32

As of Firmware 3.2.0.1.1

Access	rw
PDO Mapping	no
Units	µs
Value Range	10...50 000
Default Value	600

6.12 Analogue inputs

6.12.1 Survey

The servo controller of the contains three analogue inputs, which can be used to enter a demand value for instance For all of these inputs the following objects allow to read out the current input voltage (**analog_input_voltage**) and to determine an offset (**analog_input_offset**).

6.12.2 Description of Objects

Index	Object	Name	Type	Attr.
2400_h	ARRAY	analog_input_voltage	INT16	ro
2401_h	ARRAY	analog_input_offset	INT32	rw

6.12.2.1 2400_h: analog_input_voltage

The object record **analog_input_voltage** returns the current input voltage in millivolt considering the offset voltage.

Index	2400_h
Name	analog_input_voltage
Object Code	ARRAY
No. of Elements	3
Data Type	INT16

Sub-Index	01_h
Description	analog_input_voltage_ch_0
Access	ro
PDO Mapping	no
Units	mV
Value Range	--
Default Value	--

Sub-Index	02_h
Description	analog_input_voltage_ch_1
Access	ro
PDO Mapping	no
Units	mV
Value Range	--
Default Value	--

Sub-Index	03_h
Description	analog_input_voltage_ch_2
Access	ro
PDO Mapping	no
Units	mV
Value Range	--
Default Value	--

6.12.2.2 Object 2401_h: analog_input_offset (Offset Analogeingänge)

The object record **analog_input_offset** can be used to determine an offset voltage in millivolt for the respective analogue input. A positive offset compensates a positive dc offset voltage.

Index	2401_h
Name	analog_input_offset
Object Code	ARRAY
No. of Elements	3
Data Type	INT32

Sub-Index	01_h
Description	analog_input_offset_ch_0
Access	rw
PDO Mapping	no
Units	mV
Value Range	-10000...10000
Default Value	0

Sub-Index	02_h
Description	analog_input_offset_ch_1
Access	rw
PDO Mapping	no
Units	mV
Value Range	-10000...10000
Default Value	0

Sub-Index	03_h
Description	analog_input_offset_ch_2
Access	rw
PDO Mapping	no
Units	mV
Value Range	-10000...10000
Default Value	0

6.13 Digital inputs and outputs

6.13.1 Survey

All digital inputs of the servo controller can be read and almost all digital outputs can be set or reset using the can bus.

6.13.2 Description of Objects

6.13.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$60FD_h$	VAR	digital_inputs	UINT32	ro
$60FE_h$	ARRAY	digital_outputs	UINT32	rw
2420_h	RECORD	digital_output_state_mapping		ro
2420_h_01_h	VAR	dig_out_state_mapp_dout_1	UINT8	rw
2420_h_02_h	VAR	dig_out_state_mapp_dout_2	UINT8	rw
2420_h_03_h	VAR	dig_out_state_mapp_dout_3	UINT8	rw
2420_h_11_h	VAR	dig_out_state_mapp_ea88_0_low	UINT32	rw
2420_h_12_h	VAR	dig_out_state_mapp_ea88_0_high	UINT32	rw

6.13.2.2 Object 60FD$_h$: digital_inputs

Using object **60FD$_h$** the digital inputs can be read out:

Index	60FD$_h$
Name	digital_inputs
Object Code	VAR
Data Type	UINT32

Access	ro
PDO Mapping	yes
Units	--
Value Range	according table
Default Value	0

Bit	Value	Digital input
0	00000001$_h$	Negative limit switch
1	00000002$_h$	Positive limit switch
2	00000004$_h$	Reference switch
3	00000008$_h$	Interlock (either “controller enable” or “power stage enable” is missing)
16...23	00FF0000$_h$	Additional inputs of an optional EA88-module (EA88-0)
24...27	0F000000$_h$	DIN0...DIN3
28	10000000$_h$	DIN8
29	20000000$_h$	DIN9

6.13.2.3 Object 60FE$_h$: digital_outputs

The digital outputs can be set via the object **60FE$_h$**. For that purpose it has to be indicated which of the digital outputs are allowed to be set in the object **digital_outputs_mask**. Then the selected outputs can be set optionally via the object **digital_outputs_data**. It has to be kept in mind that a delay of up to 10 ms may occur between sending the command and a real reaction of the. The time the outputs are really set can be seen by rereading the object **60FE$_h$**.

Index	60FE$_h$
Name	digital_outputs
Object Code	ARRAY
No. of Elements	2
Data Type	UINT32

Sub-Index	01$_h$
Description	digital_outputs_data
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	depends on the state of the brake

Sub-Index	02$_h$
Description	digital_outputs_mask
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	00000000$_h$

Bit	Value	Digital Outputs
0	00000001$_h$	1 = Set brake, 0 = Release brake
25...27	0E000000$_h$	DOUT1...DOUT3

If the digital_output_mask is set accordingly, the brake can be released by clearing bit 0 of digital_output_data immediately.
This may cause the dropping of a vertical axis.

6.13.2.4 Object 2420$_h$: digital_output_state_mapping

By the object group **digital_outputs_state_mapping** different controller's status messages can be output via the digital outputs.

Each of the integrated controller's digital outputs has its own subindex. Each subindex summarises 4 outputs. Thus there is one byte for each output, where the function's number has to be entered.

If a function was assigned to a digital output and the output is then switched on or switched off via the object **digital_outputs** (**60FE$_h$**), the object **digital_outputs_state_mapping** is also set to OFF(0) resp. ON(12).

Index	2420h
Name	digital_outputs_state_mapping
Object Code	RECORD
No. of Elements	5

As of Firmware 3.2.0.1.1

Sub-Index	01h
Description	dig_out_state_mapp_dout_1
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0 ... 16, see table
Default Value	0

As of Firmware 3.2.0.1.1

Sub-Index	02h
Description	dig_out_state_mapp_dout_2
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0 ... 16, see table
Default Value	0

As of Firmware 3.2.0.1.1

Sub-Index	03h
Description	dig_out_state_mapp_dout_3
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	- ... 16, see table
Default Value	0

As of Firmware 3.2.0.1.1

Value	Description	Value	Description
0	OFF (Output is LOW)	9	Undervoltage intermediate circuit
1	Position $X_{SET} = X_{DEST}$	10	Brake released
2	Position $X_{ACT} = X_{DEST}$	11	Power stage released
3	Reserved	12	No function (ON)
4	Remaining distance trigger	13	Reserved
5	Homing operation active	14	Reserved
6	Target velocity reached	15	Linear motor identified
7	I^2t-limit active	16	Homing position valid
8	Following error		

6.14 Homing switches (Limit / Reference switch)

6.14.1 Survey

For the definition of the reference (zero) position of the servo controller optional limit switches or reference switches can be used. Further information concerning reference methods can be found in chapter *8.2,* Operating Mode »Homing mode«..

6.14.2 Description of Objects

Index	Object	Name	Type	Attr.
6510_h	RECORD	drive_data		rw

6.14.2.1 Object 6510_h_11_h: limit_switch_polarity

The polarity of the limit switches can be configured by the object **6510_h_11_h** (**limit_switch_polarity**).
For B-contacts (normally closed) zero has to be entered, for A-contacts (normally opened) one.

Index	6510_h
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	11_h
Description	limit_switch_polarity
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	1

Value	Description
0	B-contact (normally closed)

1	A-contact (normally opened)

6.14.2.2 Object 6510$_h$_12$_h$: limit_switch_selector

Using object **6510$_h$_12$_h$** (**limit_switch_selector**) the assignment of the limit switches (negative, positive) can be swapped, without changing the cabling. To swap the assignment one has to be entered.

Sub-Index	12$_h$
Description	limit_switch_selector
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	0

As of Firmware 3.5.x.1.1

Value	Meaning

0	DIN6 = E0 (limit switch negative)
	DIN7 = E1 (limit switch positive)

1	DIN6 = E1 (limit switch positive)
	DIN7 = E0 (limit switch negative)

6.14.2.3 Object 6510$_h$_14$_h$: homing_switch_polarity

The polarity of the homing switch can be configured by the object **6510$_h$_14$_h$** (**homing_switch_polarity**). For B-contacts (normally closed) zero has to be entered, for A-contacts (normally opened) one.

Sub-Index	14_h
Description	homing_switch_polarity
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1
Default Value	1

Value	Description
0	B-contact (normally closed)
1	A-contact (normally opened)

6.14.2.4 Object 6510$_h$_13$_h$: homing_switch_selector

The object **6510$_h$_13$_h$** (**homing_switch_selector**) determines, wether DIN8 or DIN9 should be used as reference switch.

Sub-Index	13$_h$
Description	homing_switch_selector
Data Type	INT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0,1
Default Value	1

Value	Description
0	DIN9
1	DIN8

6.14.2.5 Object 6510$_h$_15$_h$: limit_switch_deceleration

The object **limit_switch_deceleration** determines the deceleration used to stop the motor if a limit switch will be reached during normal operation (limit switch emergency stop).

Sub-Index	15$_h$
Description	limit_switch_deceleration
Data Type	INT32
Access	rw
PDO Mapping	no
Units	acceleration units
Value Range	0...3000000 min^{-1}/s
Default Value	2000000 min^{-1}/s

6.15 Sampling positions

6.15.1 Survey

The offer the possibility to store the actual position value at the rising or at the falling edge of a digital input. This position value can be later read and used for different purposes, e.g. for calculations within a control system.

All necessary objects are summarised in the record **sample_data**. The object **sample_mode** specifies the kind of the sampling: Should only a one-time sample event be recorded or should it be sampled continuously? By the object **sample_status** the control system can query, if a sample event has occurred. This will be signalised by a set bit, which can be also shown in the **statusword**, when the object **sample_status_mask** is correspondingly set.

The object **sample_control** is used to control the enabling of the sample events and finally the sampled positions can be read via the objects **sample_position_rising_edge** und **sample_position_falling_edge**.

In the menu Parameters / IOs / Digital Inputs / Sample-Input in the ™ can be specified, which digital input will be used.

6.15.2 Description of Objects

6.15.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$204A_h$	RECORD	sample_data		ro
$204A_h\_01_h$	VAR	sample_mode	UINT16	rw
$204A_h\_02$	VAR	sample_status	UINT8	ro
$204A_h\_03_h$	VAR	sample_status_mask	UINT8	rw
$204A_h\_04_h$	VAR	sample_control	UINT8	wo
$204A_h\_05_h$	VAR	sample_position_rising_edge	INT32	ro
$204A_h\_06_h$	VAR	sample_position_falling_edge	INT32	ro

6.15.2.2 Object 204A$_h$: sample_data

Index	204A$_h$
Name	sample_data
Object Code	RECORD
No. of Elements	6

As of Firmware 3.2.0.1.1

The following object determines, if the position is acquired on each sample event (continuous sampling) or if the sampling should be inhibited after a sample event, until the sampling is restarted. Please consider here, that a bouncing could have already activated both edges.

Sub-Index	01$_h$
Description	sample_mode
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	--
Value Range	0 ... 1, see table
Default Value	0

As of Firmware 3.2.0.1.1

Value	Meaning
0	continuous sampling
1	auto lock sampling

The following object indicates a new sample event.

Sub-Index	02_h
Description	sample_status
Data Type	UINT8
Access	ro
PDO Mapping	yes
Units	--
Value Range	0 ... 3, see table
Default Value	0

As of Firmware 3.2.0.1.1

Bit	Value	Name	Description
0	01_h	falling_edge_occurred	1 = New sample position (falling edge)
1	02_h	rising_edge_occurred	1 = New sample position (rising edge)

With the following object the bits of the object **sample_status** can be specified, which should cause the setting of bit 15 in the **statusword**. Thereby it is possible to have the information “sample event occurred” in the **statusword**, which may be transmitted cyclically anyhow. Only if this bit is set in the **statusword** it is neccessary to read the **sample_status** additionally to find out if the falling or rising edge has occurred.

Sub-Index	03_h
Description	sample_status_mask
Data Type	UINT8
Access	rw
PDO Mapping	yes
Units	--
Value Range	0 ... 3, see table
Default Value	0

As of Firmware 3.2.0.1.1

Bit	Value	Name	Description
0	01_h	falling_edge_visible	If falling_edge_occured = 1 => statusword Bit 15 = 1
1	02_h	rising_edge_visible	If rising_edge_occurred = 1 => statusword Bit 15 = 1

The setting of the respective bits in **sample_control** causes the reset of the corresponding status bit in **sample_status** and in the case of “autolock” sampling the sampling is released again.

Sub-Index	04_h
Description	sample_control
Data Type	UINT8
Access	wo
PDO Mapping	yes
Units	--
Value Range	0 ... 3, see table
Default Value	0

As of Firmware 3.2.0.1.1

Bit	Value	Name	Description
0	01_h	falling_edge_enable	Allow sampling on falling edge
1	02_h	rising_edge_enable	Allow sampling on rising edge

The following objects contain the sampled positions.

Sub-Index	05_h
Description	sample_position_rising_edge
Data Type	INT32
Access	ro
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

As of Firmware 3.2.0.1.1

Sub-Index	06_h
Description	sample_position_falling_edge
Data Type	INT32
Access	ro
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

As of Firmware 3.2.0.1.1

6.16 Brake control

6.16.1 Survey

Using the following objects it can be determined how a possibly existing motor brake will be controlled by the servo controller. The brake will always be released if the controller is enabled, i.e. if the external and internal enable is present. For the use of brakes with a high inertia a delay time can be determined to ensure the brake is locked before the power stage switches off (Dropping of vertical axis). This delay time can be set via the object **brake_delay_time**. As can be seen in the following figure the velocity demand value will be delayed for the **brake_delay_time** after enabling the servo controller. Equally the deactivation of the power stage will be delayed when disabling the servo controller.

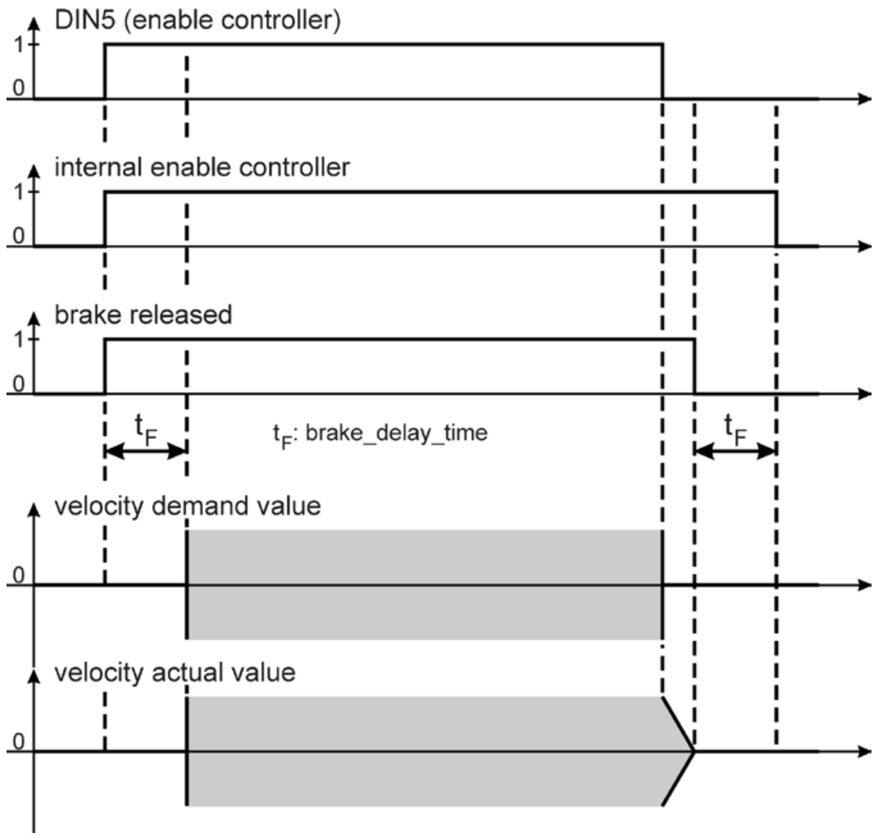


Figure 6.10: Function of brake delay (in Operating Mode Profile Velocity Mode and Operating Mode »Profile Position Mode«)

6.16.2 Description of Objects

Index	Object	Name	Type	Attr.
6510_h	RECORD	drive_data		rw

6.16.2.1 Object 6510_h_18_h: brake_delay_time

With the object **brake_delay_time** the delay of the brake can be configured.

Index	6510_h
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	18_h
Description	brake_delay_time
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	ms
Value Range	0...32000
Default Value	0

6.17 Device informations

Index	Object	Name	Type	Attr.
1018h	RECORD	identity_object		rw
6510h	RECORD	drive_data		rw

A huge number of CAN objects have been implemented to read out several device informations like type of servo controller, firmware revision and so on.

6.17.1 Description of Objects

6.17.1.1 Object 1018$_h$: identity_object

To identify the servo controller uniquely in a CANopen-network the **identity_object** according to the DS301 can be used.

A unique manufacturer code (**vendor_id**), a unique product code (**product_code**), the revision number of the CANopen implementation (**revision_number**) and the device serial number (**serial_number**) can be read.

Index	1018$_h$
Name	identity_object
Object Code	RECORD
No. of Elements	4

Sub-Index	01$_h$
Description	vendor_id
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	000000E4
Default Value	000000E4

Sub-Index	02_h
Description	product_code
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	s.u.
Default Value	s.u.

Value	Description
2045_h	C 1-02
2046_h	C 1-05
2050_h	
$204A_h$	C 3-05
$204B_h$	C 3-10

Sub-Index	03_h
Description	revision_number
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	$MMMMSSSS_h$ (M: main version, S: sub version)
Value Range	--
Default Value	--

Sub-Index	04_h
Description	serial_number
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

6.17.1.2 Object 6510_h_$A0_h$: drive_serial_number

With the object **drive_serial_number** the serial number of the servo controller can be read. This object is implemented because of terms of compatibility to older versions.

Index	6510_h
Name	drive_data
Object Code	RECORD
No. of Elements	51

Sub-Index	$A0_h$
Description	drive_serial_number
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

6.17.1.3 Object 6510_h_$A1_h$: drive_type

The object **drive_type** returns the type of servo controller. This object is implemented because of terms of compatibility to older versions.

Sub-Index	$A1_h$
Description	drive_type
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	
Value Range	see 1018_h_02_h, product_code
Default Value	see 1018_h_02_h, product_code

6.17.1.4 Object 6510$_h$_A9$_h$: firmware_main_version

The object **firmware_main_version** returns the main revision index of the firmware (product step).

Sub-Index	A9$_h$
Description	firmware_main_version
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	MMMMSSSS$_h$ (M: main version, S: sub version)
Value Range	--
Default Value	--

6.17.1.5 Object 6510$_h$_AA$_h$: firmware_custom_version

The object **firmware_custom_version** returns the version number of the customer-specific variant of the firmware.

Sub-Index	AA$_h$
Description	firmware_custom_version
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	MMMMSSSS$_h$ (M: main version, S: sub version)
Value Range	--
Default Value	--

6.17.1.6 Object 6510$_h$_AD$_h$: km_release

The version information **km_release** allows differentiating firmware versions of the same product step (**firmware_main_version**).

Sub-Index	AD_h
Description	km_release
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	$MMMMSSSS_h$ (M: main version, S: sub version)
Default Value	--

As of Firmware 3.5.x.1.1

6.17.1.7 Object 6510$_h$_AC$_h$: firmware_type

With the object **firmware_type** it can be determined for what type of device and encoder module the firmware is suitable. For the the encoder interface can not be plugged anymore. Consequently the parameter G will always return F$_h$.

Sub-Index	AC$_h$
Description	firmware_type
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	000000GX$_h$
Value Range	00000F2$_h$
Default Value	00000F2$_h$

Value (X)	Description
0$_h$	IMD-F
1$_h$	ARS
2$_h$	ARS 2000

6.17.1.8 Object 6510$_h$_B0$_h$: cycletime_current_controller

The object **cycletime_current_controller** returns the period of the current control loop in microseconds.

Sub-Index	B0$_h$
Description	cycletime_current_controller
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	µs
Value Range	--
Default Value	00000068$_h$

6.17.1.9 Object 6510$_h$_B1$_h$: cycletime_velocity_controller

The object **cycletime_velocity_controller** returns the period of the velocity control loop in microseconds.

Sub-Index	B1$_h$
Description	cycletime_velocity_controller
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	µs
Value Range	--
Default Value	000000D0$_h$

6.17.1.10 Object 6510$_h$_B2$_h$: cycletime_position_controller

The object **cycletime_position_controller** returns the period of the position control loop in microseconds.

Sub-Index	B2$_h$
Description	cycletime_position_controller
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	µs
Value Range	--
Default Value	000001A0$_h$

6.17.1.11 Object 6510$_h$_B3$_h$: cycletime_trajectory_generator

The object **cycletime_trajectory_generator** returns the period of the positioning unit in microseconds.

Sub-Index	$B3_h$
Description	cycletime_tracectory_generator
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	µs
Value Range	--
Default Value	00000341_h

6.17.1.12 Object 6510$_h$_C0$_h$: commissioning_state

The parametrization program ™ uses the object **commisioning_state** to mark what kinds of parameters have already been adjusted. The default state of this object when delivered and after **restore_default_parameter** is zero. In this case the display of the servo controller shows an "A", to indicate that no suitable parametrization has been done. For a complete set-up via CANopen it is necessary to set at least one bit of this object to suppress the "A". Of course it is possible to use this object for own applications. In this case it has to be kept in mind that ™ uses this object, too.

Sub-Index	C0$_h$
Description	commisioning_state
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	--
Default Value	0

Bit	Description	Bit	Description
0	motor rated current valid	8	Current controller gain valid
1	max_current valid	9	Reserved
2	Number of poles valid	10	Conversion factors valid
3	encoder offset / direction valid	11	velocity controller valid
4	Reserved	12	position controller valid
5	hall encoder offset / direction valid	13	Safety parameter valid
		14	Reserved
6	Reserved	15	Polarity of home switch valid
7	Absolute position encoder valid	16...31	Reserved

Caution!

The object **commissioning_state** does not contain any information if the servo controller has been parametrised correctly according to the specific application. It will only be used to mark if the corresponding item has been parametrised at all.

"A" on 7 segment display

Note that at least one bit in the object **commissioning_state** has to be set, to suppress the "A" on the display of the servo controller.

6.18 Error management

6.18.1 Survey

The offer the possibility to change the error reaction of individual events, e.g. the occurance of a following error. Thus the controller reacts different, when a certain event occurs. Depending on the settings, the drive can be decelerated, the power stage will be disabled immediately or a warning is shown on the display.

For each event a manufacturer-specific minimum error reaction, which cannot be fallen below. In that way “critical” errors like 06 0 short circuit cannot be parametrised, because an immediate deactivation is necessary, in order to protect the servo controller from damages.

If an error reaction is set to a reaction, which is lower than the minimum allowed reaction for this error, then it is set to the minimum allowed error reaction automatically. A list with all error codes is available in the manual “”.

6.18.2 Description of Objects

6.18.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
2100_h	RECORD	error_management		ro
$2100_h\_01_h$	VAR	error_number	UINT8	rw
$2100_h\_02_h$	VAR	error_reaction_code	UINT8	rw

6.18.2.2 Object 2100_h: error_management

Index	2100_h
Name	error_management
Object Code	RECORD
No. of Elements	2

As of Firmware 3.2.0.1.1

The object **error_number** contains the main error code, which reaction must be changed. The main error code is normally displayed before the hyphen, (e.g. error 08-2, main error code 8). For possible error codes see hereunto chapter 5.5.

Sub-Index	01_h
Description	error_number
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	1 ... 96
Default Value	1

As of Firmware 3.2.0.1.1

In the object **error_reaction_code** the reaction to the error can be changed. If the manufacturer's minimum reaction is fallen below, then the reaction will be restricted to the minimum reaction. The actual adjusted reaction can be aquired via reading the object.

Sub-Index	02_h
Description	error_reaction_code
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0, 1, 3, 5, 7, 8
Default Value	depends on error_number

As of Firmware 3.2.0.1.1

Value	Meaning
0	No action
1	Entry in the buffer
3	Warning on the 7 segment display
5	Disable controller
7	Brake with maximum current
8	Disable power stage

6.18.2.3 Objekt 200F$_h$: last_warning_code

Warnings are mentionable events that will not stop the movement of the drive (e.g. following error). Warnings will be displayed on the 7- segment display of the servo controller and will disappear automatically.
The last occured warning can be read by this object. Bit 15 shows if the warning is still active.

Index	200F$_h$
Name	last_warning_code
Object Code	VAR
Data Type	UINT16

As of Firmware 3.5.x.1.1

Access	ro
PDO Mapping	yes
Units	--
Value Range	--
Default Value	--

Bit	Wert	Beschreibung
0... 3	000F$_h$	Sub code of warning
4... 11	0FF0$_h$	Main code of warning
15	8000$_h$	Warning is active

7 Device Control

7.1 State diagram (State machine)

7.1.1 Survey

The following chapter describes how to control the servo controller using CANopen, i.e. how to switch on the power stage or to reset an error.
Using CANopen the complete control of the servo is done by two objects. Via the **controlword** the host is able to control the servo, as the status of the servo can be read out of the **statusword**. The following items will be used in this chapter:

State:	The servo controller is in different states dependent on for instance if the power stage is alive or if an error has occurred. States defined under CANopen will be explained in this chapter. Example: SWITCH_ON_DISABLED
State Transition:	Just as the states it is defined as well how to move from one state to another (e.g. to reset an error). These state transitions will be either executed by the host by setting bits in the **controlword** or by the servo controller itself, if an error occurs for instance.
Command:	To initiate a state transition defined bit combinations have to be set in the **controlword**. Such bit combinations are called command. Example: Enable Operation
State diagram:	All the states and all state transitions together form the so called state diagram: A survey of all states and the possible transitions between two states.

7.1.2 The state diagram of the servo controller

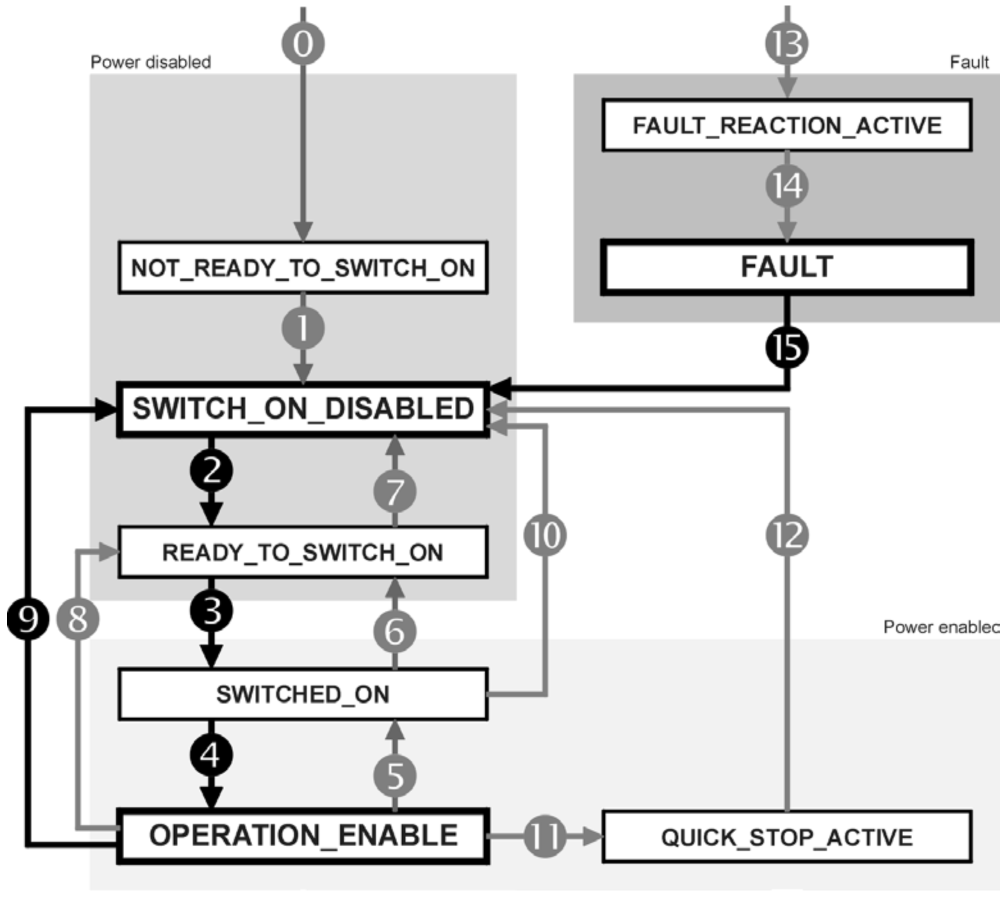


Figure 7.11: State diagram of the servo controller

The state diagram can be divided into three main parts: "Power Disabled" means the power stage is switched off and "Power Enabled" the power stage is live. The area "Fault" contains all states necessary to handle errors of the controller.

The most important states have been highlighted in the Figure: After switching on the servo controller initialises itself and reaches the state SWITCH_ON_DISABLED after all. In this state CAN communication is possible and the servo controller can be configured (e.g. the mode of operation can be set to "velocity control"). The power stage remains switched off and the motor shaft is freely rotatable. Through the state transitions 2, 3 and 4 – principally like the controller enable under CANopen - the state OPERATION_ENABLE will be reached. In this state the power stage is live and the servo controller controls the motor according to the configured mode of operation. Therefore previously ensure that the servo controller has been configured correctly and the according demand value is zero.

The state transition 9 complies with disabling the power stage, i.e. the motor is freely rotatable.

In case of a fault the servo controller branches independent of the current state lately to the state **FAULT**. Dependent on the seriousness of the fault several actions can be executed before, for instance an emergency stop (**FAULT_REACTION_ACTIVE**).

To execute the mentioned state transitions defined bit combinations have to be set in the **controlword**. To that the lower 4 bits of the **controlword** will be evaluated commonly. At first only the important transitions 2, 3, 4, 9 and 15 will be explained. A table of all possible transitions can be found at the end of this chapter.

The following chart contains the desired state transition in the 1st column. The 2nd column contains the condition for the transition (mostly a command by the host, here marked with a frame). How the command has to be built, i.e. what bits have to be set in the **controlword**, will be shown in the 3rd column (x = not relevant).

No.	Executed if	Bit combination (controlword)						Action
			Bit	3	2	1	0	
2	"Enable controller" + "Enable power stage" applying + Command $\boxed{\text{Shutdown}}$	$\boxed{\text{Shutdown}}$	=	x	1	1	0	None
3	Command $\boxed{\text{Switch On}}$	$\boxed{\text{Switch On}}$	=	x	1	1	1	Power stage will be switched on
4	Command $\boxed{\text{Enable Operation}}$	$\boxed{\text{Enable Operation}}$	=	1	1	1	1	Motor is controlled according to **modes_of_operation**
9	Command $\boxed{\text{Disable Voltage}}$	$\boxed{\text{Disable Voltage}}$	=	x	x	0	x	Power stage is disabled. The motor is freely rotatable
15	Cause of fault remedied + Command $\boxed{\text{Fault Reset}}$	$\boxed{\text{Fault Reset}}$	=	Bit 7 = ↑				Reset fault

Figure 7.12: Most important state transitions

EXAMPLE

After the servo controller has been configured it should be enabled, i.e. the power stage should be switched on:

1.) The servo is in the state **SWITCH_ON_DISABLED**.

2.) The state **OPERATION_ENABLE** should be reached.

3.) In accordance to the state diagram (Figure 7.11) the state transitions 2, 3 and 4 have to be executed.

4.) From

5.) Figure 7.12 follows:

Transition 2:	controlword = 0006_h	New state: READY_TO_SWITCH_ON *1)
Transition 3:	controlword = 0007_h	New state: SWITCHED_ON *1)
Transition 4:	controlword = $000F_h$	New state: OPERATION_ENABLE *1)

Hints:

1.) The example implies that no more bits in the **controlword** are set. (For the state transitions only the bits 0..3 are necessary).

2.) The state transitions 3 and 4 can be combined by setting the **controlword** to $000F_h$ directly. For the state transition 3 the set bit 3 is irrelevant.

*[1)] The host has to wait until the requested state can be read in the **statusword**. This will be explained more exact in the following chapter.

7.1.2.1 State diagram: States

In the following table all states and their meaning are listed:

Name	Meaning
NOT_READY_TO_SWITCH_ON	The servo controller executes its selftest. The CAN communication is not working
SWITCH_ON_DISABLED	The selftest has been completed. The CAN cimmunication is activated..
READY_TO_SWITCH_ON	The servo controller waits until the digital inputs "Enable controller" + "Enable power stage" are connected to 24V. (controller enable logic is set to “digital inputs and CAN”)
SWITCHED_ON *[1)]	The power stage is alive.
OPERATION_ENABLE *[1)]	The motor is under voltage and is controlled according to operational mode
QUICKSTOP_ACTIVE *[1)]	The **Quick Stop Function** will be executed (see: **quick_stop_option_code**). The motor is under voltage and is controlled according to the **Quick Stop Function**.
FAULT_REACTION_ACTIVE *[1)]	An error has occurred. On critical errors switching to state **Fault**. Otherwise the action according to the **fault_reaction_option_code** will be executed. The motor is under voltage and is controlled according to the **Fault Reaction Function**.
FAULT	An error has occurred. The power stage has been switched off.

*[1)] The power stage is alive

7.1.2.2 State diagram: State transitions

The following table lists all state transitions and their meaning:

No.	Executed if	Bit combination (controlword)						Action
			Bit	3	2	1	0	
0	"Power on" or Reset	internal transition						Execute selftest
1	Self test successful	internal transition						Activation of the CAN communication
2	"Enable controller" + "Enable power stage" applying + Command Shutdown	Shutdown	=	x	1	1	0	None

No.	Executed if	Bit combination (controlword)					Action
		Bit	3	2	1	0	
3	Command Switch On	Switch On =	x	1	1	1	Power stage will be switched on
4	Command Enable Operation	Enable Operation =	1	1	1	1	Motor is controlled according to operation mode
5	Command Disable Operation	Disable Operation =	0	1	1	1	Power stage is disabled. Motor is freely rotatable
6	Command Shutdown	Shutdown =	x	1	1	0	Power stage is disabled. Motor is freely rotatable
7	Command Quick Stop	Quick Stop =	x	0	1	x	
8	Command Shutdown	Shutdown =	x	1	1	0	Power stage is disabled. Motor is freely rotatable
9	Command Disable Voltage	Disable Voltage =	x	x	0	x	Power stage is disabled. Motor is freely rotatable
10	Command Disable Voltage	Disable Voltage =	x	x	0	x	Power stage is disabled. Motor is freely rotatable
11	Command Quick Stop	Quick Stop =	x	0	1	x	A braking according to quick_stop_option_code is started.
12	Braking has ended or Command Disable Voltage	Disable Voltage =	x	x	0	x	Power stage is disabled. Motor is freely rotatable
13	Error occurred	internal transition					On non-critical errors reaction according to fault_reaction_option_code. On critical error executing transition 14.
14	Error treating has ended	internal transition					Power stage is disabled. Motor is freely rotatable
15	Cause of fault remedied + Command Fault Reset	Fault Reset =	Bit 7 = ↑				Reset fault (Rising edge)

Power stage disabled

This means the transistors are not driven anymore. **If this state is reached on a rotating motor, the motor coasts down without being braked.** If a mechanical motor brake is available it will be locked.

Caution: This does not ensure that the motor is not under voltage.

Power stage enabled

This means the motor will be controlled according to the chosen mode of operation. If a mechanical motor brake is available it will be released. A defect or an incorrect parameter set-up (Motor current, number of poles, resolver offset angle, etc.) may cause an uncontrolled behaviour of the motor.

7.1.3 controlword

7.1.3.1 Object 6040$_h$: controlword

Via the **controlword** the state of the servo controller can be changed or a designated action (e.g. starting homing operation) can be executed directly. The meaning of the bits 4, 5, 6 and 8 depends on the actual operation mode (**modes_of_operation**), which will be explained in the chapter hereafter.

Index	6040$_h$
Name	controlword
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	0

Bit	Value	Function
0	0001_h	Initiating state transitions. (Bits will be evaluated commonly)
1	0002_h	
2	0004_h	
3	0008_h	
4	0010_h	new_set_point / start_homing_operation / enable_ip_mode
5	0020_h	change_set_immediatly
6	0040_h	absolute / relative
7	0080_h	reset_fault
8	0100_h	halt
9	0200_h	reserved set to 0
10	0400_h	reserved set to 0
11	0800_h	reserved set to 0
12	1000_h	reserved set to 0
13	2000_h	reserved set to 0
14	4000_h	reserved set to 0
15	8000_h	reserved set to 0

Table 7.1: Bit assignment of the controlword

As described detailed in the previous chapter the bits 0..3 are used to execute state transitions. The necessary commands are summarised in the following chart. The command Fault Reset will be executed on a rising edge of bit 7 (from 0 to 1).

command:	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0
	0080_h	0008_h	0004_h	0002_h	0001_h
Shutdown	×	×	1	1	0
Switch On	×	×	1	1	1
Disable Voltage	×	×	×	0	×
Quick Stop	×	×	0	1	×
Disable Operation	×	0	1	1	1
Enable Operation	×	1	1	1	1
Fault Reset	↑	×	×	×	×

Table 7.2: Survey of all commands (× = not relevant)

As some state transitions take time for processing, all changes written into the **controlword** have to read back from the **statusword**. Only when the requested status can be read in the **statusword**, one may write in further commands using the **controlword**.

Following the remaining bits of the **controlword** will be explained. The meaning of some bits depends on the actual operation mode (object **modes_of_operation**), i.e. if the controller will be torque or velocity controlled.

Bit 4	Depending on: **modes_of_operation:**
new_set_point	On **Profile Position Mode**: A rising edge signals that a new position parameter set should be taken over. In any case see chapter 8.3 as well.
start_homing_operation	On **Homing Mode**: A rising edge starts the configured search for reference. A falling edge stops the search immediately.
enable_ip_mode	On **Interpolated Position Mode**: This bit has to be set to evaluate the interpolation data. It will be acknowledged by the bit **ip_mode_active** in the **statusword**. In any case see chapter 8.4 as well.

Bit 5	change_set_immediatly	Only on **Profile Position Mode**:
		If this bit is cleared a current positioning order will be processed before starting a new one. If this bit is set a current positioning order will be interrupted by the new one. See also chapter 8.3.
Bit 6	relative	Only on **Profile Position Mode**:
		If this bit is set, the **target_position** of the current positioning job will be added to the **position_demand_value** of the position controller.
Bit 7	reset_fault	
		On a rising edge the servo controller tries to reset the present errors. This will only succeed if the cause of error has been remedied.
Bit 8		Depending on **modes_of_operation:**
	halt	On **Profile Position Mode**:
		If this bit is set the current positioning will be cancelled according to the object **profile_deceleration**. After stopping the bit **target_reached** (**statusword)** will be set. Resetting this bit has no effect.
	halt	On **Profile Velocity Mode**:
		If this bit is set the velocity will be reduced to zero according to the **profile_deceleration**. Resetting this bit will accelerate the motor again.
	halt	On **Profile Torque Mode**:
		If this bit is set the torque will be reduced to zero according to the **torque_slope**. Resetting this bit will accelerate the motor again
	halt	On **Homing mode**:
		If this bit is set the current homing operation will be cancelled and a homing error will be generated. Resetting this bit has no effect.

7.1.4 Reading the status of the servo controller

Similar to initiating several commands by setting bits of the **controlword**, the state of the servo controller can be read by specific bit combinations in the **statusword**.
The following chart lists all states of the state diagram and their respective bit combination occurring in the **statusword**.

State	Bit 6	Bit 5	Bit 3	Bit 2	Bit 1	Bit 0	Mask	Value
	0040_h	0020_h	0008_h	0004_h	0002_h	0001_h		
NOT_READY_TO_SWITCH_ON	0	×	0	0	0	0	$004F_h$	0000_h
SWITCH_ON_DISABLED	1	×	0	0	0	0	$004F_h$	0040_h
READY_TO_SWITCH_ON	0	1	0	0	0	1	$006F_h$	0021_h
SWITCHED_ON	0	1	0	0	1	1	$006F_h$	0023_h
OPERATION_ENABLE	0	1	0	1	1	1	$006F_h$	0027_h
QUICK_STOP_ACTIVE	0	0	0	1	1	1	$006F_h$	0007_h
FAULT_REACTION_ACTIVE	0	×	1	1	1	1	$004F_h$	$000F_h$
FAULT	0	×	1	1	1	1	$004F_h$	$000F_h$
FAULT (accord. DS402) 1)	0	×	1	0	0	0	$004F_h$	0008_h

Table 7.3: States of device (× = not relevant)

1):

In previous CANopen implementations the state FAULT has not been displayed according to DS 402. Therefore it is possible to changes this behaviour by setting Bit 7 of the object **compatibility_control** (see Chapter 6.2):
For compatibility to previous versions no changes have to be done and the numbers up to now can still be used.

EXAMPLE

The above mentioned example shows, what bits in the **controlword** have to be set to enable the servo controller. Now the requested state should be read out of the **statusword**:
Transition from **SWITCH_ON_DISABLED** to **OPERATION_ENABLE**:

1.) Write state transition 2 into the **controlword**.
2.) Wait until state **READY_TO_SWITCH_ON** occurs in the **statusword**.

Transition 2:	controlword = 0006_h	Wait until (statusword & $006F_h$) = 0021_h *1)

3.) The state transitions 3 and 4 can be written combined into the **controlword**.
4.) Wait, until the state **OPERATION_ENABLE** occurs in the **statusword**.

Transition 3+4: controlword = $000F_h$ Wait until (statusword & $006F_h$) = 0027_h *1)

Hint:

3.) The example implies, that no more bits in the **controlword** are set. (For the state transitions only the bits 0..3 are necessary).

*1)To identify a state also cleared bits have to be evaluated (see table). Therefore the **statusword** has to be masked properly.

7.1.5 statusword

7.1.5.1 Object 6041_h: statusword

Index	6041_h
Name	statusword
Object Code	VAR
Data Type	UINT16

Access	ro
PDO Mapping	yes
Units	--
Value Range	--
Default Value	--

Bit	Value	Name
0	0001_h	State of the servo controller (see Table 7.3) (These bits have to be evaluated commonly)
1	0002_h	
2	0004_h	
3	0008_h	
4	0010h	voltage_enabled
5	0020_h	State of the servo controller (see Table 7.3)
6	0040_h	
7	0080_h	warning
8	0100_h	drive_is_moving
9	0200_h	remote
10	0400_h	target_reached
11	0800_h	internal_limit_active
12	1000_h	set_point_acknowledge / speed_0 / homing_attained / ip_mode_active
13	2000_h	following_error / homing_error
14	4000_h	manufacturer_statusbit
15	8000_h	trigger_result

Table 7.4: Bit assignment of the statusword

All bits of the **statusword** are not buffered and therefore representing the actual state of the device.

In addition to the state of the device several informations can be read out directly of the **statusword**, i.e. every bit is assigned a specific event like a following error. The meaning of the bits is as follows:

Bit 4 **voltage_enabled**

This bit is set if the transistors of the power stage switched **OFF**.

If Bit 7 of object **6510_h_$F0_h$ (compatibility_control)** is set (see Chapter 6.2) $^{1)}$:

This bit is set if the transistors of the power stage switched **ON**.

CAUTION:
On a defect the motor may still be under voltage.

Bit 5	quick_stop	
		If this bit is cleared a **Quick Stop** will be executed according to the **quick_stop_option_code**.
Bit 7	warning	
		This bit is set if there is an inhibited rotating direction, because a limit switch has been activated. The setpoint inhibition is cleared, when the faults are reset. (see **controlword, fault_reset**)
Bit 8	drive_is_moving	*manufacturer specific*
		This bit is – independent of **modes_of_operation** – set, if the **velocity_actual_value** is <u>outside</u> the window determined by the object **velocity_threshold**.
Bit 9	remote	
		This bit indicates that the power stage can be enabled via the can bus. It is set if the object **enable_logic** is set accordingly.

1):

In previous CANopen implementations Bit 4 (**voltage_enabled**) has not been displayed according to DS 402. Therefore it is possible to changes this behaviour by setting Bit 7 of the object **compatibility_control** (see Chapter 6.2):
For compatibility to previous versions no changes have to be done and the numbers up to now can still be used.

Bit 10		Depends on **modes_of_operation:**
	target_reached	On **Profile Position Mode**: This bit will be set if the actual position (**position_ actual_value**) is within the configured position window (**position_window**). It will also be set if the motor stops after setting the bit **halt** in the **controlword**. It will be cleared if a new positioning is started.

	target_reached	On Profile Velocity Mode: The bit will be set if the actual velocity (velocity_actual_value) is within the configured velocity window (velocity_window, velocity_ window_time).
Bit 11	internal_limit_active	
		This bit indicates that the iit limitation is active.
Bit 12		Depends on modes_of_operation:
	set_point_acknowledge	On Profile Position Mode: This bit will be set to acknowledge the bit new_set_point in the controlword. It will be cleared if the bit new_set_point will be cleared. See chapter 8.3 as well.
	speed_0	On Profile Velocity Mode: This bit will be set if the velocity_actual_value is within the window determined by the object velocity_threshold.
	homing_attained	On Homing mode: This bit will be set if the homing operation has been finished without an error.
	ip_mode_active	On Interpolated Position Mode: This bit signals an active interpolation, i.e. interpolation data is evaluated. It will be set if requested by the bit enable_ip_mode in the controlword. In any case see chapter 8.4 as well.

Bit 13		Depends on **modes_of_operation:**
	following_error	On **Profile Position Mode**: This bit will be set if the **position_actual_value** differs from the **position_demand_value** so much that the difference is out of the tolerance window determined by the objects **following_error_window** and **following_ error_time_out**.
	homing_error	On **Homing Mode**: This bit will be set if a homing operation was cancelled by setting the bit **halt** in the **controlword**, if both limit switches are closed or the search for the switch exceeds the predefined positioning limits (**min_position_limit, max_position_limit**).
Bit 14	manufacturer_statusbit	*manufacturer specific*
		The meaning of this bit can be configured: It can be set if one or more user-defined bits of the manufacturer_statusword_1 will be set or reset. See Chapter 7.1.5.2 for details.
Bit 15	trigger_result	manufacturer specific
		The meaning of this bit can be configured: It is set, when a sample event has occured and the sample mask is set correspondingly. See herewith chapter 6.15.

7.1.5.2 Object 2000$_h$: manufacturer_statuswords

The record **manufacturer_statuswords** was introduced in order to represent controller states, which must not be contained in the cyclic polled **statusword**.

Index	2000$_h$
Name	manufacturer_statuswords
Object Code	RECORD
No. of Elements	1

As of Firmware 3.3.x.1.1

Sub-Index	01$_h$
Description	manufacturer_statusword_1
Data Type	UINT32
Access	ro
PDO Mapping	yes
Units	--
Value Range	--
Default Value	--

As of Firmware 3.3.x.1.1

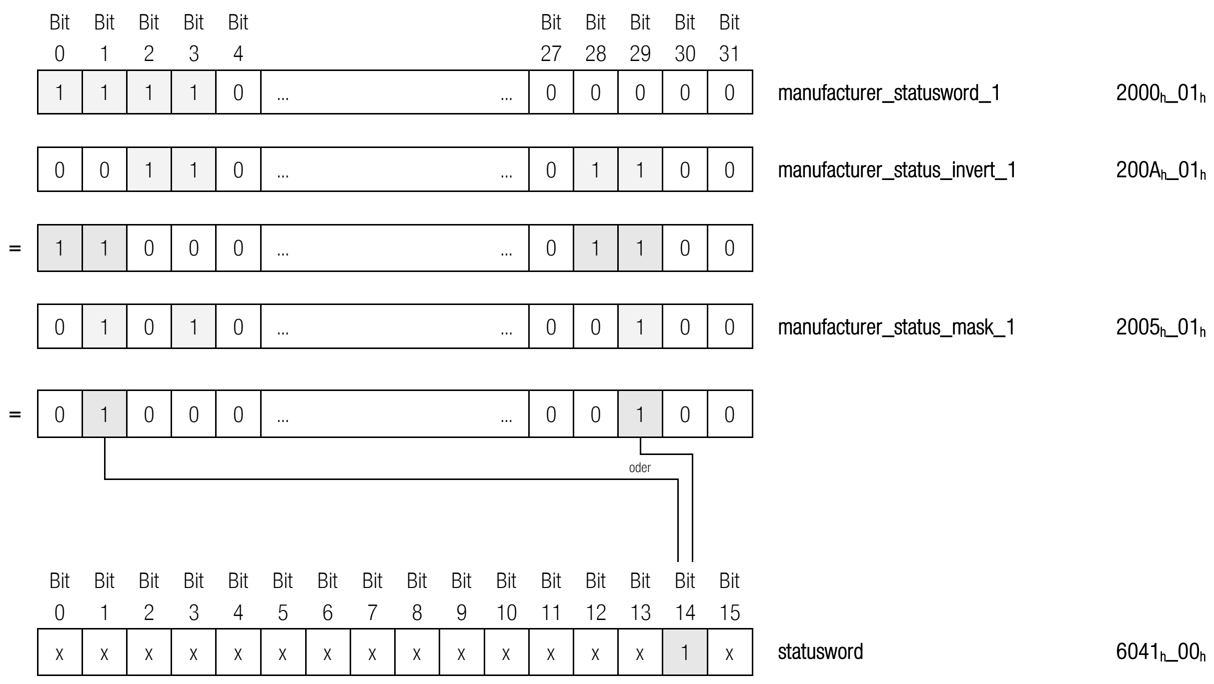
Bit	Value	Name	
0	00000001$_h$	is_referenced	As of Firmware 3.3.x.1.1
1	00000002$_h$	commutation_valid	As of Firmware 3.5.x.1.1
2	00000004$_h$	ready_for_enable	As of Firmware 3.5.x.1.1
...			
31	80000000$_h$	–	

Table 7.5: Bit assignment of manufacturer_statusword_1

Bit 0	is_referenced	
		This bit indicates, if the drive is referenced. The drive is referenced if a homing has been finished successfully or if there is no need of homing due to the connected encoder system (e.g. if an absolute angle encoder is used).
Bit 1	commutation_valid	
		This bit is set if the commutation information is valid. It is helpful if an encoder without commutation information is used (e.g. linearmotor), as the process for detection of commutation position may take several time. To avoid a timeout within a plc, this bit can be monitored.
Bit 2	ready_for_enab	
		This bit will be set if all other conditions for enabling the servo controller are available, except the *controller enable* itself. The following conditions are fulfilled if this bit it set: - The servo is error-free - The dc bus is charged completely - The encoder evaluation is ready. No process inhibiting enabling (e.g. serial communication) is active - No locking process is active (e.g. the process for detecting the motor parameter)

By means of **manufacturer_status_masks** and **manufacturer_status_invert** one or more bit of the **manufacturer_statuswords** can be mapped into bit 14 (**manufacturer_statusbit**) of the **statusword** (6041_h). All bits of the **manufacturer_statusword_1** can be previously inverted by the corresponding bit of the object **manufacturer_status_invert_1**. Thereby it is possible to check if a bit is reset. After inverting the bits, the bits are masked, i.e. only if the corresponding bit of the **manufacturer_status_mask_1** is set, this bit will furthermore take effect. If after masking at least one bit is set, bit 14 of the **statusword** will be set as well.

The following illustration exemplifies these objects:



Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 27 | Bit 28 | Bit 29 | Bit 30 | Bit 31
1 1 1 1 0 ... ... 0 0 0 0 0
manufacturer_statusword_1
2000h_01h
0 0 1 1 0 ... ... 0 1 1 0 0
manufacturer_status_invert_1
200Ah_01h
= 1 1 0 0 0 ... ... 0 1 1 0 0
0 1 0 1 0 ... ... 0 0 1 0 0
manufacturer_status_mask_1
2005h_01h
= 0 1 0 0 0 ... ... 0 0 1 0 0
oder
Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 | Bit 7 | Bit 8 | Bit 9 | Bit 10 | Bit 11 | Bit 12 | Bit 13 | Bit 14 | Bit 15
x x x x x x x x x x x x x x 1 x
statusword
6041h_00h

EXAMPLE

A) Bit 14 of the **statusword** should be set, if the drive is referenced. *is_referenced* is Bit 0 of the **manufacturer_statusword_1**

manufacturer_status_invert = 0x00000000
manufacturer_status_mask = 0x00000001 (bit 0)

B) Bit 14 of the **statusword** should be set, if the drive has no valid commutation information. *commutation_valid* is Bit 1 of the **manufacturer_statusword_1**. This bit has to be inverted, to be set, if the commutation information is invalid:

manufacturer_status_invert = 0x00000002 (bit 1)
manufacturer_status_mask = 0x00000002 (bit 1)

C) Bit 14 of the **statusword** should be set, if the drive is not ready for enabling OR the drive is referenced: *commutation_valid* is Bit 2 of the **manufacturer_statusword_1**. *is_referenced* is Bit 0. Bit 2 has to be inverted, to be set, if the drive is not ready for enabling:

manufacturer_status_invert = 0x00000004 (bit 2)
manufacturer_status_mask = 0x00000005 (bit 2, bit 0)

7.1.5.3 Objekt 2005$_h$: manufacturer_status_masks

This object record defines all bits of the **manufacturer_statuswords** that should be mapped into the **statusword**. See also chapter 7.1.5.2

Index	2005$_h$
Name	**manufacturer_status_masks**
Object Code	RECORD
No. of Elements	1

As of Firmware 3.5.x.1.1

Sub-Index	01$_h$
Description	**manufacturer_status_mask_1**
Data Type	UINT32
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	0x00000000

As of Firmware 3.5.x.1.1

7.1.5.4 Objekt 200A$_h$: manufacturer_status_invert

This object record defines all bits of the **manufacturer_statuswords** that should be mapped inverted into the **statusword**. See also chapter 7.1.5.2

Index	200A$_h$
Name	manufacturer_status_invert
Object Code	RECORD
No. of Elements	1

As of Firmware 3.5.x.1.1

Sub-Index	01$_h$
Description	manufacturer_status_invert_1
Data Type	UINT32
Access	rw
PDO Mapping	yes
Units	--
Value Range	--
Default Value	0x00000000

As of Firmware 3.5.x.1.1

7.1.6 Description of Objects

7.1.6.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$605B_h$	VAR	shutdown_option_code	INT16	rw
$605C_h$	VAR	disable_operation_option_code	INT16	rw
$605A_h$	VAR	quick_stop_option_code	INT16	rw
$605E_h$	VAR	fault_reaction_option_code	INT16	rw

7.1.6.2 Object $605B_h$: shutdown_option_code

The object **shutdown_option_code** determines the behaviour if the state transition 8 (from OPERATION ENABLE to READY TO SWITCH ON) will be executed. The object indicates the implemented behaviour of the controller and cannot be configured.

Index	$605B_h$
Name	shutdown_option_code
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	0
Default Value	0

Value	Name
0	Power stage will be switched off. Motor is freely rotatable.

7.1.6.3 Object 605C$_h$: disable_operation_option_code

The object **disable_operation_option_code** determines the behaviour if the state transition 5 (from OPERATION ENABLE to SWITCHED ON) will be executed. The object indicates the implemented behaviour of the controller and cannot be configured.

Index	605C$_h$
Name	disable_operation_option_code
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	-1
Default Value	-1

Value	Name
-1	Slow down motor with quickstop_deceleration.

7.1.6.4 Object 605A$_h$: quick_stop_option_code

The object **quick_stop_option_code** determines the behaviour if a **Quick Stop** will be executed. The object indicates the implemented behaviour of the controller and cannot be configured.

Index	605A$_h$
Name	quick_stop_option_code
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	2
Default Value	2

Value	Description
2	Slow down motor with quickstop_deceleration.

7.1.6.5 Object 605E$_h$: fault_reaction_option_code

The object **fault_reaction_option_code** determines the behaviour on a fault. Because the reaction on errors depends on the type of error in the , this object can not be configured. It always returns the value zero. The fault reaction can be separately configured for each fault group as described in chapter 6.18, Error management.

.

Index	605E$_h$
Name	fault_reaction_option_code
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	no
Units	--
Value Range	0
Default Value	0

8 Operating Modes

8.1 Adjustment of the Operating Mode

8.1.1 Survey

The are able to work in a lot of different operation modes. Only some of them are specified in detail in the CANopen specification:

- torque controlled operation
- speed controlled operation
- homing operation (search for reference)
- positioning operation
- interpolated position mode

8.1.2 Description of Objects

8.1.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
6060_h	VAR	modes_of_operation	INT8	wo
6061_h	VAR	modes_of_operation_display	INT8	ro

8.1.2.2 Object 6060_h: modes_of_operation

The operating mode of the servo controller is determined by the object **modes_of_operation**.

Index	6060_h
Name	modes_of_operation
Object Code	VAR
Data Type	INT8

Access	rw
PDO Mapping	yes
Units	--
Value Range	1, 3, 4, 6, 7
Default Value	--

Value	Operation mode
1	Profile Positioning Mode (position controller with positioning operation)
3	Profile Velocity Mode (speed controller with setpoint ramp)
4	Torque Profile Mode (torque controller with setpoint ramp)
6	Homing mode (homing operation)
7	Interpolated Position Mode

The current operating mode can only be read in the object **modes_of_operation_display**. As a change of the operating mode might require some time to process, one will have to wait until the new selected mode appears in the object **modes_of_operation_display**.

8.1.2.3 Object 6061$_{h:}$ modes_of_operation_display

The current operating mode of the servo controller can be read in the object **modes_of_operation_display**. If an operating mode is set up via the object **6060h**, in addition to the operating mode the setpoint selectors, which are necessary for the operation of the controller via CANopen, are changed too. These are:

	Profile Velocity Mode	Profile Torque Mode
Selector A	Speed setpoint (Fieldbus 1)	Torque setpoint (Fieldbus 1)
Selector B	If necessary: Torque limitation	Inactive
Selector C	Speed setpoint (Synchronous speed)	Inactive

Furthermore the setpoint ramp is activated principally. Only if the selection is set up in the mentioned manner, one of the CANopen operating modes will be returned. If these settings are changed, e.g. via the ™, a particular "user" operating mode will be returned, in order to indicate, that the selectors have been changed.

Index	6061$_h$
Name	modes_of_operation_display
Object Code	VAR
Data Type	INT8

Access	ro
PDO Mapping	yes
Units	--
Value Range	-1, 1, 3, 4, 6, 7
Default Value	3

Value	Operation mode
-1	Unknown operating mode under CANopen
-11	User Position Mode
-13	User Velocity Mode
-14	User Torque Mode
1	Profile Positioning Mode (position controller with positioning operation)
3	Profile Velocity Mode (speed controller with setpoint ramp)
4	Torque Profile Mode (torque controller with setpoint ramp)
6	Homing mode (homing operation)
7	Interpolated Position Mode

The operating mode can only be set via the object **modes_of_operation**. As a change of the operating mode might require some time, it has to be wait until the new selected mode appears in the object **modes_of_operation_display**. During this period of time it could happen that invalid operating modes (-1) are displayed for a short time.

8.2 Operating Mode »Homing mode«

8.2.1 Survey

This chapter describes how the servo controller searches the start position (also called reference point or zero point). There are various methods to determine this position. Either the limit switches at the end of the positioning range can be used or a reference switch (zero point switch) within the possible range of motion. Among some methods the zero impulse of the used encoder (resolver, incremental encoder, etc.) can be included to achieve a state that can be reproduced as good as possible.

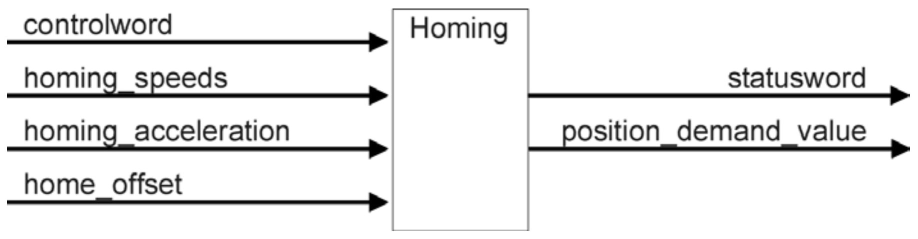


Figure 8.1: Homing Mode

The user can determine the velocity, acceleration, and the kind of homing operation. After the servo controller has found its reference the zero position can be moved to the desired point via the object **home_offset**. There are two kinds of speed for the homing operation.

The higher search speed (**speed_during_search_for_switch**) is used to find the limit switch respectively the reference switch. To determine the reference slope exactly a lower speed is used (**speed_during_search_for_zero**).

If just the position should be moved instead of finding a reference point, object **2030**$_h$ (**set_absolute_position**) can be used. For this purposes see chapter 6.7.2.15.

The movement to the zero position is in most cases not part of the homing operation. If all required values are known (i.e. if the zero position is already known by the servo controller), no physical motion will be executed. This behaviour can be changed by object **6510**$_h$**_F0**$_h$ (**compatibility_control**, see chapter 6.2.2.2), that the drive always moves to zero after completing the homing procedure successfully.

8.2.2 Description of Objects

8.2.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attribute
$607C_h$	VAR	home_offset	INT32	rw
6098_h	VAR	homing_method	INT8	rw
6099_h	ARRAY	homing_speeds	UINT32	rw
$609A_h$	VAR	homing_acceleration	UINT32	rw
2045_h	VAR	homing_timeout	UINT16	rw

8.2.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	UINT16	6.18 Device control
6041_h	VAR	statusword	UINT16	6.18 Device control

8.2.2.3 Object $607C_h$: home_offset

The object **home_offset** determines the displacement of the zero position to the limit resp. reference switch position.

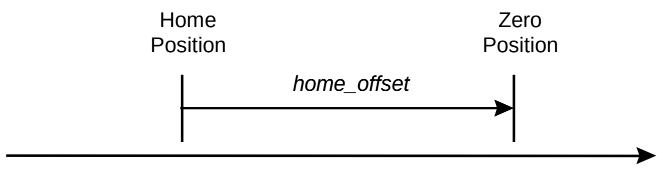


Figure 8.2: Home Offset

Index	607C$_h$
Name	home_offset
Object Code	VAR
Data Type	INT32

Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	0

8.2.2.4 Object 6098$_h$: homing_method

A number of different methods are available for a homing operation. The method that is necessary for the application can be selected via the object **homing_method**. There are four possible signals for the homing operation: The negative and positive limit switch, the reference switch and the (periodic) zero impulse of the angle encoder. Besides this the controller can refer to the negative or positive endstop without additional signal. If a method has been determined via the object **homing_method** the following parameters are fixed by that:

- The signal for reference (neg./pos. limit switch, reference switch, neg. / pos. endstop)
- The direction and process of the homing operation
- The kind of evaluation of the zero impulse of the used angle encoder.

Index	6098$_h$
Name	homing_method
Object Code	VAR
Data Type	INT8

Access	rw
PDO Mapping	yes
Units	--
Value Range	-18, -17, -2, -1, 1, 2, 7, 11, 17, 18, 23, 27, 32, 33, 34, 35
Default Value	17

Value	Direction	Target	Reference point for Home position	*DS402*
-18	Positive	Endstop	Endstop	*-18*
-17	Negative	Endstop	Endstop	*-17*

-2	Positive	Endstop	Zero impulse	*-2*
-1	Negative	Endstop	Zero impulse	*-1*
1	Negative	Limit switch	Zero impulse	*1*
2	Positive	Limit switch	Zero impulse	*2*
7	Positive	Reference switch	Zero impulse	*7*
11	Negative	Reference switch	Zero impulse	*11*
17	Negative	Limit switch	Limit switch	*17*
18	Positive	Limit switch	Limit switch	*18*
23	Positive	Reference switch	Reference switch	*23*
27	Negative	Reference switch	Reference switch	*27*
32	Negative	Zero impulse	Zero impulse	*33*
33	Positive	Zero impulse	Zero impulse	*34*
34		No run	Actual position	*35*

In previous CANopen implementations the homing methods 32, 33, 34 and 35 are not assigned according to DS402. Therefore there is the possibility to choose the assignment according to DS402 via the object **compatibility_control** (see 6.2). In this case the methods' numbers, printed in italic, should be used.
For compatibility to previous versions no changes have to be done and the numbers up to now can still be used.

The **homing_method** can only be changed, when the homing is not active. Otherwise the error message Data cannot be transferred or stored to the application because of the present device state is returned.
The homing sequence of the respective methods is explained more detailed in chapter 8.2.3.

8.2.2.5 Object 6099$_h$: homing_speeds

This object determines the speeds which are used during the homing operation.

Index	6099$_h$
Name	**homing_speeds**
Object Code	ARRAY
No. of Elements	2
Data Type	UINT32

Sub-Index	01_h
Description	speed_during_search_for_switch
Access	rw
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	100 min^{-1}

Sub-Index	02_h
Description	speed_during_search_for_zero
Access	rw
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	10 min^{-1}

The drive moves to zero after completing the homing procedure successfully, if bit 6 of the object **compatibility_control** (see chapter 6.2.2.2) is set.
If this bit is set, writing to object **speed_during_search_for_switch** will set the velocity for searching the switch as well as the velocity for moving to the zero position.

8.2.2.6 Object 609A$_h$: homing_acceleration

The objects **homing_acceleration** determine the acceleration which is used for all acceleration and deceleration operations during the search for reference.

Index	609A$_h$
Name	homing_acceleration
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	acceleration units
Value Range	--
Default Value	1000 min^{-1} / s

8.2.2.7 Object 2045$_h$: homing_timeout

The homing's maximum execution time can be monitored. For that purpose the maximum execution time is specified by the object **homing_timeout**. If this time is exceeded and the homing has not been finished yet, error 11-3 will occur.

As of Firmware 3.2.0.1.1

Index	2045$_h$
Name	homing_timeout
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	no
Units	ms
Value Range	0 (off), 1 ... 65535
Default Value	60000

8.2.3 Homing sequences

The various homing sequences are pictured in the following figures. The encirceled numbers correspond to the number of the object homing_method.

8.2.3.1 Method 1: Negative limit switch using zero impulse evaluation

If this method is used the drive first moves relatively quick into the negative direction until it reaches the negative limit switch. This is displayed in the diagram by the rising edge. Afterwards the drive slowly returns and searches for the exact position of the limit switch. The zero position refers to the first zero impulse of the angle encoder in positive direction from the limit switch.

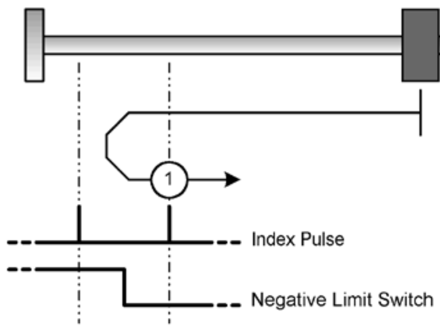


Figure 8.3: Homing operation to the negative limit switch including evaluation of the zero impulse

8.2.3.2 Method 2: Positive limit switch using zero impulse evaluation

If this method is used the drive first moves relatively quick into the positive direction until it reaches the positive limit switch. This is displayed in the diagram by the rising edge. Afterwards the drive slowly returns and searches for the exact position of the limit switch. The zero position refers to the first zero impulse of the angle encoder in negative direction from the limit switch.

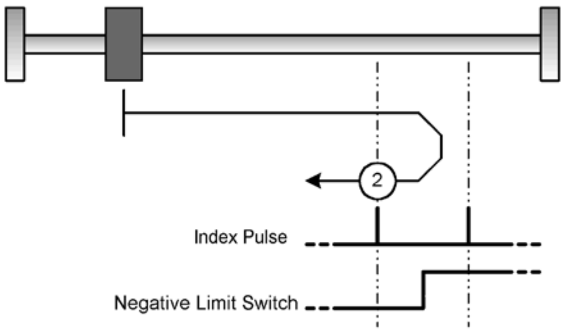


Figure 8.4: Homing operation to the positive limit switch including evaluation of the zero impulse

8.2.3.3 Methods 7 and 11: Reference switch and zero impulse evaluation

These two methods use the reference switch which is only active over parts of the distance. These reference methods are particularly useful for round-axis applications where the reference switch is activated once per revolution.
In case of method 7 the drive first moves into positive and in case of method 11 into negative direction. Depending on the direction of the motion the zero position refers to the first zero impulse in negative or positive direction from the reference switch. This can be seen in the two following diagrams.

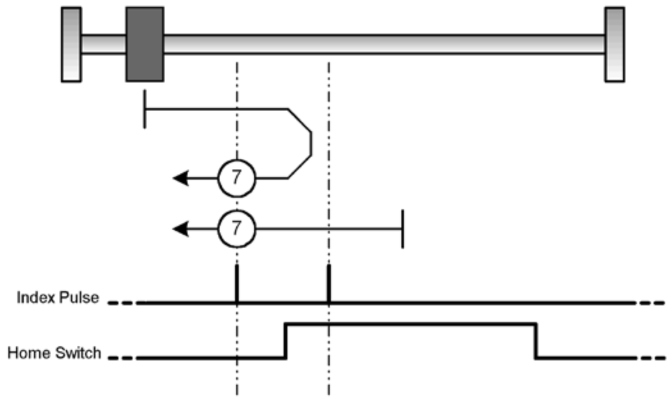


Figure 8.5: Homing operation to the reference switch evaluating the zero impulse for a positive start motion

On homing to the reference switch the first reached limit switch is used to reverse the search direction. If therupon the opposite limit switch is reached an error occurs.

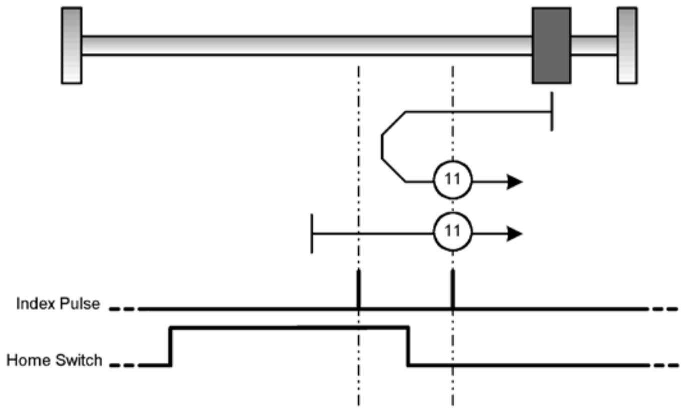


Figure 8.6: Homing operation to the reference switch evaluating the zero impulse for a negative start motion

8.2.3.4 Method 17: Homing operation to the negative limit switch

If this method is used the drive first moves relatively quick into the negative direction until it reaches the negative limit switch. This is displayed in the diagram by the rising edge. Afterwards the drive slowly returns and searches for the exact position of the limit switch. The zero position refers to the descending edge from the negative limit switch.

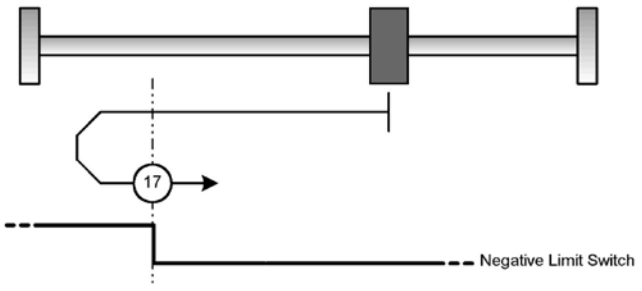


Figure 8.7: Homing operation to the negative limit switch

8.2.3.5 Method 18: Homing operation to the positive limit switch

If this method is used the drive first moves relatively quick into the positive direction until it reaches the positive limit switch. This is displayed in the diagram by the rising edge. Afterwards the drive slowly returns and searches for the exact position of the limit switch. The zero position refers to the descending edge from the positive limit switch.

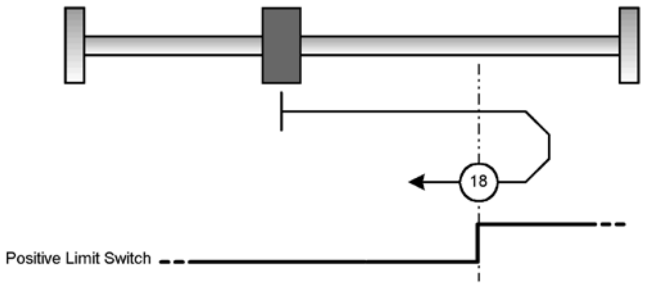


Figure 8.8: Homing operation to the positive limit switch

8.2.3.6 Methods 23 and 27: Homing operation to the reference switch

These two methods use the reference switch which only is active over part of the distance. These reference methods are particularly useful for round-axis applications where the reference switch is activated once per revolution.

In case of method 23 the drive first moves into positive and in case of method 27 into negative direction. The zero position refers to the edge from the reference switch. This can be seen in the two following diagrams.

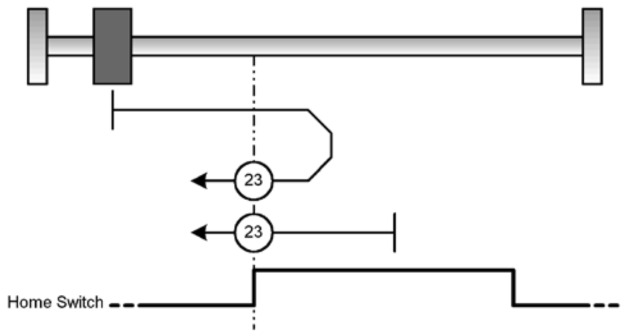


Figure 8.9: Homing operation to the reference switch for a positive start motion

On homing to the reference switch the first reached limit switch is used to reverse the search direction. If therupon the opposite limit switch is reached an error occurs.

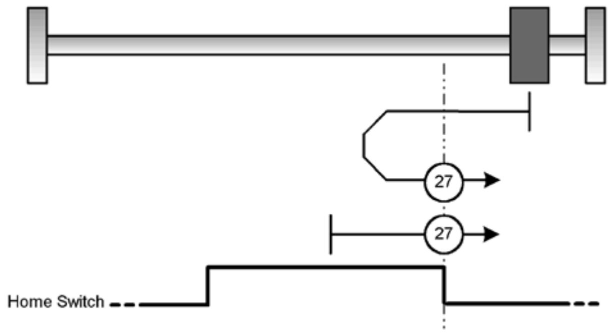


Figure 8.10: Homing operation to the reference switch for a negative start motion

8.2.3.7 Method -1: Negative stop evaluating the zero impulse

If this method is used the drive moves into negative direction until it reaches the stop. The I²t integral of the motor reaches a maximum value of 90%. The stop has to be mechanically dimensioned so that it is not damaged in case of the configured maximum current. The zero position refers to the first zero impulse of the angle encoder in positive direction from the stop.

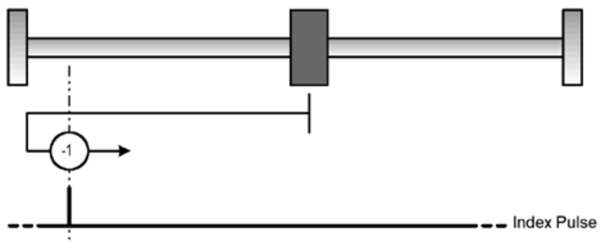


Figure 8.11: Homing operation to the negative stop evaluating the zero impulse

8.2.3.8 Method -2: Positive stop evaluating the zero impulse

If this method is used the drive moves into positive direction until it reaches the stop. The I²t integral of the motor reaches a maximum value of 90%. The stop has to be mechanically dimensioned so that it is not damaged in case of the configured maximum current. The zero position refers to the first zero impulse of the angle encoder in negative direction from the stop.

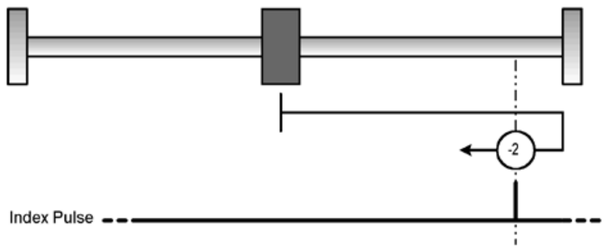


Figure 8.12: Homing operation to the positive stop evaluating the zero impulse

8.2.3.9 Method -17: Homing operation to the negative stop

If this method is used the drive moves into negative direction until it reaches the stop. The I^2t integral of the motor reaches a maximum value of 90%. The stop has to be mechanically dimensioned so that it is not damaged in case of the configured maximum current. The zero position refers directly to the endstop.

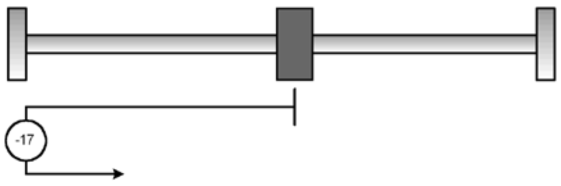


Figure 8.13: Homing operation to the negative stop

8.2.3.10 Method -18: Homing operation to the positive stop

If this method is used the drive moves into positive direction until it reaches the stop. The I^2t integral of the motor reaches a maximum value of 90%. The stop has to be mechanically dimensioned so that it is not damaged in case of the configured maximum current. The zero position refers directly to the endstop.

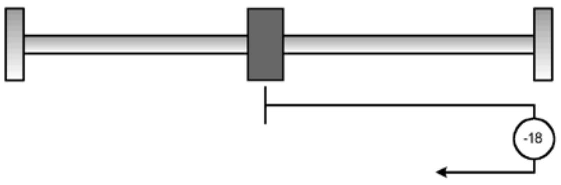


Figure 8.14: Homing operation to the positive stop

8.2.3.11 Methods 32 and 33: Homing operation to the zero impulse

For the methods 32 (*33 according to DS402*) and 33 (*34 according to DS402*) the direction of the homing operation is negative and positive, respectively. The zero position refers to the first zero impulse from the angle encoder in search direction.

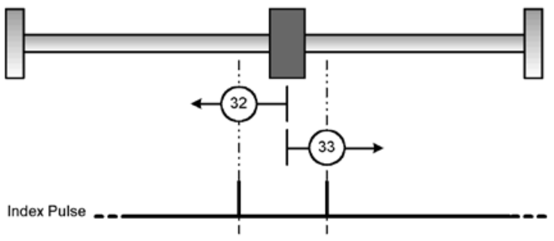


Figure 8.15: Homing operation only referring to the zero impulse

8.2.3.12 Method 34: Homing operation to the current position

On method 34 (*35 according to DS402*) the zero position is referred to the current position.
If just the position should be moved instead of finding a reference point, object **2030$_h$** (**set_absolute_position**) can be used. For this purposes see chapter 6.7.2.15.

8.2.4 Control of the homing operation

The homing operation is started by setting bit 4 in the **controlword**. The successful end of a homing operation is indicated by a set bit 12 in the object **statusword**. A set bit 13 in the object **statusword** indicates that an error has occurred during the homing operation. The error reason can be identified by the objects **error_register** and **predefined_error_field**.

Bit 4	Description
0	Homing operation is not active
$0 \rightarrow 1$	Start homing operation
1	Homing operation is active
$1 \rightarrow 0$	Interrupt homing operation

Table 8.1: Description of the bits in the controlword

Bit 13	Bit 12	Desription
0	0	Homing operation has not yet finished
0	1	Homing operation executed successfully
1	0	Homing operation not executed successfully
1	1	Illegal state

Table 8.2: Description of the bits in the statusword

8.3 Operating Mode »Profile Position Mode«

8.3.1 Survey

The structure of this operating mode is shown in Figure 8.16:

The target position (**target_position**) is passed to the trajectory generator. This generator generates a desired position value (**position_demand_value**) for the position controller that is described in the chapter **Position Controller** (position control function, chapter 0). These two function blocks can be adjusted independently from each other.

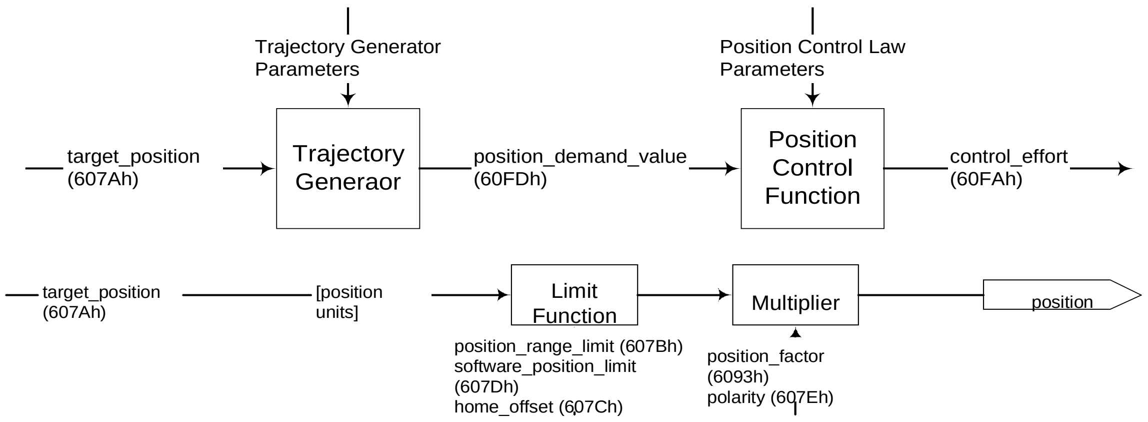


Figure 8.16: Trajectory generator and position controller

All input quantities of the trajectory generator are converted into internal quantities of the controller by means of the quantities of the **Factor group** (see chapter 6.3: Conversion factors (Factor Group)). The internal quantities are marked by an asterisk and are not imperatively needed by the user.

8.3.2 Description of Objects

8.3.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$607A_h$	VAR	target_position	INT32	rw
6081_h	VAR	profile_velocity	UINT32	rw
6082_h	VAR	end_velocity	UINT32	rw
6083_h	VAR	profile_acceleration	UINT32	rw
6084_h	VAR	profile_deceleration	UINT32	rw
6085_h	VAR	quick_stop_deceleration	UINT32	rw
6086_h	VAR	motion_profile_type	INT16	rw

8.3.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	INT16	6.16 Device control
6041_h	VAR	statusword	UINT16	6.16 Device control
$605A_h$	VAR	quick_stop_option_code	INT16	6.16 Device control
$607E_h$	VAR	polarity	UINT8	6.3 Conversion factors (Factor Group)
6093_h	ARRAY	position_factor	UINT32	6.3 Conversion factors (Factor Group)
6094_h	ARRAY	velocity_encoder_factor	UINT32	6.3 Conversion factors (Factor Group)
6097_h	ARRAY	acceleration_factor	UINT32	6.3 Conversion factors (Factor Group)

8.3.2.3 Object $607A_h$: target_position

Das Object **target_position** (Zielposition) bestimmt, an welche Position der AntriebsThe object **target_position** determines the destination the servo controller moves to. For this purpose the current adjustments of the velocity, of the acceleration, of the deceleration and the kind of motion profile (**motion_profile_type**) have to be considered. The target position (**target_position**) is interpreted either as an absolute or relative position. This depends on bit 6 (**relative**) of the object **controlword**.

Index	$607A_h$
Name	target_position
Object Code	VAR
Data Type	INT32

Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	0

8.3.2.4 Object 6081$_h$: profile_velocity

The object **profile_velocity** specifies the speed that usually is reached during a positioning motion at the end of the acceleration ramp. The object **profile_velocity** is specified in **speed_units**.

Index	6081$_h$
Name	profile_velocity
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	speed_units
Value Range	--
Default Value	1000

8.3.2.5 Object 6082$_h$: end_velocity

The object **end_velocity** defines the speed at the target position (**target_position**). Usually this object has to be set to zero so that the controller stops when it reaches the target position. For gapless sequences of positionings a value unequal zero can be set. The object **end_velocity** is specified in **speed_units** like the object **profile_velocity**.

Index	6082$_h$
Name	end_velocity
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	0

8.3.2.6 Object 6083$_h$: profile_acceleration

The object **profile_acceleration** determines the maximum acceleration used during a positioning motion. It is specified in user specific acceleration units (**acceleration_units**). (See 6.3 Conversion factors (Factor Group)).

Index	6083$_h$
Name	profile_acceleration
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	acceleration units
Value Range	--
Default Value	10000 min^{-1} /s

8.3.2.7 Object 6084$_h$: profile_deceleration

The object **profile_deceleration** specifies the maximum deceleration used during a positioning motion. This object is specified in the same units as the object **profile_acceleration**. (See chapter 6.3 Conversion factors (Factor Group)).

Index	6084$_h$
Name	profile_deceleration
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	acceleration units
Value Range	--
Default Value	10000 min^{-1} /s

8.3.2.8 Object 6085$_h$: quick_stop_deceleration

The object **quick_stop_deceleration** determines the deceleration if a Quick Stop will be executed (see chapter 7.1.2.2). The object **quick_stop_deceleration** is specified in the units as the object **profile_deceleration**.

Index	6085$_h$
Name	quick_stop_deceleration
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	acceleration units
Value Range	--
Default Value	14100 min^{-1} /s

8.3.2.9 Object 6086$_h$: motion_profile_type

The object **motion_profile_type** is used to select the kind of positioning profile. At present only a linear profile is available.

Index	6086$_h$
Name	motion_profile_type
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	0; 2
Default Value	0

Value	Profile	
0	Linear ramp	
2	Jerkfree ramp	As of Firmware 3.1.0.1.1

8.3.3 Functional Description

Two different ways to apply target positions to the servo controller are supported.

Single setpoints

After reaching the **target_position** the servo controller signals this status to the host by the bit **target_reached** (Bit 10 of **controlword**) and then receives a new setpoint. The servo controller stops at the **target_position** before starting a move to the next setpoint.

Set of setpoints

After reaching the **target_position** the servo controller immediately processes the next **target_position** which results in a move where the velocity of the drive normally is not reduced to zero after reaching a setpoint.

These two methods are controlled by the bits **new_set_point** and **change_set_immediately** in the object **controlword** and **set_point_acknowledge** in the object **statusword**. These bits are in a request-response relationship. So it is possible to prepare one positioning job while another job is still running.

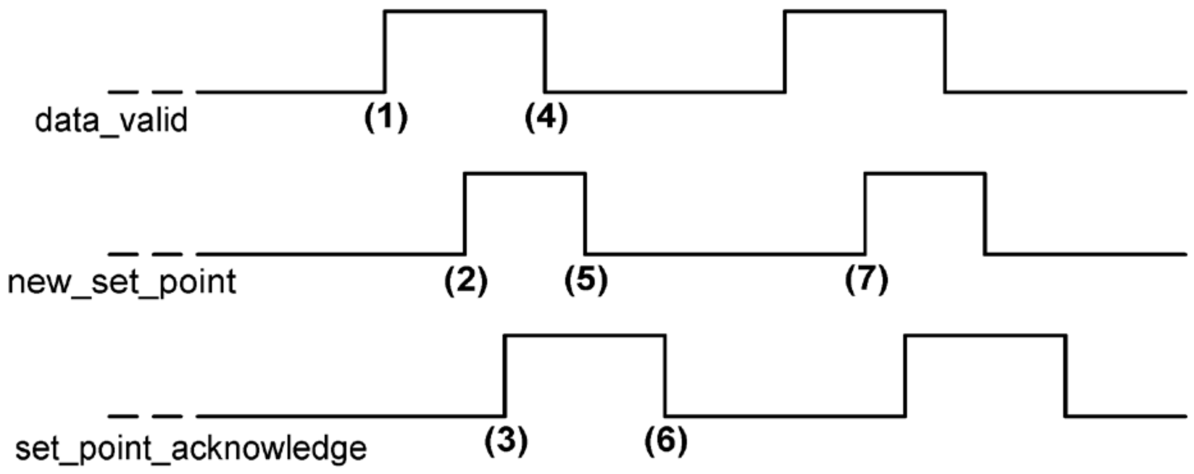


Figure 8.17: Positioning job transfer from a host

Figure 8.17 shows the communication between the host and the servo controller via the CAN bus:

At first the positioning data (**target_position, profile_velocity, end_velocity** and **profile_acceleration**) are transferred to the servo controller. After the positioning data set has been transferred completely (1) the host can start the positioning motion by setting the bit **new_set_point** in the **controlword** (2).

This will be acknowledged by the servo controller by setting the bit **set_point_acknowledge** in the **statusword** (3), when the positioning data has been copied into the internal buffer.
Afterwards the host can start to transfer a new positioning data set into the servo controller (4) and clear the bit **new_set_point** (5). The servo controller signals by a cleared **set_point_acknowledge** bit that it can accept a new drive job (6). The host has to wait for the falling edge of the bit **set_point_acknowledge** before a new positioning motion can be started (7).

In Figure 8.18 a new positioning motion is started after the previous one has been finished completely. For that purpose the host evaluates the bit **target_reached** in the object **statusword**.

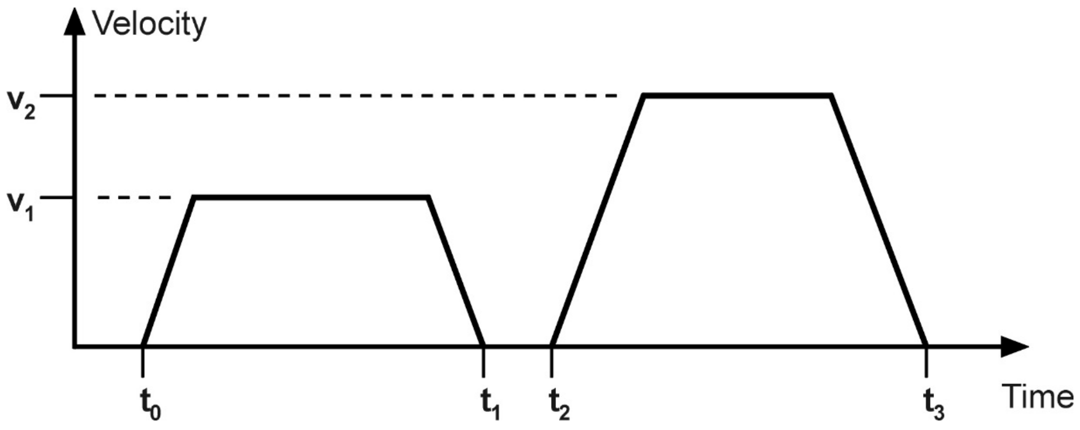


Figure 8.18: Simple positioning job

In Figure 8.19 a new positioning motion has already been started while the previous motion was still running. The host already transfers the subsequent target to the servo controller if it signals by a cleared **set_point_acknowledge** bit that it has read the buffer and started the corresponding positioning motion. In this way the positioning motions are joined together gaplessly. For this operating mode the object **end_velocity** of the first job should be configured to the same value as the object **profile_velocity** of the following job so that the servo controller does not decelerate to zero amongst the single positioning motions.

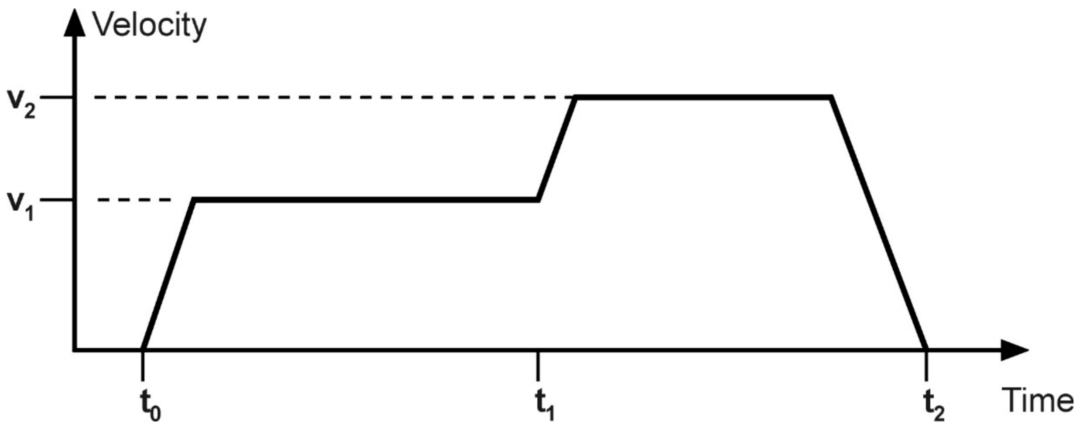


Figure 8.19: Gapless sequence of positioning jobs

If beside the bit **new_setpoint** the bit **change_set_immediately** is set in the **controlword**, too, the new positioning job will interrupt the actual job immediately and will be started instead. The actual positioning job is canceled in this case.

8.4 Interpolated Position Mode

8.4.1 Survey

The interpolated position mode (**IP**) allows cyclic sending of position demand values to the servo in a multi-axle system. Therefore the host sends synchronisation telegrams (SNYC) and position demand values in a fixed interval (synchronisation interval). The servo controller itself interpolates between two setpoints, if the synchronisation interval is larger than the position control interval as shown in the following figure:

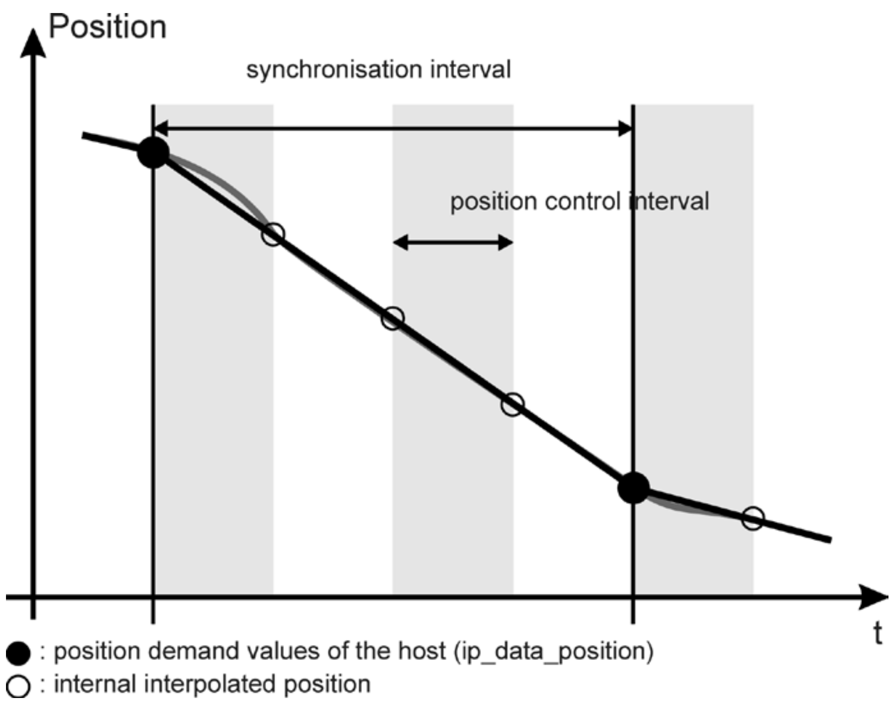

Position
synchronisation interval
position control interval
t
● : position demand values of the host (ip_data_position)
○ : internal interpolated position

Figure 8.20: Linear interpolation between two positions

In the following the objects of the **interpolated position mode** will be described first. After it a functional description will explain the activation and the order of parameterisation detailed.

8.4.2 Description of Objects

8.4.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
$60C0_h$	VAR	interpolation_submode_select	INT16	rw
$60C1_h$	REC	interpolation_data_record		rw
$60C2_h$	REC	interpolation_time_period		rw
$60C3_h$	VAR	interpolation_sync_definition	UINT8	rw
$60C4_h$	REC	interpolation_data_configuration		rw

8.4.2.2 Affected objects of other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	INT16	6.18 Device control
6041_h	VAR	statusword	UINT16	6.18 Device control
6093_h	ARRAY	position_factor	UINT32	6.3 Conversion factors (Factor Group)
6094_h	ARRAY	velocity_encoder_factor	UINT32	6.3 Conversion factors (Factor Group)
6097_h	ARRAY	acceleration_factor	UINT32	6.3 Conversion factors (Factor Group)

8.4.2.3 Object $60C0_h$: interpolation_submode_select

The obejct **interpolation_submode_select** determines the type of interpolation. At present only the manufacturer specific type „Linear interpolation without buffer“ is available.

Index	60C0$_h$
Name	interpolation_submode_select
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	-2
Default Value	-2

Value	Type of interpolation
-2	Linear interpolation without buffer

8.4.2.4 Object 60C1$_h$: interpolation_data_record

The object record **interpolation_data_record** represents the interpolation data itself. It contains the position demand value (**ip_data_position**) and a controlword (**ip_data_controlword**) that determines whether the position value is relative or absolute. The use of the controlword is optional. If it should be used it is neccessary to write subindex 2 first (**ip_data_controlword**) followed by subindex 1 (**ip_data_position**) to achieve data consistence, because the position will be copied by a write access to **ip_data_position**.

Index	60C1$_h$
Name	interpolation_data_record
Object Code	RECORD
No. of Elements	2

Sub-Index	01$_h$
Description	ip_data_position
Data Type	INT32
Access	rw
PDO Mapping	yes
Units	position units
Value Range	--
Default Value	--

Sub-Index	02$_h$
Description	ip_data_controlword
Data Type	UINT8
Access	rw
PDO Mapping	yes
Units	--
Value Range	0, 1
Default Value	0

Value	ip_data_position is
0	Absolute position
1	Relative distance

The data will be copied on a write access to subindex 1. If also subindex 2 should be used it is neccessary to write subindex 2 first followed by subindex 1 .

8.4.2.5 Object 60C2$_h$: interpolation_time_period

Using the object record **interpolation_time_period** the synchronisation interval can be determined. First the unit (ms or 1/10 ms) can be set by the object **ip_time_index**. After that the interval can be written to **ip_time_units**.

For the synchronisation the complete controller loops (Current-, velocity- and position controller) will be synchronised to the external clock. Therefore the change of the interval will only take effect after a reset. If the synchronisation interval should be changed via CAN bus, it is necessary to save the parameter set first (see chapter 0), and reset the device after that (see chapter 5.6). The external clock has to have a high precision.

Index	60C2$_h$
Name	**interpolation_time_period**
Object Code	RECORD
No. of Elements	2

Sub-Index	01$_h$
Description	**ip_time_units**
Data Type	UINT8
Access	rw
PDO Mapping	yes
Units	according to ip_time_index

Value Range	ip_time_index = -3: 1, 2,..., 9, 10
	ip_time_index = -4: 10, 20,..., 90, 100

Default Value	--

Sub-Index	02$_h$
Description	**ip_time_index**
Data Type	INT8
Access	rw
PDO Mapping	yes
Units	--
Value Range	-3, -4
Default Value	-3

Value	ip_time_units will be written in
-3	10^{-3} seconds (ms)
-4	10^{-4} seconds (0.1 ms)

The change of the synchronisation interval will only take effect after a reset. If the interval should be changed via CAN bus, it is necessary to save the parameter set first and reset the device after that.

8.4.2.6 Object 60C3$_h$: interpolation_sync_definition

The object **interpolation_sync_definition** determines the kind of synchronisation (**synchronize_on_group**) and the number (**ip_sync_every_n_event**) of synchronisations messages (SYNC) per synchronisation interval. For the only the standard SYNC telegram and 1 SYNC per interval can be set.

Index	60C3$_h$
Name	interpolation_sync_definition
Object Code	ARRAY
No. of Elements	2
Data Type	UINT8

Sub-Index	01$_h$
Description	syncronize_on_group
Access	rw
PDO Mapping	yes
Units	--
Value Range	0
Default Value	0

Value	Description
0	Use standard SYNC telegramm

Sub-Index	02_h
Description	ip_sync_every_n_event
Access	rw
PDO Mapping	yes
Units	--
Value Range	1
Default Value	1

8.4.2.7 Object 60C4$_h$: interpolation_data_configuration

By the object record **interpolation_data_configuration** the kind (**buffer_organisation**) and size (**max_buffer_size**, **actual_buffer_size**) of a possibly available buffer can be set. Additionally the access can be controlled by the objects **buffer_position** and **buffer_clear**. The object **size_of_data_record** returns the size of one buffer item. Even though no buffer is available for the interpolation type „linear interpolation without buffer“, the access has to be enabled using the object **buffer_clear**.

Index	60C4$_h$
Name	interpolation_data_configuration
Object Code	RECORD
No. of Elements	6

Sub-Index	01$_h$
Description	max_buffer_size
Data Type	UINT32
Access	ro
PDO Mapping	no
Units	--
Value Range	0
Default Value	0

Sub-Index	02$_h$
Description	actual_size
Data Type	UINT32
Access	rw
PDO Mapping	yes
Units	--
Value Range	0...max_buffer_size
Default Value	0

Sub-Index	03$_h$
Description	buffer_organisation
Data Type	UINT8
Access	rw
PDO Mapping	yes
Units	--
Value Range	0
Default Value	0

Value	Description
0	FIFO

Sub-Index	04_h
Description	buffer_position
Data Type	UINT16
Access	rw
PDO Mapping	yes
Units	--
Value Range	0
Default Value	0

Sub-Index	05_h
Description	size_of_data_record
Data Type	UINT8
Access	wo
PDO Mapping	yes
Units	--
Value Range	2
Default Value	2

Sub-Index	06_h
Description	buffer_clear
Data Type	UINT8
Access	wo
PDO Mapping	yes
Units	--
Value Range	0, 1
Default Value	0

Value	Description
0	Clear buffer / Access to $60C1_h$ disabled
1	Access to $60C1_h$ enabled

8.4.3 Functional Description

8.4.3.1 Preliminary parameterisation

Before the **interpolated position mode** can be entered, several settings have to be done: The interpolation interval (**interpolation_time_period**), i.e the time between two SYNC messages, the kind of interpolation (**interpolation_submode_select**) and the kind of synchronisation (**interpolation_sync_definition**) have to be set. Additionally the access to the position buffer has to be enabled by the object **buffer_clear** .

EXAMPLE

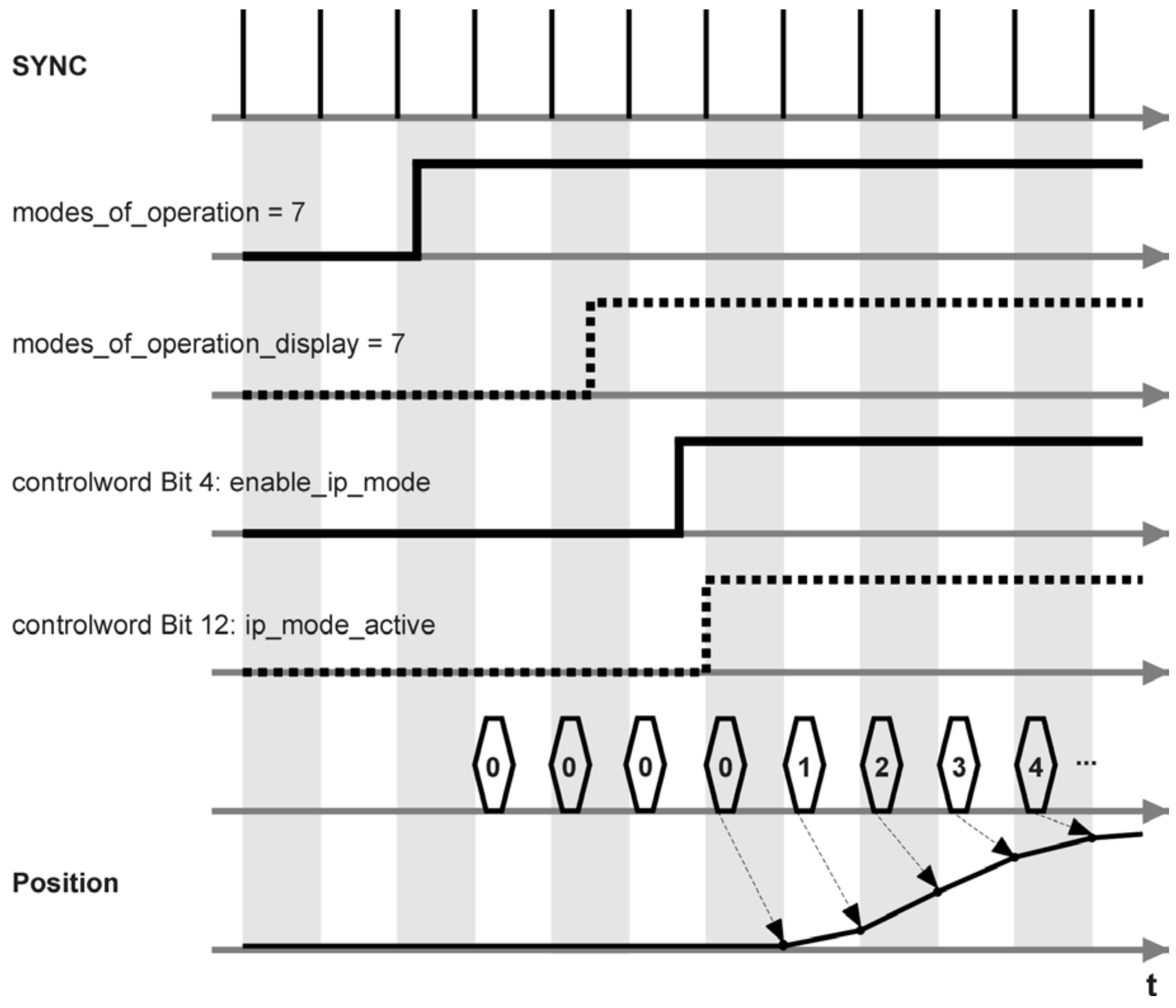
Task		CAN object / COB		
Interpolation type	-2	60C0h, interpolation_submode_select	=	-2
Time unit	0.1 ms	60C2h_02h, interpolation_time_index	=	-04
Time interval	4 ms	60C2h_01h, interpolation_time_units	=	40
Save parameter set		1010h_01h, save_all_parameters		
Execute reset		NMT reset node		
Wait for bootup		Bootup message		
Enable buffer	1	60C4h_06h, buffer_clear	=	1
Create SYNCs		SYNC (every 4 ms)		

8.4.3.2 Activation of the Interpolated Position Mode and first synchronisation

The **IP** will be activated by the object **modes_of_operation (6060$_h$)**. From this point in time the controller tries to synchronise to the external clock given by the SYNC telegrams. On success the **interpolated position mode** will be displayed in the object **modes_of_operation_display (6061$_h$)**. During the first synchronisation period "**unknown mode**" (-1) will be returned as well as if the synchronisation is lost because of an invalid interval for example.

If the **interpolated position mode** is reached the transmission of position data can be started. For logical reasons the host first reads the position actual value of the servo controller and transmits it cyclically as demand value (**interpolation_data_record**). After that the acceptance of the data can be enabled by handshake bits of the **controlword** and the **statusword**. By setting the bit **enable_ip_mode** in the **controlword** the host signals that the position data should be evaluated. The position data will not be processed until the servo controller acknowledges that with setting bit **ip_mode_selected** in the **statusword**.

This results in the following sequence:



SYNC
modes_of_operation = 7
modes_of_operation_display = 7
controlword Bit 4: enable_ip_mode
controlword Bit 12: ip_mode_active
0
0
0
0
1
2
3
4
...
Position
t

Figure 8.21: First synchronisation und data processing

No.	Event	CAN object
1	Create SYNC messages	
2	Request the operation mode „IP“	6060$_h$, modes_of_operation = 07
3	Wait for the operation mode	6061$_h$, modes_of_operation_display = 07
4	Read actual position	6064$_h$, position_actual_value
5	Rewrite it as demand value	60C1$_h$_01$_h$, ip_data_position
6	Start interpolation	6040$_h$, controlword, enable_ip_mode
7	Wait for acknowledge	6041$_h$, statusword, ip_mode_active
8	Change position setpoints according to the desired trajectory	60C1$_h$_01$_h$, ip_data_position

To prevent the further evaluation of position data the bit **enable_ip_mode** can be cleared and the operation mode can be changed after that.

8.4.3.3 Interruption of interpolation in case of an error

If a currently running interpolation (**ip_mode_active** set) will be interrupted by the occurance of an error, the servo controller reacts as specified for the certain error (i.e. disabling the controller and changing to the state **SWICTH_ON_DISABLED**).

The interpolation can only be restarted by a re-synchronisation, because the state **OPERATION_ENABLE** has to be entered again, whereby the bit **ip_mode_active** will be cleared.

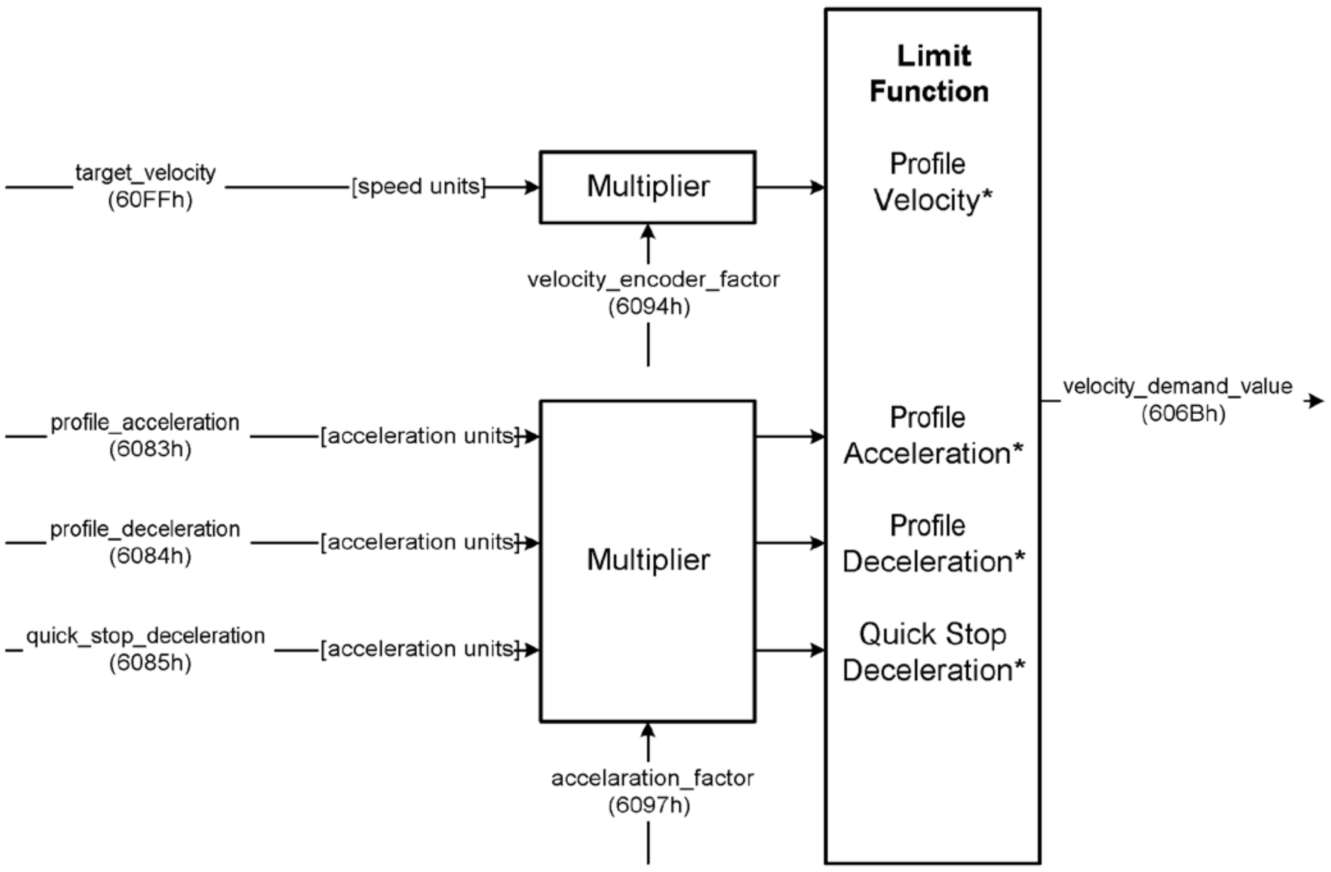
8.5 Profile Velocity Mode

8.5.1 Survey

The profile velocity mode includes the following subfunctions:

- Setpoint generation by the ramp generator
- Speed recording via the angle encoder by differentiation
- Speed control with suitable input and output signals
- Limitation of the desired torque value (**torque_demand_value**)
- Control of the actual speed (**velocity_actual_value**) with the window-function/threshold

The meaning of the following parameters is described in the chapter Profile Position Mode: **profile_acceleration, profile_deceleration, quick_stop**



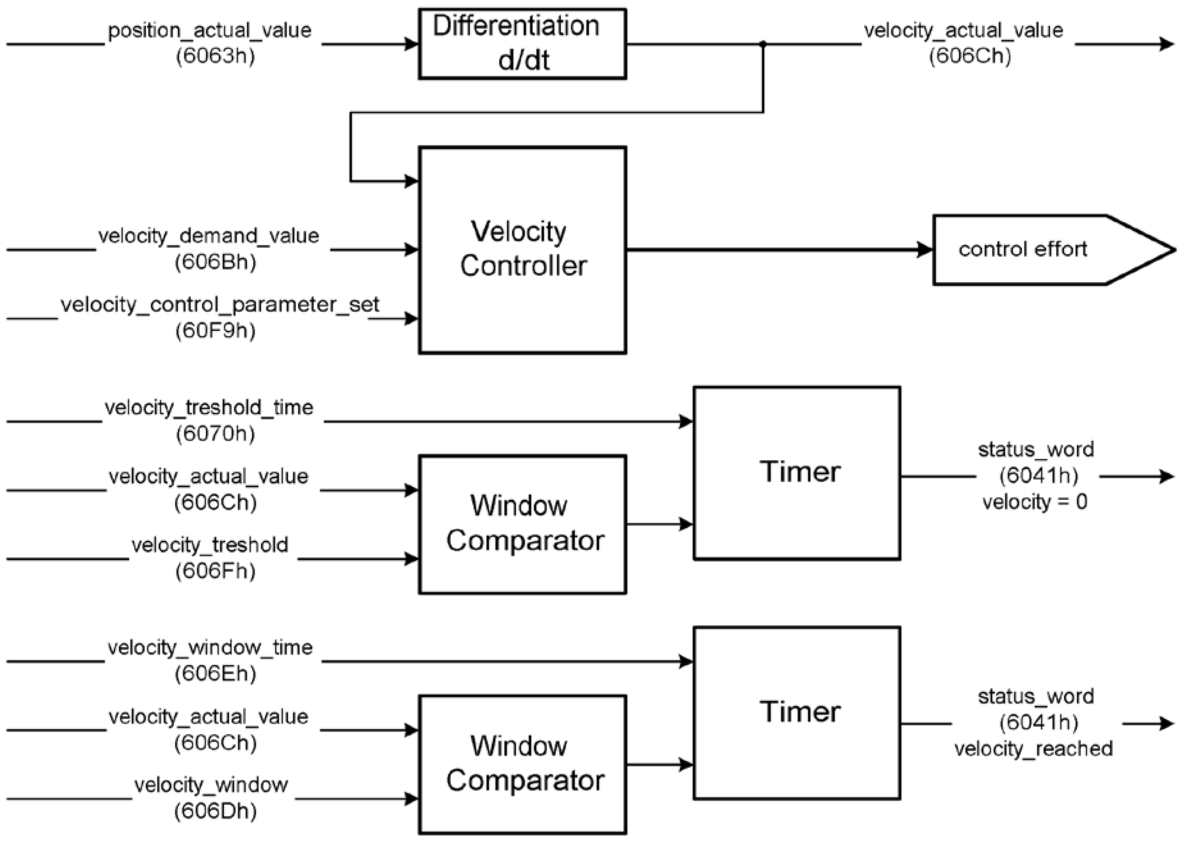


Figure 8.22: Structure of the Profile Velocity Mode

8.5.2 Description of Objects

8.5.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
6069_h	VAR	velocity_sensor_actual_value	INT32	ro
$606A_h$	VAR	sensor_selection_code	INT16	rw
$606B_h$	VAR	velocity_demand_value	INT32	ro
$202E_h$	VAR	velocity_demand_sync_value	INT32	ro
$606C_h$	VAR	velocity_actual_value	INT32	ro
$606D_h$	VAR	velocity_window	UINT16	rw
$606E_h$	VAR	velocity_window_time	UINT16	rw
$606F_h$	VAR	velocity_threshold	UINT16	rw
6080_h	VAR	max_motor_speed	UINT32	rw
$60FF_h$	VAR	target_velocity	INT32	rw

8.5.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	INT16	6.18. Device control
6041_h	VAR	statusword	UINT16	6.18. Device control
6063_h	VAR	position_actual_value*	INT32	6.7 Position Control Function
6071_h	VAR	target_torque	INT16	8.7 Profile Torque Mode
6072_h	VAR	max_torque_value	UINT16	8.7 Profile Torque Mode
$607E_h$	VAR	polarity	UINT8	6.3 Conversion factors (Factor Group)
6083_h	VAR	profile_acceleration	UINT32	8.3 Operating Mode »Profile Position Mode«
6084_h	VAR	profile_deceleration	UINT32	8.3 Operating Mode »Profile Position Mode«
6085_h	VAR	quick_stop_deceleration	UINT32	8.3 Operating Mode »Profile Position Mode«
6086_h	VAR	motion_profile_type	INT16	8.3 Operating Mode »Profile Position Mode«
6094_h	ARRAY	velocity_encoder_factor	UINT32	6.3 Conversion factors (Factor Group)

8.5.2.3 Object 6069_h: velocity_sensor_actual_value

The speed encoder is read via the object **velocity_sensor_actual_value**. The value is normalised in internal units. As no external speed encoder can be connected to servo controllers of the , the actual velocity value always has to be read via the object **$606C_h$**.

Index	**6069_h**
Name	**velocity_sensor_actual_value**
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	R / 4096 min
Value Range	--
Default Value	--

8.5.2.4 Object 606A$_h$: sensor_selection_code

The speed sensor can be selected by this object. Currently no separate speed sensor will be provided by the servo controller. Therefore only the default angle encoder can be selected.

Index	606A$_h$
Name	sensor_selection_code
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	0
Default Value	0

8.5.2.5 Object 606B$_h$: velocity_demand_value

The velocity demand value can be read via this object. It will be influenced by the ramp generator and the trajectory generator respectively. Besides this the correction speed of the position controller is added if it is activated

Index	606B$_h$
Name	velocity_demand_value
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	--

8.5.2.6 Object 202E$_h$: velocity_demand_sync_value

The speed setpoint of the synchronisation encoder can be read from this object in user-defined units. This setpoint is specified with the object **2022$_h$ synchronization_encoder_select** (chapter 6.11).

Index	202E$_h$
Name	velocity_demand_sync_value
Object Code	VAR
Data Type	INT32

As of Firmware 3.2.0.1.1

Access	ro
PDO Mapping	no
Units	velocity units
Value Range	--
Default Value	--

8.5.2.7 Object 606C$_h$: velocity_actual_value

The actual velocity value can be read via the object **velocity_actual_value**.

Index	606C$_h$
Name	velocity_actual_value
Object Code	VAR
Data Type	INT32

Access	ro
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	--

8.5.2.8 Objekt 2074$_h$: velocity_actual_value_filtered

The object **velocity_actual_value_filtered** provides a highly filtered velocity actual value that should only be used for display purposes. In contrast to **velocity_actual_value** the **velocity_actual_value_filtered** is not used for motor control but for overspeed protection. The filter time can be configured by object **2073**$_h$ (**velocity_display_filter_time**). See chapter 6.6.2.2

Index	2074$_h$
Name	velocity_actual_value_filtered
Object Code	VAR
Data Type	INT32

As of Firmware 3.5.x.1.1

Access	ro
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	--

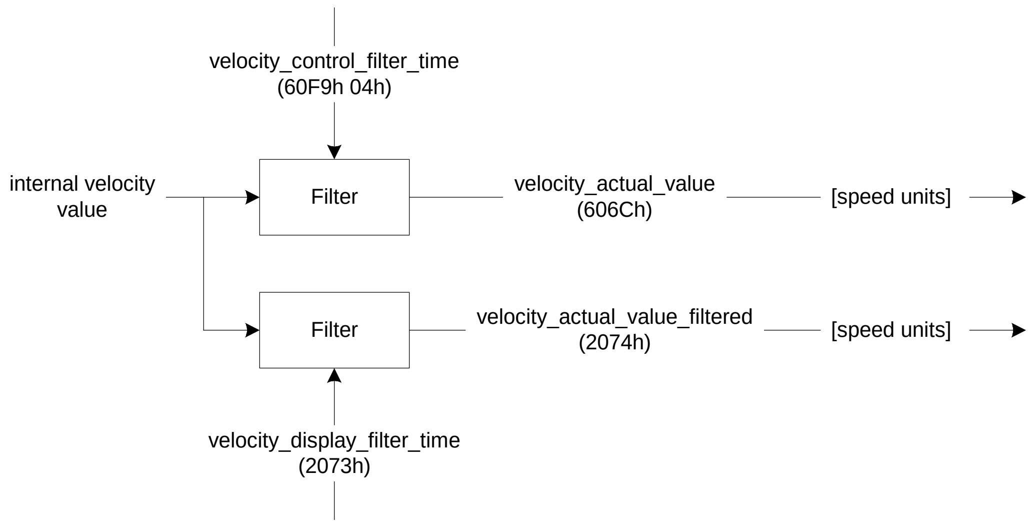


Figure 8.23: Evaluation of velocity_actual_value and velocity_actual_value_filtered

8.5.2.9 Object 606D$_h$: velocity_window

With the object **velocity_window** a tolerance window for the velocity actual value will be defined for comparing the **velocity_actual_value** with the target velocity (**target_velocity** object **60FF**$_h$). If the difference is smaller than the velocity window for a longer time than specified by the object **velocity_window_time** bit 10 (**target_reached)** will be set in the object **statusword**.

See also: Object **606E**$_h$ (**velocity_window_time**).

Index	606D$_h$
Name	velocity_window
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	speed units
Value Range	0...65536 min^{-1}
Default Value	4 min^{-1}

8.5.2.10 Object 606E$_h$: velocity_window_time

The object **velocity_window_time** serves besides the object **606D**$_h$: **velocity_window** to adjust the window comparator. It compares the **velocity_actual_value** with the **target_velocity** (object **60FF**$_h$). If the difference is smaller than specified by **velocity_window** for a longer time than specified by the object **velocity_window_time** bit 10 (**target_reached**) will be set in the object **statusword**.

Index	606E$_h$
Name	velocity_window_time
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	ms
Value Range	0...4999
Default Value	0

8.5.2.11 Object 606F$_h$: velocity_threshold

The object **velocity_threshold** determines the velocity underneath the axis is regarded as stationary. As soon as the **velocity_actual_value** exceeds the **velocity_threshold** longer than the **velocity_threshold_time** bit 12 is cleared in the **statusword**.

Index	606F$_h$
Name	velocity_threshold
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	speed units
Value Range	0...65536 min^{-1}
Default Value	10

8.5.2.12 Object 6070$_h$: velocity_threshold_time

The object **velocity_threshold** determines the velocity below the axis is regarded as stationary. As soon as the **velocity_actual_value** exceeds the **velocity_threshold** longer than the **velocity_threshold_time** bit 12 is cleared in the **statusword**.

Index	6070$_h$
Name	velocity_threshold_time
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	ms
Value Range	0...4999
Default Value	0

8.5.2.13 Object 6080$_h$: max_motor_speed

The object **max_motor_speed** specifies the maximum permissible speed for the motor in rpm. The object is used to protect the motor and can be taken from the motor specifications. The velocity set point value is limited to the value of the object **max_motor_speed**.

Index	6080$_h$
Name	max_motor_speed
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	min^{-1}
Value Range	0... 32768 min^{-1}
Default Value	32768 min^{-1}

8.5.2.14 Object 60FF$_h$: target_velocity

The object **target_velocity** is the setpoint for the ramp generator.

Index	60FF$_h$
Name	target_velocity
Object Code	VAR
Data Type	INT32

Access	rw
PDO Mapping	yes
Units	speed units
Value Range	--
Default Value	--

8.6 Speed ramps

When the object **modes_of_operation** is set to *profile_velocity_mode*, the setpoint ramp is always activated. By the objects **profile_acceleration** and **profile_deceleration** it is possible to smooth the change of velocity.

The controller allows not only the entering of accelerations and decelerations, but also to differentiate between negative and positive speed. The following figure makes this behaviour clear:

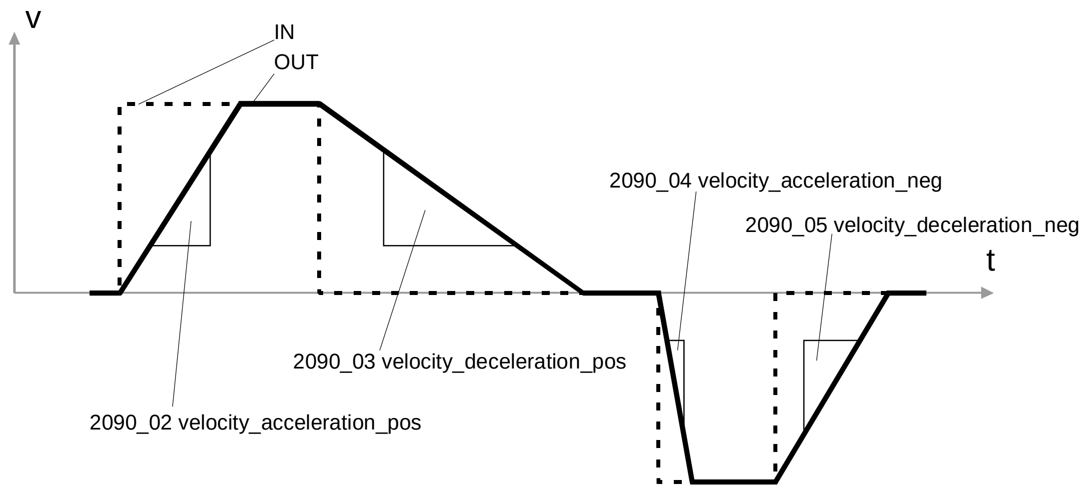

v
IN
OUT
t
2090_04 velocity_acceleration_neg
2090_05 velocity_deceleration_neg
2090_03 velocity_deceleration_pos
2090_02 velocity_acceleration_pos

Figure 8.24: Speed ramps

The record **velocity_ramps** allows the parameterisation of the 4 accelerations. Please consider, that the objects **profile_acceleration** and **profile_deceleration** change the same internal accelerations as the record **velocity_ramps**. When the object **profile_acceleration** is changed, the objects **velocity_acceleration_pos** and **velocity_acceleration_neg** are changed too. When the object **profile_deceleration** is changed, the objects **velocity_deceleration_pos** and **velocity_ deceleration_neg** are changed too. The speed ramps can be enabled resp. disabled via the object **velocity_ramps_enable**.

Index	2090$_h$
Name	**velocity_ramps**
Object Code	RECORD
No. of Elements	5

As of Firmware 3.0.x.1.1

Sub-Index	01$_h$
Description	**velocity_ramps_enable**
Data Type	UINT8
Access	rw
PDO Mapping	no
Units	--
Value Range	0: Speed ramps off 1: Speed ramps on
Default Value	1

As of Firmware 3.0.x.1.1

Sub-Index	02_h
Description	velocity_acceleration_pos
Data Type	INT32
Access	rw
PDO Mapping	no
Units	acceleration units
Value Range	--
Default Value	14 100 min^{-1}/s

Sub-Index	03_h
Description	velocity_deceleration_pos
Data Type	INT32
Access	rw
PDO Mapping	no
Units	acceleration units
Value Range	--
Default Value	14 100 min^{-1}/s

Sub-Index	04_h
Description	velocity_acceleration_neg
Data Type	INT32
Access	rw
PDO Mapping	no
Units	acceleration units
Value Range	--
Default Value	14 100 min^{-1}/s

Sub-Index	05_h
Description	velocity_deceleration_neg
Data Type	INT32
Access	rw
PDO Mapping	no
Units	acceleration units
Value Range	--
Default Value	14 100 min^{-1}/s

8.7 Profile Torque Mode

8.7.1 Survey

This chapter describes the torque controlled operation. This operating mode offers the chance to demand an external torque value (**target_torque**) which can be smoothed by the integrated ramp generator. So it is also possible to use this servo controller for trajectory control functions where both position controller and speed controller are dislocated to an external computer.

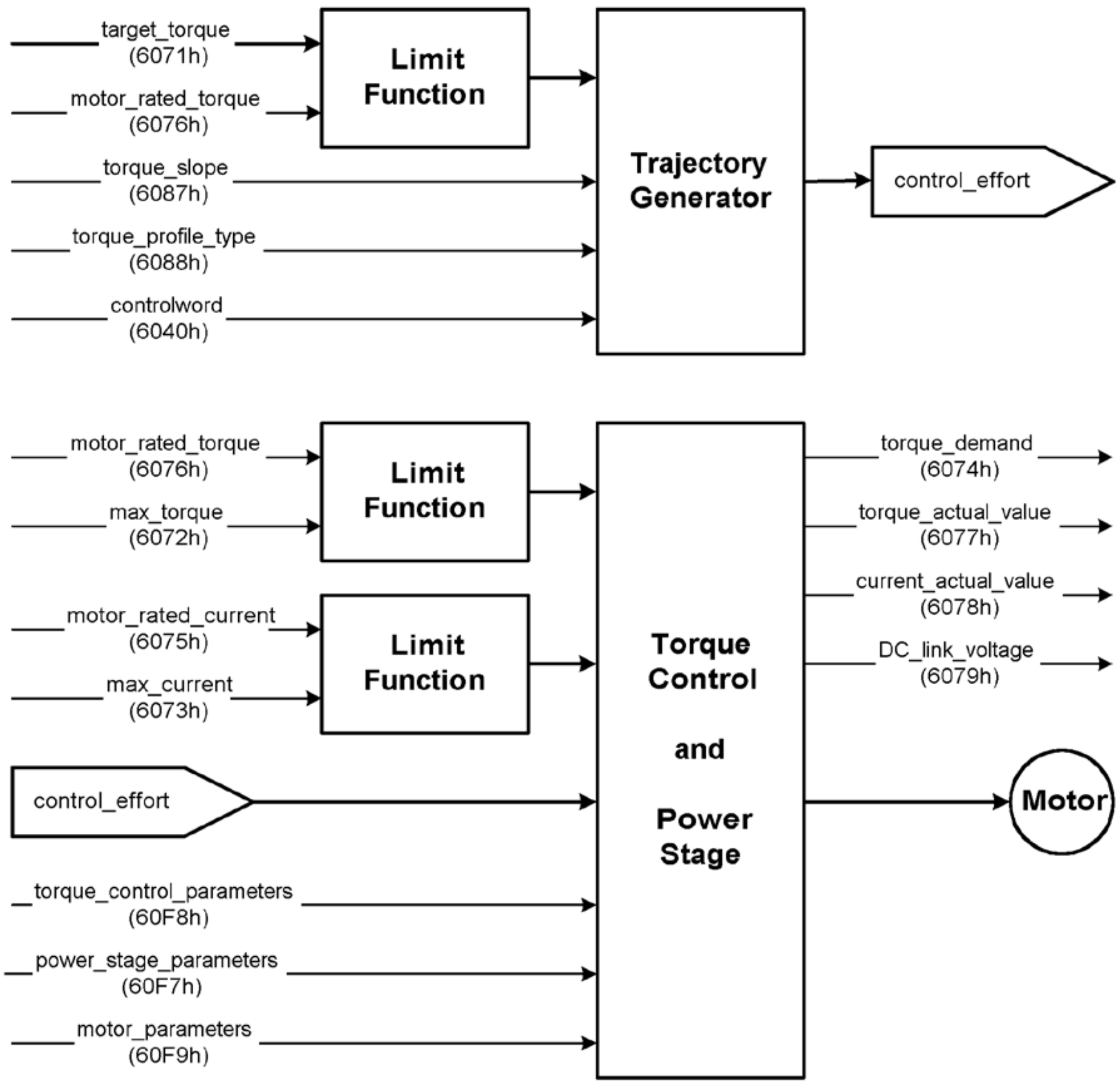


Figure 8.25: Structure of the Profile Torque Mode

The parameters **torque_slope** and **torque_profile_type** (ramp form) have to be specified for the ramp generator. If bit 8 **halt** is set in the **controlword** the ramp generator reduces the torque down to zero. Correspondingly it raises it again to the **target_torque** if bit 8 is cleared. In both cases this will be done under consideration of the ramp generator. All definitions within this document refer to rotatable motors. If linear motors are used all "torque"-objects correspond to "force" instead. For reasons of simplicity the objects do not exist twice and their names should not be modified.
The operating modes Profile Position Mode and Profile Velocity Mode need the torque controller to work properly. Therefore it is always necessary to parametrize the torque controller.

8.7.2 Description of Objects

8.7.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
6071_h	VAR	target_torque	INT16	rw
6072_h	VAR	max_torque	UINT16	rw
6074_h	VAR	torque_demand_value	INT16	ro
6076_h	VAR	motor_rated_torque	UINT32	rw
6077_h	VAR	torque_actual_value	INT16	ro
6078_h	VAR	current_actual_value	INT16	ro
6079_h	VAR	DC_link_circuit_voltage	UINT32	ro
6087_h	VAR	torque_slope	UINT32	rw
6088_h	VAR	torque_profile_type	INT16	rw
$60F7_h$	RECORD	power_stage_parameters		rw
$60F6_h$	RECORD	torque_control_parameters		rw

8.7.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	INT16	6.16 Device control
$60F9_h$	RECORD	motor_parameters		6.5 Current control and motor adaptation
6075_h	VAR	motor_rated_current	UINT32	6.5 Current control and motor adaptation
6073_h	VAR	max_current	UINT16	6.5 Current control and motor adaptation

8.7.2.3 Object 6071$_h$: target_torque

This parameter is the input value for the torque controller in Profile Torque Mode. It is specified as thousandths of the nominal torque (object **6076$_h$**).

Index	**6071$_h$**
Name	**target_torque**
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	motor_rated_torque / 1000
Value Range	-32768...32768
Default Value	0

8.7.2.4 Object 6072$_h$: max_torque

This value is the maximum permissible value of the motor. It is specified as thousandths of **motor_rated_torque** (object **6076$_h$**). If for example a double overload of the motor is permissible for a short while the value 2000 has to be entered.

The Object **6072$_h$**: **max_torque** corresponds with object **6073$_h$**: **max_current**. You are only allowed to write to one of these objects, once the object **6075$_h$**: **motor_rated_current** has been configured with a valid value.

Index	**6072$_h$**
Name	**max_torque**
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	yes
Units	motor_rated_torque / 1000
Value Range	1000...65536
Default Value	2023

8.7.2.5 Object 6074$_h$: torque_demand_value

The current demand torque can be read in thousandths of **motor_rated_torque** (6076$_h$) via this object. The internal limitations of the servo controller will be considered (current limit values and I^{2}t control).

Index	6074$_h$
Name	torque_demand_value
Object Code	VAR
Data Type	INT16

Access	ro
PDO Mapping	yes
Units	motor_rated_torque / 1000
Value Range	--
Default Value	--

8.7.2.6 Object 6076$_h$: motor_rated_torque

This object specifies the nominal torque of the motor. This value can be taken from the motor plate. It has to be entered by the unit 0.001 Nm.

Index	6076$_h$
Name	motor_rated_torque
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	0.001 Nm
Value Range	--
Default Value	296

8.7.2.7 Object 6077$_h$: torque_actual_value

The actual torque value of the motor can be read via this object in thousandths of the nominal torque (object **6076$_h$**).

Index	6077$_h$
Name	torque_actual_value
Object Code	VAR
Data Type	INT16

Access	ro
PDO Mapping	yes
Units	motor_rated_torque / 1000
Value Range	--
Default Value	--

8.7.2.8 Object 6078$_h$: current_actual_value

The actual current value of the motor can be read via this object in thousandths of the nominal current (object **6075$_h$**).

Index	6078$_h$
Name	current_actual_value
Object Code	VAR
Data Type	INT16

Access	ro
PDO Mapping	yes
Units	motor_rated_current / 1000
Value Range	--
Default Value	--

8.7.2.9 Object 6079$_h$: dc_link_circuit_voltage

The voltage in the intermediate circuit of the regulator can be read via this object. The voltage is specified in millivolt.

Index	6079$_h$
Name	dc_link_circuit_voltage
Object Code	VAR
Data Type	UINT32

Access	ro
PDO Mapping	yes
Units	mV
Value Range	--
Default Value	--

8.7.2.10 Object 6087$_h$: torque_slope

This parameter describes the modification speed of the setpoint ramp. This speed is to be specified as thousandths of the nominal torque per second. For example the desired torque value **target_torque** is raised from 0 Nm to the value **motor_rated_torque**. If the output value of the interconnected torque ramp is to reach this value within a second the value 1000 is to be entered into this object.

Index	6087$_h$
Name	torque_slope
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	motor_rated_torque / 1000 s
Value Range	--
Default Value	E310F94$_h$

8.7.2.11 Object 6088$_h$: torque_profile_type

The object **torque_profile_type** defines by which shape of curve a setpoint step is executed. Instantaneous only the linear ramp is implemented in the servo controller so only 0 can be written to this object.

Index	6088$_h$
Name	torque_profile_type
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	0
Default Value	0

Value	Description
0	Linear Ramp

9 Appendix

9.1 Characteristics of the CAN interface

The CAN interface has the following features:

- CAN specification V2.0 part A (part B passive, e. g. messages of this kind will be tolerated but not processed)
- physical layer: ISO 11898

9.2 Header File

```
/*****************************************************************************************
*                       OBJEKTE.H                                   *
*                                                           *
* Definition of CANopen -Objects                                    *
*                                                           *
* Author:       Ulf Matthiesen                                *
* Date:         05.09.2007                                    *
* Update:                                                     *
* Tests:        —                                             *
* Release:      —                                             *
* Version:      4.00                                          *
*                                                           *
*                                                           *
*                                                           *
* Format:       (0xHHHHSULL mit  HHHH = Main index                    *
*                       SU   = Sub index                        *
*                       LL   = Length in Bits + 0, if unsigned          *
*                                     + 1, if signed            *
*                                                           *
*                                                           *
*                                                           *
*                                                           *
*****************************************************************************************/

#define cUINT8   0x08
#define cUINT16  0x10
#define cUINT32  0x20
#define cINT8    0x09
#define cINT16   0x11
#define cINT32   0x21
#define cPDO     0x00 /* Appropriate bit constants can be entered here */
#define cWR      0x00 /* Appropriate bit constants can be entered here */
```

```
#define cRD     0x00 /* Appropriate bit constants can be entered here */

#define cUINT8   0x08
#define cUINT16  0x10
#define cUINT32  0x20
#define cINT8    0x09
#define cINT16   0x11
#define cINT32   0x21
#define cPDO     0x00 /* Hier geeignete Bitkonstanten einfügbar */
#define cWR      0x00 /* Hier geeignete Bitkonstanten einfügbar */
#define cRD      0x00 /* Hier geeignete Bitkonstanten einfügbar */

#define device_type                        (0x100000 + UINT32   + cRD           )
#define error_register                     (0x100100 + UINT8    + cRD       + cPDO )
#define manufacturer_status_register       (0x100200 + UINT32   + cRD           )
#define pre_defined_error_field            (0x100300 + UINT8    + cRD + cWR      )
#define standard_error_field_0             (0x100301 + UINT32   + cRD           )
#define standard_error_field_1             (0x100302 + UINT32   + cRD           )
#define standard_error_field_2             (0x100303 + UINT32   + cRD           )
#define standard_error_field_3             (0x100304 + UINT32   + cRD           )
#define cob_id_sync                        (0x100500 + UINT32   + cRD + cWR      )
#define communication_cycle_period         (0x100600 + UINT32   + cRD + cWR      )
#define synchronous_window_length          (0x100700 + UINT32   + cRD + cWR      )
#define guard_time                         (0x100C00 + UINT16   + cRD + cWR      )
#define life_time_factor                   (0x100D00 + UINT8    + cRD + cWR      )
#define store_parameters                   (0x101000 + UINT8    + cRD           )
#define save_all_parameters                (0x101001 + UINT32   + cRD + cWR      )
#define restore_parameters                 (0x101100 + UINT8    + cRD           )
#define restore_all_default_parameters     (0x101101 + UINT32   + cRD + cWR      )
#define cob_id_time_stamp_message          (0x101200 + UINT32   + cRD + cWR      )
#define cob_id_emergency_message           (0x101400 + UINT32   + cRD + cWR      )
#define consumer_heartbeat_time            (0x101600 + UINT8    + cRD           )
#define consumer_heartbeat_time_1          (0x101601 + UINT32   + cRD + cWR      )
#define producer_heartbeat_time            (0x101700 + UINT16   + cRD + cWR      )
#define identity_object                    (0x101800 + UINT8    + cRD           )
#define vendor_id                          (0x101801 + UINT32   + cRD           )
#define product_code                       (0x101802 + UINT32   + cRD           )
#define revision_number                    (0x101803 + UINT32   + cRD           )
#define serial_number                      (0x101804 + UINT32   + cRD           )
#define server_sdo_parameter               (0x120000 + UINT8    + cRD           )
#define cob_id_client_server               (0x120001 + UINT32   + cRD           )
#define cob_id_server_client               (0x120002 + UINT32   + cRD           )
#define receive_pdo_parameter_rpdo1        (0x140000 + UINT8    + cRD           )
#define cob_id_used_by_pdo_rpdo1           (0x140001 + UINT32   + cRD + cWR      )
#define transmission_type_rpdo1            (0x140002 + UINT8    + cRD + cWR      )
#define receive_pdo_parameter_rpdo2        (0x140100 + UINT8    + cRD           )
#define cob_id_used_by_pdo_rpdo2           (0x140101 + UINT32   + cRD + cWR      )
#define transmission_type_rpdo2            (0x140102 + UINT8    + cRD + cWR      )
#define receive_pdo_parameter_rpdo3        (0x140200 + UINT8    + cRD           )
#define cob_id_used_by_pdo_rpdo3           (0x140201 + UINT32   + cRD + cWR      )
#define transmission_type_rpdo3            (0x140202 + UINT8    + cRD + cWR      )
#define receive_pdo_parameter_rpdo4        (0x140300 + UINT8    + cRD           )
#define cob_id_used_by_pdo_rpdo4           (0x140301 + UINT32   + cRD + cWR      )
#define transmission_type_rpdo4            (0x140302 + UINT8    + cRD + cWR      )
#define receive_pdo_mapping_rpdo1          (0x160000 + UINT8    + cRD + cWR      )
#define first_mapped_object_rpdo1          (0x160001 + UINT32   + cRD + cWR      )
#define second_mapped_object_rpdo1         (0x160002 + UINT32   + cRD + cWR      )
```

```
#define third_mapped_object_rpdo1           (0x160003 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_rpdo1          (0x160004 + UINT32  + cRD + cWR     }
#define receive_pdo_mapping_rpdo2           (0x160100 + UINT8   + cRD + cWR     }
#define first_mapped_object_rpdo2           (0x160101 + UINT32  + cRD + cWR     }
#define second_mapped_object_rpdo2          (0x160102 + UINT32  + cRD + cWR     }
#define third_mapped_object_rpdo2           (0x160103 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_rpdo2          (0x160104 + UINT32  + cRD + cWR     }
#define receive_pdo_mapping_rpdo3           (0x160200 + UINT8   + cRD + cWR     }
#define first_mapped_object_rpdo3           (0x160201 + UINT32  + cRD + cWR     }
#define second_mapped_object_rpdo3          (0x160202 + UINT32  + cRD + cWR     }
#define third_mapped_object_rpdo3           (0x160203 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_rpdo3          (0x160204 + UINT32  + cRD + cWR     }
#define receive_pdo_mapping_rpdo4           (0x160300 + UINT8   + cRD + cWR     }
#define first_mapped_object_rpdo4           (0x160301 + UINT32  + cRD + cWR     }
#define second_mapped_object_rpdo4          (0x160302 + UINT32  + cRD + cWR     }
#define third_mapped_object_rpdo4           (0x160303 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_rpdo4          (0x160304 + UINT32  + cRD + cWR     }
#define transmit_pdo_parameter_tpdo1        (0x180000 + UINT8   + cRD           }
#define cob_id_used_by_pdo_tpdo1            (0x180001 + UINT32  + cRD + cWR     }
#define transmission_type_tpdo1             (0x180002 + UINT8   + cRD + cWR     }
#define inhibit_time_tpdo1                  (0x180003 + UINT16  + cRD + cWR     }
#define transmit_pdo_parameter_tpdo2        (0x180100 + UINT8   + cRD           }
#define cob_id_used_by_pdo_tpdo2            (0x180101 + UINT32  + cRD + cWR     }
#define transmission_type_tpdo2             (0x180102 + UINT8   + cRD + cWR     }
#define inhibit_time_tpdo2                  (0x180103 + UINT16  + cRD + cWR     }
#define transmit_pdo_parameter_tpdo3        (0x180200 + UINT8   + cRD           }
#define cob_id_used_by_pdo_tpdo3            (0x180201 + UINT32  + cRD + cWR     }
#define transmission_type_tpdo3             (0x180202 + UINT8   + cRD + cWR     }
#define inhibit_time_tpdo3                  (0x180203 + UINT16  + cRD + cWR     }
#define transmit_pdo_parameter_tpdo4        (0x180300 + UINT8   + cRD           }
#define cob_id_used_by_pdo_tpdo4            (0x180301 + UINT32  + cRD + cWR     }
#define transmission_type_tpdo4             (0x180302 + UINT8   + cRD + cWR     }
#define inhibit_time_tpdo4                  (0x180303 + UINT16  + cRD + cWR     }
#define transmit_pdo_mapping_tpdo1          (0x1A0000 + UINT8   + cRD + cWR     }
#define first_mapped_object_tpdo1           (0x1A0001 + UINT32  + cRD + cWR     }
#define second_mapped_object_tpdo1          (0x1A0002 + UINT32  + cRD + cWR     }
#define third_mapped_object_tpdo1           (0x1A0003 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_tpdo1          (0x1A0004 + UINT32  + cRD + cWR     }
#define transmit_pdo_mapping_tpdo2          (0x1A0100 + UINT8   + cRD + cWR     }
#define first_mapped_object_tpdo2           (0x1A0101 + UINT32  + cRD + cWR     }
#define second_mapped_object_tpdo2          (0x1A0102 + UINT32  + cRD + cWR     }
#define third_mapped_object_tpdo2           (0x1A0103 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_tpdo2          (0x1A0104 + UINT32  + cRD + cWR     }
#define transmit_pdo_mapping_tpdo3          (0x1A0200 + UINT8   + cRD + cWR     }
#define first_mapped_object_tpdo3           (0x1A0201 + UINT32  + cRD + cWR     }
#define second_mapped_object_tpdo3          (0x1A0202 + UINT32  + cRD + cWR     }
#define third_mapped_object_tpdo3           (0x1A0203 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_tpdo3          (0x1A0204 + UINT32  + cRD + cWR     }
#define transmit_pdo_mapping_tpdo4          (0x1A0300 + UINT8   + cRD + cWR     }
#define first_mapped_object_tpdo4           (0x1A0301 + UINT32  + cRD + cWR     }
#define second_mapped_object_tpdo4          (0x1A0302 + UINT32  + cRD + cWR     }
#define third_mapped_object_tpdo4           (0x1A0303 + UINT32  + cRD + cWR     }
#define fourth_mapped_object_tpdo4          (0x1A0304 + UINT32  + cRD + cWR     }
#define manufacturer_statuswords            (0x200000 + UINT8   + cRD           }
#define manufacturer_statusword_1           (0x200001 + UINT32  + cRD       + cPDO }
#define manufacturer_status_masks           (0x200500 + UINT8   + cRD           }
#define manufacturer_status_mask_1          (0x200501 + UINT32  + cRD + cWR + cPDO }
#define manufacturer_status_invert            (0x200A00 + UINT8   + cRD            }
#define manufacturer_status_invert_1           (0x200A01 + UINT32  + cRD + cWR + cPDO }
#define last_warning_code                    (0x200F00 + UINT16  + cRD      + cPDO }
#define tpdo1_transmit_mask                  (0x201400 + UINT8   + cRD           }
#define tpdo1_transmit_mask_low              (0x201401 + UINT32  + cRD + cWR      }
#define tpdo1_transmit_mask_high             (0x201402 + UINT32  + cRD + cWR      }
#define tpdo2_transmit_mask                  (0x201500 + UINT8   + cRD           }
#define tpdo2_transmit_mask_low              (0x201501 + UINT32  + cRD + cWR      }
#define tpdo2_transmit_mask_high             (0x201502 + UINT32  + cRD + cWR      }
#define tpdo3_transmit_mask                  (0x201600 + UINT8   + cRD           }
#define tpdo3_transmit_mask_low              (0x201601 + UINT32  + cRD + cWR      }
#define tpdo3_transmit_mask_high             (0x201602 + UINT32  + cRD + cWR      }
#define tpdo4_transmit_mask                  (0x201700 + UINT8   + cRD           }
#define tpdo4_transmit_mask_low              (0x201701 + UINT32  + cRD + cWR      }
#define tpdo4_transmit_mask_high             (0x201702 + UINT32  + cRD + cWR      }
#define encoder_emulation_data               (0x201A00 + UINT8   + cRD           }
#define encoder_emulation_resolution          (0x201A01 + INT32   + cRD + cWR      }
#define encoder_emulation_offset             (0x201A02 + INT16   + cRD + cWR      }
#define commutation_encoder_select           (0x201F00 + INT16   + cRD + cWR      }
#define position_controller_resolution        (0x202000 + UINT32  + cRD + cWR      }
#define position_encoder_selection           (0x202100 + INT16   + cRD + cWR      }
#define synchronisation_encoder_selection      (0x202200 + INT16   + cRD + cWR      }
#define synchronisation_filter_time          (0x202300 + UINT32  + cRD + cWR      }
#define encoder_x2a_data_field               (0x202400 + UINT8   + cRD           }
#define encoder_x2a_resolution               (0x202401 + UINT32  + cRD           }
#define encoder_x2a_numerator                (0x202402 + INT16   + cRD + cWR      }
#define encoder_x2a_divisor                  (0x202403 + INT16   + cRD + cWR      }
#define encoder_x10_data_field               (0x202500 + UINT8   + cRD           }
#define encoder_x10_resolution               (0x202501 + UINT32  + cRD + cWR      }
#define encoder_x10_numerator                (0x202502 + INT16   + cRD + cWR      }
#define encoder_x10_divisor                  (0x202503 + INT16   + cRD + cWR      }
#define encoder_x10_counter                  (0x202504 + UINT32  + cRD      + cPDO }
#define encoder_x2b_data_field               (0x202600 + UINT8   + cRD           }
#define encoder_x2b_resolution               (0x202601 + UINT32  + cRD + cWR      }
#define encoder_x2b_numerator                (0x202602 + INT16   + cRD + cWR      }
#define encoder_x2b_divisor                  (0x202603 + INT16   + cRD + cWR      }
#define encoder_x2b_counter                  (0x202604 + UINT32  + cRD      + cPDO }
#define encoder_emulation_resolution          (0x202800 + INT32   + cRD + cWR      }
#define position_demand_sync_value           (0x202D00 + INT32   + cRD           }
#define velocity_demand_sync_value           (0x202E00 + INT32   + cRD           }
#define synchronisation_selector_data         (0x202F00 + UINT8   + cRD           }
#define synchronisation_main                 (0x202F07 + UINT16  + cRD + cWR      }
#define set_position_absolute                (0x203000 + INT32        + cWR      }
#define torque_feed_forward                  (0x203A00 + UINT32  + cRD + cWR      }
#define homing_timeout                       (0x204500 + UINT16  + cRD + cWR      }
#define sample_data                          (0x204A00 + UINT8   + cRD           }
#define sample_mode                          (0x204A01 + UINT16  + cRD + cWR      }
#define sample_status                        (0x204A02 + UINT8   + cRD      + cPDO }
#define sample_status_mask                   (0x204A03 + UINT8   + cRD + cWR + cPDO }
#define sample_control                       (0x204A04 + UINT8        + cWR + cPDO }
#define sample_position_rising_edge          (0x204A05 + INT32   + cRD      + cPDO }
#define sample_position_falling_edge         (0x204A06 + INT32   + cRD      + cPDO }
#define velocity_display_filter_time         (0x207300 + UINT32  + cRD + cWR      }
#define velocity_actual_value_filtered       (0x207400 + INT32   + cRD      + cPDO }
#define velocity_message                     (0x207800 + UINT8   + cRD           }
#define message_target_velocity              (0x207801 + INT32   + cRD + cWR      }
#define message_velocity_window              (0x207802 + INT16   + cRD + cWR      }
#define velocity_ramps                    (0x209000 + UINT8   + cRD          }
#define velocity_rampe_enable             (0x209001 + UINT8   + cRD + cWR    }
#define velocity_acceleration_pos         (0x209002 + INT32   + cRD + cWR    }
#define velocity_deceleration_pos         (0x209003 + INT32   + cRD + cWR    }
#define velocity_acceleration_neg         (0x209004 + INT32   + cRD + cWR    }
#define velocity_deceleration_neg         (0x209005 + INT32   + cRD + cWR    }
#define error_management                  (0x210000 + UINT8   + cRD          }
#define error_number                      (0x210001 + UINT8   + cRD + cWR    }
#define error_reaction_code               (0x210002 + UINT8   + cRD + cWR    }
#define read_write_ko_nr                  (0x220000 + UINT32  + cRD + cWR    }
#define read_ko                           (0x220400 + UINT32  + cRD          }
#define write_ko                          (0x221400 + UINT32        + cWR    }
#define read_ko_record                    (0x221500 + UINT8   + cRD          }
#define read_ko_demand_value              (0x221501 + UINT32  + cRD          }
#define read_ko_actual_value              (0x221502 + UINT32  + cRD          }
#define read_ko_minimum                   (0x221503 + UINT32  + cRD          }
#define read_ko_maximum                   (0x221504 + UINT32  + cRD          }
#define analog_input_voltage              (0x240000 + UINT8   + cRD          }
#define analog_input_voltage_ch_0         (0x240001 + INT16   + cRD          }
#define analog_input_voltage_ch_1         (0x240002 + INT16   + cRD          }
#define analog_input_voltage_ch_2         (0x240003 + INT16   + cRD          }
#define analog_input_offset               (0x240100 + UINT8   + cRD          }
#define analog_input_offset_ch_0          (0x240101 + INT32   + cRD + cWR    }
#define analog_input_offset_ch_1          (0x240102 + INT32   + cRD + cWR    }
#define analog_input_offset_ch_2          (0x240103 + INT32   + cRD + cWR    }
#define current_limitation                (0x241500 + UINT8   + cRD          }
#define limit_current_input_channel       (0x241501 + INT8    + cRD + cWR    }
#define limit_current                     (0x241502 + INT32   + cRD + cWR    }
#define speed_limitation                  (0x241600 + UINT8   + cRD          }
#define limit_speed_input_channel         (0x241601 + INT8    + cRD + cWR    }
#define limit_speed                       (0x241602 + INT32   + cRD + cWR    }
#define digital_outputs_state_mapping     (0x242000 + UINT8   + cRD          }
#define dig_out_state_mapp_dout_1         (0x242001 + UINT8   + cRD + cWR    }
#define dig_out_state_mapp_dout_2         (0x242002 + UINT8   + cRD + cWR    }
#define dig_out_state_mapp_dout_3         (0x242003 + UINT8   + cRD + cWR    }
#define dig_out_state_mapp_ea88_0_low     (0x242011 + UINT32  + cRD + cWR    }
#define dig_out_state_mapp_ea88_0_high    (0x242012 + UINT32  + cRD + cWR    }
#define digital_inputs_low_byte           (0x2C0A00 + UINT8   + cRD       + cPDO }
#define error_code                        (0x603F00 + UINT16  + cRD       + cPDO }
#define controlword                       (0x604000 + UINT16  + cRD + cWR + cPDO }
#define statusword                        (0x604100 + UINT16  + cRD       + cPDO }
#define pole_number                       (0x604D00 + UINT8   + cRD + cWR + cPDO }
#define quick_stop_option_code            (0x605A00 + INT16   + cRD + cWR    }
#define shutdown_option_code              (0x605B00 + INT16   + cRD + cWR    }
#define disable_operation_option_code     (0x605C00 + INT16   + cRD + cWR    }
#define stop_option_code                  (0x605D00 + INT16   + cRD + cWR    }
#define fault_reaction_option_code        (0x605E00 + INT16   + cRD + cWR    }
#define modes_of_operation                (0x606000 + INT8    + cRD + cWR + cPDO }
#define modes_of_operation_display        (0x606100 + INT8    + cRD       + cPDO }
#define position_demand_value             (0x606200 + INT32   + cRD       + cPDO }
#define position_actual_value*            (0x606300 + INT32   + cRD       + cPDO }
#define position_actual_value             (0x606400 + INT32   + cRD       + cPDO }
#define following_error_window            (0x606500 + UINT32  + cRD + cWR + cPDO }
#define following_error_time_out          (0x606600 + UINT16  + cRD + cWR + cPDO }
#define position_window                   (0x606700 + UINT32  + cRD + cWR + cPDO }
#define position_window_time              (0x606800 + UINT16  + cRD + cWR + cPDO }
#define velocity_sensor_actual_value      (0x606900 + INT32   + cRD       + cPDO }
#define sensor_selection_code            (0x606A00 + INT16   + cRD + cWR + cPDO }
#define velocity_demand_value            (0x606B00 + INT32   + cRD       + cPDO }
#define velocity_actual_value            (0x606C00 + INT32   + cRD       + cPDO }
#define velocity_window                  (0x606D00 + UINT16  + cRD + cWR + cPDO }
#define velocity_window_time             (0x606E00 + UINT16  + cRD + cWR + cPDO }
#define velocity_threshold               (0x606F00 + UINT16  + cRD + cWR + cPDO }
#define velocity_threshold_time          (0x607000 + UINT16  + cRD + cWR + cPDO }
#define target_torque                    (0x607100 + INT16   + cRD + cWR + cPDO }
#define max_torque                       (0x607200 + UINT16  + cRD + cWR + cPDO }
#define max_current                      (0x607300 + UINT16  + cRD + cWR + cPDO }
#define torque_demand_value              (0x607400 + INT16   + cRD       + cPDO }
#define motor_rated_current              (0x607500 + UINT32  + cRD + cWR + cPDO }
#define motor_rated_torque               (0x607600 + UINT32  + cRD + cWR + cPDO }
#define torque_actual_value              (0x607700 + INT16   + cRD       + cPDO }
#define current_actual_value             (0x607800 + INT16   + cRD       + cPDO }
#define dc_link_circuit_voltage          (0x607900 + UINT32  + cRD       + cPDO }
#define target_position                  (0x607A00 + INT32   + cRD + cWR + cPDO }
#define position_range_limit             (0x607B00 + UINT8   + cRD              }
#define min_position_range_limit         (0x607B01 + INT32   + cRD + cWR + cPDO }
#define max_position_range_limit         (0x607B02 + INT32   + cRD + cWR + cPDO }
#define home_offset                      (0x607C00 + INT32   + cRD + cWR + cPDO }
#define software_position_limit          (0x607D00 + UINT8   + cRD              }
#define min_position_limit               (0x607D01 + INT32   + cRD + cWR + cPDO }
#define max_position_limit               (0x607D02 + INT32   + cRD + cWR + cPDO }
#define polarity                         (0x607E00 + UINT8   + cRD + cWR + cPDO }
#define max_motor_speed                  (0x608000 + UINT16  + cRD + cWR + cPDO }
#define profile_velocity                 (0x608100 + UINT32  + cRD + cWR + cPDO }
#define end_velocity                     (0x608200 + UINT32  + cRD + cWR + cPDO }
#define profile_acceleration             (0x608300 + UINT32  + cRD + cWR + cPDO }
#define profile_deceleration             (0x608400 + UINT32  + cRD + cWR + cPDO }
#define quick_stop_deceleration          (0x608500 + UINT32  + cRD + cWR + cPDO }
#define motion_profile_type              (0x608600 + INT16   + cRD + cWR + cPDO }
#define torque_slope                     (0x608700 + UINT32  + cRD + cWR + cPDO }
#define torque_profile_type              (0x608800 + INT16   + cRD + cWR + cPDO }
#define position_notation_index          (0x608900 + INT8    + cRD + cWR + cPDO }
#define position_dimension_index         (0x608A00 + UINT8   + cRD + cWR + cPDO }
#define velocity_notation_index          (0x608B00 + INT8    + cRD + cWR + cPDO }
#define velocity_dimension_index         (0x608C00 + UINT8   + cRD + cWR + cPDO }
#define acceleration_notation_index      (0x608D00 + INT8    + cRD + cWR + cPDO }
#define acceleration_dimension_index     (0x608E00 + UINT8   + cRD + cWR + cPDO }
#define position_encoder_resolution      (0x608F00 + UINT8   + cRD              }
#define encoder_increments               (0x608F01 + UINT32  + cRD + cWR + cPDO }
#define motor_revolutions                (0x608F02 + UINT32  + cRD + cWR + cPDO }
#define velocity_encoder_resolution      (0x609000 + UINT8   + cRD              }
#define encoder_increments_per_second    (0x609001 + UINT32  + cRD + cWR + cPDO }
#define motor_revolutions_per_second     (0x609002 + UINT32  + cRD + cWR + cPDO }
#define gear_ratio                       (0x609100 + UINT8   + cRD              }
#define motor_revolutions                (0x609101 + UINT32  + cRD + cWR + cPDO }
#define shaft_revolutions                (0x609102 + UINT32  + cRD + cWR + cPDO }
#define feed_constant                    (0x609200 + UINT8   + cRD              }
#define feed                             (0x609201 + UINT32  + cRD + cWR + cPDO }
#define shaft_revolutions                (0x609202 + UINT32  + cRD + cWR + cPDO }
#define position_factor                  (0x609300 + UINT8   + cRD              }
#define numerator                        (0x609301 + UINT32  + cRD + cWR + cPDO }
#define divisor                          (0x609302 + UINT32  + cRD + cWR + cPDO }
#define velocity_encoder_factor          (0x609400 + UINT8   + cRD              }
#define numerator                        (0x609401 + UINT32  + cRD + cWR + cPDO }
#define divisor                          (0x609402 + UINT32   + cRD + cWR + cPDO }
#define velocity_factor_1                      (0x609500 + UINT8    + cRD            }
#define numerator                           (0x609501 + UINT32   + cRD + cWR + cPDO }
#define divisor                          (0x609502 + UINT32   + cRD + cWR + cPDO }
#define velocity_factor_2                      (0x609600 + UINT8    + cRD            }
#define numerator                           (0x609601 + UINT32   + cRD + cWR + cPDO }
#define divisor                          (0x609602 + UINT32   + cRD + cWR + cPDO }
#define acceleration_factor                     (0x609700 + UINT8    + cRD            }
#define numerator                           (0x609701 + UINT32   + cRD + cWR + cPDO }
#define divisor                          (0x609702 + UINT32   + cRD + cWR + cPDO }
#define homing_method                          (0x609800 + INT8     + cRD + cWR + cPDO }
#define homing_speeds                          (0x609900 + UINT8    + cRD            }
#define speed_during_search_for_switch         (0x609901 + UINT32   + cRD + cWR + cPDO }
#define speed_during_search_for_zero          (0x609902 + UINT32   + cRD + cWR + cPDO }
#define homing_acceleration                    (0x609A00 + UINT32   + cRD + cWR + cPDO }
#define interpolation_submode_select          (0x60C000 + INT16    + cRD + cWR + cPDO }
#define interpolation_data_record             (0x60C100 + UINT8    + cRD            }
#define ip_data_position                      (0x60C101 + INT32    + cRD + cWR + cPDO }
#define ip_data_controlword                    (0x60C102 + UINT8    + cRD + cWR + cPDO }
#define interpolation_time_period             (0x60C200 + UINT8    + cRD            }
#define ip_time_units                        (0x60C201 + UINT8    + cRD + cWR + cPDO }
#define ip_time_index                        (0x60C202 + INT8     + cRD + cWR + cPDO }
#define interpolation_sync_definition          (0x60C300 + UINT8    + cRD            }
#define syncronize_on_group                    (0x60C301 + UINT8    + cRD + cWR + cPDO }
#define ip_sync_every_n_event                  (0x60C302 + UINT8    + cRD + cWR + cPDO }
#define interpolation_data_configuration       (0x60C400 + UINT8    + cRD       + cPDO }
#define max_buffer_size                       (0x60C401 + UINT32   + cRD            }
#define actual_size                         (0x60C402 + UINT32   + cRD + cWR + cPDO }
#define buffer_organisation                    (0x60C403 + UINT8    + cRD + cWR + cPDO }
#define buffer_position                      (0x60C404 + UINT16   + cRD + cWR + cPDO }
#define size_of_data_record                    (0x60C405 + UINT8          + cWR + cPDO }
#define buffer_clear                        (0x60C406 + UINT8          + cWR + cPDO }
#define torque_control_parameters             (0x60F600 + UINT8    + cRD            }
#define torque_control_gain                    (0x60F601 + UINT16   + cRD + cWR       }
#define torque_control_time                    (0x60F602 + UINT16   + cRD + cWR       }
#define velocity_control_parameter_set         (0x60F900 + UINT8    + cRD            }
#define velocity_control_gain                  (0x60F901 + UINT16   + cRD + cWR       }
#define velocity_control_time                  (0x60F902 + UINT16   + cRD + cWR       }
#define velocity_control_filter_time           (0x60F904 + UINT16   + cRD + cWR       }
#define control_effort                        (0x60FA00 + INT32    + cRD       + cPDO }
#define position_control_parameter_set         (0x60FB00 + UINT8    + cRD            }
#define position_control_gain                  (0x60FB01 + UINT16   + cRD + cWR       }
#define position_control_time                  (0x60FB02 + UINT16   + cRD + cWR       }
#define position_control_v_max                 (0x60FB04 + UINT32   + cRD + cWR       }
#define position_error_tolerance_window        (0x60FB05 + UINT32   + cRD + cWR       }
#define digital_inputs                        (0x60FD00 + UINT32   + cRD       + cPDO }
#define digital_outputs                       (0x60FE00 + UINT8    + cRD            }
#define digital_outputs_data                   (0x60FE01 + UINT32   + cRD + cWR + cPDO }
#define digital_outputs_mask                   (0x60FE02 + UINT32   + cRD + cWR + cPDO }
#define target_velocity                       (0x60FF00 + INT32    + cRD + cWR + cPDO }
#define motor_type                           (0x640200 + UINT16   + cRD       + cPDO }
#define motor_data                           (0x641000 + UINT8    + cRD            }
#define iit_time_motor                        (0x641003 + UINT16   + cRD + cWR       }
#define iit_ratio_motor                       (0x641004 + UINT16   + cRD            }
#define phase_order                          (0x641010 + UINT16   + cRD + cWR       }
#define encoder_offset_angle                   (0x641011 + INT16    + cRD + cWR + cPDO }
#define motor_temperature_sensor_polarity      (0x641014 + INT16    + cRD + cWR + cPDO }
#define supported_drive_modes            (0x650200 + UINT32  + cRD       + cPDO }
#define drive_data                       (0x651000 + UINT8   + cRD              }
#define serial_number                    (0x651001 + UINT32  + cRD              }
#define drive_code                       (0x651002 + UINT32  + cRD              }
#define user_variable_not_saved          (0x651003 + INT16   + cRD + cWR        }
#define user_variable_saved              (0x651004 + INT16   + cRD + cWR        }
#define enable_logic                     (0x651010 + UINT16  + cRD + cWR        }
#define limit_switch_polarity            (0x651011 + INT16   + cRD + cWR        }
#define limit_switch_selector            (0x651012 + INT16   + cRD + cWR        }
#define homing_switch_selector           (0x651013 + INT16   + cRD + cWR        }
#define homing_switch_polarity           (0x651014 + INT16   + cRD + cWR        }
#define limit_switch_deceleration        (0x651015 + INT32   + cRD + cWR        }
#define brake_delay_time                 (0x651018 + UINT16  + cRD + cWR        }
#define automatic_brake_delay            (0x651019 + UINT16  + cRD + cWR        }
#define position_range_limit_enable      (0x651020 + UINT16  + cRD + cWR        }
#define position_error_switch_off_limit  (0x651022 + UINT32  + cRD + cWR        }
#define motor_temperature                (0x65102E + INT16   + cRD       + cPDO }
#define max_motor_temperature            (0x65102F + INT16   + cRD + cWR        }
#define pwm_frequency                    (0x651030 + UINT16  + cRD + cWR        }
#define power_stage_temperature          (0x651031 + INT16   + cRD       + cPDO }
#define max_power_stage_temperature      (0x651032 + INT16   + cRD              }
#define nominal_dc_link_circuit_voltage  (0x651033 + UINT32  + cRD              }
#define actual_dc_link_circuit_voltage   (0x651034 + UINT32  + cRD       + cPDO }
#define max_dc_link_circuit_voltage      (0x651035 + UINT32  + cRD              }
#define min_dc_link_circuit_voltage      (0x651036 + UINT32  + cRD + cWR        }
#define enable_dc_link_undervoltage_error (0x651037 + UINT16 + cRD + cWR        }
#define iit_error_enable                 (0x651038 + UINT16  + cRD + cWR        }
#define enable_enhanced_modulation       (0x65103A + UINT16  + cRD + cWR        }
#define iit_ratio_servo                  (0x65103D + UINT16  + cRD       + cPDO }
#define nominal_current                  (0x651040 + UINT32  + cRD              }
#define peak_current                     (0x651041 + UINT32  + cRD              }
#define drive_serial_number              (0x6510A0 + UINT32  + cRD              }
#define drive_type                       (0x6510A1 + UINT32  + cRD              }
#define drive_revision                   (0x6510A2 + UINT32  + cRD              }
#define encoder_serial_number            (0x6510A3 + UINT32  + cRD              }
#define encoder_type                     (0x6510A4 + UINT32  + cRD              }
#define encoder_revision                 (0x6510A5 + UINT32  + cRD              }
#define module_serial_number             (0x6510A6 + UINT32  + cRD              }
#define module_type                      (0x6510A7 + UINT32  + cRD              }
#define module_revision                  (0x6510A8 + UINT32  + cRD              }
#define firmware_main_version            (0x6510A9 + UINT32  + cRD              }
#define firmware_custom_version          (0x6510AA + UINT32  + cRD              }
#define mdc_version                      (0x6510AB + UINT32  + cRD              }
#define firmware_type                    (0x6510AC + UINT32  + cRD              }
#define km_release                       (0x6510AD + UINT32  + cRD              }
#define cycletime_current_controller     (0x6510B0 + UINT32  + cRD              }
#define cycletime_velocity_controller    (0x6510B1 + UINT32  + cRD              }
#define cycletime_position_controller    (0x6510B2 + UINT32  + cRD              }
#define cycletime_trajectory_generator   (0x6510B3 + UINT32  + cRD              }
#define device_current                   (0x6510B8 + UINT32  + cRD              }
#define commissioning_state              (0x6510C0 + UINT32  + cRD + cWR        }
#define compatibility_control            (0x6510F0 + UINT16  + cRD + cWR        }

/*****************************************************************************/
/*   'magic' word for loading parameter                                      */
/*****************************************************************************/
#define cLOAD   0x64616F6C
```

```
#define cSAVE   0x65766173

/*****************************************************************************/
/*      modes of operation                                                   */
/*****************************************************************************/
#define cPositionMode    0x01
#define cVelocityMode    0x03
#define cTorqueMode      0x04
#define cHomingMode      0x06
#define cInterpolatedMode 0x07
#define cUnknownMode     0xFF

/*****************************************************************************/
/*      digital inputs                                                       */
/*****************************************************************************/
#define cEND0         0x00000001  /* negative limit switch              */
#define cEND1         0x00000002  /* positive limit switch              */
#define cHOME_SAMPLE   0x00000004  /* Home / Sample                     */
#define cLOCK         0x00000008  /* controller or power stage disabled */
#define cPOS0         0x01000000  /* Digital Input Target selector      */
#define cPOS1         0x02000000  /* Digital Input Target selector      */
#define cPOS2         0x04000000  /* Digital Input Target selector      */
#define cPOS3         0x08000000  /* Digital Input Target selector      */
#define cSTART        0x10000000
#define cSAMPLE       0x20000000

/* ——— Definition according to the connectors name———— */
#define cDIN0         cPOS0
#define cDIN1         cPOS1
#define cDIN2         cPOS2
#define cDIN3         cPOS3
#define cDIN5         cLOCK
#define cDIN6         cEND0
#define cDIN7         cEND1
#define cDIN8         cSTART
#define cDIN9         cSAMPLE

/*****************************************************************************/
/*      controlword                                                          */
/*****************************************************************************/
#define cwSHUT_DOWN           0x0006
#define cwSWITCH_ON           0x0007
#define cwDISABLE_VOLTAGE     0x0000
#define cwQUICK_STOP          0x0002
#define cwDISABLE_OPERATION   0x0007
#define cwENABLE_OPERATION    0x000F
#define cwFAULT_RESET         0x0080  /* Fault Reset with rising edge */

#define cwNEW_SET_POINT         0x0010
#define cwSTART_HOMING_OPERATION  0x0010
#define cwENABLE_IP_MODE        0x0010
#define cwCHANGE_SET_IMMEDIATLY  0x0020
#define cwABSOLUTE_RELATIV      0x0040
#define cwHOLD                  0x0100

/*****************************************************************************/
/*      statusword                                                           */
/*****************************************************************************/
/* state definition */
```

```
#define cdNOT_READY_TO_SWITCH_ON       0x0000
#define cdSWITCHED_ON_DISABLED         0x0040
#define cdREADY_TO_SWITCH_ON           0x0021
#define cdSWITCHED_ON                  0x0023
#define cdOPERATION_ENABLED            0x0027
#define cdFAULT                        0x000F
#define cdFAULT_REACTION_ACTIVE        0x000F
#define cdQUICK_STOP_ACTIVE            0x0007

/* Bits of status word */
#define swVOLTAGE_DISABLED     0x0010
#define swSWITCH_ON_DISABLED   0x0040
#define swWARNING              0x0080
#define swREMOTE               0x0200
#define swTARGET_REACHED       0x0400
#define swINTERNAL_LIMIT_ACTIVE 0x0800
#define swSET_POINT_ACKNOWLEDGE 0x1000
#define swSPEED0               0x1000
#define swHOMING_ATTAINED      0x1000
#define swIP_MODE_ACTIVE       0x1000
#define swFOLLOWING_ERROR      0x2000
#define swHOMING_ERROR         0x2000

/* position modes */
#define pCONTINOUS     0x0000
#define pIMMEDIATE     0x0020
#define pABSOLUTE      0x0000
#define pRELATIVE      0x0040

/* motion profile types */
#define mpLINEAR            0
```

10 Keyword index