metronix
servo drives

CANopen Manual

Servo positioning controller DIS-2

Metronix Meßgeräte und Elektronik GmbH
Kocherstraße 3
D-38120 Braunschweig
Germany

Phone: +49-(0)531-8668-0
Fax: +49-(0)531-8668-555
E-mail: vertrieb@metronix.de
http://www.metronix.de

Copyrights

The information and data in this document have been composed to the best of our knowledge. However, deviations between the document and the product cannot be excluded entirely. For the devices and the corresponding software in the version handed out to the customer, Metronix guarantees the contractual use in accordance with the user documentation. In the case of serious deviations from the user documentation, Metronix has the right and the obligation to repair, unless it would involve an unreasonable effort. A possible liability does not include deficiencies caused by deviations from the operating conditions intended for the device and described in the user documentation.

Metronix does not guarantee that the products meet the buyer's demands and purposes or that they work together with other products selected by the buyer. Metronix does not assume any liability for damages resulting from the combined use of its products with other products or resulting from improper handling of machines or systems.

Metronix Meßgeräte und Elektronik GmbH reserves the right to modify, amend, or improve the document or the product without prior notification.

This document may, neither entirely nor in part, be reproduced, translated into any other natural or machine-readable language nor transferred to electronic, mechanical, optical or any other kind of data media, without expressive authorisation by the author.

Trademarks

Any product names in this document may be registered trademarks. The sole purpose of any trademarks in this document is the identification of the corresponding products. ServoCommander™ is a registered trademark of Metronix Meßgeräte und Elektronik GmbH. *Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.*

Revision log			
Authors:	Metronix Meßgeräte und Elektronik GmbH		
Name of manual:	CANopen Manual		
Filename:	CAN_HB_DIS-2_2p0_EN.doc		
No.	Description	Revisions-index	Date of revision
001	Prerelease	0.1	26.09.2003
002	1. Version	1.0	19.12.2003
003	Translation	1.0	12-28-2005
004	Adjustments	1.1	2006-06-29
005	Update Corporate Identity – No technical changes	2.0	02.05.2011

Table of contents

Table of Figures

1 General Terms

1.1 Documentation

This manual describes how to parametrize and control the servo positioning controller DIS-2 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 DIS-2.

It contains safety notes that have to be noticed.

For more information, please refer to the manual of the servo positioning controller DIS-2.

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:

The following manual are available by this accociation:

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 servo positioning controller DIS-2 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

2 Safety Notes for electrical drives and controls

2.1 Symbols and signs

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 the case of damage resulting from non-compliance of the safety notes in this manual Metronix will assume any liability.

Prior to the initial use you must read the chapters Safety Notes for electrical drives and controls *starting on page 11*

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.

These safety instructions and all other user notes must be read prior to any work with 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 IP40..IP65 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 servo positioning controller 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 safety rules and regulations of the country in which the device will be operated must be complied with.

The environment conditions defined in the operating instructions 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 the operating instructions. 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.

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 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.

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.

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 asignment

3.1 Pin assignment

At the DIS-2 the CAN interface is already integrated in the device and therefore always available.

Dependend of the typ of hardware the CAN-bus can be found on different connectors. Additinall alternative modes for the analog and digital inputs are given cause of double layout of pins. For proper use of the CAN-bus it is recommended to do the according parameterization of the hardware.
More details can be found in the user manuals for the DIS-2.

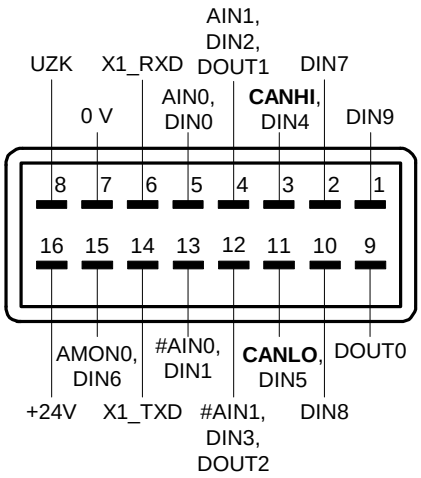


Figure 3.1: DIS-2 connector

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 servo positioning controller DIS-2

3.2 Cabling notes

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.

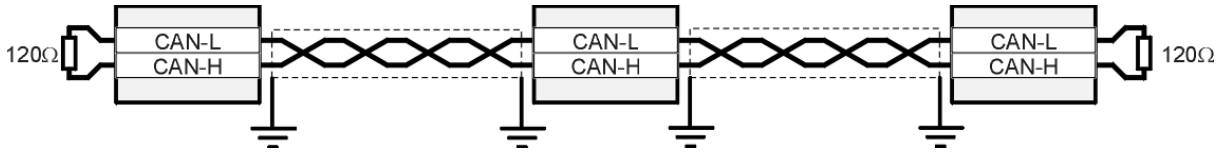


Figure 3.2: Cabling (schematically)

- 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 120Ω +/- 5%. Please note that such a resistor is often already installed in CAN cards or the PLC.
- It is recommended to ommit additional connectors inside the CAN cable. If this is not possible please use mettalized connector housings connected to the shield.
- For less noise injection principally
  - never place motor cables parallel to signal cables.
  - use only motor cables specified by METRONIX.
  - shield and earth motor cables correctly.
- For further information 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

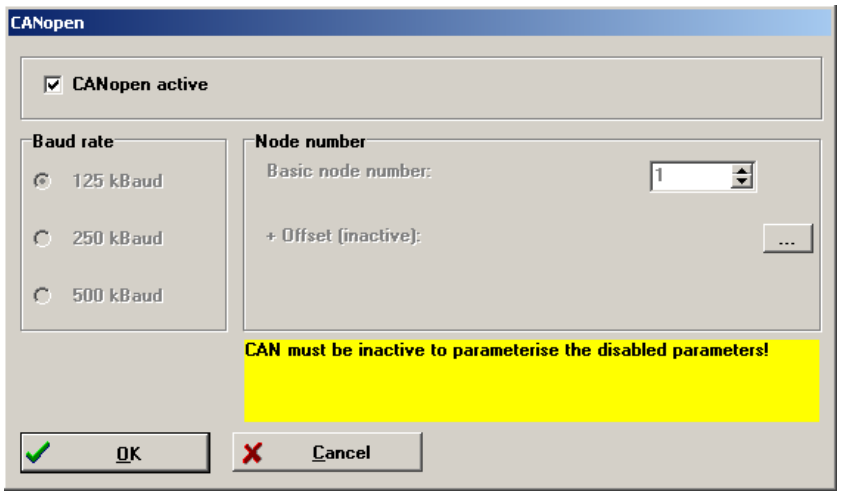
In this application no GND is used for connecting the slaves because of the same supply potential used also for the DC-Bus.

The shield is connected on both sides to the housing (PE).

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 Metronix ServoCommander.



There have to be set three 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 servo positioning controller. Therefore once after reset the combination of digital inputs DIN0 .. DIN3 dependend from the selection done in the evaluation of AIN/DIN in menu Digital inputs.

- **Baudrate**

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

Cabel length	Baudrate (kBaud)
< 100 m	500
< 250 m	250
< 500 m	125

- **Activation of CANopen**

  In this field the CAN communication can be enabled. For changing parameters the CAN communication needs to be inactive.

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

5 Acces 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 an unique *index* and *subindex* is assigned to every parameter (*CAN object*). The parameters altogether form the so called *object dictionary*.

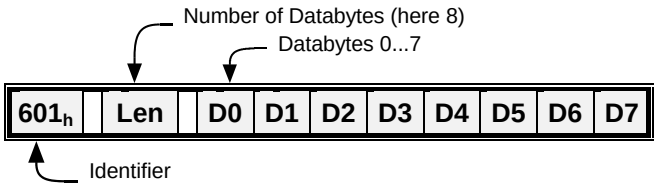
The object dictionary can be accessed via CAN bus in primarily two ways: A confirmed access with so called SDOs and a unconfirmed access using so called PDOs with no hand-shake.

As a rule the servo controller will be parametrized 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	**Em**ergen**cy** Message	Used to transmit error messages of the servo controller.
NMT	**N**etwork **M**anagemen**t**	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:



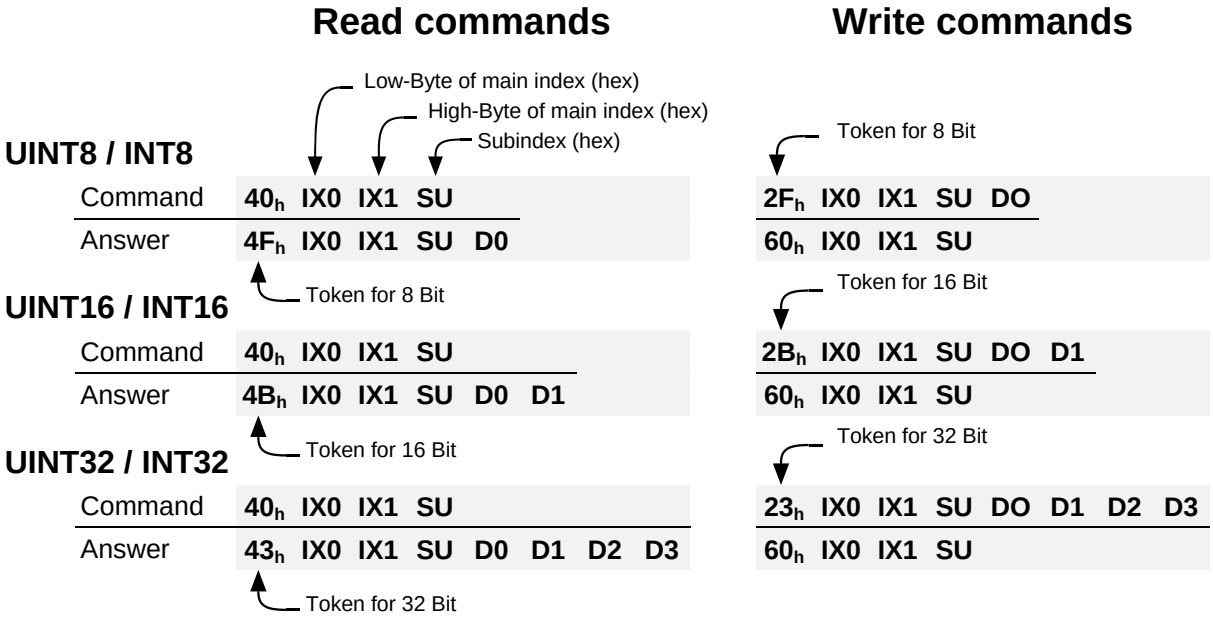
5.2 Access by SDO

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 its 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.



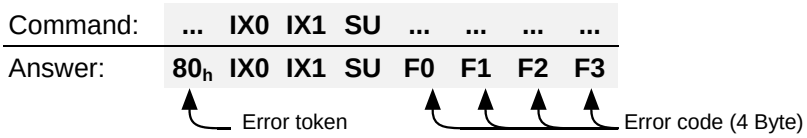
EXAMPLE

	Reading of Obj. 6061_00h Returning data: 01h	Writing of Obj. 1401_02h Data: EFh
UINT8 / INT8		
Command:	40h 61h 60h 00h	2Fh 01h 14h 02h EFh
Answer:	4Fh 61h 60h 00h 01h	60h 01h 14h 02h
	Reading of Obj. 6041_00h Returning data: 1234h	**Writing of Obj. 6040_00h Data: 03E8h**
UINT16 / INT16		
Command:	40h 41h 60h 00h	2Bh 40h 60h 00h E8h 03h
Answer:	4Bh 41h 60h 00h 34h 12h	60h 40h 60h 00h
	Reading of Obj. 6093_01h Returning data: 12345678h	**Writing of Obj. 6093_01h Data: 12345678h**
UINT32 / INT32		
Command:	40h 93h 60h 01h	23h 93h 60h 01h 78h 56h 34h 12h
Answer:	43h 93h 60h 01h 78h 56h 34h 12h	60h 93h 60h 01h

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 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
06 01 00 00_h	Unsupported access to an 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 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 01 00 01_h	Attempt to read a write only object
06 01 00 02_h	Attempt to write a read only object
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
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 *[2)]

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

*[2)] „Device State“ is used generally: For instance a wrong operation mode can be choosen or the number of mapped object is written while the PDO is active.

5.3 PDO-Message

Process **D**ata **O**bjects (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 parametrized 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.

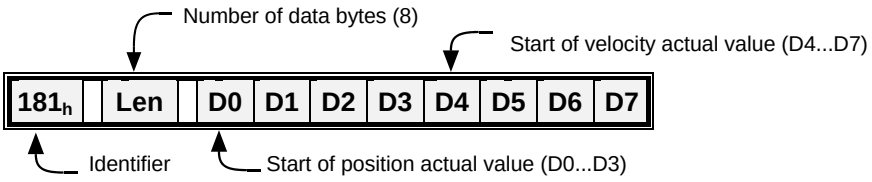
Following types of PDOs can be differenced:

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.



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 necessary to have bit 30 set always because no support for Remote Transmit is given.

It is prohibited to change the COB-ID if the PDO is not deactivated, i.e. bit 31 is set. Therefore the following sequence has to be used to change the COB-ID:

- Read the COB-ID out of the servo
- Write back the written COB-ID + $C0000000_h$
- Write the new COB-ID + $C0000000_h$
- Write the new COB-ID + 40000000_h, the PDO is active again.

Number of objects to be transmitted

number_of_mapped_objects

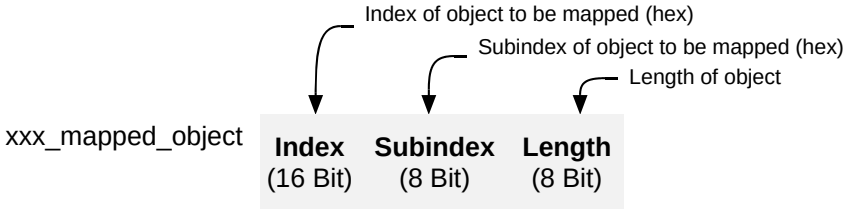
The object determines how many objects are mapped into the specific PDO. Following restrictions has to be respected:

- 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.

Transmissiontype

transmission_type und **inhibit_time**

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

Value	Description	Allowed with
00_h –$F0_h$	**SYNC message** The value determines how many SYNC messages will be ignored 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. The Receive-PDO is evaluated imidiatly	TPDOs RPDOs

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 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.) Set number of mapped objects to 0

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

2.) 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 = $60FD0020_h$**

3.) Set number of mapped objects:

The PDO contains 3 objects ⇒ **number_of_mapped_objects = 3_h**

4.) 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**

5.) Parametrize the identifier

The PDO should use identifier 187h.
If the PDO is activated, it has to be disabled at first.

Read the identifier: ⇒ **00000181_h = cob_id_used_by_pdo**

Set bit 31 (deactivate): ⇒ **cob_id_used_by_pdo = $C0000181_h$**

Write new identifier: ⇒ **cob_id_used_by_pdo = $C0000187_h$**

Activate by deleting bit 31: ⇒ **cob_id_used_by_pdo = 40000187_h**

Parametrising PDOs

Please note that it is only allowed to change the settings of the PDO if the Network state (NMT) is not **operational**. See also chapter 5.3.3

5.3.2 Objects for parameterising PDOs

The servo positioning controllers contain 4 Transmit- and 4 Receive-PDOs. The objects for parameterising these PDOs are equal for each 2 TPDOs and each 2 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, Bit 30 must be set because no remote transmit is supported
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	4

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	**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

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

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	$FFFFFFFF_h$

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

5.3.3 Activation of PDOs

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

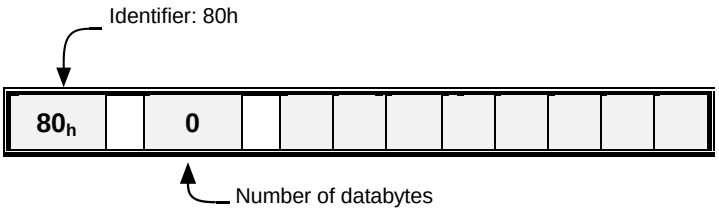
- The object **number_of_mapped_objects** has to be different from zero
- The bit 32 has to be deleted in the object **cob_id_used_for_pdos**
- The communication status of the servo has to be **operational** (see chapter 5.7, Network management)

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 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

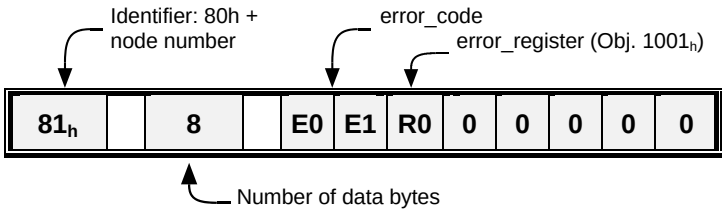
Access	rw
PDO Mapping	no
Units	
Value Range	80000080_h, 00000080_h
Default Value	00000080_h

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.).

5.5.1 Structure of an EMERGENCY message

If an error occurs the servo controller always sends an EMERGENCY message. The identifier of this message is **080_h** plus node number.
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	Anzeige (hex)	Bedeutung
3	4310	Overtemperature motor
4	4210	Overtemperature of the power stage
5	7392	SINCOS-supply
6	7391	SINCOS-RS485-communication
7	7390	SINCOS-encoder signal
8	7380	resolver
9	5113	5V-supply
10	5114	12V-supply
11	5112	24V-supply(out of range)
13	5210	Offset current measuring
14	2320	Over current in power stage
15	3220	Undervoltage DC-bus
16	3210	Overvoltage DC-bus
17	7385	Hallsensor
19	2312	I^2t- motor (I^2t is 100%)
20	2311	I^2t- servo (I^2t is 100%)
26	2380	I^2t is 80%
27	4380	Temperature motor 5°C below maximum
28	4280	Temperatur servo 5°C below Maximum
29	8611	Following error
31	8612	Limit switch
35	6199	Timeout during quickstop

error_code	Anzeige (hex)	Bedeutung
36	8A80	Homing
40	6197	Motor- and encoder identification
43	6193	Course programm unknown command
44	6192	Course programm unknown jump target
56	7510	RS232-communication
57	6191	Positioning set
58	6380	Operating mode
60	6190	Precalculation for position profile
62	6180	Stack-Overflow
63	5581	Checksummen
64	6187	Initialisation

5.5.2 Description of Objects

5.5.2.1 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	**00_h**
Description	**pre_defined_error_field**
Data Type	UINT8
Access	Rw
PDO Mapping	No
Units	--
Value Range	0 (write access): clear error buffer 0..4 (read access): errors in error buffer
Default Value	--

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	--

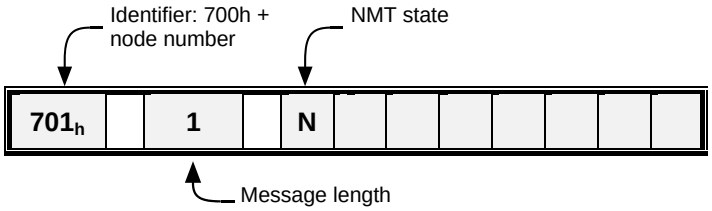
Sub-Index	**03_h**
Description	**standard_error_field_2**
Access	ro
PDO Mapping	no
Units	--
Value Range	--
Default Value	--

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

5.6 Heartbeat / Bootup (Error Control Protocol)

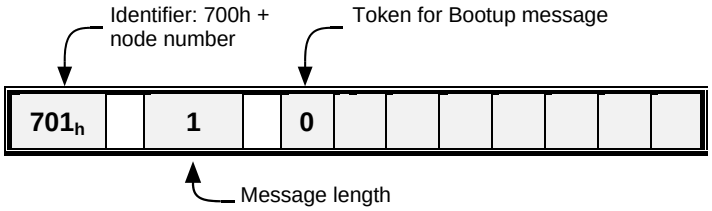
5.6.1 Structure of the heartbeat message

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. The heartbeat message will be sent with the identifier **700_h + node number**. It is only composed of 1 Byte, containing the NMT state of the servo (see Chapter 5.7, Network management).



5.6.2 Structure of the Bootup message

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 5.7, Network management)



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

5.6.3 Objects

5.6.3.1 Object 1017_h: producer_heartbeat_time

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

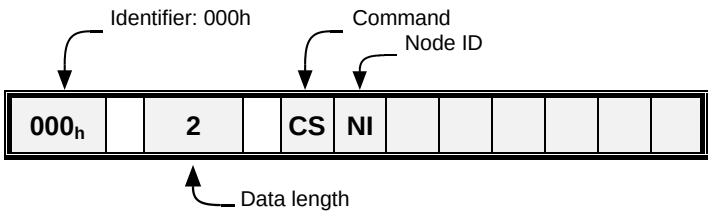
Index	**1017_h**
Name	**producer_heartbeat_time**
Object Code	VAR
Data Type	UINT16

Access	rw
PDO Mapping	no
Units	ms
Value Range	0...65536
Default Value	0

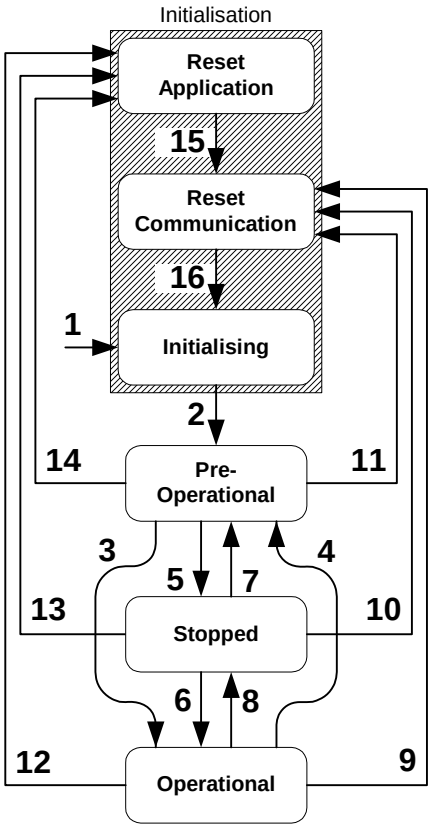
The **producer_heartbeat_time** can be saved in the parameter set. If the servo starts with a **producer_heartbeat_time** unequal zero, the Bootup messages is seen as the first heatbeat.

5.7 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.



Figgure 5.3: 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**
02_h	Stop Remote Node	5, 8	**Stopped**
80_h	Enter Pre-Operational	4, 7	**Pre-Operational**
81_h	Reset Application	12, 13, 14	**Reset Application**
82_h	Reset Communication	9, 10, 11	**Reset Communication**

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

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

5.8 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**	
RPDO1	**201_h**	
RPDO2	**301_h**	
SYNC	**080_h**	
EMCY	**080_h + node id**	
HEARTBEAT	**700_h + node id**	
BOOTUP	**700_h + node id**	
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.

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 Metronix ServoCommander or via the CAN bus. When the servo controller is switched on the **application parameter set** is copied into the **current parameter set** .

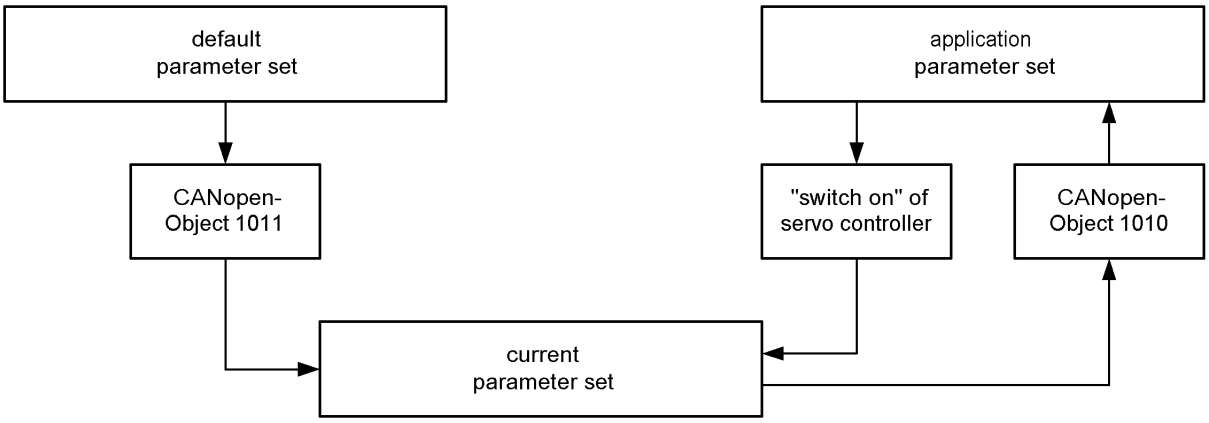
Default parameter set

- 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 $\mathbf{1011_h\_01_h}$ (**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 $\mathbf{1010_h\_01_h}$ (**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 DIS-2 ServoCommander and also transferred to the single servo controller by the parameter set-up program DIS-2 ServoCommander. 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-parametrized servo controllers and the parameter set-up software Metronix ServoCommander is not necessary for this.

It is urgently recommended to use method 2. But in this case it could happen that not all parameters can be set by the CAN-bus. If this is the case the first methode should be engaged.

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 parameter for the CAN communication (node number, baudrate and mode) 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

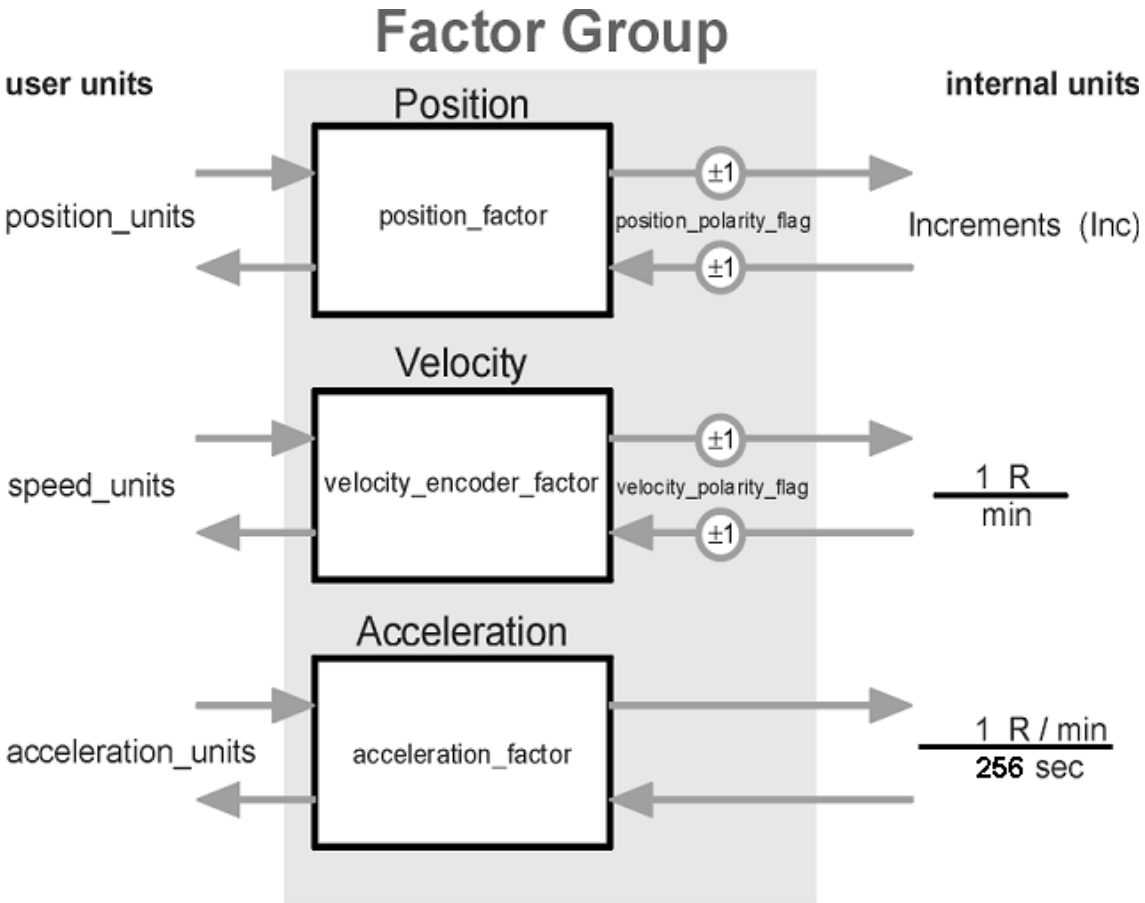
To store the **default parameter set** as **application parameter set**, the object **1010_h_01_h** (**save_all_parameters**) must be used additionally.

6.2 Conversion factors (Factor Group)

6.2.1 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 parametrized 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
Acceleration	acceleration_units	**min^{-1}/256s**	Increase of velocity per 256 seconds

6.2.2 Description of Objects

6.2.2.1 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

6.2.2.2 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:

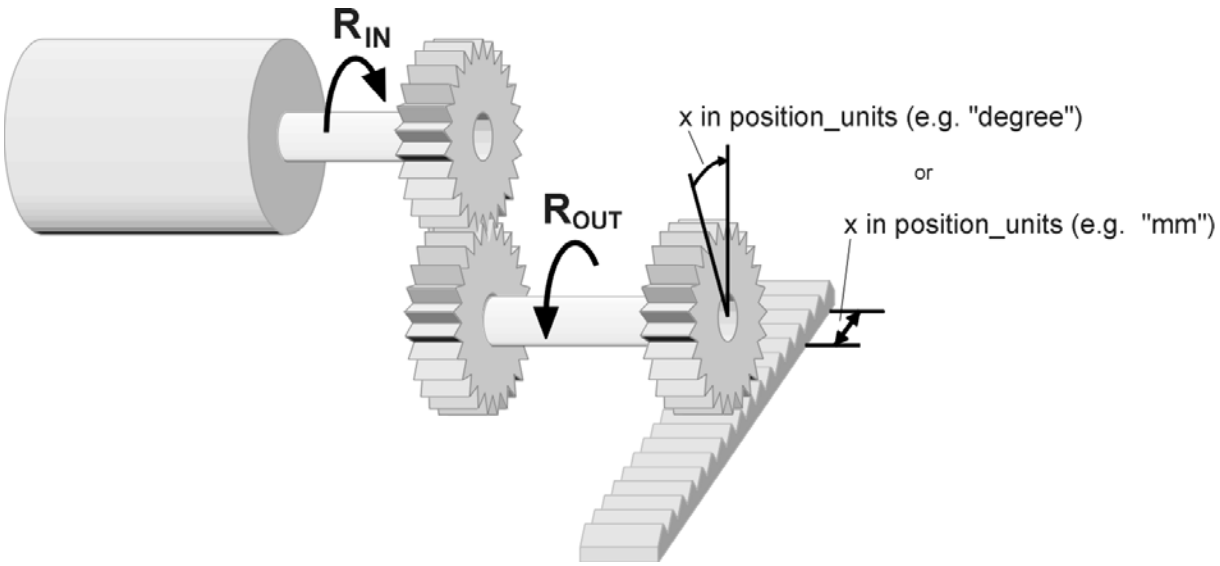


Figure 6.4: Survey: Factor Group

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:

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

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

EXAMPLE

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	**1** R_{OUT} = **65536** Inc	**1/1**	$\frac{65536\frac{Inc}{R}\cdot\frac{1R}{1R}\cdot\frac{1}{1}}{\frac{65536\,Inc}{1R}}=\frac{\mathbf{1}Inc}{\mathbf{1}Inc}$	**num: 1** **div: 1**
1/10 degree ($^{degree}/_{10}$)	**1** R_{OUT} **= 3600** $^{degree}/_{10}$	**1/1**	$\frac{65536\frac{Inc}{R}\cdot\frac{1R}{1R}}{\frac{3600\,^{\circ}/_{10}}{1R}}=\frac{\mathbf{65536}\,Inc}{\mathbf{3600}\,^{\circ}/_{10}}$	**num: 4096** **div: 225**
1/100 Rev. ($^{R}/_{100}$)	**1** R_{OUT} **= 100** $^{R}/_{100}$	**1/1**	$\frac{65536\frac{Inc}{R}\cdot\frac{1R}{1R}}{\frac{100\,R/_{100}}{1R}}=\frac{\mathbf{65536}\,Inc}{\mathbf{100}\,R/_{100}}$	**num: 16384** **div: 25**
1/100 Rev. ($^{R}/_{100}$)		**2/3**	$\frac{65536\,\frac{Inc}{R}\cdot\frac{2R}{3R}}{\frac{100\,R/_{100}}{1R}}=\frac{\mathbf{131072}\,Inc}{\mathbf{300}\,R/_{100}}$	**num: 32768** **div: 75**
1/10 mm ($^{mm}/_{10}$)	**1** R_{OUT} **= 631.5** $^{mm}/_{10}$	**1/1**	$\frac{65536\frac{Inc}{R}\cdot\frac{1R}{1R}}{\frac{631.5\,mm/_{10}}{1R}}=\frac{\mathbf{655360}\,Inc}{\mathbf{6315}\,mm/_{10}}$	**num: 131072** **div: 1263**
1/10 mm ($^{mm}/_{10}$)		**4/5**	$\frac{65536\frac{Inc}{R}\cdot\frac{4R}{5R}}{\frac{631.5\,mm/_{10}}{1R}}=\frac{\mathbf{2621440}\,Inc}{\mathbf{31575}\,mm/_{10}}$	**num: 524288** **div: 6315**

6.2.2.3 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 seconds**. 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.

gear_ratio — Ratio between revolutions on the driving side (RIN) and revolutions on the driven side (ROUT).

feed_constant — Ratio between revolutions on the driven side (ROUT) and equivalent motion in position_units (e.g. 1 rev = 360°)

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

$$\textbf{velocity\_encoder\_factor} = \frac{\textbf{numerator}}{\textbf{divisor}} = \frac{\textbf{65536} \cdot \textbf{gear\_ratio} \cdot \textbf{time\_factor\_v}}{\textbf{feed\_constant}}$$

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

1. Desired unit on the driven side (position_units)
2. feed_constant: How many position_units are 1 revolution(ROUT)
3. time_factor_v: Desired time unit contains how many seconds ?
4. gear_ratio: RIN per ROUT
5. Calculate equation

1.	2.	3.	4.	5.	ERGEBNIS Gekürzt
$^{R}/_{min}$	**1** R_{OUT} = **65536** Inc	**1** $\frac{1}{\text{min}}$ = **4096** $\frac{1}{4096\,\text{min}}$	**1/1**	$\frac{\frac{1R}{1R}\cdot\frac{1R}{1R}\cdot\frac{4096\,{}^{1}/_{4096min}}{1\,{}^{1}/_{1min}}}{\frac{1R}{1R}} = \frac{4096\,{}^{R}/_{4096min}}{1\,{}^{R}/_{min}}$	**num: 4096** **div: 1**
1/10 $^{\text{degree}}/_{s}$ ($^{\text{degree}}/_{10s}$)	**1** R_{OUT} = **3600** $^{\text{degree}}/_{10}$	**1** $\frac{1}{s}$ = **1/60** $\frac{1}{\text{min}}$ = **4096/60** $\frac{1}{4096\,\text{min}}$	**1/1**	$\frac{\frac{1R}{1R}\cdot\frac{1R}{1R}\cdot\frac{4096\,{}^{1}/_{4096min}}{60\,{}^{1}/_{1min}}}{\frac{3600\,{}^{\text{degree}}/_{10}}{1R}} = \frac{4096\,{}^{R}/_{4096min}}{216000\,{}^{\text{degree}}/_{10s}}$	**num: 16** **div: 3375**
1/100 $^{R}/_{min}$ ($^{R}/_{100\ \text{min}}$)	**1** R_{OUT} = **100** $^{U}/_{100}$	**1** $\frac{1}{\text{min}}$ = **4096** $\frac{1}{4096\,\text{min}}$	**1/1**	$\frac{\frac{1R}{1R}\cdot\frac{1R}{1R}\cdot\frac{4096\,{}^{1}/_{4096min}}{1\,{}^{1}/_{1min}}}{\frac{100\,{}^{R}/_{100}}{1R}} = \frac{4096\,{}^{R}/_{4096min}}{100\,{}^{R}/_{100min}}$	**num: 1024** **div: 25**
1/100 $^{R}/_{min}$ ($^{R}/_{100\ \text{min}}$)			**2/3**	$\frac{\frac{1R}{1R}\cdot\frac{2R}{3R}\cdot\frac{4096\,{}^{1}/_{4096min}}{1\,{}^{1}/_{1min}}}{\frac{100\,{}^{R}/_{100}}{1R}} = \frac{8192\,{}^{R}/_{4096min}}{300\,{}^{R}/_{100min}}$	**num: 2048** **div: 75**
1/10 $^{mm}/_{s}$ ($^{mm}/_{10s}$)	**1** R_{OUT} = **631.5** $^{mm}/_{10}$	**1** $\frac{1}{s}$ = **1/60** $\frac{1}{\text{min}}$ = **4096/60** $\frac{1}{4096\,\text{min}}$	**1/1**	$\frac{\frac{1R}{1R}\cdot\frac{1R}{1R}\cdot\frac{4096\,{}^{1}/_{4096min}}{60\,{}^{1}/_{1min}}}{\frac{631.5\,{}^{mm}/_{10}}{1R}} = \frac{4096\,{}^{R}/_{4096min}}{37890\,{}^{mm}/_{10s}}$	**num: 2048** **div: 18945**
1/10 $^{mm}/_{s}$ ($^{mm}/_{10s}$)			**4/5**	$\frac{\frac{1R}{1R}\cdot\frac{2R}{3R}\cdot\frac{4096\,{}^{1}/_{4096min}}{1\,{}^{1}/_{1min}}}{\frac{631.5\,{}^{R}/_{100}}{1R}} = \frac{16384\,{}^{R}/_{4096min}}{3789\,{}^{R}/_{100min}}$	**num: 16384** **div: 3789**

6.2.2.4 Object 6097_h: acceleration_factor

The object **acceleration_factor** converts all acceleration values of the application from **acceleration_units** into the internal unit **increments per second²** (65536 Increments equals 1 Revolution). 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 **velocity_encoder_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 seconds² to minutes²). 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
(e.g. **1 min²** = 1 min·1min = **60s · 1min**

gear_ratio — Ratio between revolutions on the driving side (RIN) and revolutions on the driven side (ROUT).

feed_constant — Ratio between revolutions on the driven side (ROUT) and equivalent motion in position_units (e.g. 1 rev = 360°°)

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

$$\textbf{acceleration\_factor} = \frac{\textbf{numerator}}{\textbf{divisor}} = \frac{\textbf{65536} \cdot \textbf{gear\_ratio} \cdot \textbf{time\_factor\_a}}{\textbf{feed\_constant}}$$

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

1. Desired unit on the driven side (position_units)
2. feed_constant: How many position_units are 1 revolution(ROUT)
3. time_factor_v: Desired (time unit)2 contains how many seconds2 ?
4. gear_ratio: RIN per ROUT
5. Calculate equation

1.	2.	3.	4.	5.	Result shortened
$^{R}/_{\text{min s}}$ ($^{R}/_{\text{min s}}$)	**1** R_{OUT} = **1** R_{IN}	**1** $\frac{1}{\text{min} \cdot \text{s}}$ = **256** $\frac{1}{\frac{\text{min}}{256 \cdot \text{s}}}$	**1/1**	$\frac{\frac{1R}{1R} \cdot \frac{1R}{1R} \cdot \frac{256\,{}^{1}/_{256\,min \cdot s}}{1\,{}^{1}/_{1min \cdot s}}}{\frac{1R}{1R}} = \frac{\mathbf{256}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{1}\,{}^{R}/_{min}/_{s}}$	**num: 256** **div: 1**
1/10 $^{\textbf{degree}}/_{\textbf{s}^2}$ ($^{\textbf{degree}}/_{\textbf{10s}^2}$)	**1** R_{OUT} = **3600** $^{\textbf{degree}}/_{\textbf{10}}$	**1** $\frac{1}{s^2}$ = **1/60** $\frac{1}{\text{min} \cdot \text{s}}$ = **256/60** $\frac{1}{\frac{\text{min}}{256 \cdot \text{s}}}$	**1/1**	$\frac{\frac{1R}{1R} \cdot \frac{1R}{1R} \cdot \frac{256\,{}^{1}/_{256\,min \cdot s}}{60\,{}^{1}/_{min \cdot s}}}{\frac{3600\ {}^{deg\,ree}/_{10}}{1R}} = \frac{\mathbf{256}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{216000}\ {}^{deg\,ree}/_{10s^2}}$	**num: 4** **div: 3375**
1/100 $^{R}/_{\text{min}^2}$ ($^{R}/_{\textbf{100 min}^2}$)	**1** R_{OUT} = **100** $^{R}/_{100}$	**1** $\frac{1}{\text{min}^2}$ = **60** $\frac{1}{\text{min} \cdot \text{s}}$ = **256·60** $\frac{1}{\frac{\text{min}}{256 \cdot \text{s}}}$	**1/1**	$\frac{\frac{1R}{1R} \cdot \frac{1R}{1R} \cdot \frac{15360\,{}^{1}/_{256\,min \cdot s}}{1\,{}^{1}/_{min \cdot min}}}{\frac{100\ {}^{R}/_{100}}{1R}} = \frac{\mathbf{15360}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{100}\ {}^{R}/_{100min^2}}$	**num: 3840** **div: 25**
1/100 $^{R}/_{\text{min}^2}$ ($^{R}/_{\textbf{100 min}^2}$)			**2/3**	$\frac{\frac{1R}{1R} \cdot \frac{2R}{3R} \cdot \frac{15360\,{}^{1}/_{256\,min \cdot s}}{1\,{}^{1}/_{min \cdot min}}}{\frac{100\ {}^{R}/_{100}}{1R}} = \frac{\mathbf{30720}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{300}\ {}^{R}/_{100min^2}}$	**num: 2560** **div: 25**
1/10 $^{\textbf{mm}}/_{\textbf{s}^2}$ ($^{\textbf{mm}}/_{\textbf{10s}^2}$)	**1** R_{OUT} = **631.5** $^{\textbf{mm}}/_{\textbf{10}}$	**1** $\frac{1}{s^2}$ = **1/60** $\frac{1}{\text{min} \cdot \text{s}}$ = **256/60** $\frac{1}{\frac{\text{min}}{256 \cdot \text{s}}}$	**1/1**	$\frac{\frac{1R}{1R} \cdot \frac{1R}{1R} \cdot \frac{256\,{}^{1}/_{256\,min \cdot s}}{60\,{}^{1}/_{min \cdot min}}}{\frac{631.5\ {}^{mm}/_{10deg\,ree}}{1R}} = \frac{\mathbf{256}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{37890}\ {}^{R}/_{100min^2}}$	**num: 128** **div: 18945**
1/10 $^{\textbf{mm}}/_{\textbf{s}^2}$ ($^{\textbf{mm}}/_{\textbf{10s}^2}$)			**4/5**	$\frac{\frac{1R}{5R} \cdot \frac{4R}{5R} \cdot \frac{256\,{}^{1}/_{256\,min \cdot s}}{60\,{}^{1}/_{min \cdot min}}}{\frac{631.5\ {}^{mm}/_{10deg\,ree}}{1R}} = \frac{\mathbf{1024}\,{}^{R}/_{min}/_{256 \cdot s}}{\mathbf{189450}\ {}^{R}/_{100min^2}}$	**num: 512** **div: 94725**

6.2.2.5 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)

6.3 Power stage parameters

6.3.1 Survey

The motor is fed from the intermediate circuit via the IGBTs. The power stage contains a number of security functions which can be parametrized in part:

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

6.3.2 Description of Objects

Index	Object	Name	Typ	Attr.
6510_h	VAR	drive_data		

6.3.2.1 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 control the enabling via the CAN bus. To do that the object **6510_h_10_h** (**enable_logic**) has to be set to 2 .

Index	**6510_h**
Name	**drive_data**
Object Code	RECORD
No. of Elements	44

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 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!

6.4.1 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 Metronix ServoCommander 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!

6.4.2 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	UINT32	rw
$60F6_h$	RECORD	torque_control_parameters	UINT16	rw

6.4.2.1 Object 6075_h: motor_rated_current

This value can be read on the motor plate and is specified in mA (effective value, RMS). The value is entered as root main square (RMS) value. It is not possible to enter values higher than the nominl current of the servo.

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	2870

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

6.4.2.2 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 peak_current of the servo. 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	1968

6.4.2.3 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 DIS-2 ServoCommander. 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.

Index	**$604D_h$**
Name	**Pole_number**
Object Code	VAR
Data Type	UINT8

Access	Rw
PDO Mapping	Yes
Units	--
Value Range	2... 128
Default Value	2

6.4.2.4 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

6.4.2.5 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	--

6.4.2.6 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 Metronix ServoCommander. 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

6.4.2.7 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 Metronix ServoCommander. The angle determined by the parameter set-up program Metronix ServoCommander is in the range of +/-180°. It has to be converted as follows to be written into the object **resolver_offset_angle**:

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

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	$2C60_h$ (62,4°)

6.4.2.8 Object 2415_h: current_limitation

The record **current_limitation** allows the limitation of the maximum current independent of the mode of operation (velocity control, positioning) whereby torque limited speed control is possible. The source of the torque limit can be chosen by the object **limit_current_input_channel**. Possibly sources for the torque limit are Fieldbus, RS232 or an analogue input. Depending on the chosen source the object **limit_current** determines the torque limit (Source = Fieldbus / RS232) or the scaling factor for the analogue input (Source = Analogue input). In the first case the current limit in mA can be entered directly, in the later 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	INT8
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	5650

Value	Description
0	No limitation
1	AIN0
2	AIN1
3	RS232
4	CAN

6.4.2.9 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 DIS-2 ServoCommander. 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	256

Sub-Index	**02_h**
Description	**torque_control_time**
Data Type	UINT16
Access	Rw
PDO Mapping	No
Units	µs
Value Range	100... 65500
Default Value	2000

6.5 Velocity controller

6.5.1 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 DIS-2 ServoCommander 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!

6.5.2 Description of Objects

Index	Object	Name	Type	Attr.
$60F9_h$	RECORD	velocity_control_parameters		rw

6.5.2.1 Object $60F9_h$: velocity_control_parameters

The data of the velocity controller can be taken from the parameter set-up program DIS-2 ServoCommander. 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 DIS-2 ServoCommander 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 DIS-2 ServoCommander. 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	26…64*256 (16384)
Default Value	179 (0.7)

Sub-Index	**02$_h$**
Description	**Velocity_control_time**
Data Type	UINT16
Access	Rw
PDO Mapping	No
Units	µs
Value Range	200...32000
Default Value	8000

Sub-Index	**04$_h$**
Description	**velocity_control_filter_time**
Data Type	UINT16
Access	rw
PDO Mapping	no
Units	µs
Value Range	200...32000
Default Value	1600

6.6 Position Control Function

6.6.1 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**.

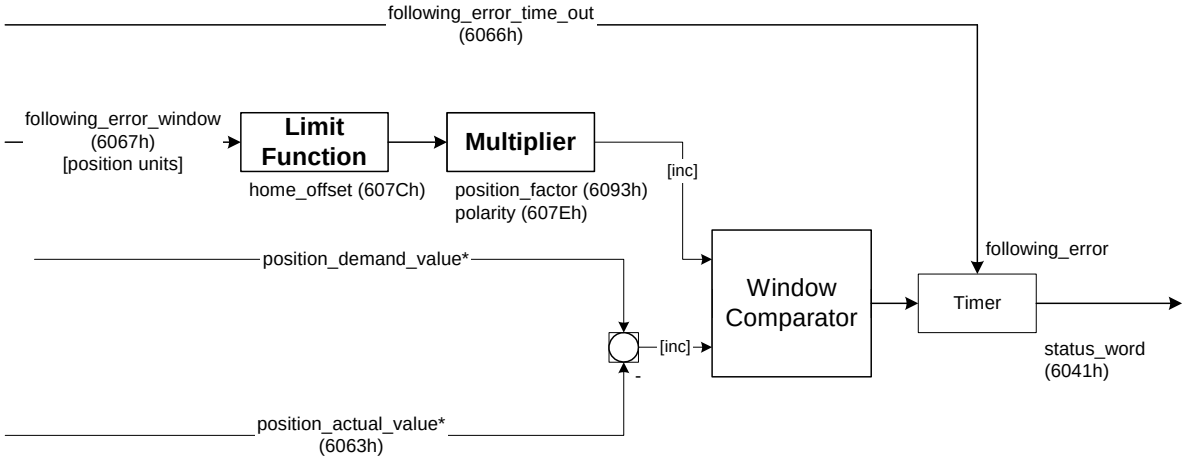


Figure 6.5: Trailing error (Following Error) – Function Survey

Figure 6.6 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.

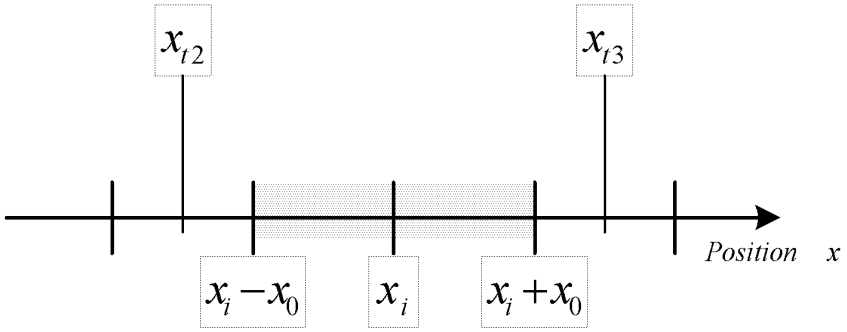


Figure 6.6: Trailing error (following error)

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.**

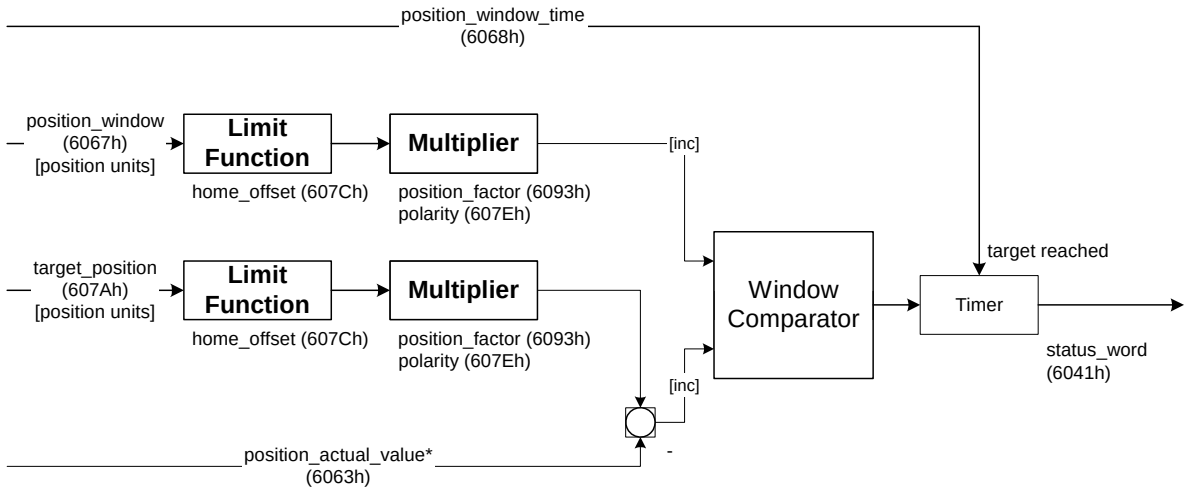


Figure 6.7: Position Reached – Function Survey

Figure 6.8 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.

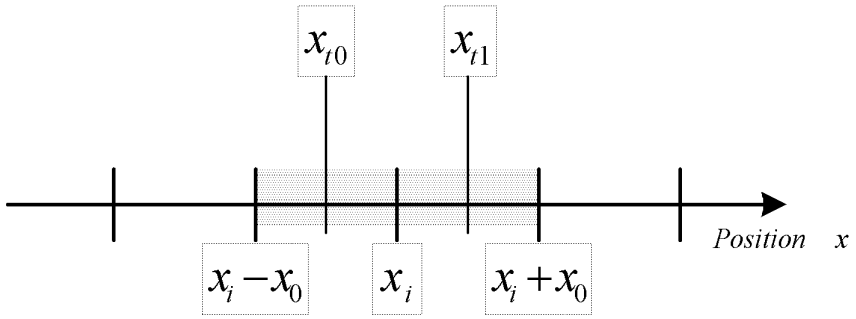


Figure 6.8: Position reached

6.6.2 Description of Objects

6.6.2.1 Objects treated in this chapter

Index	Object	Name	Type	Attr.
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
$60FA_h$	VAR	control_effort	INT32	ro
$60FB_h$	RECORD	position_control_parameter_set		rw

6.6.2.2 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«
$607E_h$	VAR	polarity	UINT8	6.2 Conversion factors (Factor Group)
6093_h	VAR	position_factor	UINT32	6.2 Conversion factors (Factor Group)
6094_h	ARRAY	velocity_encoder_factor	UINT32	6.2 Conversion factors (Factor Group)
6040_h	VAR	controlword	INT16	6.10. Device Control
6041_h	VAR	statusword	UINT16	6.10. Device Control

6.6.2.3 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 DIS-2 ServoCommander.

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.
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 DIS-2 ServoCommander the value $384 = 180_h$ has to be written into the object **position_control_gain**.

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	52 (0,20)

Sub-Index	**04$_h$**
Description	**position_control_v_max**
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	speed units
Value Range	0...32767 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	0...65536 (1 R)
Default Value	13 (0,00020 R)

6.6.2.4 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	--

6.6.2.5 Objekt 6064_h: position_actual_value

The actual position can be read by this objects. This value is given to the position controller by the angle encoder.

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	--

6.6.2.6 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** parametrized too small
- The position controller is not parametrized 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	$238E_h$ (approx. 50 degree)

6.6.2.7 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...26214
Default Value	100

6.6.2.8 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	--

6.6.2.9 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°)

6.6.2.10 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...262146
Default Value	100

6.7 Analogue inputs

6.7.1 Survey

The servo controller of the DIS-2 series contains two analogue inputs, which can be used to enter a demand value for instance. The analogue inputs can only be parameterized by the DIS-2 ServoCommander.

6.8 Digital In- and Outputs

6.8.1 Survey

All digital inputs can be read by the CAN bus. Two of digital outs can be set by the CAN bus.

6.8.2 Description of Objects

Index	Object	Name	Type	Attr.
$60FD_h$	VAR	digital_inputs	UINT32	ro
$60FE_h$	ARRAY	digital_outputs	UINT32	rw

6.8.2.1 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
3	00000008_h	Interlock (“controller enable” (DIN9) is missing)
16...25	$03FF0000_h$	DIN0…DIN9

6.8.2.2 Object $60FE_h$: digital_outputs

The digital outputs can be set via the object **$60FE_h$**. Then the 2 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	0

Bit	Value	Digital Outputs
0	00000001_h	Brake
16	00010000_h	Operational
17, 18	00060000_h	DOUT1, DOUT2

6.9 Limit switches

6.9.1 Survey

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

6.9.2 Description of Objects

Index	Object	Name	Type	Attr.
6510h	RECORD	drive_data		rw

6.9.2.1 Object $6510_h\_11_h$: limit_switch_polarity

The polarity of the limit switches can be parametrized 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. This is valid for both limit switches.

Index	**6510_h**
Name	**drive_data**
Object Code	RECORD
No. of Elements	44

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)

6.9.2.2 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...200000 min^{-1}/s
Default Value	200000 min^{-1}/s

6.10 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.

6.10.1 Description of Objects

6.10.1.1 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	0000003B
Default Value	0000003B

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
1121_h	DIS-2 48/10
1122_h	DIS-2 24/8

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	--

6.10.1.2 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

6.10.1.3 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	--

6.10.1.4 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	--

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 combination 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

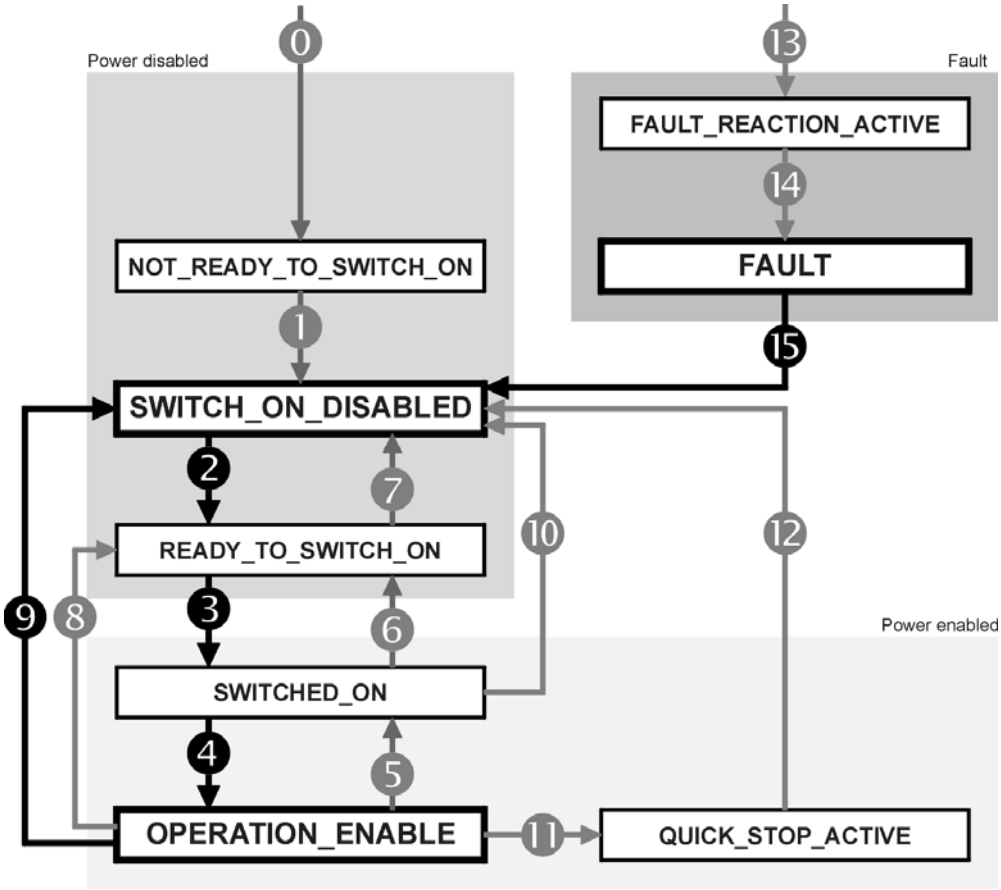


Figure 7.9: State diagram of the servo controller

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 parametrized (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 parametrized mode of operation. Therefore previously ensure that the servo controller has been parametrized correctly and the according demand value is zero.

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" applying + Command **Shutdown**	**Shutdown** =	**x**	**1**	**1**	**0**	None
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 **modes_of_operation**
9	Command **Disable Voltage**	**Disable Voltage** =	**x**	**x**	**0**	**x**	Power stage is disabled. The motor is freely rotatable
15	Cause of fault remedied + Command **Fault Reset**	**Fault Reset** =	Bit 7 = ↑				Reset fault

Figure 7.10: Most important state transitions

EXAMPLE

After the servo controller has been parametrized 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 7.9) the state transitions 2, 3 and 4 have to be executed.

4.) From Figure 7.10 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 input DIN9 "Enable controller" is connected to 24V. (controller enable logic is set to “digital inputs and CAN”)
SWITCHED_ON *1)	The power stagecan be switched on.
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" applying + Command **Shutdown**	**Shutdown** =	**x**	**1**	**1**	**0**	None
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**	

No.	Executed if	Bit combination (controlword)						Action
		Bit		3	2	1	0	
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 parametrized 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
FAULT	**0**	×	**1**	**1**	**1**	**1**	$004F_h$	$000F_h$
FAULT_REACTION_ACTIVE	**0**	×	**1**	**1**	**1**	**1**	$004F_h$	$000F_h$
QUICK_STOP_ACTIVE	**0**	**0**	**0**	**1**	**1**	**1**	$006F_h$	0007_h

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

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	0010_h	voltage_enabled
5	0020_h	State of the servo controller (see Table 7.3)
6	0040_h	
7	0080_h	warning
8	0100_h	unused
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	unused
15	8000_h	reserved

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_disable**	
		This bit is set if the transistors of the power stage switched off.
		 **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 undefined. It must not be evaluated.
Bit 8	**manufacturer specific**	
		This bit is undefined. It must not be evaluated.
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.
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 parametrized 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 parametrized velocity window. This window can be adjusted by the DIS-2 ServoCommander.
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.
	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.
Bit 14	**unused**	
		This bit is unused at present. It must not be evaluated.
Bit 15	**reserved**	manufacturer specific
		This bit must not be evaluated.

8 Operating Modes

8.1 Adjustment of the Operating Mode

8.1.1 Survey

The servo controllers of the ARS 2000 series 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**. By reading this object the last sended mode is read out. This is not corresponding to the current operation mode of the servo control

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**. An internal mode is readed if internal slelctors are set in that way that no CANopen mode is possible until a CANopen spezific mode is selected.

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, -11, -12, -13, -14, -15
Default Value	3

Value	Operation mode
-1	Unknown operating mode under CANopen
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
-11	Internal positioning mode
-12	Internal velocity control without ramps
-13	Internal velocity control with ramps
-14	Internal torque mode
-15	Internal position controller active

The operating mode can only be set via the object **modes_of_operation**. As a change of the operating mode might require some time, one will have to 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.

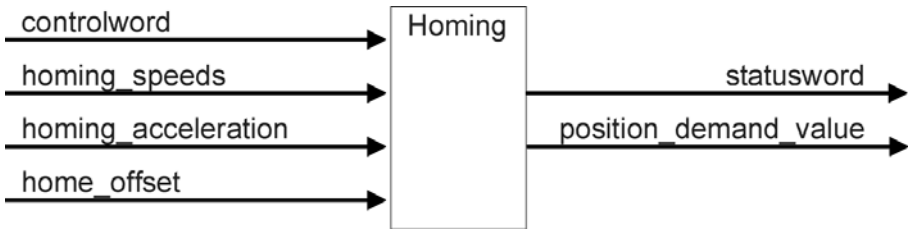


Figure 8.1: Homing Mode

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) .

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.

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

8.2.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	UINT16	7 Device control
6041_h	VAR	statusword	UINT16	7 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.

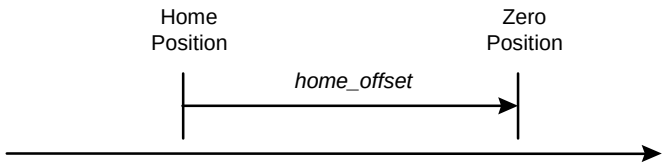


Figure 8.2: Home Offset

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, 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, 17, 18, 32, 33, 34
Default Value	17

Value	Direction	Target	Reference point for Home position
-18	Positive	Endstop	Endstop
-17	Negative	Endstop	Endstop
-2	Positive	Endstop	Zero impulse
-1	Negative	Endstop	Zero impulse
1	Negative	Limit switch	Zero impulse
2	Positive	Limit switch	Zero impulse
17	Negative	Limit switch	Limit switch
18	Positive	Limit switch	Limit switch
33	Negative	Zero impulse	Zero impulse
34	Positive	Zero impulse	Zero impulse
35		No run	Actual position

The homing sequence of the respective methods is explained more detailed in the following.

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}

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	250 min^{-1} / s

For the homing mode the servo has 4 variables which differs in two for the index searching and two for the crawl mode.

In case of having all necessary values for computing the zero Position, no movement happens. This is the case, for instance, if the northmarker is selected for the homing.

8.2.3 Homing sequences

The various homing sequences are pictured in the following figures. Die encirceled number 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.

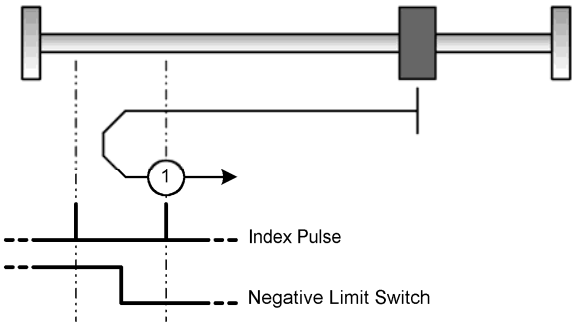


Figure 8.3: Homing operation to the negative limit switch including evaluation of the zero impulse

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.

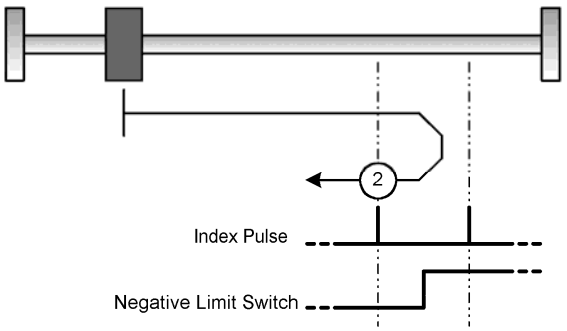


Figure 8.4: Homing operation to the positive limit switch including evaluation of the zero impulse

8.2.3.3 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.

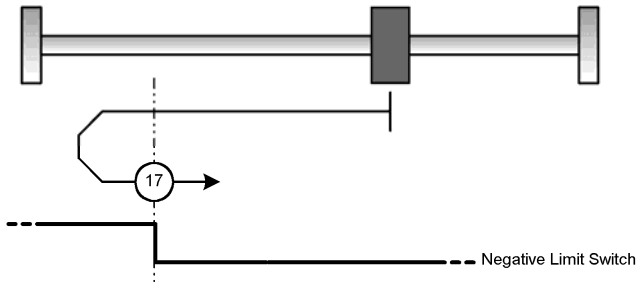


Figure 8.5: Homing operation to the negative limit switch

8.2.3.4 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.

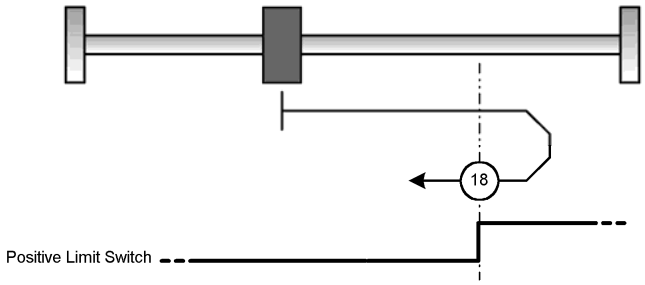


Figure 8.6: Homing operation to the positive limit switch

8.2.3.5 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 parametrized maximum current. The zero position refers to the first zero impulse of the angle encoder in positive direction from the stop.

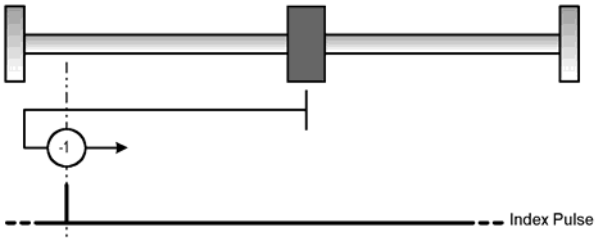


Figure 8.7: Homing operation to the negative stop evaluating the zero impulse

8.2.3.6 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 parametrized maximum current. The zero position refers to the first zero impulse of the angle encoder in negative direction from the stop.

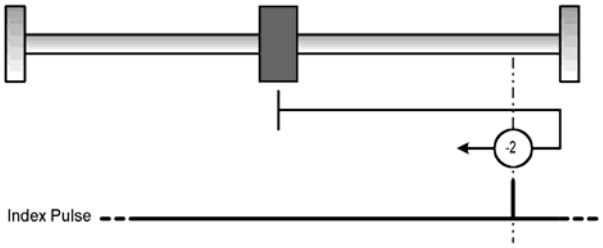


Figure 8.8: Homing operation to the positive stop evaluating the zero impulse

8.2.3.7 Methods 33 and 34: Homing operation to the zero impulse

For the methods 33 and 34 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.

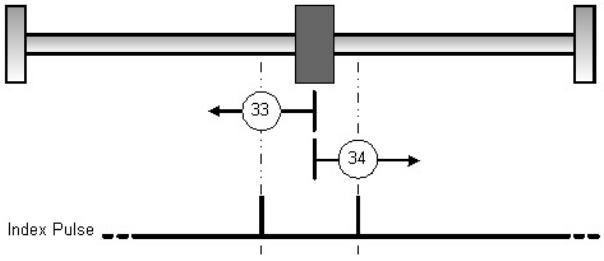


Figure 8.9: Homing operation only referring to the zero impulse

8.2.3.8 Method 35: Homing operation to the current position

On method 35 the zero position is referred to the current position.

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 **pre-defined_error_field**.

Bit 4	Description
0	Homing operation is not active
0 → 1	Start homing operation
1	Homing operation is active
1 → 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 8.10:

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 6.6). These two function blocks can be adjusted independently from each other.

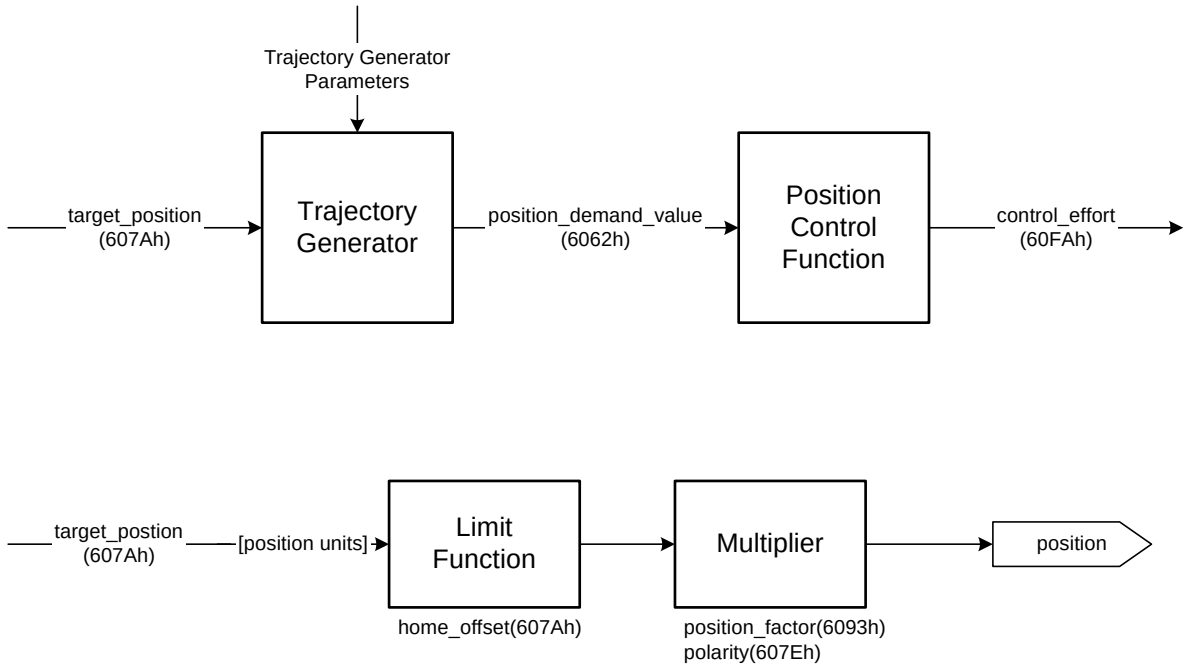


Figure 8.10: Trajectory generator and position controller

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 6.2: Conversion factors (Factor Group)). The internal quantities are marked by an asterisk and are not imperatively needed by the user.

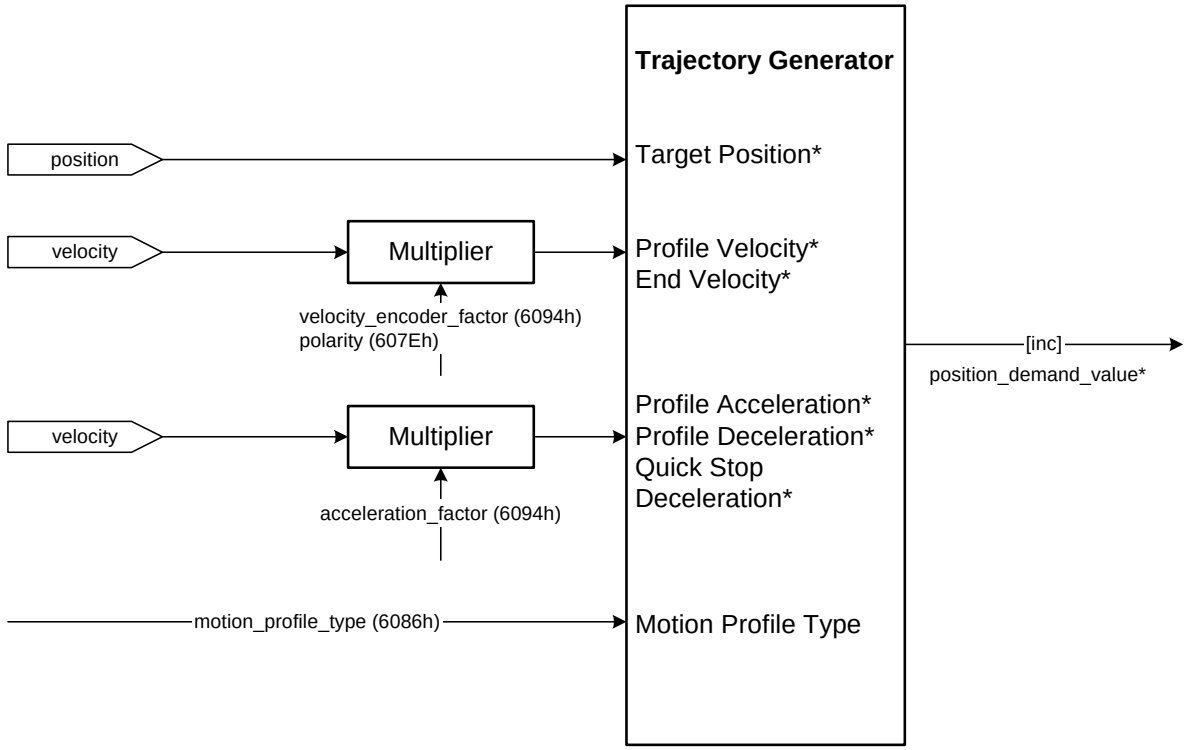


Figure 8.11: The trajectory generator

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
6040h	VAR	controlword	INT16	6.10 Device control
6041h	VAR	statusword	UINT16	6.10 Device control
605Ah	VAR	quick_stop_option_code	INT16	6.10 Device control
607Eh	VAR	polarity	UINT8	6.2 Conversion factors (Factor Group)
6093h	ARRAY	position_factor	UINT32	6.2 Conversion factors (Factor Group)
6094h	ARRAY	velocity_encoder_factor	UINT32	6.2 Conversion factors (Factor Group)
6097h	ARRAY	acceleration_factor	UINT32	6.2 Conversion factors (Factor Group)

8.3.2.3 Object $607A_h$: target_position

Das Object **target_position** (Zielposition) bestimmt, an welche Position der Antriebs-
The 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	0

8.3.2.5 Object 6082_h: end_velocity

The object **end_velocity** defines the speed at the target position (**target_position**). This object has to be set to zero so that the controller stops when it reaches the target position. A gap-less positioning with a speed high then 0 is not supported. 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	0
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 6.2 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 6.2 Conversion factors (Factor Group)).

Index	**6084_h**
Name	**profile_deceleration**
Object Code	VAR
Data Type	UINT32

Access	rw
PDO Mapping	yes
Units	speed 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	250000 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. A linear mode an a jerk-less mode is supported.

Index	**6086_h**
Name	**motion_profile_type**
Object Code	VAR
Data Type	INT16

Access	rw
PDO Mapping	yes
Units	--
Value Range	0, 1
Default Value	0

Value	Profile
0	Linear ramp
3	Jerk-less acceleration

8.3.3 Functional Description

Two modes can be choosen for doing a positioning.

1.) 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**.
The next position set can be send while the previous positioning is processed. After ending it, the new set is started direct after finishing the previous positioning

2.) Interupt of a running positioning

The ongoing positioning is interrupted and the new one is executed directly.

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.

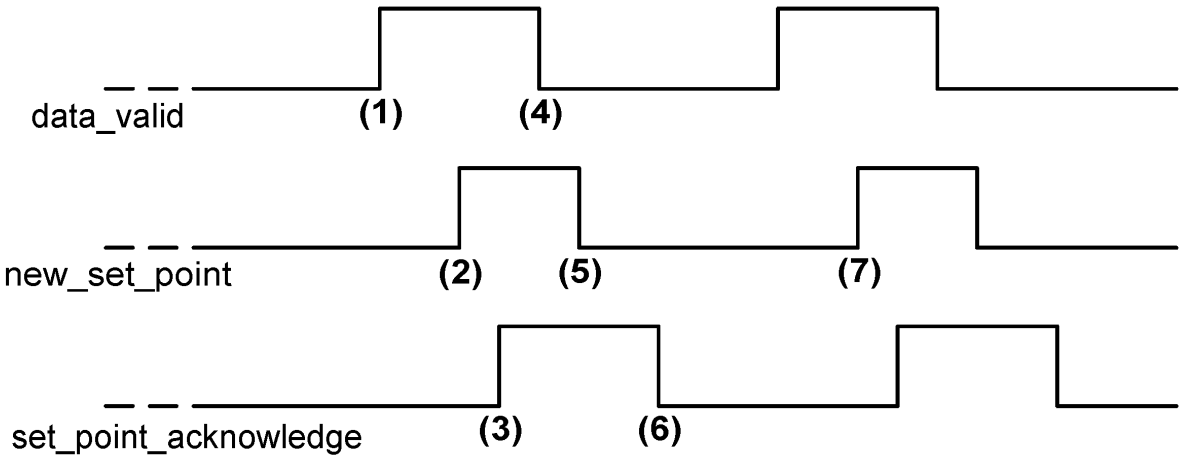


Figure 8.12: Positioning job transfer from a host

Figure 8.12 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 8.13 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**.

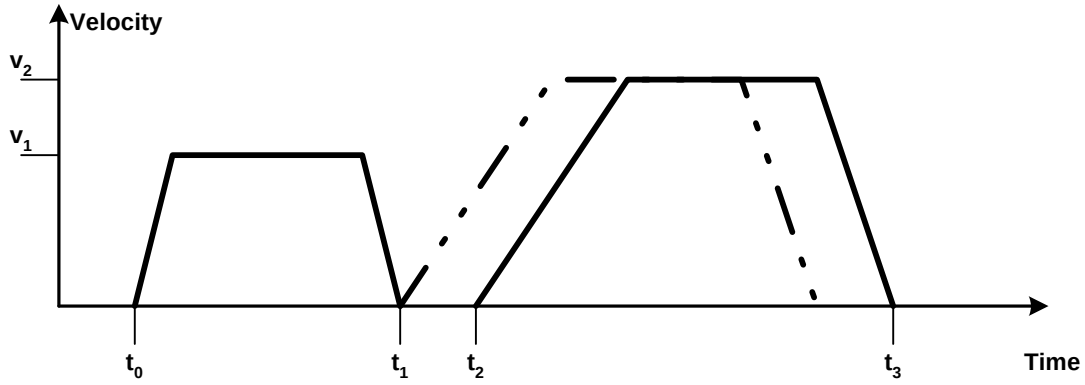


Figure 8.13: Simple positioning job

Figure 8.14 a new positioning motion has already been started while the previous motion was still running ($t_1 = t_2$).
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.
The host already transfers the subsequent target to the servo controller if it signals by a cleared **setpoint_acknowledge** bit that it has read the buffer and started the corresponding positioning motion.

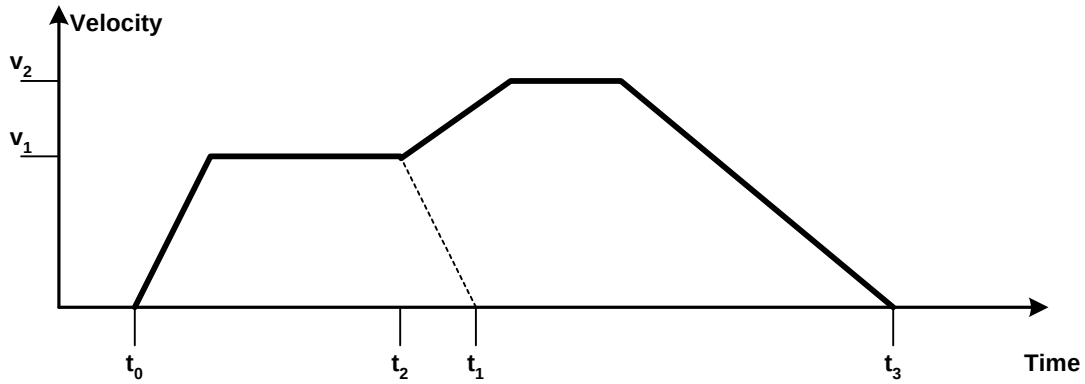


Figure 8.14: Gapless sequence of positioning jobs

If an ongoing positioning is interrupted by an positioning set marked as relative, it is not possible to say where the new target is. This is because the time for the interrupt is not known.

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:

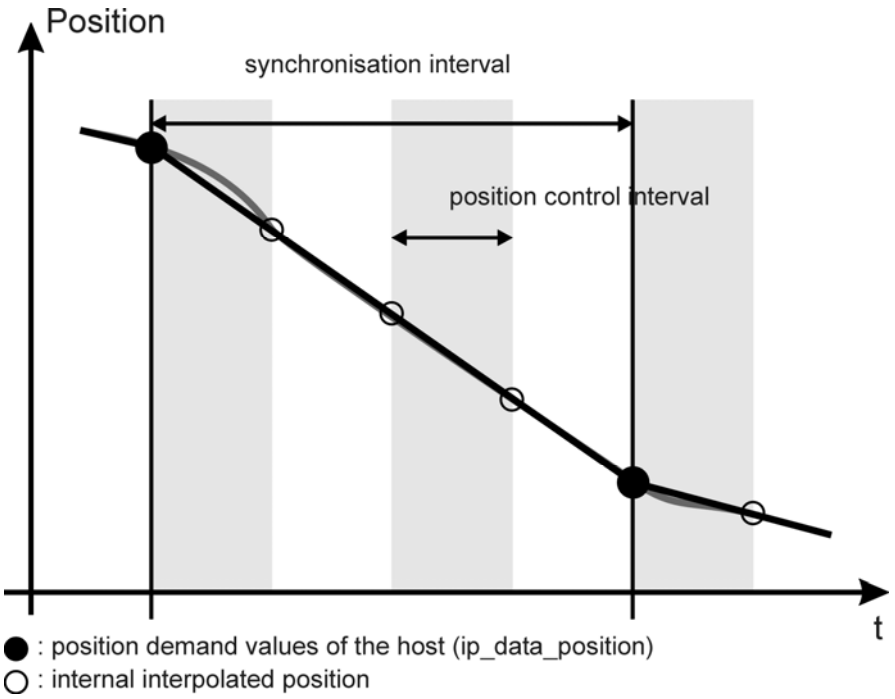


Figure 8.15: Linear interpolation between two positions

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		rw
$60C4_h$	REC	interpolation_data_configuration		rw

8.4.2.2 Affected objects of other chapters

Index	Object	Name	Type	Chapter
6040h	VAR	controlword	INT16	7 Device control
6041h	VAR	statusword	UINT16	7 Device control
6093h	ARRAY	position_factor	UINT32	6.2 Conversion factors (Factor Group)
6094h	ARRAY	velocity_encoder_factor	UINT32	6.2 Conversion factors (Factor Group)
6097h	ARRAY	acceleration_factor	UINT32	6.2 Conversion factors (Factor Group)

8.4.2.3 Object $60C0_h$: interpolation_submode_select

The obejct **interpolation_submode_select** determines the type of interpolation. 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_control-word**), 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	--

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 oder 1/10 ms) can be set by the object **ip_time_index**. After that the interval can be written to **ip_time_units**. 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: 8...40 ip_time_index = -4: 80...400
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	-4

Value	ip_time_units will be written in
-3	10^{-3} seconds (ms)
-4	10^{-4} seconds (0.1 ms)

8.4.2.6 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**). 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 8 ms	60C2h_01h, interpolation_time_units = 80
Enable buffer 1	60C4h_06h, buffer_clear = 1
Create SYNCs	SYNC (every 8 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$_h$)**. On success the **interpolated position mode** will be displayed in the object **modes_of_operation_display (6061$_h$)**. At this time internal selectors are changed to position control operation. The setpoint from the CAN bus are transferred to the position controller and extrapolated if necessary to meet the time interval.

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:

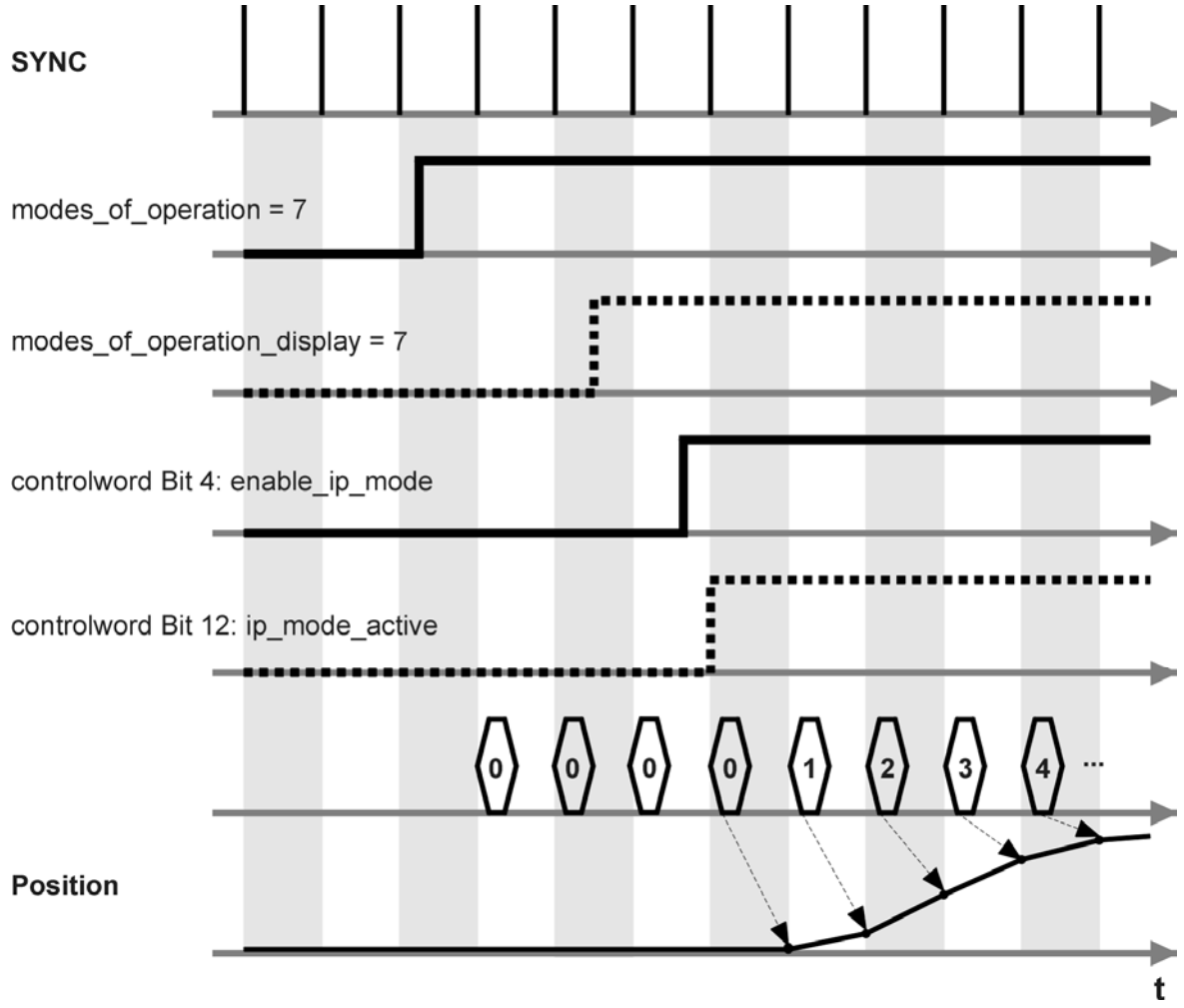


Figure 8.16: IP-Activation and data processing

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 restart of the IP-mode, because the state **OPERATION_ENABLE** has to be entered again, whereby the bit **ip_mode_active** will be cleared.

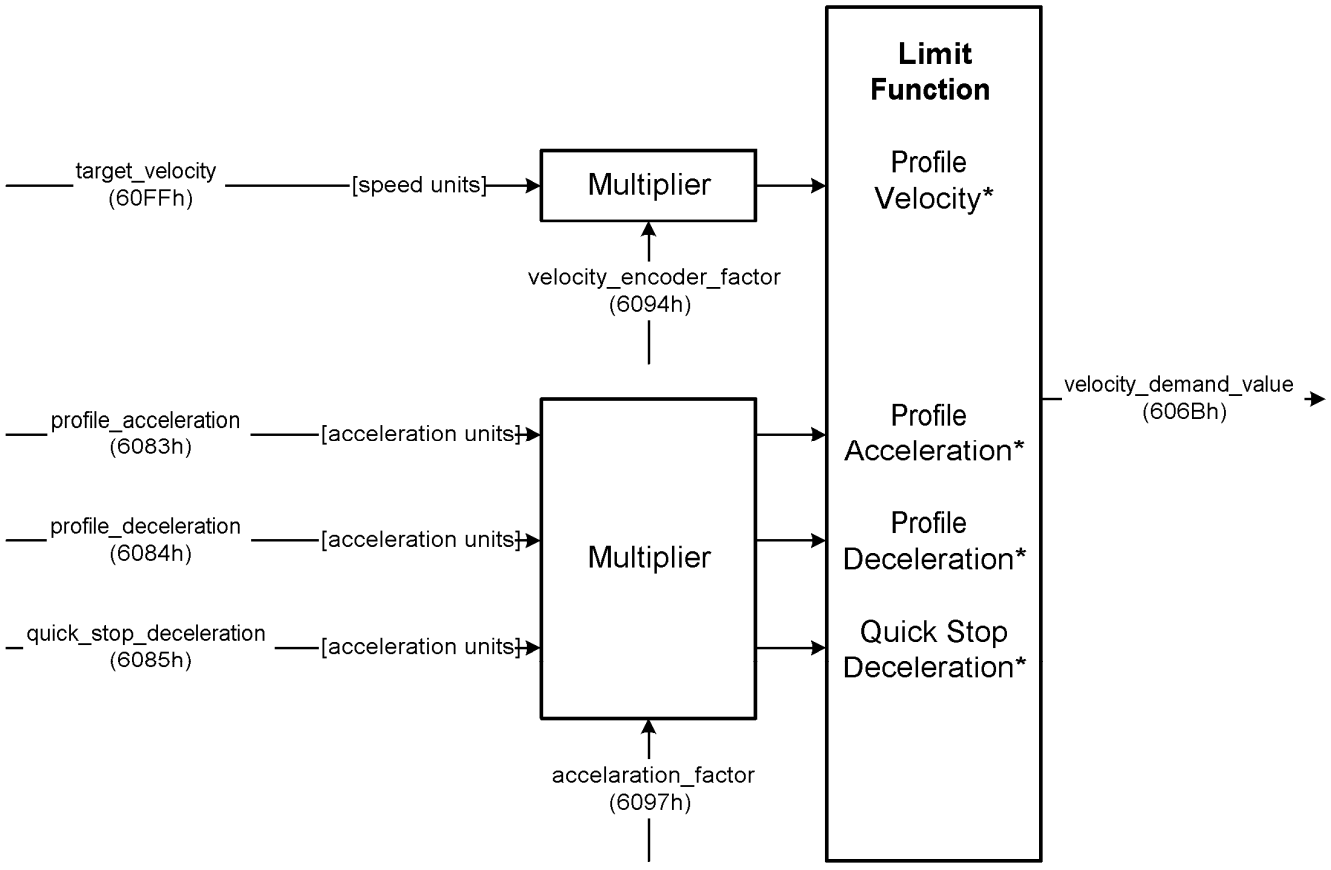
8.5 Profile Velocity Mode

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**



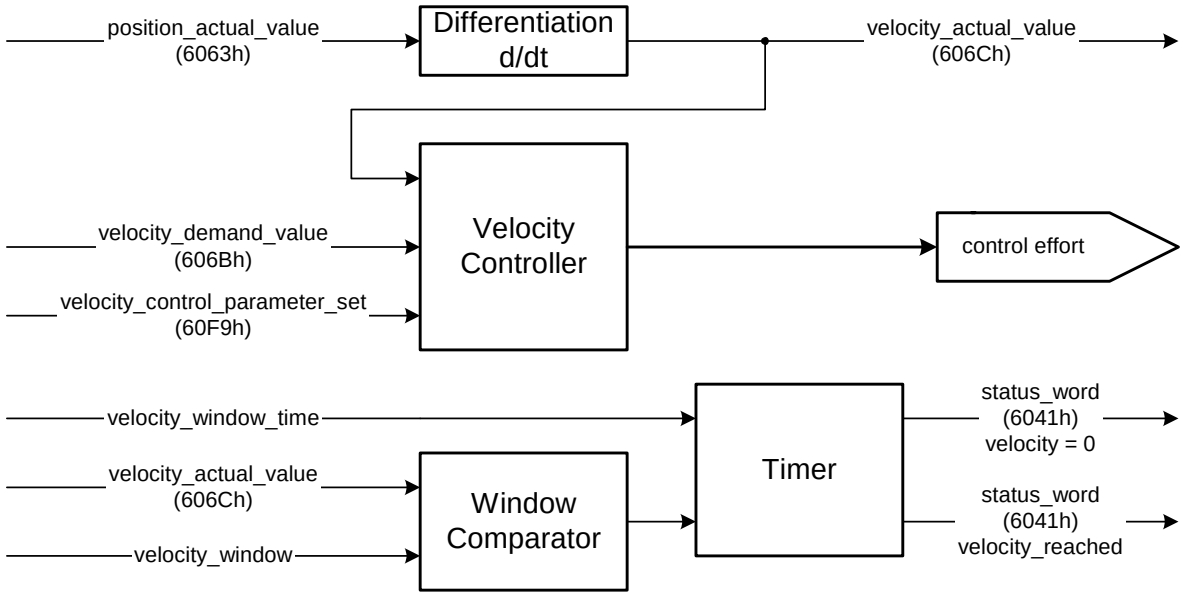


Figure 8.17: Structure of the Profile Velocity Mode

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
$606B_h$	VAR	velocity_demand_value	INT32	ro
$606C_h$	VAR	velocity_actual_value	INT32	ro
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	7. Device control
6041_h	VAR	statusword	UINT16	7. Device control
6063_h	VAR	position_actual_value*	INT32	6.6 Position Control Function
6069_h	VAR	velocity_sensor_actual_value	INT32	6.6 Position Control Function
6071_h	VAR	target_torque	INT16	8.6 Profile Torque Mode
6072_h	VAR	max_torque_value	UINT16	8.6 Profile Torque Mode
$607E_h$	VAR	polarity	UINT8	6.2 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	6.2 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 DIS-2, 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	Increments / sec
Value Range	--
Default Value	--

8.5.2.4 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	--

8.5.2.5 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	--

8.5.2.6 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	3000 min^{-1}

8.5.2.7 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	--

8.5.3 Object: Speed-Ramps

All inputs of the speed setpoint selector (default on AIN0) are feed to a ramp generator for smoothing sudden speed changes.

The ramps can be adjusteted by the following objects depending of rising, falling ramp and positive negative polarity of the speed setpoint.

If the mode profile_velocity_mode is selected all 4 ramps become active. It is not possible to deactivate the ramps by the CAN bus.

The relation between the velocity ramps and the profile acceleration is shown in the following schematic.

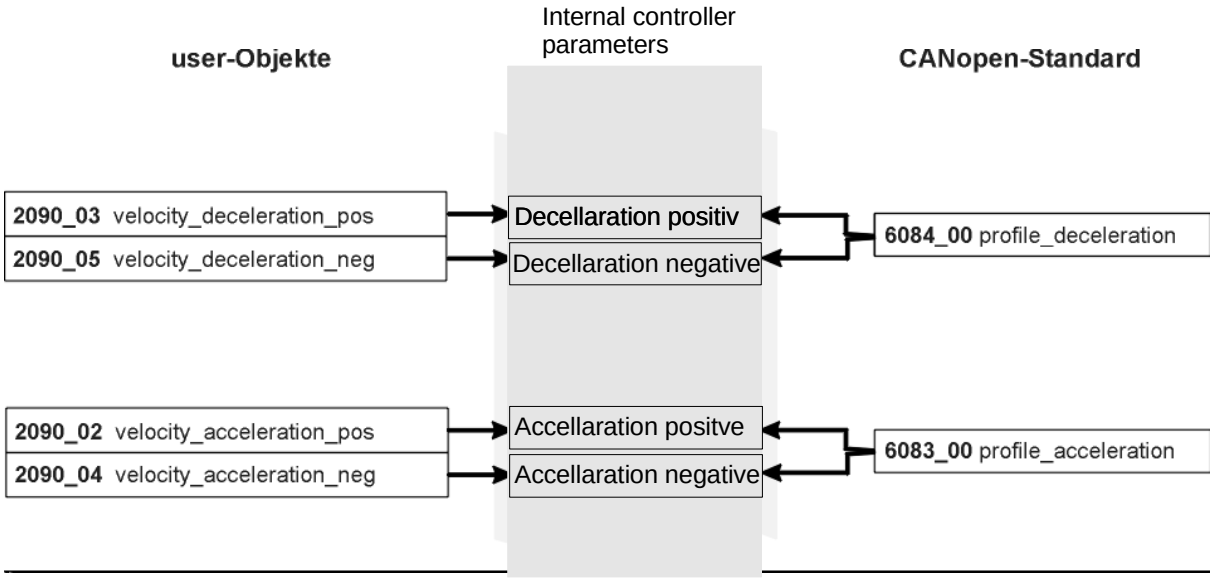


Figure 8.18: Relation of the ramps

8.5.3.1 Object 2090_h: velocity_ramps

The meaning of the objects can be seen in the following drawing:

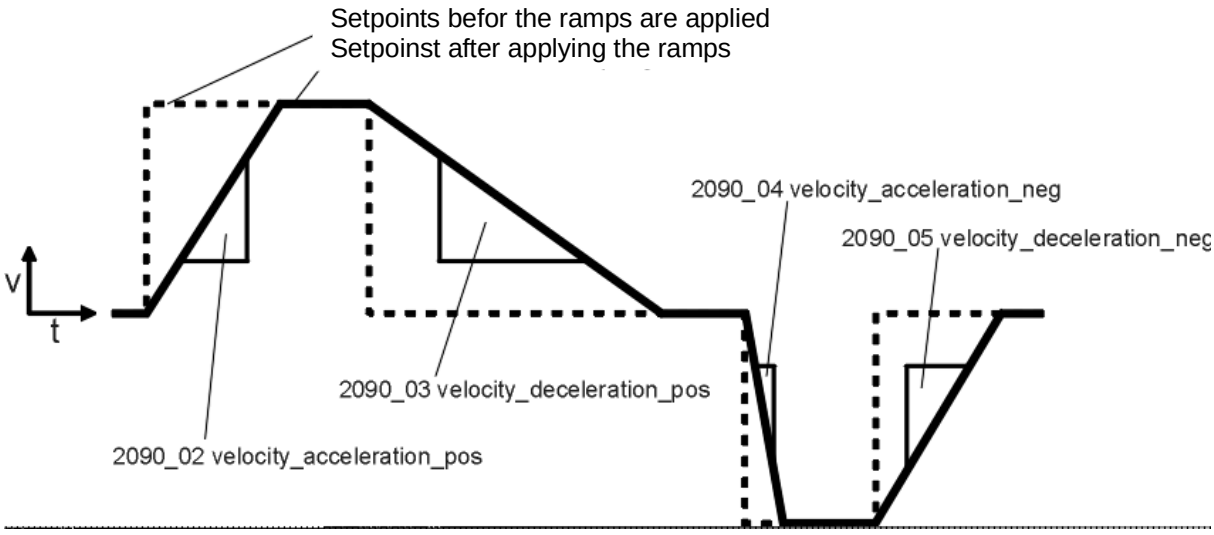


Figure 8.19: Bedeutung der Velocity_ramps

Index	**2090_h**
Name	**velocity_ramps**
Object Code	RECORD
No. of Elements	5

Sub-Index	**02_h**
Description	**velocity_acceleration_pos**
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	$0...2^{31}-1$
Default Value	$01E84800_h$

Sub-Index	**03_h**
Description	**velocity_deceleration_pos**
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	$0...2^{31}-1$
Default Value	$01E84800_h$

Sub-Index	**04_h**
Description	**velocity_acceleration_neg**
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	$0...2^{31}-1$
Default Value	$01E84800_h$

Sub-Index	**05_h**
Description	**velocity_deceleration_neg**
Data Type	UINT32
Access	rw
PDO Mapping	no
Units	--
Value Range	$0...2^{31}-1$
Default Value	$01E84800_h$

8.6 Profile Torque Mode

8.6.1 Survey

This chapter describes the torque controlled operation. This operating mode offers the chance to demand an external torque value (**target_torque**). 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.

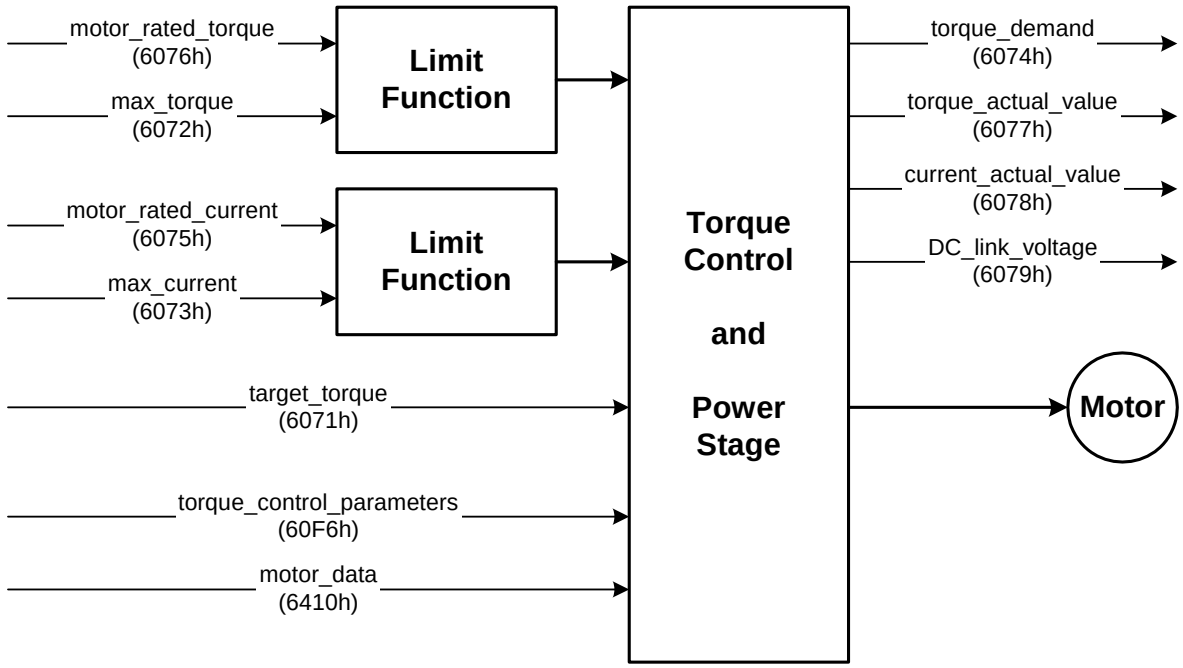


Figure 8.20: Structure of the Profile Torque Mode

The ramping is not supported. If in the **controlword** bit 8 (**halt)** is set, the current setpoint is set to zero. Additional the setpoint **target_torque** is set to **max_torque** if bit 8 is cleared.

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.

8.6.2 Description of Objects

8.6.2.1 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
$60F6_h$	RECORD	torque_control_parameters		rw

8.6.2.2 Affected objects from other chapters

Index	Object	Name	Type	Chapter
6040_h	VAR	controlword	INT16	7 Device control
6010_h	RECORD	motor_data	UINT32	6.4 Current control and motor adaptation
6075_h	VAR	motor_rated_current	UINT32	6.4 Current control and motor adaptation
6073_h	VAR	max_current	UINT16	6.4 Current control and motor adaptation

8.6.2.3 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

8.6.2.4 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 parametrized 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	1968

8.6.2.5 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^2t 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	--

8.6.2.6 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	2870

8.6.2.7 Object 6077_h: torque_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	--

8.6.2.8 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	--

8.6.2.9 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	--

9 Keyword index

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

V

Z