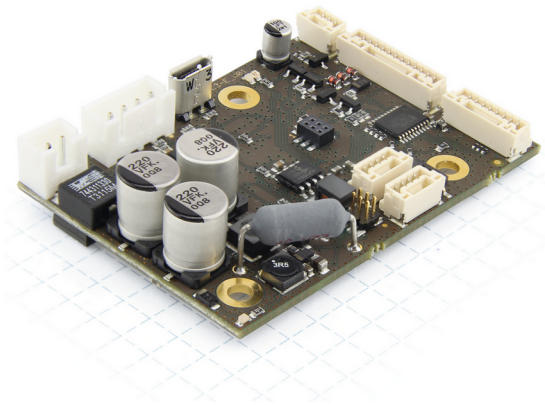
Technical Manual CL3-E

Fieldbus: CANopen, USB, Modbus RTU

For use with the following devices:

CL3-E-1-0F, CL3-E-2-0F



Valid with firmware version FIR-v1650
and since hardware version W004c

Technical Manual Version: 2.0.2

Contents

1 Introduction

The *CL3-E* is a controller for the *open loop* or *closed loop* operation of stepper motors and the *closed loop* operation of BLDC motors.

This manual describes the functions of the controller and the available operating modes. It also shows how you can address and program the controller via the communication interface.

You can find further information on the device on the Nanotec website **us.nanotec.com**.

1.1 Version information

Manual version	Date	Changes	Firmware version
1.0.0	10.09.2014	Edition	FIR-v1434
1.0.15	18.11.2014	• Error corrections • The "Mode of modulo operation" object in 2070_h was replaced with the "Positioning option code" object in **$60F2_h$**	FIR-v1446
1.1.0	11.03.2015	New chapter: • **Clock-direction mode**	FIR-v1504
1.1.1	24.04.2015	• Error corrections • New chapter **Input Routing**	FIR-v1512
1.2.0	02.10.2015	• Error corrections • New chapter **Overtemperature protection** • New chapter **Output Routing** • New section **Possible combinations of travel commands** • Addition to the connection data for the connectors • Addition to the switching thresholds for digital inputs	FIR-v1540
1.3.0	08.04.2016	• Error corrections • New chapter **Interpolated Position Mode**	FIR-v1614
1.3.1	22.07.2016	Additions and error corrections	FIR-v1626
2.0.0	01/2018	• New chapter **Environmental conditions** • New chapter **Control modes** • New chapter **Limitation of the range of motion** • New chapter **Cycle times** • New chapter **CANopen services** • Revision of chapter **Commissioning** • Additions and error corrections	FIR-v1650
2.0.1	06/2018	Additions and error corrections	FIR-v1650
2.0.2	09/2018	Pin assignments of **X5** corrected	FIR-v1650

1.2 Copyright, marking and contact

Nanotec® Electronic GmbH & Co. KG

Kapellenstraße 6

D-85622 Feldkirchen/Munich

Phone: +49 (0)89-900 686-0

Fax: +49 (0)89-900 686-50

Internet: **us.nanotec.com**

1.3 Intended use

The *CL3-E controller* is used to control stepper and BLDC motors and is designed for use under the approved **Environmental conditions**.

Any other use is considered unintended use.

Note

Changes or modification to the controller are not permitted.

1.4 Warranty and disclaimer

Nanotec produces component parts that are used in a wide range of industrial applications. The selection and use of Nanotec products is the responsibility of the system engineer and end user. Nanotec accepts no responsibility for the integration of the products in the end system.

Under no circumstances may a Nanotec product be integrated as a safety controller in a product or construction. All products containing a component part manufactured by Nanotec must, upon delivery to the end user, be provided with corresponding warning notices and instructions for safe use and safe operation. All warning notices provided by Nanotec must be passed on directly to the end user.

Our general terms and conditions apply: **en.nanotec.com/service/general-terms-and-conditions/**.

1.5 Specialist staff

Only specialists may install, program and commission the device:

- Persons who have appropriate training and experience in work with motors and their control.
- Persons who are familiar with and understand the content of this technical manual.
- Persons who know the applicable regulations.

1.6 EU directives for product safety

The following EU directives were observed:

- RoHS directive (2011/65/EU, 2015/863/EU)

1.7 Other applicable regulations

In addition to this technical manual, the following regulations are to be observed:

- Accident-prevention regulations
- Local regulations on occupational safety

1.8 Used icons

All notices are in the same format. The degree of the hazard is divided into the following classes.

CAUTION

- The CAUTION notice indicates a possibly dangerous situation.
- Failure to observe the notice **may** result in moderately severe injuries.
- Describes how you can avoid the danger.

Note

- Indicates an error source or likelihood of confusion.
- Failure to observe the notice **may** result in damage to this or other devices.
- Describes how device damage can be avoided.

Tip

Shows a tip for the application or task.

1.9 Emphasis in the text

The following conventions are used in the document:

Text set in **bold** indicates cross references and hyperlinks:

- The following bits in object **6041**$_h$ (statusword) have a special function:
- A list of available system calls can be found in chapter **System calls in a NanoJ program**.

Text set in *italics* marks named objects:

- Read the *installation manual*.
- Use the *Plug & Drive Studio* software to perform the auto setup.
- For software: You can find the corresponding information in the *Operation* tab.
- For hardware: Use the *ON/OFF* switch to switch the device on.

A text set in `Courier` marks a code section or programming command:

- The line with the `od_write(0x6040, 0x00, 5 );` command has no effect.
- The NMT message is structured as follows: `000 | 81 2A`

A text in "quotation marks" marks user input:

- Start the NanoJ program by writing object 2300$_h$, bit 0 = "1".
- If a holding torque is already needed in this state, the value "1" must be written in 3212$_h$:01$_h$.

1.10 Numerical values

Numerical values are generally specified in decimal notation. The use of hexadecimal notation is indicated by a subscript *h* at the end of the number.

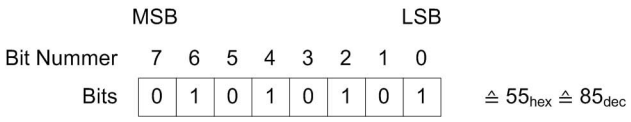
The objects in the object dictionary are written with index and subindex as follows: `<Index>:<Subindex>`

Both the index as well as the subindex are specified in hexadecimal notation. If no subindex is listed, the subindex is 00_h.

Example: Subindex 5 of object 1003_h is addressed with $1003_h:05_h$, subindex 00 of object 6040_h with 6040_h.

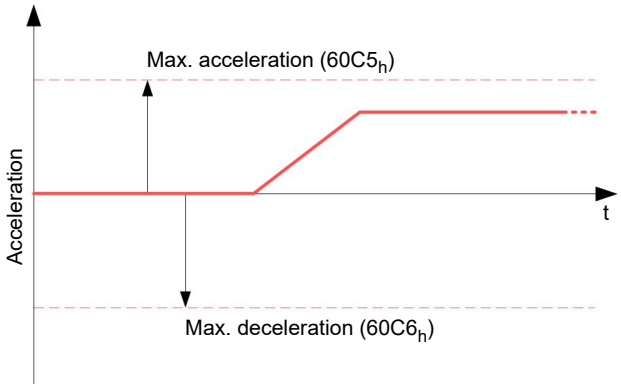
1.11 Bits

The numbering of individual bits in an object always begins with the LSB (bit number 0). See the following figure, which uses data type *UNSIGNED8* as an example.



1.12 Counting direction (arrows)

In figures, the counting direction is always in the direction of an arrow. Objects $60C5_h$ and $60C6_h$ depicted as examples in the following figure are both specified as positive.



2 Safety and warning notices

Note

- Damage to the controller.
- Changing the wiring during operation may damage the controller.
- Only change the wiring in a de-energized state. After switching off, wait until the capacitors have discharged.

Note

- Fault of the controller due to excitation voltage of the motor.
- Voltage peaks during operation may damage the controller.
- Install suitable circuits (e.g., charging capacitor) that reduce voltage peaks.

Note

- There is no polarity reversal protection.
- Polarity reversal results in a short-circuit between supply voltage and GND (earth) via the power diode.
- Install a line protection device (fuse) in the supply line.

Note

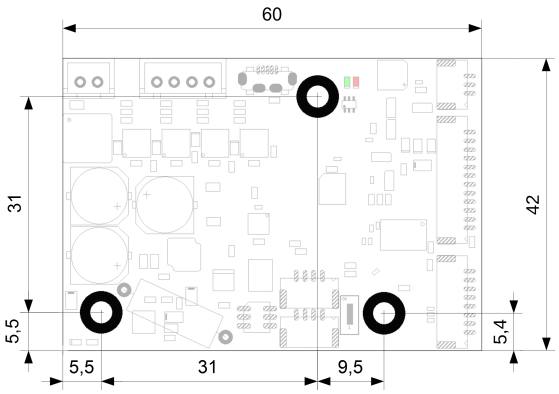
- The device contains components that are sensitive to electrostatic discharge.
- Improper handling can damage the device.
- Observe the basic principles of ESD protection when handling the device.

3 Technical details and pin assignment

3.1 Environmental conditions

Environmental condition	Value
Protection class	No IP protection
Ambient temperature (operation)	-10 … +40°C
Air humidity (non-condensing)	0 … 95 %
Altitude of site above *sea level* (without drop in performance)	1500 m
Ambient temperature (storage)	-25 … +85°C

3.2 Dimensioned drawing



3.3 Electrical properties and technical data

Property	Description / value
Operating voltage	12 V DC to 24 V DC +/-5%
Rated current	3 A_{rms}
Peak current	CL3-E-1-0F (*low current*): 3 A_{rms} CL3-E-2-0F (*low current*): 6 A_{rms}
Commutation	Stepper motor – open loop, stepper motor – closed loop with encoder, BLDC motor – closed loop with Hall sensor, and BLDC motor – closed loop with encoder
Operating modes	*Profile Position Mode*, *Profile Velocity Mode*, *Profile Torque Mode*, *Velocity Mode*, *Homing Mode*, *Interpolated Position Mode*, *Cyclic Sync Position Mode*, *Cyclic Sync Velocity Mode*, *Cyclic Synchronous Torque Mode*, *Clock-Direction Mode*

Property	Description / value
Set value setting / programming	*Clock-direction*, *analog*, *NanoJ program*
Interfaces	CANopen, USB, RS-485 (Modbus RTU), RS-232 (Modbus RTU)
Inputs	• 5 digital inputs 5 V • 1 analog input, 10-bit resolution, 0-10 V or 0-20 mA (switchable by means of software, default setting is 0-10 V) • 1 analog input, 10-bit resolution, 0-10 V
Outputs	3 outputs, (open drain, 0 switching, max. 24 V and 100 mA)
Protection circuit	Overvoltage and undervoltage protection Overtemperature protection (> 75° Celsius on the power board) Polarity reversal protection: In the event of a polarity reversal, a short-circuit will occur between supply voltage and GND over a power diode; a line protection device (fuse) is therefore necessary in the supply line. The values of the fuse are dependent on the application and must be dimensioned • greater than the maximum current consumption of the controller • less than the maximum current of the voltage supply. If the fuse value is very close to the maximum current consumption of the controller, a medium / slow tripping characteristics should be used.

3.4 Overtemperature protection

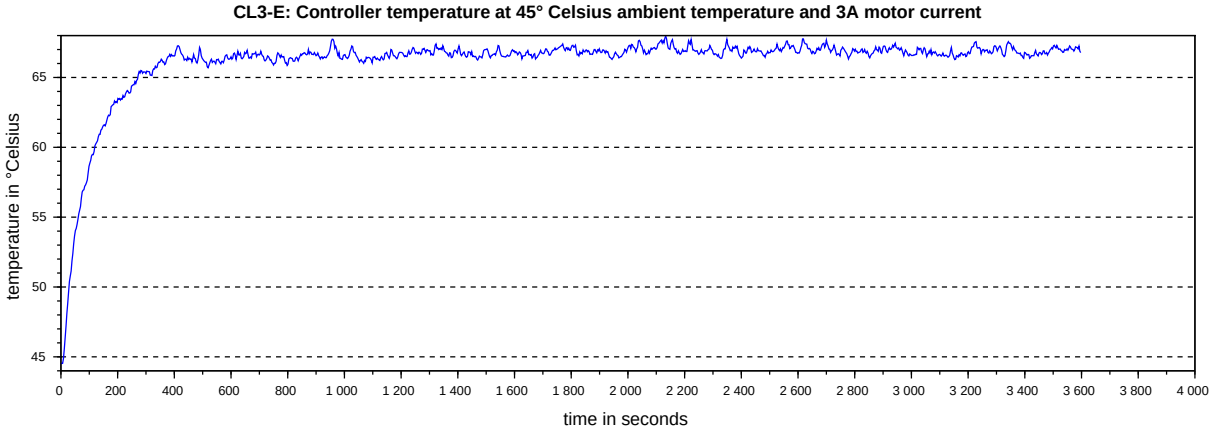
Above a temperature of approx. 75°C on the power board the power part of the controller switches off and the error bit is set (see objects **1001**$_h$ and **1003**$_h$). After cooling down and confirming the error (see **table for the controlword**, "Fault reset"), the controller again functions normally.

The following temperature test results provide information on the temperature behavior of this controller.

Temperature tests are performed under the following conditions:

- Operating voltage: 24 V DC
- Motor current: 3 A rms
- Operation mode: Velocity Mode, full step, 30 rpm
- Ambient temperature: 45 °C
- Altitude of site: 500 m above sea level
- No external cooling in the climatic chamber, e.g., via fan

The following graphic shows the results of the temperature tests:



Summary:

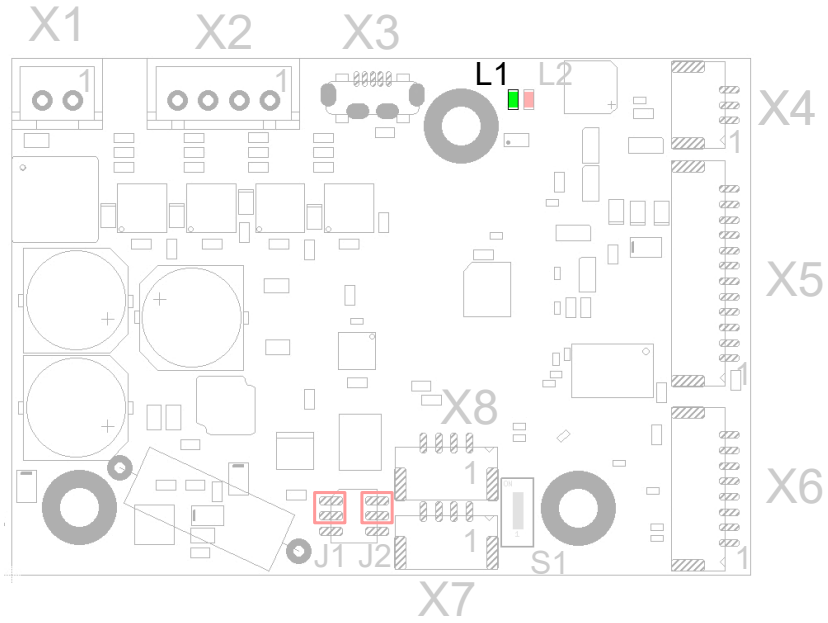
At 45°C (+24 V, 3 A rms, Velocity Mode 30 rpm), the controller was in operation for longer than 1 hour without having been switched off. The temperature was stable at approx. 67°C.

Note

Aside from the motor, the exact temperature behavior is also largely dependent on the flange connection and the heat transfer there as well as on the convection in the machine. For this reason, we recommend always performing an endurance test in the actual environment for applications in which current level and ambient temperature pose a problem.

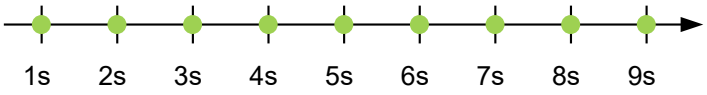
3.5 LED signaling

3.5.1 Power LED



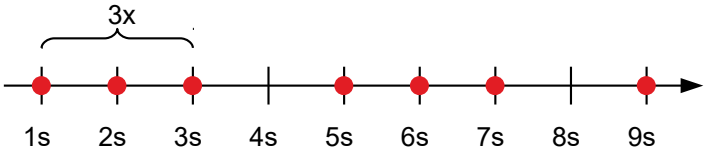
Normal operation

In normal operation, the green power LED L1 flashes briefly once per second.



Case of an error

If an error has occurred, the LED turns red and signals an error number. In the following figure, the error number 3 is signaled.



The following table shows the meaning of the error numbers.

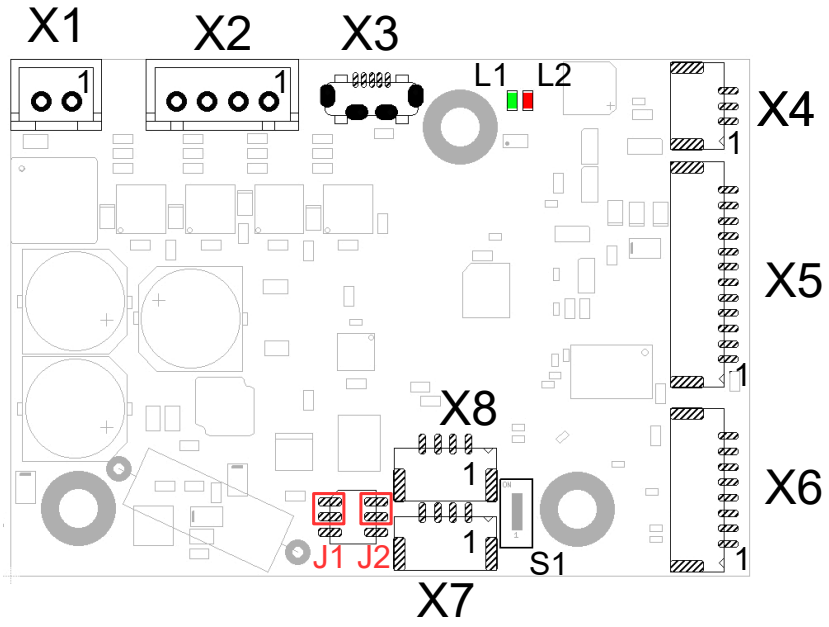
Flash rate	Error
1	General
2	Voltage
3	Temperature
4	Overcurrent
5	Controller
6	Watchdog-Reset

Note

For each error that occurs, a more precise error code is stored in object **1003**$_h$.

3.6 Pin assignment

3.6.1 Overview



Connector	Function
X1	Voltage supply
X2	Motor connection
X3	Micro USB
X4	RS-232 connection
X5	Digital/analog inputs and outputs
X6	Encoder/Hall sensor
X7	CANopen / RS-485 IN
X8	CANopen / RS-485 OUT
S1	Switch for 120 ohm termination resistor
J1	Jumper: switches between CAN_L or RS-485-
J2	Jumper: switches between CAN_H or RS-485+
L1	Status LED green
L2	Status LED red

3.6.2 Connector X1 - voltage supply

Voltage source

The operating or supply voltage supplies a battery, a transformer with rectification and filtering, or a switching power supply.

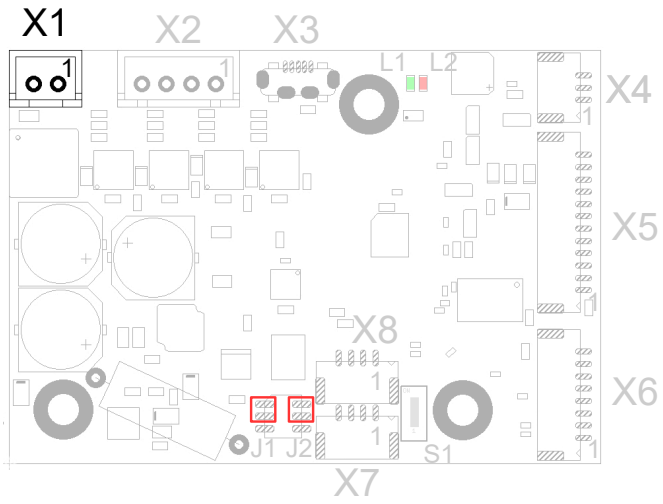
Note

- EMC: For a DC power supply line longer than 30 m or when using the motor on a DC bus, additional interference-suppression and protection measures are necessary.
- An EMI filter is to be inserted in the DC supply line as close as possible to the controller/motor.
- Long data or supply lines are to be routed through ferrites.

Connections

Connector type: JST XH

In the following figure, pin 1 is marked with a "1".



PIN	Function	Note
1	+UB	12 V - 24 V ±5%
2	GND	

Permissible operating voltage

The maximum operating voltage is 28.5 V DC. If the input voltage of the controller exceeds this threshold value, the motor is switched off and an error triggered. Above 27.5 V, the integrated ballast circuit (3 W power) is activated.

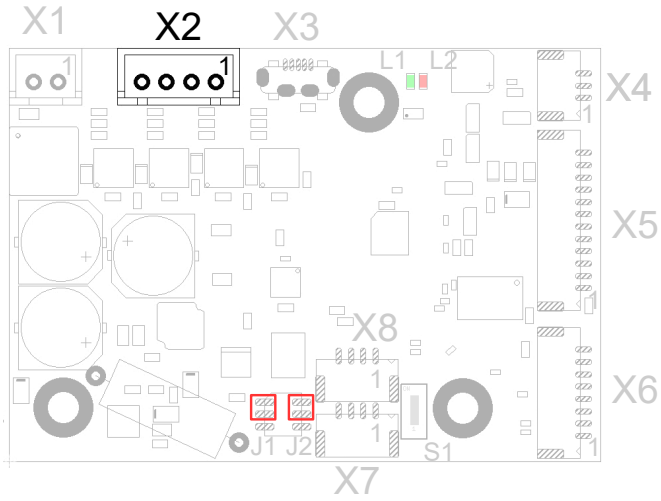
The minimum operating voltage is 10 V DC. If the input voltage of the controller falls below this threshold value, the motor is switched off and an error triggered.

A charging capacitor of at least 4700 µF / 50 V (approx. 1000 µF per ampere rated current) must be connected to the supply voltage to avoid exceeding the permissible operating voltage (e.g., during braking).

3.6.3 Connector X2 – motor connection

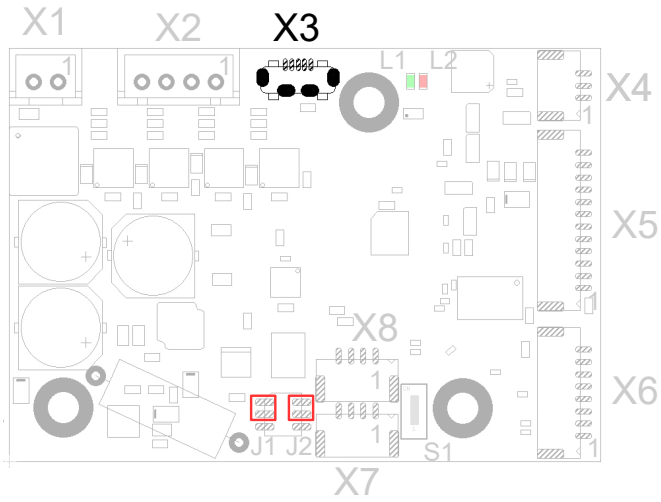
Connector type: JST XH

In the following figure, pin 1 is marked with a "1".



PIN	Function (stepper motor)	Function (BLDC)
1	A	U
2	A\	V
3	B	W
4	B\	Not used

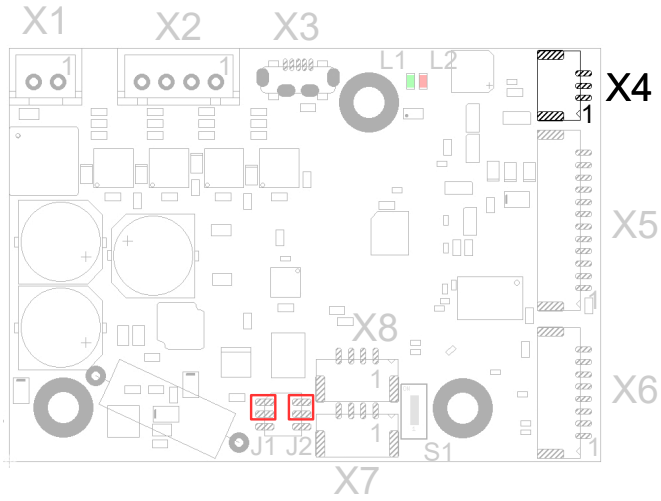
3.6.4 Connector X3 – micro USB



3.6.5 Connector X4 – RS-232 connection

Connector type: JST GH

In the following figure, pin 1 is marked with a "1".

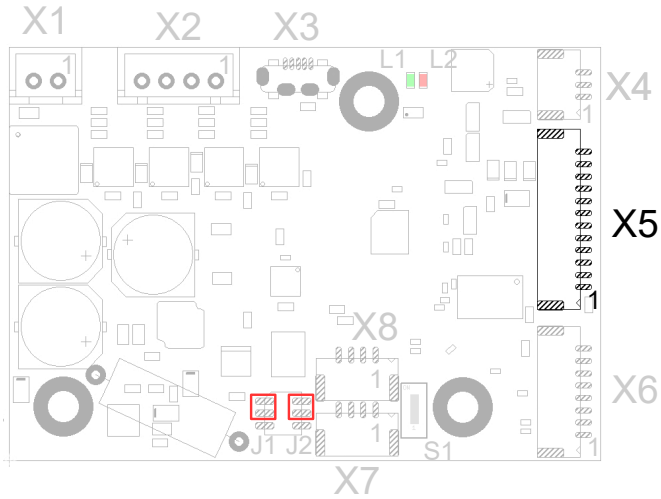


PIN	Function	Note
1	RS-232-RX	
2	RS-232-TX	
3	GND	

3.6.6 Connector X5 – digital/analog inputs and outputs

Connector type: JST GH

In the following figure, pin 1 is marked with a "1".



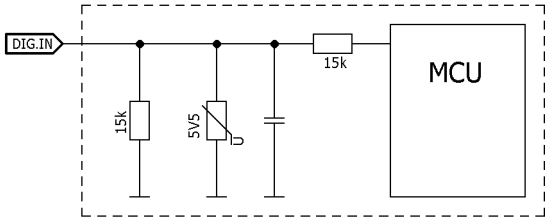
PIN	Function	Note
1	+10 V DC	Output voltage, max. 200 mA
2	Digital input 1	5 V signal, max. 1 MHz
3	Digital input 2	5 V signal, max. 1 MHz
4	Digital input 3	5 V signal, max. 1 MHz ("direction" in clock-direction mode)
5	Digital input 4	5 V signal, max. 1 MHz ("clock" in clock-direction mode)
6	Digital input 5	5 V signal, max. 1 MHz

PIN	Function	Note
7	Analog input 1	10 bit, 0-10 V or 0-20 mA, switchable by means of software with object **3221**$_h$, default setting: 0-10 V
8	Analog input 2	10 bit, 0-10 V, not switchable by means of software
9	Digital output 1	Open drain, max. 24 V/100 mA
10	Digital output 2	Open drain, max. 24 V/100 mA
11	Digital output 3	Open drain, max. 24 V/100 mA
12	GND	

The following switching thresholds apply for inputs 1 to 5:

Switching thresholds	
On	**Off**
> approx. 3 V	< approx. 1 V

The current consumption is approximately 0.4 mA. The following internal circuit diagram applies for the digital inputs:



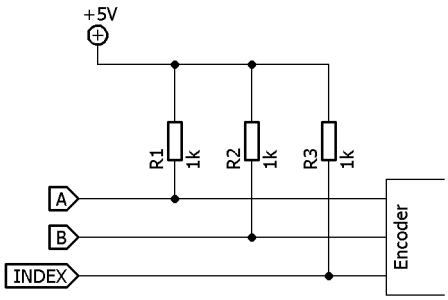
3.6.7 Connector X6 – encoder/Hall sensor

Note

The controller with hardware version W004b does **not** function with the following encoders without additional wiring (see below):

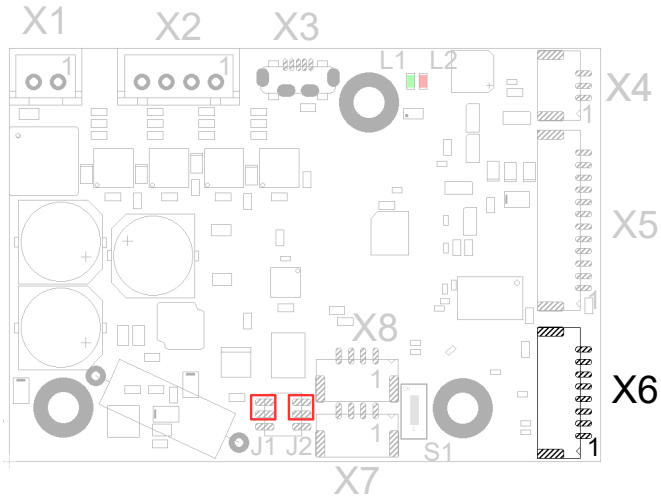
- WEDS5541
- WEDS5546
- HEDS5540

With these encoders, a PULL-UP resistor must be connected to 5 V on cables A, B and INDEX.



Connector type: JST GH

In the following figure, pin 1 is marked with a "1".

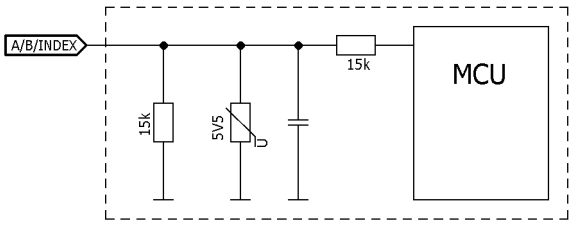


PIN	Function	Note
1	+5 V DC	Supply voltage for encoder/Hall sensor, max. 200 mA
2	A	5 V signal, max. 1 MHz
3	B	5 V signal, max. 1 MHz
4	Index	5 V signal
5	H1	5 V signal, max. 1 MHz
6	H2	5 V signal, max. 1 MHz
7	H3	5 V signal, max. 1 MHz
8	GND	

The following switching thresholds apply for the encoder inputs:

Switching thresholds	
Safe switch on	**Safe switch off**
> approx. 2.8 V	< approx. 1.1 V

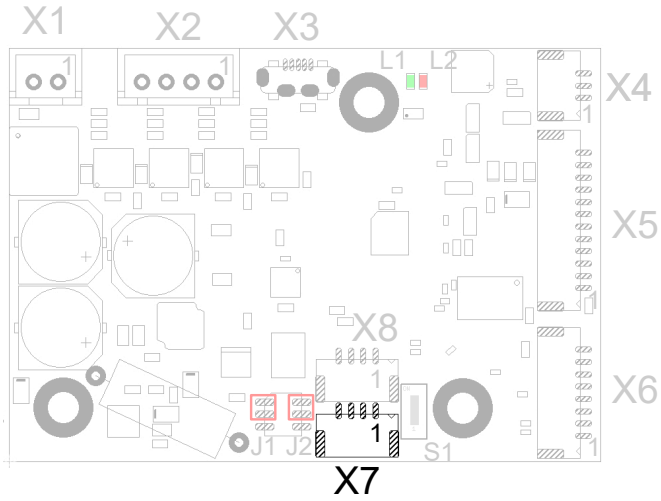
The internal wiring of the encoder inputs is shown in the following.



3.6.8 Connector X7 – CANopen/RS-485 IN

Connector type: JST GH

In the following figure, pin 1 is marked with a "1".



PIN	CANopen function	RS-485 function	Note
1	+UB Logic	+UB Logic	24 V DC input, external logic supply for communication, input voltage, current consumption approx. 36 mA
2	CAN+	RS-485+	The changeover is performed via jumper J2.
3	CAN-	RS-485-	The changeover is performed via jumper J1.
4	GND	GND	

Note

The windings of the motor are not supplied by the logic supply.

RS-485 line polarization

Note

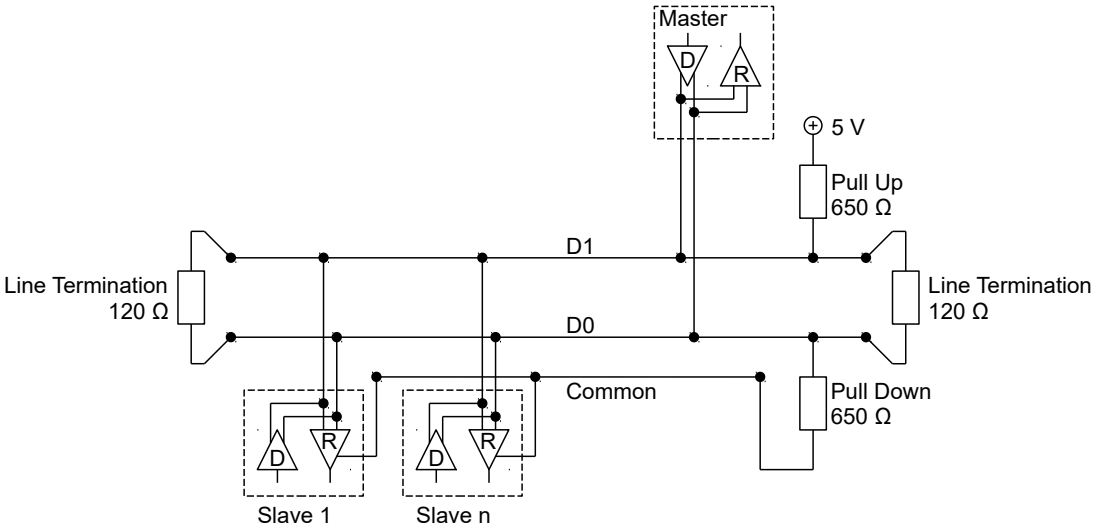
The controller is **not** equipped with line polarization and expects the master device to have one.

If the master device on the bus does not have line polarization of its own, a pair of resistors must be attached to the RS-485 balanced cables:

- A pull-up resistor to a 5V voltage on the RS-485+ (D1) cable
- A pull-down resistor to earth (GND) on the RS-485- (D0) cable

The value of these resistors must be between 450 ohm and 650 ohm. A 650 ohm resistor permits a higher number of devices on the bus.

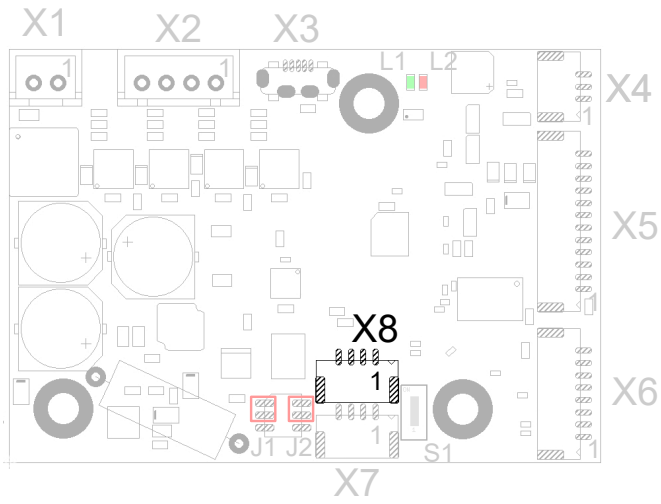
In this case, a line polarization must be attached at a location for the entire serial bus. In general, this location should be on the master device or its connection. All other devices then no longer need to implement line polarization.



3.6.9 Connector X8 – CANopen/RS-485 OUT

Connector type: JST GH

In the following figure, pin 1 is marked with a "1".



PIN	CANopen function	RS-485 function	Note
1	+UB Logic	+UB Logic	24 V DC input, external logic supply for communication, input voltage, current consumption approx. 36 mA
2	CAN+	RS-485+	The changeover is performed via jumper J2.
3	CAN-	RS-485-	The changeover is performed via jumper J1.
4	GND	GND	

Note

The windings of the motor are not supplied by the logic supply.

RS-485 line polarization

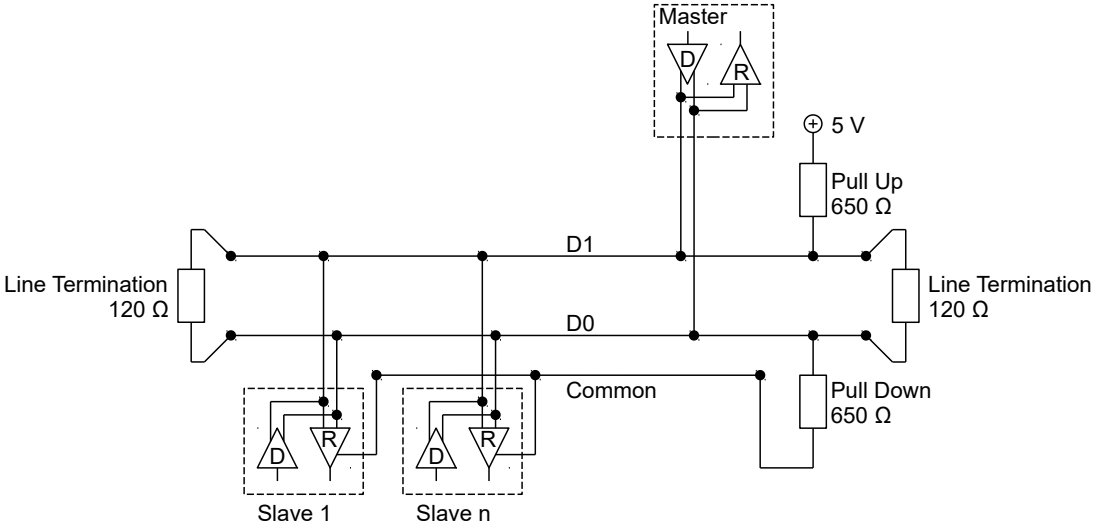
Note
The controller is **not** equipped with line polarization and expects the master device to have one.

If the master device on the bus does not have line polarization of its own, a pair of resistors must be attached to the RS-485 balanced cables:

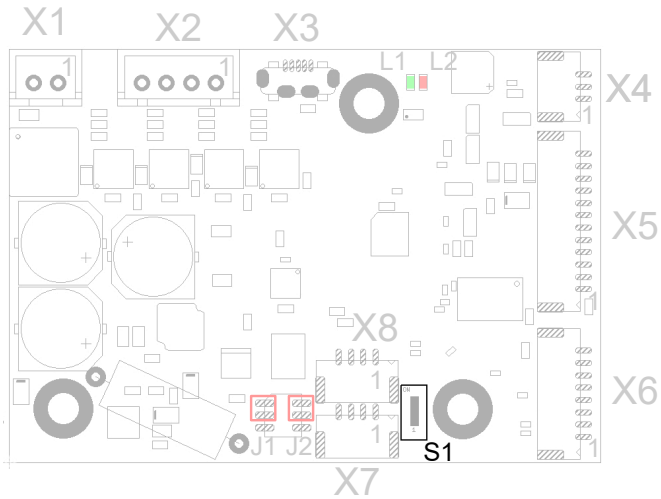
- A pull-up resistor to a 5V voltage on the RS-485+ (D1) cable
- A pull-down resistor to earth (GND) on the RS-485- (D0) cable

The value of these resistors must be between 450 ohm and 650 ohm. A 650 ohm resistor permits a higher number of devices on the bus.

In this case, a line polarization must be attached at a location for the entire serial bus. In general, this location should be on the master device or its connection. All other devices then no longer need to implement line polarization.



3.6.10 Switch S1 – Termination resistor



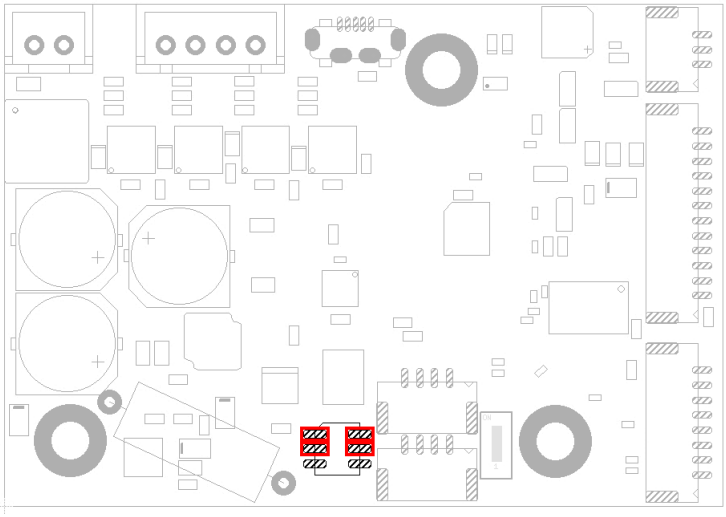
A termination with 120 ohm between CAN+ and CAN- or RS-485- and RS-485+ can thereby be switched on or off.

3.6.11 Jumper J1/J2

With these jumpers, it is possible to change between CANopen or RS-485.

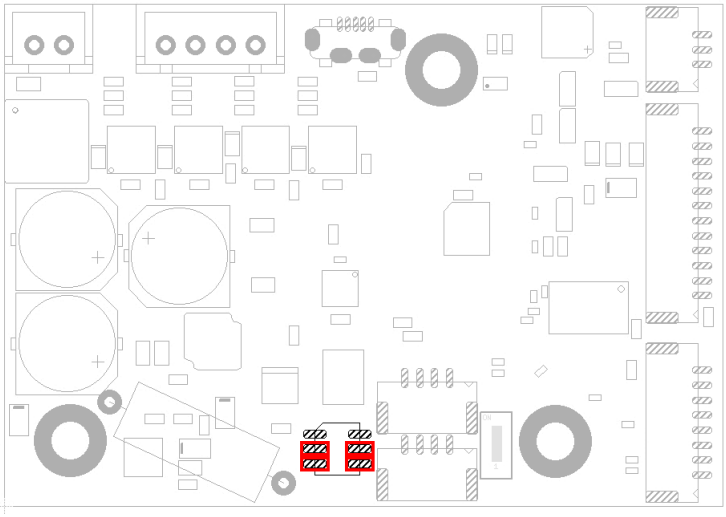
RS-485 setting

To use the RS-485 bus, jumpers J1 and J2 must be plugged in facing the middle of the board (see following figure).



CANopen setting

To use the CANopen bus, jumpers J1 and J2 must be plugged in facing the edge of the board (see following figure).



4 Commissioning

Described in this chapter is how you establish communication with the controller and set the necessary parameters to make the motor ready for operation. You can configure the controller via USB, Modbus RTU (RS-485/RS-232) or the CANopen bus.

The *Plug & Drive Studio* software offers you an option for performing the configuration and adapting the controller to the connected motor. You can find further information in document *Plug & Drive Studio: Quick Start Guide* at **us.nanotec.com**.

Observe the following note:

Note

- EMC: Current-carrying cables – particularly around supply and motor cables – produce electromagnetic alternating fields.
- These can interfere with the motor and other devices. Nanotec recommends the following measures:
- Use shielded cables and earth the cable shielding on both ends over a short distance.
- Use cables with cores in twisted pairs.
- Keep power supply and motor cables as short as possible.
- Earth motor housing with large contact area over a short distance.
- Lay supply, motor and control cables physically separate from one another.

4.1 Configuration via USB

4.1.1 General

The following options are available for configuring the controller via USB:

Configuration file

This file can be saved to the controller via the USB connection. For further information, read chapters **USB connection** and **Configuration file**.

NanoJ program

This program can be programmed, compiled and then transferred to the controller with *NanoJ* via USB. For further information, read chapters **NanoJ program** and **Programming with NanoJ**.

After connecting to a voltage supply, the controller reads out the configuration in the following order:

1. The configuration file is read out and processed.
2. The NanoJ program is started.

4.1.2 USB connection

If the controller is connected to a PC via a USB cable, the controller behaves like a removable storage device. No further drivers are required.

Three files are displayed: the configuration file (`cfg.txt`), the NanoJ program (`vmmcode.usr`) and the information file (`info.bin`), where the serial numbers and firmware version of the product can be found.

You can thereby store the configuration file or the NanoJ program on the controller. The voltage supply of the controller must also be connected during USB operation.

Note

- Only use a standard Micro USB cable. Never use a USB cable that manufacturers of mobile phones include with their products. These USB cables could have a different plug shape or pin assignment.
- Do not save any files on the controller other than those listed below:

  1. `cfg.txt`
  2. `vmmcode.usr`
  3. `info.bin`
  4. `reset.txt`
  5. `firmware.bin`

  Any other file is deleted when the voltage supply of the controller is switched on!

Tip

Because it is often necessary during commissioning to copy the same file to the controller following an update, it is recommended that a script file be used to perform this task.

- Under Windows, you can create a text file with file extension `bat` and the following content:

```
copy <SOURCE> <TARGET>
```

- Under Linux, you can create a script with file extension `sh` and the following content:

```
#!/bin/bash
cp <SOURCE> <TARGET>
```

4.1.3 Configuration file

General

The `cfg.txt` configuration file is used to preset values for the object dictionary to a certain value during startup. This file uses a special syntax to make accessing the objects of the object dictionary as easy as possible. The controller evaluates all assignments in the file from top to bottom.

Note

If you delete the configuration file, the controller recreates the file (without content) on the next restart.

Reading and writing the file

How to access the file:

1. Connect and switch on the voltage supply.
2. Connect the controller to your PC using the USB cable.
3. After the PC has detected the device as a removable storage device, navigate in the Explorer to the directory of the controller. File `cfg.txt` (for a PD4C, the file is named `pd4ccfg.txt`) is stored there.
4. Open this file with a simple text editor, such as Notepad or Vi. Do not use any programs that use markup (LibreOffice or similar).

Tip

To be able to connect the controller with *Plug & Drive Studio* via the *virtual COM port* mit verbinden zu können, insert the following line:

```
2102:00=0x19000F
```

After you have made changes to the file, proceed as follows to apply the changes:

1. Save the file if you have not yet already done so.
2. Disconnect the USB cable from the controller.
3. Disconnect the voltage supply from the controller for approx. 1 second until the power LEDs stop flashing.
4. Reconnect the voltage supply. When the controller is now restarted, the values in the configuration file are read out and applied.

Tip

To restart the controller, you can also copy an empty `reset.txt` file to the controller. This restarts the controller. The `reset.txt` file is deleted on the next restart.

Structure of the configuration file

Comments

Lines that begin with a semicolon are ignored by the controller.

Example

```
; This is a comment line
```

Assignments

Note

Before setting a value, determine its data type (see chapter **Description of the object dictionary**)! The controller does not validate entries for logical errors!

Values in the object dictionary can be set with the following syntax:

```
<Index>:<Subindex>=<Value>
```

<Index>

This value corresponds to the index of the object and is interpreted as a hexadecimal number. The value must always be specified with four digits.

<Subindex>

This value corresponds to the subindex of the object and is interpreted as a hexadecimal number. The value must always be specified with two digits.

<Value>

The value that is to be written in the object is interpreted as a hexadecimal number. Hexadecimal numbers are to be prefixed with "`0x`".

Example

Set object 2031_h:00 (rated current) to the value "600" (mA):

```
2031:00=600
```

Set object 3202_h:00 to the value "8" (activate current reduction while at a standstill in *open loop* mode):

```
3202:00=8
```

Set object 2057_h:00 to the value "512" and object 2058_h to the value "4" (*quarter step* step mode in clock-direction mode):

```
2057:00=512
```

```
2058:00=4
```

Note

- There must be no blank characters to the left and right of the equal sign. The following assignments are not correct:
  `6040:00 =5`
  `6040:00= 5`
  `6040:00 = 5`
- The number of places must not be changed. The index must be four characters long and the subindex two characters long. The following assignments are not correct:
  `6040:0=6`
  `6040=6`
- Blank spaces at the start of the line are not permitted.

4.1.4 NanoJ program

A *NanoJ program* can be executed on the controller. To load and start a program on the controller, proceed as follows:

1. Write and compile your program as described in chapter **Programming with NanoJ**.
2. Connect the voltage supply to the controller and switch on the voltage supply.
3. Connect the controller to your PC using the USB cable.
4. After the PC has detected the device as a removable storage device, open an Explorer window and delete file `vmmcode.usr` on the controller.
5. Navigate in the Explorer to the directory with your program. The compiled file has the same name as the source code file, only with file extension `.usr`. Rename this file `vmmcode.usr`.
6. Copy file `vmmcode.usr` to the controller.
7. Disconnect the voltage supply from the controller for approx. 1 second until the power LEDs stop flashing.
8. Reconnect the voltage supply. When the controller now starts, the new *NanoJ program* is read in and started.

Tip

To restart the controller, you can also copy an empty `reset.txt` file to the controller. This restarts the controller. The `reset.txt` file is deleted on the next restart.

Note

- The *NanoJ program* on the controller must have file name `vmmcode.usr`.
- If the *NanoJ program* was deleted, an empty file named `vmmcode.usr` is created the next time the controller is started.

Tip

It is possible to automate the deletion of the old *NanoJ program* and the copying of the new one with a script file:

- Under Windows, you can create a file with file extension `bat` and the following content:

```
copy <SOURCE_PATH>\<OUTPUT>.usr <TARGET>:\vmmcode.usr
```

  For example:

```
copy c:\test\main.usr n:\vmmcode.usr
```

- Under Linux, you can create a script with file extension `sh` and the following content:

```
#!/bin/bash
cp <SOURCE_PATH>/<OUTPUT>.usr <TARGET_PATH>/vmmcode.usr
```

4.2 Configuration via CANopen

All settings for CANopen can be written in file `cfg.txt` or via the memory mechanism (for further information, see chapter **Saving objects**).

The data are read out in the following sequence here:

1. First, the stored values are applied.
2. Next, the values from `cfg.txt` are applied.

4.2.1 Communication settings

Described in the following chapters is how you can change the communication settings.

The controller is configured per default for node-ID 127 and a baud rate of 1 Mbaud.

Setting node-ID and baud rate

In the default setting, the controller starts with a node-ID of 127. If a different node-ID is needed, the new value of the node-ID is entered in object **2009**$_h$.

In the default setting, the controller starts with a baud rate of 1 MBd. The baud rate is entered in object **2005**$_h$. The value for the corresponding baud rate can be found in the following table.

Value		Baud rate in kBd
dec	hex	
129	81	10

Value		Baud rate in kBd
dec	hex	
130	82	20
131	83	50
132	84	125
133	85	250
134	86	500
136	88	1000

4.2.2 Establishing communication

CANopen

Before commissioning, we recommend reading chapters **Pin assignment** and **Configuration via CANopen**.

1. Connect the CANopen master to the controller via the CAN- and CAN+ cables. Check the connection of your CAN-GND and that the necessary **120 ohm termination resistor** is present between CAN+ and CAN-.
2. Supply the controller with voltage.
3. Change the configuration values if necessary, see **Configuration via CANopen**.
   The controller is set per default to node-ID 127, baud rate 1 Mbaud.
4. To test the interface, send bytes `40 41 60 00 00 00 00 00` to the controller.
   Statusword (6041_h) was read; you receive this response: `4B 41 60 00 XX XX 00 00`.

4.3 Configuring via Modbus RTU

Described in the following chapters is how you can establish the communication.

The controller is set at the factory to slave address 5, baud rate 19200 Kbaud, even parity, 1 stop bit.

4.3.1 Communication settings

The following settings can be performed:

Configuration	Object	Value range	Factory settings
Slave address	**2028**$_h$	1 to 247	5
Baud rate	**202A**$_h$	7200 to 256000	19200
Parity	**202D**$_h$	• None: 0x00 • Even: 0x04 • Odd: 0x06	0x04 (Even)

The number of data bits is always "8" here. The number of stop bits is dependent on the parity setting:

- No parity: 2 stop bits
- "Even" or "Odd" parity: 1 stop bit

The following baud rates are supported:

- 7200
- 9600

- 14400
- 19200
- 38400
- 56000
- 57600
- 115200
- 128000
- 256000

4.3.2 Establishing communication

1. Connect the *Modbus master* to the controller via the RS-485+ and RS-485- (see **Connector X7 – CANopen/RS-485 IN**) or RS-232-Tx and RS232-Rx (see **Connector X4 – RS-232 connection**) cables.

   If using RS-485, mount jumpers J1 and J2 in the correct position (see **RS-485 setting**).
2. Supply the controller with voltage.
3. Change the configuration values if necessary.

   The controller is set at the factory to slave address 5, baud rate 19200 Kbaud, even parity, 1 stop bit.

   The following settings can be performed:

Configuration	Object	Value range	Factory settings
Slave address	**2028**$_h$	1 to 247	5
Baud rate	**202A**$_h$	7200 to 256000	19200
Parity	**202D**$_h$	• None: 0x00 • Even: 0x04 • Odd: 0x06	0x04 (Even)

4. To test the interface, send bytes `05 65 55 8A AE` to the controller (you can find a detailed description of the Modbus function codes in chapter **Modbus RTU**).
   The object dictionary is read out.

4.4 Setting the motor data

Prior to commissioning, the motor controller requires a number of values from the motor data sheet.

- Number of pole pairs: Object **2030**$_h$:00$_h$ (pole pair count) The number of motor pole pairs is to be entered here. With a stepper motor, the number of pole pairs is calculated using the step angle, e.g., 1.8° = 50 pole pairs, 0.9° = 100 pole pairs (see step angle in motor data sheet). With BLDC motors, the number of pole pairs is specified directly in the motor data sheet.
- Setting the motor current / motor type:
  - Stepper motor only: Object **2031**$_h$:00$_h$: Rated current (bipolar) in mA (see motor data sheet)
    - Object **2031**$_h$:00$_h$: Rated current (bipolar) in mA (see motor data sheet)
    - Object **3202**$_h$:00$_h$ (Motor Drive Submode Select): Defines motor type stepper motor, activates current reduction on motor standstill: 0000008h. See also chapter **Commissioning *open loop***.
  - BLDC motor only:
    - Object **2031**$_h$:00$_h$ Peak current in mA (see motor data sheet)
    - Object **203B**$_h$:01$_h$ Rated current in mA (see motor data sheet)
    - Object **203B**$_h$:02$_h$ Maximum duration of the peak current in ms (for initial commissioning, a value of 100 ms is recommended; this value is to be adapted later to the specific application).

- Object **3202**$_h$:00$_h$ (Motor Drive Submode Select): Defines motor type BLDC: 00000041h

4.5 Connecting the motor

After setting the motor parameters, see **Setting the motor data**, connect the motor and, if applicable, the present sensors (encoders / Hall sensors).

- Connect the motor:
  - to connection X2, see **Connector X2 – motor connection**
- Connect encoders / Hall sensors:
  - to connection X6, see **Connector X6 – encoder/Hall sensor**

4.6 Auto setup

To determine a number of parameters related to the motor and the connected sensors (encoders/Hall sensors), an auto setup is performed. **Closed Loop** operation requires a successfully completed auto setup.

Note

- Note the following prerequisites for performing the auto setup:
- The motor must be load-free.
- The motor must not be touched.
- The motor must be able to turn freely in any direction.
- No NanoJ programs may be running (object 2300$_h$:00$_h$ bit 0 = "0", see **2300h NanoJ Control**).

Tip

Execution of the auto setup requires a relatively large amount of processor computing power. During the auto setup, this may result in fieldbuses not being operated in a timely manner.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Tip

As long as the motor connected to the controller or the sensors for feedback (encoders/Hall sensors) are not changed, auto setup is only to be performed once during initial commissioning.

4.6.1 Parameter determination

Auto setup determines various parameters of the connected motor and of the present sensors by means of multiple test runs and measurement runs. To a certain extent, the type and number of parameters are dependent on the respective motor configuration.

Parameter	All motors independent of the configuration
Motor type (stepper motor or BLDC motor)	X

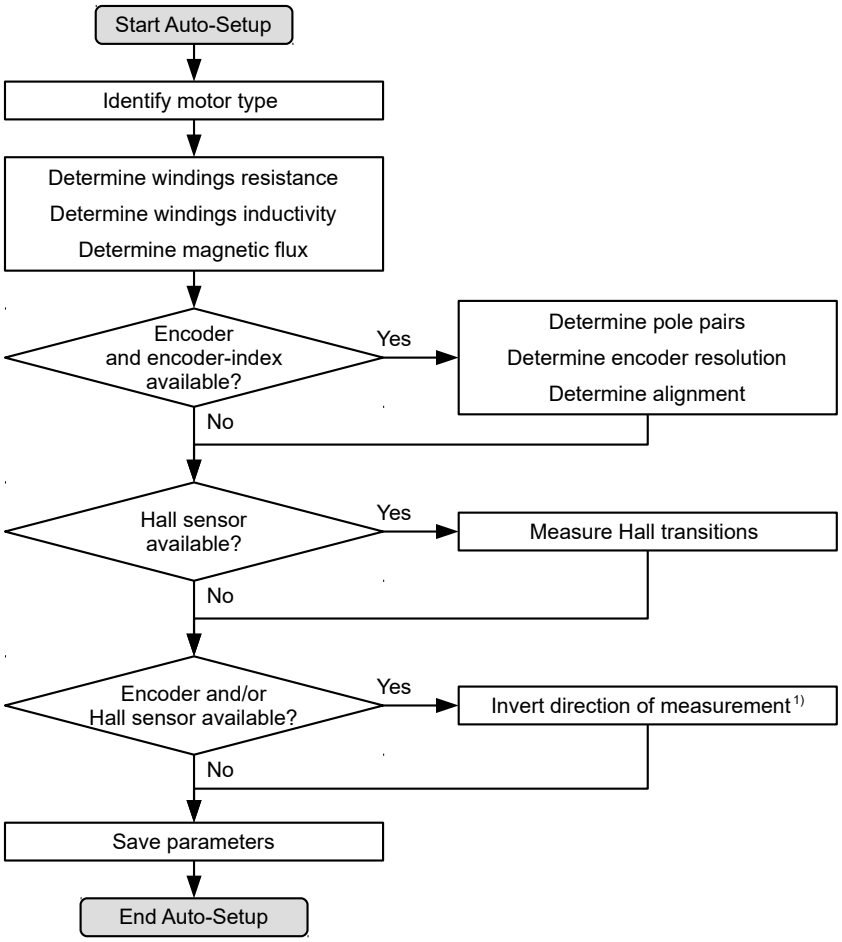
Parameter	All motors independent of the configuration
Winding resistance	X
Winding inductance	X
Interlinking flux	X

Parameter	Motor without encoder	Motor with encoder and index	Motor with encoder without index
Encoder resolution	-	X	---
Alignment (shifting of the electrical zero to the index.)	-	X	---

Parameter	Motor without Hall sensor	Motor with Hall sensor
Hall transitions	-	X

4.6.2 Execution

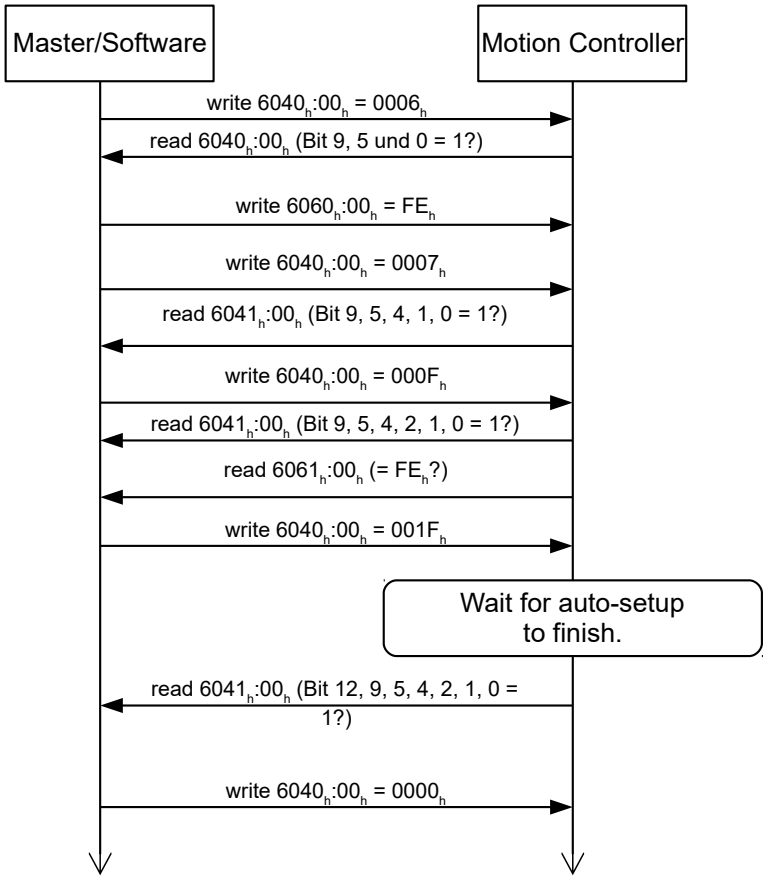
1. To preselect the *auto setup* operating mode, enter the value "-2" (="FE_h") in object 6060_h:00_h. The *power state machine* must now switch to the *Operation enabled* state, see **CiA 402 Power State Machine**.
2. Start *auto setup* by setting bit 4 *OMS* in object 6040_h:00_h (controlword).



While the auto setup is running, the following tests and measurements are performed in succession:

1) To determine the values, the direction of the measurement method is reversed and edge detection re-evaluated.

Value 1 in bit 12 *OMS* in object 6041_h:00_h (statusword) indicates that the auto setup was completely executed and ended. In addition, bit 10 *TARG* in object 6041_h:00_h can be used to query whether (= "1") or not (= "0") an encoder index was found.



4.6.3 Parameter memory

After a successful *auto setup*, the determined parameter values are automatically taken over into the corresponding objects and stored with the storage mechanism, see **Saving objects** and **1010h Store Parameters**. Categories *Drive* 1010_h:05_h and *Tuning* 1010_h:06_h are used.

CAUTION

- After executing auto setup mode, the internal coordinate system is no longer valid.
- *Homing* alone does not suffice! If the controller is not restarted, unexpected reactions may result.
- Restart the device after an auto setup!

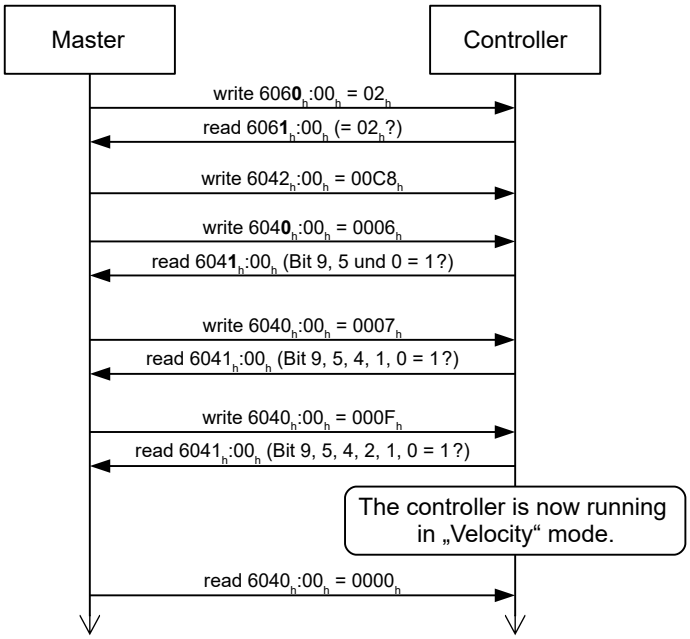
4.7 Test run

After configuring and the auto setup, a test run can be performed. As an example, the **Velocity** operating mode is used.

The values are transferred from your *CANopen master* or *Modbus master* to the controller. After every transfer, the *master* should use the status objects of the controller to ensure successful parameterization.

1. Select the *Velocity* mode by setting object **6060**$_h$ (Modes Of Operation) to the value "2".
2. Write the desired speed in **6042**$_h$.
3. Switch the *power state machine* to the *Operation enabled* state, see **CiA 402 Power State Machine**.

   The following sequence starts *Velocity* mode; the motor turns at 200 rpm.



4. To stop the motor, set controlword (**6040**$_h$) to "0".

5 General concepts

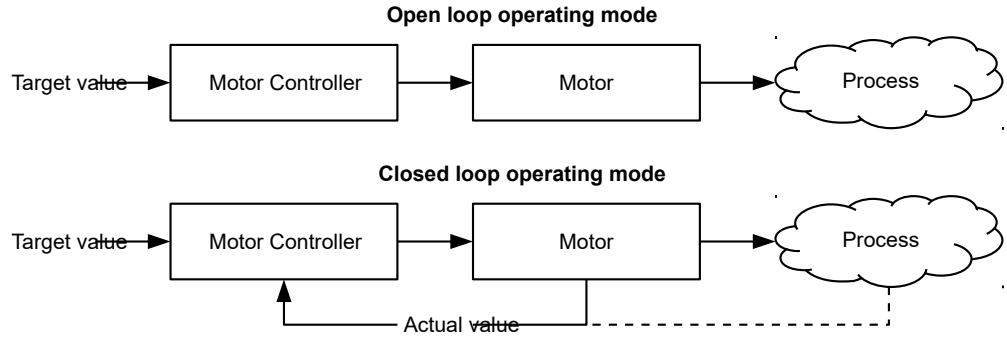
5.1 Control modes

5.1.1 General

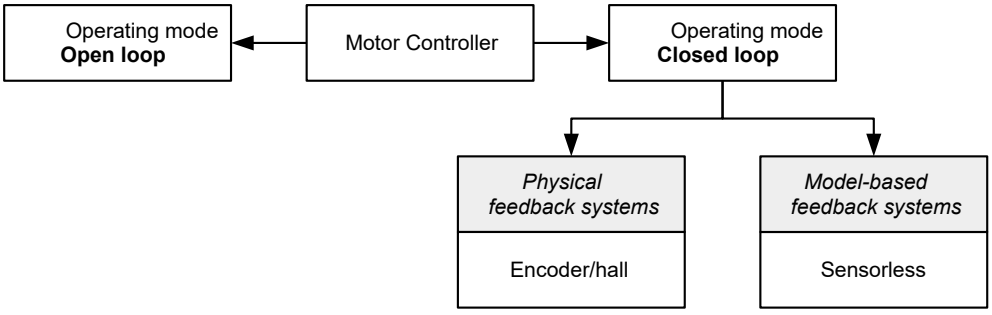
The control mode of systems without feedback is called *open loop*, the mode with feedback is called *closed loop*. In the *closed loop* control mode, it is initially irrelevant whether the fed back signals come from the motor itself or from the influenced process.

For controllers with feedback, the measured control variable (actual value) is constantly compared with a set point (set value). In the event of deviations between these values, the controller readjusts according to the specified control parameters.

Pure controllers, on the other hand, have no feedback for the value that is to be regulated. The set point (set value) is only specified.



In addition to the physical feedback systems (e.g., via encoders or Hall sensors), model-based feedback systems, collectively referred to as sensorless systems, are also used. Both feedback systems can also be used in combination to further improve the control quality.



Summarized in the following are all possible combinations of control modes and feedback systems with respect to the motor technology. Support of the respective control mode and feedback is controller-specific and is described in chapters **Pin assignment** and **Operating modes**.

Control mode	Stepper motor	BLDC motor
Open Loop	yes	no
Closed Loop	yes	yes

Feedback	Stepper motor	BLDC motor
Hall	no	yes
Encoder	yes	yes

Feedback	Stepper motor	BLDC motor
Sensorless	yes	yes

Various operating modes can be used depending on the control mode. The following list contains all the types of operation that are possible in the various control modes.

Operating mode	Control mode	
	Open Loop	Closed Loop
Profile Position	yes	yes
Velocity	yes	yes
Profile Velocity	yes	yes
Profile Torque	no[1)]	yes
Homing	yes[2)]	yes
Interpolated Position Mode	yes[3)]	yes
Cyclic Synchronous Position	yes[3)]	yes
Cyclic Synchronous Velocity	yes[3)]	yes
Cyclic Synchronous Torque	no[1)]	yes
Clock-direction	yes	yes

1) The **Profile Torque** and **Cyclic Synchronous Torque** torque operating modes are not possible in the *open loop* control mode due to a lack of feedback.

2) Exception: Homing on block is not possible due to a lack of feedback.

3) Because ramps and speeds in operating modes **Cyclic Synchronous Position** and **Cyclic Synchronous Velocity** follow from the specified points of the master, it is not normally possible to preselect these parameters and to ascertain whether a step loss can be excluded. It is therefore not advisable to use these operating modes in combination with *open loop* control mode.

5.1.2 Open Loop

Introduction

Open loop mode is only used with stepper motors and is, by definition, a control mode without feedback. The field rotation in the stator is specified by the controller. The rotor directly follows the magnetic field rotation without step losses as long as no limit parameters, such as the maximum possible torque, are exceeded. Compared to *closed loop*, no complex internal control processes are needed in the controller. As a result, the requirements on the controller hardware and the controller logic are very low. *Open loop* mode is used primarily with price-sensitive applications and simple movement tasks.

Because, unlike *closed loop*, there is no feedback for the current rotor position, no conclusion can be drawn on the counter torque being applied to the output side of the motor shaft. To compensate for any torque fluctuations that arise on the output shaft of the motor, in *open loop* mode, the controller always supplies the maximum possible (e.g., specified by parameters) set current to the stator windings over the entire speed range. The high magnetic field strength thereby produced forces the rotor to assume the new steady state in a very short time. This torque is, however, opposite that of rotor's inertia. Under certain operating conditions, this combination is prone to resonances, comparable to a spring-mass system.

Commissioning

To use *open loop* mode, the following settings are necessary:

- In object **2030**$_h$ (Pole Pair Count), enter the number of pole pairs (see motor data sheet: for a stepper motor with 2 phases, a step angle of 1.8° corresponds to 50 pole pairs and 0.9° corresponds to 100 pole pairs).
- In object **2031**$_h$ (Max Current), enter the maximum current in mA (see motor data sheet).
- In object **3202**$_h$ (Motor Drive Submode Select), set bit 0 (CL/OL) to the value "0".
- If the clock-direction mode is to be used, then observe chapter **Clock-direction mode**.

If necessary, current reduction on motor standstill should be activated to reduce the power loss and heat build-up. To activate current reduction, the following settings are necessary:

- In object **3202**$_h$ (Motor Drive Submode Select), set bit 3 (CurRed) to "1".
- In object **2036**$_h$ (Open Loop Current Reduction Idle Time), the time in milliseconds is specified that the motor must be at a standstill before current reduction is activated.
- In object **2037**$_h$ (Open Loop Current Reduction Value/factor), the root mean square is specified to which the rated current is to be reduced if current reduction is activated in *open loop* and the motor is at a standstill.

Optimizations

Depending on the system, resonances may occur in *open loop* mode; susceptibility to resonances is particularly high at low loads. Practical experience has shown that, depending on the application, various measures are effective for largely reducing resonances:

- Reduce or increase current, see object **2031**$_h$ (Max Current). Excessive torque reserve promotes resonances.
- Reduce or increase the operating voltage, taking into account the product-specific ranges (with sufficient torque reserve). The permissible operating voltage range can be found in the product data sheet.
- Optimize the control parameters of the current controller via objects **3210**$_h$:09$_h$ (I_P) and **3210**$_h$:0A$_h$ (I_I).
- Adjustments to the acceleration, deceleration and/or target speed depending on the selected control mode:

  Profile Position operating mode

  Objects **6083**$_h$ (Profile Acceleration), **6084**$_h$ (Profile Deceleration) and **6081**$_h$ (Profile Velocity).

  Velocity operating mode

  Objects **6048**$_h$ (Velocity Acceleration), **6049**$_h$ (Velocity Deceleration) and **6042**$_h$ (Target Velocity).

  Profile Velocity operating mode

  Objects **6083**$_h$ (Profile Acceleration), **6084**$_h$ (Profile Deceleration) and **6081**$_h$ (Profile Velocity).

  Homing operating mode

  Objects **609A**$_h$ (Homing Acceleration), **6099**$_h$:01$_h$ (Speed During Search For Switch) and **6099**$_h$:02$_h$ (Speed During Search For Zero).

  Interpolated Position Mode operating mode

  The acceleration and deceleration ramps can be influenced with the higher-level controller.

  Cycle Synchronous Position operating mode

  The acceleration and deceleration ramps can be influenced via the external "position specification / time unit" targets.

  Cycle Synchronous Velocity operating mode

  The acceleration and deceleration ramps can be influenced via the external "position specification / time unit" targets.

Clock-Direction operating mode

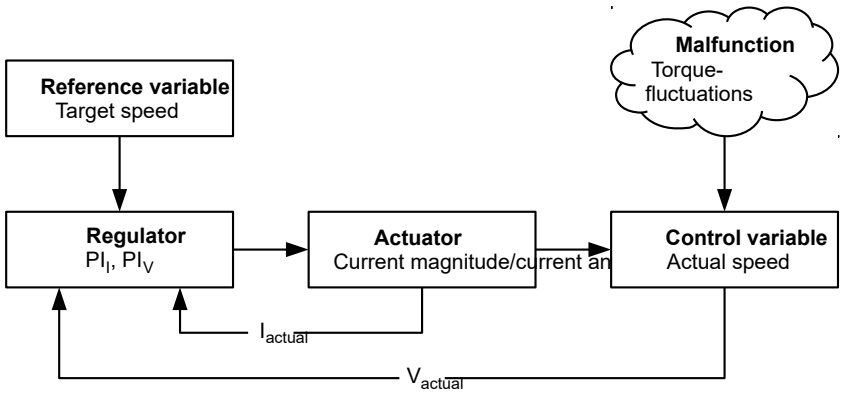
Change of the step resolution via objects **2057**$_h$ (Clock Direction Multiplier) and **2058**$_h$ (Clock Direction Divider). Optimize acceleration / deceleration ramps by adjusting the pulse frequency to pass through the resonance range as quickly as possible.

5.1.3 Closed Loop

Introduction

The *closed loop* theory is based on the idea of a control loop. A disturbance acting on a system should be compensated for quickly and without lasting deviation to adjust the control variable back to the set point.

Closed loop using a speed control as an example:



PI_I	=	Proportional-integral current control loop
PI_V	=	Proportional-integral velocity control loop
I_{actual}	=	Actual current
V_{actual}	=	Actual speed

The *closed loop* method is also referred to as "sine commutation via an encoder with field-oriented control". At the heart of *closed loop* technology is the performance-adjusted current control as well as the feedback of the actual values of the process. Using the encoder signals, the rotor orientation is recorded and sinusoidal phase currents generated in the motor windings. Vector control of the magnetic field ensures that the magnetic field of the stator is always perpendicular to that of the rotor and that the field strength corresponds precisely to the desired torque. The current thereby controlled in the windings provides a uniform motor force and results in an especially smooth-running motor that can be precisely regulated.

The feedback of the control variables necessary for *closed loop* mode can be realized with various technologies. In addition to the physical feedback with encoders or Hall sensors, it is also possible to virtually record the motor parameters through software-based model calculation. Physical variables, such as speed or back-EMF, can be reconstructed with the help of a so-called "observer" from the data of the current controller. With this sensorless technology, one has a "virtual rotary encoder", which – above a certain minimum speed – supplies the position and speed information with the same precision as a real optical or magnetic encoder.

All controllers from Nanotec that support *closed loop* mode implement a field oriented control with sine commutated current control. Thus, the stepper motors and BLDC motor are controlled in the same way as a servo motor. With *closed loop* mode, step angle errors can be compensated for during travel and load angle errors corrected within one full step.

Commissioning

An auto setup must be performed before using *closed loop* mode. The auto setup operating mode automatically determines the necessary parameters (e.g., motor data, feedback systems) that are necessary for optimum operation of the field oriented control. All information necessary for performing the auto setup can be found in chapter **Auto setup**.

To use *closed loop* mode, certain settings are necessary depending on the motor type and feedback; see chapter **Setting the motor data**. Bit 0 in **3202**$_h$ must be set . If the encoder is used for the commutation, the index of the encoder must be passed over at least once after switching on (bit 15 in **6041h Statusword** is set).

5.2 CiA 402 Power State Machine

5.2.1 State machine

CiA 402

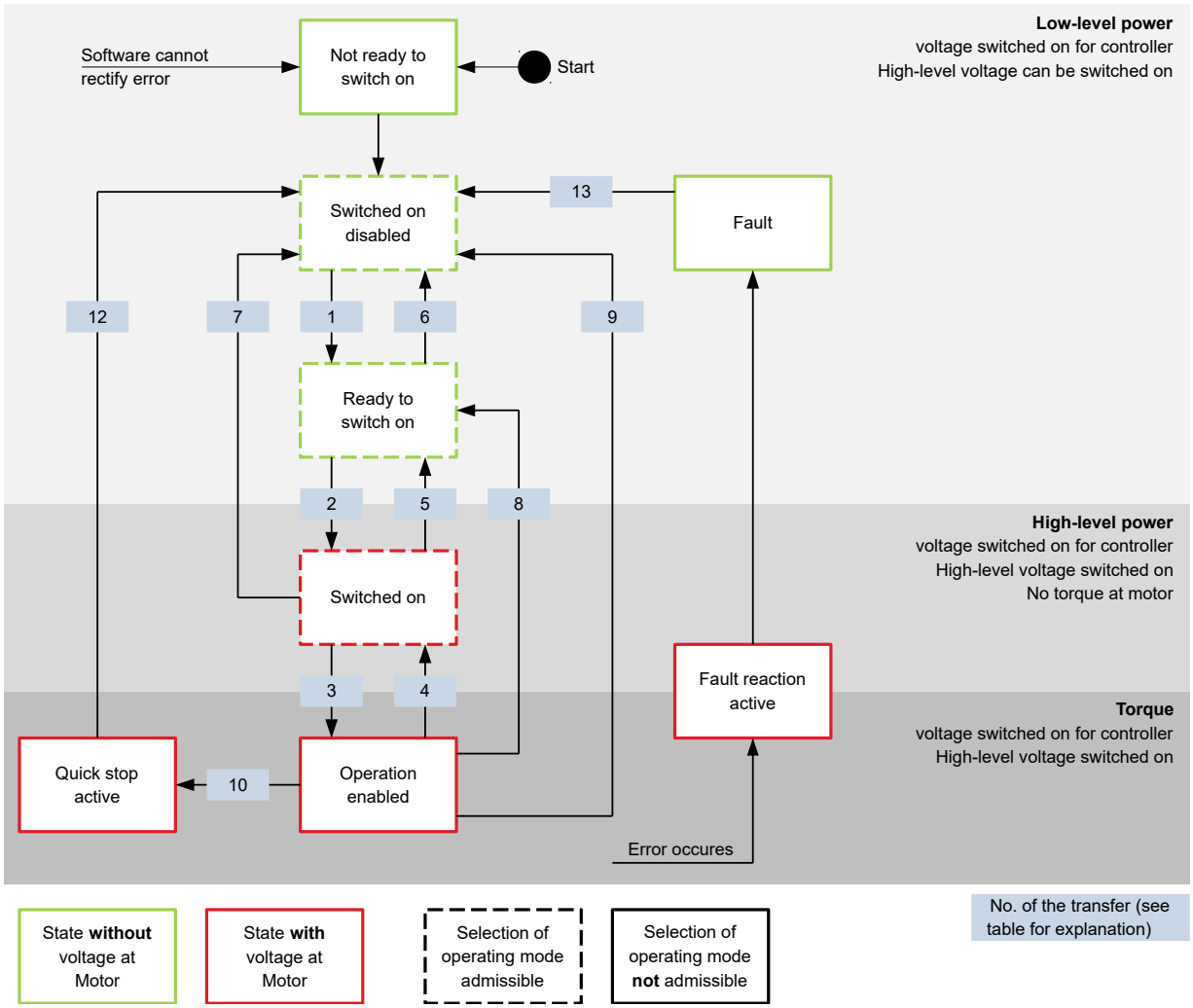
To switch the controller to the ready state, it is necessary to run through a *state machine*. This is defined in *CANopen standard 402*. State changes are requested in object **6040**$_h$ (controlword). The actual state of the state machine can be found in object **6041**$_h$ (statusword).

Controlword

State changes are requested via object **6040**$_h$ (controlword).

State transitions

The diagram shows the possible state transitions.



Listed in the following table are the bit combinations for the controlword that result in the corresponding state transitions. An X here corresponds to a bit state that requires no further consideration. The only exception is the resetting of the error (fault reset): the transition is only requested by the rising edge of the bit.

Command	Bit in object 6040_h					Transition
	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0	
Shutdown	0	X	1	1	0	1, 5, 8
Switch on	0	0	1	1	1	2
Disable voltage	0	X	X	0	X	6, 7, 9, 12
Quick stop	0	X	0	1	X	10
Disable operation	0	0	1	1	1	4
Enable operation	0	1	1	1	1	3
Fault reset	⌐↑	X	X	X	X	13

Holding torque in the *Switched on* state

Ex works, no holding torque is built up in the *Switched on* state. If a holding torque is already needed in this state, the value "1" must be written in **3212**$_h$:01$_h$.

Note

If the *Holding torque in the switched on state* option is active, changing the operating mode may cause the motor to jerk.

Statusword

Listed in the following table are the bit masks that break down the state of the controller.

Statusword (6041$_h$)	State
xxxx xxxx x0xx 0000	Not ready to switch on
xxxx xxxx x1xx 0000	Switch on disabled
xxxx xxxx x01x 0001	Ready to switch on
xxxx xxxx x01x 0011	Switched on
xxxx xxxx x01x 0111	Operation enabled
xxxx xxxx x00x 0111	Quick stop active
xxxx xxxx x0xx 1111	Fault reaction active
xxxx xxxx x0xx 1000	Fault

After switching on and successfully completing the self-test, the controller reaches the *Switch on disabled* state.

Operating mode

The set operating mode (**6060**$_h$) does not become active until the *Operation enabled* state. The actually active operating mode is displayed in **6061**$_h$.

The operating mode can only be set or changed in the following states (see states enclosed in a dashed border in the diagram):

- Switch on disabled
- Ready to switch on
- Switched on

It is not possible to change the operating mode in running operation (*Operation enabled*). The *Fault* state is exited if bit 7 in object **6040**$_h$ (controlword) is set from "0" to "1" (rising edge).

Note

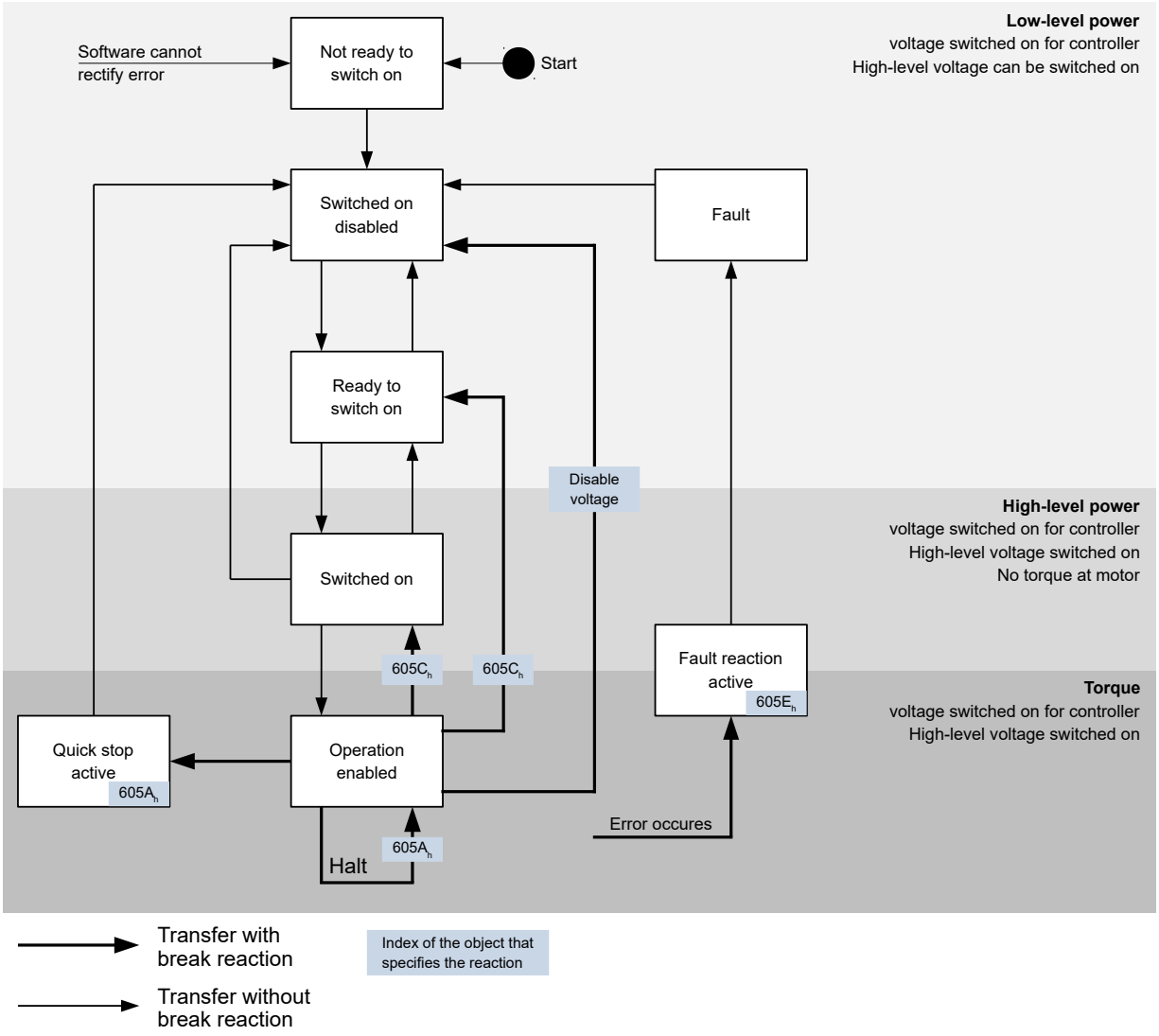
If an unrecoverable error occurs, the controller changes to the *Not ready to switch on* state and remains there.

5.2.2 Behavior upon exiting the *Operation enabled* state

Halt motion reactions

Various halt motion reactions can be programmed upon exiting the *Operation enabled* state.

The following graphic shows an overview of the halt motion reactions.



Quick stop active

Transition to the *Quick stop active* state (quick stop option):

In this case, the action stored in object **605A$_h$** is executed (see following table).

Value in object 605A$_h$	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with *slow down ramp* (braking deceleration depending on operating mode) and subsequent state change to *Switch on disabled*
2	Braking with *quick stop ramp* and subsequent state change to *Switch on disabled*
3 … 32767	Reserved

Ready to switch on

Transition to the *Ready to switch on* state (shutdown option):

In this case, the action stored in object **605B$_h$** is executed (see following table).

Value in object 605B$_h$	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with *slow down ramp* (braking deceleration depending on operating mode) and subsequent state change to *Switch on disabled*
2 … 32767	Reserved

Switched on

Transition to the *Switched on* state (disable operation option):

In this case, the action stored in object **605C$_h$** is executed (see following table).

Value in object 605C$_h$	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with *slow down ramp* (braking deceleration depending on operating mode) and subsequent state change to *Switch on disabled*
2 … 32767	Reserved

Halt

The bit is valid in the following modes:

- **Profile Position**
- **Velocity**
- **Profile Velocity**
- **Profile Torque**
- **Interpolated Position Mode**

When setting bit 8 in object **6040$_h$** (controlword), the reaction stored in **605D$_h$** is executed (see following table):

Value in object 605D$_h$	Description
-32768 … 0	Reserved
1	Braking with *slow down ramp* (braking deceleration depending on operating mode)
2	Braking with *quick stop ramp* (braking deceleration depending on operating mode)
3 … 32767	Reserved

Fault

Case of an error (fault):

If an error occurs, the motor will brake according to the value stored in object **605E$_h$**.

Value in object $605E_h$	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with *slow down ramp* (braking deceleration depending on operating mode)
2	Braking with *quick stop ramp* (braking deceleration depending on operating mode)
3 … 32767	Reserved

Following error

If a following error occurs, the motor will brake according to the value stored in object **3700_h**.

Value	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with *slow down ramp* (braking deceleration depending on operating mode)
2	Braking with *quick stop ramp* (braking deceleration depending on operating mode)
3 … 32767	Reserved

Following error monitoring can be deactivated by setting object **6065_h** to the value "-1" ($FFFFFFFF_h$).

5.3 User-defined units

The controller supports the possibility to set user-defined units. It is thereby possible to set and read out the corresponding parameters, e.g., directly in degrees [°], [mm], etc.

5.3.1 Calculation formulas for user units

Position information

All position values in *open loop* and *closed lop* mode are specified in the resolution of the virtual position encoder. This is calculated from the virtual encoder increments (**$608F_h$**:1_h (Encoder Increments)) per motor revolutions (**$608F_h$**:2_h (Motor Revolutions)):

$$\text{Virutal encoder posiiton resolution} = \frac{\text{Encoder increments } (608F_h{:}01)}{\text{Motor revolutions } (608F_h{:}02)}$$

If value **$608F_h$**:1_h or value **$608F_h$**:2_h is set to "0", the controller uses "1" in subsequent calculations. The factory settings are:

- Encoder increments **$608F_h$**:1 = "2000"
- Motor revolutions **$608F_h$**:2 = "1"

Example

$608F_h$:2_h is set to the value "1", **$608F_h$**:1_h is set to the value "2000" (default). Thus, the user unit is 2000 increments per revolution. For a stepper motor with step angle of 1.8°, this corresponds to the *one tenth* step mode.

With a target position (**607A**$_h$) of 2000, the motor moves exactly one mechanical revolution

The physical resolution of the connected position encoder (of the present feedback in general) is set in object **2052**$_h$ or determined by **Auto setup**.

Gear ratio

The gear ratio is calculated from motor revolutions (**6091**$_h$:1 (Motor Revolutions)) per axis rotation (**6091**$_h$:2 (Shaft Revolutions)) as follows:

$$\text{Gear ratio} = \frac{\text{Motor revolution } (6091_h{:}1)}{\text{Shaft revolution } (6091_h{:}2)}$$

If object **6091**$_h$:1 or object **6091**$_h$:2 is set to "0", the firmware sets the value to "1".

Feed constant

The feed constant is calculated from the feed (**6092**$_h$:1 (Feed Constant) per revolution of the drive axis (**6092**$_h$:2 (Shaft Revolutions) as follows:

$$\text{Feed rate} = \frac{\text{Feed } (6092_h{:}1)}{\text{Revolution of the drive axis } (6092_h{:}2)}$$

This is helpful for specifying the lead screw pitch for a linear axis.

If object **6092**$_h$:1 or object **6092**$_h$:2 is set to "0", the firmware sets the value to "1".

Position

The current position in user units (**6064**$_h$) and the target position (**607A**$_h$) are calculated as follows:

$$\text{Position} = \frac{608F_h{:}01 \times \text{Feed constant } (6092_h)}{608F_h{:}02 \times \text{Gear ratio } (6091_h)}$$

Speed

The speed presets of the following objects can also be specified in user units:

Object	Mode	Meaning
606B$_h$	Profile Velocity Mode	Output value of the ramp generator
60FF$_h$	Profile Velocity Mode	Speed preset
6099$_h$	Homing Mode	Speed for searching for the index / switch
6081$_h$	Profile Position Mode	Target speed
6082$_h$	Profile Position Mode	Final speed
2032$_h$	Profile Torque	Maximum speed

The internal unit is revolutions per second (rps).

The factor n for the speed is calculated from the factor for the numerator (**2061**$_h$) divided by the factor for the denominator (**2062**$_h$).

$$n_{velocity} = \frac{2061_h}{2062_h}$$

When entering values, the following applies correspondingly: Internal value = n_{speed} x input value

When outputting values, the following applies correspondingly: Output value = internal value / n_{speed}

Example

2061$_h$ is set to the value "1", **2062**$_h$ is set to the value "60" (default). Thus, the user unit is "revolutions per minute" and n_{speed} = 1/60.

If **60FF**$_h$ is written with the value "300", the internal value is set to 300 rpm x 1/60 = 5 rps.

If the motor turns at an internal speed of 5 rps, object **606B**$_h$ is set to a speed of 5 / 1/60 = 300 rpm.

Acceleration

The acceleration can also be specified in user units:

Object	Mode	Meaning
609A$_h$	Homing Mode	Acceleration
6083$_h$	Profile Position Mode	Acceleration
6084$_h$	Profile Position Mode	Braking deceleration
60C5$_h$	Profile Velocity Mode	Acceleration
60C6$_h$	Profile Position Mode	Braking deceleration
6085$_h$	"Quick stop active" state (**CiA 402 Power State Machine**)	Braking deceleration

The internal unit is revolutions per second^2 (rps^2).

The factor n for the acceleration is calculated from the scaling factor for the numerator (**2063**$_h$) divided by the scaling factor for the denominator (**2064**$_h$).

$$n_{Acceleration} = \frac{2063_h}{2064_h}$$

When entering values, the following applies correspondingly: Internal value = $n_{acceleration}$ x input value

Example

2063$_h$ is set to the value "1", **2064**$_h$ is set to the value "60". Thus, the user unit is *revolutions per minute per second* and $n_{acceleration}$ = 1/60.

If **60C5**$_h$ is set to the value "600", the internal value is set to 600 rp(s*min) x 1/60 = 10 rps^2.

If object **2063**$_h$ or object **2064**$_h$ is set to "0", the firmware sets the value to "1".

Jerk

For the jerk, objects **60A4**$_h$:1$_h$ to **60A4**$_h$:4$_h$ can be specified in user units. These objects only affect *Profile Position Mode* and *Profile Velocity Mode*.

The internal unit is revolutions per second^3 (rps^3).

The factor n for the acceleration is calculated from the factor for numerator (**2065**$_h$) divided by the factor for the denominator (**2066**$_h$).

$$n_{Jerk} = \frac{2065_h}{2066_h}$$

When entering values, the following applies correspondingly: Internal value = n_{jerk} x input value

Example

2063$_h$ is set to the value "1", **2064**$_h$ is set to the value "60". Thus, the user unit is "revolutions per minute per second squared" and n_{jerk} = 1/60.

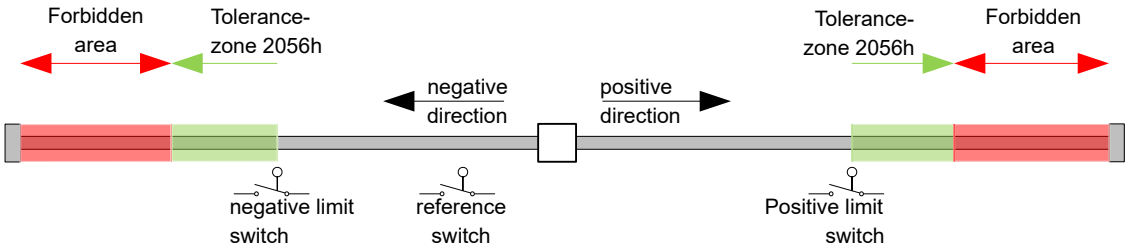
If **60A4**$_h$ is set to the value "500", the internal value is set to 500 rp(min * s^2) x 1/60 = 8.3 rps^3.

If object **2065**$_h$ or object **2066**$_h$ is set to "0", the firmware sets the value to "1".

5.4 Limitation of the range of motion

The digital inputs can be used as limit switches, as is described in chapter **Digital inputs**, if you activate this function for the inputs. The controller also supports software limit switches.

5.4.1 Tolerance bands of the limit switches



The previous figure shows the breakdown of the tolerance bands next to the limit switches:

- The tolerance zone begins immediately after the limit switch. Free movement is possible in this zone. The length of the zone can be set in object **2056**$_h$.
- If the motor moves into the forbidden range, the controller triggers an immediate stop and it switches to the *fault* state, see also **State transitions**.

5.4.2 Software limit switches

The controller takes into account software limit switches (**607D**$_h$ (Software Position Limit)). Target positions (**607A**$_h$) are limited by **607D**$_h$; the demand position (**6062**$_h$) may not be larger than the limits in **607D**$_h$. If the motor is located outside of the permissible range when setting up the limit switches, only travel commands in the direction of the permissible range are accepted.

5.5 Cycle times

The controller operates with a cycle time of 1 ms. This means that data are processed every 1 ms; multiple changes to a value (e.g., value of an object or level at a digital input) within one ms cannot be detected.

The following table includes an overview of the cycle times of the various processes.

Task	Cycle time
Application	1 ms
NanoJ application	1 ms
Current controller	31.25 µs (32 kHz)
Speed controller	31.25 µs (32 kHz)
Position controller	31.25 µs (32 kHz)

6 Operating modes

6.1 Profile Position

6.1.1 Overview

Description

Profile Position Mode is used to move to positions relative to the last target position or to an absolute position (last reference position). During the movement, the limit values for the speed, starting acceleration/braking deceleration and jerks are taken into account.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Activation

To activate the mode, the value "1" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 4 starts a travel command. This is carried out on a transition from "0" to "1".
- Bit 5: If this bit is set to "1", a travel command triggered by bit 4 is immediately executed. If it is set to "0", the just executed travel command is completed and only then is the next travel command started.
- Bit 6: With "0", the target position (**607A**$_h$) is absolute and with "1" the target position is relative. The reference position is dependent on bits 0 and 1 of object **60F2**$_h$.
- Bit 8 (Halt): If this bit is set to "1", the motor stops. On a transition from "1" to "0", the motor accelerates with the set start ramp to the target speed. On a transition from "0" to "1", the motor brakes and comes to a standstill. The braking deceleration is dependent here on the setting of the "Halt Option Code" in object **605D**$_h$.
- Bit 9 (Change on setpoint): If this bit is set, the speed is not changed until the first target position is reached. This means that, before the first target is reached, no braking is performed, as the motor should not come to a standstill at this position.

Controlword 6040$_h$		
Bit 9	**Bit 5**	**Definition**
X	1	The new target position is moved to immediately.
0	0	Positioning is completed before moving to the next target position with the new limits.
1	0	The current target position is only passed through; afterwards, the new target position is moved to with the new values.

For further information, see figure in "**Setting travel commands**".

Note

Bit 9 in the controlword is ignored if the ramp speed is not met at the target point. In this case, the controller would need to reset and take a run-up to reach the preset.

Statusword

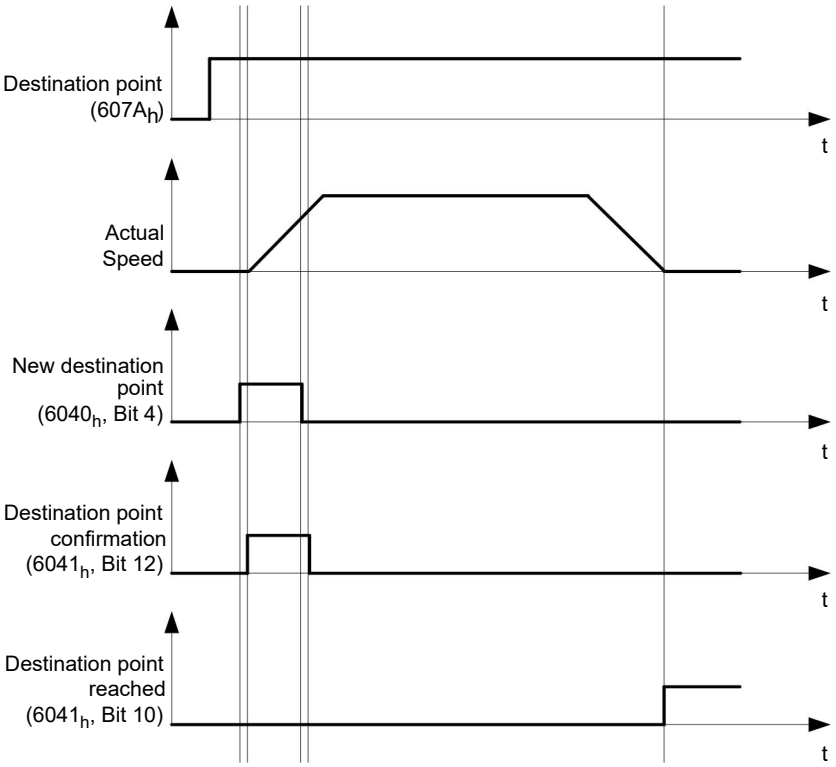
The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 10 (Target Reached): This bit is set to "1" if the last target was reached and the motor remains within a tolerance window (**6067**$_h$) for a preset time (**6068**$_h$).
- Bit 11: Limit exceeded: The demand position is above or below the limit values set in **607D**$_h$.
- Bit 12 (Set-point acknowledge): This bit confirms receipt of a new and valid set point. It is set and reset in sync with the "New set-point" bit in the controlword.
  There is an exception in the event that a new movement is started before another one has completed and the next movement is not to occur until after the first one has finished. In this case, the bit is reset if the command was accepted and the controller is ready to execute new travel commands. If a new travel command is sent even though this bit is still set, the newest travel command is ignored.
  The bit is not set if one of the following conditions is met:
  - The new target position can no longer be reached while adhering to all boundary conditions.
  - A target position was already traveled to and a target position was already specified. A new target position can only be specified after the current positioning has been concluded.
- Bit 13 (Following Error): This bit is set in *closed loop* mode if the following error is greater than the set limits (**6065**$_h$ (Following Error Window) and **6066**$_h$ (Following Error Time Out)).

6.1.2 Setting travel commands

Travel command

In object **607A**$_h$ (Target Position), the new target position is specified in user units (see "**User-defined units**"). The travel command is then triggered by setting bit 4 in object **6040**$_h$ (controlword). If the target position is valid, the controller responds with bit 12 in object **6041**$_h$ (statusword) and begins the positioning move. As soon as the position is reached, bit 10 in the statusword is set to "1".



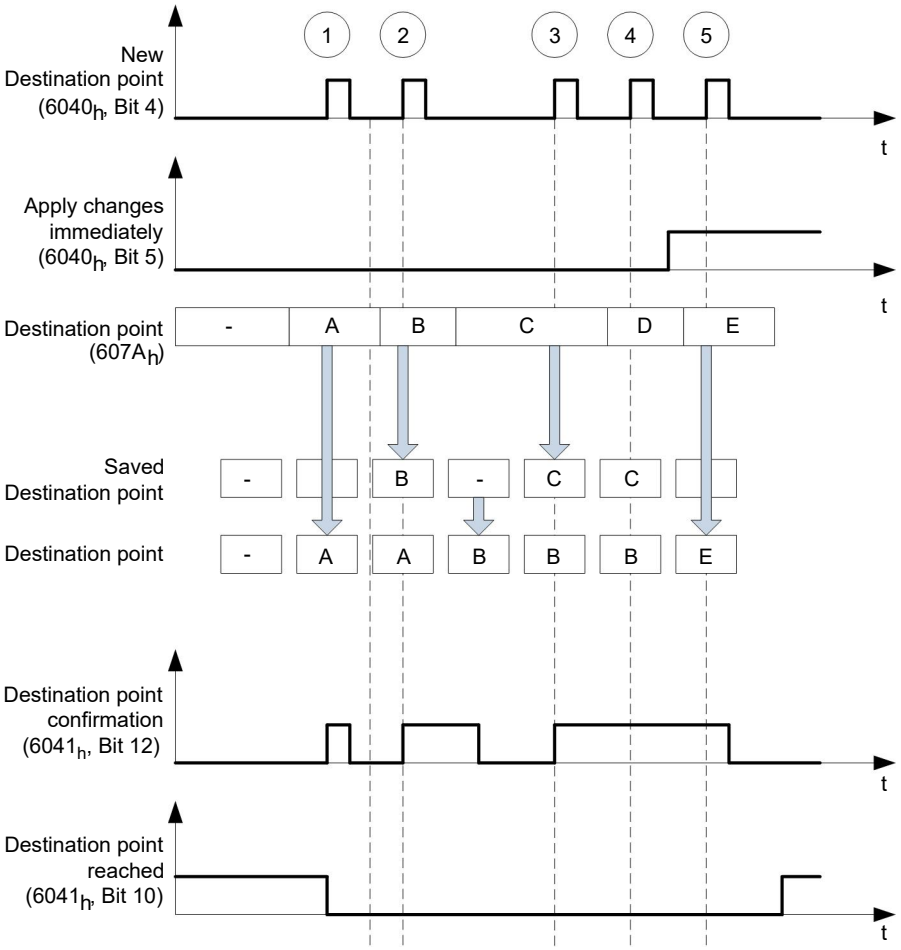
The controller can also reset bit 4 in object **6040**$_h$ (controlword) on its own. This is set with bits 4 and 5 of object **60F2**$_h$.

Other travel commands

Bit 12 in object **6041**$_h$ (statusword, set-point acknowledge) changes to "0" if another travel command can be buffered (see time 1 in the following figure). As long as a target position is being moved to, a second target position can be passed to the controller in preparation. All parameters – such as speed, acceleration, braking deceleration, etc. – can thereby be reset (time 2). If the buffer is empty, the next time can be queued up (time 3).

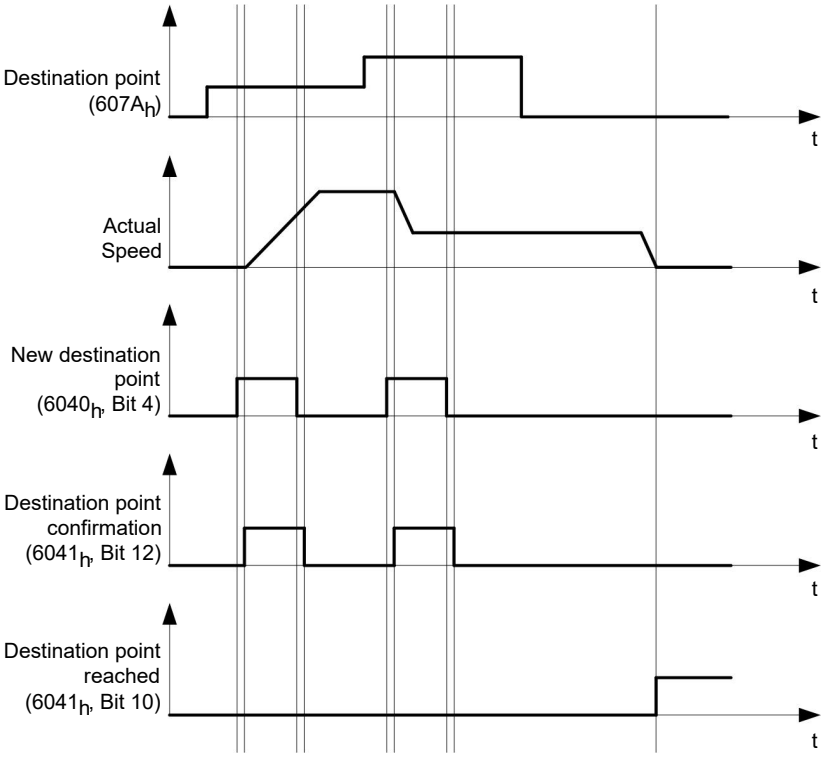
If the buffer is already full, a new set point is ignored (time 4). If bit 5 in object **6040**$_h$ (controlword, bit: "Change Set-Point Immediately") is set, the controller operates without the buffer; new travel commands are implemented directly (time 5).

Times



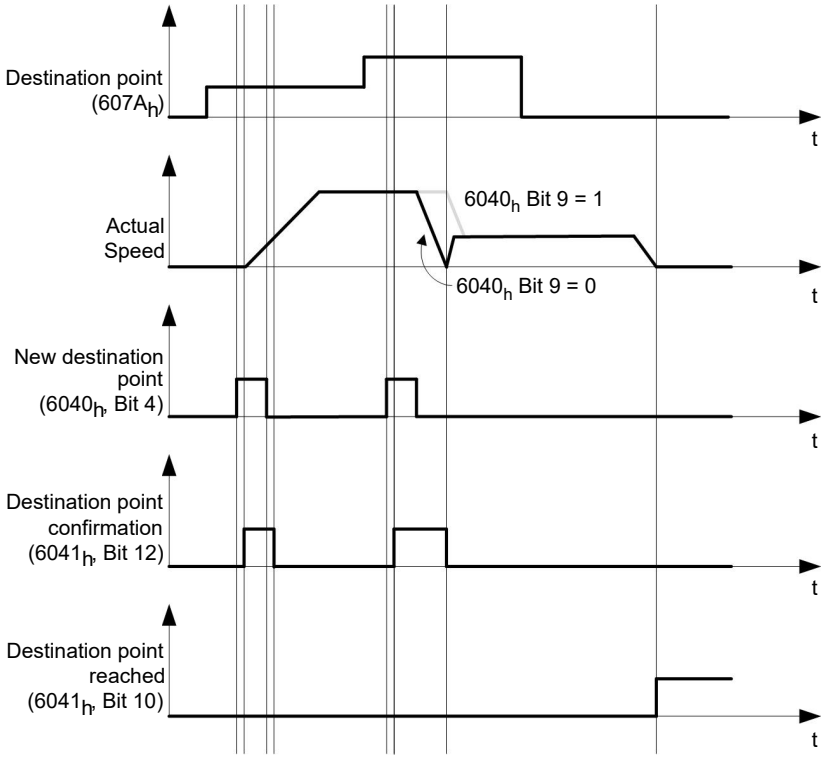
Transition procedure for second target position

The following graphic shows the transition procedure for the second target position while moving to the first target position. In this figure, bit 5 of object **6040**$_h$ (controlword) is set to "1"; the new target value is, thus, taken over immediately.



Possibilities for moving to a target position

If bit 9 in object **6040**$_h$ (controlword) is equal to "0", the current target position is first moved to completely. In this example, the final speed (**6082**$_h$) of the target position is equal to zero. If bit 9 is set to "1", the profile speed (**6081**$_h$) is maintained until the target position is reached; only then do the new boundary conditions apply.



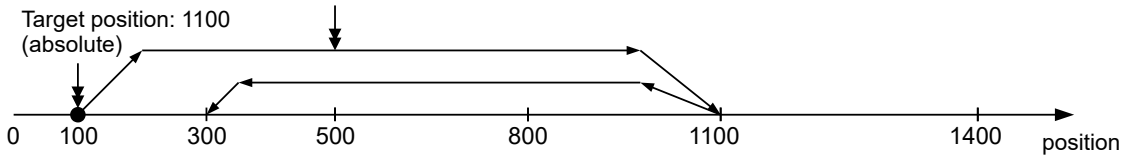
Possible combinations of travel commands

To provide a better overview of the travel commands, combinations of travel commands are listed and depicted in this chapter.

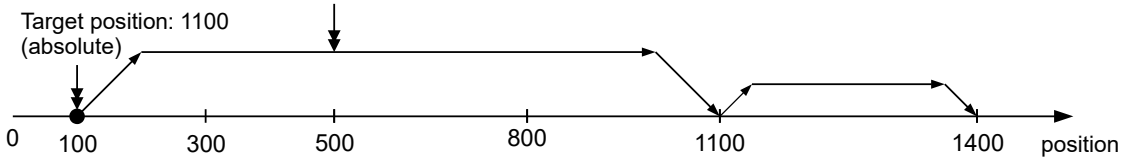
The following applies for the figures below:

- A double arrow indicates a new travel command.
- The first travel command at the start is always an absolute travel command to position 1100.
- The second movement is performed at a lower speed so as to present the graphs in a clear manner.

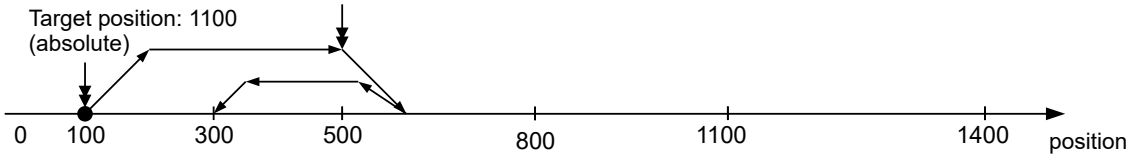
- Change on setpoint (6040_h:00 Bit 5 = 0)
- Move absolute (6040_h:00 Bit 6 = 0)
- Target position: 300



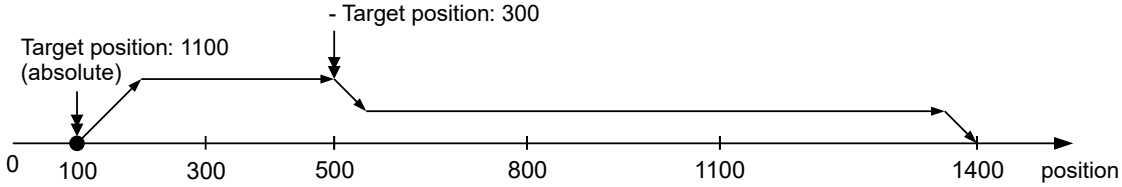
- Relative to the preceding target position ($60F2_h$:00 = 0)
- Change on setpoint (6040_h:00 Bit 5 = 0)
- Move relative (6040_h:00 Bit 6 = 1)
- Target position: 300

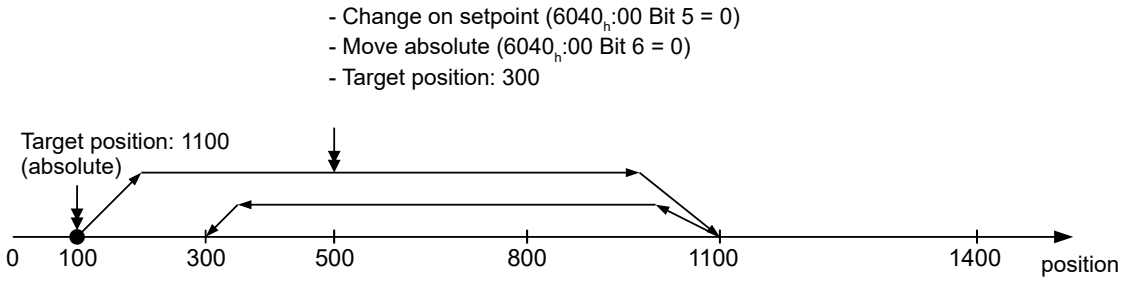


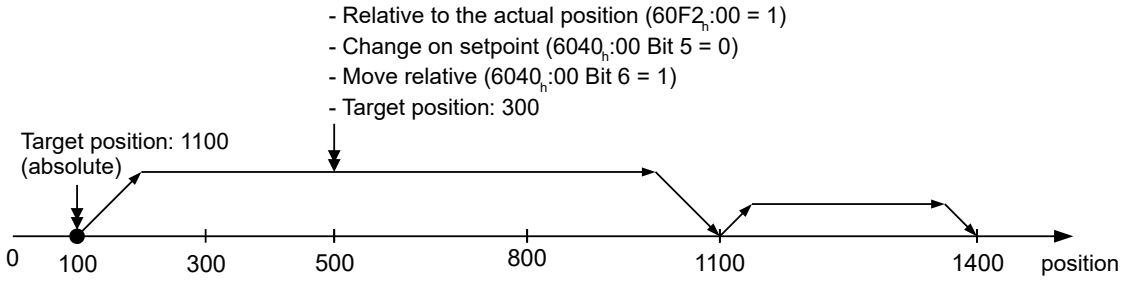
- Change set immediately (6040_h:00 Bit 5 = 1)
- Move absolute (6040_h:00 Bit 6 = 0)
- Target position: 300

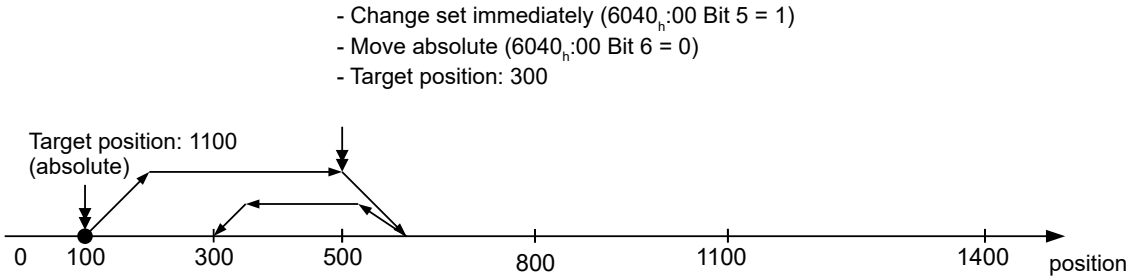


- Relative to the preceding target position ($60F2_h$:00 = 0)
- Change set immediately (6040_h:00 Bit 5 = 1)
- Move relative (6040_h:00 Bit 6 = 1)
- Target position: 300



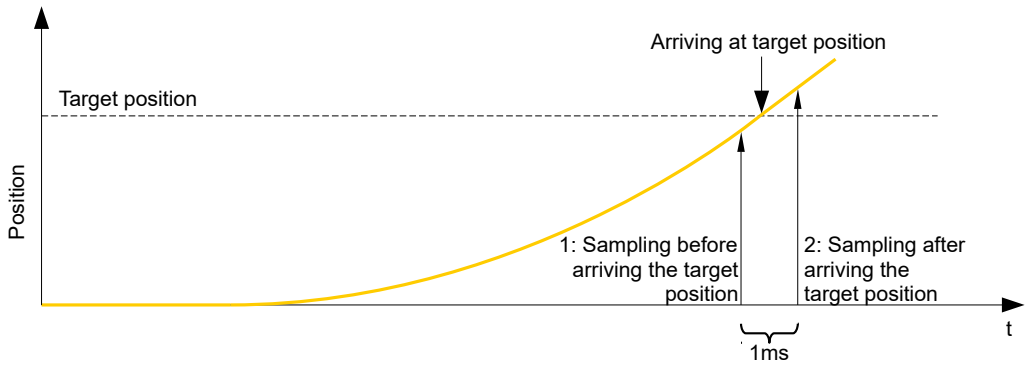






6.1.3 Loss of accuracy for relative movements

When linking together relative movements, a loss of accuracy may occur if the final speed is not set to zero. The following graphic illustrates the reason.



The current position is sampled once per millisecond. It is possible that the target position is reached between two samples. If the final speed is not equal to zero, then, after the target position is reached, the sample is used as an offset as the basis for the subsequent movement. As a result, the subsequent movement may go somewhat farther than expected.

6.1.4 Boundary conditions for a positioning move

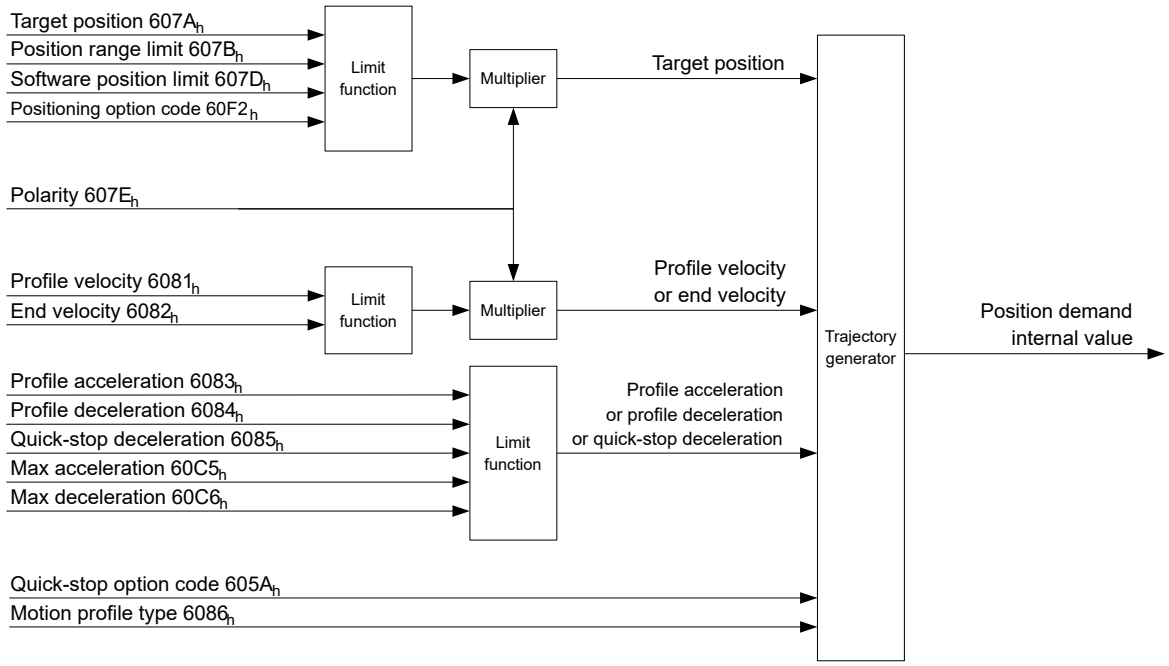
Object entries

The boundary conditions for the position that has been moved to can be set in the following entries of the object dictionary:

- **607A**$_h$: (Target Position): Planned target position
- **607D**$_h$: (Software Position Limit): Definition of the limit stops (see chapter **Software limit switches**)
- **607C**$_h$ (Home Offset): Specifies the difference between the zero position of the controller and the reference point of the machine in **user-defined units**. (See "**Homing**")
- **607B**$_h$ (Position Range Limit): Limits of a modulo operation for replicating an endless rotation axis
- **607**$_h$ (Polarity): Direction of rotation
- **6081**$_h$ (Profile Velocity): Maximum speed with which the position is to be approached
- **6082**$_h$ (End Velocity): Speed upon reaching the target position
- **6083**$_h$ (Profile Acceleration): Desired starting acceleration
- **6084**$_h$ (Profile Deceleration): Desired braking deceleration
- **6085**$_h$ (Quick Stop Deceleration): Emergency-stop braking deceleration in case of the "Quick stop active" state of the "CiA 402 Power State Machine"
- **6086**$_h$ (Motion Profile Type): Type of ramp to be traveled; if the value is "0", the jerk is not limited; if the value is "3", the values of 60A4$_h$:1$_h$–4$_h$ are set as limits for the jerk.
- **60C5**$_h$ (Max Acceleration): The maximum acceleration that may not be exceeded when moving to the end position
- **60C6**$_h$ (Max Deceleration): The maximum braking deceleration that may not be exceeded when moving to the end position
- **60A4**$_h$ (Profile Jerk), subindex 01$_h$ to 04$_h$: Objects for specifying the limit values for the jerk.
- **60F2**$_h$: (Positioning Option Code): Defines the positioning behavior

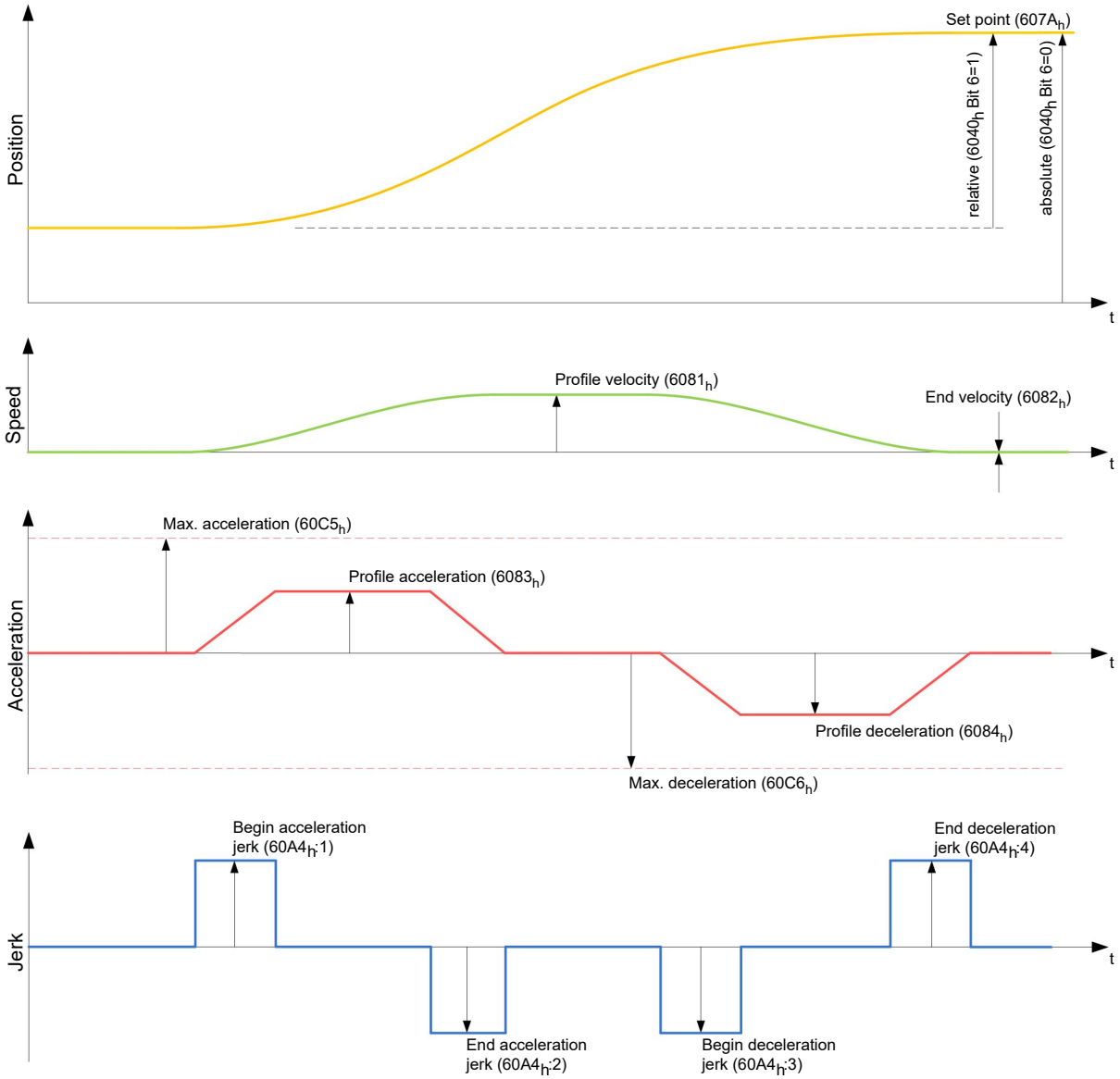
Objects for the positioning move

The following graphic shows the objects involved in the boundary conditions of the positioning move.



Parameters for the target position

The following graphic shows an overview of the parameters that are used for moving to a target position (figure not to scale).



6.1.5 Jerk-limited mode and non-jerk-limited mode

Description

A distinction is made between the "jerk-limited" and "non-jerk-limited" modes.

Jerk-limited mode

Jerk-limited positioning can be achieved by setting object **6086**$_h$ to "3". The entries for the jerks in subindices :1$_h$–4$_h$ of object **60A4** thereby become valid.

Non-jerk-limited mode

A "non-jerk-limited" ramp is traveled if the entry in object **6086**$_h$ is set to "0" (default setting).

6.2 Velocity

6.2.1 Description

This mode operates the motor at a preset target speed, similar to a frequency inverter. Unlike the *Profile Velocity Mode*, this mode does not permit the selection of jerk-limited ramps.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

6.2.2 Activation

To activate the mode, the value "2" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

6.2.3 Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 8 (Halt): If this bit is set to "1", the motor stops. On a transition from "1" to "0", the motor accelerates with the acceleration ramp to the target speed. On a transition from "0" to "1", the motor brakes according to the deceleration ramp and comes to a standstill.

6.2.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 11: Limit exceeded: The target speed is above or below the set limit values.

6.2.5 Object entries

The following objects are necessary for controlling this mode:

- **604C**$_h$ (Dimension Factor):
  The unit for speed values is defined here for the following objects. If subindices 1 and 2 are set to the value "1", the speed is specified in revolutions per minute.
  Otherwise, subindex 1 contains the multiplier and subindex 2 the divisor of the fraction by which the speed values are multiplied in revolutions per second to calculate the desired user unit, see **User-defined units**. Object **2060**$_h$ is used to select whether the revolutions are electrical (**2060**$_h$ = 0) or mechanical (**2060**$_h$ = 1).
- **6042**$_h$: Target Velocity.
  The target speed is set here in user-defined units.
- **6048**$_h$: Velocity Acceleration
  This object defines the acceleration. Subindex 1 contains the change in speed, subindex 2 the corresponding time in seconds. Both together are used to calculate the acceleration:

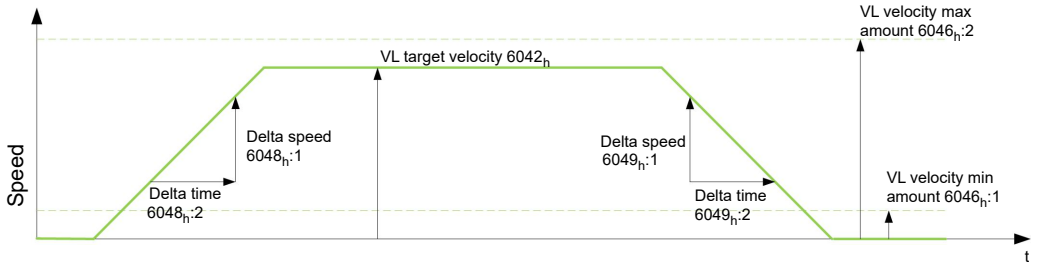
$$\text{VL velocity acceleration} = \frac{\text{Delta speed } (6048_h{:}1)}{\text{Delta time } (6048_h{:}2)}$$

- **6049**$_h$ (Velocity Deceleration):
  This object defines the deceleration (deceleration ramp). The subindices here are arranged as described in object **6048**$_h$; the change in speed is to be specified with positive sign.
- **6046**$_h$ (Velocity Min Max Amount):
  The limitations of the target speeds are specified in this object.
  The minimum speed is set in **6046**$_h$:1$_h$. If the target speed (**6042**$_h$) falls below the minimum speed, the value is limited to the minimum speed **6046**$_h$:1$_h$.
  The maximum speed is set in **6046**$_h$:2$_h$. If the target speed (**6042**$_h$) exceeds the maximum speed, the value is limited to the maximum speed **6046**$_h$:2$_h$.
- **604A**$_h$ (Velocity Quick Stop):
  This object can be used to set the quick-stop ramp. Subindices 1 and 2 are identical to those described for object **6048**$_h$.

The following objects can be used to check the function:

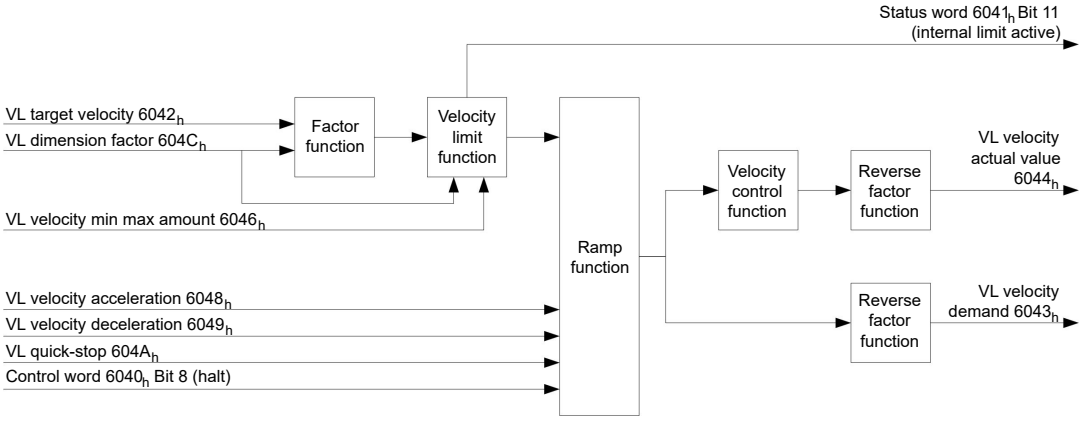
- **6043**$_h$ (Vl Velocity Demand)
- **6044**$_h$ (Vl Velocity Actual Value)

Speeds in Velocity Mode



Objects for Velocity Mode

The ramp generator follows the target speed, remaining within the set speed and acceleration limits. As long as a limit is active, bit 11 in object **6041**$_h$ is set (internal limit active).



6.3 Profile Velocity

6.3.1 Description

This mode operates the motor in Velocity Mode with extended (jerk-limited) ramps.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

6.3.2 Activation

To activate the mode, the value "3" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

6.3.3 Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 8 (Halt): If this bit is set to "1", the motor stops. On a transition from "1" to "0", the motor accelerates with the set start ramp to the target speed. On a transition from "0" to "1", the motor brakes and comes to a standstill.

6.3.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 10 (target speed reached; Target Reached): In combination with bit 8 in the controlword, this bit specifies whether the target speed is reached, if braking is taking place or if the motor is at a standstill (see table).

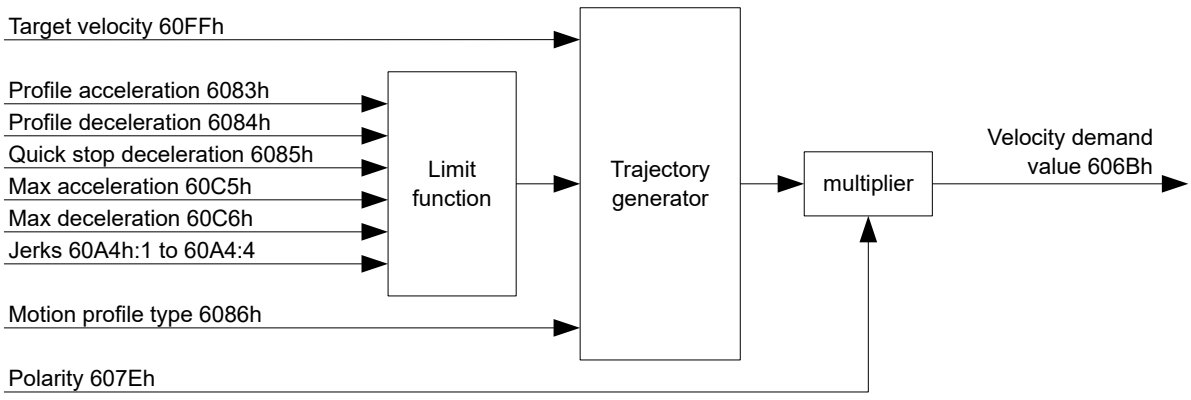
6041_h Bit 10	**6040_h Bit 8**	**Description**
0	0	Target speed not reached
0	1	Axis braking
1	0	Target speed within target window (defined in **$606D_h$**h and **$606E_h$**)
1	1	Axis speed is 0

6.3.5 Object entries

The following objects are necessary for controlling this mode:

- **$606B_h$** (Velocity Demand Value):
  This object contains the output of the ramp generator, which simultaneously serves as the preset value for the speed controller.
- **$606C_h$** (Velocity Actual Value):
  Indicates the current actual speed.
- **$606D_h$** (Velocity Window):
  This value specifies by how much the actual speed may vary from the set speed for bit 10 (target speed reached; Target Reached") in object **6041_h** (statusword) to be set to "1".
- **$606E_h$** (Velocity Window Time):
  This object specifies how long the actual speed and the set speed must be close to one another (see **$606D_h$** "Velocity Window") for bit 10 "Target speed reached" in object **6041_h** (statusword) to be set to "1".
- **$607E_h$** (Polarity):
  If bit 6 is set to "1" here, the sign of the target speed is reversed.
- **6083_h** (Profile acceleration):
  Sets the value for the acceleration ramp in Velocity Mode.
- **6084_h** (Profile Deceleration):
  Sets the value for the deceleration ramp in Velocity Mode.
- **6085_h** (Quick Stop Deceleration):
  Sets the value for the deceleration ramp for rapid braking in Velocity Mode.
- **6086_h** (Motion Profile Type):
  The ramp type can be selected here ("0" = trapezoidal ramp, "3" = jerk-limited ramp).
- **$60FF_h$** (Target Velocity):
  Specifies the target speed that is to be reached.

Objects in Profile Velocity Mode

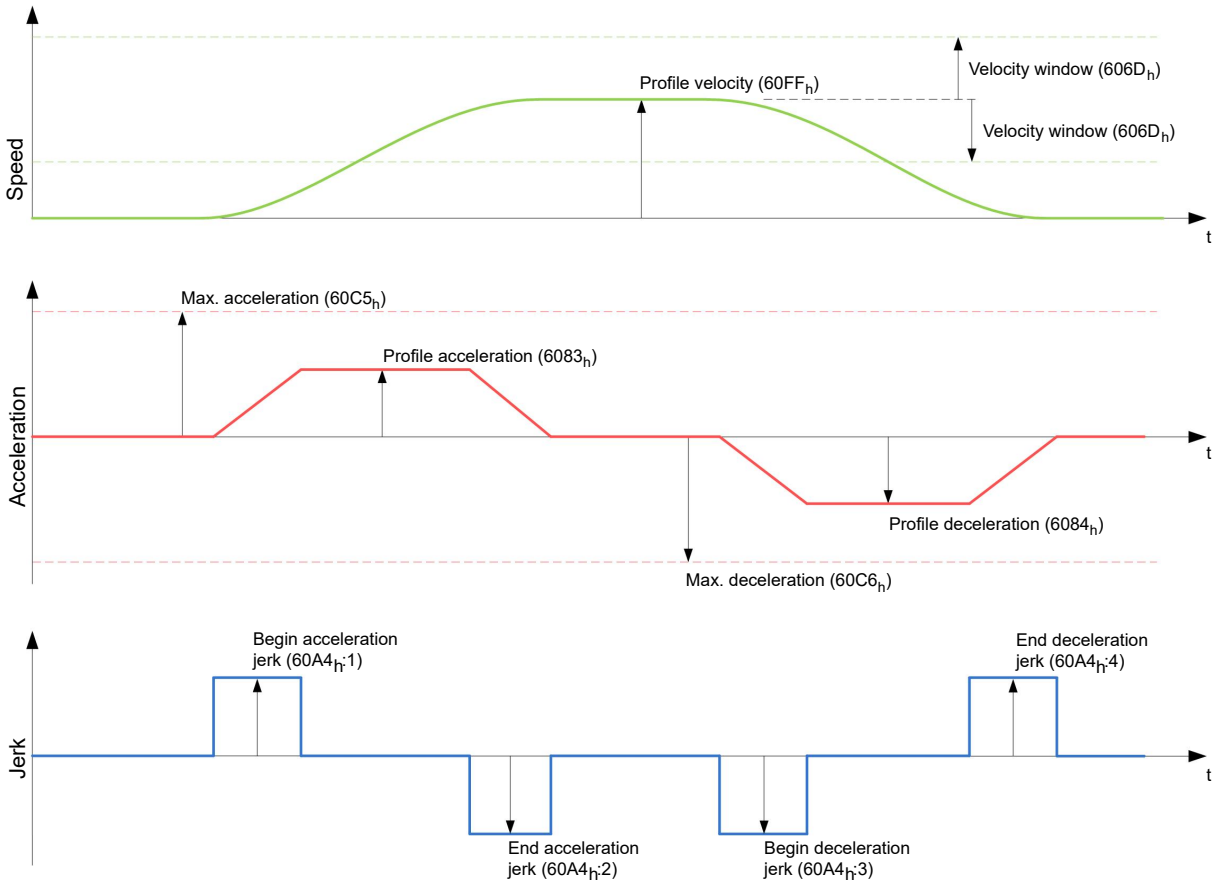


Activation

After the mode is selected in object **6060**$_h$ (Modes Of Operation) and the "Power State machine" (see "**CiA 402 Power State Machine**") is switched to *Operation enabled*, the motor is accelerated to the target speed in object **60FF**$_h$ (see following figures). The speed and acceleration values are taken into account here; for jerk-limited ramps, the jerk-limit values are also taken into account.

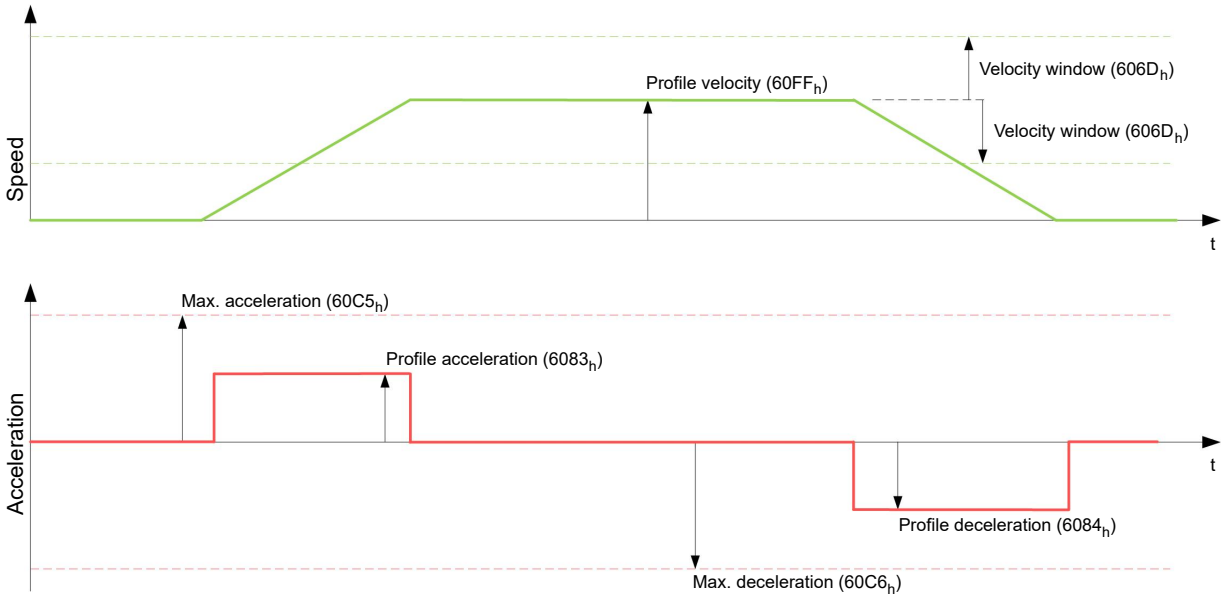
Limitations in the jerk-limited case

The following figure shows the adjustable limits in the jerk-limited case (**6086**$_h$ = 3).



Limitations in the trapezoidal case

This figure shows the adjustable limitations for the trapezoidal case (**6086**$_h$ = 0).



6.4 Profile Torque

6.4.1 Description

In this mode, the torque is preset as a set value and reached via a ramp function.

Note

This mode only functions if **closed loop** is activated, see also **Commissioning Closed Loop**.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

6.4.2 Activation

To activate the mode, the value "4" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

6.4.3 Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 8 (Halt): If this bit is set to "1", the motor stops. If this bit is set from "1" to "0", the motor is started up according to the presets. When setting from "0" to "1", the motor is again brought to a standstill, taking the preset values into consideration.

6.4.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 10 (Target Reached): In combination with bit 8 of object **6040**$_h$ (controlword), this bit indicates whether the specified torque is reached (see following table). The target is considered having been met if the current torque (**6077h Torque Actual Value**) is within a tolerance window (**203Dh Torque Window**) for a specified time (**203Eh Torque Window Time**).

6040_h Bit 8	6041_h Bit 10	Description
0	0	Specified torque not reached
0	1	Specified torque reached
1	0	Axis accelerated
1	1	Axis speed is 0

- Bit 11: Limit exceeded: The target torque (**6071_h**) exceeds the maximum torque entered in **6072_h**.

6.4.5 Object entries

All values of the following entries in the object dictionary are to be specified as a thousandth of the maximum torque, which corresponds to the rated current (**$203B_h$**:01_h) . This includes the objects:

- **6071_h** (Target Torque):
  Target torque
- **6072_h** (Max Torque):
  Maximum torque during the entire ramp (accelerate, maintain torque, decelerate)
- **6074_h** (Torque Demand):
  Current output value of the ramp generator (torque) for the controller
- **6087_h** (Torque Slope):
  Max. change in torque per second

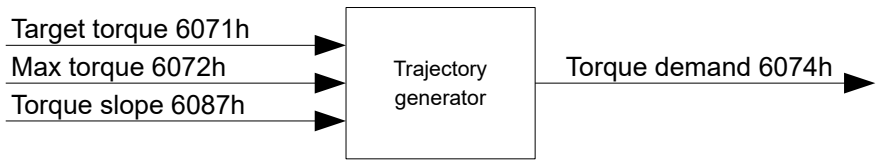
Note

These values are not limited to 100% of the rated current (**$203B_h$**:01_h). Torque values greater than the rated torque (generated from the rated current) can be achieved if the maximum duration of the peak current (**$203B_h$**:02_h) is set (see **I2t Motor overload protection**). All torque objects are limited by the peak current.

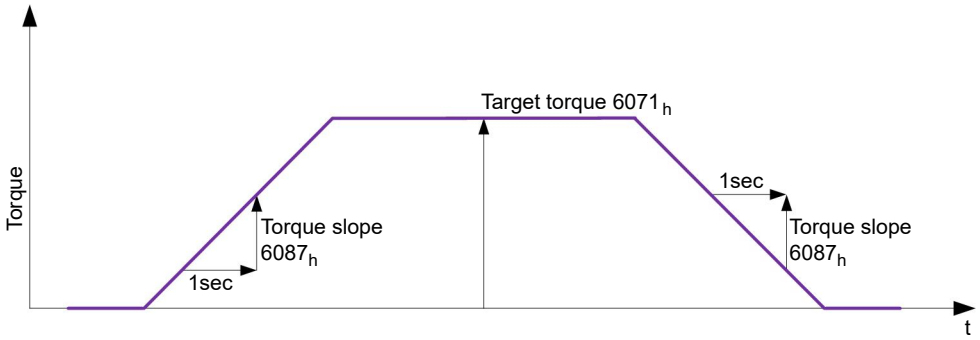
The following objects are also needed for this operating mode:

- **3202_h** Bit 5 (Motor Drive Submode Select):
  If this bit is set to "0", the drive controller is operated in the torque-limited Velocity Mode, i.e., the maximum speed can be limited in object **2032_h** and the controller can operate in field weakening mode.
  If this bit is set to "1", the controller operates in the ("Real") Torque Mode; the maximum speed cannot be limited here and field weakening mode is not possible.

Objects of the ramp generator



Torque curve



6.5 Homing

6.5.1 Overview

Description

The purpose of the homing method is to align the position zero point of the controller with an encoder index or position switch.

Activation

To activate the mode, the value "6" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

If home switches and/or limit switches are used, these special functions must first be activated in the I/O configuration (see "**Digital inputs and outputs**").

Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 4: If the bit is set to "1", referencing is started. This is performed until either the reference position is reached or bit 4 is reset to "0".

Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

Bit 13	Bit 12	Bit 10	Description
0	0	0	Homing is performed
0	0	1	Homing is interrupted or not started
0	1	0	Homing confirmed, but target not yet reached
0	1	1	Homing completed
1	0	0	Error during homing, motor still turning
1	0	1	Error during homing, motor at standstill

Object entries

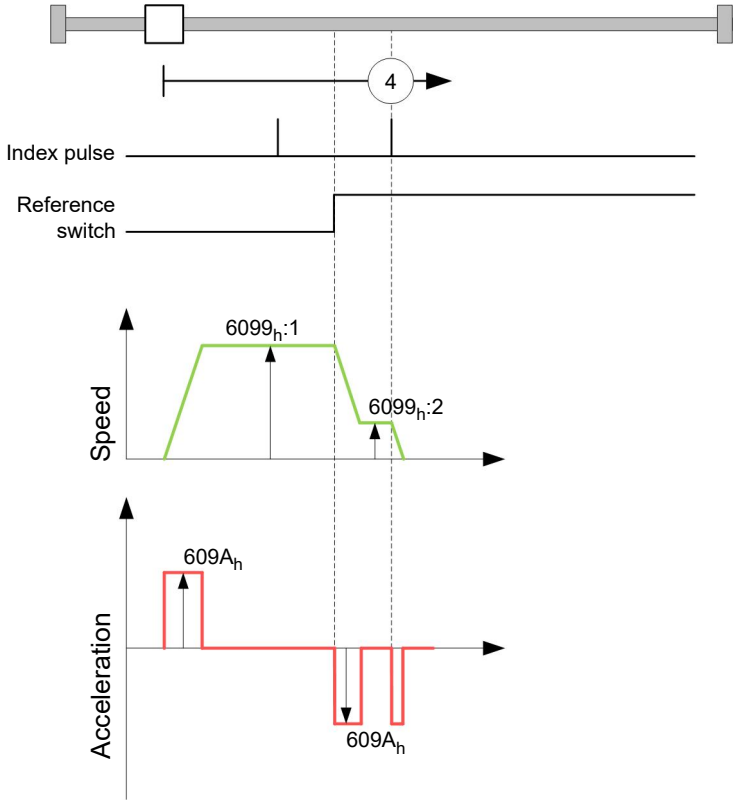
The following objects are necessary for controlling this mode:

- **607C**$_h$ (Home Offset): Specifies the difference between the zero position of the controller and the reference point of the machine in **user-defined units**.
- **6098**$_h$ (Homing Method):
  Method to be used for referencing (see "**Homing method**")

- **6099**$_h$:01$_h$ (Speed During Search For Switch):
  Speed for the search of the switch
- **6099**$_h$:02$_h$ (Speed During Search For Zero):
  Speed for the search of the index
- **609A**$_h$ (Homing Acceleration):
  Starting acceleration and braking deceleration for homing
- **2056**$_h$ (Limit Switch Tolerance Band):
  After reaching the positive or negative limit switch, the controller permits a tolerance range in which the motor can continue to run. If this tolerance range is exceeded, the motor stops and the controller switches to the "Fault" state. If limit switches can be actuated during homing, the tolerance range should be selected such that the motor does not exit the tolerance range during braking. Homing cannot otherwise be successfully performed. After homing is completed, the tolerance range can be reset to "0" if this is required by the application.
- **203A**$_h$:01$_h$ (Minimum Current For Block Detection):
  Minimum current threshold which, if exceeded, is to detect the blocking of the motor at a block.
- **203A**$_h$:02$_h$ (Period Of Blocking):
  Specifies the time in ms that the motor is to continue to run against the block after block detection.

Homing speeds

The figure shows the homing speeds using method 4 as an example:



6.5.2 Homing method

Description

The homing method is written as a number in object **6098**$_h$ and decides whether, on a switch edge (rising/falling), a current threshold for block detection or an index pulse is referenced or in which direction homing starts. Methods that use the index pulse of the encoder lie in the number range 1 to 14, 33 and 34. Methods that do not use the index pulse of the encoder lie between 17 and 30, but are identical to methods 1 to 14 with respect to the travel profiles. These number are shown in circles in the following figures. Methods for which no limit switches are used and, instead, travel against a block is to be detected, a minus must be placed before the method number when making the call.

In the following graphics, the negative movement direction is to the left. The *limit switch* is located before the respective mechanical block; the *home switch* is located between the two limit switches. The index pulses come from the connected encoder.

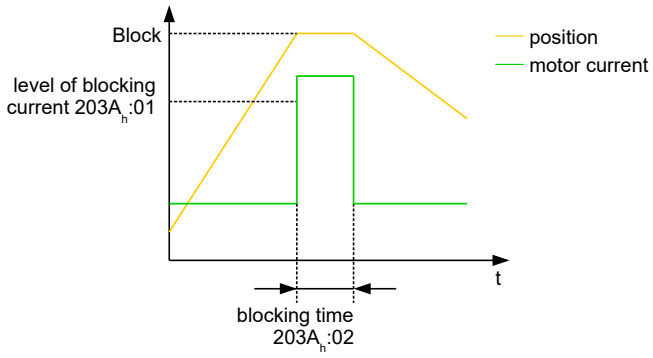
For methods that use homing on block, the same figures apply as for the methods with limit switch. Because nothing is different aside from the missing limit switches, the same figures are used. For the figures here, the limit switches must be replaced with a mechanical block.

Homing on block

Homing on block currently only functions in *closed loop* mode.

"Homing on block" functions like every homing method with the difference that instead of a limit switch, a block (limit stop) is used for positioning. Two settings are to be made here:

1. Current level: In object **203A$_h$**:01, the current level is defined above which movement against the block is detected.
2. Blocking duration: In object **203A$_h$**:02, the duration during which the motor moves against the block is set.



Overview of methods

Methods 1 to 14 as well as 33 and 34 use the index pulse of the encoder.

Methods 17 to 32 are identical to methods 1 to 14 with the difference that only limit or home switches are used for referencing and not the index pulse.

- Methods 1 to 14 use an index pulse.
- Methods 17 to 30 do not use an index pulse.
- Methods 33 and 34 reference only to the next index pulse.
- Method 35 references to the current position.

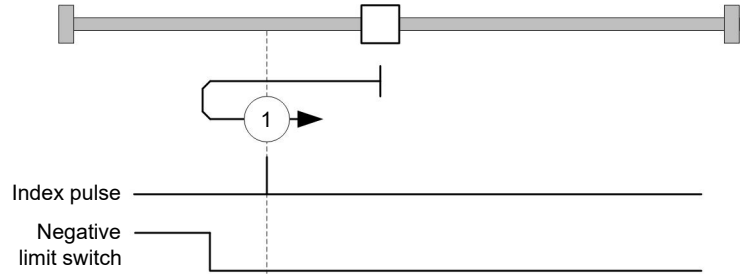
The following methods can be used for homing on block:

- Methods -1 to -2 and -7 to -14 contain an index pulse
- Methods -17 to -18 and -23 to -30 have no index pulse

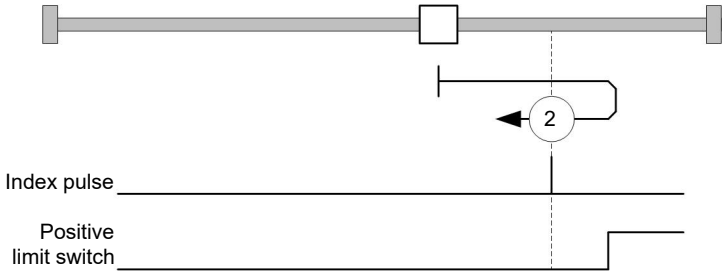
Methods 1 and 2

Reference to limit switches and index pulse.

Method 1 references to negative limit switch and index pulse:



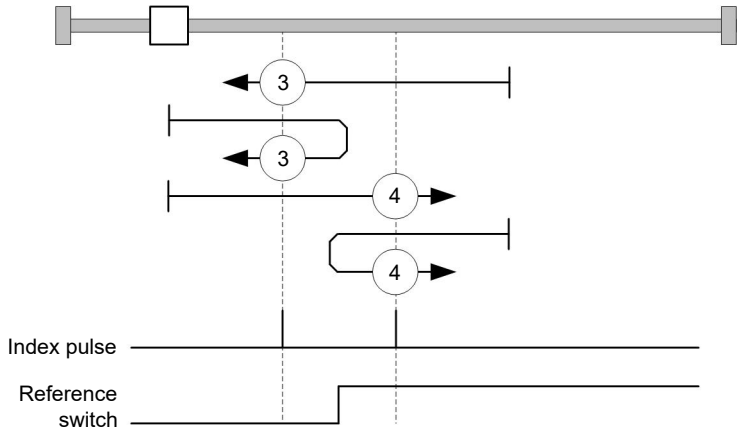
Method 2 references to positive limit switch and index pulse:



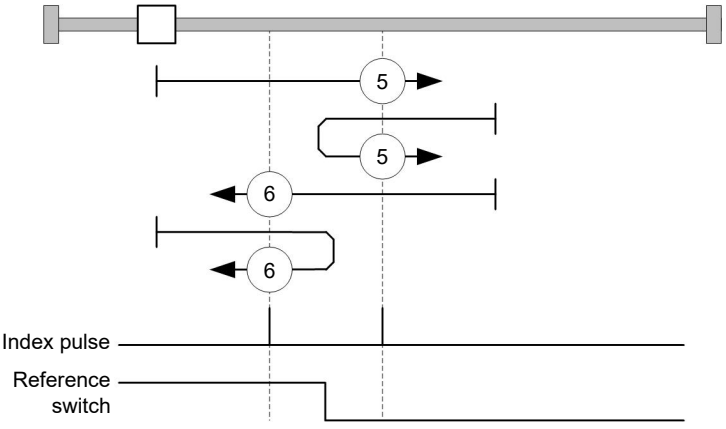
Methods 3 to 6

Reference to the switching edge of the home switch and index pulse.

With methods 3 and 4, the left switching edge of the home switch is used as reference:



With methods 5 and 6, the right switching edge of the home switch is used as reference:

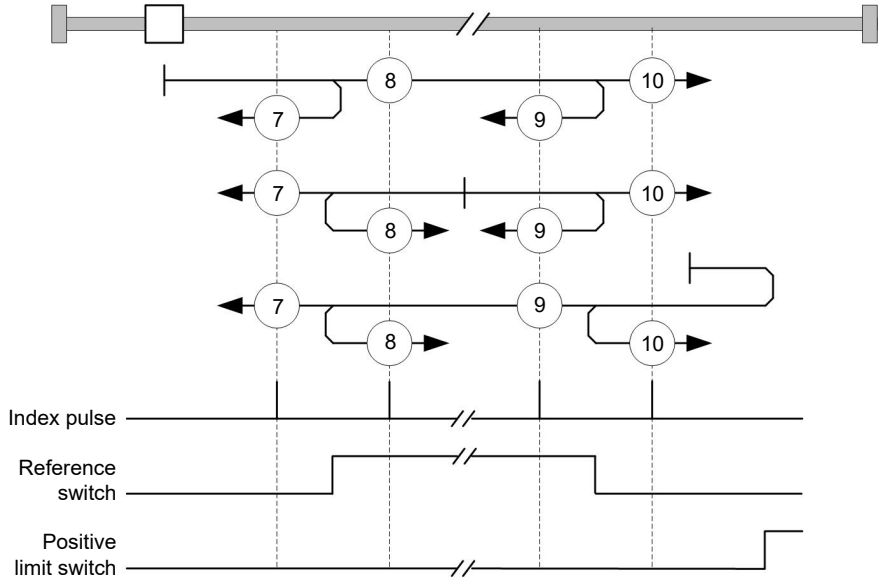


Methods 7 to 14

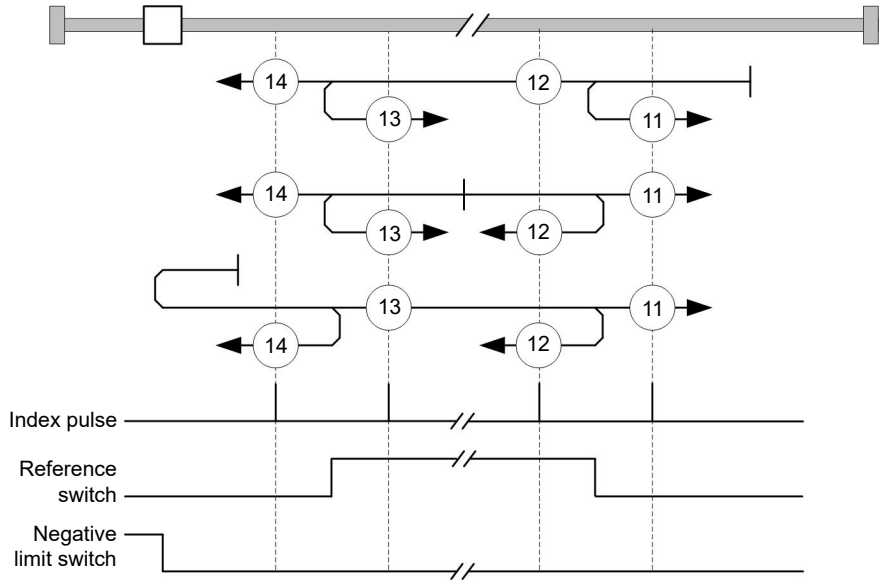
Reference to the home switch and index pulse (with limit switches).

With these methods, the current position relative to the home switch is not important. With method 10, for example, referencing is always performed to the index pulse to the right of the right edge of the home switch.

Methods 7 to 10 take the positive limit switch into account:



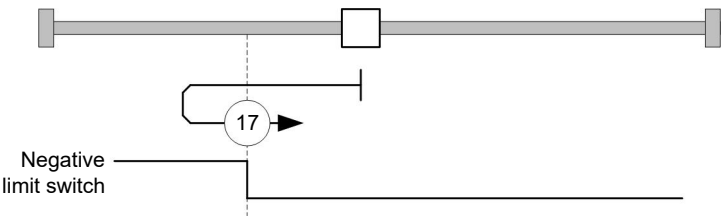
Methods 11 to 14 take the negative limit switch into account:



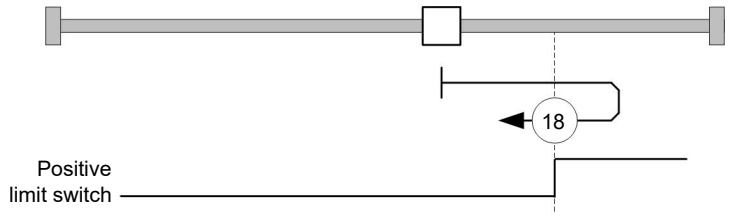
Methods 17 and 18

Reference to the limit switch without the index pulse.

Method 17 references to the negative limit switch:



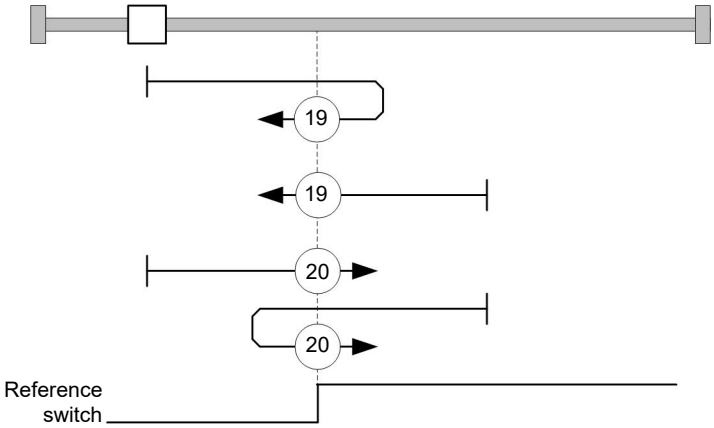
Method 18 references to the positive limit switch:



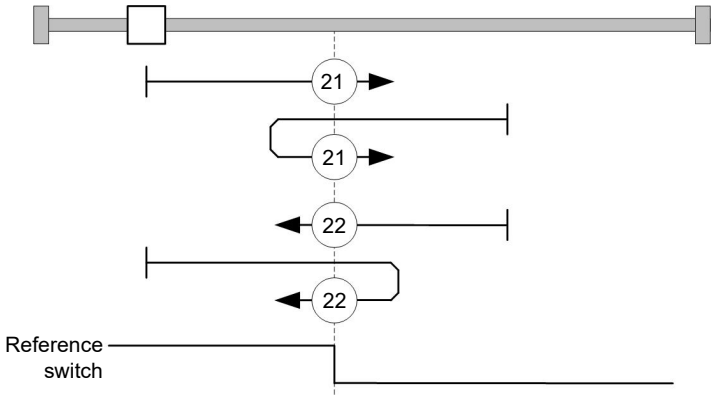
Methods 19 to 22

Reference to the switching edge of the home switch without the index pulse.

With methods 19 and 20 (equivalent to methods 3 and 4), the left switching edge of the home switch is used as reference:



With methods 21 and 22 (equivalent to methods 5 and 6), the right switching edge of the home switch is used as reference:

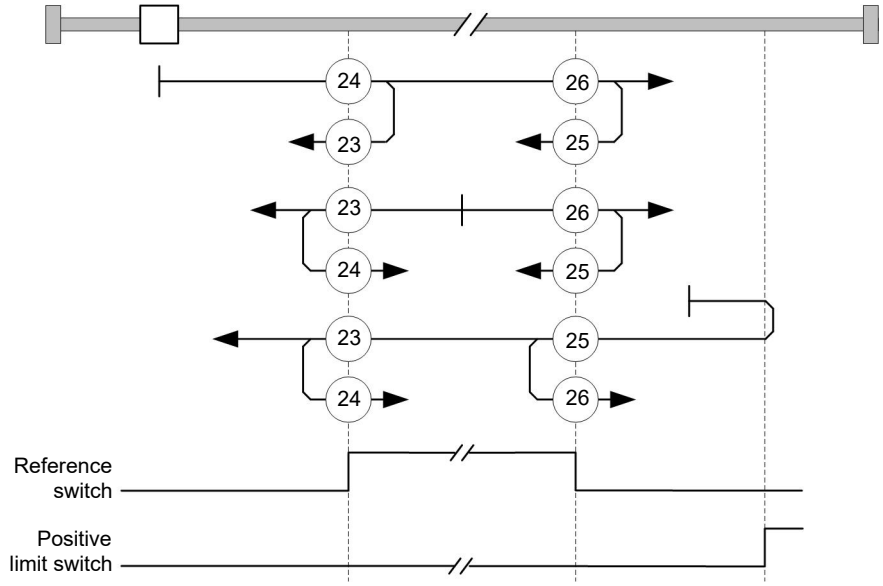


Methods 23 to 30

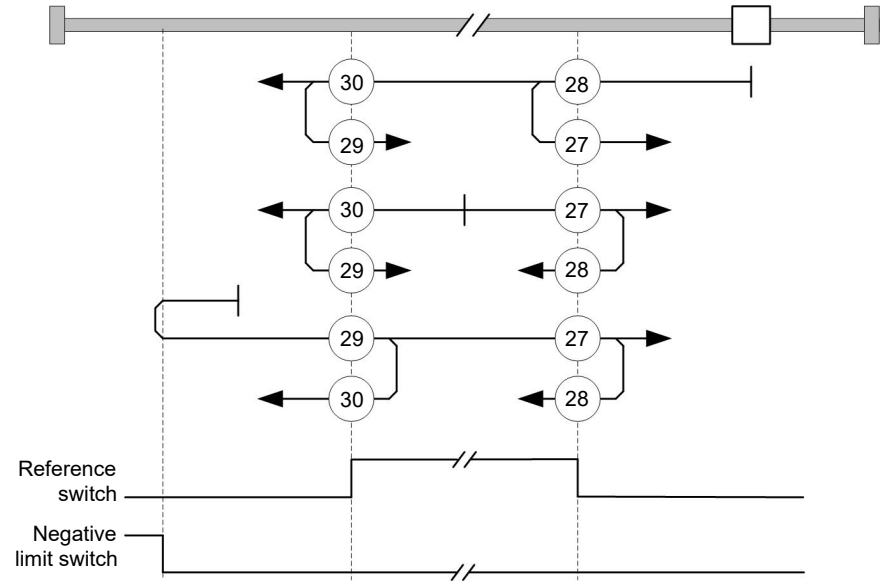
Reference to the home switch without the index pulse (with limit switches).

With these methods, the current position relative to the home switch is not important. With method 26, for example, referencing is always performed to the index pulse to the right of the right edge of the home switch.

Methods 23 to 26 take the positive home switch into account:



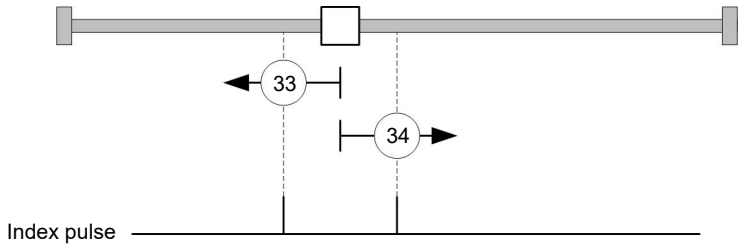
Methods 27 to 30 take the negative home switch into account:



Methods 33 and 34

Reference to the next index pulse.

With these methods referencing is only performed to the respective subsequent index pulse:



Method 35

References to the current position.

Note

For Homing Mode 35, it is not necessary to switch the **CiA 402 Power State Machine** to the "Operation enabled" state. When energizing the motor windings in *open loop* mode, it is thereby possible to prevent the current position from not being exactly 0 after Homing Mode 35.

6.6 Interpolated Position Mode

6.6.1 Overview

Description

Interpolated Position Mode is used to synchronize multiple axes. For this purpose, a higher-level controller performs the ramp and path calculation and passes the respective demand position, at which the axis is to be located at a certain time, to the controller. The controller interpolates between these intermediate position points.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Synchronization with the SYNC object

For Interpolated Position Mode, it is necessary that the controller synchronizes with the SYNC object (depending on the fieldbus). This SYNC object is to be sent by the higher-level controller in regular intervals. Synchronization occurs as soon as the controller is switched to the *Operational* NMT mode.

Note

Where possible, it is recommended that a time interval of the *SYNC object* be used.

6.6.2 Activation

To activate the mode, the value "7" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

6.6.3 Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 4 activates the interpolation when it is set to "1".
- Bit 8 (Halt): If this bit is set to "1", the motor stops. On a transition from "1" to "0", the motor accelerates with the set start ramp to the target speed. On a transition from "0" to "1", the motor brakes and comes to a standstill. The braking deceleration is dependent here on the setting of the "Halt Option Code" in object **605D**$_h$.

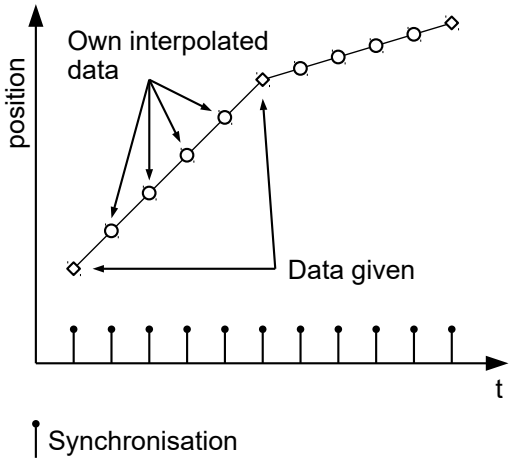
6.6.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 10: Target position reached: This bit is set to "1" if the target position was reached (if the halt bit in the controlword is "0") or the axis has speed 0 (if the halt bit in the last control word was "1").
- Bit 11: Limit exceeded: The demand position is above or below the limit values set in **607D**$_h$.
- Bit 12 (IP mode active): This bit is set to "1" if interpolation is active.

6.6.5 Use

The controller follows a linearly interpolated path between the current position and the preset target position. The (next) target position must be written in record **60C1**$_h$:01$_h$.



In the current implementation, only

- linear interpolation
- and a target position

are supported.

6.6.6 Setup

The following setup is necessary:

- **60C2**$_h$:01$_h$: Time between two passed target positions in ms.
- **60C4**$_h$:06$_h$: This object is to be set to "1" to be able to modify the target position in object **60C1**$_h$:01$_h$.
- To be able to turn the motor, the *power state machine* is to be set to the *Operation enabled* state (see **CiA 402 Power State Machine**)

6.6.7 Operation

After setting up, the task of the higher-level controller is to write the target positions to object **60C1**$_h$:01$_h$ in time.

6.7 Cyclic Synchronous Position

6.7.1 Overview

Description

In this mode, the controller receives an absolute position preset via the fieldbus at fixed time intervals (referred to in the following as a *cycle*). The controller then no longer calculates any ramps, but rather only follows the presets.

The target position is transferred cyclically (via *PDO*). Bit 4 in the controlword does not need to be set (unlike the **Profile Position** mode).

Note

The target is absolute and, thus, independent of how often it was sent per *cycle*.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Activation

To activate the mode, the value "8" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

Controlword

In this mode, the bits of controlword **6040**$_h$ have no special function.

Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

Bit	Value	Description
8	0	The controller is not in sync with the fieldbus
8	1	The controller is in sync with the fieldbus
10	0	Reserved
10	1	Reserved
12	0	Controller does not follow the target; the preset of **607A**$_h$ (Target Position) is ignored
12	1	Controller follows the target; object **607A**$_h$ (Target Position) is used as the input for position control.
13	0	No following error
13	1	Following error

Bit 11: Limit exceeded: The demand position is above or below the limit values set in **607D**$_h$.

6.7.2 Object entries

The following objects are necessary for controlling this mode:

- **607A**$_h$ (Target Position): This object must be written cyclically with the position set value.
- **607B**$_h$ (Position Range Limit): This object contains the preset for an overrun or underrun of the position specification.
- **607D**$_h$ (Software Position Limit): This object defines the limitations within which the position specification (607A$_h$) must be located.
- **6065**$_h$ (Following Error Window): This object specifies a tolerance corridor in both the positive and negative direction from the set specification. If the actual position is outside of this corridor for longer than the specified time (**6066**$_h$), a following error is reported.
- **6066**$_h$ (Following Error Time Out): This object specifies the time range in milliseconds. If the actual position is outside of the position corridor (**6065**$_h$) for longer than this time range, a following error is triggered.
- **6085**$_h$ (Quick-Stop Deceleration): This object contains the braking deceleration for the case that a quick-stop is triggered.
- **605A**$_h$ (Quick-Stop Option Code): This object contains the option that is to be executed in the event of a quick-stop.
- **6086**$_h$ (Motion Profile Type):
- **60C2**$_h$:01$_h$ (Interpolation Time Period): This object specifies the time of a *cycle*; a new set value must be written in **607A**$_h$ in these time intervals.

The following applies here: cycle time = value of **60C2**$_h$:01$_h$ * 10$^{\text{value of 60C2:02}}$ seconds.

- **60C2**$_h$:02$_h$ (Interpolation Time Index): This object specifies the time basis of the cycles. Currently, only value **60C2**$_h$:02$_h$=-3 is supported; this yields a time basis of 1 millisecond.

The following objects can be read in this mode:

- **6064**$_h$ (Position Actual Value)
- **606C**$_h$ (Velocity Actual Value)
- **60F4**$_h$ (Following Error Actual Value)

6.8 Cyclic Synchronous Velocity

6.8.1 Overview

Description

In this mode, the controller passes a speed preset via the fieldbus at fixed time intervals (referred to in the following as a *cycle*). The controller then no longer calculates any ramps, but rather only follows the presets.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Activation

To activate the mode, the value "9" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

Controlword

In this mode, the bits of controlword **6040**$_h$ have no special function.

Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

Bit	Value	Description
8	0	The controller is not in sync with the fieldbus
8	1	The controller is in sync with the fieldbus
10	0	Reserved
10	1	Reserved
12	0	Controller does not follow the target; the preset of **60FF**$_h$ (Target Velocity) is ignored
12	1	Controller follows the target; object **60FF**$_h$ (Target Velocity) is used as the input for position control.
13	0	Reserved
13	1	Reserved

6.8.2 Object entries

The following objects are necessary for controlling this mode:

- **60FF**$_h$ (Target Velocity): This object must be written cyclically with the speed set value.

- **6085**$_h$ (Quick-Stop Deceleration): This object contains the braking deceleration for the case that a quick-stop is triggered (see "**CiA 402 Power State Machine**").
- **605A**$_h$ (Quick-Stop Option Code): This object contains the option that is to be executed in the event of a quick-stop (see "**CiA 402 Power State Machine**").
- **60C2**$_h$:01$_h$ (Interpolation Time Period): This object specifies the time of a *cycle*; a new set value must be written in **60FF**$_h$ in these time intervals.
  The following applies here: cycle time = value of **60C2**$_h$:01$_h$ * $10^{\text{value of 60C2:02}}$ seconds.
- **60C2**$_h$:02$_h$ (Interpolation Time Index): This object specifies the time basis of the cycles. Currently, only value **60C2**$_h$:02$_h$=-3 is supported; this yields a time basis of 1 millisecond.

The following objects can be read in this mode:

- **606C**$_h$ (Velocity Actual Value)
- **607E**$_h$ (Polarity)

6.9 Cyclic Synchronous Torque

6.9.1 Overview

Description

In this mode, the controller passes an absolute torque preset via the fieldbus at fixed time intervals (referred to in the following as a *cycle*). The controller then no longer calculates any ramps, but rather only follows the presets.

Note

This mode only functions if **closed loop** is activated, see also **Commissioning closed loop**.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

Activation

To activate the mode, the value "10" must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

Controlword

In this mode, the bits of controlword **6040**$_h$ have no special function.

Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

Bit	Value	Description
8	0	The controller is not in sync with the fieldbus
8	1	The controller is in sync with the fieldbus
10	0	Reserved
10	1	Reserved
12	0	Controller does not follow the target; the preset of **6071**$_h$ (Target Torque) is ignored

Bit	Value	Description
12	1	Controller follows the target; object **6071**$_h$ (Target Torque) is used as the input for position control.
13	0	Reserved
13	1	Reserved

6.9.2 Object entries

The following objects are necessary for controlling this mode:

- **6071**$_h$ (Target Torque): This object must be written cyclically with the torque set value and is to be set relative to **6072**$_h$.
- **6072**$_h$ (Max Torque): Describes the maximum permissible torque.
- **60C2**$_h$:01$_h$ (Interpolation Time Period): This object specifies the time of a *cycle*; a new set value must be written in **6071**$_h$ in these time intervals.
  The following applies here: cycle time = value of **60C2**$_h$:01$_h$ * $10^{\text{value of 60C2:02}}$ seconds.
- **60C2**$_h$:02$_h$ (Interpolation Time Index): This object specifies the time basis of the cycles. Currently, only value **60C2**$_h$:02$_h$=-3 is supported; this yields a time basis of 1 millisecond.

The following objects can be read in this mode:

- **606C**$_h$ (Velocity Actual Value)

6.10 Clock-direction mode

6.10.1 Description

In clock-direction mode, the motor is operated via two inputs by a higher-level positioning controller with clock and direction signal. On each clock signal, the motor moves one step in the direction corresponding to the direction signal.

Note

The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

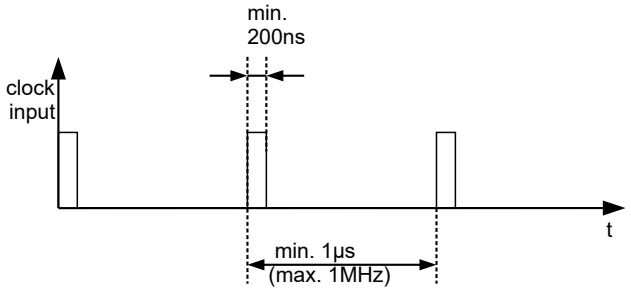
6.10.2 Activation

To activate the mode, the value "-1" (or "FFh") must be set in object **6060**$_h$ (Modes Of Operation) (see "**CiA 402 Power State Machine**").

6.10.3 General

The following data apply for every subtype of the clock-direction mode:

- The maximum frequency of the input pulse is 1 MHz; the ON pulse should not be less than 200 ns.



- The steps are scaled using objects **2057**$_h$ and **2058**$_h$. The following formula applies here:

$$\text{step width per pulse} = \frac{2057_h}{2058_h}$$

The "step size per pulse" value is set to 128 (**2057**$_h$=128 and **2058**$_h$=1) ex works, which corresponds to a quarter step per pulse. A full step is the value "512", a half step per pulse corresponds to "256", etc.

Note

For a stepper motor with 50 pole pairs, 200 full steps correspond to one mechanical revolution of the motor shaft.
In *clock-direction mode*, the BLDC motors are also handled as stepper motors by the controller. This means that for a BLDC motor with, e.g., 3 pole pairs, 12 (=4*3) full steps correspond to one revolution.

Note

If there is a change of direction, a time of at least 35 µs must elapse before the new clock signal is applied.

6.10.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

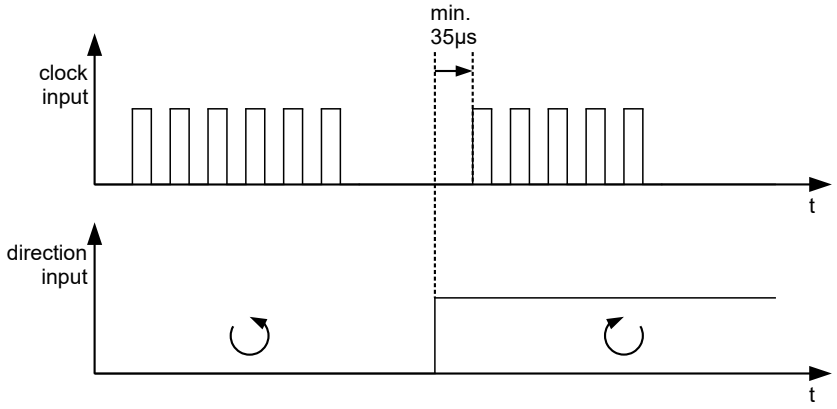
- Bit 13 (Following Error): This bit is set in *closed loop* mode if the following error is greater than the set limits (**6065**$_h$ (Following Error Window) and **6066**$_h$ (Following Error Time Out)).

6.10.5 Subtypes of the clock-direction mode

Clock-direction mode (TR mode)

To activate the mode, object **205B**$_h$ must be set to the value "0" (factory settings).

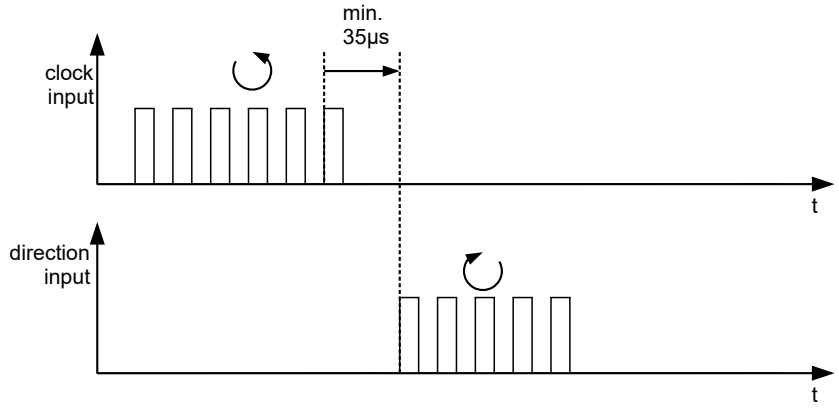
In this mode, the pulses must be preset via the clock input; the signal of the direction input specifies the direction of rotation here (see following graphic).



Right / left rotation mode (CW / CCW mode)

To activate the mode, object **205B**$_h$ must be set to the value "1".

In this mode, the input that is used decides the direction of rotation (see following graphic).



6.11 Auto setup

6.11.1 Description

To determine a number of parameters related to the motor and the connected sensors (encoders/Hall sensors), an auto setup is performed. **Closed Loop** operation requires a successfully completed *auto setup*. *Auto setup* is only to be performed once during commissioning as long as the motor connected to the controller is not changed. For details, see the corresponding section in chapter **Commissioning**.

Note
The limit switches and, thus, the tolerance bands are active in this mode. For further information on the limit switches, see **Limitation of the range of motion**.

6.11.2 Activation

To activate the mode, the value "-2" (="FE$_h$") must be set in object **6060**$_h$ (Modes Of Operation) (see **CiA 402 Power State Machine**).

6.11.3 Controlword

The following bits in object **6040**$_h$ (controlword) have a special function:

- Bit 4 starts a travel command. This is carried out on a transition from "0" to "1".

6.11.4 Statusword

The following bits in object **6041**$_h$ (statusword) have a special function:

- Bit 10: Indexed: indicates whether (= "1") or not (= "0") an encoder index was found.
- Bit 12: Aligned: this bit is set to "1" after *auto setup* has concluded

7 Special functions

7.1 Digital inputs and outputs

This controller is equipped with digital inputs and outputs.

7.1.1 Bit assignment

The software of the controller assigns each input and output two bits in the respective object (e.g., **60FDh Digital Inputs** or **60FEh Digital Outputs**):

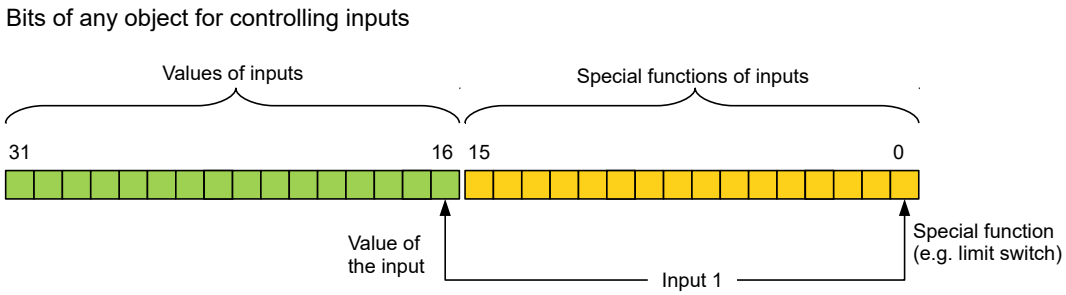
1. The first bit corresponds to the special function of an output or input. These functions are always available on bits 0 to 15 (inclusive) of the respective object. These include the limit switches and the home switch for the digital inputs and the brake control for the outputs.
2. The second bit shows the output/input as a level; these are then available on bits 16 to 31.

Example

To manipulate the value of output 2, always use bit 17 in **60FE**$_h$.

To activate the "negative limit switch" special function of input 1, set bit 0 in **3240**$_h$:01$_h$; to query the status of the input, read bit 0 in **60FD**$_h$. Bit 16 in **60FD**$_h$ also shows the status of input 1 (independent of whether or not the special function of the input was activated).

This assignment is graphically illustrated in the following drawing.



7.1.2 Digital inputs

Overview

Note

For digital inputs with 5 V, the length of the supply lines must not exceed 3 meters.

Note

The digital inputs are sampled once per millisecond. Signal changes at the input less than one millisecond in duration are not processed.

The following inputs are available:

Input	Special function	Switching threshold switchable	Differential / single-ended
1	Negative limit switch	no, 5 V	single-ended
2	Positive limit switch	no, 5 V	single-ended
3	Home switch	no, 5 V	single-ended
4	None	no, 5 V	single-ended
5	None	no, 5 V	single-ended

Object entries

The value of an input can be manipulated using the following OD settings, whereby only the corresponding bit acts on the input here.

- **3240**$_h$:01$_h$ (Special Function Enable): This bit allows special functions of an input to be switched off (value "0") or on (value "1"). If input 1 is not used as, e.g., a negative limit switch, the special function must be switched off to prevent an erroneous response to the signal generator. The object has no effect on bits 16 to 31.
  The firmware evaluates the following bits:
  - Bit 0: Negative limit switch
  - Bit 1: Positive limit switch
  - Bit 2: Home switch

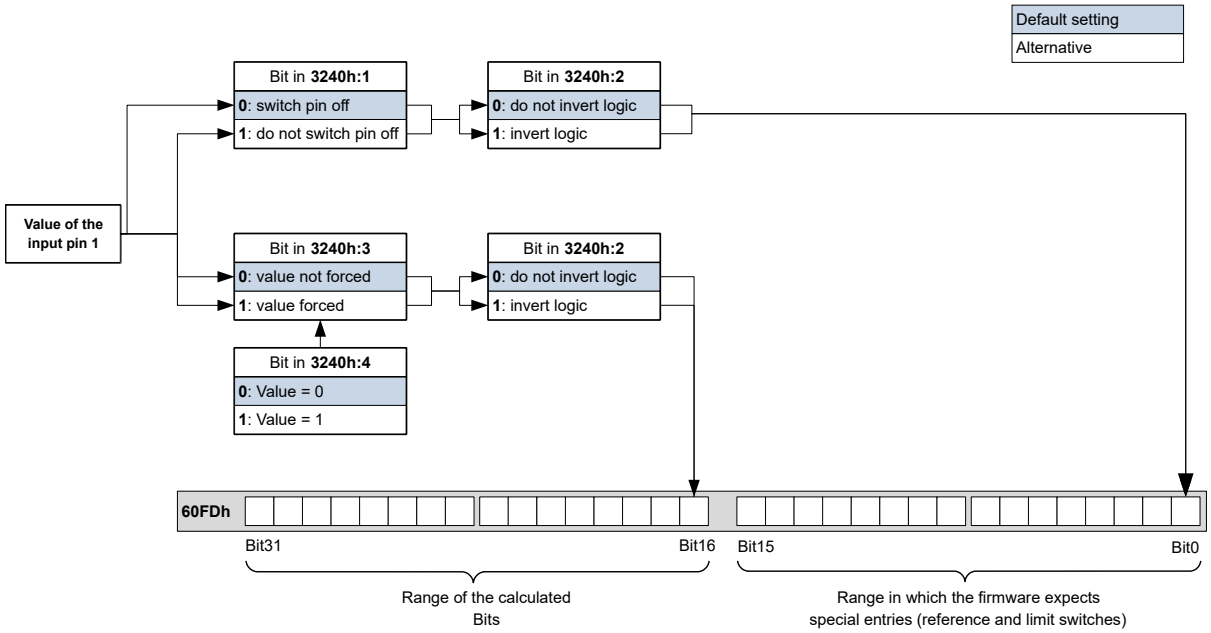
  If, for example, two limit switches and one home switch are used, bits 0–2 in **3240**$_h$:01$_h$ must be set to "1".

- **3240**$_h$:02$_h$ (Function Inverted): This bit switches from normally open logic (a logical high level at the input yields the value "1" in object **60FD**$_h$) to normally closed logic (the logical high level at the input yields the value "0"). This applies for the special functions (except for the clock and direction inputs) and for the normal inputs. If the bit has the value "0", normally open logic applies; for the value "1", normally closed logic applies.
- **3240**$_h$:03$_h$ (Force Enable): This bit switches on the software simulation of input values if it is set to "1". In this case, the actual values are no longer used in object **3240**$_h$:04$_h$, but rather the set values for the respective input.
- **3240**$_h$:04$_h$ (Force Value): This bit specifies the value that is to be read as the input value if the same bit was set in object **3240**$_h$:03$_h$.
- **3240**$_h$:05$_h$ (Raw Value): This object contains the unmodified input value.
- **60FD**$_h$ (Digital Inputs): This object contains a summary of the inputs and the special functions.

Computation of the inputs

Computation of the input signal using the example of input 1:

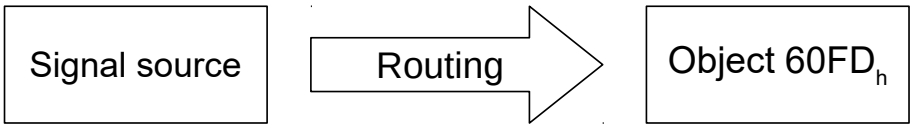
The value at bit 0 of object **60FD**$_h$ is interpreted by the firmware as negative limit switch; the result of the complete computation is stored in bit 16.



Input Routing

Principle

To perform the assignment of the inputs more flexibly, there is a mode called *Input Routing Mode*. This assigns a signal of a source to a bit in object **60FD**$_h$.



Activation

This mode is activated by setting object **3240**$_h$:08$_h$ (Routing Enable) to 1.

Note

Entries **3240**$_h$:01$_h$ to **3240**:04$_h$ then have **no** function until Input Routing is again switched off.

Note

If *Input Routing* is switched on, the initial values of **3242**$_h$ are changed and correspond to the function of the input as it was before activation of *Input Routing*. The inputs of the controller behave the same with activation of *Input Routing*. Therefore, you should not switch back and forth between the normal mode and *Input Routing*.

Routing

Object 3242$_h$ determines which signal source is routed to which bit of **60FD**$_h$. Subindex 01$_h$ of 3242$_h$ determines bit 0, subindex 02$_h$ determines bit 1, and so forth. You can find the signal sources and their numbers in the following lists.

Number		
dec	**hex**	**Signal source**
00	00	Signal is always 0
01	01	Physical input 1
02	02	Physical input 2
03	03	Physical input 3
04	04	Physical input 4
05	05	Physical input 5
06	06	Physical input 6
07	07	Physical input 7
08	08	Physical input 8
09	09	Physical input 9
10	0A	Physical input 10
11	0B	Physical input 11
12	0C	Physical input 12
13	0D	Physical input 13
14	0E	Physical input 14
15	0F	Physical input 15
16	10	Physical input 16
65	41	Hall input "U"
66	42	Hall input "V"
67	43	Hall input "W"
68	44	Encoder input "A"
69	45	Encoder input "B"
70	46	Encoder input "Index"
71	47	USB Power Signal

The following table describes the inverted signals of the previous table.

Number		
dec	**hex**	**Signal source**
128	80	Signal is always 1
129	81	Inverted physical input 1
130	82	Inverted physical input 2
131	83	Inverted physical input 3
132	84	Inverted physical input 4
133	85	Inverted physical input 5
134	86	Inverted physical input 6
135	87	Inverted physical input 7
136	88	Inverted physical input 8
137	89	Inverted physical input 9
138	8A	Inverted physical input 10
139	8B	Inverted physical input 11
140	8C	Inverted physical input 12
141	8D	Inverted physical input 13
142	8E	Inverted physical input 14

Number		
dec	**hex**	**Signal source**
143	8F	Inverted physical input 15
144	90	Inverted physical input 16
193	C1	Inverted Hall input "U"
194	C2	Inverted Hall input "V"
195	C3	Inverted Hall input "W"
196	C4	Inverted encoder input "A"
197	C5	Inverted encoder input "B"
198	C6	Inverted encoder input "Index"
199	C7	Inverted USB power signal

Example

Input 1 is to be routed to bit 16 of object **60FD**$_h$:

The number of the signal source for input 1 is "1". The routing for bit 16 is written in 3242_h:11_h.

Hence, object 3242_h:11_h must be set to the value "1".

7.1.3 Digital outputs

Outputs

The outputs are controlled via object **60FE**$_h$. Here, output 1 corresponds to bit 16 in object **60FE**$_h$, output 2 corresponds to bit 17, etc., as with the inputs. The outputs with special functions are again entered in the firmware in the lower bits 0 to 15. The only bit assigned at the present time is bit 0, which controls the motor brake.

Wiring

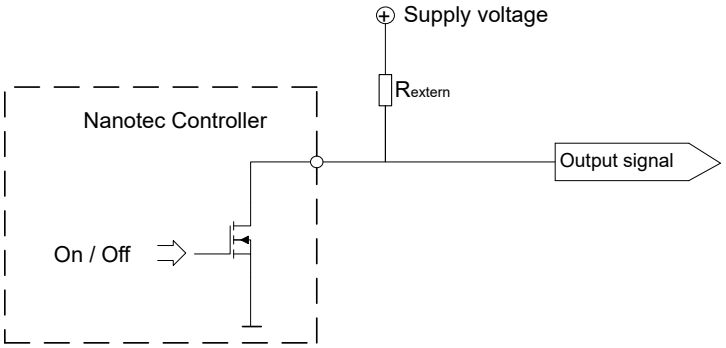
Note

Always observe the maximum capacity of the output (see **Pin assignment**).

The outputs are implemented as *open drain*. Hence, an external voltage supply is always necessary.

Example

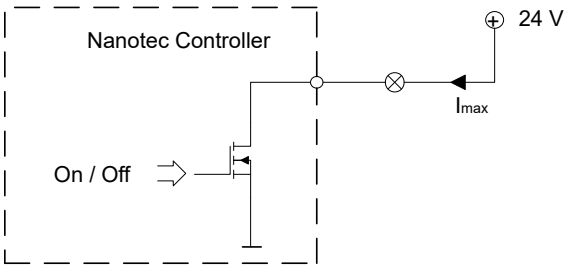
The digital output signal should continue to be used. For this purpose, a circuit as shown in the following figure is to be realized.



With a supply voltage of +24 V, a resistance value $R_{external}$ of 10 kΩ is recommended.

Example

A simple load is to be used with the digital output.



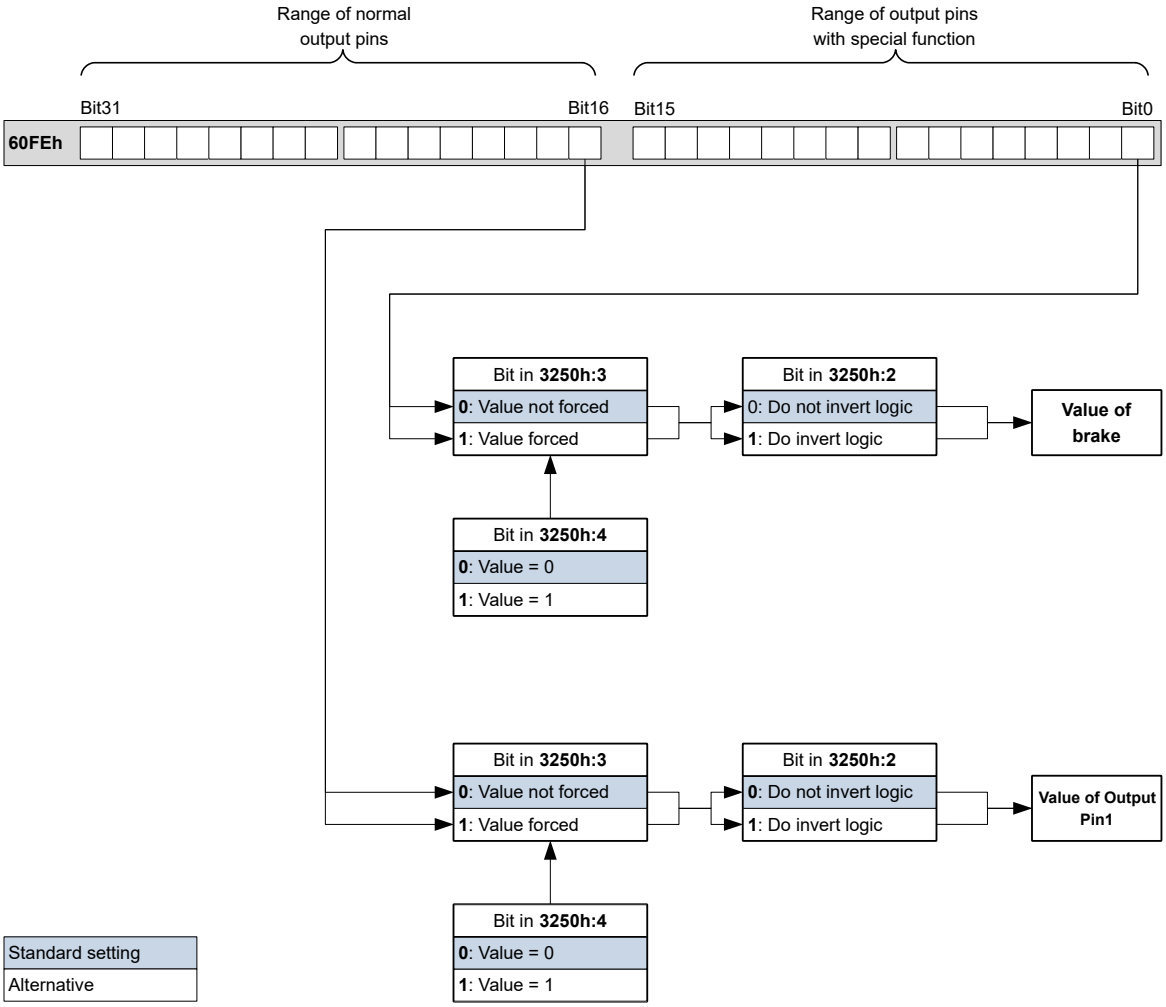
Object entries

Additional OD entries are available for manipulating the value of the outputs (see the following example for further information). As with the inputs, only the bit at the corresponding location acts on the respective output:

- **3250**$_h$:01$_h$: No function.
- **3250**$_h$:02$_h$: This is used to switch the logic from *normally open* to *normally closed*. Configured as *normally open*, the input outputs a logical high level if the bit is "1". With the *normally closed* configuration, a logical low level is output accordingly for a "1" in object **60FE**$_h$.
- **3250**$_h$:03$_h$: If a bit is set here, the output is controlled manually. The value for the output is then in object **3250**$_h$:4$_h$; this is also possible for the brake output.
- **3250**$_h$:04$_h$: The bits in this object specify the output value that is to be applied at the output if manual control of the output is activated by means of object **3250**$_h$:03$_h$.
- **3250**$_h$:05$_h$: This object has no function and is included for reasons of compatibility.

Computation of the outputs

Example for calculating the bits of the outputs:

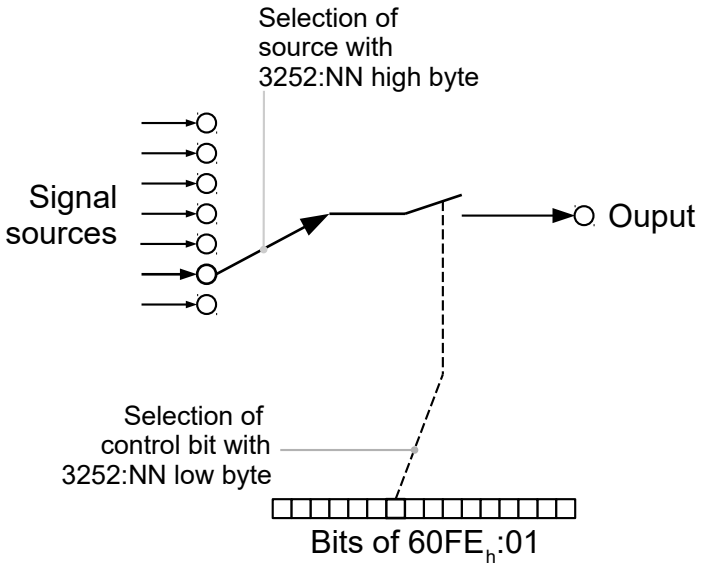


Output Routing

Principle

The "Output Routing Mode" assigns an output a signal source; a control bit in object **60FE**$_h$:01$_h$ switches the signal on or off.

The source is selected with **3252**$_h$:01 to 05 in the "high byte" (bit 15 to bit 8). The assignment of a control bit from object **60FE**$_h$:01$_h$ is performed in the "low byte" (bit 7 to bit 0) of **3252**$_h$:01$_h$ to 05 (see following figure).



Activation

This mode is activated by setting object **3250**$_h$:08$_h$ (Routing Enable) to 1.

Note

Entries **3250**$_h$:01$_h$ to **3250**:04$_h$ then have **no** function until "Output Routing" is switched off again.

Routing

The subindex of object **3252**$_h$ determines which signal source is routed to which output. The output assignments are listed in the following:

Subindex 3252$_h$	Output Pin
01$_h$	Configuration of the brake output (if available)
02$_h$	Configuration of output 1
03$_h$	Configuration of output 2 (if available)
04$_h$	Configuration of output 3 (if available)
05$_h$	Configuration of output 4 (if available)

Note

The maximum output frequency of the brake output, output 1 and output 2 is 10 kHz. All other outputs can only produce signals up to 500 Hz.

Subindices **3252**$_h$:01$_h$ to 05$_h$ are 16 bits wide, whereby the high byte selects the signal source (e.g., the PWM generator) and the low byte determines the control bit in object **60FE**$_h$:01.

Bit 7 of **3252**$_h$:01$_h$ to 05 inverts the controller from object **60FE**$_h$:01. Normally, value "1" in object **60FE**$_h$:01 switches on the signal; if bit 7 is set, the value "0" switches on the signal.

Number in 3252:01 to 05	
00XX$_h$	Output is always "1"

Number in 3252:01 to 05	
$01XX_h$	Output is always "0"
$02XX_h$	Encoder signal (**6063_h**) with frequency divider 1
$03XX_h$	Encoder signal (**6063_h**) with frequency divider 2
$04XX_h$	Encoder signal (**6063_h**) with frequency divider 4
$05XX_h$	Encoder signal (**6063_h**) with frequency divider 8
$06XX_h$	Encoder signal (**6063_h**) with frequency divider 16
$07XX_h$	Encoder signal (**6063_h**) with frequency divider 32
$08XX_h$	Encoder signal (**6063_h**) with frequency divider 64
$09XX_h$	Position Actual Value (**6064_h**) with frequency divider 1
$0AXX_h$	Position Actual Value (**6064_h**) with frequency divider 2
$0BXX_h$	Position Actual Value (**6064_h**) with frequency divider 4
$0CXX_h$	Position Actual Value (**6064_h**) with frequency divider 8
$0DXX_h$	Position Actual Value (**6064_h**) with frequency divider 16
$0EXX_h$	Position Actual Value (**6064_h**) with frequency divider 32
$0FXX_h$	Position Actual Value (**6064_h**) with frequency divider 64
$10XX_h$	PWM signal that is configured with object **2038_h**:05_h and 06_h
$11XX_h$	Inverted PWM signal that is configured with object **2038_h**:05_h and 06_h

Example

The encoder signal (**6063_h**) is to be applied to output 1 with a frequency divider 4. The output is to be controlled with bit 5 of object **60FE**:01.

- **3250_h**:08_h = 1 (activate routing)
- **3252_h**:02_h = 0405_h ($04XX_h$ + 0005_h) Dabei ist:
- $04XX_h$: Encoder signal with frequency divider 4
- 0005_h: Selection of bit 5 of **60FE**:01

The output is switched on by setting bit 5 in object **60FE**:01.

Example

The PWM signal is to be applied to output 2. Bit 0 of **60FE**:01_h should be used as control bit.

- **3250_h**:08_h = 1 (activate routing)
- **3252_h**:03_h = 1080_h (=$10XX_h$ + 0080_h). Where:
  - $10XX_h$: PWM signal
  - 0080_h: Selection of the inverted bit 0 of object **60FE**:01

7.2 I^2t Motor overload protection

7.2.1 Description

Note

For stepper motors, only the rated current is specified, not a maximum current. No liability is therefore assumed when using I^2t with stepper motors.

The goal of I^2t motor overload protection is to protect the motor from damage and, at the same time, operate it normally up to its thermal limit.

This function is only available if the controller is in the **closed loop mode** (bit 0 of object **3202**$_h$ must be set to "1").

There is an exception: If I^2t is activated in *open loop* mode, the current is limited to the set rated current, even if the set maximum current is larger. This function was implemented for safety reasons so that one can switch from *closed loop* mode with very high, brief maximum current to *open loop* mode without damaging the motor.

7.2.2 Object entries

The following objects affect I^2t motor overload protection:

- **2031**$_h$: Peak Current – specifies the maximum current in mA.
- **203B**$_h$:1$_h$ Nominal Current – specifies the rated current in mA.
- **203B**$_h$:2$_h$ Maximum Duration Of Peak Current – specifies the maximum duration of the maximum current in ms.

The following objects indicate the current state of I^2t:

- **203B**$_h$:3$_h$ Threshold – specifies the limit in mAs that determines whether the maximum current or rated current is switched to.
- **203B**$_h$:4$_h$ CalcValue – specifies the calculated value that is compared with the threshold for setting the current.
- **203B**$_h$:5$_h$ LimitedCurrent – shows the momentary current value that was set by I^2t.
- **203B**$_h$:6$_h$ Status:
  - Value = "0": I^2t deactivated
  - Value = "1": I^2t activated

7.2.3 Activation

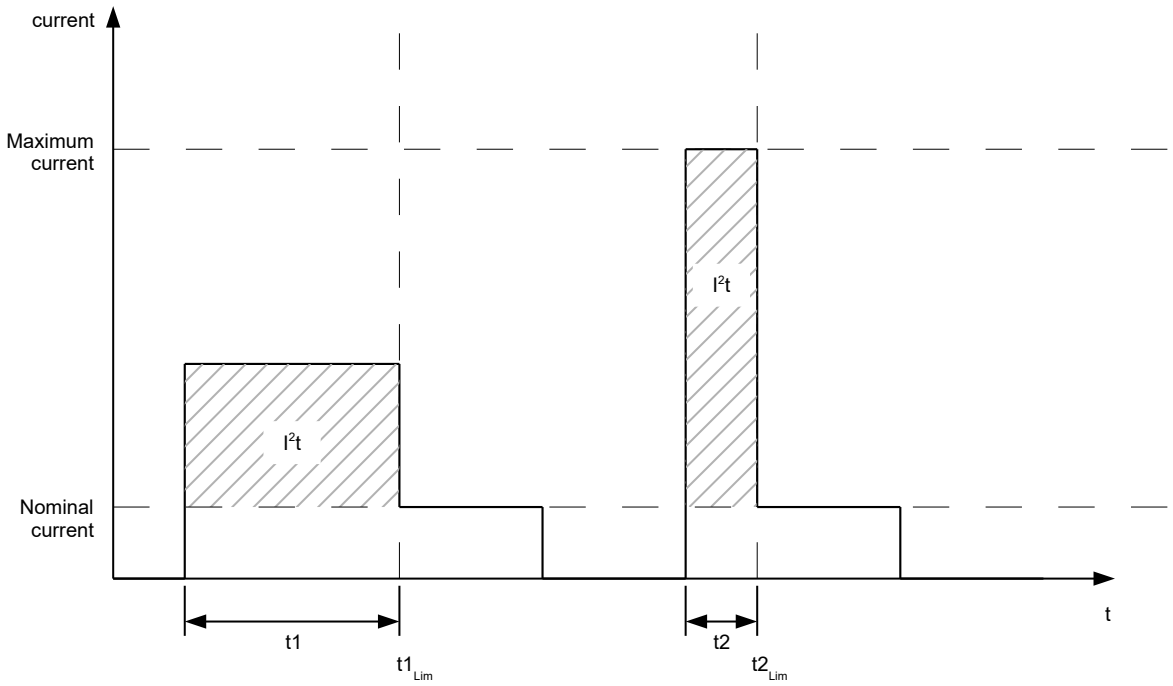
Closed loop must be activated, (bit 0 of object **3202**$_h$ set to "1", see also chapter **Closed Loop**). To activate the mode, the three object entries mentioned above (**2031**$_h$, **203B**$_h$:1,$_h$ **203B**$_h$:2$_h$) must have been appropriately specified. This means that the maximum current must be greater than the rated current and a time value for the maximum duration of the maximum current must be entered. If these conditions are not met, the I^2t functionality remains deactivated.

7.2.4 Function of I^2t

From the specification of rated current, maximum current and maximum duration of the maximum current, an I^2t_{Lim} is calculated.

The motor can run with maximum current until the calculated I^2t_{Lim} is reached. The current is then immediately reduced to the rated current.

The relationships are illustrated again in the following diagram.



In the first section, t1, the current value is higher than the rated current. At time $t1_{Lim}$, I^2t_{Lim} is reached and the current is limited to the rated current. A current that corresponds to the maximum current then occurs for a period of time t2. Hence, the value for I^2t_{Lim} is reached more quickly than in time t1.

7.3 Saving objects

Note

Improper use of the function can result in it no longer being possible to start the controller. Therefore, carefully read the entire chapter before using the function.

Note

As an alternative, objects can also be set and saved using the configuration file. Note that this file has higher priority. Objects that are saved both with the mechanism described here as well as in the configuration file take the value of the configuration file.

7.3.1 General

Many objects in the object dictionary can be saved and then automatically reloaded the next time the controller is switched on or reset. Furthermore, the saved values are also retained following a firmware update.

Only entire collections of objects (referred to in the following as *categories*) can be saved together; individual objects cannot be saved.

An object can be assigned one of the following *categories*:

- Communication: Parameters related to external interfaces, such as PDO configuration etc.
- Application: Parameters related to operating modes.
- Customer: Parameters that are written and read by the customer/user only and are ignored by the controller firmware.
- Drive: Parameters related to the motor and the sensors (BLDC/Stepper, *Closed/Open Loop*…). Some are set and saved by auto setup.

- Tuning: Parameters related to motor and encoder that are set either by auto setup or that can be found in the data sheets, e.g., pole pairs and maximum current.

If an object is not assigned one of these *categories*, it cannot be saved, e.g., statusword and all objects whose value is dependent on the current state of the controller.

The objects in each *category* are listed below. In chapter **Description of the object dictionary**, the corresponding *category* for each object is also specified.

7.3.2 Category: communication

- **1005**$_h$: COB-ID Sync
- **1007**$_h$: Synchronous Window Length
- **100C**$_h$: Guard Time
- **100D**$_h$: Live Time Factor
- **1014**$_h$: COB-ID EMCY
- **1017**$_h$: Producer Heartbeat Time
- **1029**$_h$: Error Behavior
- **1400**$_h$: Receive PDO 1 Communication Parameter
- **1401**$_h$: Receive PDO 2 Communication Parameter
- **1402**$_h$: Receive PDO 3 Communication Parameter
- **1403**$_h$: Receive PDO 4 Communication Parameter
- **1404**$_h$: Receive PDO 5 Communication Parameter
- **1405**$_h$: Receive PDO 6 Communication Parameter
- **1406**$_h$: Receive PDO 7 Communication Parameter
- **1407**$_h$: Receive PDO 8 Communication Parameter
- **1600**$_h$: Receive PDO 1 Mapping Parameter
- **1601**$_h$: Receive PDO 2 Mapping Parameter
- **1602**$_h$: Receive PDO 3 Mapping Parameter
- **1603**$_h$: Receive PDO 4 Mapping Parameter
- **1604**$_h$: Receive PDO 5 Mapping Parameter
- **1605**$_h$: Receive PDO 6 Mapping Parameter
- **1606**$_h$: Receive PDO 7 Mapping Parameter
- **1607**$_h$: Receive PDO 8 Mapping Parameter
- **1800**$_h$: Transmit PDO 1 Communication Parameter
- **1801**$_h$: Transmit PDO 2 Communication Parameter
- **1802**$_h$: Transmit PDO 3 Communication Parameter
- **1803**$_h$: Transmit PDO 4 Communication Parameter
- **1804**$_h$: Transmit PDO 5 Communication Parameter
- **1805**$_h$: Transmit PDO 6 Communication Parameter
- **1806**$_h$: Transmit PDO 7 Communication Parameter
- **1807**$_h$: Transmit PDO 8 Communication Parameter
- **1A00**$_h$: Transmit PDO 1 Mapping Parameter
- **1A01**$_h$: Transmit PDO 2 Mapping Parameter
- **1A02**$_h$: Transmit PDO 3 Mapping Parameter
- **1A03**$_h$: Transmit PDO 4 Mapping Parameter
- **1A04**$_h$: Transmit PDO 5 Mapping Parameter
- **1A05**$_h$: Transmit PDO 6 Mapping Parameter
- **1A06**$_h$: Transmit PDO 7 Mapping Parameter
- **1A07**$_h$: Transmit PDO 8 Mapping Parameter
- **2005**$_h$: CANopen Baudrate
- **2007**$_h$: CANopen Config
- **2009**$_h$: CANopen NodeID
- **2028**$_h$: MODBUS Slave Address
- **202A**$_h$: MODBUS RTU Baudrate

- **202D**$_h$: MODBUS RTU Parity
- **2102**$_h$: Fieldbus Module Control
- **3502**$_h$: MODBUS Rx PDO Mapping
- **3602**$_h$: MODBUS Tx PDO Mapping

7.3.3 Category: application

- **2033**$_h$: Plunger Block
- **2034**$_h$: Upper Voltage Warning Level
- **2035**$_h$: Lower Voltage Warning Level
- **2036**$_h$: Open Loop Current Reduction Idle Time
- **2037**$_h$: Open Loop Current Reduction Value/factor
- **2038**$_h$: Brake Controller Timing
- **203A**$_h$: Homing On Block Configuration
- **203D**$_h$: Torque Window
- **203E**$_h$: Torque Window Time
- **2056**$_h$: Limit Switch Tolerance Band
- **2057**$_h$: Clock Direction Multiplier
- **2058**$_h$: Clock Direction Divider
- **205B**$_h$: Clock Direction Or Clockwise/Counter Clockwise Mode
- **2060**$_h$: Compensate Polepair Count
- **2061**$_h$: Velocity Numerator
- **2062**$_h$: Velocity Denominator
- **2063**$_h$: Acceleration Numerator
- **2064**$_h$: Acceleration Denominator
- **2065**$_h$: Jerk Numerator
- **2066**$_h$: Jerk Denominator
- **2084**$_h$: Bootup Delay
- **2300**$_h$: NanoJ Control
- **2410**$_h$: NanoJ Init Parameters
- **2800**$_h$: Bootloader And Reboot Settings
- **320A**$_h$: Motor Drive Sensor Display Open Loop
- **320B**$_h$: Motor Drive Sensor Display Closed Loop
- **3210**$_h$: Motor Drive Parameter Set
- **3212**$_h$: Motor Drive Flags
- **3221**$_h$: Analogue Inputs Control
- **3240**$_h$: Digital Inputs Control
- **3242**$_h$: Digital Input Routing
- **3250**$_h$: Digital Outputs Control
- **3252**$_h$: Digital Output Routing
- **3321**$_h$: Analogue Input Offset
- **3322**$_h$: Analogue Input Pre-scaling
- **3700**$_h$: Following Error Option Code
- **4013**$_h$: HW Configuration
- **6040**$_h$: Controlword
- **6042**$_h$: Vl Target Velocity
- **6046**$_h$: Vl Velocity Min Max Amount
- **6048**$_h$: Vl Velocity Acceleration
- **6049**$_h$: Vl Velocity Deceleration
- **604A**$_h$: Vl Velocity Quick Stop
- **604C**$_h$: Vl Dimension Factor
- **605A**$_h$: Quick Stop Option Code
- **605B**$_h$: Shutdown Option Code

- **605C**$_h$: Disable Option Code
- **605D**$_h$: Halt Option Code
- **605E**$_h$: Fault Option Code
- **6060**$_h$: Modes Of Operation
- **6065**$_h$: Following Error Window
- **6066**$_h$: Following Error Time Out
- **6067**$_h$: Position Window
- **6068**$_h$: Position Window Time
- **606D**$_h$: Velocity Window
- **606E**$_h$: Velocity Window Time
- **6071**$_h$: Target Torque
- **6072**$_h$: Max Torque
- **607A**$_h$: Target Position
- **607B**$_h$: Position Range Limit
- **607C**$_h$: Home Offset
- **607D**$_h$: Software Position Limit
- **607E**$_h$: Polarity
- **6081**$_h$: Profile Velocity
- **6082**$_h$: End Velocity
- **6083**$_h$: Profile Acceleration
- **6084**$_h$: Profile Deceleration
- **6085**$_h$: Quick Stop Deceleration
- **6086**$_h$: Motion Profile Type
- **6087**$_h$: Torque Slope
- **608F**$_h$: Position Encoder Resolution
- **6091**$_h$: Gear Ratio
- **6092**$_h$: Feed Constant
- **6098**$_h$: Homing Method
- **6099**$_h$: Homing Speed
- **609A**$_h$: Homing Acceleration
- **60A4**$_h$: Profile Jerk
- **60C1**$_h$: Interpolation Data Record
- **60C2**$_h$: Interpolation Time Period
- **60C4**$_h$: Interpolation Data Configuration
- **60C5**$_h$: Max Acceleration
- **60C6**$_h$: Max Deceleration
- **60F2**$_h$: Positioning Option Code
- **60FE**$_h$: Digital Outputs
- **60FF**$_h$: Target Velocity

7.3.4 Category: drive

- **3202**$_h$: Motor Drive Submode Select

7.3.5 Category: tuning

- **2030**$_h$: Pole Pair Count
- **2031**$_h$: Maximum Current
- **2032**$_h$: Maximum Speed
- **203B**$_h$: I2t Parameters
- **2050**$_h$: Encoder Alignment
- **2051**$_h$: Encoder Optimization
- **2052**$_h$: Encoder Resolution
- **2059**$_h$: Encoder Configuration

7.3.6 Starting the save process

CAUTION

The motor must be at a standstill during the save process and may not be started up while saving.

Note

- The fieldbus function may be affected while saving.
- Saving may take a few seconds. Under no circumstances may you interrupt the voltage supply while saving. The state of the saved objects is otherwise undefined.
- Always wait until the controller has signaled that the save process has been successfully completed with the value "1" in the corresponding subindex in object **1010**$_h$.

There is a subindex in object **1010**$_h$ for each *category*. To save all objects of this *category*, the value "65766173$_h$" must be written in the subindex. [1] The controller signals the end of the save process by overwriting the value with a "1".

The following table shows which subindex of object **1010**$_h$ is responsible for which *category*.

Subindex	Category
01$_h$	All categories
02$_h$	Communication
03$_h$	Application
04$_h$	Customer
05$_h$	Drive
06$_h$	Tuning

7.3.7 Discarding the saved data

If all objects or one *category* of saved objects is to be deleted, value "64616F6C$_h$" must be written in object **1011**$_h$. [2] The following subindices correspond to a *category* here:

Subindex	Category
01$_h$	All categories (reset to factory settings) with the exception of category 06$_h$ (Tuning)
02$_h$	Communication
03$_h$	Application
04$_h$	Customer
05$_h$	Drive
06$_h$	Tuning

The saved objects are subsequently discarded. After the data have been deleted, the controller automatically restarts.

[1] This corresponds to the decimal of 1702257011$_d$ or the ASCII string `save`.
[2] This corresponds to the decimal of 1684107116$_d$ or the ASCII string `load`.

Note

Objects of *category* 06_h (Tuning) are determined by **Auto setup** and are not reset when resetting to factory settings with subindex 01_h (thereby making it unnecessary to again perform an auto setup). You can reset these objects with subindex 06_h.

7.3.8 Verifying the configuration

Object **1020**$_h$ can be used to verify the configuration. It acts as a modification marker similar to common text editors: as soon as a file is modified in the editor, a marker (usually an asterisk) is added.

The entries of object **1020**$_h$ can be written with a date and time and then saved together with all other savable objects with **1010**$_h$:01.

The entries of **1020**$_h$ are reset to "0" as soon as a savable object (including **1010**$_h$:0x,$_h$ except for **1010**$_h$:01_h and **1020**$_h$) is written.

The following sequence makes verification possible:

1. An external tool or master configures the controller.
2. The tool or master sets the value in object **1020**$_h$.
3. The tool or master activates the saving of all objects **1010**$_h$:01_h = 65766173_h. The date and time in object **1020**$_h$ are also saved.

After the controller is restarted, the master can check the value in **1020**$_h$:01_h and **1020**:01_h. If one of the values is "0", the object dictionary was changed after the saved values were loaded. If the date or time in **1020** does not correspond to the expected value, objects were probably saved with values other than those that were expected.

8 CANopen

The controller can be addressed by means of CANopen and can function in a network as a *slave*. In this chapter, the services of the CANopen communication structure are described. In addition, the CANopen messages are individually broken down.

CANopen references: **www.can-cia.org**

- *CiA 301 CANopen application layer and communication profile - Application layer and communication profile*, Date: 21.02.2011, Version: 4.2.0
- *CiA 402 Device profile for drives and motion control - Part 1: General definitions*, Date: 14.12.2007, Version: 3.0.0
- *CiA 402 Drives and mot ion cont rol device profile - Part 2: Operation modes and application data*, Data 14.12.2007, Version: 3.0
- *CiA 402 Drives and motion control device profile - Part 3: PDO mapping*, Data 14.12.2007, Version: 3.0
- *CiA 306 Electronic device description - Part 1: Electronic Data Sheet and Device Configuration File*, Data: 08.02.2012, Version: 1.3.5

8.1 General

Tip

- Only 11-bit CAN-IDs are currently supported.
- With CANopen, the data are always sent over the bus in little-endian format.

8.1.1 CAN message

CAN messages are described often in this chapter; these are written in the following format:

```
583 | 41 09 10 00 1E 00 00 00
```

```
183R | DLC=0
```

Where the following convention is used:

- All numbers are written in hexadecimal notation; due to the abbreviated notation, the leading `0x` is omitted.
- Normal data message: The CAN-ID of the CAN message is prefixed with `583` (i.e., 583_h or 1411_d) in the above example. The data and the CAN-ID are separated from the data with a pipe character.
- RTR message (remote transmission request): If an `R` follows the CAN-ID instead of the data, the length of the *DLC* (Download Content) is specified; in the above example, the length of the *DLC* is 0.

8.2 CANopen services

The CANopen stack offers the services listed in the following table; more detailed descriptions can be found in the respective chapters.

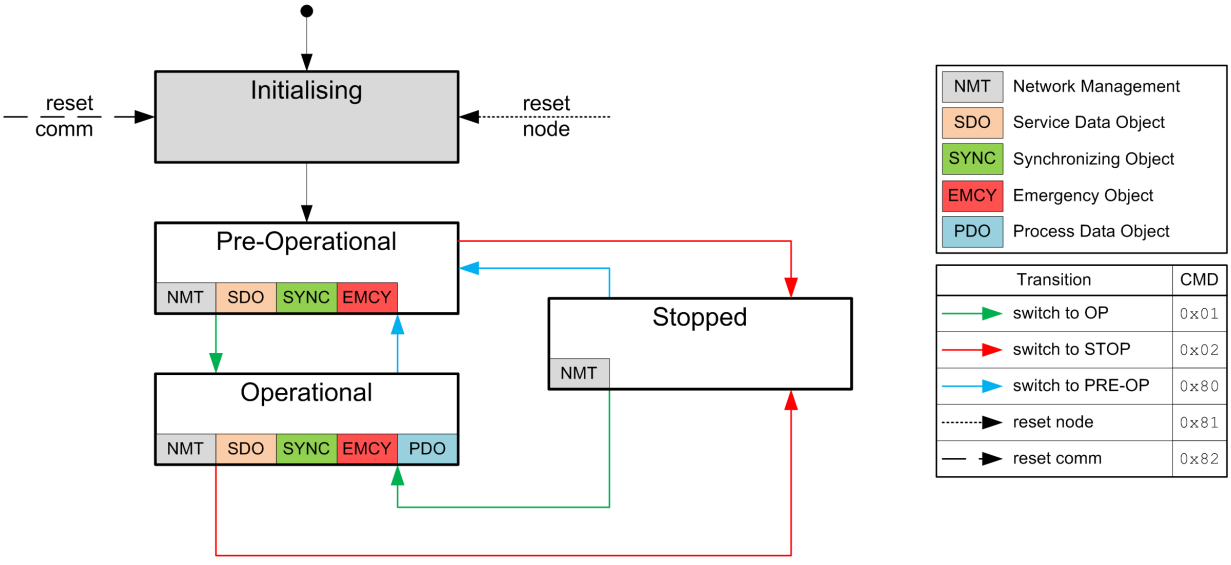
Default CAN-ID	Service	Description in
000_h	Network Management (NMT)	Section **Network Management (NMT)**
080_h	Synchronization Object	Section **Synchronization object (SYNC)**
080_h+Node-ID	Emergency	Section **Emergency Object (EMCY)**
180_h+Node-ID	TX Process Data Objects (PDO)	Section **Process Data Object (PDO)**
200_h+Node-ID	RX Process Data Objects (PDO)	
280_h+Node-ID	TX Process Data Objects (PDO)	

Default CAN-ID	Service	Description in
300_h+Node-ID	RX Process Data Objects (PDO)	
380_h+Node-ID	TX Process Data Objects (PDO)	
400_h+Node-ID	RX Process Data Objects (PDO)	
480_h+Node-ID	TX Process Data Objects (PDO)	
500_h+Node-ID	RX Process Data Objects (PDO)	
580_h+Node-ID	TX Service Data Objects (SDO)	Section **Service Data Object (SDO)**
600_h+Node-ID	RX Service Data Objects (SDO)	
700_h+Node-ID	BOOT-UP Protocol	Section **Boot-Up Protocol**
700_h+Node-ID	Nodeguarding and Heartbeat	Section **Heartbeat and nodeguarding**

8.2.1 Network Management (NMT)

Network Management is oriented towards CANopen devices and follows a master-slave structure. NMT requires a CANopen device in the network that performs the role of the CANopen master. All other devices have the role of the NMT slave. Each NMT slave can be addressed via its individual node-ID in the range from [1–127]. NMT services can be used to initiate, start, monitor, reset or stop CANopen devices.

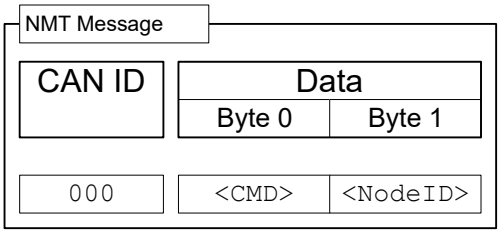
In doing so, the controller follows the state diagram shown in the following figure. The "Initialization" state is only reached after switching on or by sending a "Reset Communication" or "Reset Node" NMT command. The "Pre-Operational" state is automatically activated after initialization.



Shown in the following table is an overview of the activity of the services in the respective states. Note that the *Stopped* state stops communication completely and only permits controller of the NMT state machine.

Service	Initializing	Pre-Operational	Operational	Stopped
PDO			Active	
SDO		Active	Active	
SYNC		Active	Active	
EMCY		Active	Active	
BOOT-UP	Active			
NMT		Active	Active	Active

The "Network Management" message has CAN-ID 0. A message is always two bytes long and has the following structure:



Here, the <CMD> corresponds to one of the following bytes (see also the legend in the figure of the **NMT state diagram**):

<CMD>	Meaning
01_h	Switch to the "Operational" state
02_h	Switch to the "Stop" state
80_h	Switch to the "Pre-Operational" state
81_h	Reset Node
82_h	Reset Communication

The value for <Node-ID> can be 00_h; in this case, the NMT command applies for all devices on the CAN bus (broadcast). If a number not equal to zero is used, only the device with the corresponding node-ID is addressed.

The "Reset Node" command completely restarts the controller; the "Reset Communication" command only resets the CANopen settings and restarts communication.

Example: If all devices on the CAN bus are to be switched to the "Stop" operating state, a broadcast with the "Switch to the Stop state" command can be used. The NMT message is structured as follows:

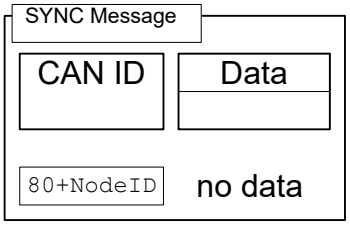
```
000 | 02 00
```

If only the device with node-ID 42 is to be completely restarted, the following CAN message is to be sent:

```
000 | 81 2A
```

8.2.2 Synchronization object (SYNC)

The Synchronization object is used to simultaneously validate the time of PDO data for all devices on the bus. The sync message is structured as follows:

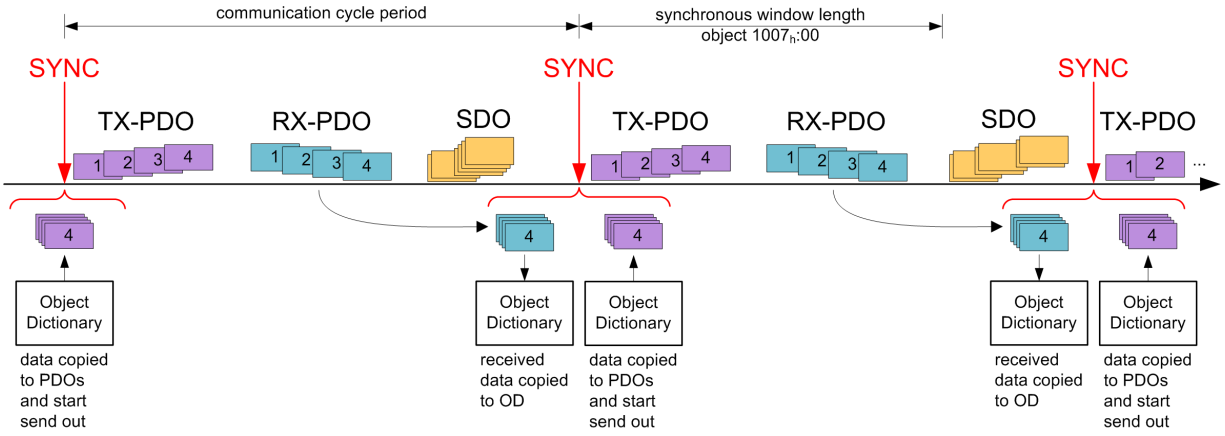


For SYNC operation, transmission mode (Transmission Type) 0 is normally used for the RX-PDOs (data are valid with the next SYNC); for TX-PDOs, a transmission mode between 1 and 240 is selected. (Details: see chapter **Process Data Object (PDO)**).

After receiving a SYNC message, there is a time window ("synchronous window") within which PDO messages can be sent and received; after the time of the window has elapsed, all devices must stop sending PDOs. The "synchronous window length" can be set in milliseconds in object **1007**$_h$:00$_h$.

A typical CAN-SYNC operation is divided into four phases (see also the following figure):

1. The SYNC message is received. The previously received RX-PDO data are thereby copied to the object dictionary (if present). At that time, the data are also sampled and copied to the TX-PDOs and the sending of these messages initiated.
2. The TX-PDOs are then sent by all slaves on the bus.
3. Afterwards, the PDOs are sent by the CANopen master. After the "synchronous window length" time has elapsed, no further PDOs are permitted.
4. SDO messages can be exchanged at the latest when the "synchronous window" is closed again.



8.2.3 Emergency Object (EMCY)

A message of type "Emergency" is sent whenever an error occurs in the controller that was not caused by an SDO access. This service is unconfirmed and is sent with CAN-ID 80$_h$+Node-ID.

The emergency message is structured as follows:

Emergency Message (EMCY)

CAN ID	Data							
	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
80+NodeID	<EMCY Error Code>		<E-REG>	<Manufactor Specific Error Field>				

A total of three error codes are transferred here: the "Emergency Error Code" (`<EMCY Error Code>`), the content of the "Error Register" object (**1001**$_h$, `<E-REG>`) and a manufacturer-specific code (`Manufacturer Specific Error`)

Error handling

A module for error handling processes all errors that occur internally. Each error is classified into an error class.

Each error that occurs is handled as follows:

- The bit that belongs to the error in the "Error Register" object (**1001**$_h$) is set.
- Three pieces of information are then written together in the "Pre-defined Error Field" object (**1003**$_h$:01):
  1. The *Emergency Error Code*
  2. The *Error Register*
  3. The manufacturer-specific error code

- If no further errors are pending, the following message is sent:
  `80 + Node-ID | 00 00 00 00 00 00 00 00`

8.2.4 Service Data Object (SDO)

A "Service Data Object" permits read or write access of the object dictionary.

In the following, the owner of the object dictionary is referred to as the "server"; the CAN node – which wants to request or write the data – is referred to as the "client". An "upload" refers to the reading of a value of an object from the object dictionary; a "download" refers to the writing of a value in the object dictionary. In addition, the following abbreviations are used in the diagrams:

- `<IDX>`: Index of the object that is to be read or written in the object dictionary; the LSB of the index is in byte 1 here. Example: The statusword of the controller has index 6041_h; byte 1 is then written with 41_h and byte 2 with 60_h. With **Expedited Transfer**, the SDO answer contains the same index as that of the request.
- `<SUBIDX>`: Subindex of the object in the object dictionary from 00_h to FF_h. With **Expedited Transfer**, the answer of the SDO message of the controller also contains the subindex of the request.

Because CAN messages of type SDO contain a large amount of metadata, SDO messages should only be used to configure the controller. Should it be necessary to exchange data cyclically during running operation, it makes more sense to make use of CANopen messages of type PDO (see subsection **Process Data Object**).

The SDO transfers are divided into three types of access:

- "expedited transfer" for transferring objects with up to four bytes.
- "normal transfer" for transferring any number of bytes, whereby each CAN message is confirmed individually.
- "block transfer" is also for any number of bytes; here, a given block of CAN tickets is confirmed at once.

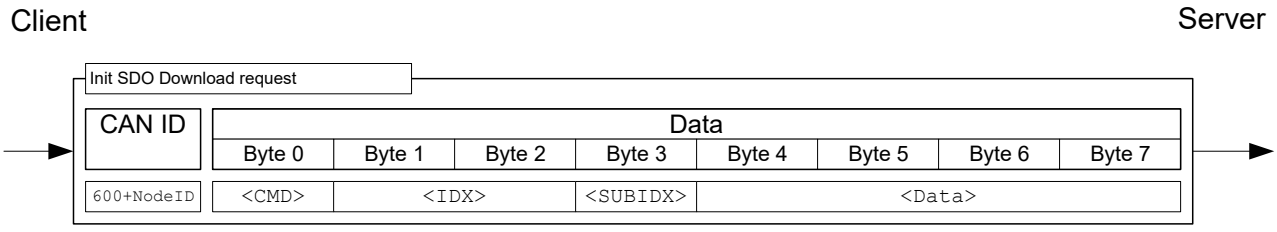
An SDO message is sent to CAN-ID 600_h + node-ID, the answer comes from CAN-ID 580_h + node-ID.

Expedited Transfer

This method is used to write (download) or read (upload) values in objects of type (UN)SIGNED8, INTEGER16 oder INTEGER32 in the object dictionary. This service is confirmed, i.e., each access is answered with data, with a confirmation or with an error message.

SDO Download

An expedited SDO message for writing data in the object dictionary of the server is structured as follows:



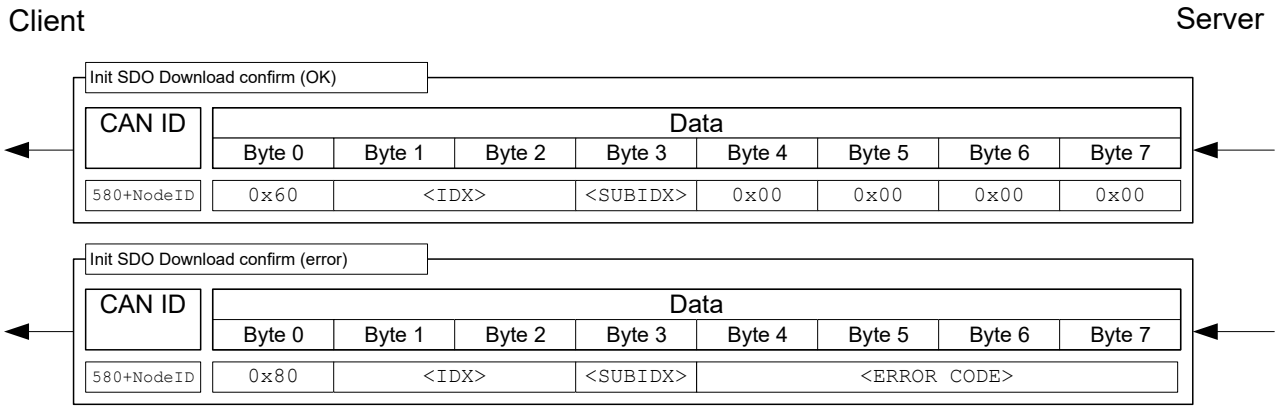
Here, the `<CMD>` byte is dependent on the length of the data that are to be written. `<CMD>` can be one of the following values:

- 1 byte data length: $2F_h$
- 2 byte data length: $2B_h$

- 3 byte data length: 27_h
- 4 byte data length: 23_h

The <Data> field is written with the data that are to be written; the LSB of the data is entered in byte 4.

The answer from the server is either a confirmation of the write operation or an error message (structure of the messages: see following figure). In the latter case, the reason for the error is also sent with the data (see list of the SDO error messages in section **SDO error messages**).



Example: Set object **607A**$_h$:00_h (target position, SIGNED32) to value $3E8_h$ (=1000_d) of a controller with node-ID 3:

```
603 | 23 7A 60 00 E8 03 00 00
```

Where

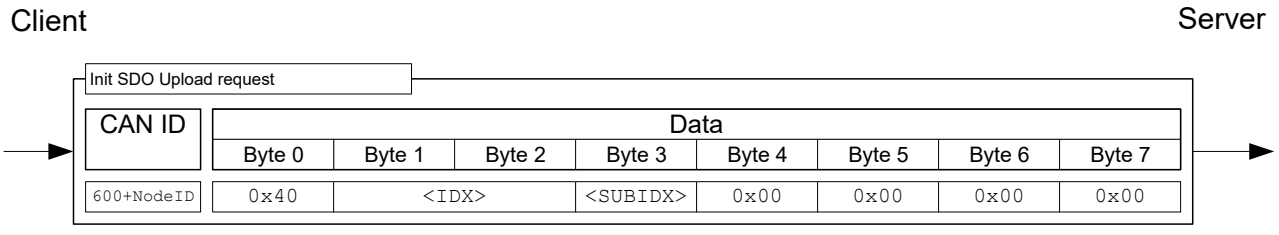
- Byte 1 (23_h): SDO expedited download, 4 bytes of data (SIGNED32)
- Bytes 2 and 3 ($7A_h$ 60_h): index of object is $607A_h$
- Byte 4 (00_h): subindex of object is 00_h
- Bytes 5 to 8 ($E8_h$ 03_h 00_h 00_h): value of object: $000003E8_h$

If successful, the controller responds with this message:

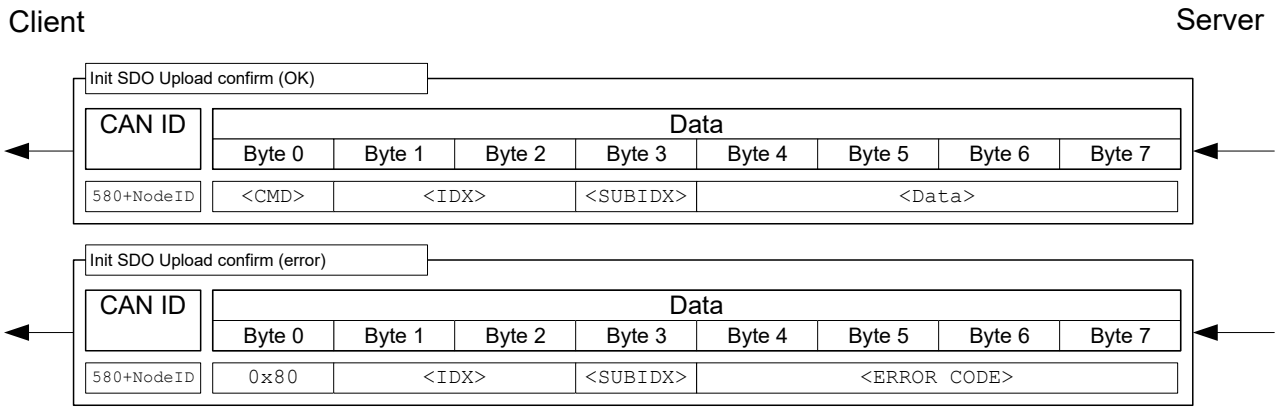
```
583 | 60 7A 60 00 00 00 00 00
```

SDO upload

A CAN message for reading an object from the object dictionary has the following structure:



The server responds with one of the following messages.



The length of the data is encrypted in the <CMD> of the answer:

- 1 byte data length: $4F_h$
- 2 byte data length: $4B_h$
- 3 byte data length: 47_h
- 4 byte data length: 43_h

The LSB of the data is again in byte 4 here.

In case of an error, the reason for the error is also specified in the data (see list of SDO error messages in **SDO error messages**).

Example: To read the "statusword" object (**6041**$_h$:00) from the object dictionary, it suffices to send the following message (always 8 bytes):

```
603 | 40 41 60 00 00 00 00 00
```

The controller generally responds with the following message:

```
583 | 4B 41 60 00 40 02 00 00
```

Where

- Byte 1 ($4B_h$): SDO expedited upload, 2 bytes of data (UNSIGNED16)
- Bytes 2 and 3 (41_h 60_h): index of object is 6041_h
- Byte 4 (00_h): subindex of object is 00_h
- Bytes 5 to 6 (40_h 02_h): value of object: 0240_h
- Bytes 7 to 8 (00_h 2_h h h): empty. An SDO message always consists of 8 bytes.

Normal Transfer

The CANopen "expedited" transfer is limited to maximum four bytes; to exceed this limit, the so-called "normal transfer" must be supported. With this type of transfer, the content of multiple messages is grouped together with respect to content; such a block of messages is referred to in the following as a "transfer". Each message within a transfer is confirmed individually here.

At the time the document was produced, this is only necessary for objects of type "String". Because a string has the "read only" access restriction, an SDO download is not necessary; only the SDO upload is therefore discussed in this document.

Lack of support of the "normal transfer" of a master

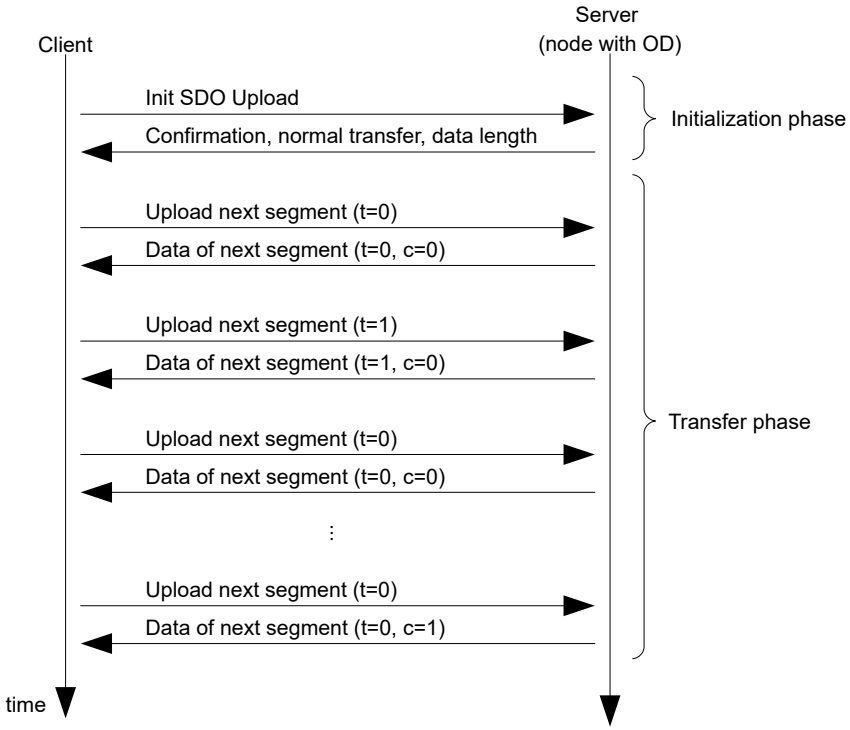
If the controller is to be operated by a master that does not support "normal transfer", access of objects with the String data type can also be handled in another way: each string can be read out character by character with an SDO upload to subindex 1 and the subsequent subindices.

Example: Object **6505**$_h$ (http drive catalogue address) is to be read out. If the master supports "normal transfer", it suffices to begin the upload of the object with subindex 00; the controller automatically

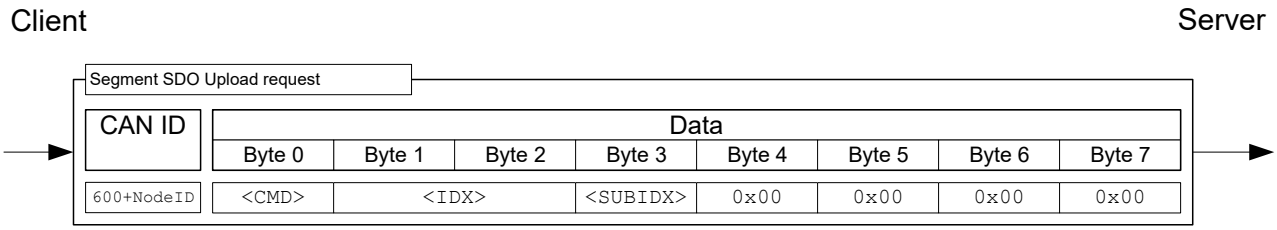
switches over to "normal transfer". If the master only supports "expedited transfer", objects 6505_h:01, 6505_h:02, 6505_h:03, etc., can be used to read the string character by character.

SDO upload

Shown in the following figure is the procedure for an "SDO upload" (client requests that the content of an object be sent to it). The transfer is broken down into two phases: an initialization phase and a transfer phase.



As with an "expedited transfer", the upload begins with the client sending an "Init SDO Update" to the server (see following figure).



The answer for a "normal transfer" does not contain the quantity of bytes to be received encoded in the <CMD>. It is instead entered in the data range as can be seen in the following figure in the <DATA LENGTH> area.

Client Server

Init SDO Upload confirm

CAN ID	Data							
	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
580+NodeID	0x41	<IDX>		<SUBIDX>	<DATA LENGTH>			

The initialization is thereby concluded; all that remains is the upload of the data. A data packet is requested with the following SDO request:

Client Server

Segment SDO Upload request

CAN ID	Data							
	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
600+NodeID	<CMD>	<IDX>		<SUBIDX>	0x00	0x00	0x00	0x00

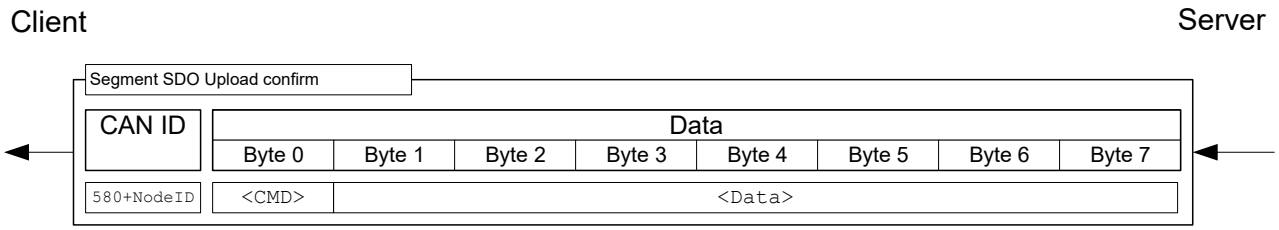
Byte 0 with command <CMD> is structured as follows:

<CMD>

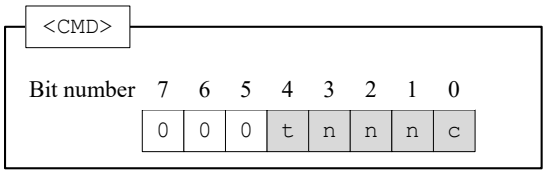
Bit number	7	6	5	4	3	2	1	0
	0	1	1	t	0	0	0	0

The bit with designation t alternates with each request ("toggle bit"). It begins each transfer with 0, even if the previous transfer was aborted.

The controller responds to the above message with the data, whereby the message is structured as follows:



Byte 0 with <CMD> is structured as follows:



The bits have the following meaning here:

t (toggle bit)

The bit alternates with each message sequence; it does not change within a sequence between "request" and "response".

n (number of bytes)

These three bits specify how many bytes contain *no* data. Example: If bits 2 and 1 are set to 0 and bit 3 is set to 1, then $011_b = 03_d$ bytes are not valid. This, in turn, means that byte 1 to byte 4 contain allowed values and byte 5 to byte 7 should be disregarded.

c (more segments)

If no further SDO segments are sent and this is the last segment, the bit is set to 1.

Example: In this example, the "Manufacturer Software Version" object (**$100A_h$**) is to be read. The node-ID of the node in this example is 3.

The corresponding SDO message sequence is listed in the following table. The string that is to be read out varies from controller to controller.

COB-ID	Daten	Beschreibung
603_h	40 0A 10 00 00 00 00 00	Init Upload; Index: $100A_h$; Subindex: 00
583_h	41 0A 10 00 11 00 00 00	Init Upload; Size: indicated; transfer type: normal; Num of bytes: 17; Index: $100A_h$; Subindex: 00
603_h	60 0A 10 00 00 00 00 00	Upload Segment Req.; Toggle bit: not set
583_h	00 46 49 52 2D 76 31 37	Upload Segment Conf.; More segments: yes; num of bytes: 7; Toggle bit: not set
603_h	70 0A 10 00 00 00 00 00	Upload Segment Req.; Toggle bit: set
583_h	10 34 38 2D 42 35 33 38	Upload Segment Conf.; More segments: yes; num of bytes: 7; Toggle bit: set
603_h	60 0A 10 00 00 00 00 00	Upload Segment Req.; Toggle bit: not set

COB-ID	Daten	Beschreibung
583_h	09 36 36 32 00 00 00 00	Upload Segment Conf.; More segments: no (last segment); num of bytes: 3; Toggle bit: not set

All data of the transfer grouped together yield the following string (ASCII values):

```
46 49 52 2D 76 31 37 34 38 2D 42 35 33 38 36 36 32
```

This corresponds to string: "FIR-v1748-B538662"

Aborting the SDO transfer

Both the server and the client are authorized to abort the current transfer *at any time*. To do this, an "Abort SDO Transfer" must be sent; this is depicted in the following.

Abort SDO Transfer

CAN ID	Data						
	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
600/580 + NodeID	0x80	<IDX>		<SUBIDX>	<ERROR CODE>			

After receiving the message, the SDO transfer is considered ended; the service is *not* confirmed. A new SDO transfer must then be started from the very beginning. Transfer of the `<ERROR CODE>` is optional; the controller does not evaluate the code.

SDO error messages

In case of an error, an error number specifying the reason for the error is also sent in the data area.

Error Code	Description
05030000_h	*toggle bit not changed*: Valid only with "normal transfer" or "block transfer". The bit, which is to alternate after each transfer, did not change its state.
05040001_h	*command specifier unknown*: Byte 0 of the data block contains a command that is not allowed.
06010000_h	*unsupported access*: If "complete access" was requested via CAN over EtherCAT (CoE) (is not supported.)
06010002_h	*read only entry*: An attempt was made to write to a constant or read-only object.
06020000_h	*object not existing*: An attempt was made to access a non-existing object (index incorrect).
06040041_h	*object cannot be pdo mapped*: An attempt was made to map an object in the PDO for which that is not permissible.
06040042_h	*mapped pdo exceed pdo*: If the desired object were to be attached to the PDO mapping, the 8 bytes of the PDO mapping would be exceeded.
06070012_h	*parameter length too long*: An attempt was made to write to an object with too much data; for example, with `<CMD>=`23_h (4 bytes) to an object of type Unsigned8, `<CMD>=`$2F_h$ would be correct.
06070013_h	*parameter length too short*: At attempt was made to write to an object with too little data; for example, with `<CMD>=`$2F_h$ (1 byte) to an object of type Unsigned32, `<CMD>=`23_h would be correct.
06090011_h	*subindex not existing*: An attempt was made to access an invalid subindex of an object; the index, on the other hand, would exist.

Error Code	Description
06090031_h	*value too great*: Some objects are subject to restrictions in the size of the value; in this case, an attempt was made to write an excessively large value to the object. For example, the "Pre-defined error field: Number of errors" object for **1003_h**:00 may only be set to the value "0"; all other numerical values result in this error.
06090032_h	*value too small*: Some objects are subject to restrictions in the size of the value. In this case, an attempt was made to write a value that is too small to the object.
08000000_h	*general error*: General error that does not fit in any other category.
08000022_h	*data cannot be read or stored in this state*: The parameters of the PDOs may only be changed in the *Stopped* or "Pre-Operational" state. Write access of objects 1400_h to 1407_h, 1600_h to 1607_h, 1800_h to 1807_h and $1A00_h$ to $1A07_h$ is not permissible in the "Operational" state.

8.2.5 Process Data Object (PDO)

A message that only contains process data is referred to as a "Process Data Object" (PDO). The PDO is intended for data that need to be exchanged cyclically. The idea behind a PDO message is to remove all additional information (index, subindex and data length) from a CAN message and to only fill the CAN message with data. The source and target information for the PDO are stored separately in the so-called PDO mapping.

PDOs can only be used if the NMT state machine is in the "Operational" state (see section **Network Management (NMT)**); the PDOs must be configured in the "Pre-Operational" NMT state.

The controller supports a total of eight independent PDO mappings; each corresponding PDO message can have a maximum of eight bytes (= 64 bits) of user data. It is thereby possible to, for example, transfer two UNSIGNED32 values or one UNSIGNED32 and one UNSIGNED08; the message does not need to use all eight data bytes. The PDOs differ yet again in the configuration in the send and receive configuration. The receive configuration describes the processing for PDO messages that are sent, and the send configuration describes the PDO messages that are to be sent.

RX configuration

To configure an RX-PDO, three object categories in the object dictionary must be taken into account:

1. The objects that describe the functionality of the mapping.
2. The objects that describe the content of the mapping.
3. The objects that are to receive the received data.

Configuration of the functionality (communication parameter)

The configuration of the first mapping is stored in the subindices of object 1400_h. The second mapping is configured in 1401_h and so on. In the following, we refer to $140N_h$. Here, the configuration affects the COB-ID of the PDO message and the transfer type.

Objects $140N_h$ have only three subindices:

- Subindex 0 (max. subindex): Total number of subindices
- Subindex 1 (COB-ID): The COB-ID is stored here. For PDO mappings 1–4 (1600_h–1603_h), the CAN-ID is fixed depending on the node-ID and only the valid bit (bit 31) can be set in the COB-ID. From 1604_h–1607_h, the CAN-ID can be set independently (with the restriction that it not be used by other services, see table at the start of chapter **CANopen services**) as can the valid bit. The change of a COB-ID does not take effect until *after* the controller or communication is restarted (see **Network Management (NMT)**).

Mapping	COB-ID
1600_h	200_h + Node-ID
1601_h	300_h + Node-ID

Mapping	COB-ID
1602_h	400_h + Node-ID
1603_h	500_h + Node-ID
1604_h	xxx_h + Node-ID
1605_h	xxx_h + Node-ID
1606_h	xxx_h + Node-ID
1607_h	xxx_h + Node-ID

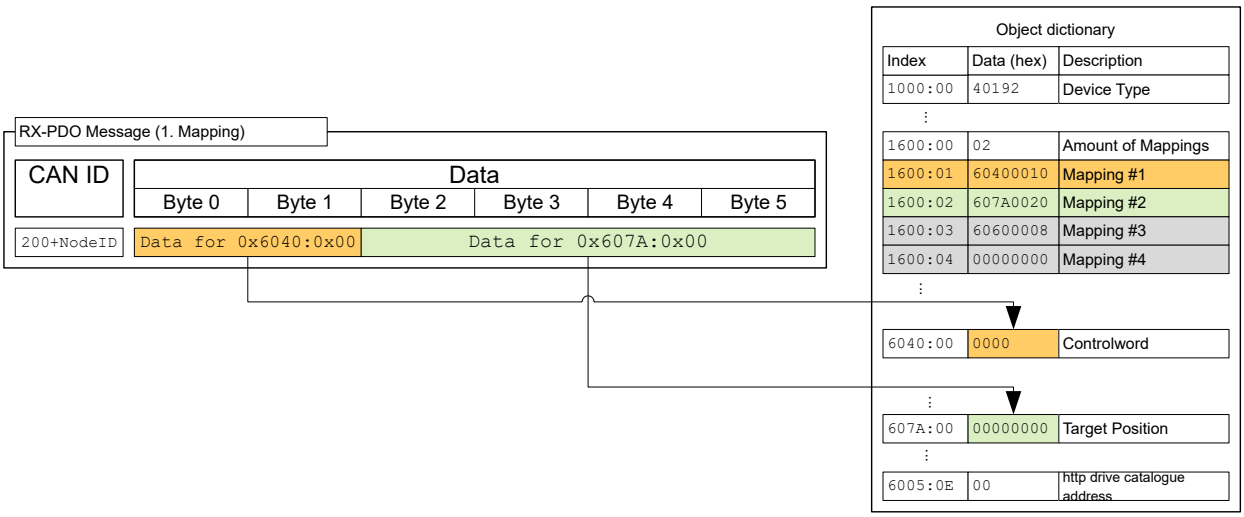
- Subindex 2 (transmission type): A number is stored in this subindex that defines the time at which the received data become valid. The number and the corresponding meaning can be found in the following table.

$140N_h:02_h$	Meaning
00_h-$F0_h$	Synchronous: The data are buffered and not until the next SYNC message is received do they become valid and are they taken over into the object dictionary.
$F1_h$-FD_h	Reserved
FE_h, FF_h	Asynchronous: The data become valid when the PDO message is received and are taken over into the object dictionary.

Content of a mapping

The configuration of the content of a mapping is structured as follows (see also the following figure as an example):

- All subindices of a configuration object belong together. Thus, 1600_h with all subindices describes the first mapping, 1601_h the second RX-PDO mapping, etc.
- Subindex 00_h specifies how many objects are in a mapping. It simultaneously specifies how many of the subindices are valid. If object 1600_h:00_h is set to "0", RX mapping is thereby completely switched off. In the example shown in the following figure, two objects are thus mapped; object 1600_h:03_h and 1600_h:04_h is, therefore, not active (shown in gray).
- Each subindex from 1600_h:01_h to 1600_h:$0F_h$ describes one target of the mapping sequentially and without gaps. The index, subindex and bit length are thereby encoded. Example from the following figure: The first two bytes of the message are to be written in object 6040_h:00_h. In hexadecimal notation, the content of 1600_h:01_h then consists of
  `<Index><Subindex><Bit length>`
  or `60400010`. The second mapping (1600_h:02_h) contains the entry `607A0020`. Thus, it maps the following four bytes (=20_hBit) in object $607A_h$:00_h



TX configuration

To configure an RX-PDO, three object categories in the object dictionary must be taken into account:

1. The objects that describe the functionality of the mapping.
2. The objects that describe the content of the mapping.
3. The objects that are to receive the data that are to be sent.

Also note that the time at which the data are copied to the TX-PDO message and the time of sending do not need to be the same (dependent on mode).

Configuration of the functionality (communication parameter)

The configuration of the functionality of the first mapping is stored in the subindices of object 1800_h. The second mapping is configured in 1801_h and so on. In the following, we refer to $180N_h$. Here, the configuration affects the COB-ID of the PDO message and the transfer type.

Objects $180N_h$ have the following subindices:

- Subindex 0 (max. subindex): Total number of subindices
- Subindex 1 (COB-ID): The COB-ID is stored here. For PDO mappings 1–4 ($1A00_h$–$1A03_h$), the CAN-ID is fixed depending on the node-ID and only the valid bit (bit 31) can be set in the COB-ID. From $1A04_h$–$1A07_h$, the CAN-ID can be set independently (with the restriction that it not be used by other services, see table at the start of chapter **CANopen services**) as can the valid bit. A COB-ID change does not take effect until *after* the controller or communication is restarted (see **Network Management (NMT)**).

Mapping	COB-ID
$1A00_h$	180_h + Node-ID
$1A01_h$	280_h + Node-ID
$1A02_h$	380_h + Node-ID
$1A03_h$	480_h + Node-ID
$1A04_h$	xxx_h + Node-ID
$1A05_h$	xxx_h + Node-ID
$1A06_h$	xxx_h + Node-ID
$1A07_h$	xxx_h + Node-ID

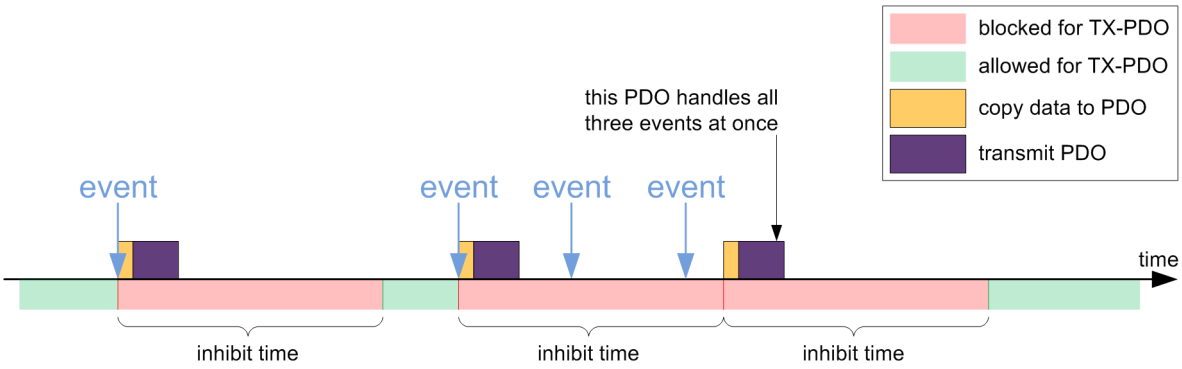
- Subindex 2 (transmission type): A number is stored in this subindex that defines the time at which the data are to be copied into the PDO message and when this is to be sent. The number and the corresponding meaning can be found in the following table. Below, we refer to an *Event* that can trigger the copying and/or sending of the data. This *Event* consists of three events, which can be considered independently of one another:
  - The NMT state machine is switched to "operational".
  - The current data have changed with respect to the last PDO message.
  - The *Event Timer* has expired (see $180N_h$:5).

  If the *Event Timer* is used, it is handled independently of the changes; the *Event Timer* is not restarted until the current event timer expires, not because of another *Event*.

$180N_h:02_h$	Meaning
0	Synchronous (acyclic): The data are copied to the TX-PDO upon arrival of the SYNC but are not sent until the *Event*.
01_h-$F0_h$	Synchronous (cyclic): The data are copied upon arrival of the nth SNCY message and are sent immediately thereafter (n corresponds to the number 1 to 240, transmission type "1" sends the new data on each SYNC).
$F1_h$-FB_h	Reserved

180N$_h$:02$_h$	Meaning
FC$_h$	RTR-Only (synchronous): The data are copied upon arrival of each SYNC message but are sent only upon request with an RTR message.
FD$_h$	RTR-Only (event-driven): The data are copied to the TX-PDO message upon receipt of an RTR message and sent immediately thereafter.
FE$_h$, FF$_h$	The data are copied upon arrival of the *Event* and sent immediately.

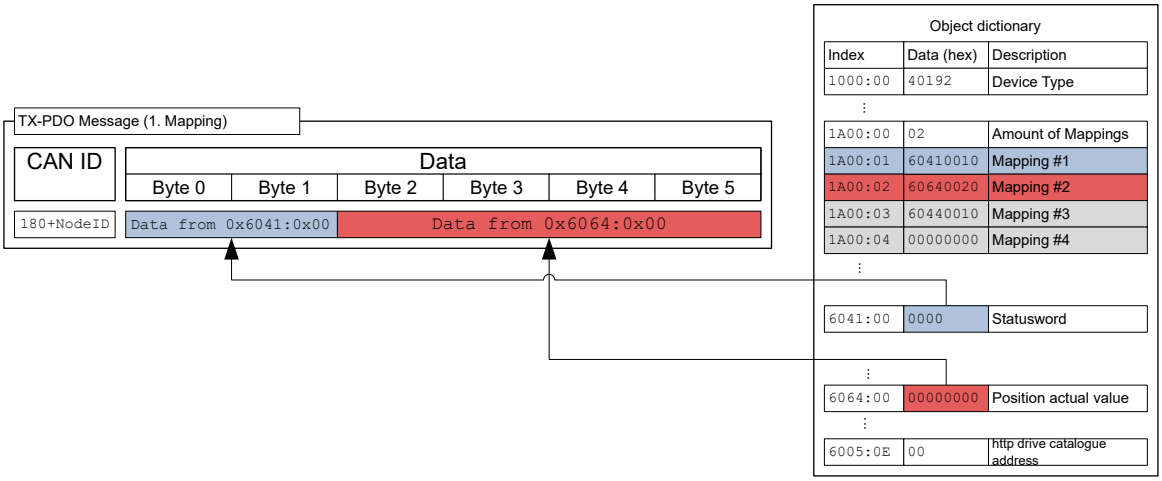
- Subindex 3 (inhibit time): This subindex contains a time lock in ms (see following figure). This can be used to set a time that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs. This is intended to prevent PDOs from being sent continuously if the mapped object constantly changes.
- Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.
- Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.



Content of a mapping

The configuration of the content of a mapping is structured as follows (see the following figure as an example):

- All subindices of a configuration object belong together. Thus, 1A00$_h$ with all subindices describes the first mapping, 1A01$_h$ the second RX-PDO mapping, etc.
- Subindex 00 specifies how many objects are in a mapping. It simultaneously specifies how many of the subindices are valid. If object 1A00$_h$:00$_h$ is set to "0", RX mapping is thereby completely switched off. In the following example, two objects are thereby mapped in entries 1A00$_h$:01$_h$ – 1A00$_h$:02$_h$. The objects in entries 1A00$_h$:03$_h$ – 1A00$_h$:04$_h$ are, thus, not mapped (shown in gray).
- Each subindex from 1A00$_h$:01$_h$ to 1A00$_h$:0F$_h$ respectively describes sequentially and without gaps (dummy objects can be used for gaps) one source of the mapping. The index, subindex and bit length are thereby encoded. Example from the following figure: The first two bytes of the message are to be read from object 6041$_h$:00$_h$. In hexadecimal notation, the content of 1A00$_h$:01$_h$ then consists of `<Index><Subindex><Bit Length>`, or `60410010`. The second mapping (1A00$_h$:02$_h$) contains the entry `60640020`. Thus, it maps the following four bytes (corresponds to 32 bits) from object 6064$_h$:00$_h$ in the TX-PDO message.



Presetting

The following configuration is preset:

RX-PDO

1. Mapping (CAN-ID: 200_h + Node-ID):

- **6040_h**:00_h (controlword)
- **6060_h**:00_h (mode of operation)
- **3202_h**:02_h (motor drive submode select)

2. Mapping (CAN-ID: 300_h + Node-ID):

- **$607A_h$**:00_h (target position)
- **6081_h**:00_h (profile velocity)

3. Mapping (CAN-ID: 400_h + Node-ID): object **6042_h**:00_h (vl target velocity)

4. Mapping (CAN-ID: 500_h + Node-ID): object **$60FE_h$**:01_h (digital outputs)

TX-PDO

1. Mapping (CAN-ID: 180_h + Node-ID):

- **6041_h**:00_h (statusword)
- **6061_h**:00_h (Position actual value)

2. Mapping (CAN-ID: 280_h + Node-ID): **6064_h**:00_h (Position actual value)

3. Mapping (CAN-ID: 380_h + Node-ID): **6044_h**:00_h (vl velocity actual value)

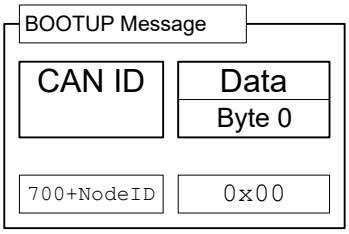
4. Mapping (CAN-ID: 480_h + Node-ID): object **$60FD_h$**:00_h (digital inputs)

Changing PDO mapping

1. Deactivate the PDO by setting the *Valid Bit* (bit 31) of subindex 01h of the corresponding communication parameter (e.g., 1400_h:01_h) to "1".
2. Deactivate the mapping by setting subindex 00h of the corresponding mapping parameter (e.g., 1600_h:00_h) to "0".
3. Change the mapping in the desired subindices (e.g., 1600_h:01_h).
4. Activate the mapping by writing the number of objects that are to be mapped in subindex 00h of the corresponding mapping parameter (e.g., 1600_h:00_h).
5. Activate the PDO by setting bit 31 of subindex 01h of the corresponding communication parameter (e.g., 1400_h:01_h) to "0".

8.2.6 Boot-Up Protocol

If the CAN slave reaches the "Pre-Operational" NMT state (see following figure), the following message is sent to signal operational readiness.



This service is unconfirmed; there is no response.

Note

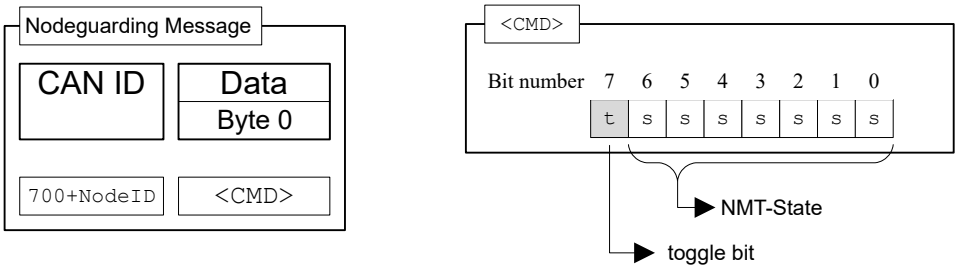
The boot loader sends its own boot-up message. This can be suppressed, see object **2007**$_h$:00

8.2.7 Heartbeat and nodeguarding

With the "heartbeat" and "nodeguarding" services (often also referred to as "lifeguarding"), switched-off or hung devices on the CAN bus can be detected. For this purpose, the NMT master cyclically requests a message with the current NMT state of the slave (nodeguarding). The alternative is that each slave sends a message unprompted and cyclically (heartbeat). A combination of nodeguarding and heartbeat is not permissible. Furthermore, it is recommended that heartbeat be given preference over nodeguarding, as nodeguarding results in a higher load on the CAN bus.

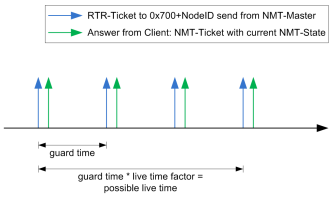
Nodeguarding

This service is based on the fact that the NMT master sends an RTR message with CAN-ID 700$_h$ + node-ID to the respective slave. The slave must then send a message as response; this message is structured as follows. Bit 7 alternates here on each transfer, thereby allowing one to determine if a message was lost. Entered in bits 6 to 0 is the current NMT status of the slave.



With nodeguarding, there exist three time intervals (see also the following figure):

1. *guard time*: The time between two RTR messages. This can be different for each CAN node and is stored in the slave in object **100C**$_h$:00 (unit: milliseconds)
2. *live time factor*: A multiplier for the *guard time*; this is stored in the CAN slave in object **100D**$_h$:00 and can be different for each slave on the CAN bus.
3. *possible live time*: The time produced by multiplying *guard time* and *live time factor*.

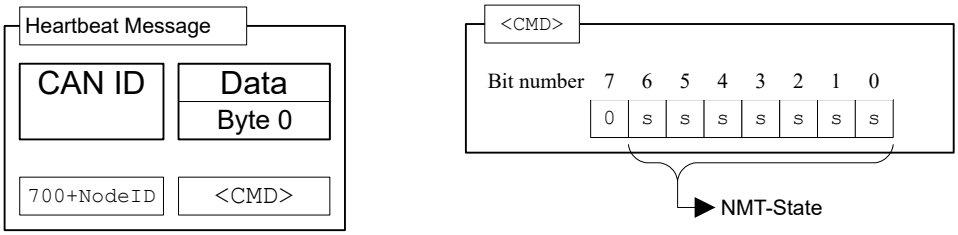


The following conditions are checked during nodeguarding:

1. The NMT master must send the RTR request within the "possible live time".
2. The slave must send the response to the RTR request within the "possible live time".
3. The slave must respond with its NMT state. In addition, the "toggle bit" must be set correctly.

Heartbeat

If heartbeat is activated, the slave sends its NMT state to the CAN bus unprompted and cyclically. This service is activated by setting the *producer heartbeat time* object in object **1017**$_h$:00$_h$ to a value other than zero. The *producer heartbeat time* is measured in milliseconds. The message sent by the slave has the form shown below:



The slave must send the heartbeat message within the *heartbeat consumer time*. This time is known only to the master and is not stored in the controller.

9 Modbus RTU

The controller can be addressed by means of Modbus RTU and can function in an RS-485 network as a *slave*. In this chapter, the function codes of the Modbus communication structure are described.

Modbus references: **www.modbus.org**.

- *MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3*, Date: 26.04.2014, Version: 1.1b3
- *MODBUS over Serial Line Specification and Implementation Guide V1.02*, Date: 20.12.2006, Version: 1.02

The I/O data with any preconfigured drive values (see **Process data objects (PDO)**) can be sent with the standard Modbus function codes. To configure your own I/O data, however, function code 2Bh (CAN Encapsulation) must be supported by the Modbus master in order for the parameters to be read and written independent of the process image.

If the master does not support this function code, the I/O image can be configured and stored using *Plug & Drive Studio*. The master can then access the data using the standard Modbus function codes.

Configuration via the configuration file is not otherwise possible (see chapter **Configuration via USB**).

9.1 RS-232 and RS-485

The "Two-Wire Modbus Interface" electrical interface is supported by the controller in accordance with standard EIA/TIA-485 (RS-485).

For a connection via RS-485, make certain that jumpers J1 and J2 are plugged in correctly (see chapter **Jumper J1/J2** for further information). When using the RS-232 interface, the corresponding connector is to be used; no further configuration is necessary.

In principle, Modbus RTU can be operated with RS-232 and RS-485. An unused bus can be deactivated with object **2102**$_h$. Object **2103**$_h$ indicates the active fieldbuses.

9.2 Modbus Modicon notation with PLCs

Many PLC devices use the Modicon addressing model. This notation is not used in the Modbus standard.

The following address notation is relevant for Nanotec controllers:

- Input register 30001 - 39999 is mapped to Modbus telegram address 0 (0_h) - 9998 ($270E_h$).
- Holding register 40001 - 49999 is mapped to Modbus telegram address 0 (0_h) - 9998 ($270E_h$).

Note

Where Modbus addresses are mentioned in the manual, it may be necessary to implement the register addresses in the PLC in accordance with *Modicon notation*.

9.3 General

Modbus is generally big-endian based.

The only exceptions are the commands with function codes 43 ($2B_h$), 101 (65_h) and 102 (66_h), which are based on CANopen. For the data values of these commands, the little-endian format applies. The remainder of the Modbus message, on the other hand, is big-endian based, as in the past.

Example

Command $2B_h$: With this command, the value 12345678_h is written in object 0123_h (does not exist):

SA	FC	Data	CRC
05	2B	0D 01 00 01 23 01 00 00 00 00 04 **78 56 34 12**	67 35

SA

Slave address

FC

Function code

Data

Data range, decoding is dependent on the used function code

CRC

Cyclic redundancy check

9.4 Function codes

The following "function codes" are supported:

	Name	Function code	Subfunction code
Data access (16-bit)	Read Holding Registers	03 (03_h)	
	Read Input Register	04 (04_h)	
	Write Single Register	06 (06_h)	
	Write Multiple Registers	22 (16_h)	
	Read/Write Multiple Registers	23 (17_h)	
Diagnosis	Clear Counters and Diagnostic Register	08 (08_h)	10 ($0A_h$)
	Return Bus Message Count	08 (08_h)	11 ($0B_h$)
	Return Bus Communication Error Count	08 (08_h)	12 ($0C_h$)
	Return Bus Exception Error Count	08 (08_h)	13 ($0D_h$)
	Return Server Message Count	08 (08_h)	14 ($0E_h$)
	Return Server No Response Count	08 (08_h)	15 ($0F_h$)
	Return Server NAK Count	08 (08_h)	16 (10_h)
	Return Server Busy Count	08 (08_h)	17 (11_h)
	Return Bus Character Overrun Count	08 (08_h)	18 (12_h)
Miscellaneous	Encapsulated Interface Transport	43 ($2B_h$)	13 ($0D_h$)
	Read complete object dictionary start	101 (65_h)	85 (55_h)
	Read complete object dictionary next	101 (65_h)	170 (AA_h)
	Read complete array or record start	102 (66_h)	85 (55_h)
	Read complete array or record next	102 (66_h)	170 (AA_h)

9.5 Function code descriptions

9.5.1 FC 3 (03_h) Read Input Registers / FC 4 (04_h) Read Holding Registers

With this function code, one 16-bit value or multiple 16-bit values can be read. This function can be applied to NanoJ objects (see **NanoJ objects**) or process data objects (min. 4-byte alignment, see **Process data objects (PDO)**).

Request		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	03_h / 04_h
Start address	2 bytes	0000_h to $FFFF_h$
Number of registers	2 bytes	1 to ($7D_h$)
CRC	2 bytes	

Response ("M" corresponds to the number of registers to be read)		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	03_h / 04_h
Number of bytes	1 byte	2 * M
Register value	2 bytes	
CRC	2 bytes	

Error		
Name	**Length**	**Value**
Slave address	1 byte	
Error code	1 byte	83_h / 84_h
Exception code	1 byte	01, 02, 03 or 04
CRC	2 bytes	

Example

Below is an example of a read request and response of register 5000 (1388_h) and of the following register (2 registers):

Request

SA	FC	Data	CRC
05	03	13 88 00 02	41 21

Response

SA	FC	Data	CRC
05	03	04 02 40 00 00	41 21

9.5.2 FC 6 (06_h) Write Single Register

This function code can be used to write a single 16-bit value. The function can be used on process data objects (see **Process data objects (PDO)**).

Request		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	06_h
Register address	2 bytes	0000_h to $FFFF_h$
Register value	2 bytes	0000_h to $FFFF_h$
CRC	2 bytes	

Response		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	06_h
Register address	2 bytes	0000_h to $FFFF_h$
Register value	2 bytes	0000_h to $FFFF_h$
CRC	2 bytes	

Error		
Name	**Length**	**Value**
Slave address	1 byte	
Error code	1 byte	86_h
Exception code	1 byte	01, 02, 03 or 04
CRC	2 bytes	

Example

Below is an example of a write request and response in register 6000 (1770_h) with the value "0001_h":

Request

SA	FC	Data	CRC
05	06	17 70 00 01	4D E1

Response

SA	FC	Data	CRC
05	06	17 70 00 01	4D E1

9.5.3 FC 16 (10_h) Write Multiple Registers

With this function code, one 16-bit value or multiple 16-bit values can be written. The function can be applied to NanoJ objects (see **Process data objects (PDO)**) or process data objects (see **NanoJ objects**).

Request ("N" is the number of registers to be written)		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	10_h
Start address	2 bytes	0000_h to $FFFF_h$
Number of registers	2 bytes	0001_h to $007B_h$
Number of bytes	1 byte	2 * N
Register value	N * 2 bytes	
CRC	2 bytes	

Response		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	10_h
Start address	2 bytes	0000_h to $FFFF_h$
Number of registers	2 bytes	0001_h to $007B_h$
CRC	2 bytes	

Error		
Name	**Length**	**Value**
Slave address	1 byte	
Error code	1 byte	90_h
Exception code	1 byte	01, 02, 03 or 04
CRC	2 bytes	

Example

Below is an example for writing values "0102_h" and "0304_h" starting with register address 6000 (1770_h), number of registers is 2, length of the data is 4:

Request

SA	FC	Data	CRC
05	10	17 70 00 02 04 01 02 03 04	AB 44

Response

SA	FC	Data	CRC
05	10	17 70 00 02	44 23

9.5.4 FC 17 (11_h) Report Server ID

This function code can be used to read the description of the type, the current status and other information about the device.

Request		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	11_h
CRC	2 bytes	

Response		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	03_h
Number of bytes	1 byte	01_h
Run Indicator Status	1 byte	00_h = OFF, FF_h = ON
Additional data		
CRC	2 bytes	

Error		
Name	**Length**	**Value**
Slave address	1 byte	
Error code	1 byte	91_h
Exception code	1 byte	01 or 04
CRC	2 bytes	

Example

Below is an example of a request/response for ID and status:

Request

SA	FC	CRC
05	11	C2 EC

Response

SA	FC	Data	CRC
05	11	02 05 FF	0F EC

9.5.5 FC 23 (17_h) Read/Write Multiple registers

With this function code, one 16-bit value or multiple 16-bit values can be simultaneously read and written. The function can be applied to NanoJ objects (see **Process data objects (PDO)**) or process data objects (see **NanoJ objects**).

Request ("N" is the number of registers to be read):		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	17_h
Read: Start address	2 bytes	0000_h to $FFFF_h$
Read: Number of registers	2 bytes	0001_h to 0079_h
Write: Start address	2 bytes	0000_h to $FFFF_h$
Write: Number of registers	2 bytes	0001_h to 0079_h
Write: Number of bytes	1 byte	2 * N
Write: Register value	N * 2 bytes	
CRC	2 bytes	

Response ("M" corresponds to the number of bytes to be written):		
Name	**Length**	**Value**
Slave address	1 byte	
Function code	1 byte	17_h
Number of bytes	1 byte	2 * M
Registers read	M * 2 bytes	
CRC	2 bytes	

Error		
Name	**Length**	**Value**
Slave address	1 byte	
Error code	1 byte	97_h
Exception code	1 byte	01, 02, 03 or 04
CRC	2 bytes	

Example

Below is an example for reading two registers beginning with register 5000 (1388_h) and for writing two registers beginning with register 6000 (1770_h) with 4 bytes and data "0102_h" and "0304_h":

Request

SA	FC	Data	CRC
05	17	13 88 00 02 17 70 00 02 04 01 02 03 04	56 6A

Response

SA	FC	Data	CRC
05	17	04 02 40 00 00	0F EC

9.5.6 FC 8 (08_h) Diagnostics

Modbus function code FC08 offers numerous tests for checking the communication system between client and server or for checking various internal error states within the server.

This function uses a two-byte subfunction code in the request for defining the type of test. In a normal response, the server repeats both, the function and the subfunction code. Some diagnoses contain data of the device in the data field of the normal response.

Request:

Name	Length	Value
Function code	1 byte	08_h
Subfunction code	2 bytes	
Data	N x 2 bytes	

Response:

Name	Length	Value
Function code	1 byte	08_h
Subfunction code	2 bytes	
Data	N x 2 bytes	

Error:

Name	Length	Value
Function code	1 byte	88_h
Exception code	1 byte	01 or 03 or 04

FC 8.10 (08_h.$0A_h$) Clear Counters and Diagnostic Register

The objective of this request is to reset all counters and diagnosis registers. Counters are also reset when the controller is switched on.

Subfunction	Data range	
	Request	Response
00_h $0A_h$	00_h - 00_h	Echo of the request data

Example

Request

SA	FC	Data	CRC
05	08	00 0A 00 00	56 6A

Response

SA	FC	Data	CRC
05	08	00 0A 00 00	C1 8D

FC 8.11 (08_h.$0B_h$) Return Bus Message Count

The response data range returns the number of messages detected by the communications system since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller.

Subfunction	Data range	
	Request	Response
00_h $0B_h$	00_h - 00_h	Total Message Count

FC 8.12 (08_h.$0C_h$) Return Bus Communication Error Count

The response data range returns the number of CRC errors since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller.

Subfunction	Data range	
	Request	Response
00_h $0C_h$	00_h - 00_h	CRC Error Count

Example

Request

SA	FC	Data	CRC
05	08	00 0C 00 00	21 8C

Response

SA	FC	Data	CRC
05	08	00 0C 00 00	21 8C

FC 8.13 (08_h.$0D_h$) Return Bus Exception Error Count

The response data range returns the number of Modbus exceptions since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller.

Subfunction	Data range	
	Request	Response
00_h $0D_h$	00_h - 00_h	Exception Error Count

Example

Request

SA	FC	Data	CRC
05	08	00 0D 00 00	70 4C

Response

SA	FC	Data	CRC
05	08	00 0D 00 00	70 4C

FC 8.14 (08_h.$0E_h$) Return Server Message Count

The response data range returns the number of messages addressed to the device and the number of broadcast messages that were processed by the controller. The number of messages since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller are counted.

Subfunction	Data range	
	Request	Response
00_h $0E_h$	00_h - 00_h	Server Message Count

Example

Request

SA	FC	Data	CRC
05	08	00 0E 00 00	80 4C

Response

SA	FC	Data	CRC
05	08	00 0E 00 00	80 4C

FC 8.15 (08_h.$0F_h$) Return Server No Response Count

The response data range returns the number of messages addressed to the controller for which no response was returned (neither normal response nor exception response). The number of messages since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller are counted.

Subfunction	Data range	
	Request	Response
00_h $0F_h$	00_h - 00_h	No Response Count

Example

Request

SA	FC	Data	CRC
05	08	00 0F 00 00	D1 8C

Response

SA	FC	Data	CRC
05	08	00 0F 00 00	D1 8C

FC 8.16 (08_h.10_h) Return Server NAK Count

The response data range returns the number of messages for which a "Negative Acknowledge (NAK)" exception response was returned. The number of messages since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller are counted.

Subfunction	Data range	
	Request	Response
00_h - 10_h	00_h - 00_h	Server NAK Count

Example

Request

SA	FC	Data	CRC
05	08	00 10 00 00	E0 4A

Response

SA	FC	Data	CRC
05	08	00 10 00 00	E0 4A

FC 8.17 (08_h.11_h) Return Server Busy Count

The response data range returns the number of messages for which a "Server Device Busy" exception response was returned. The number of messages since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller are counted.

Subfunction	Data range	
	Request	Response
00_h - 11_h	00_h - 00_h	Server NAK Count

Example

Request

SA	FC	Data	CRC
05	08	00 11 00 00	B1 8A

Response

SA	FC	Data	CRC
05	08	00 11 00 00	B1 8A

FC 8.18 (08_h.12_h) Return Bus Character Overrun Count

The response data range returns the number of messages addressed to the controller that could not be processed due to a character overrun. The number of messages since the last restart, "Clear Counters and Diagnostic Register" request, or switching on of the controller are counted. A character overrun occurs when characters arrive at the controller faster than they can be stored or by the loss of a character due to a hardware malfunction.

Subfunction	Data range	
	Request	Response
00_h - 12_h	00_h - 00_h	Server Character Overrun Count

Example

Request

SA	FC	Data	CRC
05	08	00 12 00 00	41 8A

Response

SA	FC	Data	CRC
05	08	00 12 00 00	41 8A

9.5.7 FC 43 ($2B_h$) Encapsulated Interface Transport

This function facilitates simple access of the CANopen object dictionary. Further details can be found in the following documentation:

1. *MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3*, Date: 26.04.2014, Version: 1.1b3

2. *CiA 309 Draft Standard Proposal - Access from other networks - Part 2: Modbus/TCP mapping V1.3*, Date: 30.07.2015, Version: 1.3

Note

For the messages of the Encapsulated Interface Transport, another byte sequence applies in part, see chapter **General**.

Definition of the request and response:

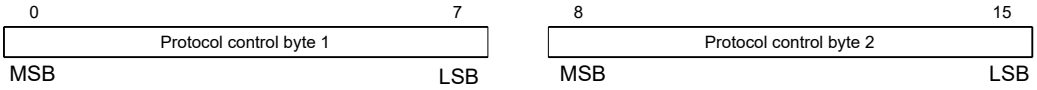
Name	Length	Example/number range
Slave address	1 byte	
Function code	1 byte	$2B_h$ (43_d)
MEI type	1 byte	$0D_h$ (13_d)
Protocol options Range	2 to 5 bytes	
Address and data range	N bytes	
CRC	2 bytes	

Protocol options Range

Name	Length	Example/number range
Protocol control	1 to 2 bytes	See description
Reserved	1 byte	Always 0
(Optional) Counter byte	1 byte	
(Optional) Network ID	1 byte	
(Optional) Encoded data	1 byte	

Protocol control:

The "Protocol control" field contains the flags that are needed for controlling the message protocols. The bytes of the "Protocol control" field are defined as follows if the "extended" flag was set (the second byte is otherwise omitted):



The most significant bit (MSB) is bit 0 for "protocol control" byte 1 and bit 8 for "protocol control" byte 2. The least significant bit (LSB) is bit 7 for "protocol control" byte 1 and bit 15 for "protocol control" byte 2.

Bit	Name	Description
0	"Extended" flag	This bit is used if the object dictionary data set is larger than would fit in a Modbus command. The data set then spans over multiple Modbus messages; each message contains part of the data set. "0" = No multiple message transaction or the end of the multiple message transaction. "1" = Part of a multiple message transaction.

Bit	Name	Description
1	Extended protocol control	Length of the protocol control, the value "0" indicates a length of 1 byte, the value "1" indicates a length of 2 bytes.
2	Counter byte option	This bit is set to "1" to indicate that the "counter byte" field is used in this message. If this bit is set to "0", the "counter byte" field does not exist in this message.
3 and 4	Reserved	0
5	Network ID option	Not supported, must be "0".
6	Encoded data option	Not supported, must be "0".
7	Access flag	This bit indicates the access method of the requested command. "0" = read, "1" = write.
8 to 15	Reserved	0

Address and data range

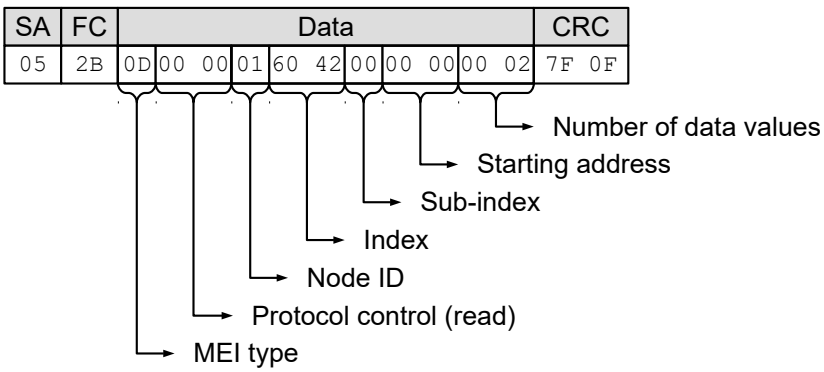
The address and data range is defined in the following table:

Name	Byte size and byte order	Example / range
Node ID	1 byte	01_h to $7F_h$
Index	1 byte, high	0000_h to $FFFF_h$
	1 byte, low	
Subindex	1 byte	00_h to FF_h
Start address	1 byte, high	0000_h to $FFFF_h$
	1 byte, low	
Number of data values	1 byte, high	0000_h to $00FD_h$
	1 byte, low	
Write/read data	n bytes	The data are encoded as described in chapter **General**.

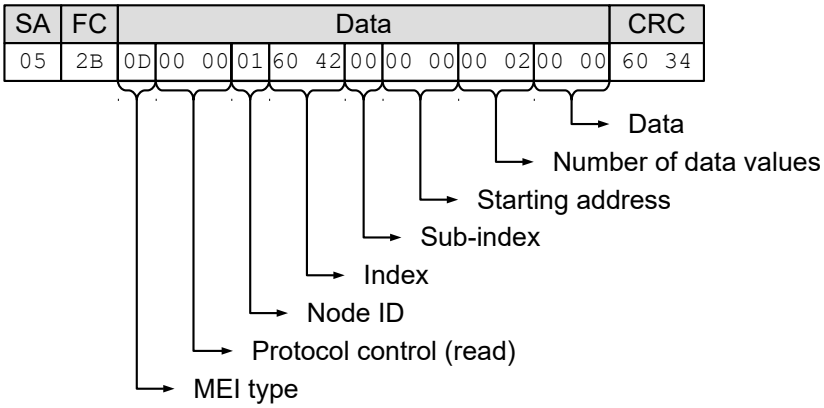
Example:

To read object 6042_h:00_h (16-bit value), the following message must be sent by the master (all values are in hexadecimal notation, the slave ID of the controller is "5").

Request



Response



Shown as an additional example below, a sequence of Modbus messages is sent from the master to the slave to rotate the motor in "Velocity" mode:

Set 6060 = "02_h" (Velocity mode)

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 60 00 00 00 00 01 02	C9 2F

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 60 00 00 00 00 00	A9 89

Set 2031 = 203E8$_h$" (1000 mA)

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 20 31 00 00 00 00 04 E8 03 00 00	C3 53

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 20 31 00 00 00 00 00	E5 CC

Set 6040 = "00_h"

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 02 00 00	1C 2E

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 00	AE E9

Set 6040 = "80$_h$"

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 02 80 00	7D EE

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 00	AE E9

Set 6040 = "06$_h$"

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 02 06 00	1F 8E

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 00	AE E9

Set 6040 = "07$_h$"

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 02 07 00	1E 1E

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 00	AE E9

Set 6040 = "0F$_h$"

Request

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 02 0F 00	19 DE

Response

SA	FC	Data	CRC
05	2B	0D 01 00 01 60 40 00 00 00 00 00	AE E9

Below are two examples for reading an object:

Read 6041_h:00_h

Request

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 41 00 00 00 00 02	7F 3C

Response

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 41 00 00 00 00 02 37 06	B6 13

Read 6061_h:00_h

Request

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 61 00 00 00 00 01	38 5D

Response

SI	FC	Data	CRC
05	2B	0D 00 00 01 60 61 00 00 00 00 01 00	5C D2

Error reaction

In the event of an error, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	6
MEI type	1 byte	$0D_h$
Exception code	1 byte	CE_h
Error code	4 bytes	CANopen error code
CRC	2 bytes	

In the event that the unsupported control option bit is set, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	2 + length of "supported protocol control"
MEI type	1 byte	$0D_h$
Exception code	1 byte	AE_h
Supported protocol control	1 or 2 bytes	See following table
CRC	2 bytes	

Bit	Name	Description
0	"Extended" flag	This bit is used if the object dictionary data set is larger than would fit in a Modbus command. The data set then spans over multiple Modbus messages; each message contains part of the data set. "0" = No multiple message transaction or the end of the multiple message transaction. "1" = Part of a multiple message transaction.
1	Extended protocol control	Length of the protocol control, the value "0" indicates a length of 1 byte, the value "1" indicates a length of 2 bytes.
2	Counter byte option	This bit is set to "1" to indicate that the "counter byte" field is used in this message. If this bit is set to "0", the "counter byte" field does not exist in this message.
3 and 4	Reserved	0
5	Network ID option	Not supported, must be "0".
6	Encoded data option	Not supported, must be "0".
7	Access flag	This bit indicates the access method of the requested command. "0" = read, "1" = write.
8 to 15	Reserved	0

The following example shows an error in the event of a faulty request. The request reads **6061_h**:00 with a length of 2 bytes, but the object has a size of just 1 byte:

Request

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 60 00 00 00 00 02	79 8D

Response

SA	FC	Data	CRC
05	2B	FF 00 06 0D CE 12 00 07 06	AC 3C

9.5.8 FC 101 (65_h) Read complete object dictionary

This function code is used to read out the complete object dictionary.

To start or restart the reading out of the object dictionary, subfunction code 55_h must be sent. This code resets reading out of the object dictionary on object 0000_h. All subsequent object dictionary frames must then contain subfunction code AA_h. At the end, once all objects have been read out, an "Error Response" is generated with the abort code "No data available".

The format of each "read object" is as follows:

Request:

Name	Length	Value / note
Slave address	1 byte	
Function code	1 byte	65_h
Subfunction code	1 byte	55_h or AA_h
Length of the data	1 byte	00_h
CRC	2 bytes	

Response:

Name	Length	Value / note
Slave address	1 byte	65_h
Function code	1 byte	
Subfunction code	1 byte	
Length of the data	1 byte	
n times "object dictionary frame"	1 - 252 bytes	
CRC	2 bytes	

An object dictionary frame consists of the following bytes:

Name		Value / note
Index Low Byte	1 byte	
Index High Byte	1 byte	
Subindex	1 byte	
Number of bytes	1 byte	Number m of the valid data in the data field
Data byte	m-1 byte	

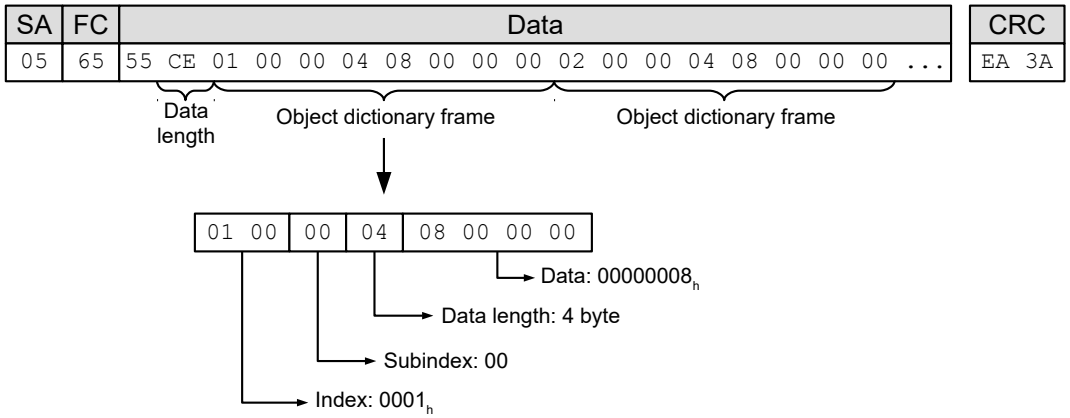
Example

All of the following numerical values are in hexadecimal format. The address of the slave is "5".

Start reading of the object dictionary with request:

SA	FC	Data	CRC
05	65	55 00	2F A7

The response is:



Read out the next part of the object dictionary with the request:

SA	FC	Data	CRC
05	65	AA 00	6E 57

The response is:

SA	FC	Data	CRC
05	65	AA CD 21 00 0A 02 07 00 21 00 0B 02 07 00 21 00 0C 02 ...	NN NN

Repeat reading of the object dictionary with the previous request until the response is an error:

SA	FC	Data	CRC
05	E5	0D	EA 94

Error reaction

In the event of an error, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	6
MEI type	1 byte	$0D_h$
Exception code	1 byte	CE_h
Error code	4 bytes	CANopen error code
CRC	2 bytes	

In the event that the unsupported control option bit is set, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)

Name	Length	Example value
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	2 + length of "supported protocol control"
MEI type	1 byte	$0D_h$
Exception code	1 byte	AE_h
Supported protocol control	1 or 2 bytes	See following table
CRC	2 bytes	

Bit	Name	Description
0	"Extended" flag	This bit is used if the object dictionary data set is larger than would fit in a Modbus command. The data set then spans over multiple Modbus messages; each message contains part of the data set. "0" = No multiple message transaction or the end of the multiple message transaction. "1" = Part of a multiple message transaction.
1	Extended protocol control	Length of the protocol control, the value "0" indicates a length of 1 byte, the value "1" indicates a length of 2 bytes.
2	Counter byte option	This bit is set to "1" to indicate that the "counter byte" field is used in this message. If this bit is set to "0", the "counter byte" field does not exist in this message.
3 and 4	Reserved	0
5	Network ID option	Not supported, must be "0".
6	Encoded data option	Not supported, must be "0".
7	Access flag	This bit indicates the access method of the requested command. "0" = read, "1" = write.
8 to 15	Reserved	0

The following example shows an error in the event of a faulty request. The request reads **6061_h**:00 with a length of 2 bytes, but the object has a size of just 1 byte:

Request

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 60 00 00 00 00 02	79 8D

Response

SA	FC	Data	CRC
05	2B	FF 00 06 0D CE 12 00 07 06	AC 3C

9.5.9 FC 102 (66_h) Read complete array or record

This function code is used to read out the complete array or record from the object dictionary.

To start or restart the reading out of the array, subfunction code 55_h must be sent. This code resets reading out on the object with subindex 00_h. All subsequent requests must then contain subfunction code AA_h. At the end, once all objects have been read out, an "Error Response" is generated.

The format of each "read object" is as follows:

Request:

Name	Length	Value / note
Slave address	1 byte	
Function code	1 byte	66_h
Subfunction code	1 byte	55_h or AA_h
Length of the data	1 byte	00_h
Index of the array to be read	2 bytes	
CRC	2 bytes	

Response:

Name	Length	Value / note
Slave address	1 byte	65_h
Function code	1 byte	
Subfunction code	1 byte	
Length of the data	1 byte	
n times object dictionary frame	1 - 252 bytes	
CRC	2 bytes	

An object dictionary frame consists of the following bytes:

Name		Value / note
Index Low Byte	1 byte	
Index High Byte	1 byte	
Subindex	1 byte	
Number of bytes	1 byte	Number m of the valid data in the data field
Data byte	m-1 byte	

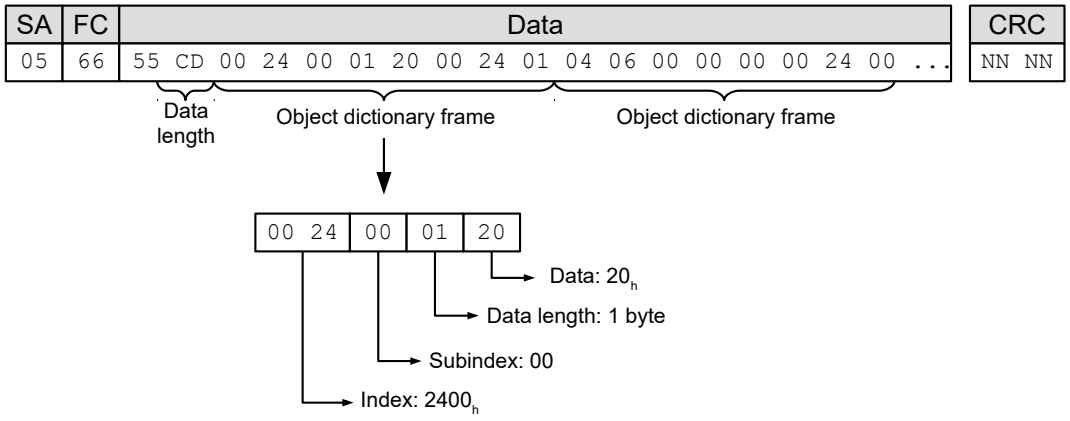
Example

All of the following numerical values are in hexadecimal format; the index of the object that is to be read is 2400_h. The address of the slave is "5"$_h$.

Start reading of the array with request:

SA	FC	Data	CRC
05	66	55 00 24 00	02 8A

The response is:



Error reaction

In the event of an error, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	6
MEI type	1 byte	$0D_h$
Exception code	1 byte	CE_h
Error code	4 bytes	CANopen error code
CRC	2 bytes	

In the event that the unsupported control option bit is set, the following error message is sent:

Name	Length	Example value
Slave address	1 byte	
Function code	1 byte	$2B_h$ +80_h (171_d = 43_d + 128_d) (indicates error)
Modbus exception code	1 byte	FF_h ("extended exception")
Extended exception length	2 bytes	2 + length of "supported protocol control"
MEI type	1 byte	$0D_h$
Exception code	1 byte	AE_h
Supported protocol control	1 or 2 bytes	See following table
CRC	2 bytes	

Bit	Name	Description
0	"Extended" flag	This bit is used if the object dictionary data set is larger than would fit in a Modbus command. The data set then spans over multiple Modbus messages; each message contains part of the data set. "0" = No multiple message transaction or the end of the multiple

Bit	Name	Description
		message transaction. "1" = Part of a multiple message transaction.
1	Extended protocol control	Length of the protocol control, the value "0" indicates a length of 1 byte, the value "1" indicates a length of 2 bytes.
2	Counter byte option	This bit is set to "1" to indicate that the "counter byte" field is used in this message. If this bit is set to "0", the "counter byte" field does not exist in this message.
3 and 4	Reserved	0
5	Network ID option	Not supported, must be "0".
6	Encoded data option	Not supported, must be "0".
7	Access flag	This bit indicates the access method of the requested command. "0" = read, "1" = write.
8 to 15	Reserved	0

The following example shows an error in the event of a faulty request. The request reads **6061**$_h$:00 with a length of 2 bytes, but the object has a size of just 1 byte:

Request

SA	FC	Data	CRC
05	2B	0D 00 00 01 60 60 00 00 00 00 02	79 8D

Response

SA	FC	Data	CRC
05	2B	FF 00 06 0D CE 12 00 07 06	AC 3C

9.6 Process data objects (PDO)

As with CANopen, a process image can be configured for input and output values with Modbus. This image only contains the data values of one or more objects without additional information, such as length, index or subindex. A single message can thereby be used to read or write multiple objects at the same time.

9.6.1 Configuration

The configuration of the image is referred to as "mapping" and is written in the following objects:

- 3502_h for the Modbus Rx (master → slave) PDO mapping
- 3602_h for Modbus Tx (slave → master) PDO mapping

Both objects contain an array of 16 entries each. Subindex 00 specifies the number of valid entries here.

Objects 3502_h and 3602_h can be written with messages with Modbus function code $2B_h$.

9.6.2 Transfer

The data are written sequentially in the message without gaps and alignment.

If alignment is required (e.g., 16-bit alignment), additional "dummy objects" can be incorporated in the message. Dummy objects are only ever transferred with the data value "0". These objects are listed in the following table.

Index	Data type
0002_h	Signed integer (8 bit)
0003_h	Signed integer (16 bit)
0004_h	Signed integer (32 bit)
0005_h	Unsigned integer (8 bit)
0006_h	Unsigned integer (16 bit)
0007_h	Unsigned integer (32 bit)

Mapping is as follows:

- The PDO RX image begins at Modbus register address 6000_d (1770_h).
- The PDO TX image begins at Modbus register address 5000_d (1388_h).

Read/write access can be performed simultaneously with function code 17_h or with the 03_h, 04_h, 06_h, 10_h commands on the respective RX/TX images.

Example

The following objects are to be set in the mapping:

- **3602_h**:00_h = "6_h" (6 values are mapped)
- **3602_h**:01_h = "60410010_h" (object **6041_h**:00_h, length 16 bits is mapped)
- **3602_h**:02_h = "00050008_h" (dummy object 0005_h:00_h, length 8 bits is mapped)
- **3602_h**:03_h = "60610008_h" (object **6061_h**:00_h, length 8 bits is mapped)
- **3602_h**:04_h = "60640020_h" (object **6064_h**:00_h, length 32 bits is mapped)
- **3602_h**:05_h = "60440010_h" (object **6044_h**:00_h, length 16 bits is mapped)
- **3602_h**:06_h = "$60FD0020_h$" (object **$60FD_h$**:00_h, length 32 bits is mapped)

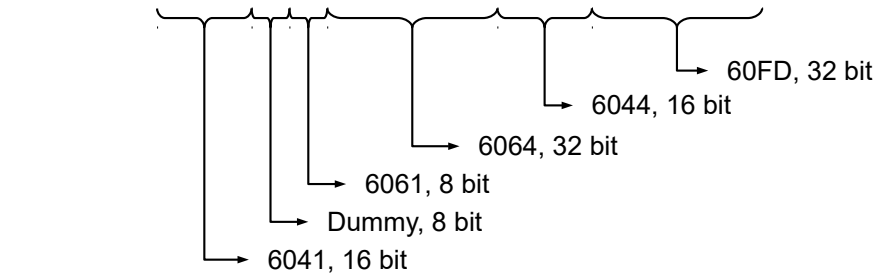
After the mapping for object **6061_h**:00_h, a dummy object is inserted so that the next object **6064_h**:00_h can be aligned to 32 bit.

Rx message: The master sends the slave the following message:

SA	FC	Data	CRC
05	04	13 88 00 07	34 E2

Tx message: The slave sends following response to the master:

SA	FC	Data							CRC
05	04	0E	06 40	00	02	00 00 00 00	00 00	00 00 00 00	1C 98



9.7 NanoJ objects

NanoJ objects **2400_h** NanoJ Input and **2500_h** (NanoJ Output) are, like the process image, mapped to the Modbus register:

- **2500_h** with 32 x 32 bit values is mapped to the Modbus register address beginning with 2000_d ($BB8_h$) and can only be read in this way.

- **2400**$_h$ with 32 x 32 bit values is mapped to the Modbus register address beginning with 3000$_d$ (7D0$_h$) and can only be written in this way.

To access, commands with function codes 03$_h$, 04$_h$, 10$_h$ and 17$_h$ can be used. For purposes of data consistency, the restriction that the address must be 32-bit aligned and that at least 32 bits must always be written during a write operation applies.

Example

Request: The master sends the slave the following message:

SA	FC	Data	CRC
05	17	07 D0 00 08 0B B8 00 08 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	41 21

Reply: The slave sends the master the following response:

SA	FC	Data	CRC
05	17	10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	50 9D

10 Programming with *NanoJ*

NanoJ is a programming language similar to *C* or *C++*. NanoJ is integrated in the *Plug & Drive Studio* software. You can find further information in document *Plug & Drive Studio: Quick Start Guide* at **us.nanotec.com**.

10.1 NanoJ program

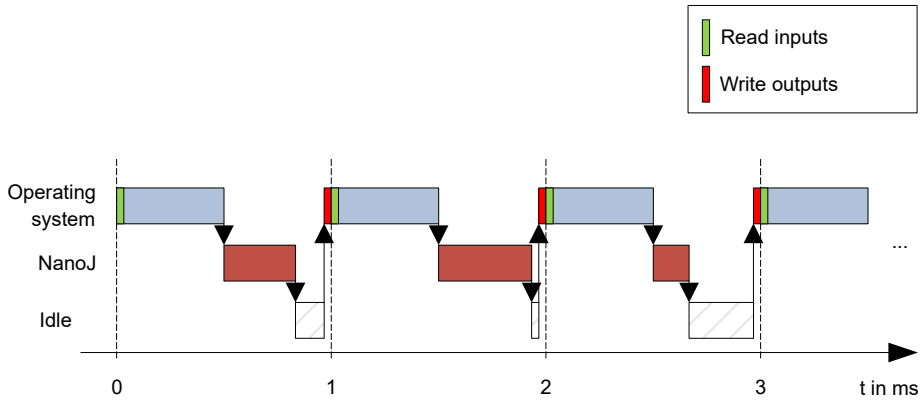
A *NanoJ program* makes a protected runtime environment available within the firmware. Here, the user can create his own processes. These can then trigger functions in the controller by, for example, reading or writing entries in the object dictionary.

Through the use of protective mechanisms, a *NanoJ program* is prevented from crashing the firmware. In the worst case, the execution is interrupted with an error code stored in the object dictionary.

If the *NanoJ program* was loaded on the controller, it is automatically executed after the controller is switched on or restarted.

10.1.1 Available computing time

A *NanoJ program* receives computing time cyclically in a 1 ms clock (see following figure). Because computing time is lost through interrupts and system functions of the firmware, only approx. 30% – 50% of computing time is available to the user program (depending on operating mode and application). In this time, the user program must run through the cycle and either complete the cycle or yield the computing time by calling the `yield()` function. In the former case, the user program is restarted with the start of the next 1 ms cycle; the latter results in the program being continued on the next 1 ms cycle with the command that follows the `yield()` function.



If the *NanoJ program* needs more time than was allotted, it is ended and an error code set in the object dictionary.

Tip

When developing user programs, the runtime behavior must be carefully examined, especially for more time-intensive tasks. For example, it is therefore recommended that tables be used instead of calculating a sine value using a `sin` function.

Note

If the *NanoJ program* does not yield the computing time after too long a time, it is ended by the operating system. In this case, the number `4` is entered in the statusword for object 2301_h; in the error register for object 2302_h, the number `5` (timeout) is noted, see **2301h NanoJ Status** and **2302h NanoJ Error Code**.

10.1.2 Sandbox

Using processor-specific features, a so-called *sandbox* is generated. When used in the sandbox, a user program can only access specially assigned memory areas and system resources. For example, an attempt to directly write to a processor IO register is acknowledged with an *MPU Fault* and the user program terminated with the corresponding error code in the object dictionary.

10.1.3 NanoJ program – communication possibilities

A *NanoJ program* has a number of possibilities for communicating with the controller:

- Read and write OD values using PDO mapping
- Directly read and write OD values using system calls
- Call other system calls (e.g., write debug output)

The OD values of the user program are made available in the form of variables via *PDO mapping*. Before a user program receives the 1 ms time slot, the firmware transfers the values from the object dictionary to the variables of the user program. As soon as the user program receives computing time, it can manipulate these variables as regular C variables. At the end of the time slot, the new values are then automatically copied by the firmware back to the respective OD entries.

To optimize the performance, three types of mapping are defined: input, output, and input/output (In, Out, InOut).

- *Input mappings* can only be read; they are not transferred back to the object dictionary.
- *Output mappings* can only be written.
- *Input/output mappings*, on the other hand, can both be read and written.

The set mappings can be read and checked via the GUI for objects 2310_h, 2320_h, and 2330_h. Up to 16 entries are allowed for each mapping.

Whether a variable is stored in the input, output or data range is controlled in *NanoJEasy* via the specification of the *linker section*.

10.1.4 Executing a NanoJ program

When executing a cycle, the *NanoJ program* essentially consists of the following three steps with respect to the PDO mapping:

1. Read values from the object dictionary and copy them to the input and output areas
2. Execute a user program
3. Copy values from the output and input areas back to the object dictionary

The configuration of the copy processes is based on the CANopen standard.

In addition, values of the object dictionary can be accessed via system calls. This is generally slower; mappings are therefore to be preferred. The number of mappings is limited (16 entries each in In/Out/InOut).

Tip

Nanotec recommends: Map OD entries that are used and changed frequently and use system calls to access OD entries that are used less frequently.

A list of available system calls can be found in chapter **System calls in a NanoJ program**.

Tip

Nanotec recommends accessing a given OD value either by mapping or using a system call with `od_write()`. If both are used simultaneously, the system call has no effect.

10.1.5 NanoJ program – OD entries

The *NanoJ program* is controlled and configured in object range 2300_h to 2330_h (see **2300h NanoJ Control**).

OD-Index	Name and description
2300_h	**2300h NanoJ Control**
2301_h	**2301h NanoJ Status**
2302_h	**2302h NanoJ Error Code**
2310_h	**2310h NanoJ Input Data Selection**
2320_h	**2320h NanoJ Output Data Selection**
2330_h	**2330h NanoJ In/output Data Selection**

Example:

To select and start the *TEST1.USR* user program, the following sequence can, for example, be used:

- Check entry **2302_h** for error code.
- If no error:
  Start the *NanoJ program* by writing object **2300_h**, bit 0 = "1".

Note

It can take up to 200 ms for the NanoJ program to start.

- Check entry **2302_h** for error code and object **2301_h**, bit 0 = "1".
- Rename file `TEST1.USR` with `vmmcode.usr`.
- Copy file `vmmcode.usr` to the controller via USB.
- Start the NanoJ program by writing object **2300_h**, bit 0 = "1" or by restarting the controller.
- Check entry **2302_h** for error code and object **2301_h**, bit 0 = "1" (NanoJ program running).

Note

Due to limitations in the USB implementation, file "VMMCODE.USR" is, following a restart of the controller, set to a size of 16 kB and the creation date set to 13.03.2012.

To stop a running program: write entry **2300_h** with bit 0 value = "0".

10.1.6 Structure of a NanoJ program

A user program consists of at least two instructions:

- the preprocessor instruction `#include "wrapper.h"`
- the `void user(){}` function

The code to be executed can be stored in the `void user()` function.

Note

The file names of the user programs must not be longer than eight characters plus three characters in the suffix; file name `main.cpp` is permissible, file name `aLongFileName.cpp` is not permissible.

Note

In the *NanoJ program*, only global variables are permitted and they may only be initialized within code. It then follows:

- No `new` operator
- No constructors
- No initialization of global variables outside of code

Examples:

The global variable is to be initialized within the `void user()` function:

```
unsigned int i;
void user(){
 i = 1;
 i += 1;
}
```

The following assignment is not correct:

```
unsigned int i = 1;
 void user() {
 i += 1;
}
```

10.1.7 NanoJ program example

The example shows the programming of a square wave signal in object 2500_h:01_h.

```
// file main.cpp
map S32 outputReg1 as inout 0x2500:1
#include "wrapper.h"

// user program
void user()
{
  U16 counter = 0;
  while( 1 )
  {
    ++counter;

    if( counter < 100 )
     InOut.outputReg1 = 0;
    else if( counter < 200 )
      InOut.outputReg1 = 1;
    else
      counter = 0;

    // yield() 5 times (delay 5ms)
    for(U08 i = 0; i < 5; ++i )
      yield();
  }
```

```
}// eof
```

You can find other examples at **us.nanotec.com**

10.2 Mapping in the NanoJ program

With this method, a variable in the *NanoJ program* is linked directly with an entry in the object dictionary. The creation of the mapping must be located at the start of the file here, even before the `#include "wrapper.h"` instruction. A comment is permitted above the mapping.

Tip

Nanotec recommends:

- Use mapping if you need to access an object in the object dictionary frequently, e.g., *controlword* 6040_h or *statusword* 6041_h.
- The `od_write()` and `od_read()` functions are better suited for accessing objects a single time, see **Accessing the object dictionary**.

10.2.1 Declaration of the mapping

The declaration of the mapping is structured as follows:

```
map <TYPE> <NAME> as <input|output|inout> <INDEX>:<SUBINDEX>
```

Where:

- `<TYPE>`

  The data type of the variable; U32, U16, U08, S32, S16 or S08.
- `<NAME>`
  The name of the variable as it is used in the user program.
- `<input|output|inout>`

  The read and write permission of a variable: a variable can be declared as an `input`, `output` or `inout`. This defines whether a variable is readable (`input`), writable (`output`) or both (`inout`) and the structure by means of which it must be addressed in the program.
- `<INDEX>:<SUBINDEX>`

  Index and subindex of the object to be mapped in the object dictionary.

Each declared variable is addressed in the user program via one of the three structures: *In*, *Out* or *InOut* depending on the defined write and read direction.

10.2.2 Example of mapping

Example of a mapping and the corresponding variable accesses:

```
map U16 controlWord as output 0x6040:00
map U08 statusWord as input 0x6041:00
map U08 modeOfOperation as inout 0x6060:00

#include "wrapper.h"

void user()
{
  [...]
  Out.controlWord = 1;
```

```
  U08 tmpVar = In.statusword;
  InOut.modeOfOperation = tmpVar;
  [...]
}
```

10.2.3 Possible error at `od_write()`

A possible source of errors is a write access with the `od_write()` function (see **System calls in a NanoJ program**) of an object in the object dictionary that was simultaneously created as mapping. The code listed in the following is incorrect:

```
map U16 controlWord as output 0x6040:00
#include " wrapper.h"
void user()
{
 [...]
  Out.controlWord = 1;
  [...]
  od_write(0x6040, 0x00, 5 ); // der Wert wird durch das Mapping
 überschrieben
  [...]
}
```

The line with the `od_write(0x6040, 0x00, 5 );` command has no effect. As described in the introduction, all mappings are copied to the object dictionary at the end of each millisecond.

This results in the following sequence:

1. The `od_write` function writes the value `5` in object 6040_h:00_h.
2. At the end of the 1 ms cycle, the mapping is written that also specifies object 6040_h:00_h, however, with the value `1`.
3. From the perspective of the user, the `od_write` command thus serves no purpose.

10.3 System calls in a NanoJ program

With system calls, it is possible to call up functions integrated in the firmware directly from a user program. Because direct code execution is only possible in the protected area of the sandbox, this is implemented via so-called *Cortex-Supervisor-Calls* (Svc Calls). An interrupt is triggered when the function is called. The firmware thus has the possibility of temporarily allowing code execution outside of the sandbox. Developers of user programs do not need to worry about this mechanism – for them, the system calls can be called up like normal C functions. Only the *wrapper.h* file needs to be integrated as usual.

10.3.1 Accessing the object dictionary

void **od_write** (U32 index, U32 subindex, U32 value)

This function writes the transferred value to the specified location in the object dictionary.

index	Index of the object to be written in the object dictionary
subindex	Subindex of the object to be written in the object dictionary
value	Value to be written

Note

It is highly recommended that the processor time be passed on with `yield()` after calling a `od_write()`. The value is immediately written to the OD. For the firmware to be able to trigger actions that are dependent on this, however, it must receive computing time. This, in turn, means that the user program must either be ended or interrupted with `yield()`.

U32 **od_read** (U32 index, U32 subindex)

This function reads the value at the specified location in the object dictionary and returns it.

index	Index of the object to be read in the object dictionary
subindex	Subindex of the object to be read in the object dictionary
Output value	Content of the OD entry

Note

Active waiting for a value in the object dictionary should always be associated with a `yield()`.

Example

```
while (od_read(2400,2) != 0) // wait until 2400:2 is set
{ yield(); }
```

10.3.2 Process control

```
void yield()
```

This function returns the processor time to the operating system. In the next time slot, the program continues at the location after the call.

```
void sleep (U32 ms)
```

This function returns the processor time to the operating system for the specified number of milliseconds. The user program is then continued at the location after the call.

ms	Time to be waited in milliseconds

11 Description of the object dictionary

11.1 Overview

This chapter contains a description of all objects.

You will find information here on:

- Functions
- Object descriptions ("Index")
- Value descriptions ("Subindices")
- Descriptions of bits
- Description of the object

11.2 Structure of the object description

The description of the object entries always has the same structure and usually consists of the following sections:

Function

The function of the object dictionary is briefly described in this section.

Object description

This table provides detailed information on the data type, preset values and similar. An exact description can be found in section "**Object description**"

Value description

This table is only available with the "Array" or "Record" data type and provides exact information about the sub-entries. A more exact description of the entries can be found in section "**Value description**"

Description

Here, more exact information on the individual bits of an entry is provided or any compositions explained. A more exact description can be found in section "**Description**"

11.3 Object description

The object description consists of a table that contains the following entries:

Index

Designates the object index in hexadecimal notation.

Object name

The name of the object.

Object Code

The type of object. This can be one of the following entries:

- VARIABLE: In this case, the object consists of only a variable that is indexed with subindex 0.
- ARRAY: These objects always consists of a subindex 0 – which specifies the number of sub-entries – and the sub-entries themselves, beginning with index 1. The data type within an array never changes, i.e., sub-entry 1 and all subsequent entries are always of the same data type.
- ARRAY: These objects always consists of a subindex 0 – which specifies the number of sub-entries – and the sub-entries themselves, beginning with index 1. Unlike an ARRAY, the data type of the sub-entries can vary. This means that, e.g., sub-entry 1 may be of a different data type than sub-entry 2.

- VISIBLE_STRING: The object describes a character string coded in ASCII. The length of the string is specified in subindex 0; the individual characters are stored beginning in subindex 1. These character strings are **not** terminated by a null character.

Data type

The size and interpretation of the object is specified here. The following notation is used for the "VARIABLE" object code:

- A distinction is made between entries that are signed; these are designated with the prefix "SIGNED". For entries that are unsigned, the prefix "UNSIGNED" is used.
- The size of the variable in bits is placed before the prefix and can be 8, 16 or 32.

Savable

Described here is whether this object is savable and, if so, in which category.

Firmware version

The firmware version beginning with which the object is available is entered here.

Change history (ChangeLog)

Any changes to the object are noted here.

There are also the following table entries for the "VARIABLE" data type:

Access

The access restriction is entered here. The following restrictions are available:

- "read/write": The object can both be read as well as written
- "read only": The object can only be read from the object dictionary. It is not possible to set a value.

PDO mapping

Some bus systems, such as CANopen or EtherCAT, support PDO mapping. Described in this table entry is whether the object can be inserted into a mapping and, if so, into which. The following designations are available here:

- "no": The object may not be entered in a mapping.
- "TX-PDO": The object may be entered in an RX mapping.
- "RX-PDO": The object may be entered in a TX mapping.

Allowed values

In some cases, only certain values may be written in the object. If this is the case, these values are listed here. If there are no restrictions, the field is empty.

Preset value

To bring the controller to a secured state when switching on, it is necessary to preset a number of objects with values. The value that is written in the object when the controller is started is noted in this table entry.

11.4 Value description

Note

For the sake of clarity, a number of subindices are grouped together if the entries all have the same name.

Listed in the table with the "Value description" heading are all data for sub-entries with subindex 1 or higher. The table contains the following entries:

Subindex

Number of the currently written sub-entry.

Name

Name of the sub-entry.

Data type

The size and interpretation of the sub-entry is specified here. The following notation always applies here:

- A distinction is made between entries that are signed; these are designated with the prefix "SIGNED". For entries that are unsigned, the prefix "UNSIGNED" is used.
- The size of the variable in bits is placed before the prefix and can be 8, 16 or 32.

Access

The access restriction for the sub-entry is entered here. The following restrictions are available:

- "read/write": The object can both be read as well as written
- "read only": The object can only be read from the object dictionary. It is not possible to set a value.

PDO mapping

Some bus systems, such as CANopen or EtherCAT, support PDO mapping. Described in this table entry is whether the sub-entry can be inserted into a mapping and, if so, into which. The following designations are available here:

- "no": The object may not be entered in a mapping.
- "TX-PDO": The object may be entered in an RX mapping.
- "RX-PDO": The object may be entered in a TX mapping.

Allowed values

In some cases, only certain values may be written in the sub-entry. If this is the case, these values are listed here. If there are no restrictions, the field is empty.

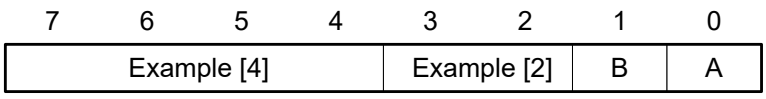
Preset value

To bring the controller to a secured state when switching on, it is necessary to preset a number of sub-entries with values. The value that is written in the sub-entry when the controller is started is noted in this table entry.

11.5 Description

This section may be present if use requires additional information. If individual bits of an object or sub-entry have different meaning, diagrams as shown in the following example are used.

Example: The object is 8 bits in size; bit 0 and bit 1 have different functions. Bits 2 and 3 are grouped into one function; the same applies for bits 4 to 7.



Example [4]

Description of bit 4 up to and including bit 7; these bits are logically related. The 4 in square brackets specifies the number of related bits. A list with possible values and their description is often attached at this point.

Example [2]

Description of bits 3 and 2; these bits are logically related. The 2 in square brackets specifies the number of related bits.

- Value 00_b: The description here applies if bit 2 and bit 3 are "0".
- Value 01_b: The description here applies if bit 2 is "0" and bit 3 is "1".
- Value 10_b: The description here applies if bit 2 is "1" and bit 3 is "0".
- Value 11_b: The description here applies if bit 2 and bit 3 are "1".

B

Description of bit B; no length is specified for a single bit.

A

Description of bit A; bits with a gray background are not used.

1000h Device Type

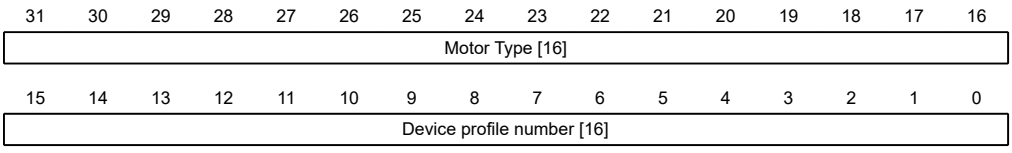
Function

Describes the controller type.

Object description

Index	1000_h
Object name	Device Type
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	00060192_h
Firmware version	FIR-v1426
Change history	

Description



Motor Type[16]

Describes the supported motor type. The following values are possible:

- Bit 23 to bit 16: Value "1": Servo drive
- Bit 23 to bit 16: Value "2": Stepper motor

Device profile number[16]

Describes the supported CANopen standard.

Values:

0192_h or 0402_d (preset value): The CiA 402 standard is supported.

1001h Error Register

Function

Error register: The corresponding error bit is set in case of an error. If the error no longer exists, it is deleted automatically.

Object description

Index	1001_h
Object name	Error Register
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00_h
Firmware version	FIR-v1426
Change history	

Description

7	6	5	4	3	2	1	0
MAN	RES	PROF	COM	TEMP	VOL	CUR	GEN

GEN
General error

CUR
Current

VOL
Voltage

TEMP
Temperature

COM
Communication

PROF
Relates to the device profile

RES
Reserved, always "0"

MAN
Manufacturer-specific: The motor turns in the wrong direction.

1003h Pre-defined Error Field

Function

This object contains an error stack with up to eight entries.

Object description

Index	1003_h
Object name	Pre-defined Error Field
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Number Of Errors
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Standard Error Field
Data type	UNSIGNED32

Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	Standard Error Field
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	

Preset value	00000000_h

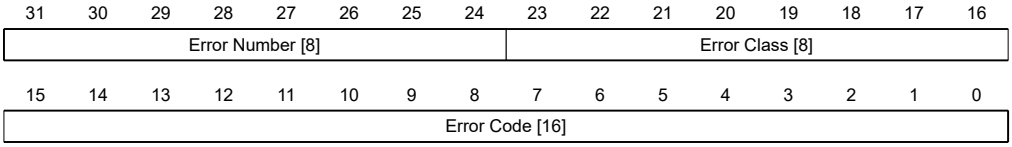
Description

General function

If a new error occurs, it is entered in subindex 1. The already existing entries in subindices 1 to 7 are moved back one position. The error in subindex 7 is thereby removed.

The number of errors that have already occurred can be read from the object with subindex 0. If no error is currently entered in the error stack, it is not possible to read one of the eight subindices 1–8 and an error (abort code = 08000024_h) is sent in response. If a "0" is written in subindex 0, counting starts again from the beginning.

Bit description



Error Number [8]

This can be used to pinpoint the cause of the error. The meaning of the number can be found in the following table.

Error number	Description
0	Watchdog-Reset
1	Input voltage too high
2	Output current too high
3	Input voltage too low
4	Error at fieldbus
5	Motor turns – in spite of active block – in the wrong direction
6	CANopen only: NMT master takes too long to send nodeguarding request
7	Encoder error due to electrical fault or defective hardware
8	Encoder error; index not found during the auto setup
9	Error in the AB track
10	Positive limit switch and tolerance zone exceeded
11	Negative limit switch and tolerance zone exceeded
12	Device temperature above 80°C
13	The values of object **6065_h** (Following Error Window) and object **6066_h** (Following Error Time Out) were exceeded; a fault was triggered.
14	Nonvolatile memory full; controller must be restarted for cleanup work.
15	Motor blocked
16	Nonvolatile memory damaged; controller must be restarted for cleanup work.
17	CANopen only: Slave took too long to send PDO messages.
18	Hall sensor faulty
19	CANopen only: PDO not processed due to a length error
20	CANopen only: PDO length exceeded
21	Nonvolatile memory full; controller must be restarted for cleanup work.
22	Rated current must be set ($203B_h$:01_h)

Error number	Description
23	Encoder resolution, number of pole pairs and some other values are incorrect.
24	Motor current is too high, adjust the PI parameters.
25	Internal software error, generic
26	Current too high at digital output
27	CANopen only: Unexpected sync length
28	EtherCAT only: The motor was stopped because EtherCAT switched state from OP to either SafeOP or PreOP without first stopping the motor.

Error Class[8]

This byte is identical to object **1001**$_h$

Error Code[16]

Refer to the following table for the meaning of the bytes.

Error Code	Description
1000$_h$	General error
2300$_h$	Current at the controller output too large
3100$_h$	Overvoltage/undervoltage at controller input
4200$_h$	Temperature error within the controller
6010$_h$	Software reset (watchdog)
6100$_h$	Internal software error, generic
6320$_h$	Rated current must be set (203B$_h$:01$_h$)
7121$_h$	Motor blocked
7305$_h$	Incremental encoder or Hall sensor faulty
7600$_h$	Nonvolatile memory full or corrupt; restart the controller for cleanup work
8000$_h$	Error during fieldbus monitoring
8130$_h$	CANopen only: "Life Guard" error or "Heartbeat" error
8200$_h$	CANopen only: Slave took too long to send PDO messages.
8210$_h$	CANopen only: PDO was not processed due to a length error
8220$_h$	CANopen only: PDO length exceeded
8611$_h$	Position monitoring error: Following error too large
8612$_h$	Position monitoring error: Limit switch and tolerance zone exceeded
9000$_h$	EtherCAT: Motor running while EtherCAT changes from OP -> SafeOp, PreOP, etc.

1005h COB-ID Sync

Function

Defines the COB-ID of the SYNC message for the SYNC protocol. The value must correspond to an 11-bit-long CAN-ID and is evaluated when the controller is restarted or on a Reset Communication command. The generation of sync messages is not supported.

Note
If the CAN-ID is not to correspond to the default value of 80_h, it must be ensured that only not-yet unassigned or reserved CAN-IDs are used.

Object description

Index	1005_h
Object name	COB-ID Sync
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000080_h
Firmware version	FIR-v1426
Change history	

1007h Synchronous Window Length

Function

This object contains the length of the time window in microseconds for synchronous PDOs. If the synchronous time window has elapsed, all synchronous TxPDOs are rejected and an EMCY message sent. The RxPDOs are also rejected up to the next SYNC message.

The value "0" switches off the time window, thereby allowing the PDOs to be sent at any time.

This object is only available in device variants with CANopen connection.

Object description

Index	1007_h
Object name	Synchronous Window Length
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

1008h Manufacturer Device Name

Function

Contains the device name as character string.

Object description

Index	1008_h
Object name	Manufacturer Device Name
Object Code	VARIABLE
Data type	VISIBLE_STRING
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	• CL3-E-1-0F: CL3-E-1-0F • CL3-E-2-0F: CL3-E-2-0F
Firmware version	FIR-v1426
Change history	

1009h Manufacturer Hardware Version

Function

This object contains the hardware version as character string.

Object description

Index	1009_h
Object name	Manufacturer Hardware Version
Object Code	VARIABLE
Data type	VISIBLE_STRING
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	0
Firmware version	FIR-v1426
Change history	

100Ah Manufacturer Software Version

Function

This object contains the software version as character string.

Object description

Index	$100A_h$
Object name	Manufacturer Software Version
Object Code	VARIABLE
Data type	VISIBLE_STRING

Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	FIR-v1650-B527540
Firmware version	FIR-v1426
Change history	

100Ch Guard Time

Function

Object $100C_h$ multiplied by object **100Dh Live Time Factor** yields the so-called lifetime for the Lifeguarding / Nodeguarding protocol. The value is specified in milliseconds. See also **Nodeguarding**.

Note

The *Heartbeat protocol* has a higher priority than *Nodeguarding*. If both protocols are activated simultaneously, the Node Guarding Timer is suppressed, but no EMCY message is sent either.

Object description

Index	$100C_h$
Object name	Guard Time
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

100Dh Live Time Factor

Function

This object is a multiplier which, multiplied by object **$100C_h$**, yields the time window for the *Nodeguarding* protocol in milliseconds. See also **Nodeguarding**.

Note

The *Heartbeat protocol* has a higher priority than *Nodeguarding*. If both protocols are activated simultaneously, the Node Guarding Timer is suppressed, but no EMCY message is sent either.

This object is only available in device variants with CANopen connection.

Object description

Index	$100D_h$
Object name	Live Time Factor
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h
Firmware version	FIR-v1426
Change history	

1010h Store Parameters

Function

This object is used to start the saving of objects.

Note
As an alternative, objects can also be set and saved using the configuration file. Note that this file has higher priority. Objects that are saved both with the mechanism described here as well as in the configuration file take the value of the configuration file.

Object description

Index	1010_h
Object name	Store Parameters
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object name" entry changed from "Store Parameter" to "Store Parameters". Firmware version FIR-v1436: The number of entries was changed from 3 to 4. Firmware version FIR-v1512: The number of entries was changed from 4 to 5. Firmware version FIR-v1540: The number of entries was changed from 5 to 7.

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	06_h

Subindex	01_h
Name	Save All Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	02_h
Name	Save Communication Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	03_h
Name	Save Application Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	04_h
Name	Save Customer Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	05_h
Name	Save Drive Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	00000001_h

Subindex	06_h
Name	Save Tuning Parameters To Non-volatile Memory
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Description

Each subindex of the object stands for a certain memory class. By reading out the entry, it is possible to determine whether (value "1") or not (value="0") this memory category can be saved.

To start the save process of a memory category, value "65766173_h" must be written in the corresponding subindex. This corresponds to the decimal of 1702257011_d or the ASCII string `save`. As soon as the saving process is completed, the save command is again overwritten with the value "1", since saving is possible again.

For a detailed description, see chapter **Saving objects**.

1011h Restore Default Parameters

Function

This object can be used to reset all or part of the object dictionary to the default values.

Object description

Index	1011_h
Object name	Restore Default Parameters
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "Restore Default Parameter" to "Restore Default Parameters". Firmware version FIR-v1436: The number of entries was changed from 2 to 4. Firmware version FIR-v1512: The number of entries was changed from 4 to 5. Firmware version FIR-v1512: "Name" entry changed from "Restore The Comm Default Parameters" to "Restore Communication Default Parameters". Firmware version FIR-v1512: "Name" entry changed from "Restore The Application Default Parameters" to "Restore Application Default Parameters".

	Firmware version FIR-v1540: The number of entries was changed from 5 to 7.

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	06_h

Subindex	01_h
Name	Restore All Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	02_h
Name	Restore Communication Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	03_h
Name	Restore Application Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	04_h
Name	Restore Customer Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000001_h

Subindex	05_h
Name	Restore Drive Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	06_h
Name	Restore Tuning Default Parameters
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

If the value $64616F6C_h$ (or 1684107116_d or ASCII `load`) is written in this object, part or all of the object dictionary is reset to the default values. The subindex that is used decides which range is reset.

For a detailed description, see chapter **Discarding the saved data**.

1014h COB-ID EMCY

Function

This object describes the COB-ID of the "Emergency Service" under CANopen.

With the *Valid Bit* (bit 31) = "1", the **Emergency Service** can be deactivated; the service is active with the value "0". Every time the controller is restarted, bits 0 to 30 are generated according to the node-ID.

Object description

Index	1014_h
Object name	COB-ID EMCY
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: "Access" table entry for subindex 00 changed from "read only" to "read/write".

1017h Producer Heartbeat Time

Function

This object defines the cycle time of the *heartbeat* of the "Network Management" CANopen service in milliseconds. If the object is set to the value 0, no heartbeat message is sent. See also **Heartbeat**.

Note

The *Heartbeat protocol* has a higher priority than *Nodeguarding*. If both protocols are activated simultaneously, the Node Guarding Timer is suppressed, but no EMCY message is sent either.

This object is only available in device variants with CANopen connection.

Object description

Index	1017_h
Object name	Producer Heartbeat Time
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

1018h Identity Object

Function

This object returns general information on the device, such as manufacturer, product code, revision and serial number.

Tip

Have these values ready in the event of service inquiries.

Object description

Index	1018_h
Object name	Identity Object
Object Code	RECORD
Data type	IDENTITY
Savable	no
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	04_h

Subindex	01_h
Name	Vendor-ID
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	$0000026C_h$

Subindex	02_h
Name	Product Code
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	• CL3-E-1-0F: $0000000D_h$ • CL3-E-2-0F: $0000000E_h$

Subindex	03_h
Name	Revision Number
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	06720000_h

Subindex	04_h
Name	Serial Number
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

1020h Verify Configuration

Function

This object indicates the date and time that the configuration was stored.

A network configuration tool or a CANopen manager can use this object to verify the configuration after a reset and, if necessary, perform a new configuration.

The tool must set the date and time before the storage mechanism is started (see chapter **Saving objects**).

Object description

Index	1020_h
Object name	Verify Configuration
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: verify
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Configuration Date
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Configuration Time
Data type	UNSIGNED32
Access	read / write

PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Subindex 01_h (configuration date) is to contain the number of days since 1 January 1984.

Subindex 02_h (configuration time) is to contain the number of milliseconds since midnight.

1029h Error Behavior

Function

This object is used to define what the *NMT state* of the controller should be in case of an error. See also chapter **Network Management (NMT)**.

Object description

Index	1029_h
Object name	Error Behavior
Object Code	ARRAY
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1738-B501312
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Communication Error
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	02_h
Name	Internal Device Error
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Description

The subindices have the following function:

- 01_h: This subindex is used to define how to respond in case of a communication error:
  - Value "$00"_h$: The controller switches to the *Pre-Operational* state (if previously in the *Operational* state).
  - Value "$01"_h$: The controller does not change state.
  - Value "$02"_h$: The controller switches to the *Stopped* state.
- 02_h: This subindex is used to define how to respond to the remaining errors (except for communication errors):
  - Value "$00"_h$: The controller switches to the *Pre-Operational* state (if previously in the *Operational* state).
  - Value "$01"_h$: The controller does not change state.
  - Value "$02"_h$: The controller switches to the *Stopped* state.

1400h Receive PDO 1 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1600_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1400_h
Object name	Receive PDO 1 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	

Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1401h Receive PDO 2 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1601_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1401_h
Object name	Receive PDO 2 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1402h Receive PDO 3 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1602_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1402_h
Object name	Receive PDO 3 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1403h Receive PDO 4 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1603_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1403_h
Object name	Receive PDO 4 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication

Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1404h Receive PDO 5 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1604_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1404_h

Object name	Receive PDO 5 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	80000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1405h Receive PDO 6 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1605_h.
See chapter **Process Data Object (PDO)**.

Object description

Index	1405_h
Object name	Receive PDO 6 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	80000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1406h Receive PDO 7 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1606_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1406_h
Object name	Receive PDO 7 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	80000000_h

Subindex	02_h

Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1407h Receive PDO 8 Communication Parameter

Function

Contains the communication parameters for the receiving-side mapping (RX-PDO) in object 1607_h. See chapter **Process Data Object (PDO)**.

Object description

Index	1407_h
Object name	Receive PDO 8 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	COB-ID

Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	80000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

For details see chapter on **configuring the RX-PDO mapping**.

1600h Receive PDO 1 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 1). The PDO was previously configured via **1400h Receive PDO 1 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1600_h
Object name	Receive PDO 1 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1600h Drive Control" to "1600h Receive PDO 1 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Drive Control" to "Receive PDO 1 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60400010_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60600008_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	32020020_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

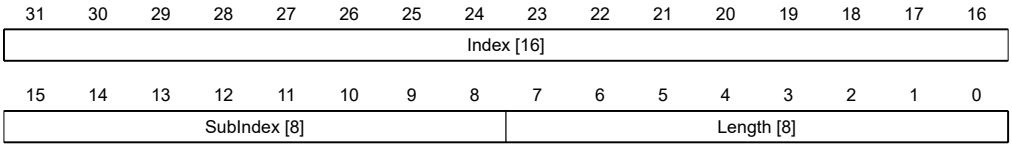
Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1601h Receive PDO 2 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 2). The PDO was previously configured via **1401h Receive PDO 2 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1601_h
Object name	Receive PDO 2 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1601h Positioning Control" to "1601h Receive PDO 2 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Positioning Control" to "Receive PDO 2 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$607A0020_h$

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60810020_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

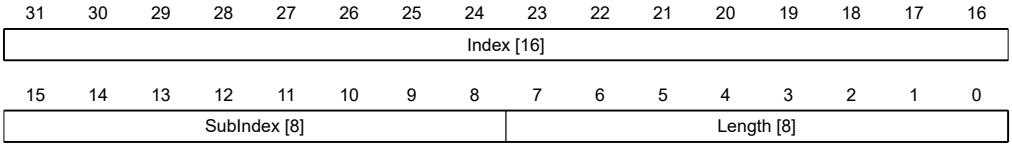
Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1602h Receive PDO 3 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 3). The PDO was previously configured via **1402h Receive PDO 3 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1602_h
Object name	Receive PDO 3 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1602h Velocity Control" to "1602h Receive PDO 3 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Velocity Control" to "Receive PDO 3 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60420010_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1603h Receive PDO 4 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 4). The PDO was previously configured via **1403h Receive PDO 4 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1603_h
Object name	Receive PDO 4 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1603h Output Control" to "1603h Receive PDO 4 Mapping Parameter".

	Firmware version FIR-v1426: "Object Name" entry changed from "Output Control" to "Receive PDO 4 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$60FE0120_h$

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1604h Receive PDO 5 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 5). The PDO was previously configured via **1404h Receive PDO 5 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1604_h

Object name	Receive PDO 5 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1605h Receive PDO 6 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 6). The PDO was previously configured via **1405h Receive PDO 6 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1605_h
Object name	Receive PDO 6 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h

Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1606h Receive PDO 7 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 7). The PDO was previously configured via **1406h Receive PDO 7 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1606_h
Object name	Receive PDO 7 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1607h Receive PDO 8 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can receive (RX-PDO 8). The PDO was previously configured via **1407h Receive PDO 8 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	1607_h
Object name	Receive PDO 8 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h

Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write

PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

1800h Transmit PDO 1 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 1. See chapter **Process Data Object (PDO)**.

Object description

Index	1800_h
Object name	Transmit PDO 1 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	

Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1801h Transmit PDO 2 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 2. See chapter **Process Data Object (PDO)**.

Object description

Index	1801_h
Object name	Transmit PDO 2 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1802h Transmit PDO 3 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 3. See chapter **Process Data Object (PDO)**.

Object description

Index	1802_h
Object name	Transmit PDO 3 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time

Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1803h Transmit PDO 4 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 4. See chapter **Process Data Object (PDO)**.

Object description

Index	1803_h
Object name	Transmit PDO 4 Communication Parameter
Object Code	RECORD

Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write

PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1804h Transmit PDO 5 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 5. See chapter **Process Data Object (PDO)**.

Object description

Index	1804_h
Object name	Transmit PDO 5 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$C0000000_h$

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h

Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1805h Transmit PDO 6 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 6. See chapter **Process Data Object (PDO)**.

Object description

Index	1805_h
Object name	Transmit PDO 6 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only

PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$C0000000_h$

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1806h Transmit PDO 7 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 7. See chapter **Process Data Object (PDO)**.

Object description

Index	1806_h
Object name	Transmit PDO 7 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	$C0000000_h$

Subindex	02_h
Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1807h Transmit PDO 8 Communication Parameter

Function

Contains the communication parameters for the sending-side mapping (TX-PDO) 8. See chapter **Process Data Object (PDO)**.

Object description

Index	1807_h
Object name	Transmit PDO 8 Communication Parameter
Object Code	RECORD
Data type	PDO_COMMUNICATION_PARAMETER
Savable	yes, category: communication
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	COB-ID
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$C0000000_h$

Subindex	02_h

Name	Transmission Type
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FF_h

Subindex	03_h
Name	Inhibit Time
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0064_h

Subindex	04_h
Name	Compatibility Entry
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	05_h
Name	Event Timer
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0000_h

Description

Subindex 01_h (COB-ID): The COB-ID is stored here.

Subindex 02_h (transmission type): A number is stored in this subindex that defines the time at which the received data become valid.

Subindex 3 (inhibit time): This can be used to set a time in milliseconds that must elapse after the sending of a PDO before the PDO is sent another time. This time only applies for asynchronous PDOs.

Subindex 4 (compatibility entry): This subindex has no function and exists only for compatibility reasons.

Subindex 5 (event timer): This time (in ms) can be used to trigger an *Event* which handles the copying of the data and the sending of the PDO.

For details, see chapter on **configuring the TX-PDO mapping**.

1A00h Transmit PDO 1 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 1). The PDO was previously configured via **1800h Transmit PDO 1 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A00_h$
Object name	Transmit PDO 1 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1A00h Drive Status" to "1A00h Transmit PDO 1 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Drive Status" to "Transmit PDO 1 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60410010_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60610008_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

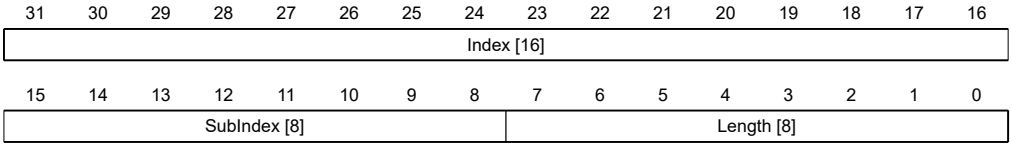
Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A01h Transmit PDO 2 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 2). The PDO was previously configured via **1801h Transmit PDO 2 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A01_h$
Object name	Transmit PDO 2 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1A01h Positioning Status" to "1A01h Transmit PDO 2 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Positioning Status" to "Transmit PDO 2 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60640020_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

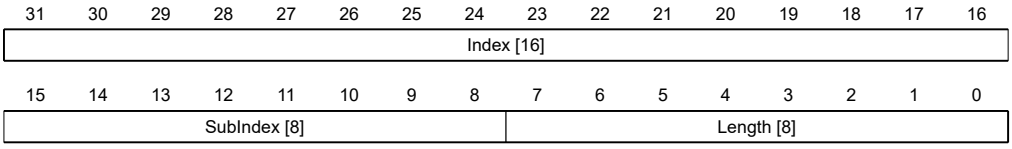
Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A02h Transmit PDO 3 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 3). The PDO was previously configured via **1802h Transmit PDO 3 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A02_h$
Object name	Transmit PDO 3 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1A02h Velocity Status" to "1A02h Transmit PDO 3 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Velocity Status" to "Transmit PDO 3 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60440010_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

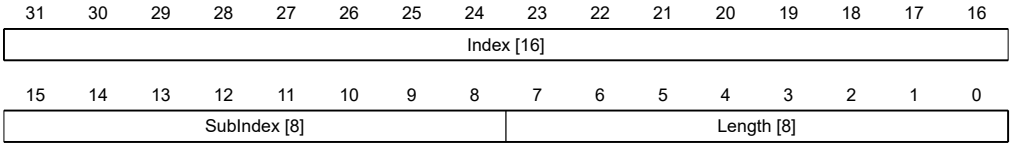
Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A03h Transmit PDO 4 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 4). The PDO was previously configured via **1803h Transmit PDO 4 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A03_h$
Object name	Transmit PDO 4 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: "Heading" entry changed from "1A03h Input Status" to "1A03h Transmit PDO 4 Mapping Parameter". Firmware version FIR-v1426: "Object Name" entry changed from "Input Status" to "Transmit PDO 4 Mapping Parameter".

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$60FD0020_h$

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

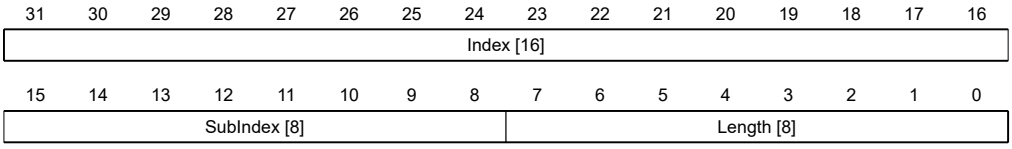
Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A04h Transmit PDO 5 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 5). The PDO was previously configured via **1804h Transmit PDO 5 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A04_h$
Object name	Transmit PDO 5 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

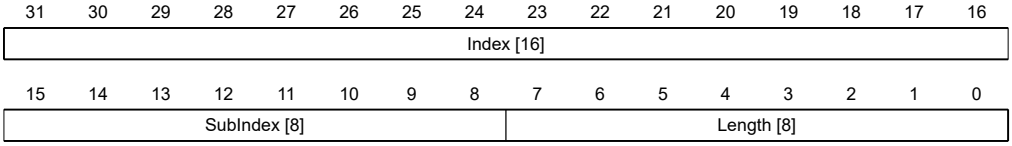
Subindex	08_h

Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A05h Transmit PDO 6 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 6). The PDO was previously configured via **1805h Transmit PDO 6 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A05_h$
Object name	Transmit PDO 6 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h

Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

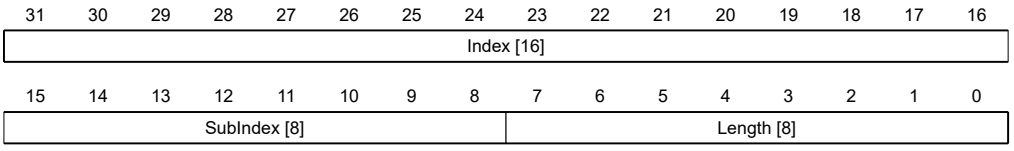
Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A06h Transmit PDO 7 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 7). The PDO was previously configured via **1806h Transmit PDO 7 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A06_h$
Object name	Transmit PDO 7 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

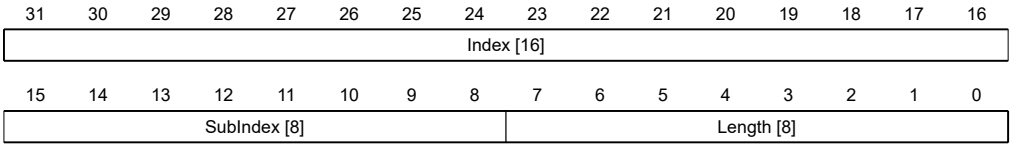
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1A07h Transmit PDO 8 Mapping Parameter

Function

This object contains the mapping parameters for PDOs that the controller can send (TX-PDO 8). The PDO was previously configured via **1807h Transmit PDO 8 Communication Parameter**. See chapter **Process Data Object (PDO)**.

Object description

Index	$1A07_h$
Object name	Transmit PDO 8 Mapping Parameter
Object Code	RECORD
Data type	PDO_MAPPING
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614

Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h
Name	1st Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	2nd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	3rd Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	4th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	5th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	6th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

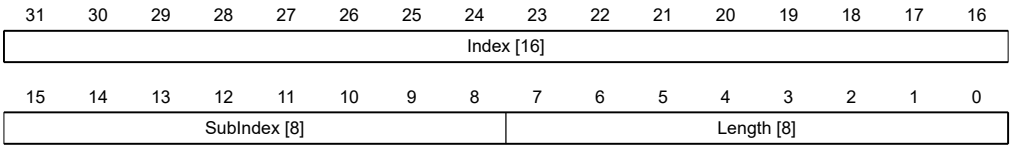
Subindex	07_h
Name	7th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	8th Object To Be Mapped
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–8) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

Subindex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

1F50h Program Data

Function

This object is used to program memory areas of the controller. Each entry stands for a certain memory area.

Object description

Index	$1F50_h$
Object name	Program Data
Object Code	ARRAY
Data type	DOMAIN
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Program Data Bootloader/firmware
Data type	DOMAIN
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0

Subindex	02_h

Name	Program Data NanoJ
Data type	DOMAIN
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0

Subindex	03_h
Name	Program Data DataFlash
Data type	DOMAIN
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0

Description

1F51h Program Control

Function

This object is used to control the programming of memory areas of the controller. Each entry stands for a certain memory area.

Object description

Index	$1F51_h$
Object name	Program Control
Object Code	ARRAY
Data type	UNSIGNED8
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	

Preset value	03_h

Subindex	01_h
Name	Program Control Bootloader/firmware
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	02_h
Name	Program Control NanoJ
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	03_h
Name	Program Control DataFlash
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Description

1F57h Program Status

Function

This object indicates the programming status during the programming of memory areas of the controller. Each entry stands for a certain memory area.

Object description

Index	$1F57_h$
Object name	Program Status
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	

Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Program Status Bootloader/firmware
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Program Status NanoJ
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Program Status DataFlash
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

2005h CANopen Baudrate

Function

This object contains the baud rate of the CANopen bus.

Object description

Index	2005_h
Object name	CANopen Baudrate
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	88_h
Firmware version	FIR-v1426
Change history	

Description

The baud rates are to be set according to the following table. Each value outside of this table is interpreted as 1000 kBd.

Value		**Baud rate**
dec	**hex**	**in kBd**
129	81	10
130	82	20
131	83	50
132	84	125
133	85	250
134	86	500
136	88	1000

2007h CANopen Config

Function

This object can be used to perform various settings for CANopen.

Object description

Index	2007_h
Object name	CANopen Config
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: communication

Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	BL Config
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

The subindices have the following functions:

- Subindex 01: If the value "1" is written in the object, the boot loader suppresses the boot-up message and only the firmware sends a BOOTUP message. With a "0", the boot loader and the firmware each send a BOOTUP message.

2009h CANopen NodeID

Function

This object contains the node-ID of the controller. See chapter *Commissioning*.

Object description

Index	2009_h
Object name	CANopen NodeID
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read / write
PDO mapping	no

Allowed values	
Preset value	$7F_h$
Firmware version	FIR-v1426
Change history	

2028h MODBUS Slave Address

Function

This object contains the slave address for Modbus.

Object description

Index	2028_h
Object name	MODBUS Slave Address
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	1-247
Preset value	05_h
Firmware version	FIR-v1436
Change history	

202Ah MODBUS RTU Baudrate

Function

This object contains the baudrate of modbus in Bd.

Object description

Index	$202A_h$
Object name	MODBUS RTU Baudrate
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00004B00_h$
Firmware version	FIR-v1436
Change history	

202Ch MODBUS RTU Stop Bits

Function

This object contains the number of stop-bits of modbus interface.

Object description

Index	$202C_h$
Object name	MODBUS RTU Stop Bits
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h
Firmware version	FIR-v1436
Change history	

Description

Number of Stopbits	Value in object $202C_h$
1	0
2	2

202Dh MODBUS RTU Parity

Function

This object configures the parity and stop bits for MODBUS RTU.

Object description

Index	$202D_h$
Object name	MODBUS RTU Parity
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	04_h
Firmware version	FIR-v1540
Change history	

Description

The following values apply:

- Value "0x00": Party none, stop bits 2
- Value "0x04": Pairty Even, stop bits 1
- Value "0x06": Pairty odd, stop bits 1

2030h Pole Pair Count

Function

Contains the number of pole pairs of the connected motor.

Object description

Index	2030_h
Object name	Pole Pair Count
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000032_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: "Savable" entry changed from "no" to "yes, category: tuning".

2031h Maximum Current

Function

If **I^2t monitoring** is not active, the rms current specified in the motor data sheet is entered here in mA. If **closed loop** mode is used or if **I^2t monitoring** is activated, the maximum current value is specified here in mA.

Within the controller, the entered value is always interpreted as the root mean square.

Object description

Index	2031_h
Object name	Maximum Current
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0000012C_h$
Firmware version	FIR-v1426

Change history	Firmware version FIR-v1614: "Savable" entry changed from "yes, category: application" to "yes, category: tuning". Firmware version FIR-v1614: "Object Name" entry changed from "Peak Current" to "Max Current".

2032h Maximum Speed

Function

Specifies the maximum permissible speed of the motor in **user-defined units**.

Object description

Index	2032_h
Object name	Maximum Speed
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00030D40_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1614: "Savable" entry changed from "yes, category: application" to "yes, category: tuning".

Description

Note

The object is not taken into account in the **Cyclic Synchronous Velocity** and **Homing** operating modes. In the **Velocity** and **Profile Velocity** operating modes, it is only taken into account if an S-ramp (position ramp, see **3202h Motor Drive Submode Select**) is used.

2033h Plunger Block

Function

The object prevents traveling too far in an undesired direction.

Object description

Index	2033_h
Object name	Plunger Block
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

Description

An electronic locking bolt is thereby realized.

The value 0 switches off monitoring.

The value 100, for example, means that the drive may rotate any distance in the negative direction, but as soon as it moves more than 100 steps in the positive direction, the motor is stopped immediately and an error triggered.

When winding thread, for example, it is thereby possible to prevent accidental unwinding.

2034h Upper Voltage Warning Level

Function

This object contains the threshold value for the "overvoltage" error in millivolts.

Object description

Index	2034_h
Object name	Upper Voltage Warning Level
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00006F54_h$
Firmware version	FIR-v1426
Change history	

Description

If the input voltage of the controller exceeds this threshold value, the motor is switched off and an error triggered. This error is reset automatically if the input voltage is less than (voltage of object 2034_h minus 2 volts).

2035h Lower Voltage Warning Level

Function

This object contains the threshold value for the "Undervoltage" error in millivolts.

Object description

Index	2035_h
Object name	Lower Voltage Warning Level
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00002710_h
Firmware version	FIR-v1426
Change history	

Description

If the input voltage of the controller falls below this threshold value, the motor is switched off and an error triggered. The error is reset automatically if the input voltage exceeds the voltage of object **2035_h** plus 2 volts.

2036h Open Loop Current Reduction Idle Time

Function

This object describes the time in milliseconds that the motor must be at a standstill before current reduction is activated.

Object description

Index	2036_h
Object name	Open Loop Current Reduction Idle Time
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$
Firmware version	FIR-v1426
Change history	

2037h Open Loop Current Reduction Value/factor

Function

This object describes the rms current to which the motor current is to be reduced if current reduction is activated in open loop (bit 3 in **3202_h** = "1") and the motor is at a standstill.

Object description

Index	2037_h
Object name	Open Loop Current Reduction Value/factor
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$FFFFFFCE_h$
Firmware version	FIR-v1426
Change history	

Description

Value of 2037_h greater than or equal to 0 and less than value 2031_h

Current is reduced to the value entered here. The value is in mA and interpreted as root mean square.

Value of 2037_h in the range from -1 to -100

The entered value is interpreted as a percentage and determines the reduction of the rated current in **2037_h**. The value in **2031_h** is used for the calculation.

Example: Object **2031_h** has the value 4200 mA. The value -60 in **2037_h** reduces the current by 60% of **2031_h**. The result is a current reduction to a root mean square of **2031_h** * (**2037_h** + 100) / 100 = 1680 mA.

The value -100 in **2037_h** would, for example, mean that a current reduction is set to a root mean square of 0 mA.

Note

If the rated current is greater than 0 in **$203B_h$**:01, the smaller of **2031_h** and **$203B_h$**:01 is used as the rated current for calculating the current reduction.

2038h Brake Controller Timing

Function

This object contains the times for the *brake control* in milliseconds as well as the PWM frequency and the duty cycle.

Object description

Index	2038_h
Object name	Brake Controller Timing
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application

Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	06_h

Subindex	01_h
Name	Close Brake Idle Time
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	02_h
Name	Shutdown Power Idle Time
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	03_h
Name	Open Brake Delay Time
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	04_h
Name	Start Operation Delay Time
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	05_h
Name	PWM Frequency
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	between 0 and 2000 ($7D0_h$)
Preset value	00000000_h

Subindex	06_h
Name	PWM Duty Cycle
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	0, between 2 and 100 (64_h)
Preset value	00000000_h

Description

The subindices have the following functions:

- 01_h: Time between motor standstill and the closing of the brake.
- 02_h: Time between the closing of the brake and the switching off of the motor current.
- 03_h: Time between the switching on of the motor current and opening of the brake.
- 04_h: Time between the opening of the brake and when the *Operation enabled* state of the **CiA 402 Power State Machine** is reached.
- 05_h: Frequency of the brake PWM in hertz.
- 06_h: Duty cycle of the brake PWM in percent.

2039h Motor Currents

Function

This object contains the measured motor currents in mA.

Object description

Index	2039_h
Object name	Motor Currents
Object Code	ARRAY
Data type	INTEGER32
Savable	no
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1504: "PDO mapping" table entry for subindex 01 changed from "no" to "TX-PDO". Firmware version FIR-v1504: "PDO mapping" table entry for subindex 02 changed from "no" to "TX-PDO".

	Firmware version FIR-v1504: "PDO mapping" table entry for subindex 03 changed from "no" to "TX-PDO". Firmware version FIR-v1504: "PDO mapping" table entry for subindex 04 changed from "no" to "TX-PDO".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	04_h

Subindex	01_h
Name	I_d
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	I_q
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	I_a
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	I_b
Data type	INTEGER32
Access	read only

PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

203Ah Homing On Block Configuration

Function

This object contains the parameters for *Homing on Block* (see chapter **Homing**)

Object description

Index	$203A_h$
Object name	Homing On Block Configuration
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Access	
PDO mapping	
Allowed values	
Preset value	
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: The number of entries was changed from 4 to 3. Firmware version FIR-v1540: "Name" entry changed from "Period Of Blocking" to "Block Detection time". Firmware version FIR-v1614: "Data Type" entry changed from "UNSIGNED32" to "INTEGER32". Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application". Firmware version FIR-v1614: "Data Type" entry changed from "UNSIGNED32" to "INTEGER32". Firmware version FIR-v1614: "Data Type" entry changed from "UNSIGNED32" to "INTEGER32".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h

Name	Minimum Current For Block Detection
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FFFFFFBA$_h$

Subindex	02$_h$
Name	Block Detection Time
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	000000C8$_h$

Description

The subindices have the following function:

- 01$_h$: Specifies the current limit value above which blocking is to be detected. Positive numerical values specify the current limit in mA, negative numbers specify a percentage of object **2031**$_h$:01$_h$. Example: The value "1000" corresponds to 1000 mA (= 1 A); the value "-70" corresponds to 70% of **2031**$_h$.
- 02$_h$: Specifies the time in ms that the motor is to continue to travel against the block after block detection.

203Bh I2t Parameters

Function

This object contains the parameters for I^2t monitoring.

I^2t monitoring is activated by entering a value greater than 0 in **203B**$_h$:01 and **203B**$_h$:02 (see **I2t Motor overload protection**).

With one exception, I^2t monitoring can only be used for *closed loop* mode: If I^2t is activated in *open loop* mode, the current is reduced to the smaller of **203B**$_h$ and **2031**$_h$.

Object description

Index	203B$_h$
Object name	I2t Parameters
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: tuning
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1512: "Savable" entry changed from "no" to "yes, category: application". Firmware version FIR-v1512: The number of entries was changed from 7 to 8.

	Firmware version FIR-v1614: "Savable" entry changed from "yes, category: application" to "yes, category: tuning".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	07_h

Subindex	01_h
Name	Nominal Current
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Maximum Duration Of Peak Current
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Threshold
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	CalcValue
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	05_h
Name	LimitedCurrent
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	Status
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	ActualResistance
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

The subindices are divided into two groups: subindex 01_h and 02_h contain parameters for the control, subindices 03_h to 06_h are status values. The functions are as follows:

- 01_h: The rated current specified in the motor data sheet is entered here in mA. This must be smaller than the current entered in object **2031**$_h$, otherwise monitoring is not activated. The specified value is interpreted as root mean square.
- 02_h: Specifies the maximum duration of the peak current in ms.
- 03_h: Threshold, specifies the limit in mA that determines whether the maximum current or rated current is switched to.
- 04_h: CalcValue, specifies the calculated value that is compared with the threshold for setting the current.
- 05_h: LimitedCurrent, contains the momentary current as root mean square set by I^2t.
- 06_h: Current status. If the sub-entry value is "0", I^2t is deactivated; if the value is "1", I^2t is activated.

203Dh Torque Window

Function

Specifies a symmetrical range relative to the target torque within which the target is considered having been met.

If the value is set to "FFFFFFFF"$_h$, monitoring is switched off, the "Target reached" bit in object **6041**$_h$ (controlword) is never set.

Object description

Index	203D$_h$
Object name	Torque Window
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000$_h$
Firmware version	FIR-v1540
Change history	Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application".

203Eh Torque Window Time

Function

The current torque must be within the "Torque Window" (**203D**$_h$) for this time (in milliseconds) for the target torque to be considered having been met.

Object description

Index	203E$_h$
Object name	Torque Window Time
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000$_h$
Firmware version	FIR-v1540
Change history	Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application".

2050h Encoder Alignment

Function

This value specifies the offset between the index of the encoder and the electric field.

Object description

Index	2050_h
Object name	Encoder Alignment
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: "Savable" entry changed from "no" to "yes, category: tuning".

Description

The exact determination is only possible via **auto setup**. The presence of this value is necessary for *closed loop* mode with encoder.

2051h Encoder Optimization

Function

Contains compensation values for achieving better runout in *closed loop* mode.

Object description

Index	2051_h
Object name	Encoder Optimization
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: tuning
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: "Savable" entry changed from "no" to "yes, category: tuning".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Parameter 1
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Parameter 2
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Parameter 3
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

The exact determination is only possible via **auto setup**.

2052h Encoder Resolution

Function

Contains the physical resolution of the encoder that is used for commutation.

Object description

Index	2052_h
Object name	Encoder Resolution
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00001000_h
Firmware version	FIR-v1426

Change history	Firmware version FIR-v1540: "Savable" entry changed from "no" to "yes, category: tuning".

Description

A negative value means that the encoder is driven in the opposite direction of the motor. This can be corrected by reversing the polarity of a motor winding.

Tip

The unit is "pulses per revolution" (ppr), which corresponds to four times the resolution in "counts per revolution" (cpr) (quadrature). This means that for an encoder with a resolution of, e.g., 1000 increments per revolution, the value in 2052_h is 4000.

2056h Limit Switch Tolerance Band

Function

Specifies how far a limit switch may be passed over in the positive or negative direction before the controller triggers an error.

This tolerance band is necessary, for example, to complete homing operations – in which limit switches can be actuated – error free.

Object description

Index	2056_h
Object name	Limit Switch Tolerance Band
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	$000001F4_h$
Firmware version	FIR-v1426
Change history	

2057h Clock Direction Multiplier

Function

The clock count value in clock/direction mode is multiplied by this value before it is processed further.

Object description

Index	2057_h
Object name	Clock Direction Multiplier
Object Code	VARIABLE
Data type	INTEGER32

Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000080_h
Firmware version	FIR-v1426
Change history	

2058h Clock Direction Divider

Function

The clock count value in clock/direction mode is divided by this value before it is processed further.

Object description

Index	2058_h
Object name	Clock Direction Divider
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h
Firmware version	FIR-v1426
Change history	

2059h Encoder Configuration

Function

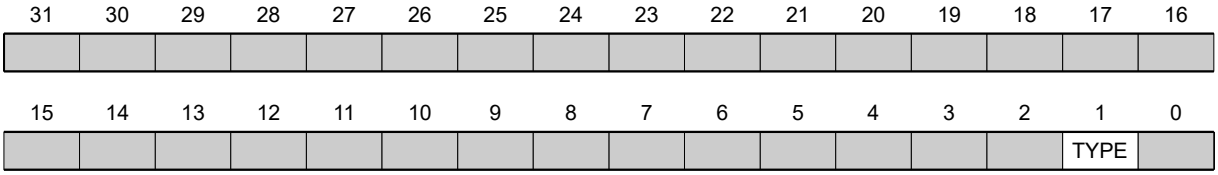
This object can be used to switch the supply voltage and the type of encoder.

Object description

Index	2059_h
Object name	Encoder Configuration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: tuning
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426

Change history	Firmware version FIR-v1614: "Savable" entry changed from "yes, category: application" to "yes, category: tuning".

Description



TYPE

Defines the type of encoder. For a differential encoder, the bit must have the value "0". For a single-ended encoder, the bit must be set to "1".

205Ah Encoder Boot Value

Function

Tip

This object only has a function when using an absolute encoder. If an absolute encoder is not used, the value is always 0.

The initial encoder position when switching on the controller (in **user-defined units**) can be read from this object.

Object description

Index	$205A_h$
Object name	Encoder Boot Value
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1446
Change history	Firmware version FIR-v1512: "Access" table entry for subindex 00 changed from "read/write" to "read only".

205Bh Clock Direction Or Clockwise/Counter Clockwise Mode

Function

This object can be used to switch the clock-direction mode (value = "0") to the right/left rotation mode (value = "1").

Object description

Index	$205B_h$
Object name	Clock Direction Or Clockwise/Counter Clockwise Mode
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1504
Change history	

2060h Compensate Polepair Count

Function

Allows motion blocks to be assigned independent of motor.

Object description

Index	2060_h
Object name	Compensate Polepair Count
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h
Firmware version	FIR-v1426
Change history	

Description

If this entry is set to 1, the number of pole pairs is automatically included in the calculation of all speed, acceleration and jerk parameters.

If the value is 0, the **number of pole pairs** is included in the preset values as with standard stepper motor controllers and must be taken into account if the motor is changed.

2061h Velocity Numerator

Function

Contains the counter that is used for converting from user-defined speed values to the internal revolutions/second. See chapter **User-defined units**.

Object description

Index	2061_h
Object name	Velocity Numerator
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h
Firmware version	FIR-v1426
Change history	

2062h Velocity Denominator

Function

Contains the denominator that is used for converting from user-defined speed values to the internal revolutions/second. See chapter **User-defined units**.

Object description

Index	2062_h
Object name	Velocity Denominator
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0000003C_h$
Firmware version	FIR-v1426
Change history	

2063h Acceleration Numerator

Function

Contains the counter that is used for converting from user-defined acceleration values to the internal revolutions/second^2. See chapter **User-defined units**.

Object description

Index	2063_h
Object name	Acceleration Numerator
Object Code	VARIABLE
Data type	UNSIGNED32

Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h
Firmware version	FIR-v1426
Change history	

2064h Acceleration Denominator

Function

Contains the denominator that is used for converting from user-defined acceleration values to the internal revolutions/second^2. See chapter **User-defined units**.

Object description

Index	2064_h
Object name	Acceleration Denominator
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0000003C_h$
Firmware version	FIR-v1426
Change history	

2065h Jerk Numerator

Function

Contains the counter that is used for converting from user-defined jerk values to the internal revolutions/second ^{3}. See chapter **User-defined units**.

Object description

Index	2065_h
Object name	Jerk Numerator
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h
Firmware version	FIR-v1426

Change history	

2066h Jerk Denominator

Function

Contains the denominator that is used for converting from user-defined jerk values to the internal revolutions/second ^{3}. See chapter **User-defined units**.

Object description

Index	2066_h
Object name	Jerk Denominator
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0000003C_h$
Firmware version	FIR-v1426
Change history	

2084h Bootup Delay

Function

Defines the period between the time that supply voltage is applied to the controller and the functional readiness of the controller in milliseconds.

Object description

Index	2084_h
Object name	Bootup Delay
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

2101h Fieldbus Module Availability

Function

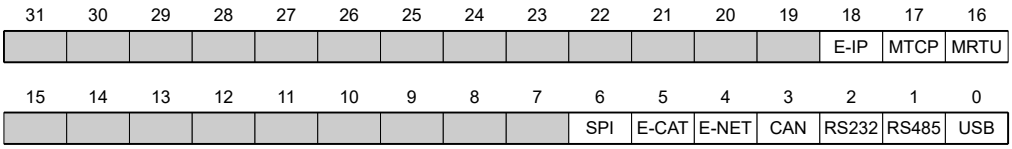
Shows the available fieldbuses.

Object description

Index	2101_h
Object name	Fieldbus Module Availability
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	$0019000F_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Object Name" entry changed from "Fieldbus Module" to "Fieldbus Module Availability".

Description

Bits 0 to 15 represent the physical interface, bits 16 to 31 the used protocol (if necessary).



USB

Value = "1": The USB fieldbus is available.

RS-485

Value = "1": An RS-485 interface is available.

RS-232

Value = "1": An RS-232 interface is available.

CAN

Value = "1": The CANopen fieldbus is available.

E-NET

Value = "1": An Ethernet interface is available.

E-CAT

Value = "1": An EtherCAT interface is available.

SPI

Value = "1": An SPI interface is available.

MRTU

Value = "1": The used protocol is Modbus RTU.

MTCP

Value = "1": The used protocol is Modbus TCP.

E-IP

Value = "1": The used protocol is EtherNet/IP™.

2102h Fieldbus Module Control

Function

This object can be used to activate/deactivate certain fieldbuses (physical interfaces and protocols).

Object description

Index	2102_h
Object name	Fieldbus Module Control
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0009000F_h$
Firmware version	FIR-v1540
Change history	Firmware version FIR-v1626: "Savable" entry changed from "yes, category: application" to "yes, category: communication".

Description

Object **2103**$_h$:1_h contains all physical interfaces/protocols that can be activated/deactivated. These can be switched in this object (2102_h). The current status of the activated fieldbuses is in object **2103**$_h$:2_h.

The following distribution of the bits applies here:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
													E-IP	MTCP	MRTU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									SPI	E-CAT	E-NET	CAN	RS232	RS485	USB

USB

USB interface

RS-485

RS-485 interface

RS-232

RS-232 interface

CAN
CANopen interface

E-NET
EtherNet interface

E-CAT
EtherCAT interface

SPI
SPI interface

MRTU
Modbus RTU protocol

MTCP
Modbus TCP protocol

E-IP
EtherNet/IP™ protocol

2103h Fieldbus Module Status

Function

Shows the active fieldbuses.

Object description

Index	2103_h
Object name	Fieldbus Module Status
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Fieldbus Module Disable Mask
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	$0010000E_h$

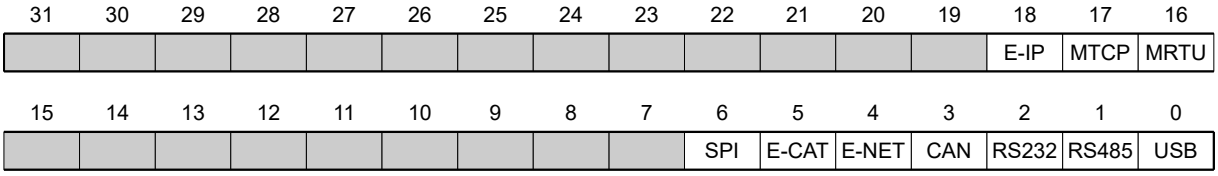
Subindex	02_h
Name	Fieldbus Module Enabled
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	$0009000F_h$

Description

Subindex 1 (Fieldbus Module Disable Mask): This subindex contains all physical interfaces and protocols that can be activated or deactivated. A value "1" means that this fieldbus can be deactivated.

Subindex 2 (Fieldbus Module Enabled): This subindex contains all currently activated physical interfaces and protocols. The value "1" means that that the fieldbus is active.

The following distribution of the bits applies for subindices 1 and 2:



USB
USB interface

RS-485
RS-485 interface

RS-232
RS-232 interface

CAN
CANopen interface

E-NET
EtherNet interface

E-CAT
EtherCAT interface

SPI
SPI interface

MRTU

Modbus RTU protocol

MTCP

Modbus TCP protocol

E-IP

EtherNet/IP™ protocol

2300h NanoJ Control

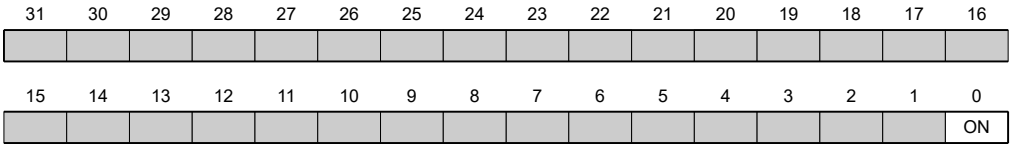
Function

Controls the execution of a NanoJ program.

Object description

Index	2300_h
Object name	NanoJ Control
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Control" to "NanoJ Control".

Description



ON

Switches the NanoJ program on (value = "1") or off (value = "0").

With a rising edge in bit 0, the program is first reloaded and the variable range reset.

Note

Startup of the NanoJ program can take up to 200 ms.

2301h NanoJ Status

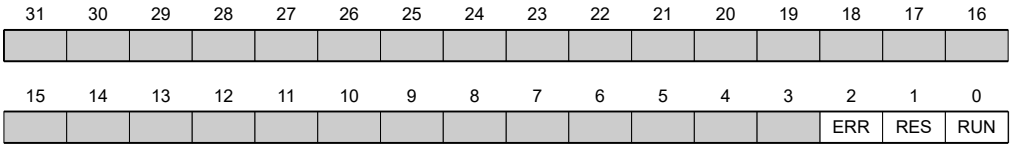
Function

Indicates the operating state of the user program.

Object description

Index	2301_h
Object name	NanoJ Status
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Status" to "NanoJ Status".

Description



RUN

Value = "0": Program is stopped, value = "1": NanoJ program is running.

RES

Reserved.

ERR

Program was ended with an error. Cause of the error can be read from object 2302_h.

2302h NanoJ Error Code

Function

Indicates which error occurred during the execution of the user program.

Object description

Index	2302_h
Object name	NanoJ Error Code
Object Code	VARIABLE
Data type	UNSIGNED32

Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Error Code" to "NanoJ Error Code".

Description

Error codes during program execution:

Number	Description
0000_h	Not an error
0001_h	Firmware does not (yet) support the used function
0002_h	Not or incorrectly initialized pointer
0003_h	Impermissible access to system resource
0004_h	Hard fault (internal error)
0005_h	Code executed too long without yield() or sleep()
0006_h	Impermissible access to system resource
0007_h	Too many variables on the stack
0100_h	Invalid NanoJ program file

Error when accessing an object:

Number	Description
$10xxxxyy_h$	Invalid mapping in the NanoJ program file: The value in "xxxx" specifies the index, the value in "yy" specifies the subindex of the object that should – but cannot – be mapped.
1000_h	Access of a nonexistent object in the object dictionary
1001_h	Write access of a write-protected entry in the OD
1002_h	Internal file system error

File system error codes when loading the user program:

Number	Description
10002_h	Internal file system error
10003_h	Storage medium not ready
10004_h	File not found
10005_h	Folder not found
10006_h	Invalid file name/folder name
10008_h	Access of file not possible
10009_h	File/directory object is invalid
$1000A_h$	Storage medium is read-only
$1000B_h$	Drive number is invalid

Number	Description
$1000C_h$	Working range of the drive is invalid
$1000D_h$	No valid file system on the drive
$1000E_h$	Creation of the file system failed
$1000F_h$	Access not possible within the required time
10010_h	Access was rejected

230Fh Uptime Seconds

Function

This object contains the operating hours in seconds since the last time the controller was started.

Note

This object is not stored; counting begins with "0" again after switching on.

Object description

Index	$230F_h$
Object name	Uptime Seconds
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1436
Change history	

2310h NanoJ Input Data Selection

Function

Describes the object dictionary entries that are copied to the PDO mapping input of the NanoJ program.

Object description

Index	2310_h
Object name	NanoJ Input Data Selection
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read / write
PDO mapping	no

Allowed values	
Preset value	
Firmware version	FIR-v1650-B472161
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Input Data Selection" to "NanoJ Input Data Selection". Firmware version FIR-v1650-B472161: "Savable" entry changed from "yes, category: application" to "no". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 00 changed from "read/write" to "read only". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 01 changed from "read/write" to "read only".

Value description

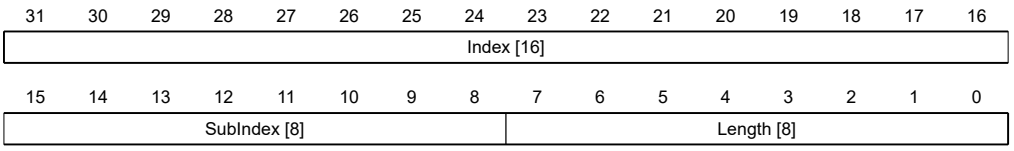
Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	10_h

Subindex	01_h - 10_h
Name	Mapping #1 - #16
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Each subindex (1–16) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

SubIndex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

2320h NanoJ Output Data Selection

Function

Describes the object dictionary entries that are copied into the output PDO mapping of the VMM program after it is executed.

Object description

Index	2320_h
Object name	NanoJ Output Data Selection
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1650-B472161
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Output Data Selection" to "NanoJ Output Data Selection". Firmware version FIR-v1650-B472161: "Savable" entry changed from "yes, category: application" to "no". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 00 changed from "read/write" to "read only". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 01 changed from "read/write" to "read only".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	10_h

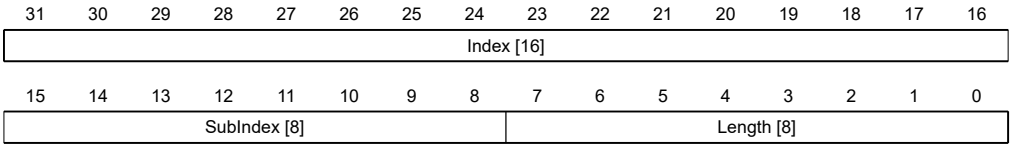
Subindex	01_h - 10_h
Name	Mapping #1 - #16
Data type	UNSIGNED32
Access	read only
PDO mapping	no

Allowed values	
Preset value	00000000_h

Description

Each subindex (1–16) describes a different mapped object.

A mapping entry consists of four bytes, which are structured according to the following graphic.



Index [16]

This contains the index of the object to be mapped.

SubIndex [8]

This contains the subindex of the object to be mapped.

Length [8]

This contains the length of the object to be mapped in units of bits.

2330h NanoJ In/output Data Selection

Function

Describes the object dictionary entries that are first copied to the input PDO mapping of the NanoJ program and, after it is executed, are copied back to the output PDO mapping.

Object description

Index	2330_h
Object name	NanoJ In/output Data Selection
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1650-B472161
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM In/output Data Selection" to "NanoJ In/output Data Selection". Firmware version FIR-v1650-B472161: "Savable" entry changed from "yes, category: application" to "no". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 00 changed from "read/write" to "read only". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 01 changed from "read/write" to "read only".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	10_h

Subindex	01_h - 10_h
Name	Mapping #1 - #16
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

2400h NanoJ Inputs

Function

Located here is an array with 32, 32-bit integer values that is not used within the firmware and serves only for communicating with the user program via the fieldbus.

Object description

Index	2400_h
Object name	NanoJ Inputs
Object Code	ARRAY
Data type	INTEGER32
Savable	no
Firmware version	FIR-v1426
Change history	The number of entries was changed from 2 to 33. Firmware version FIR-v1436: "Object Name" entry changed from "VMM Inputs" to "NanoJ Inputs". Firmware version FIR-v1436: "Name" entry changed from "VMM Input N#" to "NanoJ Input N#".

Value description

Subindex	00_h
Name	Highest Sub-index Supported

Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	20_h

Subindex	01_h - 20_h
Name	NanoJ Input #1 - #32
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

Here, it is possible to pass, e.g., preset values, to the VMM program.

2410h NanoJ Init Parameters

Function

This object functions identically to object **2400**$_h$ with the difference that this object can be stored.

Object description

Index	2410_h
Object name	NanoJ Init Parameters
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1450
Change history	Firmware version FIR-v1450: "Data Type" entry changed from "INTEGER32" to "UNSIGNED8".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no

Allowed values	
Preset value	20_h

Subindex	01_h - 20_h
Name	NanoJ Init Parameter #1 - #32
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

2500h NanoJ Outputs

Function

Located here is an array with 32, 32-bit integer values that is not used within the firmware and serves only for communicating with the user program via the fieldbus.

Object description

Index	2500_h
Object name	NanoJ Outputs
Object Code	ARRAY
Data type	INTEGER32
Savable	no
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Outputs" to "NanoJ Outputs". Firmware version FIR-v1436: "Name" entry changed from "VMM Output N#" to "NanoJ Output N#".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	20_h

Subindex	01_h - 20_h
Name	NanoJ Output #1 - #32
Data type	INTEGER32
Access	read / write

PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Description

Here, the VMM program can store results which can then be read out via the fieldbus.

2600h NanoJ Debug Output

Function

This object contains debug output of a user program.

Object description

Index	2600_h
Object name	NanoJ Debug Output
Object Code	ARRAY
Data type	UNSIGNED8
Savable	no
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1436: "Object Name" entry changed from "VMM Debug Output" to "NanoJ Debug Output".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	01_h - 40_h
Name	Value #1 - #64
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	00_h

Description

Here, the NanoJ program stores the debug output that was called up with the `VmmDebugOutputString()`, `VmmDebugOutputInt()` and similar functions.

2800h Bootloader And Reboot Settings

Function

With this object, a reboot of the firmware can be triggered and the short circuiting of the motor windings in bootloader mode switched off and on.

Object description

Index	2800_h
Object name	Bootloader And Reboot Settings
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Reboot Command
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Reboot Delay Time In Ms
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Bootloader HW Config
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

The subindices have the following function:

- 01_h: If the value $746F6F62_h$ is entered here, the firmware is rebooted.
- 02_h: Time in milliseconds: delays the reboot of the firmware by the respective time.
- 03_h: Bit 0 can be used to switch short circuiting of the motor windings in boot loader mode off and on:
  - Bit 0 = 1: Short circuiting of the motor windings in bootloader mode is switched off.
  - Bit 0 = 0: Short circuiting of the motor windings in bootloader mode is switched on.

3202h Motor Drive Submode Select

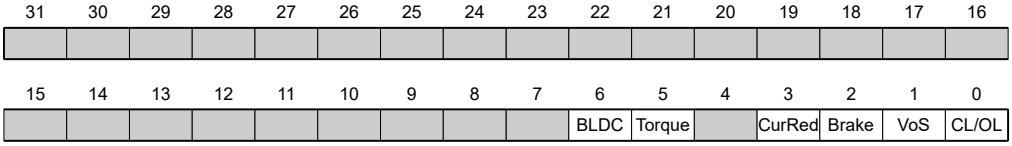
Function

Controls the controller mode, such as the changeover between *closed loop* / *open loop* and whether Velocity Mode is simulated via the S-controller or functions with a real V-controller in *closed loop*.

Object description

Index	3202_h
Object name	Motor Drive Submode Select
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: drive
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1540: "Savable" entry changed from "yes category: application" to "yes, category: travel". Firmware version FIR-v1540: "Savable" entry changed from "yes category: travel" to "yes, category: drive".

Description



CL/OL

Changeover between *open loop* and *closed loop*

- Value = "0": *open loop*
- Value = "1": *closed loop*

VoS

Value = "1": Simulate V-controller with an S-ramp: simulate the speed modes through continuous position changes

Brake

Value = "1": Switch on **automatic brake control**

CurRed (Current Reduction)

Value = "1": Current reduction activated in *open loop*

Torque

only active in operating modes **Profile Torque** and **Cyclic Synchronous Torque**

Value = "1": M-controller is active, otherwise a V-controller is superimposed: no V-controller is used in the torque modes for speed limiting, thus object **2032**$_h$ is ignored; **3210**$_h$:3 and **3210**$_h$:4 have no effect on the control.

BLDC

Value = "1": Motor type "BLDC" (brushless DC motor)

320Ah Motor Drive Sensor Display Open Loop

Function

This can be used to change the source for objects **6044**$_h$ and **6064**$_h$ in *open loop* mode.

Object description

Index	320A$_h$
Object name	Motor Drive Sensor Display Open Loop
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00$_h$

Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	04_h

Subindex	01_h
Name	Commutation
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Torque
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Velocity
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	04_h
Name	Position
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Description

The following subindices have a function:

- 01_h: Not used
- 02_h: Not used

- 03_h: Changes the source of object **6044_h**:
  - Value = "-1": The internally calculated set value is entered in object **6044_h**
  - Value = "0": The value is kept at 0
  - Value = "1": The encoder value is entered in object **6044_h**
- 04_h: Changes the source of **6064_h**:
  - Value = "-1": The internally calculated set value is entered in object **6064_h**
  - Value = "0": The value is kept at 0
  - Value = "1": The encoder value is entered in object **6064_h**

320Bh Motor Drive Sensor Display Closed Loop

Function

This can be used to change the source for objects **6044_h** and **6064_h** in *closed loop* mode.

Object description

Index	$320B_h$
Object name	Motor Drive Sensor Display Closed Loop
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	04_h

Subindex	01_h
Name	Commutation
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Torque
Data type	INTEGER32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Velocity
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	04_h
Name	Position
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Description

The following subindices have a function:

- 01_h: Not used
- 02_h: Not used
- 03_h: Changes the source of object **6044_h**:
  - Value = "-1": The internally calculated set value is entered in object **6044_h**
  - Value = "0": The value is kept at 0
  - Value = "1": The encoder value is entered in object **6044_h**
- 04_h: Changes the source of object **6064_h**:
  - Value = "-1": The internally calculated set value is entered in object **6064_h**
  - Value = "0": The value is kept at 0
  - Value = "1": The encoder value is entered in object **6064_h**

3210h Motor Drive Parameter Set

Function

Contains the P and I components of the current, distance and position controllers for *open loop* (only current controller activated) and *closed loop*.

Object description

Index	3210_h
Object name	Motor Drive Parameter Set
Object Code	ARRAY

Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Name" entry changed from "S_P" to "Position Loop, Proportional Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "S_I" to "Position Loop, Integral Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "V_P" to "Velocity Loop, Proportional Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "V_I" to "Velocity Loop, Integral Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "Id_P" to "Flux Current Loop, Proportional Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "Id_I" to "Flux Current Loop, Integral Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "Iq_P" to "Torque Current Loop, Proportional Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "Iq_I" to "Torque Current Loop, Integral Gain (closed loop)". Firmware version FIR-v1626: "Name" entry changed from "I_P" to "Torque Current Loop, Proportional Gain (dspDrive – Stepper Motor, open loop)". Firmware version FIR-v1626: "Name" entry changed from "I_I" to "Torque Current Loop, Integral Gain (dspDrive – Stepper Motor, open loop)". Firmware version FIR-v1650-B472161: "Name" entry changed from "Torque Current Loop, Proportional Gain (dspDrive – Stepper Motor, open loop)" to "Torque Current Loop, Proportional Gain (open loop)". Firmware version FIR-v1650-B472161: "Name" entry changed from "Torque Current Loop, Integral Gain (dspDrive – Stepper Motor, open loop)" to "Torque Current Loop, Integral Gain (open loop)". Firmware version FIR-v1650-B472161: "Data type" entry changed from "INTEGER32" to "UNSIGNED32". Firmware version FIR-v1650-B472161: "Data type" entry changed from "INTEGER32" to "UNSIGNED32".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	$0A_h$

Subindex	01_h
Name	Position Loop, Proportional Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000800_h

Subindex	02_h
Name	Position Loop, Integral Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Velocity Loop, Proportional Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00001B58_h$

Subindex	04_h
Name	Velocity Loop, Integral Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000004_h

Subindex	05_h
Name	Flux Current Loop, Proportional Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000668A0_h$

Subindex	06_h
Name	Flux Current Loop, Integral Gain (closed Loop)
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00002EE0_h$

Subindex	07_h
Name	Torque Current Loop, Proportional Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000668A0_h$

Subindex	08_h
Name	Torque Current Loop, Integral Gain (closed Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$00002EE0_h$

Subindex	09_h
Name	Torque Current Loop, Proportional Gain (open Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0001D4C0_h$

Subindex	$0A_h$
Name	Torque Current Loop, Integral Gain (open Loop)
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$0000AFC8_h$

Description

- Subindex 00_h: Number of entries
- Subindex 01_h: Proportional component of the S-controller (position)
- Subindex 02_h: Integral component of the S-controller (position)
- Subindex 03_h: Proportional component of the V-controller (speed)
- Subindex 04_h: Integral component of the V-controller (speed)

- Subindex 05_h: (Closed loop) Proportional component of the current controller of the field-forming component
- Subindex 06_h: (Closed loop) Integral component of the current controller of the field-forming component
- Subindex 07_h: (Closed loop) Proportional component of the current controller of the torque-forming component
- Subindex 08_h: (Closed loop) Integral component of the current controller of the torque-forming component
- Subindex 09_h: (Open loop) Proportional component of the current controller of the field-building component
- Subindex $0A_h$: (Open loop) Integral component of the current controller of the field-forming component

3212h Motor Drive Flags

Function

This object determines whether or not the output voltage for the motor is active in the "switched on" mode of the CiA 402 state machine. The direction of the rotating field can also be changed.

Note

Changes in subindex 02 do not take effect until after the control is restarted. Afterwards, **Auto setup** must again be performed.

Object description

Index	3212_h
Object name	Motor Drive Flags
Object Code	ARRAY
Data type	INTEGER8
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1450
Change history	Firmware version FIR-v1512: The number of entries was changed from 2 to 3.

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Enable Legacy Power Mode
Data type	INTEGER8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	02_h
Name	Override Field Inversion
Data type	INTEGER8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	03_h
Name	Do Not Touch Controller Settings
Data type	INTEGER8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Description

Valid values for subindex 01_h:

- Value = "0": In the "Switched on" state of the **CiA 402 Power State Machine**, the output voltage for the motor (PWM) is permanently set to 50%; no holding torque is built up.
- Value = "1": In the "Switched on" state of the **CiA 402 Power State Machine**, the output voltage for the motor (PWM) is active via the controller; holding torque is built up. The motor remains at a standstill.

Valid values for subindex 02_h:

- Value = "0": Use default values of the firmware
- Value = "1": Force non-inversion of the rotating field (mathematically positive)
- Value = "-1": Force inversion of the rotating field (mathematically negative)

Valid values for subindex 03_h:

- Value = "0": **Auto setup** detects the motor type (stepper motor or BLDC motor) and uses the corresponding pre-configured parameter set.
- Value = "1": Perform **auto setup** with the values for the controller that were entered in object **3210_h** before the auto setup; the values in **3210_h** are not changed.

3220h Analog Inputs

Function

Displays the instantaneous values of the analog inputs in digits.

With object **3221**$_h$, the respective analog input can be configured as current or voltage input.

Object description

Index	3220$_h$
Object name	Analog Inputs
Object Code	ARRAY
Data type	INTEGER16
Savable	no
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00$_h$
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02$_h$

Subindex	01$_h$
Name	Analogue Input 1
Data type	INTEGER16
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000$_h$

Subindex	02$_h$
Name	Analogue Input 2
Data type	INTEGER16
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000$_h$

Description

Formula for converting from digits to the respective unit:

- Voltage input: x digits * 10 V / 1024 digits
- Current input: x digits * 20 mA / 1024 digits

3221h Analogue Inputs Control

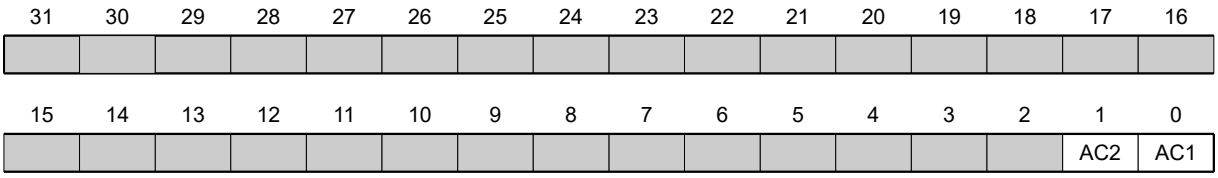
Function

With this object, an analog input can be switched from voltage measurement to current measurement.

Object description

Index	3221_h
Object name	Analogue Inputs Control
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

Description



In general: If a bit is set to the value "0", the analog input measures the voltage; if the bit is set to the value "1", the current is measured.

AC1

Setting for analog input 1

AC2

Setting for analog input 2

3240h Digital Inputs Control

Function

With this object, digital inputs can be manipulated as described in chapter **Digital inputs and outputs**.

The following applies for all subindices:

- Bits 0 to 15 control the special functions.
- Bits 16 to 31 control the level of the outputs.

Object description

Index	3240_h
Object name	Digital Inputs Control

Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: Subindex 01_h: "Name" entry changed from "Special Function Disable" to "Special Function Enable" Firmware version FIR-v1512: The number of entries was changed from 8 to 9.

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	08_h

Subindex	01_h
Name	Special Function Enable
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Function Inverted
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Force Enable
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	Force Value
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	Raw Value
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	Input Range Select
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	Differential Select
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	Routing Enable
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

The subindices have the following function:

- **3240**$_h$:01$_h$ (Special Function Enable): This bit allows special functions of an input to be switched off (value "0") or on (value "1"). If input 1 is not used as, e.g., a negative limit switch, the special function must be switched off to prevent an erroneous response to the signal generator. The object has no effect on bits 16 to 31.
  The firmware evaluates the following bits:
  - Bit 0: Negative limit switch
  - Bit 1: Positive limit switch
  - Bit 2: Home switch

  If, for example, two limit switches and one home switch are used, bits 0–2 in **3240**$_h$:01$_h$ must be set to "1".

- **3240**$_h$:02$_h$ (Function Inverted): This bit switches from normally open logic (a logical high level at the input yields the value "1" in object **60FD**$_h$) to normally closed logic (the logical high level at the input yields the value "0"). This applies for the special functions (except for the clock and direction inputs) and for the normal inputs. If the bit has the value "0", normally open logic applies; for the value "1", normally closed logic applies.
- **3240**$_h$:03$_h$ (Force Enable): This bit switches on the software simulation of input values if it is set to "1". In this case, the actual values are no longer used in object **3240**$_h$:04$_h$, but rather the set values for the respective input.
- **3240**$_h$:04$_h$ (Force Value): This bit specifies the value that is to be read as the input value if the same bit was set in object **3240**$_h$:03$_h$.
- **3240**$_h$:05$_h$ (Raw Value): This object contains the unmodified input value.
- **60FD**$_h$ (Digital Inputs): This object contains a summary of the inputs and the special functions.

3242h Digital Input Routing

Function

This object determines the source of the input routing that ends in **60FD**$_h$.

Object description

Index	3242$_h$
Object name	Digital Input Routing
Object Code	ARRAY
Data type	UNSIGNED8
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1504
Change history	

Value description

Subindex	00$_h$
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no

Allowed values	
Preset value	24_h

Subindex	01_h - 24_h
Name	Input Source #1 - #36
Data type	UNSIGNED8
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	00_h

Description

Subindex 01_h contains the source for bit 0 of object **60FD**. Subindex 02_h contains the source for bit 1 of object **60FD** and so on.

The number that is written in a subindex determines the source for the corresponding bit. The following table lists all possible signal sources.

Number		
dec	**hex**	**Signal source**
00	00	Signal is always 0
01	01	Physical input 1
02	02	Physical input 2
03	03	Physical input 3
04	04	Physical input 4
05	05	Physical input 5
06	06	Physical input 6
07	07	Physical input 7
08	08	Physical input 8
09	09	Physical input 9
10	0A	Physical input 10
11	0B	Physical input 11
12	0C	Physical input 12
13	0D	Physical input 13
14	0E	Physical input 14
15	0F	Physical input 15
16	10	Physical input 16
65	41	Hall input "U"
66	42	Hall input "V"
67	43	Hall input "W"
68	44	Encoder input "A"
69	45	Encoder input "B"
70	46	Encoder input "Index"
71	47	USB Power Signal
72	48	"Ethernet active" status
73	49	DIP switch 1

Number		
dec	**hex**	**Signal source**
74	4A	DIP switch 2
75	4B	DIP switch 3
76	4C	DIP switch 4
77	4D	DIP switch 5
78	4E	DIP switch 6
79	4F	DIP switch 7
80	50	DIP switch 8
128	80	Signal is always 1
129	81	Inverted physical input 1
130	82	Inverted physical input 2
131	83	Inverted physical input 3
132	84	Inverted physical input 4
133	85	Inverted physical input 5
134	86	Inverted physical input 6
135	87	Inverted physical input 7
136	88	Inverted physical input 8
137	89	Inverted physical input 9
138	8A	Inverted physical input 10
139	8B	Inverted physical input 11
140	8C	Inverted physical input 12
141	8D	Inverted physical input 13
142	8E	Inverted physical input 14
143	8F	Inverted physical input 15
144	90	Inverted physical input 16
193	C1	Inverted Hall input "U"
194	C2	Inverted Hall input "V"
195	C3	Inverted Hall input "W"
196	C4	Inverted encoder input "A"
197	C5	Inverted encoder input "B"
198	C6	Inverted encoder input "Index"
199	C7	Inverted USB power signal
200	C8	"Ethernet active" inverted status
201	C9	Inverted DIP switch 1
202	CA	Inverted DIP switch 2
203	CB	Inverted DIP switch 3
204	CC	Inverted DIP switch 4
205	CD	Inverted DIP switch 5
206	CE	Inverted DIP switch 6
207	CF	Inverted DIP switch 7
208	D0	Inverted DIP switch 8

3250h Digital Outputs Control

Function

This object can be used to control the digital outputs as described in chapter " **Digital inputs and outputs**".

The following applies for all subindices:

- Bits 0 to 15 control the special functions.
- Bits 16 to 31 control the level of the outputs.

Object description

Index	3250_h
Object name	Digital Outputs Control
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1426: Subindex 01_h: "Name" entry changed from "Special Function Disable" to "Special Function Enable" Firmware version FIR-v1446: "Name" entry changed from "Special Function Enable" to "No Function". Firmware version FIR-v1512: The number of entries was changed from 6 to 9.

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	08_h

Subindex	01_h
Name	No Function
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Function Inverted

Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Force Enable
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	04_h
Name	Force Value
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	05_h
Name	Raw Value
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	06_h
Name	Reserved1
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	07_h
Name	Reserved2
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO

Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	Routing Enable
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

The subindices have the following function:

- 01_h: No function.
- 02_h: This subindex is used to invert the logic (from normally closed logic to normally open logic).
- 03_h: This subindex is used to force the output value if the bit has the value "1". The level of the output is defined in subindex 4_h.
- 04_h: This subindex is used to define the level to be applied to the output. The value "0" returns a logical low level at the digital output; the value "1", on the other hand, returns a logical high level.
- 05_h: The bit combination applied to the outputs is stored in this subindex.

3252h Digital Output Routing

Function

This object assigns a signal source to an output; this signal source can be controlled with **60FE**$_h$.

Object description

Index	3252_h
Object name	Digital Output Routing
Object Code	ARRAY
Data type	UNSIGNED16
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8

Access	read only
PDO mapping	no
Allowed values	
Preset value	05_h

Subindex	01_h
Name	Output Control #1
Data type	UNSIGNED16
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	1080_h

Subindex	02_h
Name	Output Control #2
Data type	UNSIGNED16
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	0090_h

Subindex	03_h
Name	Output Control #3
Data type	UNSIGNED16
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	0091_h

Subindex	04_h
Name	Output Control #4
Data type	UNSIGNED16
Access	read / write
PDO mapping	TX-PDO
Allowed values	
Preset value	0092_h

Subindex	05_h
Name	Output Control #5
Data type	UNSIGNED16
Access	read / write
PDO mapping	TX-PDO
Allowed values	

Preset value	0093_h

3320h Read Analogue Input

Function

Displays the instantaneous values of the analog inputs in user-defined units.

Object description

Index	3320_h
Object name	Read Analogue Input
Object Code	ARRAY
Data type	INTEGER32
Savable	no
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Number Of Analogue Inputs
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Analogue Input 1
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Analogue Input 2
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Description

The user-defined units are made up of offset (**3321**$_h$) and pre-scaling value (**3322**$_h$). If both object entries are still set to the default values, the value in **3320**$_h$ is specified in the "ADC digits" unit.

Formula for converting from digits to the respective unit:

Voltage input: x digits * 10 V / 1024 digits

Current input: x digits * 20 mA / 1024 digits

The following applies for the sub-entries:

- Subindex 00$_h$: Number of analog inputs
- Subindex 01$_h$: Analog value 1
- Subindex 02$_h$: Analog value 2

3321h Analogue Input Offset

Function

Offset that is added to the read analog value (**3320**$_h$) before dividing by the divisor from object **3322**$_h$.

Object description

Index	3321$_h$
Object name	Analogue Input Offset
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00$_h$
Name	Number Of Analogue Inputs
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02$_h$

Subindex	01$_h$
Name	Analogue Input 1
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000$_h$

Subindex	02_h
Name	Analogue Input 2
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

- Subindex 00_h: Number of offsets
- Subindex 01_h: Offset for analog input 1
- Subindex 02_h: Offset for analog input 2

3322h Analogue Input Pre-scaling

Function

Value by which the read analog value (**3320_h**, **3321_h**) is divided before it is written in object **3320_h**.

Object description

Index	3322_h
Object name	Analogue Input Pre-scaling
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Number Of Analogue Inputs
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Analogue Input 1
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	All values permitted except 0
Preset value	00000001_h

Subindex	02_h
Name	Analogue Input 2
Data type	INTEGER32
Access	read / write
PDO mapping	no
Allowed values	All values permitted except 0
Preset value	00000001_h

Description

The subindices contain:

- Subindex 00_h: Number of divisors
- Subindex 01_h: Divisor for analog input 1
- Subindex 02_h: Divisor for analog input 2

3502h MODBUS Rx PDO Mapping

Function

Objects for the rx mapping can get written in this object.

Object description

Index	3502_h
Object name	MODBUS Rx PDO Mapping
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	08_h

Subindex	01_h
Name	Value #1

Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60400010_h

Subindex	02_h
Name	Value #2
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00050008_h

Subindex	03_h
Name	Value #3
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60600008_h

Subindex	04_h
Name	Value #4
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	32020020_h

Subindex	05_h
Name	Value #5
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$607A0020_h$

Subindex	06_h
Name	Value #6
Data type	UNSIGNED32
Access	read / write
PDO mapping	no

Allowed values	
Preset value	60810020_h

Subindex	07_h
Name	Value #7
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60420010_h

Subindex	08_h
Name	Value #8
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$60FE0120_h$

Subindex	09_h
Name	Value #9
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0A_h$
Name	Value #10
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0B_h$
Name	Value #11
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0C_h$
Name	Value #12
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0D_h$
Name	Value #13
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0E_h$
Name	Value #14
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0F_h$
Name	Value #15
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	10_h
Name	Value #16
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

3602h MODBUS Tx PDO Mapping

Function

Objects for the tx mapping can get written in this object.

Object description

Index	3602_h
Object name	MODBUS Tx PDO Mapping
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: communication
Access	read / write
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1614
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	06_h

Subindex	01_h
Name	Value #1
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60410010_h

Subindex	02_h
Name	Value #2
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00050008_h

Subindex	03_h
Name	Value #3
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60610008_h

Subindex	04_h
Name	Value #4
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60640020_h

Subindex	05_h
Name	Value #5
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	60440010_h

Subindex	06_h
Name	Value #6
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$60FD0020_h$

Subindex	07_h
Name	Value #7
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	08_h
Name	Value #8
Data type	UNSIGNED32

Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	09_h
Name	Value #9
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0A_h$
Name	Value #10
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0B_h$
Name	Value #11
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0C_h$
Name	Value #12
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0D_h$
Name	Value #13
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	

Preset value	00000000_h

Subindex	$0E_h$
Name	Value #14
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	$0F_h$
Name	Value #15
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Subindex	10_h
Name	Value #16
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

3700h Following Error Option Code

Function

The object contains the action that is to be executed if a following error is triggered.

Object description

Index	3700_h
Object name	Following Error Option Code
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$FFFF_h$
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … -2	Reserved
-1	No reaction
0	Immediate stop
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode)
2	Braking with "quick stop ramp" (deceleration (deceleration ramp) depending on operating mode)
3 … 32767	Reserved

4012h HW Information

Function

This object contains information about the hardware.

Object description

Index	4012_h
Object name	HW Information
Object Code	ARRAY
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	EEPROM Size In Bytes
Data type	UNSIGNED32
Access	read only

PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Subindex 01: Contains the size of the connected EEPROM in bytes. The value "0" means that no EEPROM is connected.

4013h HW Configuration

Function

This object is used to set certain hardware configurations.

Object description

Index	4013_h
Object name	HW Configuration
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	HW Configuration #1
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000000_h

Description

Bit 0: reserved

4014h Operating Conditions

Function

This object is used to read out the current environment values for the controller.

Object description

Index	4014_h
Object name	Operating Conditions
Object Code	ARRAY
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1540
Change history	Firmware version FIR-v1650-B472161: "Access" table entry for subindex 01 changed from "read/write" to "read only". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 02 changed from "read/write" to "read only". Firmware version FIR-v1650-B472161: "Name" entry changed from "Temperature PCB [d?C]" to "Temperature PCB [Celsius * 10]". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 03 changed from "read/write" to "read only".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	03_h

Subindex	01_h
Name	Voltage UB Power [mV]
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	

Preset value	00000000_h

Subindex	02_h
Name	Voltage UB Logic [mV]
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Subindex	03_h
Name	Temperature PCB [Celsius * 10]
Data type	INTEGER32
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h

Description

The subindices contain:

- 01_h: Current voltage supply voltage in [mV]
- 02_h: Current logic voltage in [mV]
- 03_h: Current temperature in [d°C] (tenths of degree)

4040h Drive Serial Number

Function

This object contains the serial number of the controller.

Object description

Index	4040_h
Object name	Drive Serial Number
Object Code	VARIABLE
Data type	VISIBLE_STRING
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	0
Firmware version	FIR-v1450
Change history	

4041h Device Id

Function

This object contains the ID of the device.

Object description

Index	4041_h
Object name	Device Id
Object Code	VARIABLE
Data type	OCTET_STRING
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	0
Firmware version	FIR-v1540
Change history	

Description

603Fh Error Code

Function

This object returns the error code of the last error that occurred.

It corresponds to the lower 16 bits of object **1003_h**. For the description of the error codes, refer to object **1003_h**.

Object description

Index	$603F_h$
Object name	Error Code
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

Description

For the meaning of the error, see object **1003_h** (Pre-defined Error Field).

6040h Controlword

Function

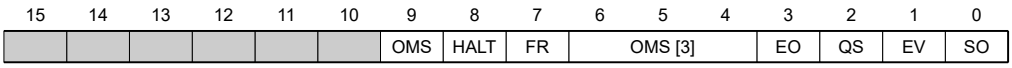
This object controls the **CiA 402 Power State Machine**.

Object description

Index	6040_h
Object name	Controlword
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

Description

Parts of the object are, with respect to function, dependent on the currently selected mode.



SO (Switched On)

Value = "1": Switches to the "Switched on" state

EV (Enable Voltage)

Value = "1": Switches to the "Enable voltage" state

QS (Quick Stop)

Value = "0": Switches to the "Quick stop" state

EO (Enable Operation)

Value = "1": Switches to the "Enable operation" state

OMS (Operation Mode Specific)

Meaning is dependent on the selected operating mode

FR (Fault Reset)

Resets an error (if possible)

HALT

Value = "1": Triggers a halt; valid in the following modes:

- **Profile Position**
- **Velocity**
- **Profile Velocity**
- **Profile Torque**

- **Interpolated Position Mode**

6041h Statusword

Function

This object returns information about the status of the **CiA 402 Power State Machine**.

Object description

Index	6041_h
Object name	Statusword
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

Description

Parts of the object are, with respect to function, dependent on the currently selected mode.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLA		OMS [2]		ILA	TARG	REM	SYNC	WARN	SOD	QS	VE	FAULT	OE	SO	RTSO

RTSO (Ready To Switch On)
Value = "1": Controller is in the "Ready to switch on" state

SO (Switched On)
Value = "1": Controller is in the "Switched on" state

OE (Operation Enabled)
Value = "1": Controller is in the "Operation enabled" state

FAULT
Error occurred

VE (Voltage Enabled)
Voltage applied

QS (Quick Stop)
Value = "0": Controller is in the "Quick stop" state

SOD (Switched On Disabled)
Value = "1": Controller is in the "Switched on disabled" state

WARN (Warning)
Value = "1": Warning

SYNC (synchronization)

Value = "1": Controller is in sync with the fieldbus; value = "0": Controller is not in sync with the fieldbus

REM (Remote)

Remote (value of the bit is always "1")

TARG

Target reached

ILA (Internal Limit Reached)

Limit exceeded

OMS (Operation Mode Specific)

Meaning is dependent on the selected operating mode

CLA (Closed Loop Available)

Value = "1": Auto setup was successful and encoder index seen: closed loop mode possible

Listed in the following table are the bit masks that break down the state of the controller.

Statusword (6041_h)	State
xxxx xxxx x0xx 0000	Not ready to switch on
xxxx xxxx x1xx 0000	Switch on disabled
xxxx xxxx x01x 0001	Ready to switch on
xxxx xxxx x01x 0011	Switched on
xxxx xxxx x01x 0111	Operation enabled
xxxx xxxx x00x 0111	Quick stop active
xxxx xxxx x0xx 1111	Fault reaction active
xxxx xxxx x0xx 1000	Fault

6042h Vl Target Velocity

Function

Specifies the target speed in **user-defined units**.

Object description

Index	6042_h
Object name	Vl Target Velocity
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$00C8_h$
Firmware version	FIR-v1426

Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

6043h Vl Velocity Demand

Function

Specifies the current target speed in user units.

Object description

Index	6043_h
Object name	Vl Velocity Demand
Object Code	VARIABLE
Data type	INTEGER16
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

6044h Vl Velocity Actual Value

Function

Specifies the current actual speed in **user-defined units**.

In *open loop* mode, the source of this object can be set with object **320A**$_h$:03_h to either the internal, calculated value or to the encoder.

Object description

Index	6044_h
Object name	Vl Velocity Actual Value
Object Code	VARIABLE
Data type	INTEGER16
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

6046h Vl Velocity Min Max Amount

Function

This object can be used to set the minimum speed and maximum speed in **user-defined units**.

Object description

Index	6046_h
Object name	Vl Velocity Min Max Amount
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	MinAmount
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	MaxAmount
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$00004E20_h$

Description

Subindex 1 contains the minimum speed.

Subindex 2 contains the maximum speed.

If the value of the target speed (object **6042**$_h$) specified here is less than the minimum speed, the minimum speed applies and bit 11 (Internal Limit Reached) in **6041h Statusword**$_h$ is set.

A target speed greater than the maximum speed sets the speed to the maximum speed and bit 11 (Internal Limit Reached) in **6041h Statusword**$_h$ is set.

6048h Vl Velocity Acceleration

Function

Sets the acceleration ramp in Velocity Mode (see **Velocity**).

Object description

Index	6048$_h$
Object name	Vl Velocity Acceleration
Object Code	RECORD
Data type	VELOCITY_ACCELERATION_DECELERATION
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00$_h$
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02$_h$

Subindex	01$_h$
Name	DeltaSpeed
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	000001F4$_h$

Subindex	02$_h$
Name	DeltaTime
Data type	UNSIGNED16
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0001$_h$

Description

The acceleration is specified as a fraction in **user-defined units**:

Speed change per change in time.

Subindex 01_h: Contains the change in speed.

Subindex 02_h: Contains the change in time.

6049h VI Velocity Deceleration

Function

Sets the deceleration (deceleration ramp) in Velocity Mode (see **Velocity**).

Object description

Index	6049_h
Object name	VI Velocity Deceleration
Object Code	RECORD
Data type	VELOCITY_ACCELERATION_DECELERATION
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	DeltaSpeed
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$000001F4_h$

Subindex	02_h
Name	DeltaTime
Data type	UNSIGNED16
Access	read / write
PDO mapping	RX-PDO

Allowed values	
Preset value	0001_h

Description

The deceleration is specified as a fraction in **user-defined units**:

Speed change per change in time.

Subindex 01_h: Contains the change in speed.

Subindex 02_h: Contains the change in time.

604Ah VI Velocity Quick Stop

Function

This object defines the deceleration (deceleration ramp) if the Quick Stop state is initiated in **Velocity Mode**.

Object description

Index	$604A_h$
Object name	VI Velocity Quick Stop
Object Code	RECORD
Data type	VELOCITY_ACCELERATION_DECELERATION
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	DeltaSpeed
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00001388_h

Subindex	02_h

Name	DeltaTime
Data type	UNSIGNED16
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0001_h

Description

The deceleration is specified as a fraction in **user-defined units**:

Speed change per change in time.

Subindex 01_h: Contains the change in speed.

Subindex 02_h: Contains the change in time.

604Ch VI Dimension Factor

Function

The unit for speed values is defined here for the objects associated with **Velocity Mode**.

Object description

Index	$604C_h$
Object name	VI Dimension Factor
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	VI Dimension Factor Numerator
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	

Preset value	00000001_h

Subindex	02_h
Name	VI Dimension Factor Denominator
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$0000003C_h$

Description

If subindex 1 is set to the value "1" and subindex 2 is set to the value "1"; the speed is specified in revolutions per minute.

Otherwise, subindex 1 contains the denominator (multiplier) and subindex 2 contains the numerator (divisor) with which the internal speed values are converted to revolutions per second. If subindex 1 is set to the value "1" and subindex 2 is set to the value "60" (factory setting), the speed is specified in revolutions per minute (1 revolution per 60 seconds).

605Ah Quick Stop Option Code

Function

The object contains the action that is to be executed on a transition of the **CiA 402 Power State Machine** to the Quick Stop state.

Object description

Index	$605A_h$
Object name	Quick Stop Option Code
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0001_h
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode) and subsequent state change to "Switch on disabled"

Value	Description
2	Braking with "quick stop ramp" and subsequent state change to "Switch on disabled"
3 … 32767	Reserved

605Bh Shutdown Option Code

Function

This object contains the action that is to be executed on a transition of the **CiA 402 Power State Machine** from the *Operation enabled* state to the *Ready to switch on* state.

Object description

Index	$605B_h$
Object name	Shutdown Option Code
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0001_h
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode) and subsequent state change to "Switch on disabled"
2 … 32767	Reserved

605Ch Disable Option Code

Function

This object contains the action that is to be executed on a transition of the **CiA 402 Power State Machine** from the "Operation enabled" state to the "Switched on" state.

Object description

Index	$605C_h$
Object name	Disable Option Code
Object Code	VARIABLE
Data type	INTEGER16

Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0001_h
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode) and subsequent state change to "Switch on disabled"
2 … 32767	Reserved

605Dh Halt Option Code

Function

The object contains the action that is to be executed if bit 8 (Halt) is set in controlword **6040_h**.

Object description

Index	$605D_h$
Object name	Halt Option Code
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0001_h
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … 0	Reserved
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode)
2	Braking with "quick stop ramp" (deceleration (deceleration ramp) depending on operating mode)
3 … 32767	Reserved

605Eh Fault Option Code

Function

The object contains the action specifying how the motor is to be brought to a standstill in case of an error.

Object description

Index	$605E_h$
Object name	Fault Option Code
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0002_h
Firmware version	FIR-v1426
Change history	

Description

Value	Description
-32768 … -1	Reserved
0	Immediate stop
1	Braking with "slow down ramp" (deceleration (deceleration ramp) depending on operating mode)
2	Braking with "quick stop ramp" (deceleration (deceleration ramp) depending on operating mode)
3 … 32767	Reserved

6060h Modes Of Operation

Function

The desired operating mode is entered in this object.

Object description

Index	6060_h
Object name	Modes Of Operation
Object Code	VARIABLE
Data type	INTEGER8
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	

Preset value	00_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

Description

Mode	Description
-2	Auto setup
-1	Clock-direction mode
0	No mode change/no mode assigned
1	Profile Position Mode
2	Velocity Mode
3	Profile Velocity Mode
4	Profile Torque Mode
5	Reserved
6	Homing Mode
7	Interpolated Position Mode
8	Cyclic Synchronous Position Mode
9	Cyclic Synchronous Velocity Mode
10	Cyclic Synchronous Torque Mode

6061h Modes Of Operation Display

Function

Indicates the current operating mode. See also **6060h Modes Of Operation**.

Object description

Index	6061_h
Object name	Modes Of Operation Display
Object Code	VARIABLE
Data type	INTEGER8
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00_h
Firmware version	FIR-v1426
Change history	

6062h Position Demand Value

Function

Indicates the current demand position in **user-defined units**.

Object description

Index	6062_h
Object name	Position Demand Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

6063h Position Actual Internal Value

Function

Contains the current rotary encoder position in increments. Unlike objects **6062_h** and **6064_h**, this value is not set to "0" following a **Homing** operation.

Note

If the encoder resolution in object **2052_h** = `0`, the numerical values of this object are invalid.

Object description

Index	6063_h
Object name	Position Actual Internal Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

6064h Position Actual Value

Function

Contains the current actual position in **user-defined units**.

In *open loop* mode, the source of this object can be set with object **320A**$_h$:04$_h$ to either the internal, calculated value or to the encoder.

Note

If the encoder resolution in object **2052**$_h$ = 0, the numerical values of this object are invalid.

Object description

Index	6064$_h$
Object name	Position Actual Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000$_h$
Firmware version	FIR-v1426
Change history	

6065h Following Error Window

Function

Defines the maximum allowed **following error** in **user-defined units** symmetrically to the **demand position**.

Object description

Index	6065$_h$
Object name	Following Error Window
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000100$_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1504: "Savable" entry changed from "no" to "yes, category: application".

Description

If the actual position deviates so much from the demand position that the value of this object is exceeded, bit 13 in object **6041**$_h$ is set. The deviation must last longer than the time in object **6066**$_h$.

If the value of the "Following Error Window" is set to "FFFFFFFF"$_h$, following error monitoring is switched off.

A reaction to the following error can be set in object **3700**$_h$. If a reaction is defined, an error is also entered in object **1003**$_h$.

6066h Following Error Time Out

Function

Time in milliseconds until a larger following error results in an error message.

Object description

Index	6066$_h$
Object name	Following Error Time Out
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0064$_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1504: "Savable" entry changed from "no" to "yes, category: application".

Description

If the actual position deviates so much from the demand position that the value of object **6065**$_h$ is exceeded, bit 13 in object **6041**$_h$ is set. The deviation must persist for longer than the time defined in this object.

A reaction to the following error can be set in object **3700**$_h$. If a reaction is defined, an error is also entered in object **1003**$_h$.

6067h Position Window

Function

Specifies a range symmetrical to the target position within which that target is considered having been met in modes **Profile Position** and **Interpolated Position Mode**.

Object description

Index	6067$_h$
Object name	Position Window
Object Code	VARIABLE

Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$0000000A_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1504: "Savable" entry changed from "no" to "yes, category: application".

Description

If the current position deviates from the target position by less than the value of this object, bit 10 in object **6041_h** is set. The condition must be satisfied for longer than the time defined in object **6066_h**.

If the value is set to "FFFFFFFF"$_h$, monitoring is switched off.

6068h Position Window Time

Function

The current position must be within the "Position Window" (**6067_h**) for this time in milliseconds for the target position to be considered having been met in the **Profile Position** and **Interpolated Position Mode** modes.

Object description

Index	6068_h
Object name	Position Window Time
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0064_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1504: "Savable" entry changed from "no" to "yes, category: application".

Description

If the current position deviates from the target position by less than the value of object **6067_h**, bit 10 in object **6041_h** is set. The condition must be satisfied for longer than the time defined in object **6066_h**.

606Bh Velocity Demand Value

Function

Speed specification in **user-defined units** for the controller in **Profile Velocity Mode**.

Object description

Index	$606B_h$
Object name	Velocity Demand Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

Description

This object contains the output of the ramp generator, which simultaneously serves as the preset value for the speed controller.

606Ch Velocity Actual Value

Function

Current actual speed in **user-defined units**.

Object description

Index	$606C_h$
Object name	Velocity Actual Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

606Dh Velocity Window

Function

Specifies a symmetrical range relative to the target speed within which the target is considered having been met in the **Profile Velocity** mode.

Object description

Index	$606D_h$

Object name	Velocity Window
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$001E_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application".

Description

If the current speed deviates from the set speed by less than the value of this object, bit 10 in object **6041_h** is set. The condition must be satisfied for longer than the time defined in object **6066_h** (see also **statusword in Profile Velocity Mode**).

606Eh Velocity Window Time

Function

The current speed must be within the "Velocity Window" (**$606D_h$**) for this time (in milliseconds) for the target to be considered having been met.

Object description

Index	$606E_h$
Object name	Velocity Window Time
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application".

Description

Description

If the current speed deviates from the set speed by less than the value of object **$606D_h$**, bit 10 in object **6041_h** is set. The condition must be satisfied for longer than the time defined in object **6066** (see also **statusword in Profile Velocity Mode**).

6071h Target Torque

Function

This object contains the target torque for the **Profile Torque** and **Cyclic Synchronous Torque** modes in tenths of a percent of the rated torque.

Object description

Index	6071_h
Object name	Target Torque
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

Description

This object is calculated as thousandths of the torque, e.g., the value "500" means "50%" of the rated torque; "1100" is equivalent to 110%. The rated torque corresponds to the rated current in object **$203B_h$**:01.

The target torque may not exceed the peak torque (proportional to the peak current in **2031_h**).

6072h Max Torque

Function

The object describes the maximum torque for the **Profile Torque** and **Cyclic Synchronous Torque** modes in tenths of a percent of the rated torque.

Object description

Index	6072_h
Object name	Max Torque
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

Description

This object is calculated as thousandths of the torque, e.g., the value "500" means "50%" of the rated torque; "1100" is equivalent to 110%. The rated torque corresponds to the rated current in object **203B**$_h$:01.

The target torque may not exceed the peak torque (proportional to the peak current in **2031**$_h$).

6074h Torque Demand

Function

Current torque set value requested by the ramp generator in tenths of a percent of the nominal torque for the internal controller.

Object description

Index	6074$_h$
Object name	Torque Demand
Object Code	VARIABLE
Data type	INTEGER16
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	0000$_h$
Firmware version	FIR-v1426
Change history	

Description

This object is calculated as thousandths of the torque, e.g., the value "500" means "50%" of the rated torque; "1100" is equivalent to 110%. The rated torque corresponds to the rated current in object **203B**$_h$:01.

The target torque may not exceed the peak torque (proportional to the peak current in **2031**$_h$).

6077h Torque Actual Value

Function

This object indicates the current torque value in tenths of a percent of the nominal torque for the internal controller.

Object description

Index	6077$_h$
Object name	Torque Actual Value
Object Code	VARIABLE
Data type	INTEGER16
Savable	no
Access	read only
PDO mapping	TX-PDO

Allowed values	
Preset value	0000_h
Firmware version	FIR-v1540
Change history	

Description

This object is calculated as thousandths of the torque, e.g., the value "500" means "50%" of the rated torque; "1100" is equivalent to 110%. The rated torque corresponds to the rated current in object **$203B_h$**:01.

The target torque may not exceed the peak torque (proportional to the peak current in **2031_h**).

607Ah Target Position

Function

This object specifies the target position in **user-defined units** for the **Profile Position** and **Cyclic Synchronous Position** modes.

Object description

Index	$607A_h$
Object name	Target Position
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$00000FA0_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

607Bh Position Range Limit

Function

Contains the minimum and maximum position in **user-defined units**.

Object description

Index	$607B_h$
Object name	Position Range Limit
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426

Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Min Position Range Limit
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Max Position Range Limit
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

If this range is exceeded or not reached, an overflow occurs. To prevent this overflow, limit values for the target position can be set in object **607D**$_h$ ("Software Position Limit").

607Ch Home Offset

Function

Specifies the difference between the zero position of the controller and the reference point of the machine in **user-defined units**.

Object description

Index	$607C_h$
Object name	Home Offset
Object Code	VARIABLE
Data type	INTEGER32

Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

607Dh Software Position Limit

Function

Defines the limit positions relative to the reference point of the application in **user-defined units**.

Object description

Index	$607D_h$
Object name	Software Position Limit
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Min Position Limit
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Subindex	02_h
Name	Max Position Limit
Data type	INTEGER32

Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

The target position must lie within the limits set here. Prior to every check, the respective Home Offset (**$607C_h$**) is subtracted:

Corrected Min Position Limit = Min Position Limit–Home Offset

Corrected Max Position Limit = Max Position Limit–Home Offset.

607Eh Polarity

Function

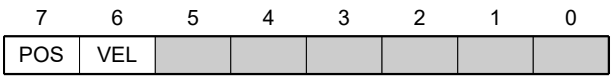
With this object, the direction of rotation can be reversed.

Object description

Index	$607E_h$
Object name	Polarity
Object Code	VARIABLE
Data type	UNSIGNED8
Savable	yes, category: application
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h
Firmware version	FIR-v1426
Change history	

Description

The following generally applies for direction reversal: If a bit is set to the value "1", reversal is activated. If the value is "0", the direction of rotation is as described in the respective mode.



VEL (Velocity)

Direction of rotation reversal in the following modes:

- **Profile Velocity Mode**
- **Cyclic Synchronous Velocity Mode**
- **Velocity Mode**

POS (Position)

Direction of rotation reversal in the following modes:

- **Profile Position Mode**
- **Cyclic Synchronous Position Mode**

6081h Profile Velocity

Function

Specifies the maximum travel speed in **user-defined units**.

Object description

Index	6081_h
Object name	Profile Velocity
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$000001F4_h$
Firmware version	FIR-v1426
Change history	

6082h End Velocity

Function

Specifies the speed at the end of the traveled ramp in **user-defined units**.

Object description

Index	6082_h
Object name	End Velocity
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

6083h Profile Acceleration

Function

Specifies the maximum acceleration in **user-defined units**.

Object description

Index	6083_h
Object name	Profile Acceleration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$000001F4_h$
Firmware version	FIR-v1426
Change history	

6084h Profile Deceleration

Function

Specifies the maximum deceleration (deceleration ramp) in **user-defined units**.

Object description

Index	6084_h
Object name	Profile Deceleration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$000001F4_h$
Firmware version	FIR-v1426
Change history	

6085h Quick Stop Deceleration

Function

Specifies the maximum Quick Stop Deceleration in **user-defined units**.

Object description

Index	6085_h
Object name	Quick Stop Deceleration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write

PDO mapping	RX-PDO
Allowed values	
Preset value	00001388_h
Firmware version	FIR-v1426
Change history	

6086h Motion Profile Type

Function

Specifies the ramp type for the **Profile Position** and **Profile Velocity** modes.

Object description

Index	6086_h
Object name	Motion Profile Type
Object Code	VARIABLE
Data type	INTEGER16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0000_h
Firmware version	FIR-v1426
Change history	

Description

Value = "0": = Trapezoidal ramp

Value = "3": Ramp with limited jerk

6087h Torque Slope

Function

This object contains the slope of the torque in Torque mode.

Object description

Index	6087_h
Object name	Torque Slope
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Firmware version	FIR-v1426
Change history	

Description

This object is calculated as thousandths of the torque, e.g., the value "500" means "50%" of the rated torque; "1100" is equivalent to 110%. The rated torque corresponds to the rated current in object **203B$_h$**:01.

The target torque may not exceed the peak torque (proportional to the peak current in **2031$_h$**).

608Fh Position Encoder Resolution

Function

Virtual encoder increments per revolution. See chapter **User-defined units**.

Object description

Index	608F$_h$
Object name	Position Encoder Resolution
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00$_h$
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02$_h$

Subindex	01$_h$
Name	Encoder Increments
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	000007D0$_h$

Subindex	02$_h$
Name	Motor Revolutions

Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Description

Position Encoder Resolution = Encoder Increments (**608F**$_h$:01_h) / Motor Revolutions (**608F**$_h$:02_h)

6091h Gear Ratio

Function

Number of motor revolutions per output shaft revolution.

Object description

Index	6091_h
Object name	Gear Ratio
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Motor Revolutions
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	02_h
Name	Shaft Revolutions

Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Description

Gear Ratio = Motor Revolutions (**6091_h**:01_h) / Shaft Revolutions (**6091_h**:02_h)

6092h Feed Constant

Function

Feed in the case of a linear drive; in **user-defined units** per revolution on the drive.

Object description

Index	6092_h
Object name	Feed Constant
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Feed
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000001_h

Subindex	02_h
Name	Shaft Revolutions

Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000001_h

Description

Feed Constant = Feed (**6092_h**:01_h) / Shaft Revolutions (**6092_h**:02_h)

6098h Homing Method

Function

This object defines the **Homing method** in **Homing** Mode.

Object description

Index	6098_h
Object name	Homing Method
Object Code	VARIABLE
Data type	INTEGER8
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	23_h
Firmware version	FIR-v1426
Change history	

6099h Homing Speed

Function

Specifies the speeds for Homing Mode (**6098_h**) in **user-defined units**.

Object description

Index	6099_h
Object name	Homing Speed
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Speed During Search For Switch
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000032_h

Subindex	02_h
Name	Speed During Search For Zero
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$0000000A_h$

Description

This value is calculated with the numerator in object **2061_h** and the dominator in object **2062_h**.

The speed for the search for the switch is specified in subindex 1.

The (lower) speed for the search for the reference position is specified in subindex 2.

Note

- The speed in subindex 2 is simultaneously the initial speed when starting the acceleration ramp. If this is set too high, the motor loses steps or fails to turn at all. If the setting is too high, the index marking will be overlooked. The speed in subindex 2 should therefore be less than 1000 steps per second.
- The speed in subindex 1 must be greater than the speed in subindex 2.

609Ah Homing Acceleration

Function

Specifies the acceleration ramp for Homing Mode in **user-defined units**.

Object description

Index	$609A_h$
Object name	Homing Acceleration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	$000001F4_h$
Firmware version	FIR-v1426
Change history	

Description

The ramp is only used when starting up. When the switch is reached, the motor immediately switches to the lower speed; when the end position is reached, it immediately stops.

60A4h Profile Jerk

Function

In the case of a ramp with limited jerk, the size of the jerk can be entered in this object. An entry with the value "0" means that the jerk is not limited.

Object description

Index	$60A4_h$
Object name	Profile Jerk
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1614: "Name" entry changed from "End Acceleration Jerk" to "Begin Deceleration Jerk". Firmware version FIR-v1614: "Name" entry changed from "Begin Deceleration Jerk" to "End Acceleration Jerk".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	

Preset value	04_h

Subindex	01_h
Name	Begin Acceleration Jerk
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	02_h
Name	Begin Deceleration Jerk
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	03_h
Name	End Acceleration Jerk
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Subindex	04_h
Name	End Deceleration Jerk
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	$000003E8_h$

Description

- Subindex 01_h (*Begin Acceleration Jerk*): Initial jerk during acceleration
- Subindex 02_h (*Begin Deceleration Jerk*): Initial jerk during braking
- Subindex 03_h (*End Acceleration Jerk*): Final jerk during acceleration
- Subindex 04_h (*End Deceleration Jerk*): Final jerk during braking

60C1h Interpolation Data Record

Function

This object contains the demand position in **user-defined units** for the interpolation algorithm for the **Interpolated Position** operating mode.

Object description

Index	$60C1_h$
Object name	Interpolation Data Record
Object Code	ARRAY
Data type	INTEGER32
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1512
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	1st Set-point
Data type	INTEGER32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000_h

Description

The value is taken over at the next synchronization time.

60C2h Interpolation Time Period

Function

This object contains the interpolation time.

Object description

Index	$60C2_h$
Object name	Interpolation Time Period
Object Code	RECORD
Data type	INTERPOLATION_TIME_PERIOD
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1426
Change history	

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	02_h

Subindex	01_h
Name	Interpolation Time Period Value
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	02_h
Name	Interpolation Time Index
Data type	INTEGER8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	FD_h

Description

The subindices have the following functions:

- 01_h: Interpolation time.
- 02_h: Power of ten of the interpolation time: must have the value -3 (corresponds to the time basis in milliseconds).

The following applies here: cycle time = value of **60C2**$_h$:01_h * 10 $^{\text{value of 60C2:02}}$ seconds.

60C4h Interpolation Data Configuration

Function

This object offers the maximum buffer size, specifies the configured buffer organization of the interpolated data and offers objects for defining the size of the record and for deleting the buffer. It is also used to store the position of other data points.

Object description

Index	$60C4_h$
Object name	Interpolation Data Configuration
Object Code	RECORD
Data type	INTERPOLATION_DATA_CONFIGURATION
Savable	yes, category: application
Access	read only
PDO mapping	no
Allowed values	
Preset value	
Firmware version	FIR-v1512
Change history	Firmware version FIR-v1540: "Access" table entry for subindex 05 changed from "read/write" to "write only". Firmware version FIR-v1540: "Access" table entry for subindex 06 changed from "read/write" to "write only". Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application". Firmware version FIR-v1650-B472161: "Access" table entry for subindex 01 changed from "read/write" to "read only".

Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	06_h

Subindex	01_h
Name	MaximumBufferSize
Data type	UNSIGNED32
Access	read only
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	02_h
Name	ActualBufferSize
Data type	UNSIGNED32
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00000001_h

Subindex	03_h
Name	BufferOrganization
Data type	UNSIGNED8
Access	read / write
PDO mapping	no
Allowed values	
Preset value	00_h

Subindex	04_h
Name	BufferPosition
Data type	UNSIGNED16
Access	read / write
PDO mapping	no
Allowed values	
Preset value	0001_h

Subindex	05_h
Name	SizeOfDataRecord
Data type	UNSIGNED8
Access	write only
PDO mapping	no
Allowed values	
Preset value	04_h

Subindex	06_h
Name	BufferClear
Data type	UNSIGNED8

Access	write only
PDO mapping	no
Allowed values	
Preset value	00_h

Description

The value of subindex 01_h contains the maximum possible number of interpolated records.

The value of subindex 02_h contains the current number of interpolated records.

If subindex 03_h is "00_h", this means a FIFO buffer organization; if it is "01_h", it specifies a ring buffer organization.

The value of subindex 04_h is unitless and specifies the next free buffer entry point.

The value of subindex 05_h is specified in units of "byte". If the value "00_h" is written in subindex 06_h, it deletes the received data in the buffer, deactivates access and deletes all interpolated records. If the value "01_h" is written in subindex 06_h, it activates access to the input buffer.

60C5h Max Acceleration

Function

This object contains the maximum permissible acceleration for the **Profile Position** and **Profile Velocity** modes.

Object description

Index	$60C5_h$
Object name	Max Acceleration
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00001388_h
Firmware version	FIR-v1426
Change history	

60C6h Max Deceleration

Function

This object contains the maximum permissible deceleration (deceleration ramp) for the **Profile Position** and **Profile Velocity** modes.

Object description

Index	$60C6_h$
Object name	Max Deceleration
Object Code	VARIABLE

Data type	UNSIGNED32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00001388_h
Firmware version	FIR-v1426
Change history	

60F2h Positioning Option Code

Function

The object describes the positioning behavior in **Profile Position** mode.

Object description

Index	$60F2_h$
Object name	Positioning Option Code
Object Code	VARIABLE
Data type	UNSIGNED16
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	0001_h
Firmware version	FIR-v1446
Change history	Firmware version FIR-v1614: "Savable" entry changed from "no" to "yes, category: application".

Description

Only the following bits are supported at the present time:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MS	RESERVED [3]			IP OPTION [4]				RADO [2]		RRO [2]		CIO [2]		REL. OPT. [2]	

REL. OPT. (Relative Option)

These bits determine the behavior with relative rotating movement in "Profile Position" mode if bit 6 of controlword **6040_h** = "1" is set.

Bit 1	Bit 0	Definition
0	0	Position movements are executed relative to the previous (internal absolute) target position (each relative to 0 if there is no previous target position)
0	1	Position movements are executed relative to the preset value (or output) of the ramp generator.

Bit 1	Bit 0	Definition
1	0	Position movements are performed relative to the current position (object **6064**$_h$).
1	1	Reserved

RRO (Request-Response Option)

These bits determine the behavior when passing controlword **6040**$_h$ bit 5 ("new setpoint") – in this case, the controller releases the bit itself. This eliminates the need to externally reset the bit to "0" afterwards. After the bit is set to the value "0" by the controller, bit 12 ("setpoint acknowledgment") is also set to the value "0" in statusword **6041**$_h$.

Note

These options cause the controller to modify object controlword **6040**$_h$.

Bit 5	Bit 4	Definition
0	0	The functionality is as described under **Setting travel commands**.
0	1	The controller releases the "new setpoint" bit as soon as the current targeted movement has reached its target.
1	0	The controller releases the "new setpoint" bit as soon this is possible for the controller.
1	1	Reserved

RADO (Rotary Axis Direction Option)

These bits determine the direction of rotation in "Profile Position" mode.

Bit 7	Bit 6	Definition
0	0	Normal positioning similar to a linear axis: If one of the "Position Range Limits" – **607B**$_h$:01$_h$ and 02$_h$ – is reached or exceeded, the preset is automatically transferred to the other end of the limit. Only with this bit combination is a movement greater than the modulo value possible.
0	1	Positioning only in negative direction: If the target position is greater than the current position, the axis moves to the target position via the "Min Position Range Limit" from object **607D**$_h$:01$_h$.
1	0	Positioning only in positive direction: If the target position is less than the current position, the axis moves to the target position via the "Max Position Range Limit" from object **607D**$_h$:01$_h$.
1	1	Positioning with the shortest distance to the target position. If the difference between the current position and the target position in a 360° system is less than 180°, the axis moves in the positive direction.

60F4h Following Error Actual Value

Function

This object contains the current following error in **user-defined units**.

Object description

Index	$60F4_h$
Object name	Following Error Actual Value
Object Code	VARIABLE
Data type	INTEGER32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

60FDh Digital Inputs

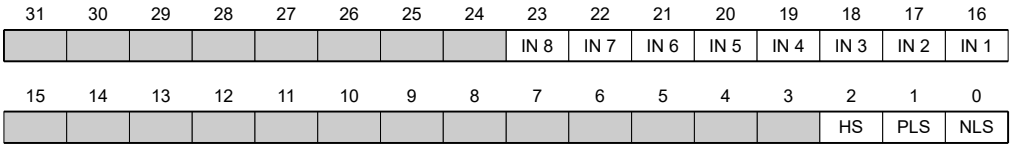
Function

With this object, the **digital inputs** of the motor can be read.

Object description

Index	$60FD_h$
Object name	Digital Inputs
Object Code	VARIABLE
Data type	UNSIGNED32
Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	00000000_h
Firmware version	FIR-v1426
Change history	

Description



NLS (Negative Limit Switch)
Negative limit switch

PLS (Positive Limit Switch)
Positive limit switch

HS (Home Switch)
Home switch

IN n (Input n)
Input n – the number of used bits is dependent on the given controller.

60FEh Digital Outputs

Function

With this object, the **digital outputs** of the motor can be written.

Object description

Index	$60FE_h$
Object name	Digital Outputs
Object Code	ARRAY
Data type	UNSIGNED32
Savable	yes, category: application
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

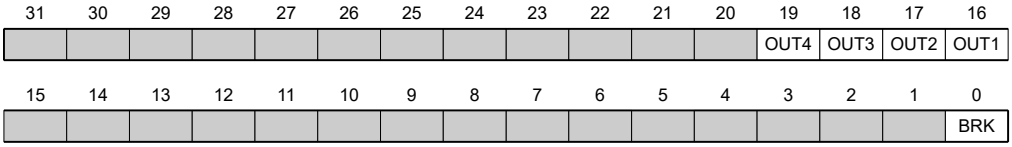
Value description

Subindex	00_h
Name	Highest Sub-index Supported
Data type	UNSIGNED8
Access	read only
PDO mapping	no
Allowed values	
Preset value	01_h

Subindex	01_h
Name	Digital Outputs #1
Data type	UNSIGNED32
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000001_h

Description

To write the outputs, the entries in object **3250**$_h$, subindex 02$_h$ to 05$_h$, must also be taken into account.



BRK (Brake)

Bit for the brake output (if the controller supports this function).

OUT n (Output No n)

Bit for the respective digital output; the exact number of digital outputs is dependent on the controller.

60FFh Target Velocity

Function

In this object, the target speed for the **Profile Velocity** and **Cyclic Synchronous Velocity** modes is entered in **user-defined units**.

Object description

Index	60FF$_h$
Object name	Target Velocity
Object Code	VARIABLE
Data type	INTEGER32
Savable	yes, category: application
Access	read / write
PDO mapping	RX-PDO
Allowed values	
Preset value	00000000$_h$
Firmware version	FIR-v1426
Change history	Firmware version FIR-v1626: "Savable" entry changed from "no" to "yes, category: application".

6502h Supported Drive Modes

Function

The object describes the supported operating modes in object **6060**$_h$.

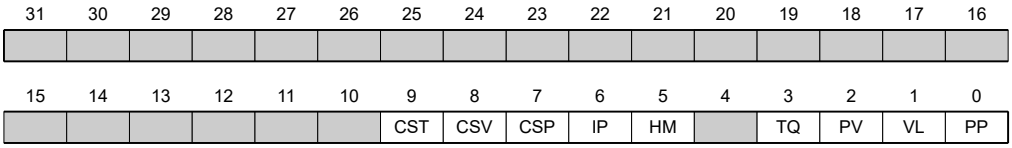
Object description

Index	6502$_h$
Object name	Supported Drive Modes
Object Code	VARIABLE
Data type	UNSIGNED32

Savable	no
Access	read only
PDO mapping	TX-PDO
Allowed values	
Preset value	000003EF$_h$
Firmware version	FIR-v1426
Change history	

Description

The set bit specifies whether the respective mode is supported. If the value of the bit is "0", the mode is not supported.



PP
Profile Position Mode

VL
Velocity Mode

PV
Profile Velocity Mode

TQ
Torque Mode

HM
Homing Mode

IP
Interpolated Position Mode

CSP
Cyclic Synchronous Position Mode

CSV
Cyclic Synchronous Velocity Mode

CST
Cyclic Synchronous Torque Mode

6505h Http Drive Catalogue Address

Function

This object contains the manufacturer's web address as a character string.

Object description

Index	6505_h
Object name	Http Drive Catalogue Address
Object Code	VARIABLE
Data type	VISIBLE_STRING
Savable	no
Access	read only
PDO mapping	no
Allowed values	
Preset value	http://www.nanotec.de
Firmware version	FIR-v1426
Change history	

12 Copyrights

12.1 Introduction

Integrated in the Nanotec software are components from products from external software manufacturers. In this chapter, you will find the copyright information regarding the used external software sources.

12.2 AES

FIPS-197 compliant AES implementation

Based on XySSL: Copyright (C) 2006-2008 Christophe Devine

Copyright (C) 2009 Paul Bakker <polarssl_maintainer at polarssl dot org>

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution; or, the application vendor's website must provide a copy of this notice.
- Neither the names of PolarSSL or XySSL nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The AES block cipher was designed by Vincent Rijmen and Joan Daemen.

http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

12.3 MD5

MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm

Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

12.4 uIP

Copyright (c) 2005, Swedish Institute of Computer Science

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

12.5 DHCP

Copyright (c) 2005, Swedish Institute of Computer Science

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

12.6 CMSIS DSP Software Library

Copyright (C) 2010 ARM Limited. All rights reserved.

12.7 FatFs

FatFs - FAT file system module include file R0.08 (C)ChaN, 2010

FatFs module is a generic FAT file system module for small embedded systems.

This is a free software that opened for education, research and commercial

developments under license policy of following trems.

Copyright (C) 2010, ChaN, all right reserved.

The FatFs module is a free software and there is NO WARRANTY.

No restriction on use. You can use, modify and redistribute it for

personal, non-profit or commercial product UNDER YOUR RESPONSIBILITY.

Redistributions of source code must retain the above copyright notice.

12.8 Protothreads

Protothread class and macros for lightweight, stackless threads in C++.

This was "ported" to C++ from Adam Dunkels' protothreads C library at: http://www.sics.se/~adam/pt/

Originally ported for use by Hamilton Jet (www.hamiltonjet.co.nz) by Ben Hoyt, but stripped down for public release. See his blog entry about it for more information: http://blog.micropledge.com/2008/07/protothreads/

Original BSD-style license

Copyright (c) 2004-2005, Swedish Institute of Computer Science.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the Institute and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the Institute or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

12.9 lwIP

Copyright (c) 2001-2004 Swedish Institute of Computer Science.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This file is part of the lwIP TCP/IP stack.

Author: Adam Dunkels <adam@sics.se>