CANopen Programming

TECHNOSOFT

User Manual

TECHNOSOFT

CANopen Programming
User Manual

P091.063.UM.0210

Technosoft S.A.

Avenue des Alpes 20
CH-2000 NEUCHATEL
Switzerland
Tel.: +41 (0) 32 732 5500
Fax: +41 (0) 32 732 5504
contact@technosoftmotion.com
www.technosoftmotion.com

Read This First

Whilst Technosoft believes that the information and guidance given in this manual is correct, all parties must rely upon their own skill and judgment when making use of it. Technosoft does not assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

About This Manual

This book describes how to program the CANopen versions of Technosoft intelligent drives. These drives support CANopen protocol in conformance with the **DS-301** communication profile and the **DSP-402** device profile. The manual presents the object dictionary associated with these two profiles. As an option, it is also available on request the **DS-401** device profile. The manual also explains how to combine the Technosoft Motion Language commands and the **CANopen** protocol commands in order to distribute the application between the CANopen master and the Technosoft drives. In order to operate the Technosoft IDM680 drives, you need to pass through 3 steps:

- **Step 1 Hardware installation**
- **Step 2 Drive setup** using Technosoft **EasySetUp** software for drive commissioning
- **Step 3 Motion programming** using one of the options:
  A. A **CANopen master**
  B. The drive **built-in motion controller** executing a Technosoft Motion Language (**TML**) program developed using Technosoft **EasyMotion Studio** software
  C. A **TML_LIB motion library for PCs** (Windows or Linux)
  D. A **TML_LIB motion library for PLCs**
  E. A **distributed control** approach which combines the above options, like for example a host calling motion functions programmed on the drives in TML

Scope of This Manual

This manual applies to the following families of Technosoft intelligent drives and motors. Certain features are different depending on the family (see a detailed comparison table in the appendix).

1. **MotionChip II** family: IPS110, PIM2401, PIM2403, ISCM4805, ISCM8005, IPS210, ISCM4805 DIN, ISCM8005 DIN, IBL2401, IBL2403, IDM240, IDM640, IDM3000, IM233-MA, IS23-MA

Each product from these families can be delivered in 2 versions:

- Standard – using highly optimized Technosoft TMLCAN protocol via CANbus
- CANopen – using CANopen DS301 and DSP402 communication and device profiles. DS401 also available on request.

This manual refers to CANopen versions only. The CANopen versions provide slightly less motion programming flexibility compared with the standard versions and fewer motion modes options, but can be integrated in existent applications already using CANopen communication.

Important! CANopen versions of Technosoft intelligent drives and motors have other order codes than the standard versions. They also use a different firmware.

2. **MotionChip III** family: IDM680

Notational Conventions

This document uses the following conventions:

- **TML** – Technosoft Motion Language
- **IU** – drive/motor internal units
- **ControlWord.5** – bit 5 of ControlWord data
- **MotionChip II** – generic name for the second generation family of Technosoft intelligent drives or motors
- **MotionChip III** – generic name for the third generation family of Technosoft intelligent drives

Related Documentation

Help of the EasySetUp software – describes how to use **EasySetUp** to quickly setup any Technosoft drive for your application using only 2 dialogues. The output of EasySetUp is a set of setup data that can be downloaded into the drive EEPROM or saved on a PC file. At power-on, the drive is initialized with the setup data read from its EEPROM. With EasySetUp it is also possible to retrieve the complete setup information from a previously programmed drive. EasySetUp includes a firmware programmer, which allows you to update your drive firmware to the latest revision. **EasySetUp can be downloaded free of charge from Technosoft web page**

User Manual of each intelligent drive or motor – describes the hardware including the technical data, the connectors, the wiring diagrams needed for installation and detailed setup information.

Help of the EasyMotion Studio software – describes how to use the EasyMotion Studio to create motion programs using the Technosoft Motion Language (TML). EasyMotion Studio platform includes **EasySetUp** for the drive/motor setup, and a **Motion Wizard** for the motion programming. The Motion Wizard provides a simple, graphical way of creating motion programs and automatically generates all the TML instructions. *With EasyMotion Studio you can fully benefit from a key advantage of Technosoft drives – their capability to execute complex motions without requiring an external motion controller, thanks to their built-in motion controller.* **A demo version of EasyMotion Studio (with the corresponding EasySetUp part fully functional) can be downloaded free of charge from Technosoft web page**

TML_LIB v2.0 **(part no. P091.040.v20.UM.xxxx)** – explains how to program in **C, C++, Visual Basic or Delphi Pascal** a motion application for the Technosoft intelligent

drives using TML_LIB v2.0 motion control library for PCs. The manual includes over 40 ready-to-run examples that can be executed on **Windows** or **Linux**

TML_LIB_LabVIEW v2.0 **(part no. P091.040.LABVIEW.v20.UM.xxxx)** – explains how to program in **LabVIEW** a motion application for the Technosoft intelligent drives using TML_LIB_LabVIEW v2.0 motion control library for PCs. The manual includes over 40 ready-to-run examples.

TML_LIB_S7 **(part no. P091.040.S7.UM.xxxx)** – explains how to program a PLC Siemens series S7-300 or S7-400 with a motion application for the Technosoft intelligent drives using TML_LIB_S7 motion control library. The manual includes over 40 ready-to-run examples. The library is PLCOpen compatible.

TML_LIB_CJ1 **(part no. P091.040.CJ1.UM.xxxx)** – explains how to program a PLC Omron series CJ1 with a motion application for the Technosoft intelligent drives using TML_LIB_CJ1 motion control library for PCs. The manual includes over 40 ready-to-run examples. The library is **PLCOpen** compatible.

TechnoCAN **(part no. P091.063.TechnoCAN.UM.xxxx)** – presents TechnoCAN protocol – an extension of the CANopen communication profile used for TML commands

If you Need Assistance …

If you want to …	**Contact Technosoft at …**
Visit Technosoft online	World Wide Web: http://www.technosoftmotion.com/
Receive general information or assistance	World Wide Web: http://www.technosoftmotion.com/ Email: contact@technosoftmotion.com
Ask questions about product operation or report suspected problems Make suggestions about or report errors in documentation	Fax: (41) 32 732 55 04 Email: hotline@technosoftmotion.com

Contents

1. Introduction

1.1. Installing EasySetUp

EasySetUp is a PC software platform for the setup of the Technosoft drives. It can be downloaded ***free of charge*** from Technosoft web page. EasySetUp comes with an ***Update via Internet tool*** through which you can check if your software version is up-to-date, and when necessary download and install the latest updates.

EasySetUp can be installed independently or together with **EasyMotion Studio** platform for motion programming using TML. You will need EasyMotion Studio only if you plan to use the advanced features presented in **Chapter 16**. A **demo version of EasyMotion Studio** including the **fully functional version of EasySetUp** can be downloaded free of charge from Technosoft web page.

On request, EasySetUp can be provided on a CD too. In this case, after installation, use the update via internet tool to check for the latest updates. Once you have started the installation package, follow its indications.

1.2. Getting Started with EasySetUp

Using EasySetUp you can quickly setup a drive for your application. The drive can be:

- directly connected with your PC via a serial RS 232 link
- any drive from a CANbus network where the PC is serially linked with one of the other drives.

The output of EasySetUp is a set of *setup data*, which can be downloaded into the drive EEPROM or saved on your PC for later use.

EasySetUp includes a set of evaluation tools like the Data Logger, the Control Panel and the Command Interpreter which help you to quickly measure, check and analyze your drive commissioning.

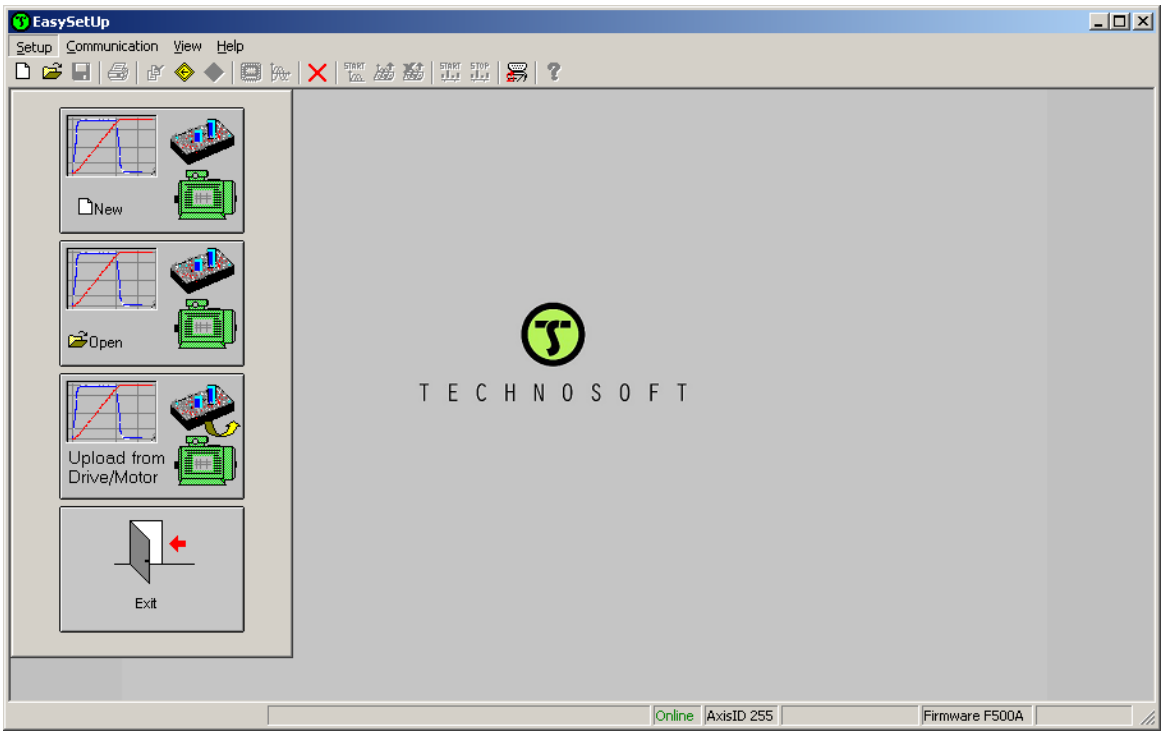
EasySetUp works with **setup** data. A **setup** contains all the information needed to configure and parameterize a Technosoft drive. This information is preserved in the drive EEPROM in the *setup table*. The setup table is copied at power-on into the RAM memory of the drive and is used during runtime. With EasySetUp it is also possible to retrieve the complete setup information from a drive previously programmed.

Note that with EasySetUp you do only your drive/motor commissioning. For motion programming you have the following options:

- Use a **CANopen** master
- Use **EasyMotion Studio** to create and download a **TML** program into the drive/motor memory and from the CANopen master send commands to execute the TML code and monitor the execution (see **Chapter 16**).

1.2.1. Establish communication

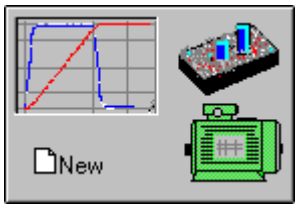
EasySetUp starts with an empty window from where you can create a **New** setup, **Open** a previously created setup which was saved on your PC, or **Upload** the setup from the drive/motor.



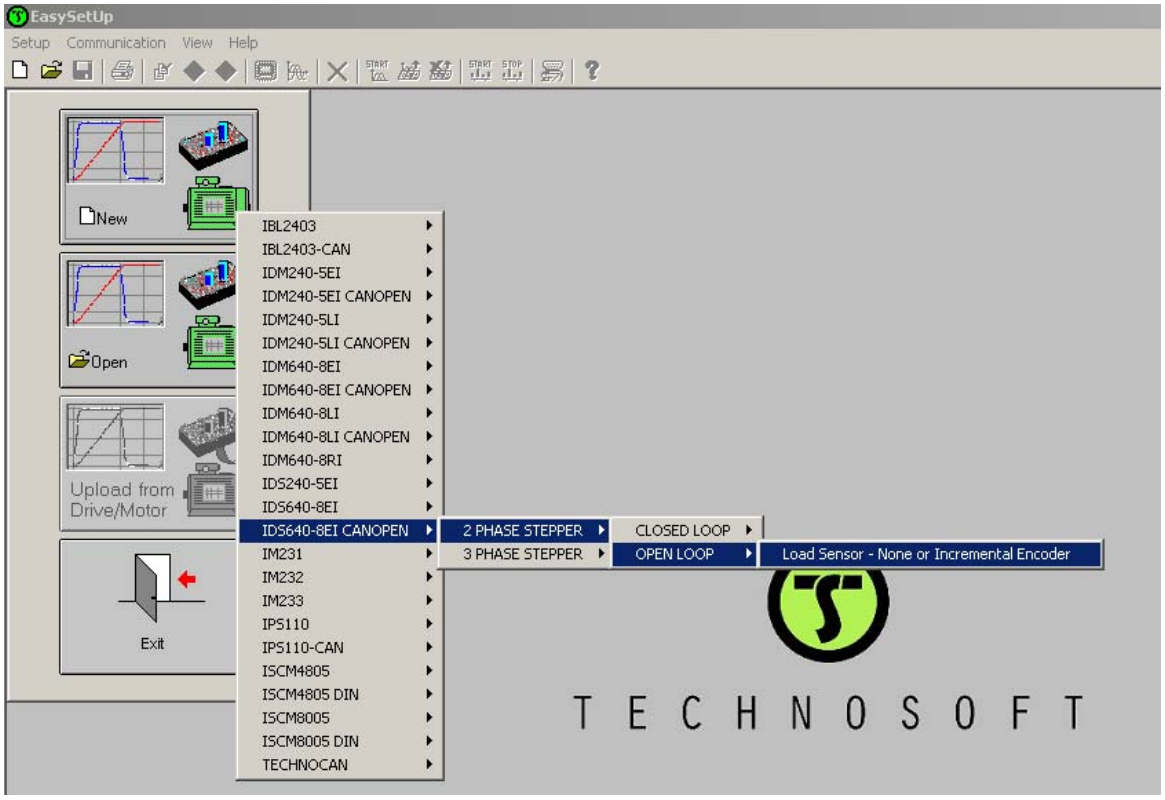
Before selecting one of the above options, you need to establish the communication with the drive you want to commission. Use menu command **Communication | Setup** to check/change your PC communication settings. Press the **Help** button of the dialogue opened. Here you can find detailed information about how to setup your drive and do the connections. Power on the drive, then close the Communication | Setup dialogue with OK. If the communication is established, EasySetUp displays in the status bar (the bottom line) the text "**Online**" plus the axis ID of your drive/motor and its firmware version. Otherwise the text displayed is "**Offline**" and a communication error message tells you the error type. In this case, return to the Communication | Setup dialogue, press the Help button and check troubleshoots.

Remark: *When first started, EasySetUp tries to communicate via RS-232 and COM1 with a drive having axis ID=255 (default communication settings). If your drive is powered with all the DIP switches OFF and it is connected to your PC port COM1 via an RS-232 cable, the communication shall establish automatically. If the drive has a different axis ID and you don't know it, select in the Communication | Setup dialogue at "Axis ID of drive/motor connected to PC" the option* ***Autodetected****.*

1.2.2. Setup drive/motor

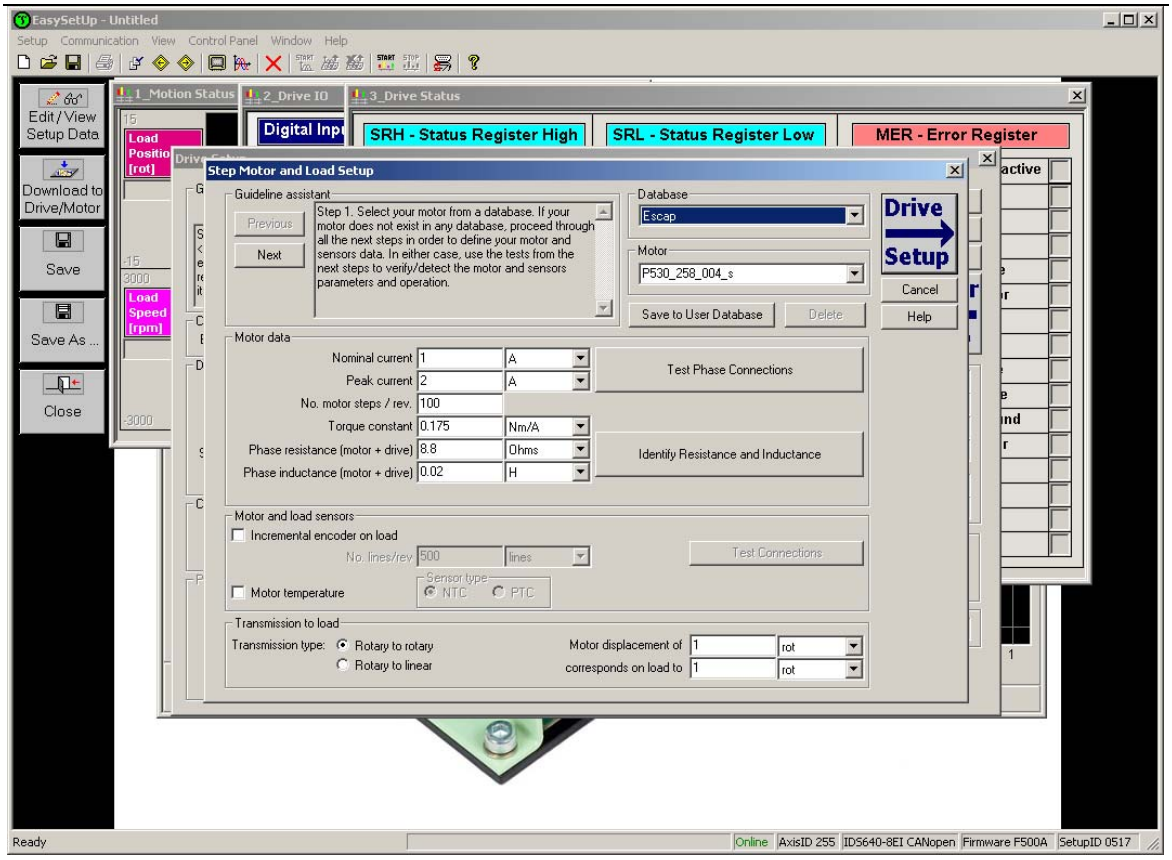
Press **New** button 

and select your drive type.



The selection continues with the motor technology (for example: 2 phase stepper, 3 phase stepper) the control mode (for example: open-loop or closed-loop) and type of feedback device (for example: none or incremental encoder).

The selection opens 2 setup dialogues: for **Motor Setup** and for **Drive setup** through which you can configure and parameterize a Technosoft drive, plus several predefined control panels customized for the product selected.



In the **Motor setup** dialogue you can introduce the data of your motor and the associated sensors. Data introduction is accompanied by a series of tests having as goal to check the connections to the drive and/or to determine or validate a part of the motor and sensors parameters. In the **Drive setup** dialogue you can configure and parameterize the drive for your application. In each dialogue you will find a **Guideline Assistant**, which will guide you through the whole process of introducing and/or checking your data. Close the Drive setup dialogue with **OK** to keep all the changes regarding the motor and the drive setup.

1.2.3. Download setup data to drive/motor

Press the **Download to Drive/Motor** button 

to download your setup data in the drive/motor EEPROM memory in the *setup table*. From now on, at each power-on, the setup data is copied into the drive/motor RAM memory which is used during runtime. It is also possible to **Save** 

the setup data on your PC and use it in other applications.

To summarize, you can define or change the setup data in the following ways:

- create a new setup data by going through the motor and drive dialogues
- use setup data previously saved in the PC
- upload setup data from a drive/motor EEPROM memory

1.2.4. Evaluate drive/motor behaviour (optional)

You can use the **Data Logger** or the **Control Panel** evaluation tools to quickly measure and analyze your application behavior. In case of errors like protections triggered, use the Drive Status control panel to find the cause.

1.3. Changing the drive Axis ID

The axis ID of a Technosoft drives drive can be set in 2 ways:

- Hardware (H/W) – in the range 1 to 31 or 255. The hardware selection can be done using DIP switches or jumpers, according to the drive model.
- Software – any value between 1 and 255, stored in the setup table

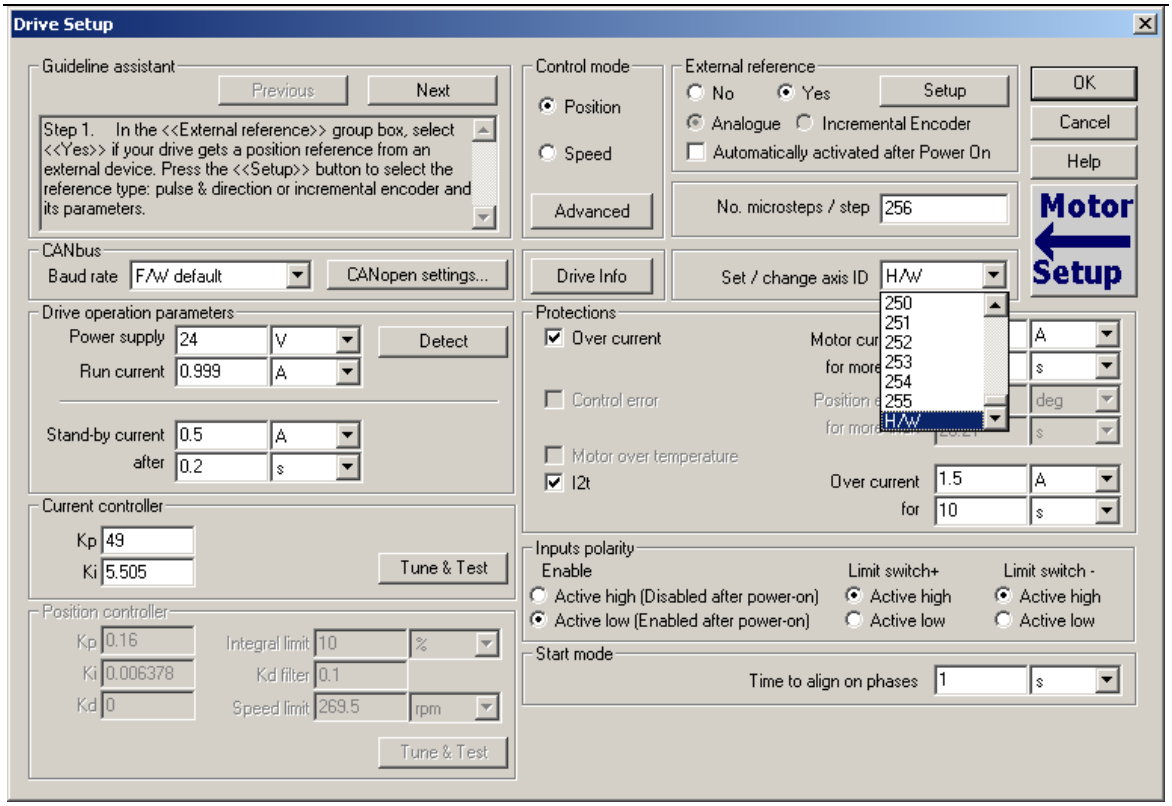
The axis ID is initialized at power on, using the following algorithm:

a) If a valid setup table exists, with the value read from it. This value can be an axis number 1 to 255 or can indicate that axis ID will be set by hardware.

b) If the setup table is invalid, with the last value set with a valid setup table. This value can be an axis number 1 to 255 or can indicate that axis ID will be set by hardware

c) If there is no axis ID set by a valid setup table, it will be set by hardware

Remark: *If a drive axis ID was previously set by software but its value is unknown, you can find it by selecting in the Communication | Setup dialogue at "Axis ID of drive/motor connected to PC" the option* ***Autodetected****. Apply this solution only if this drive is connected directly with your PC via an RS-232 link. If this drive is part of a CANbus network and the PC is serially connected with another drive, use the menu command* ***Communication | Scan CAN Network****.*



1.4. Setting the CANbus rate

The Technosoft intelligent drives/motors can work with the following rates on the CAN: 125kHz, 250kHz, 500kHz, 800kHz[1], 1MHz. In the Drive Setup dialogue you can choose the initial CAN rate after power on. This information is stored in the setup table. The CAN rate is initialized using the following algorithm:

a) If a valid setup table exists, with the CAN rate value read from it. This can be any of the supported rates or can indicate to use the firmware default (F/W default) value, which is 500kHz

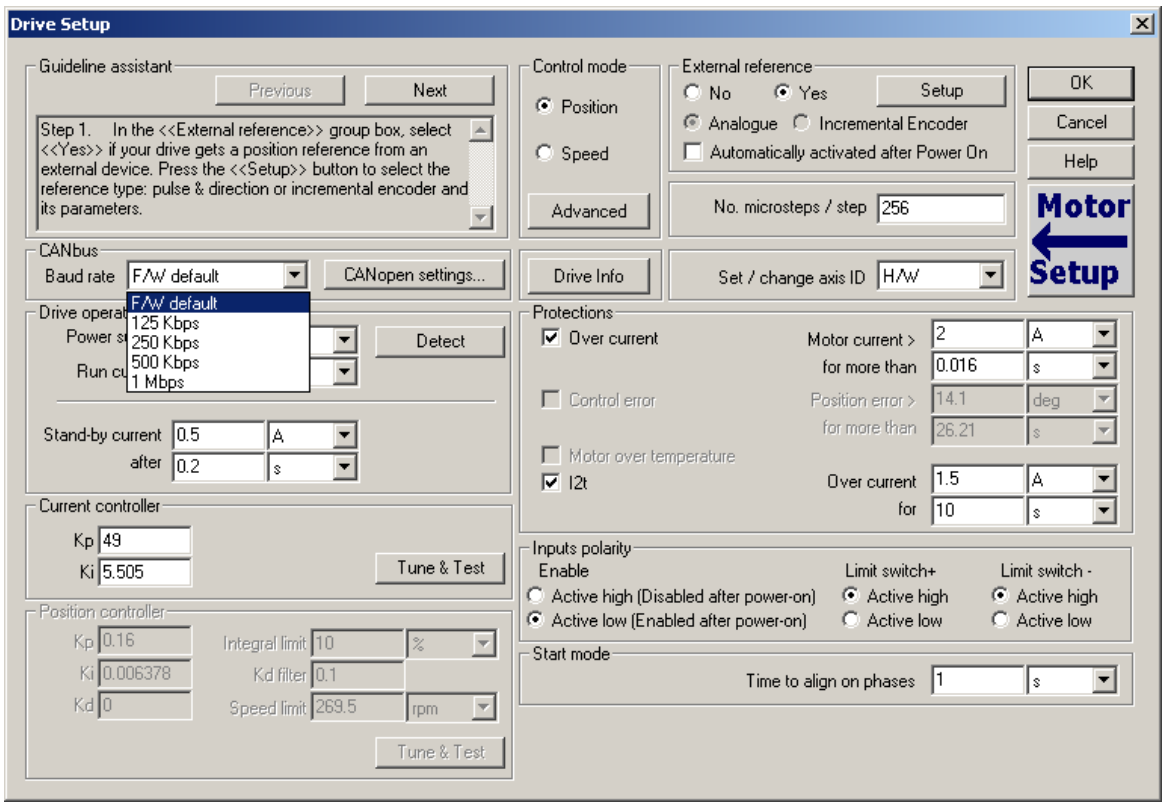
b) If the setup table is invalid, with the last CAN rate value set with a valid setup table. This can be any of the supported rates or can indicate to use the firmware default (F/W default) value

c) If there is no CAN rate value set by a valid setup table, with the firmware default value i.e. 500kHz

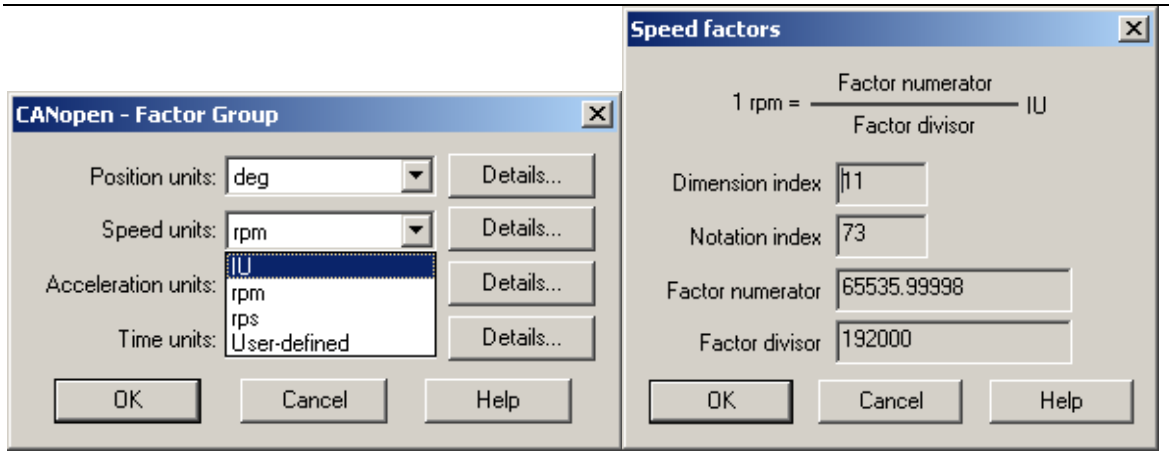
[1] 800 kHz not available for MotionChip III family

1.5. CANopen factor group setting[2]

By pressing the CANopen Settings button, you can choose the initial values after power on for the CANopen factor group settings. The factor group settings describe the scaling factors for position, speed, acceleration and time objects. In the factor group dialogue you can select the units to use when writing to these objects or reading them. You can either choose one of the standard units defined in the CANopen standard DS-402 or define your own unit.



[2] Only available for MotionChip III family



In the last case, it is your responsibility to set the factor numerator and divisor as well as its dimension and notation index. The factor group settings are stored in the setup table. By default the drive uses its internal units. The correspondence between the drive internal units and the SI units is presented in the drives' user manual.

1.6. Creating an Image File with the Setup Data

Once you have validated your setup, you can create with the menu command **Setup | Create EEPROM Programmer File** a software file (with extension **.sw**) which contains all the setup data to write in the EEPROM of your drive.

A software file is a text file that can be read with any text editor. It contains blocks of data separated by an empty line. Each block of data starts with the block start address, followed by data values to place in ascending order at consecutive addresses: first data – to write at start address, second data – to write at start address + 1, etc. All the data are hexadecimal 16- bit values (maximum 4 hexadecimal digits). Each line contains a single data value. When less then 4 hexadecimal digits are shown, the value must be right justified. For example 92 represent 0x0092.

The **.sw** file can be programmed into a drive:

- from a CANopen master, using the communication objects for writing data into the drive EEPROM (see **Chapter 15** for detailed example)
- from a host PC or PLC, using the TML_LIB functions for writing data into the drive EEPROM
- using the EEPROM Programmer tool, which comes with EasySetUp but may also be installed separately. The EEPROM Programmer was specifically designed for repetitive fast and easy programming of **.sw** files into the Technosoft drives during production

1.7. Motion Programming using a CANopen Master

The Technosoft intelligent drives/motors support the CiA draft standard **DS-301 v4.02** CANopen Application Layer and Communication Profile. It also conforms to the CiA draft standard proposal **DSP-402 v2.0** CANopen Device Profile for Drives and Motion Control. Detailed descriptions of the

objects supported by the drives are presented in the following chapters. **DSP-401 v2.0** CANopen Device Profile for I/O Modules is also available as an option upon request.

1.7.1. TechnoCAN Extension

In order to take full advantage of the powerful Technosoft Motion Language (TML) built into the intelligent drives/motors, Technosoft has developed an extension to CANopen, called TechnoCAN through which TML commands can be exchanged with the drives. Thanks to TechnoCAN you can inspect or reprogram any of the Technosoft drives from a CANopen network using EasySetUp or EasyMotion Studio and an RS-232 link between your PC and any of the drives.

TechnoCAN uses only message identifiers outside of the range used by the CANopen predefined connection set (as defined by CiA DS301 v4.02). Thus, TechnoCAN protocol and CANopen protocol can co-exist and communicate simultaneously on the same physical CAN bus, without disturbing each other.

1.7.2. Checking Setup Data Consistency

During the configuration phase, a CANopen master can quickly verify using the checksum objects and a reference **.sw** file (see 1.6 and 15 for details) whether the non-volatile EEPROM memory of an intelligent drive/motor contains the right information. If the checksum reported by the drive doesn't match with that computed from the **.sw** file, the CANopen master can download the entire **.sw** file into the drive EEPROM using the communication objects for writing data into the drive EEPROM.

1.8. Using the built-in Motion Controller and TML

One of the key advantages of the Technosoft drives is their capability to execute complex motions without requiring an external motion controller. This is possible because Technosoft drives offer in a single compact package both a state of art digital drive and a powerful motion controller.

1.8.1. Technosoft Motion Language Overview

Programming motion directly on a Technosoft drive requires to create and download a TML (Technosoft Motion Language) program into the drive memory. The TML allows you to:

- Set various motion modes (profiles, PVT, PT, electronic gearing or camming[3], etc.)
- Change the motion modes and/or the motion parameters
- Execute homing sequences
- Control the program flow through:
  - Conditional jumps and calls of TML functions

[3] Electronic gearing and camming are not available by default on the CANopen versions of MotionChip II family. However, they are available on request as special firmware versions.

    - Interrupts generated on pre-defined or programmable conditions (protections triggered, transitions of limit switch or capture inputs, etc.)
    - Waits for programmed events to occur
- Handle digital I/O and analogue input signals
- Execute arithmetic and logic operations
- Perform data transfers between axes
- Control motion of an axis from another one via motion commands sent between axes
- Send commands to a group of axes (multicast). This includes the possibility to start simultaneously motion sequences on all the axes from the group
- Synchronize all the axes from a network

In order to program a motion using TML you need EasyMotion Studio software platform.

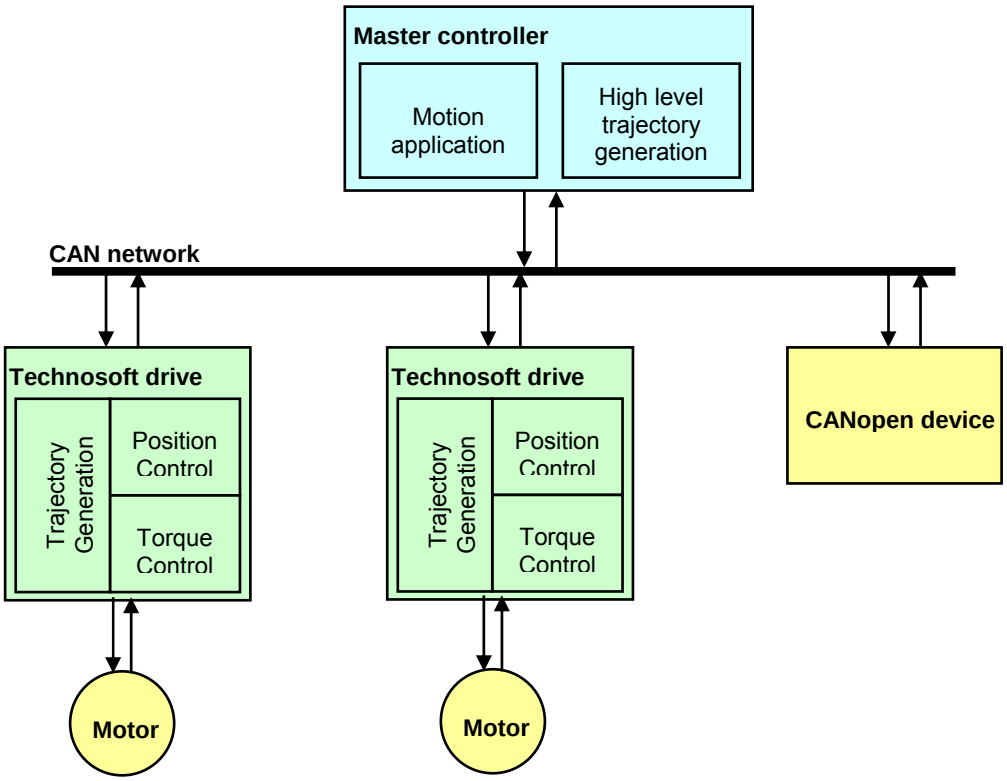
Chapter 16 describes in detail how the TML features can be combined with the CANopen programming.

2. CAN and the CANopen protocol

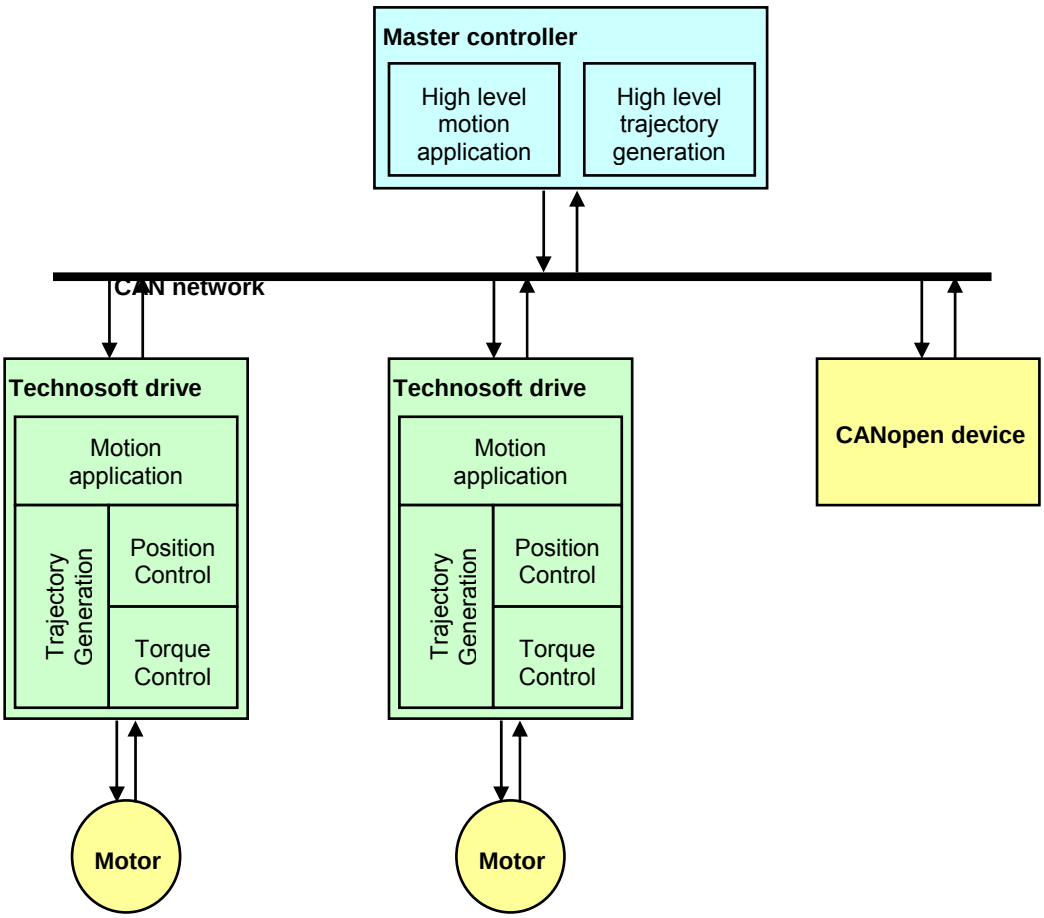
CAN (Controller Area Network) is a serial bus system used in a broad range of automation control systems. The CAN specifies the data link and the physical connection over which lays the CANopen, a high level protocol specifying how various types of devices can use the CAN network.

2.1. CAN Architecture

CAN provides distributed control of the motion application, the control loops are closed locally not on the master controller. The master controller coordinates multiple devices through the commands it sends and receives information about the status of the devices.



Technosoft extended the concept of distributed motion application allowing splitting the motion application between the Technosoft drives and the CANopen master. Using TML the user can build complex motion applications locally, on each drive, leaving on the CANopen master only a high level motion application and thus reducing the CAN master complexity. The master has the vision of the motion application, specific tasks being executed on the Technosoft drives.



2.2. Accessing CANopen devices

A CANopen device is controlled through read/write operations to/from objects performed by a CANopen master (PC or PLC).

2.2.1. Object dictionary

The Object Dictionary is a group of objects that describe the complete functionality of a device by way of communication objects and is the link between the communication interface and the application. All communication objects of a device (application data and configuration parameters) are described in the Object Dictionary in a standardized way.

2.2.2. Object access using index and sub-index

The objects defined for a device are accessed using a 16-bit index and an 8-bit sub-index. In case of arrays and records there is an additional sub-index for each element of the array or record.

2.2.3. Service Data Objects (SDO)

Service Data Objects are used by CANopen master to access any object from the drive's Object Dictionary. Both expedited and segmented SDO transfers are supported (see DS301 v4.02 for details). The SDOs are typically used for drive configuration after power-on, for PDO mapping and for infrequent low priority communication.

SDO transfers are confirmed services. In case of an error, an Abort SDO message is transmitted with one of the codes listed in **Table 2.1**.

Table 2.1 *SDO Abort Codes*

Abort code	Description
0503 0000h	Toggle bit not alternated
0504 0001h	Client/server command specifier not valid or unknown
0601 0000h	Unsupported access to an object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility error in the device.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0800 0000h	General error
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.

2.2.4. Process Data Objects (PDO)

Process Data Objects are used for high priority, real-time data transfers between CANopen master and the drives. The PDOs are unconfirmed services and are performed with no protocol overhead. Transmit PDOs are used to send data from the drive, and receive PDOs are used to receive data. The Technosoft drives accept 4 transmit PDOs and 4 receive PDOs. The contents of the PDOs can be set according with the application needs through the dynamic PDO-mapping. This operation can be done during the drive configuration phase using SDOs.

A PDO is defined by two objects: the communication object and the mapping object. The communication object defines the COB-ID of the PDO, the transmission type and the event triggering the transmission. The mapping object contains the descriptions of the objects mapped into the PDO, i.e. the index, sub-index and size of the mapped objects.

2.3. Objects that define SDOs and PDOs

2.3.1. Object 1200h: Server SDO Parameter

The object contains the COB-IDs of the messages used for the SDO protocol. The COBID of the SDO packages received by the drive, stored in sub-index 01, is computed as 600h + drive Node ID. The COB ID of the SDO packages sent by the drive, stored in sub-index 02, is computed as 580h + drive Node ID.

Object description:

Index	1200_h
Name	Server SDO Parameter
Object code	RECORD
Data type	SDO Parameter

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	2
Default value	2

Sub-index	01_h
Description	SDO receive COB-ID
Access	RO
PDO mapping	No
Value range	UNSIGNED32
Default value	600_h + Node-ID

Sub-index	02_h
Description	SDO transmit COB-ID
Access	RO
PDO mapping	No
Value range	UNSIGNED32
Default value	580_h + Node-ID

2.3.2. Object 1400h: Receive PDO1 Communication Parameters

The object contains the communication parameters of the receive PDO1. Sub-index 1_h contains the COB ID of the PDO. The transmission type (sub-index 2_h) defines the reception character of the PDO.

Object description:

Index	1400_h
Name	RPDO1 Communication Parameter
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	2

Sub-index	01_h
Description	COB-ID RPDO1
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	200_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No

Value range	UNSIGNED8
Default value	255

***Table 2.2** PDO COB-ID entry description*

Bit	Value	Meaning
31	0	PDO exists / is valid
	1	PDO does not exist / is not valid
30	0	RTR allowed on this PDO
	1	No RTR allowed on this PDO
29	0	11 bit ID
	1	29 bit ID
28…11	0	If bit 29=0
	X	If bit 29=1: Bit 11...28 of 29-bit PDO COB-ID
10...0	X	Bit 0...10 of PDO COB-ID

It is not allowed to change bits 0-29 while the PDO exists (bit 31=0).

2.3.3. Object 1401h: Receive PDO2 Communication parameters

The object contains the communication parameters of the receive PDO2. Sub-index 1_h contains the COB ID of the PDO. The transmission type (sub-index 2_h) defines the reception character of the PDO. The receive PDO2 COB-ID entry description is identical with the one of the receive PDO1 (see **Table 2.2**).

Object description:

Index	1401_h
Name	RPDO2 Communication Parameter
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	2

Sub-index	01_h
Description	COB-ID RPDO2
Access	RW

PDO mapping	No
Value range	UNSIGNED32
Default value	300_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

2.3.4. Object 1402h: Receive PDO3 Communication parameters

The object contains the communication parameters of the receive PDO3. Sub-index 1_h contains the COB ID of the PDO. The transmission type (sub-index 2_h) defines the reception character of the PDO. The receive PDO3 COB-ID entry description is identical with the one of the receive PDO1 (see **Table 2.2**).

Object description:

Index	1402_h
Name	RPDO3 Communication Parameter
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	2

Sub-index	01_h
Description	COB-ID RPDO3
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	400_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

2.3.5. Object 1403h: Receive PDO4 Communication parameters

The object contains the communication parameters of the receive PDO4. Sub-index 1_h contains the COB ID of the PDO. The transmission type (sub-index 2_h) defines the reception character of the PDO. The receive PDO4 COB-ID entry description is identical with the one of the receive PDO1 (see **Table 2.2**).

Object description:

Index	1403_h
Name	RPDO4 Communication Parameter
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	2

Sub-index	01_h
Description	COB-ID RPDO2
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	500_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No

Value range	UNSIGNED8
Default value	255

2.3.6. Object 1600h: Receive PDO1 Mapping Parameters

This object contains the mapping parameters of the receive PDO1. The sub-index 00_h contains the number of valid entries within the mapping record. This number of entries is also the number of the objects that shall be transmitted/received with the corresponding PDO. The sub-indices from 01_h to the number of entries contain the information about the mapped objects. These entries describe the PDO contents by their index, sub-index and length. The length entry contains the length of the mapped object in bits and is used to verify the overall mapping length.

The structure of the entries from sub-index 01_h to the number of entries is as follows:

MSB		**LSB**
Index (16 bits)	Sub-index (8 bits)	Object length (8 bits)

In order to change the PDO mapping, first the PDO has to be disabled - the object $160x_h$ sub-index 00_h has to be set to 0. Now the objects can be remapped. If a wrong mapping parameter is introduced (object does not exist, the object can not be mapped or wrong mapping length is detected) the SDO transfer will be aborted with an appropriate error code (0602 0000_h or 0604 0041_h). After all objects are mapped, sub-index 00_h has to be set to the valid number of mapped objects thus enabling the PDO.

If data types (index 01_h - 07_h) are mapped, they serve as “dummy entries”. The corresponding data is not evaluated by the drive. This feature can be used to transmit data to several drives using only one PDO, each drive using only a part of the PDO. This feature is only valid for receive PDOs.

Object description:

Index	1600_h
Name	RPDO1 Mapping Parameters
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	1

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No

Value range	UNSIGNED32
Default value	60400010_h – control word

2.3.7. Object 1601h: Receive PDO2 Mapping Parameters

This object contains the mapping parameters of the receive PDO2. The sub-index 00_h contains the number of valid entries within the mapping record. This number of entries is also the number of the objects that shall be transmitted/received with the corresponding PDO.

Object description:

Index	1601_h
Name	RPDO2 Mapping Parameter
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	2

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60400010_h – control word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60600008_h – modes of operation

2.3.8. Object 1602h: Receive PDO3 Mapping Parameters

This object contains the mapping parameters of the receive PDO3. The sub-index 00_h contains the number of valid entries within the mapping record. This number of entries is also the number of the objects that shall be transmitted/received with the corresponding PDO.

Object description:

Index	1602_h
Name	RPDO3 Mapping Parameter
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	2

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60400010_h – control word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	$607A0020_h$ – target position

2.3.9. Object 1603h: Receive PDO4 Mapping Parameters

This object contains the mapping parameters of the receive PDO4. The sub-index 00_h contains the number of valid entries within the mapping record. This number of entries is also the number of the objects that shall be transmitted/received with the corresponding PDO.

Object description:

Index	1603_h
Name	RPDO4 Mapping Parameters
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	2

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60400010_h – control word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	$60FF0020_h$ – target velocity

2.3.10. Object 1800h: Transmit PDO1 Communication parameters

This object contains the communication parameters of the transmit PDO1. For detailed description see object 1400_h (Receive PDO1 communication parameters)

Object description:

Index	1800_h
Name	TPDO1 Communication Parameters
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	5

Sub-index	01_h
Description	COB-ID TPDO1
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	180_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

Sub-index	03_h
Description	Reserved

Sub-index	04_h
Description	Reserved

Sub-index	05_h
Description	Event timer
Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

2.3.11. Object 1801h: Transmit PDO2 Communication parameters

This object contains the communication parameters of the transmit PDO2. For detailed description see object 1400_h (Receive PDO1 communication parameters)

Object description:

Index	1801_h
Name	TPDO2 Communication Parameters
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	5

Sub-index	01_h
Description	COB-ID TPDO2
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	280_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

Sub-index	03_h
Description	Reserved

Sub-index	04_h
Description	Reserved

Sub-index	05_h
Description	Event timer
Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

2.3.12. Object 1802h: Transmit PDO3 Communication parameters

This object contains the communication parameters of the transmit PDO3. For detailed description see object 1400_h (Receive PDO1 communication parameters)

Object description:

Index	1802_h
Name	TPDO3 Communication Parameters
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	5

Sub-index	01_h
Description	COB-ID TPDO3
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	380_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

Sub-index	03_h
Description	Reserved

Sub-index	04_h
Description	Reserved

Sub-index	05_h
Description	Event timer
Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

2.3.13. Object 1803h: Transmit PDO4 Communication parameters

This object contains the communication parameters of the transmit PDO4.

Object description:

Index	1803_h
Name	TPDO4 Communication Parameter
Object code	RECORD
Data type	PDO CommPar

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	-
Default value	5

Sub-index	01_h
Description	COB-ID TPDO4
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	480_h + Node-ID

Sub-index	02_h
Description	Transmission type
Access	RW
PDO mapping	No
Value range	UNSIGNED8
Default value	255

Sub-index	03_h
Description	Reserved

Sub-index	04_h
Description	Reserved

Sub-index	05_h
Description	Event timer
Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

2.3.14. Object 1A00h: Transmit PDO1 Mapping Parameters

This object contains the mapping parameters of the transmit PDO1. For detailed description see object 1600_h (Receive PDO1 mapping parameters)

Object description:

Index	$1A00_h$
Name	TPDO1 Mapping Parameters
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	1

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60410010_h – status word

2.3.15. Object 1A01h: Transmit PDO2 Mapping Parameters

This object contains the mapping parameters of the transmit PDO2. For detailed description see object 1600_h (Receive PDO1 mapping parameters)

Object description:

Index	$1A01_h$
Name	TPDO2 Mapping Parameter
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of mapped objects
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	2

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60410010_h – status word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32

Default value	60610008_h – modes of operation display

2.3.16. Object 1A02h: Transmit PDO3 Mapping Parameters

This object contains the mapping parameters of the transmit PDO3. For detailed description see object 1600_h (Receive PDO1 mapping parameters). By default, this PDO is disabled.

Object description:

Index	$1A02_h$
Name	TPDO3 Mapping Parameter
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	0

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60410010_h – status word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60640020_h – position actual value

2.3.17. Object 1A03h: Transmit PDO4 Mapping Parameters

This object contains the mapping parameters of the transmit PDO4. For detailed description see object 1600_h (Receive PDO1 mapping parameters). By default, this PDO is disabled.

Object description:

Index	$1A03_h$
Name	TPDO4 Mapping Parameter
Object code	RECORD
Data type	PDO Mapping

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RW
PDO mapping	No
Value range	0: deactivated 1 – 64: activated
Default value	0

Sub-index	01_h
Description	1st mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	60410010_h – status word

Sub-index	02_h
Description	2nd mapped object
Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	$606C0020_h$ – velocity actual value

2.4. Dynamic mapping of the PDOs

Follow the next steps to change the default mapping of a PDO:

1. **Disable the PDO**. In the PDO's mapping object (index 1600_h-1603_h for RxPDOs and $1A00_h$-$1A03_h$ for TxPDOs) set the first sub-index (the number of mapped objects) to 0. This will disable the PDO.
2. **Change the communication parameters**. If necessary change the communication parameters of the PDO (index 1400_h-1403_h for RxPDOs and 1800_h-1803_h for TxPDOs): its COB-ID, the transmission type or the event that triggers the transmission.

3. **Map the new objects**. Write in the PDO's mapping object (index 1600_h-1603_h for RxPDOs and $1A00_h$-$1A03_h$ for TxPDOs) sub-indexes (1-8) the description of the objects that will be mapped. You can map up to 8 objects having 1 byte size.
4. **Enable the PDO**. In sub-index 0 of the PDO's associated mapping object (index 1600_h-1603_h for RxPDOs and $1A00_h$-$1A03_h$ for TxPDOs) write the number of mapped objects.

Example: Map the receive PDO3 of axis number 10 with **ControlWord** (index 6040_h) and **Modes of Operation** (index 6060_h).

1. **Disable the PDO**. Write zero in object 1602_h sub-index 0, this will disable the PDO.

Send the following message (SDO access to object 1602_h sub-index 0, 8-bit value 0):

COB-ID	**60A**
Data	**2F 02 16 00 00 00 00 00**

2. **Change the communication parameters.** For example purposes the communication parameters default values are acceptable.
3. **Map the new objects**.
   a. Write in object 1602_h sub-index 1 the description of the Control Word:

Index	Sub-index	Length	
6040_h	**00_h**	**10_h**	**60400010_h**

Send the following message (SDO access to object 1602_h sub-index 1, 32-bit value 60400010_h):

COB-ID	**60A**
Data	**23 02 16 01 10 00 40 60**

   b. Write in object 1602_h sub-index 2 the description of the Mode of Operation:

Index	Sub-index	Length	
6060_h	**00_h**	**08_h**	**60600008_h**

Send the following message (SDO access to object 1602_h sub-index 2, 32-bit value 60600008_h):

COB-ID	**60A**
Data	**23 02 16 02 08 00 60 60**

4. **Enable the PDO**. Set the object 1602_h sub-index 0 with the value 2.

Send the following message (SDO access to object 1602_h sub-index 0, 8-bit value 2):

COB-ID	**60A**
Data	**2F 02 16 00 02 00 00 00**

This page is empty

3. Network Management

3.1. Overview

The Network Management (NMT) services initialize, start, monitor, reset or stop the CANopen nodes. The NMT requires a node in the network (a PC or a PLC) to be designed as a network manager while the Technosoft intelligent drives/motors are the NMT slaves. The NMT services are fulfilled by the NMT objects described later in this chapter.

3.1.1. Device control

Through Module Control Services, the NMT master controls the state of the NMT slaves. The following states are implemented on the Technosoft drives:

State	Description
Pre-operational	The drive enters the pre-operational state after finishing its initialization. In this state the communication between the CANopen master and the drive can be done only via SDOs. PDOs are not allowed.
Operational	This is the normal operating state of the drives. The communication through SDO and PDO is allowed
Stopped	In this state the drive stops the communication except the network management messages.

The network manager can change the state of the drives using one of the following services:

Service	Description
Start Remote Node	The NMT master sets the state of the selected NMT slave to operational
Stop Remote Node	The NMT master sets the state of the selected NMT slave to stopped
Enter Pre-Operational	The NMT master sets the state of the selected NMT slave to pre-operational
Reset Node	The NMT master sets the state of the selected NMT slave to the “reset application” sub-state. In this state the drives perform a software reset and enter the pre-operational state.
Reset Communication	The NMT master sets the state of the selected NMT slave to the “reset communication” sub-state. In this state the drives resets their communication and enter the pre-operational state.

All the services are unconfirmed.

3.1.2. Device monitoring

In addition to controlling the drive states the NMT provides services for monitoring the nodes in the network. The monitoring services are achieved mainly through the periodical transmission of

messages by the network manager, with answers from the slaves, or messages sent by the slaves without master intervention. Monitoring services can use the Node Guarding protocol (including Life Guarding) or the Heartbeat protocol.

Node guarding protocol

The master polls each NMT slave at regular time intervals. This time interval is called the guard time and may be different for each NMT slave. The slaves answer with a node-guarding message containing their state. This allows both the master and the slave to identify a network error if either the remote request or the guarding messages stop.

The node life time is computed as the product between the guard time (index $100C_h$) and the life time factor (index $100D_h$). If the drive is not accessed within the life time then a Life Time event occurs and an emergency telegram is sent.

Heartbeat protocol

The Heartbeat protocol defines an error control service without the need of remote frames. It implies independent and cyclical transmission of a telegram by the drive (the Heartbeat producer) indicating the drives current state. The time interval between two heartbeat messages is specified through producer heartbeat time (index 1017_h). The master (Heartbeat consumer) guards the reception of the heartbeat messages within the Heartbeat Consumer Time. If the value of this object is 0, the heartbeat transmission is disabled. If the master doesn't receive the heartbeat message this indicates a problem with the drive or with its network connection.

Bootup protocol

This protocol is used by the drive to signal to the network master that it has entered the state pre-operational.

3.1.3. Synchronisation between devices

The synchronization message (SYNC) allows synchronizing the devices in the network and triggering the synchronous transmission of PDOs. The SYNC producer broadcasts the synchronization object periodically. This service is unconfirmed. Technosoft intelligent drives/motors can act both as SYNC consumer and producer.

For time critical applications, which require more accurate synchronization, the Technosoft drives can use the optional high-resolution synchronization protocol, which employs a special form of time stamp message. The High Resolution Time Stamp object (index 1013_h) contains a time stamp with a resolution of 1µs. The object can be mapped into a PDO in order to define a high-resolution time stamp message. The PDO should be configured for synchronous transmission. When one of the Technosoft drives is set as synchronization master, the High resolution time stamp is by default sent using the COB ID defined in COB-ID High Resolution Time Stamp object (index 2004_h).

3.1.4. Emergency messages

A drive sends an emergency message (EMCY) when a drive internal error occurs. An emergency message is transmitted only once per 'error event'. As long as no new errors occur, the drive will not transmit further emergency messages.

The emergency error codes supported by the Technosoft drives are listed in **Table 3.1**. Details regarding the conditions that may generate emergency messages are presented at object Motion Error Register index 2000_h.

Table 3.1 *Emergency Error Codes*

Error code (hex)	Description
0000	Error Reset or No Error
1000	Generic Error
2310	Continuous over-current
2340	Short-circuit
3210	DC-link over-voltage
3220	DC-link under-voltage
4280	Over temperature motor
4310	Over temperature drive
5441	Drive disabled due to enable input
5442	Negative limit switch active
5443	Positive limit switch active
6100	Invalid setup data
7500	Communication error
8110	CAN overrun (message lost)
8130	Life guard error or heartbeat error
8331	I2t protection triggered
8580	Position wraparound
8611	Control error / Following error
9000	Command error
FF01	Generic interpolated position mode error (PVT / PT error)
FF02	Change set acknowledge bit wrong value
FF03	Specified homing method not available
FF04	A wrong mode is set in object 6060h, modes_of_operation
FF05[4]	Specified digital I/O line not available

The Emergency message contains of 8 data bytes having the following contents:

[4] Only valid for MotionChip III family

0-1	2	3-7
Emergency Error Code	Error Register (Object 1001h)	Manufacturer specific error field

3.2. Network management objects

The section describes the objects related to network management

3.2.1. Object 1001h: Error Register

This object is an error register for the device. The device can map internal errors in this byte. This entry is mandatory for all devices. It is a part of an Emergency object.

Object description:

Index	1001_h
Name	Error register
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RO
PDO mapping	No
Value range	UNSIGNED8
Default value	No

Table 3.2 *Bit description of the object*

Bit	Description
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication error
5	Device profile specific
6	Reserved (always 0)
7	Manufacturer specific.

Valid bits while an error occurs – bit 0 and bit 4. The other bits will remain 0.

3.2.2. Object 1005h: COB-ID of the SYNC Message

This object defines the COB-ID of the Synchronization Object (SYNC) and whether the drive generates the SYNC or not.

Object description:

Index	1005_h
Name	COB-ID SYNC Message
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	80_h

The structure of the parameter is the following:

Table 3.3 *Bit description of the object*

Bit	Value	Description
31	X	Reserved
30	0	Drive does not generate synchronization messages
	1	Drive is the synchronization master (SYNC producer)
29	0	Use 11 bit identifier
	1	Use 29 bit identifier
28…11	X	Bit 11...28 of 29-bit SYNC COB-ID
10...0	X	Bit 0...10 of SYNC COB-ID

The first transmission of SYNC object starts within 1 sync cycle after setting bit 30 to 1. It is not allowed to change bit 0...29, while the object exists (bit 30 = 1).

3.2.3. Object 1006h: Communication Cycle Period

The object defines the time interval between SYNC messages expressed in µs. A drive sends SYNC messages if it is configured to send SYNC messages through object 1005_h and the object 1006_h is set with a non-zero value.

Object description:

Index	1006_h
Name	Communication cycle period
Object code	VAR

Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	0

3.2.4. Object 100Ch: Guard Time

The Guard Time object multiplied with Lifetime Factor (index $100D_h$) gives the Lifetime of the drive for the Life Guarding Protocol. The Guard Time is expressed in ms. When the Life Guarding Protocol is not used the object must be set to 0. When the Node Guarding is active, i.e. the network manager sends the Node Guarding messages, the Life Guarding Protocol checks if the master has stopped sending messages or not. The decision of Node Guarding failure is taken if no message from the master is received within the period defined as Lifetime.

Object description:

Index	$100C_h$
Name	Guard time
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

3.2.5. Object 100Dh: Life Time Factor

The lifetime factor multiplied with the guard time gives the lifetime for the Life Guarding Protocol. Must be 0 if not used.

Object description:

Index	$100D_h$
Name	Life time factor
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED8

Default value	0

3.2.6. Object 1013h: High Resolution Time Stamp

This object contains a time stamp with a resolution of 1µs. It can be used in order to synchronize the drives in the CANopen network.

When setting up the synchronization mechanism, the master has to map the object 1013_h on a receive PDO whose COB-ID should be identical on all the slave drives that need to be synchronized.

This object has to be written immediately after the SYNC object. Upon the reception of this object, the drive will adjust its internal clock so that to compensate for the difference between the received value and its internal value.

Object description:

Index	1013_h
Name	High resolution time stamp
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

3.2.7. Object 2004h: COB-ID of the High-resolution time stamp

This object defines the COB-ID used by the high-resolution time stamp message sent by the synchronization master (SYNC producer) in order to achieve synchronization on the network.

Object description:

Index	2004_h
Name	COB-ID High resolution time stamp
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	100_h

The structure of the parameter is the following:

Bit	Value	Meaning
31	0	High resolution time stamp exists / is valid
	1	High resolution time stamp does not exist / is not valid
30	0	Reserved (always 0)
29	0	11 bit ID
	1	29 bit ID
28…11	X	Bit 11...28 of 29-bit High resolution time stamp COB-ID
10...0	X	Bit 0...10 of High resolution time stamp COB-ID

It is not allowed to change bits 0-29 while the object exists (bit 31=0).

This object will be used when a Technosoft drive is required to be the master for the synchronization messages. In this case, the CANopen master does not need to map the 1013_h into a receive PDO. The procedure to activate the synchronization is the following:

1. **Set the SYNC interval**. Write the desired SYNC interval into the object 1006_h (Communication Cycle Period). For example – 20 ms.

Send the following message (SDO access to object 1006_h sub-index 0, 32-bit value 0x4E20 = 20000 µs = 20 ms):

COB-ID	**60A**
Data	**23 06 10 00 20 4E 00 00**

2. **Activate the SYNC producer**. Set bit 30 in object 1005_h (COB-ID of SYNC Message).

Send the following message (SDO access to object 1005_h sub-index 0, 32-bit value 40000080_h):

COB-ID	**60A**
Data	**23 05 10 00 80 00 00 40**

3.2.8. Object 1014h: COB-ID Emergency Object

Index 1014h defines the COB-ID of the Emergency Object (EMCY).

Object description:

Index	1014_h
Name	COB-ID Emergency message
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED32
Default value	80h + Node-ID

***Table 3.4** Structure of the EMCY Identifier*

MSB				LSB
31	30	29	28 - 11	10 - 0
0/1	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
0/1	0	1	29 –bit Identifier	

***Table 3.5** Description of EMCY COB-ID entry*

Bit	Value	Description
31 (MSB)	0	EMCY exists / is valid
	1	EMCY does not exist / is not valid
30	0	Drive does not generate synchronization messages
	1	Drive is the synchronization master (SYNC producer)
29	0	Use 11 bit identifier
	1	Use 29 bit identifier
28…11	0 X	If bit 29 = 0 Bit 11...28 of 29-bit SYNC COB-ID
10...0 (LSB)	X	Bit 0...10 of COB-ID

It is not allowed to change Bits 0-29, while the object exists (Bit 31=0).

3.2.9. Object 1017h: Producer Heartbeat Time

This object defines the cycle time of the heartbeat (if not equal to zero). If the heartbeat is not used, this object must have the default value 0. The time has to be a multiple of 1 ms.

Object description:

Index	1017_h
Name	Producer Heartbeat Time
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	0

4. Drive control and status

4.1. Overview

The state machine from **Device Profile Drives and Motion Control** describes the drive status and the possible control sequences of the drive. The drive states can be changed by the **Control Word** and/or according to internal events. The drive current state is reflected in the **Status Word**. The state machine presented in **Figure 4.1** describes the state machine of the drive with respect to the control of the power stage as a result of user commands and internal drive faults.

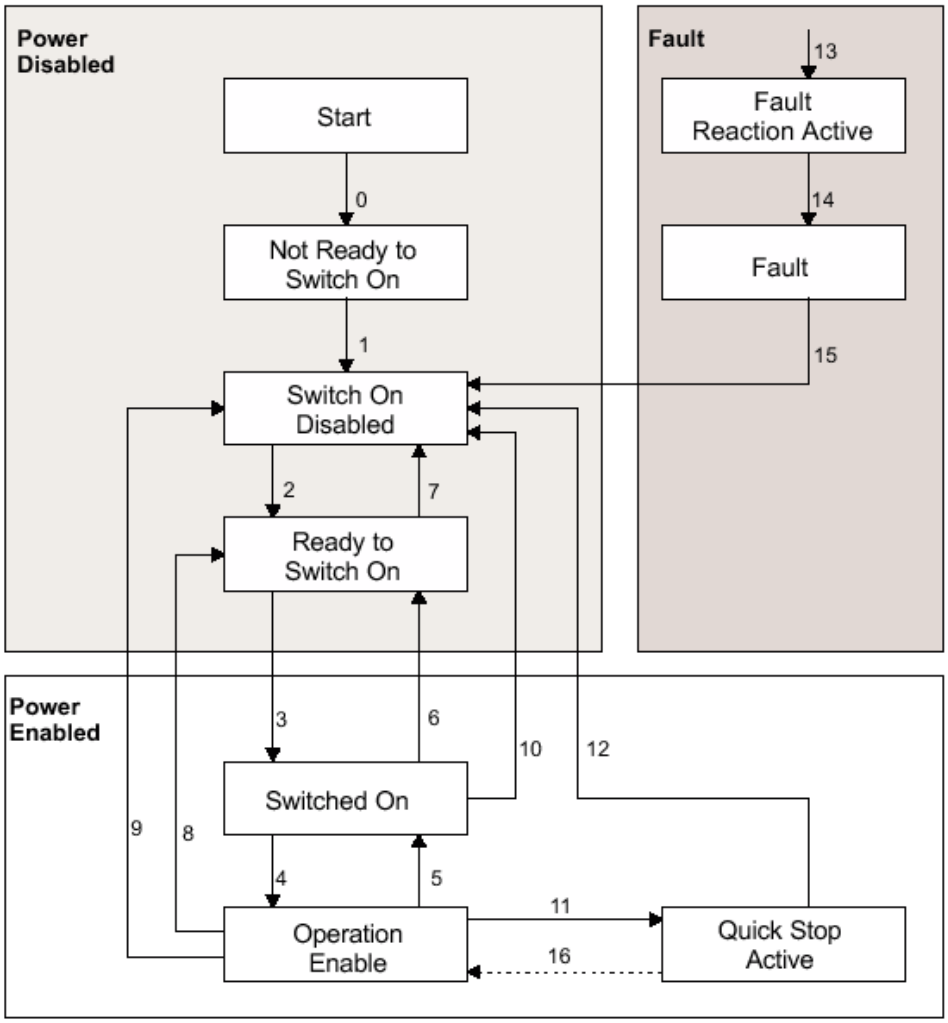


Figure 4.1 *Drive's status machine. States and transitions*

***Table 4.1** Drive States*

State	Description
Not Ready to Switch On	The drive performs basic initializations after power-on. The drive function is disabled The transition to this state is automatic.
Switch On Disabled	The drive basic initializations are done and the green led must turn-on if no error is detected. The drive is not ready to switch on; any drive parameters can be modified, including a complete update of the whole EEPROM data (setup table, TML program, cam files, etc.) The motor supply can be switched on, but the motion functions cannot be carried out yet. The transition to this state is automatic.
Ready to Switch On	The motor supply voltage may be switched on, most of the drive parameter settings can still be modified, motion functions cannot be carried out yet.
Switched On (Operation Disabled)	The motor supply voltage must be switched on. The power stage is switched on (enabled). If the operation mode set performs position control, the motor is held in position. If the operation mode set performs speed control, the motor is kept at zero speed. If the operation mode is torque external, the motor is kept with zero torque. The motion functions cannot be carried out yet.
Operation Enable	No fault present, power stage is switched on, motion functions are enabled. This corresponds to the normal operation of the drive.
Quick Stop Active	Drive has been stopped with the quick stop deceleration. The power stage is enabled. If the drive was operating in position control when quick stop command was issued, the motor is held in position. If the drive was operating in speed control, the motor is kept at zero speed. If the drive was operating in torque control, the motor is kept at zero torque.
Fault Reaction Active	The drive performs a default reaction to the occurrence of an error condition
Fault	The drive remains in fault condition, until it receives a Reset Fault command. If following this command, all the bits from the Motion Error Register are reset, the drive exits the fault state

***Table 4.2** Drive State Transitions*

Transition	**Event**	**Action**
0	Reset	Initialization
1	Initialization completed successfully. Drive is ready to operate.	None

2	Bit 1 *Disable Voltage* and Bit 2 *Quick Stop,* are set in Control Word (*Shutdown* command). Motor voltage may be present.	None
3	Bit 0 is also set in Control Word (*Switch On* command)	Power stage is switched on (enabled), provided that the enable input is on enable status. Drive has torque and depending on the mode of operation set, may held motor at the present position, keep the motor at zero speed or keep the motor at zero torque
4	Bit 3 is also set Control Word (*Enable Operation* command)	Motion function is enabled, depending on the mode that is set
5	Bit 3 is cancelled in Control Word (*Disable Operation* command)	Motion function is inhibited. Drive is stopped, using the acceleration rate set for position or speed profiles. Depending on the mode of operation set before the Disable Operation command, the motor may be held at the present position, kept at zero speed or zero torque
6	Bit 0 is cancelled in Control Word (*Shutdown* command)	Power stage is disabled. Drive has no torque. Motor is free to rotate
7	Bit 1 or 2 is cancelled in Control Word (*Quick Stop* or *Disable Voltage* command)	None
8	Bit 0 is cancelled in Control Word (*Shutdown* command)	Power stage is disabled. Drive has no torque. Motor is free to rotate
9	Bit 1 is cancelled in Control Word (*Disable Voltage* command)	Power stage is disabled. Drive has no torque. Motor is free to rotate
10	Bit 1 or 2 is cancelled in Control Word (*Quick Stop* or *Disable Voltage* command)	Power stage is disabled. Drive has no torque. Motor is free to rotate
11	Bit 2 is cancelled in Control Word (*Quick Stop* command)	Drive is stopped with the quick-stop deceleration rate. The power stage remains enabled. Depending on the mode of operation set before the Quick Stop command, the motor may be held at the present position, kept at zero speed or zero torque.
12	*Quick Stop* is completed or bit 1 is cancelled in Control Word (*Disable Voltage* command)	Output stage is disabled. Drive has no torque.
13	A fault has occurred	Execute specific fault treatment routine
14	The fault treatment is complete	The drive function is disabled

15	Bit 7 is set in Control Word (*Reset Fault* command)	Some of the bits from Motion Error Register are reset. If all the error conditions are reset, the drive returns to Switch On Disabled status. After leaving the state *Fault* the bit *Fault Reset* of the *control_word* has to be cleared by the host.
16	Bit 2 is set in Control Word (*Enable Operation* command). This transition is possible if *Quick-Stop-Option-Code* is 5, 6, 7 or 8	Drive exits from Quick Stop state. Drive function is enabled.

4.2. Drive control and status objects

4.2.1. Object 6040h: Control Word

The object controls the status of the drive. It is used to enable/disable the power stage of the drive, start/halt the motions and to clear the fault status. The status machine is controlled through the Control Word.

Object description:

Index	6040_h
Name	Control word
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Yes
Units	-
Value range	0 … 65535
Default value	No

The Control Word has the following bit assignment:

Table 4.3 *Bit Assignment in Control Word*

Bit	Value	Meaning
15[5]	0	Registration mode inactive
	1	Activate registration mode

[5] Only valid for MotionChip III family

14	0	When an update is performed, update the demand values for speed and position with the actual values of speed and position
	1	When an update is performed, keep unchanged the demand values for speed and position
13		When it is set it cancels the execution of the TML function called through object 2006h. The bit is automatically reset by the drive when the command is executed.
12	0	No action
	1	If bit 14 = 1 – Force *position demand value* to 0 If bit 14 = 0 – Force *position actual value* to 0 This bit is valid regardless of the status of the drive or other bits in control_word
11		Manufacturer Specific - Operation Mode Specific. The meaning of this bit is detailed further in this manual for each operation mode
10-9		Reserved. Writes have no effect. Read as 0
8	0	No action
	1	Halt command – the motor will slow down on slow down ramp
7	0	No action
	1	Reset Fault. The faults are reset on 0 to 1 transition of this bit. After a Reset Fault command, the master has to reset this bit.
4-6		Operation Mode Specific. The meaning of these bits is detailed further in this manual for each operation mode
3		Enable Operation
2		Quick Stop
1		Enable Voltage
0		Switch On

4.2.2. Object 6041h: Status Word

Object description:

Index	6041_h
Name	Status word
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Yes

Units	-
Value range	0 … 65535
Default value	No

The Status Word has the following bit assignment:

Table 4.4 *Bit Assignment in Status Word*

Bit	Value	Description
15	0	Axis off. Power stage is disabled. Motor control is not performed
	1	Axis on. Power stage is enabled. Motor control is performed
14	0	No event set or the programmed event has not occurred yet
	1	Last event set has occurred
13..12		Operation Mode Specific. The meaning of these bits is detailed further in this manual for each operation mode
11		Internal Limit Active
10		Target reached
9	0	Remote – drive is in local mode and will not execute the command message.
	1	Remote – drive parameters may be modified via CAN and the drive will execute the command message.
8	0	No TML function or homing is executed. The execution of the last called TML function or homing is completed.
	1	A TML function or homing is executed. Until the function or homing execution ends or is aborted, no other TML function / homing may be called
7	0	No Warning
	1	Warning. A TML function / homing was called, while another TML function / homing is still in execution. The last call is ignored.
6		Switch On Disabled.
5		Quick Stop. When this bit is zero, the drive is performing a quick stop
4	0	Motor supply voltage is present
	1	Motor supply voltage is absent
3		Fault. If set, a fault condition is or was present in the drive.
2		Operation Enabled
1		Switched On

0		Ready to Switch On

4.2.3. Object 1002h: Manufacturer Status Register

This object is a common status register for manufacturer specific purposes.

Object description:

Index	1002_h
Name	Manufacturer status register
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RO
PDO mapping	Optional
Value range	UNSIGNED32
Default value	No

Table 4.5 *Bit Assignment in Manufacturer Status Register*

Bit	Value	Description
31	1	Drive/motor in fault status
30	1	Reference position in absolute electronic camming mode reached – valid only for MotionChip III family
29	1	Reserved
28	1	Gear ratio in electronic gearing mode reached – valid only for MotionChip III family
27	1	Drive I2t protection warning level reached
26	1	Motor I2t protection warning level reached
25	1	Target command reached
24	1	Capture event/interrupt triggered
23	1	Limit switch negative event / interrupt triggered
22	1	Limit switch positive event / interrupt triggered
21	1	AUTORUN mode enabled
20	1	Position trigger 4 reached
19	1	Position trigger 3 reached
18	1	Position trigger 2 reached
17	1	Position trigger 1 reached

16	1	Drive/motor initialization performed
15…0		Same as Object 6041h, Status Word

4.2.4. Object 6060h: Modes of Operation

The object selects the mode of operation of the drive.

Object description:

Index	6060_h
Name	Modes of Operation
Object code	VAR
Data type	INTEGER8

Entry description:

Access	WO
PDO mapping	Yes
Units	-
Value range	-128 … 127
Default value	No

Data description:

Value	**Description**
-128…-6	Reserved
-5	Manufacturer specific – External Reference Torque Mode[6]
-4	Manufacturer specific – External Reference Speed Mode[7]
-3	Manufacturer specific – External Reference Position Mode[8]
-2	Manufacturer specific – Electronic Camming Position Mode[9]
-1	Manufacturer specific – Electronic Gearing Position Mode[10]
0	Reserved
1	Profile Position Mode
2	Reserved
3	Profile Velocity Mode
4,5	Reserved

[6] Only available for MotionChip III family. Available also on request for MotionChip II family

[7] Only available for MotionChip III family. Available also on request for MotionChip II family

[8] Only available for MotionChip III family. Available also on request for MotionChip II family

[9] Only available for MotionChip III family. Available also on request for MotionChip II family

[10] Only available for MotionChip III family. Available also on request for MotionChip II family

6	Homing Mode
7	Interpolated Position Mode
8…127	Reserved

Remark: *The actual mode is reflected in object 6061h (Modes of Operation Display).*

4.2.5. Object 6061h: Modes of Operation Display

The object reflects the actual mode of operation set with object Modes of Operation (index 6060_h)

Object description:

Index	6061_h
Name	Modes of Operation Display
Object code	VAR
Data type	INTEGER8

Entry description:

Access	RO
PDO mapping	Possible
Units	-
Value range	-128 … 127
Default value	No

Data description: Same as for object 6060h, Modes of Operation.

4.2.6. Object 2002h: Drive Status Mask

The Drive Status Mask offers the possibility to choose which changes of the 32 bits from the Manufacturer Status Register (1002_h) bits should be signaled via asynchronous (event triggered) TPDOs when the register is mapped inside any TPDOs.

For the 16 least significant bits (the one corresponding to the Status Word), both the Status Word and the Manufacturer Status Register will be sent, if both of them are mapped on TPDOs.

The manufacturer status register is continuously monitored for bit changes. When a bit changes, if the corresponding bit from the Drive Status Mask is set to 1, the drives send the TPDOs defined as asynchronous which include either the Status Word of Manufacturer Status Register. By default, the Status word is mapped to TPDO1,2,3 and 4. Only TPDO1 and 2 are enabled by default.

The Drive Status Mask has the same bit codification as the Manufacturer Status Register and the following meaning:

1 – unmask bit. If the same bit from Manufacturer Status Register is changed the drive sends the asynchronous (event triggered) TPDOs including the Status Word and / or Manufacturer Status Register

0 – mask bit. If the same bit from Manufacturer Status Register is changed, no asynchronous (event triggered) TPDOs are sent due to Manufacturer Status Register modification

Object description:

Index	2002_h
Name	Drive status mask
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Units	-
Value range	UNSIGNED32
Default value	$0000347F_h$

4.3. Error monitoring

4.3.1. Object 2000h: Motion Error Register

The Motion Error Register displays all the drive possible errors. A bit set to 1 signals that a specific error has occurred. When the error condition disappears or the error is reset using a Fault Reset command, the corresponding bit is reset to 0.

The Motion Error Register is continuously checked for changes of the bits status. When a bit is set (e.g. an error has occurred), if the corresponding bit from Motion Error Register Mask (2001_h) is set to 1, an emergency message with the specific error code is sent. When a bit is reset, if the corresponding bit from Motion Error Register Mask (2001_h) is set to 1, an emergency message for error reset is sent.

Object description:

Index	2000_h
Name	Motion Error Register
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Units	-
Value range	0 … 65535
Default value	0

Table 4.6 *Bit Assignment in Motion Error Register*

Bit	Description
15	Drive disabled due to enable input. Set when enable input is on disable state. Reset when enable input is on enable state

14	Command error. This bit is set in several situations. They can be distinguished either by the associated emergency code, or in conjunction with other bits: 0xFF03 - Specified homing method not available 0xFF04 - A wrong mode is set in object 6060h, modes_of_operation 0xFF05 - Specified digital I/O line not available A function is called during the execution of another function (+ set bit 7 of object 6041h, status word) Update of operation mode received during a transition This bit acts just as a warning.
13	Under-voltage. Set when protection is triggered. Reset by a Reset Fault command
12	Over-voltage. Set when protection is triggered. Reset by a Reset Fault command
11	Over temperature drive. Set when protection is triggered. Reset by a Reset Fault command.
10	Over temperature motor. Set when protection is triggered. Reset by a Reset Fault command. This protection may be activated if the motor has a PTC or NTC temperature contact.
9	I^2T protection. Set when protection is triggered. Reset by a Reset Fault command
8	Over current. Set when protection is triggered. Reset by a Reset Fault command
7	Negative limit switch active. Set when LSN input is in active state. Reset when LSN input is inactive state
6	Positive limit switch active. Set when LSP input is in active state. Reset when LSP input is inactive state
5	Motor position wraps around. Set when protection is triggered. Reset by a Reset Fault command
4	Communication error. Set when protection is triggered. Reset by a Reset Fault command
3	Control error (position/speed error too big). Set when protection is triggered. Reset by a Reset Fault command
2	Invalid setup data. Set when the EEPROM stored setup data is not valid or not present.
1	Short-circuit. Set when protection is triggered. Reset by a Reset Fault command
0	CAN error. Set when CAN controller is in error mode. Reset by a Reset Fault command

4.3.2. Object 2001h: Motion Error Register Mask

Object description:

Index	2001_h
Name	Motion Error Register Mask
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW

PDO mapping	Possible
Units	-
Value range	0 … 65535
Default value	$FFFF_h$

The Motion Error Register Mask offers the possibility to choose which of the errors set or reset in the Motion Error Register to be signaled via emergency messages. The Motion Error Register Mask has the same bit codification as the Motion Error Register (see **Table 4.6**) and the following meaning:

1 – Send an emergency message when the corresponding bit from the Motion Error Register is set

0 – Don't send an emergency message when the corresponding bit from the Motion Error Register is set

4.3.3. Object 605Eh: Fault reaction option code[11]

This parameter determines what action should be taken if a non-fatal error occurs in the drive. The non-fatal errors are by default the following:

Under-voltage

Over-voltage

Drive over-temperature

Motor over-temperature

Object description:

Index	$605E_h$
Name	Fault reaction option code
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RW
PDO mapping	No
Value range	-32768 … 32767
Default value	2

Data description:

Value	**Description**
-32768…-2	Manufacturer specific
-1	No action

[11] Only valid for MotionChip III family

0	Disable drive, motor is free to rotate
1	Reserved
2	Slow down with quick stop ramp
3…32767	Reserved

4.3.4. Object 6007h: Abort connection option code[12]

The object sets the action performed by the drive when the connection to the network is lost.

Object description:

Index	6007_h
Name	Abort connection option code
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RW
PDO mapping	Possible
Value range	-32768…32767
Default value	0

Table 4.7 *Abort connection option codes values:*

Option code	Description
0	No action
1	Malfunction – the drive will enter into Fault status
2…32767	Reserved
-32768…-1	Manufacturer specific (reserved)

4.4. Digital I/O control and status objects

4.4.1. Object 60FDh: Digital inputs[13]

The object contains the actual value of the digital inputs available on the drive. Each bit from the object corresponds to a digital input (manufacturer specific or device profile defined). If a bit is SET, then the status of the corresponding input is logical ‘1’ (high). If the bit is RESET, then the corresponding drive input status is logical ‘0’ (low).

[12] Only valid for MotionChip III family

[13] Only valid for MotionChip III family

Remarks:

The device profile defined inputs (limit switches, home input and interlock) are mapped also on the manufacturer specific inputs. Hence, when one of these inputs changes the status, then both bits change, from the manufacturer specific list and from the device profile list.

The number of available digital inputs is product dependent. Check the drive user manual for the available digital inputs.

Object description:

Index	$60FD_h$
Name	Digital inputs
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RO
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

	Bit	Description
Manufacturer specific	31	IN15 status
	30	IN14 status
	29	IN13 status
	28	IN12 status
	27	IN11 status
	26	IN10 status
	25	IN9 status
	24	IN8 status
	23	IN7 status
	22	IN6 status
	21	IN5 status
	20	IN4 status
	19	IN3 status
	18	IN2 status
	17	IN1 status
	16	IN0 status
Device profile defined	15..4	Reserved
	3	Interlock (Drive enable)
	2	Home switch status
	1	Positive limit switch status

	0	Negative limit switch status

4.4.2. Object 2042h: Digital inputs mask[14]

The role of this object is to provide a mask through which the user can program which drive inputs should be actively monitored in the event that object $60FD_h$ is mapped on a transmit PDO.

Object description:

Index	2042_h
Name	Digital inputs mask
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Units	-
Value range	UNSIGNED32
Default value	$E00F0000_h$

Data description:

MSB **LSB**

Manufacturer specific	reserved	Interlock (enable)	Home switch	Positive limit switch	Negative limit switch
31 16	15 4	3	2	1	0

Data description – manufacturer specific:

Bit	Meaning	Bit	Meaning
31	IN15 mask	23	IN7 mask
30	IN14 mask	22	IN6 mask
29	IN13 mask	21	IN5 mask
28	IN12 mask	20	IN4 mask
27	IN11 mask	19	IN3 mask
26	IN10 mask	18	IN2 mask
25	IN9 mask	17	IN1 mask
24	IN8 mask	16	IN0 mask

If the any of the non-reserved bits is **SET**, then the corresponding drive input will be actively monitored. If the bit is **RESET**, then the corresponding drive input active monitoring would stop.

[14] Only valid for MotionChip III family

4.4.3. Object 2040h: Digital inputs status[15]

The current status of the drive inputs can be monitored using this object. If a bit is SET, then the status of the corresponding input is logical '1' (high). If the bit is RESET, then the corresponding drive input status is logical '0' (low).

Remarks:

The number of available digital inputs is product dependent. Check the drive user manual for the available digital inputs.

Object description:

Index	2040_h
Name	Digital inputs status
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Value range	UNSIGNED16
Default value	0

Data description:

Bit	Meaning	Bit	Meaning
15	Drive enable status	7	IN7 status
14	Negative limit switch status	6	IN6 status
13	Positive limit switch status	5	IN5 status
12	IN12 status	4	IN4 status
11	IN11 status	3	IN3 status
10	IN10 status	2	IN2 status
9	IN9 status	1	IN1 status
8	IN8 status	0	IN0 status

4.4.4. Object 60FEh: Digital outputs[16]

The object defines the digital outputs of the drive that are controlled. The first sub-index defines the outputs that will be controlled by the mask set in the second sub-index. The second sub-index describes a mask that specifies how the selected outputs will be commanded. If any of the bits is

[15] Only valid for MotionChip II family

[16] Only valid for MotionChip III family

SET, then the corresponding drive output will be switched to logical ‘1’ (high). If the bit is **RESET**, then the corresponding drive output will be switched to logical ‘0’ (low).

Remarks:

The actual number of available digital outputs is product dependent. Check the drive user manual for the available digital outputs.

If an unavailable digital output is specified, the drive will issue an emergency message.

Object description:

Index	$60FE_h$
Name	Digital outputs
Object code	ARRAY
Data type	UNSIGNED32

Entry description:

Sub-index	0
Description	Number of entries
Access	RO
PDO mapping	No
Value range	1…2
Default value	2

Sub-index	1
Description	Physical outputs
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

Sub-index	2
Description	Bit mask
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

Table 4.8 *Bits mask description:*

	Bit	Description

Manufacturer Specific	31	OUT15 command
	30	OUT14 command
	29	OUT13 command
	28	OUT12 command
	27	OUT11 command
	26	OUT10 command
	25	OUT9 command
	24	OUT8 command
	23	OUT7 command
	22	OUT6 command
	21	OUT5 command
	20	OUT4 command
	19	OUT3 command
	18	OUT2 command
	17	OUT1 command
	16	OUT0 command
Device profile Defined	15…1	Reserved
	0	Set brake

4.4.5. Object 2045h: Digital outputs status[17]

The actual status of the drive outputs can be monitored using this object.

Object description:

Index	2045_h
Name	Digital outputs status
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Units	-

[17] Only valid for MotionChip III family

Value range	UNSIGNED16
Default value	No

Data description:

Bit	Meaning	Bit	Meaning
15	OUT15 status	7	OUT7 status
14	OUT14 status	6	OUT6 status
13	OUT13 status	5	OUT5 status
12	OUT12 status	4	OUT4 status
11	OUT11 status	3	OUT3 status
10	OUT10 status	2	OUT2 status
9	OUT9 status	1	OUT1 status
8	OUT8 status	0	OUT0 status

If the any of the bits is **SET**, then the corresponding drive output status is logical '1' (high). If the bit is **RESET**, then the corresponding drive output status is logical '0' (low).

4.4.6. Object 2043h: Digital outputs command[18]

This object is used to control the status of the digital outputs of the drive.

Remarks:

The actual number of available digital outputs is product dependent. Check the drive user manual for the available digital outputs.

Object description:

Index	2043_h
Name	Digital outputs command
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	Possible
Units	-
Value range	UNSIGNED16
Default value	No

Data description:

Bit	Meaning	Bit	Meaning

[18] Only valid for MotionChip II family

15	Ready output command	7	Ready output selection
14	Error output command	6	Error output selection
13	OUT5 command	5	OUT5 selection
12	OUT4 command	4	OUT4 selection
11	OUT3 command	3	OUT3 selection
10	OUT2 command	2	OUT2 selection
9	OUT1 command	1	OUT1 selection
8	OUT0 command	0	OUT0 selection

If any of the *selection* bits (0…7) is **SET**, then the corresponding drive output will be selected to be switched according to the information in the *command* bits (8…15). If an output port is selected and the command bit is **SET**, then the corresponding drive output will be switched to logical "1" (high). If the command bit is **RESET**, then the corresponding drive output will be switched to logical "0" (low).

4.4.7. Object 2046h: Analogue input: Reference[19]

The object contains the actual value of the analog reference applied to the drive. Through this object one can supervise the analogue input dedicated to receive the analogue reference in the external control modes.

Object description:

Index	2046_h
Name	Analogue input: Reference
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Units	-
Value range	0 … 65472
Default value	No

4.4.8. Object 2047h: Analogue input: Feedback[20]

The object contains the actual value of the analogue feedback applied to the drive.

Object description:

Index	2047_h

[19] Only valid for MotionChip III family

[20] Only valid for MotionChip III family

Name	Analogue input: Feedback
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Units	-
Value range	0 … 65272
Default value	No

4.4.9. Object 2055h: DC-link voltage[21]

The object contains the actual value of the DC-link voltage. The object is expressed in internal voltage units.

Object description:

Index	2055_h
Name	Analogue input: DC-link voltage
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Units	DC-VU
Value range	0 … 65472
Default value	No

4.4.10. Object 2058h: Drive Temperature[22]

The object contains the actual drive temperature. The object is expressed in temperature internal units.

Object description:

Index	2058_h
Name	Analogue input for drive temperature
Object code	VAR
Data type	UNSIGNED16

[21] Only valid for MotionChip III family

[22] Only valid for MotionChip III family

Entry description:

Access	RO
PDO mapping	Possible
Units	-
Value range	0 … 65535
Default value	No

4.5. Protections Setting Objects

4.5.1. Object 2050h: Over-current protection level

The Over-Current Protection Level object together with object Over-Current Time Out (2051_h) defines the drive over-current protection limits. The object defines the value of current in the drive, over which the over-current protection will be activated, if lasting more than a time interval that is specified in object 2051_h. It is set in current internal units.

Object description:

Index	2050_h
Name	Over-current protection level
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Units	CU
Value range	0 … 32767
Default value	No

4.5.2. Object 2051h: Over-current time out

The Over-Current time out object together with object Over-Current Protection Limit (2050_h) defines the drive over-current protection limits. The object sets the time interval after which the over-current protection is triggered if the drive current exceeds the value set through object 2050_h. It is set in time internal units.

Object description:

Index	2051_h
Name	Over-current time out
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW

PDO mapping	Possible
Units	TU
Value range	0 … 65535
Default value	No

4.5.3. Object 2052h: Motor nominal current

The object sets the maximum motor current RMS value for continuous operation. This value is used by the I2t motor protection and one of the start methods. It is set in current internal units. See object 2053 for more details about the I2t motor protection.

Object description:

Index	2052_h
Name	Motor nominal current
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Units	CU
Value range	0 … 32767
Default value	No

4.5.4. Object 2053h: I2t protection integrator limit

Objects 2053_h and 2054_h contain the parameters of the I^2t protection (against long-term motor over-currents). Their setting must be coordinated with the setting of the object 2052_h, motor nominal current. Select a point on the I^2t motor thermal protection curve, which is characterized by the points I_I2t (current, [A]) and t_I2t: (time, [s]) (see **Figure 4.2**)

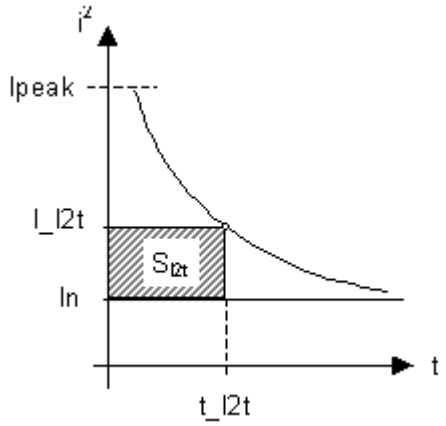


Figure 4.2 *I2t motor thermal protection curve*

The points I_I2t and t_I2t on the motor thermal protection curve together with the nominal motor current **In** define the surface S_{I2t}. If the motor instantaneous current is greater than the nominal current In and the I2t protection is activated, the difference between the square of the instantaneous current and the square of the nominal current is integrated and compared with the SI2t value (see **Figure 4.3**). When the integral equals the SI2t surface, the I2t protection is triggered.

Object description:

Index	2053_h
Name	I2t protection integrator limit
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	0 … 2^{31}-1
Default value	No

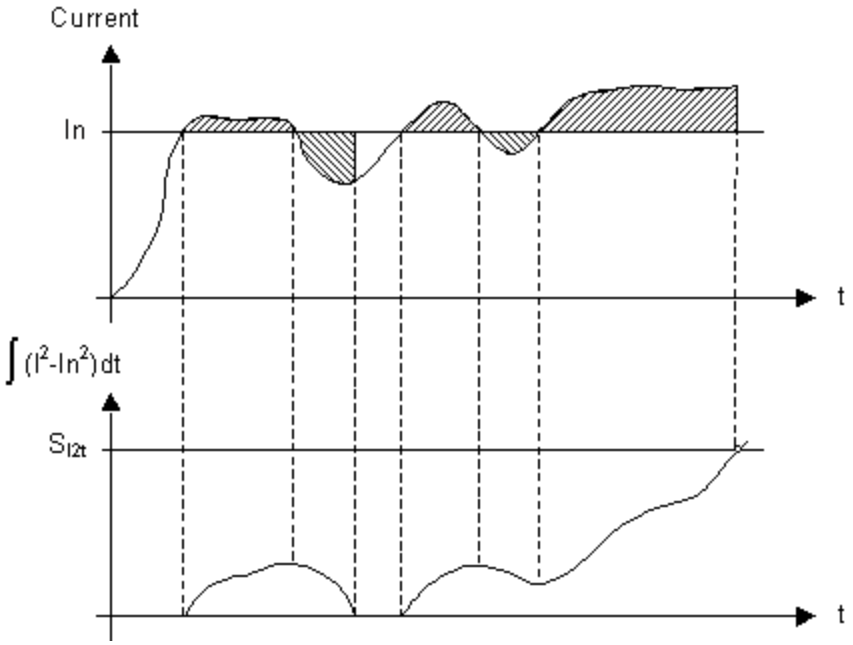


Figure 4.3 *I2t protection implementation*

The computation formula for the i2t protection integrator limit (I2TINTLIM) is

$$I2TINTLIM = \frac{(I\_I2t)^2 - (In)^2}{32767^2} \cdot 2^{26}$$

where I_I2t and In are represented in current units (CU).

4.5.5. Object 2054h: I2t protection scaling factor

Object description:

Index	2054_h
Name	I2t protection scaling factor
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	0 … 65535
Default value	No

The computation formula for the i2t protection scaling factor (SFI2T) is

$$SFI2T = 2^{26} \cdot \frac{Ts\_S}{t\_I2t}$$

where Ts_S is the sampling time of the speed control loop [s], and t_I2t is the I2t protection time corresponding to the point on the graphic in **Figure 4.2**

4.6. Drive info objects

4.6.1. Object 1000h: Device Type

The object contains information about drive type and its functionality. The 32-bit value contains 2 components of 16-bits: the 16 LSB describe the CiA standard that is followed.

Object description:

Index	1000_h
Name	Device type
Object code	VAR
Data type	UNSIGNED32

Value description:

Access	RO
PDO mapping	NO
Value range	UNSIGNED32

Default value	60192_h for MotionChip III family 20192_h for MotionChip II family of servo drives 40192_h for MotionChip II family of stepper drives

4.6.2. Object 6502h: Supported drive modes

This object gives an overview of the operating modes supported on the Technosoft drives. Each bit from the object has assigned an operating mode. If the bit is set then the drive supports the associated operating mode.

Object description:

Index	6502_h
Name	Supported drive modes
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RO
PDO mapping	Possible
Value range	UNSIGNED32
Default value	$001F0065_h$ for MotionChip III family 00000065_h for MotionChip II family

The modes of operation supported by the Technosoft drives, and their corresponding bits, are the following:

Data description:

MSB												**LSB**
0	0	x	…	x	0	0	1	1	0	0	1	0	1
Manufacturer specific					reserved		ip	hm	reserved	tq	pv	vl	pp
31	21	20	…	16	15	7	6	5	4	3	2	1	0

Data description – manufacturer specific:

Bit	**Description**
31 … 21	Reserved
20	External Reference Torque Mode
19	External Reference Speed Mode
18	External Reference Position Mode
17	Electronic Gearing Position Mode

4.6.3. Object 1008h: Manufacturer Device Name[23]

The object contains the manufacturer device name in ASCII form, maximum 15 characters.

Object description:

Index	1008_h
Name	Manufacturer device name
Object code	VAR
Data type	Visible String

Entry description:

Access	Const
PDO mapping	No
Value range	No
Default value	IDM680

4.6.4. Object 100Ah: Manufacturer Software Version[24]

The object contains the firmware version programmed on the drive in ASCII form with the maximum length of 15 characters.

Object description:

Index	$100A_h$
Name	Manufacturer software version
Object code	VAR
Data type	Visible String

Entry description:

Access	Const
PDO mapping	No
Value range	No
Default value	Product dependent

4.6.5. Object 2060h: Software version of a TML application[25]

By inspecting this object the user can find out the software version of the TML application (drive setup plus motion setup and eventually cam tables) that is stored in the EEPROM memory of the

[23] Only valid for MotionChip III family

[24] Only valid for MotionChip III family

[25] Only valid for MotionChip III family

drive. The object stores the software version coded in a string of 4 elements, grouped in a 32-bit variable. Each byte represents an ASCII character.

Object description:

Index	2060_h
Name	Software version of TML application
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RO
PDO mapping	No
Units	-
Value range	No
Default value	No

Example:

If object 2060_h contains the value 0x322E3156, then the software version of the TML application is read as:

0x56 – ASCII code of letter **V**

0x31 – ASCII code of number **1**

0x2E – ASCII code of character **.**

0x32 – ASCII code of number **2**

So the version is **V1.2**.

4.6.6. Object 1018h: Identity Object

Contains the unique VendorID allocated to Technosoft.

Object description:

Index	1018_h
Name	Identitty Object
Object code	RECORD
Data type	Identity

Entry description:

Sub-index	00_h
Description	Number of entries
Access	RO
PDO mapping	No
Value range	1..4
Default value	1

Sub-index	01_h
Description	Vendor ID
Access	RO
PDO mapping	No
Value range	UNSIGNED32
Default value	000001A3h

4.7. Miscellaneous Objects

4.7.1. Object 2025h: Stepper current in open-loop operation[26]

In this object one can set the level of the current to be applied when controlling a stepper motor in open loop operation at runtime.

Object description:

Index	2025_h
Name	Stepper current in open-loop operation
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RW
PDO mapping	Possible
Units	CU
Value range	-32768 … 32767
Default value	No

4.7.2. Object 2026h: Stand-by current for stepper in open-loop operation[27]

In this object one can set the level of the current to be applied when controlling a stepper motor in open loop operation in stand-by.

Object description:

Index	2026_h
Name	Stand-by current for stepper in open-loop operation
Object code	VAR
Data type	INTEGER16

[26] Only valid for MotionChip III family & the intelligent stepper drives of the MotionChip II family

[27] Only valid for MotionChip III family & the intelligent stepper drives of the MotionChip II family

Entry description:

Access	RW
PDO mapping	Possible
Units	CU
Value range	-32768 … 32767
Default value	No

4.7.3. Object 2027h: Timeout for stepper stand-by current[28]

In this object one can set the amount of time after the value set in object 2026h, *stand-by current for stepper in open-loop operation* will activate as the reference for the current applied to the motor after the reference has reached the target value.

Object description:

Index	2027_h
Name	Timeout for stepper stand-by current
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Units	TU
Value range	0 … 65535
Default value	No

4.7.4. Object 2075h: Position triggers[29]

This object is used in order to define a set of 4 position values whose proximity will be signaled through bits 17-20 of object 1002_h, *Manufacturer Status Register*. If the *position actual value* is over a certain value set as a position trigger, then the corresponding bit in *Manufacturer Status Register* will be set.

Object description:

Index	2075_h
Name	Position triggers
Object code	ARRAY
Data type	INTEGER32

Entry description:

[28] Only valid for MotionChip III family & the intelligent stepper drives of the MotionChip II family

[29] Only valid for MotionChip III family

Sub-index	00_h
Description	Number of sub-indexes
Access	RO
PDO mapping	No
Default value	4

Sub-index	01_h – 04_h
Description	Position trigger 1 - 4
Access	RW
PDO mapping	Possible
Value range	INTEGER32
Default value	No

4.7.5. Object 2076h: Save current configuration

This object is used in order to enable saving the current configuration of the operating parameters of the drive. These parameters are the ones that are set when doing the setup of the drive. The purpose of this object is to be able to save the new values of these parameters in order to be re-initialized at subsequent system re-starts.

Writing any value in this object will trigger the save in the non-volatile EEPROM memory of the current drive operating parameters.

Object description:

Index	2076_h
Name	Save current configuration
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Value range	UNSIGNED16
Default value	-

Remark:

There is a special case of using this object for an IDM680-8EI with SSI encoder and IDM680-8BI with BiSS encoder – as these 2 sensors use the same SPI interface used for communication with the EEPROM. The steps to correctly use this object in these cases are the following:

1. **Set the drive into *Switch on Disabled* status**.
2. **Enable EEPROM**. Send the following TechnoCAN message (assumes the NodeID is 10):

COB-ID	**12A**
Data	**00 95**

3. **Save the parameters**. Write value 1 inside object 2076_h.

Send the following message (SDO write to object 2076_h sub-index 0, 16-bit value 0001_h):

COB-ID	**60A**
Data	**2B 76 20 00 01 00 00 00**

4. **Reset the drive**. Needed in order to re-establish encoder connection.

Send the following message (NMT message for reset node to drive with NodeID 10):

COB-ID	**0**
Data	**81 0A**

4.7.6. Object 2078h: Execute auto-tuning for Linear Halls configuration[30]

This object is used in order to start the automatic sequence of tuning the linear halls sensors in a configuration having a brushless motor with linear hall sensors.

Writing any value in this object will trigger the start of the automatic tuning sequence.

Object description:

Index	2078_h
Name	Execute auto-tuning for Linear Halls configuration
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Value range	UNSIGNED16
Default value	-

[30] Only valid for MotionChip III family

5. Factor group[31]

The MotionChip III drives family offers the possibility to interchange physical dimensions and sizes into the device internal units. This chapter describes the factors that are necessary to do the interchanges.

Remark:

The MotionChip II drives family currently does not support the factor group and its objects.

The factors defined in Factor Group set up a relationship between device internal units and physical units. The actual factors used for scaling are the *position factor* (object 6093_h), the *velocity encoder factor* (object 6094_h), the *acceleration factor* (object 6097_h) and the *time encoder factor* (object 2071_h). Writing a non-zero value into the respective dimension index objects validates these factors. The notation index objects are used for status only and can be set by the user depending on each user-defined value for the factors.

5.1. Factor group objects

5.1.1. Object 6089h: Position notation index

The *position notation index* is used to scale the following objects*:*

- Position actual value
- Position demand value
- Target position
- Position window
- Following error window
- Following error actual value

Object description:

Index	6089_h
Name	Position notation index
Object code	VAR
Data type	INTEGER8

Entry description:

Access	RW
PDO mapping	Possible
Value range	-128 … 127
Default value	0

[31] This chapter, along with all the objects described inside it, is only valid for MotionChip III family

5.1.2. Object 608Ah: Position dimension index

The *position dimension index* is used to scale the following objects*:*

- Position actual value
- Position demand value
- Target position
- Position window
- Following error window
- Following error actual value

Object description:

Index	$608A_h$
Name	Position dimension index
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RW
PDO mapping	Possible
Value range	0 … 255
Default value	0

5.1.3. Object 608Bh: Velocity notation index

The *velocity notation index* is used to scale the following objects*:*

- Velocity actual value
- Velocity demand value
- Target velocity
- Profile velocity

Object description:

Index	$608B_h$
Name	Velocity notation index
Object code	VAR
Data type	INTEGER8

Entry description:

Access	RW
PDO mapping	Possible
Value range	-128 … 127
Default value	0

5.1.4. Object 608Ch: Velocity dimension index

The *velocity dimension index* is used to scale the following objects*:*

- Velocity actual value
- Velocity demand value
- Target velocity
- Profile velocity

Object description:

Index	$608C_h$
Name	Velocity dimension index
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RW
PDO mapping	Possible
Value range	0 … 255
Default value	0

5.1.5. Object 608Dh: Acceleration notation index

The *acceleration notation index* is used to scale the following objects*:*

- Profile acceleration
- Quick stop deceleration

Object description:

Index	$608D_h$
Name	Acceleration notation index
Object code	VAR
Data type	INTEGER8

Entry description:

Access	RW
PDO mapping	Possible
Value range	-128 … 127
Default value	0

5.1.6. Object 608Eh: Acceleration dimension index

The *acceleration dimension index* is used to scale the following objects:

- Profile acceleration
- Quick stop deceleration

Object description:

Index	$608E_h$
Name	Acceleration dimension index
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RW
PDO mapping	Possible
Value range	0 … 255
Default value	0

5.1.7. Object 206Fh: Time notation index

The *time dimension index* is used to scale the following objects:

- Following error time out
- Position window time
- Jerk time
- Max slippage time out
- Over-current time out

Object description:

Index	$206F_h$
Name	Time notation index
Object code	VAR

Data type	INTEGER8

Entry description:

Access	RW
PDO mapping	Possible
Value range	-128 … 127
Default value	0

5.1.8. Object 2070h: Time dimension index

The *time dimension index* is used to scale the following objects:

- Following error time out
- Position window time
- Jerk time
- Max slippage time out
- Over-current time out

Object description:

Index	2070_h
Name	Time dimension index
Object code	VAR
Data type	UNSIGNED8

Entry description:

Access	RW
PDO mapping	Possible
Value range	0 … 255
Default value	0

5.1.9. Object 6093h: Position factor (DS402)

The *position factor* converts the desired position (in position units) into the internal format (in increments):

$$Position[IU] = Position[UserUnits] \times \frac{PositionFactor.Numerator}{PositionFactor.Divisor}$$

Object description:

Index	6093_h

Name	Position factor
Object code	ARRAY
Number of elements	2
Data type	UNSIGNED32

Entry description:

Sub-index	01_h
Description	Numerator
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

Sub-index	02_h
Description	Divisor
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

5.1.10. Object 6094h: Velocity encoder factor (DS402)

The *velocity encoder factor* converts the desired velocity (in velocity units) into the internal format (in increments).

$$Velocity[IU] = Velocity[UserUnits] \times \frac{VelocityEncoderFactor.Numerator}{VelocityEncoderFactor.Divisor}$$

Object description:

Index	6094_h
Name	Velocity encoder factor
Object code	ARRAY
Number of elements	2
Data type	UNSIGNED32

Entry description:

Sub-index	01_h
Description	Numerator
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

Sub-index	02_h
Description	Divisor
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

5.1.11. Object 6097h: Acceleration factor (DS402)

The *acceleration factor* converts the velocity (in acceleration units/sec^2) into the internal format (in increments/sampling^2).

$$Acceleration[IU] = Acceleration[UserUnits] \times \frac{AccelerationFactor.Numerator}{AccelerationFactor.Divisor}$$

Object description:

Index	6097_h
Name	Acceleration factor
Object code	ARRAY
Number of elements	2
Data type	UNSIGNED32

Entry description:

Sub-index	01_h
Description	Numerator
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32

Default value	1

Sub-index	02_h
Description	Divisor
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

5.1.12. Object 2071h: Time factor

The *time factor* converts the desired time values (in time units) into the internal format (in speed / position loop samplings).

$$Time[IU] = Time[UserUnits] \times \frac{TimeFactor.Numerator}{TimeFactor.Divisor}$$

Object description:

Index	2071_h
Name	Time factor
Object code	ARRAY
Number of elements	2
Data type	UNSIGNED32

Entry description:

Sub-index	01_h
Description	Numerator
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

Sub-index	02_h
Description	Divisor
Access	RW

PDO mapping	Possible
Value range	UNSIGNED32
Default value	1

6. Homing Mode

6.1. Overview

Homing is the method by which a drive seeks the home position. There are various methods to achieve this position using the four available sources for the homing signal: limit switches (negative and positive), home switch and index pulse.

A homing move is started by setting bit 4 of the *Control Word* object (index 0x6040). The results of a homing operation can be accessed in the *Status Word* (index 0x6041).

A homing mode is chosen by writing a value to homing_method which will clearly establish:

1. the homing signal (positive limit switch, negative limit switch, home switch)
2. the direction of actuation and where appropriate
3. the position of the index pulse.

The user can specify the home method, the home offset, the speed and the acceleration.

The **home offset** is the difference between the zero position for the application and the machine home position. During homing the home position is found and once the homing is completed the zero position is offset from the home position by adding the home_offset to the home position. This is illustrated in the following diagram.

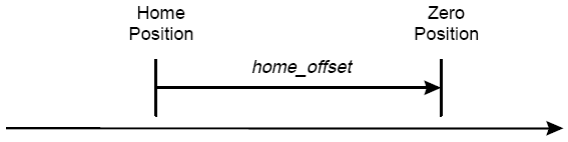


Figure 6.1 *Home Offset*

There are two **homing speeds:** a fast speed (which is used to find the home switch), and a slow speed (which is used to find the index pulse).

The **homing acceleration** establishes the acceleration to be used for all accelerations and decelerations with the standard homing modes.

The homing method descriptions in this document are based on those in the Profile for Drives and Motion Control (DSP402 Standard).

As in the figure below for each homing method we will present a diagram that clearly describes the sequence of homing operation.

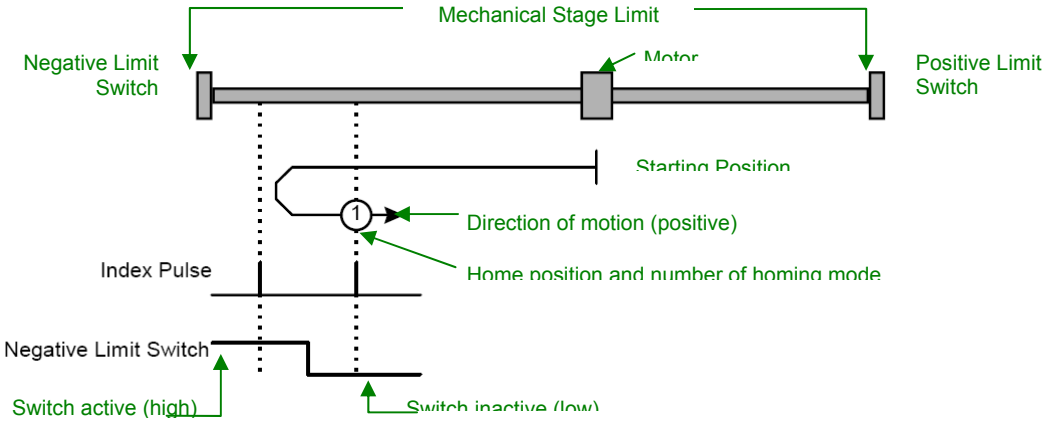


Figure 6.2 *Homing method diagram*

Method 1: Homing on the Negative Limit Switch.

If the negative limit switch is inactive (low) the initial direction of movement is leftward (negative sense). After negative limit switch is reached the motor will reverse the motion, moving in the positive sense with slow speed. The home position is at the first index pulse to the right of the position where the negative limit switch becomes inactive.

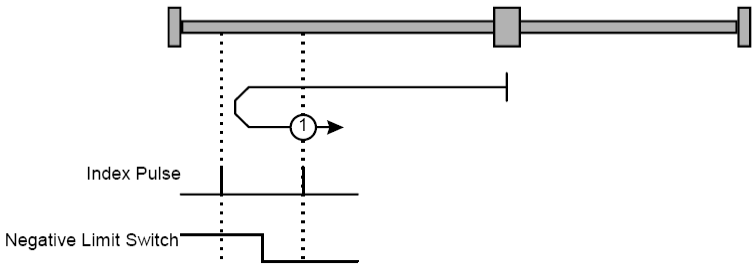


Figure 6.3 *Homing on the Negative Limit Switch*

Method 2: Homing on the Positive Limit Switch.

If the positive limit switch is inactive (low) the initial direction of movement is rightward (negative sense). After positive limit switch is reached the motor will reverse the motion, moving in the negative sense with slow speed. The home position is at the first index pulse to the left of the position where the positive limit switch becomes inactive.

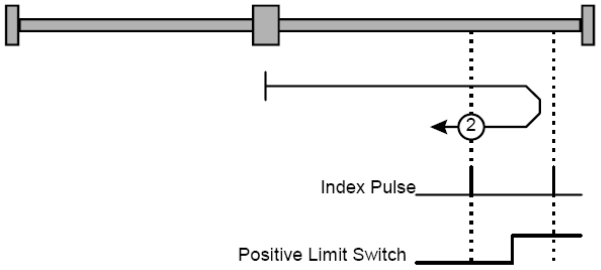


Figure 6.4 *Homing on the Positive Limit Switch*

Methods 3 and 4: Homing on the Positive Home Switch and Index Pulse.

The home position is at the index pulse either after home switch high-low transition (method 3) or after home switch low-high transition (method 4).

The initial direction of movement is dependent on the state of the home switch (if low - move positive, if high - move negative).

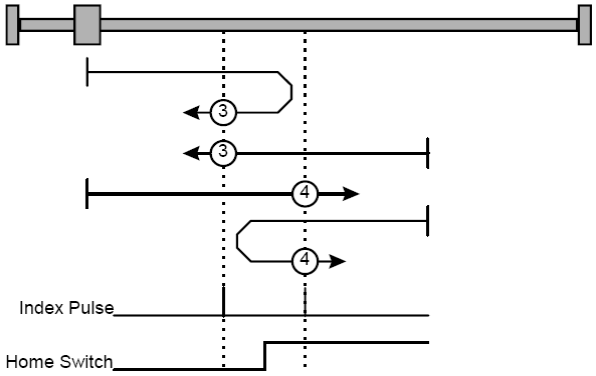


Figure 6.5 *Homing on the Negative Home Switch and Index Pulse*

For **method 3**, if home input is high the initial direction of movement will be negative, or positive if home input is low, and reverse (with slow speed) after home input low-high transition. The motor will stop at first index pulse after home switch high-low transition.

For **method 4**, if home input is low the initial direction of movement will be positive, or negative if home input is high, and reverse (with slow speed) after home input high-low transition. The motor will stop at first index pulse after home switch low-high transition.

In all cases after home switch transition the speed of the movement is slow.

Methods 5 and 6: Homing on the Negative Home Switch and Index Pulse.

The home position is at the index pulse either after home switch high-low transition (method 5) or after home switch low-high transition (method 6).

The initial direction of movement is dependent on the state of the home switch (if high - move positive, if low - move negative).

For **method 5**, if home input is high the initial direction of movement will be positive, or negative if home input is low, and reverse (with slow speed) after home input low-high transition. The motor will stop at first index pulse after home switch high-low transition.

For **method 6**, if home input is low the initial direction of movement will be negative, or positive if home input is high, and reverse (with slow speed) after home input high-low transition. The motor will stop at first index pulse after home switch low-high transition.

In all cases after home switch transition the speed of the movement is slow.

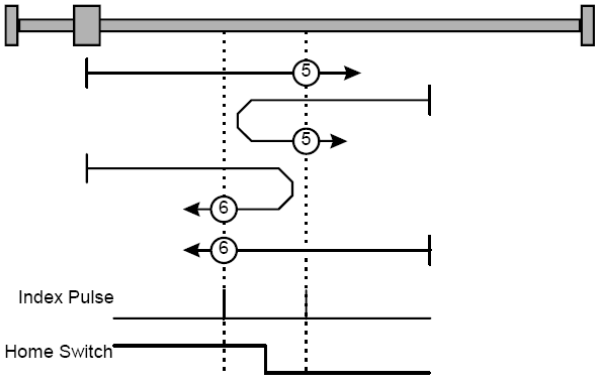


Figure 6.6 *Homing on the Negative Home Switch and Index Pulse*

Methods 7 to14: Homing on the Negative Home Switch and Index Pulse.

These methods use a home switch that is active over only a portion of the travel distance, in effect the switch has a 'momentary' action as the axle's position sweeps past the switch.

Using methods 7 to 10 the initial direction of movement is to the right (positive), and using methods 11 to 14 the initial direction of movement is to the left (negative), except the case when the home switch is active at the start of the motion (initial direction of motion is dependent on the edge being sought – the rising edge or the falling edge).

The home position is at the index pulse on either side of the rising or falling edges of the home switch, as shown in the following two diagrams.

If the initial direction of movement leads away from the home switch, the drive will reverse on encountering the relevant limit switch (negative limit switch for methods 7 to 10, or positive limit switch for methods 11 to 14).

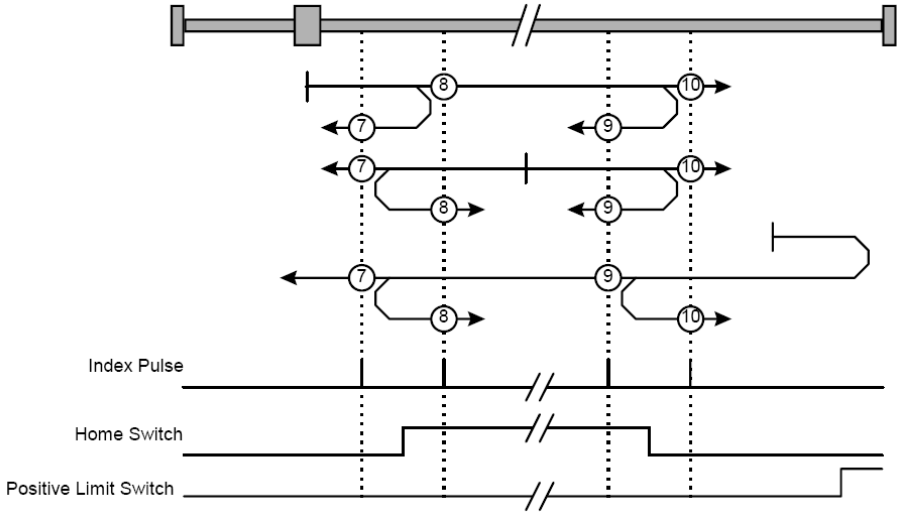


Figure 6.7 *Homing on the Home Switch and Index Pulse – Positive Initial Move*

Using **method 7** the initial move will be positive if home input is low and reverse after home input low-high transition, or move negative if home input is high. Reverse also if the positive limit switch is reached. Stop at first index pulse after home switch active region ends (high-low transition). In all cases after high-low home switch transition the motor speed will be slow.

Using **method 8** the initial move will be positive if home input is low, or negative if home input is high and reverse after home input high-low transition. Reverse also if the positive limit switch is reached. Stop at first index pulse after home switch active region starts (low-high transition). In all cases after low-high home switch transition the motor speed will be slow.

Using **method 9** the initial move will be positive and reverse(slow speed) after home input high-low transition. Reverse also if the positive limit switch is reached. Stop at first index pulse after home switch active region starts (low-high transition).

Using **method 10** the initial move will be positive. Reverse if the positive limit switch is reached, then reverse once again after home input low-high transition. Stop at first index pulse after home switch active region ends (high-low transition). In all cases after high-low home switch transition the motor speed will be slow.

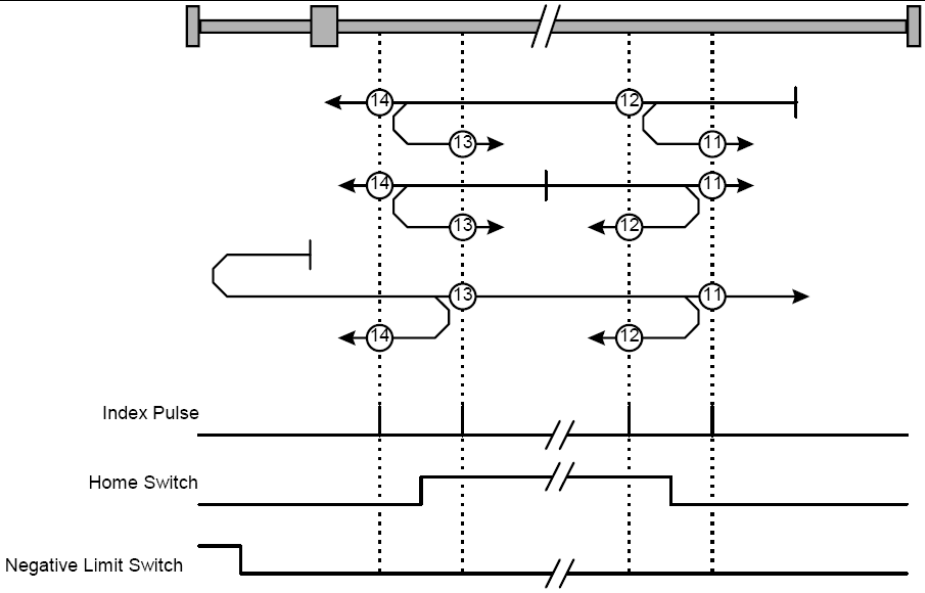


Figure 6.8 *Homing on the Home Switch and Index Pulse – Positive Initial Move*

Using **method 11** the initial move will be negative if home input is low and reverse after home input low-high transition. Reverse also if the negative limit switch is reached. If home input is high move positive. Stop at first index pulse after home switch active region ends (high-low transition). In all cases after high-low home switch transition the motor speed will be slow.

Using **method 12** the initial move will be negative if home input is low. If home input is high move positive and reverse after home input high-low transition. Reverse also if the negative limit switch is reached. Stop at first index pulse after home switch active region starts (low-high transition). In all cases after low-high home switch transition the motor speed will be slow.

Using **method 13** the initial move will be negative and reverse after home input high-low transition. Reverse also if the negative limit switch is reached. Stop at first index pulse after home switch active region starts (low-high transition). In all cases after high-low home switch transition the motor speed will be slow.

Using **method 14** the initial move will be negative. Reverse if the negative limit switch is reached, then reverse once again after home input low-high transition. Stop at first index pulse after home switch active region ends (high-low transition). In all cases after high-low home switch transition the motor speed will be slow.

Methods 15 and 16: Reserved

Methods 17 to 30: Homing without an Index Pulse

These methods are similar to methods 1 to 14 except that the home position is not dependent on the index pulse but only on the relevant home or limit switch transitions.

Using **method 17** if the negative limit switch is inactive (low) the initial direction of movement is leftward (negative sense). After negative limit switch reached the motor will reverse the motion,

moving in the positive sense with slow speed. The home position is at the right of the position where the negative limit switch becomes inactive.

Using **method 18** if the positive limit switch is inactive (low) the initial direction of movement is rightward (negative sense). After positive limit switch reached the motor will reverse the motion, moving in the negative sense with slow speed. The home position is at the left of the position where the positive limit switch becomes inactive.

For example methods 19 and 20 are similar to methods 3 and 4 as shown in the following diagram.

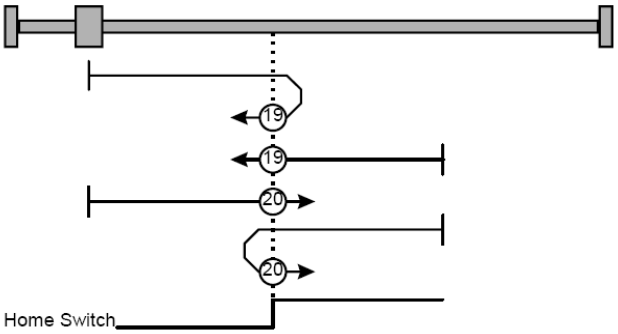


Figure 6.9 *Homing on the Positive Home Switch*

Using **method 19**, if home input is high, the initial direction of movement will be negative, or positive if home input is low, and reverse (with slow speed) after home input low-high transition. The motor will stop right after home switch high-low transition.

Using **method 20**, if home input is low, the initial direction of movement will be positive, or negative if home input is high, and reverse (with slow speed) after home input high-low transition. The motor will stop after right home switch low-high transition.

Using **method 21**, if home input is high, the initial direction of movement will be positive, or negative if home input is low, and reverse (with slow speed) after home input low-high transition. The motor will stop right after home switch high-low transition.

Using **method 22**, if home input is low, the initial direction of movement will be negative, or positive if home input is high, and reverse (with slow speed) after home input high-low transition. The motor will stop right after home switch low-high transition.

Using **method 23** the initial move will be positive if home input is low and reverse after home input low-high transition, or move negative if home input is high. Reverse also if the positive limit switch is reached. Stop right after home switch active region ends (high-low transition).

Using **method 24** the initial move will be positive if home input is low, or negative if home input is high and reverse after home input high-low transition. Reverse also if the positive limit switch is reached. Stop right after home switch active region starts (low-high transition).

Using **method 25** the initial move will be positive and reverse after home input high-low transition. Reverse also if the positive limit switch is reached. Stop right after home switch active region starts (low-high transition).

Using **method 26** the initial move will be positive. Reverse if the positive limit switch is reached, then reverse once again after home input low-high transition. Stop right after home switch active region ends (high-low transition).

Using **method 27** the initial move will be negative if home input is low and reverse after home input low-high transition. Reverse also if the negative limit switch is reached. If home input is high move positive. Stop right after home switch active region ends (high-low transition).

Using **method 28** the initial move will be negative if home input is low. If home input is high move positive and reverse after home input high-low transition. Reverse also if the negative limit switch is reached. Stop right after home switch active region starts (low-high transition).

Using **method 29** the initial move will be negative and reverse after home input high-low transition. Reverse also if the negative limit switch is reached. Stop right after home switch active region starts (low-high transition).

Using **method 30** the initial move will be negative. Reverse if the negative limit switch is reached, then reverse once again after home input low-high transition. Stop right after home switch active region ends (high-low transition).

Methods 31 and 32: Reserved

Methods 33 and 34: Homing on the Index Pulse

Using **methods 33** or **34** the direction of homing is negative or positive respectively. During those procedure the motor will move only at slow speed. The home position is at the index pulse found in the selected direction.

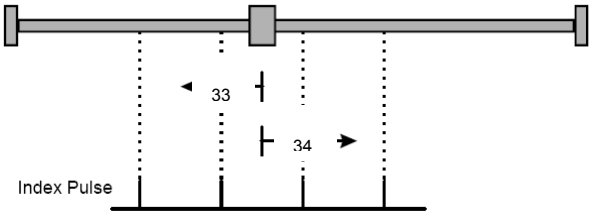


Figure 6.10 *Homing on the Index Pulse*

Method 35: Homing on the Current Position

In **method 35** the current position is taken as the home position.

6.2. Homing Mode Objects

This chapter describes the method by which the drive seeks the home position. There are 35 built-in homing methods, as described in **paragraph 6.1**. Using the EasyMotion Studio software, one can alter each of these homing methods to create a custom homing method.

You can select which homing method to be used by writing the appropriate number in the object 6098h *homing method*.

The user can specify the speeds and acceleration to be used during the homing. There is a further object *homing offset* that allows the user to displace zero in the user's coordinate system from the home position.

In the homing mode, the bits in *control word* and *status word* have the following meaning:

Control word in homing mode

MSB									**LSB**
See 6040h		Halt	See 6040h	Reserved		Homing operation start	See 6040h		
15	9	8	7	6	5	4	3	0	

Table 6.1 *Control Word bits description for Homing Mode*

Name	**Value**	**Description**
Homing operation start	0	Homing mode inactive
	0 -> 1	Start homing mode
	1	Homing mode active
	1 -> 0	Interrupt homing mode
Halt	0	Execute the instruction of bit 4
	1	Stop drive with *homing acceleration*

Status word in homing mode

MSB									LSB
See 6041h		Homing error	Homing attained	See 6041h	Target reached	See 6041h			
15	14	13	12	11	10	9			0

Table 6.2 *Status Word bits description for Homing Mode*

Name	**Value**	**Description**
Target reached	0	Halt = 0: Home position not reached Halt = 1: Drive decelerates
	1	Halt = 0: Home position reached Halt = 1: Velocity of drive is 0
Homing attained	0	Homing mode not yet completed
	1	Homing mode carried out successfully
Homing error	0	No homing error
	1	Homing error occurred; homing mode not carried out successfully.

6.2.1. Object 607Ch: Home offset

The *home offset* is the difference between the zero position for the application and the machine home position (found during homing), and it is given in position units. All subsequent absolute moves after the homing is finished will be taken relative to this zero position.

Object description:

Index	$607C_h$
Name	Home offset
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	Possible
Units	PU
Value range	INTEGER32
Default value	0

6.2.2. Object 6098h: Homing method

The *homing method* determines the method that will be used during homing.

Object description:

Index	6098_h
Name	Homing method
Object code	VAR
Data type	INTEGER8

Entry description:

Access	RW
PDO mapping	Possible
Value range	INTEGER8
Default value	0

Data description:

Value	**Description**
-128 … -1	Reserved
0	No homing operation required
1 … 35	Methods 1 to 35
36 … 127	reserved

There are 35 built-in homing methods, conforming to DSP402 device profile. Using the EasyMotion Studio software, one can alter each of these homing methods to create a custom one.

6.2.3. Object 6099h: Homing speeds

This object defines the speeds used during homing. It is given in velocity units. There are 2 homing speeds; in a typical cycle the faster speed is used to find the home switch and the slower speed is used to find the index pulse.

Object description:

Index	6099_h
Name	Homing speeds
Object code	ARRAY
Data type	UNSIGNED32

Entry description:

Sub-index	0
Description	Number of entries
Access	RO
PDO mapping	No
Value range	2
Default value	2

Sub-index	1
Description	Speed during search for switch
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

Sub-index	2
Description	Speed during search for zero
Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	0

6.2.4. Object 609Ah: Homing acceleration

The *homing acceleration* establishes the acceleration to be used for all the accelerations and decelerations with the standard homing modes and is given in acceleration units.

Object description:

Index	$609A_h$

Name	Homing acceleration
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Units	AU
Value range	UNSIGNED32
Default value	-

Example: Execute homing method number 18.

1. **Start remote node**. Send a NMT message to start the node id 6.

Send the following message:

COB-ID	**0**
Data	**01 06**

2. **Ready to switch on.** Change the node state from *Switch on disabled* to *Ready to switch on* by sending the shutdown command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**06 00**

3. **Switch on.** Change the node state from *Ready to switch on* to *Switch on* by sending the switch on command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**07 00**

4. **Enable operation.** Change the node state from *Switch on* to *Operation enable* by sending the enable operation command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**0F 00**

5. **Homing speed during search for zero.** Set the speed during search for zero to 150 rpm. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object 6099_h sub-index 2 expressed in encoder counts per sample is 50000_h.

Send the following message (SDO access to object 6099_h sub-index 2, 32-bit value 00050000_h):

COB-ID	**606**
Data	**23 99 60 02 00 00 05 00**

6. **Homing speed during search for switch.** Set the speed during search for switch to 600 rpm. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object 6099_h sub-index 1 expressed in encoder counts per sample is 140000_h.

Send the following message (SDO access to object 6099_h sub-index 1, 32-bit value 00140000_h):

COB-ID	**606**
Data	**23 99 60 01 00 00 14 00**

7. **Homing acceleration.** The homing acceleration establishes the acceleration to be used with the standard homing moves. Set this value at 5 rot/s^2. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object $609A_h$ expressed in encoder counts per square sample is $28F_h$.

Send the following message (SDO access to object $609A_h$, 32-bit value $0000028F_h$):

COB-ID	**606**
Data	**23 9A 60 00 8F 02 00 00**

8. **Home offset.** Set the home offset to 1 rotation. By using and 500 lines incremental encoder the corresponding value of object $607C_h$ expressed in encoder counts is $7D0_h$.

Send the following message (SDO access to object $607C_h$, 32-bit value $000007D0_h$):

COB-ID	**606**
Data	**23 7C 60 00 D0 07 00 00**

9. **Homing method.** Select homing method number 18.

Send the following message (SDO access to object 6098_h, 8-bit value 12_h):

COB-ID	**606**
Data	**2F 98 60 00 12 00 00 00**

10. **Mode of operation.** Select homing mode.

Send the following message (SDO access to object 6060_h, 8-bit value 6_h):

COB-ID	**606**
Data	**2F 60 60 00 06 00 00 00**

11. **Start the homing.**

Send the following message

COB-ID	**206**
Data	**1F 00**

12. **Press for 5s the LSP button.**
13. **Wait for homing to end.**
14. **Check the value of motor actual position.**

Send the following message (SDO access to object 6064_h):

COB-ID	**606**
Data	**40 64 60 00 00 00 00 00**

The node will return the value of motor actual position that should be the same with the value of home offset (plus or minus few encoder counts depending on your position tuning).

7. Position Profile Mode

7.1. Overview

In Position Profile Mode the drive controls the position.

The Position Profile Mode supports 2 motion modes:

- **Trapezoidal profile**. The built-in reference generator computes the position profile with a trapezoidal shape of the speed, due to a limited acceleration. The CANopen master specifies the absolute or relative **Target Position** (index $607A_h$), the **Profile Velocity** (index 6081_h) and the **Profile Acceleration** (6083_h)

  In relative mode, the position to reach can be computed in 2 ways: standard (default) or additive. In standard relative mode, the position to reach is computed by adding the position increment to the instantaneous position in the moment when the command is executed. In the additive relative mode, the position to reach is computed by adding the position increment to the previous position to reach, independently of the moment when the command was issued. Bit 11 of *Control Word* activates the additive relative mode.

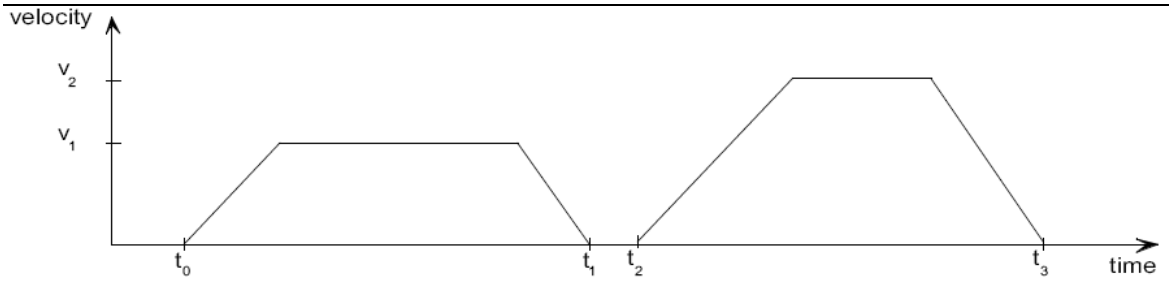
- **S-curve profile** The built-in reference generator computes a position profile with an S-curve shape of the speed. This shape is due to the jerk limitation, leading to a trapezoidal or triangular profile for the acceleration and an S-curve profile for the speed. The CANopen master specifies the absolute or relative **Target Position** (index $607A_h$), the **Profile Velocity** (index 6081_h), the **Profile Acceleration** (6083_h) and the jerk rate. The jerk rate is set indirectly via the **Jerk time** (index 2023_h), which represents the time needed to reach the maximum acceleration starting from zero.

There are two different ways to apply *target positions* to a drive, controlled by the *change set immediately* bit in Control Word:

7.1.1. Discrete motion profile (*change set immediately* = 0)

After reaching the *target position* the drive unit signals this status to a CANopen master and then receives a new set-point. After reaching a *target position* the velocity normally is reduced to zero before starting a move to the next set-point.

After the *target position* is sent to the drive, the CANopen master has to set the *new set-point* bit in *control word*. The drive responds with bit *set-point acknowledge* set in *status word*. After that, the master has to reset bit *new set-point* to 0. Following this action, the drive will signalize that it can accept a new set-point by resetting *set-point acknowledge* bit in *status word* after the reference generator has reached the designated demand position.



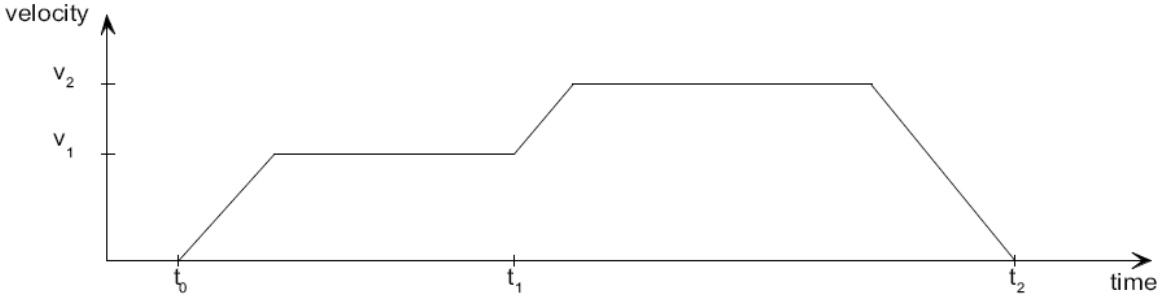
7.1.2. Continuous motion profile (*change set immediately* = 1)

The drive unit immediately processes the next *target position*, even if the actual movement is not completed. The drive readapts the actual move to the new target position.

In this case, the handshake presented for *change set immediately* = 0 is not necessary. By setting the *new set-point* bit, the master will trigger the immediate update of the target position. In this case, if the *target position* is set as relative, also bit 11 is taken into consideration (with or without additive movement).

Remark:

In case object 6086h (Motion Profile Type) is set to 3 (jerk-limited ramp = S-curve profile), then *change set immediately* bit must be 0, else a command error is issued.



7.1.3. Control word and status word in Position Profile Mode

In the profile position mode, the bits in *control word* and *status word* will have the following meaning:

Control word in profile position mode

MSB										**LSB**
See 6040h		Operation Mode	See 6040h		Halt	See 6040h	Abs/rel	Change set immediately	New set-point	See 6040h
15	12	11	10	9	8	7	6	5	4	3 0

***Table 7.1** Control Word bits description for Position Profile Mode*

Name	Value	Description
Operation Mode	0	Trapezoidal profile - In case the movement is relative, do not add the new target position to the old demand position S-curve profile – Stop the motion with S-curve profile (jerk limited ramp)
	1	Trapezoidal profile - In case the movement is relative, add the new target position to the old demand position to obtain the new target position S-curve profile – Stop the motion with trapezoidal profile (linear ramp)
New set-point	0	Do not assume *target position*
	1	Assume *target position* (update the new motion parameters)
Change set immediately	0	Finish the actual positioning and then start the next positioning
	1	Interrupt the actual positioning and start the next positioning. Valid only for linear ramp profile.
Abs / rel	0	*Target position* is an absolute value
	1	*Target position* is a relative value
Halt	0	Execute positioning
	1	Stop drive with *profile acceleration*

Status word in profile position mode

MSB **LSB**

See 6041h	Following error	Set-point acknowledge	See 6041h	Target reached	See 6041h
15 14	13	12	11	10	9 0

***Table 7.2** Status Word bits description for Position Profile Mode*

Name	Value	Description
Target reached	0	Halt = 0: *Target position* not reached Halt = 1: Drive decelerates
	1	Halt = 0: *Target position* reached Halt = 1: Velocity of drive is 0
Set-point acknowledge	0	Trajectory generator will accept a new set-point
	1	Trajectory generator will not accept a new set-point.
Following error	0	No following error
	1	Following error

7.2. Position Profile Mode Objects

7.2.1. Object 607Ah: Target position

The *target position* is the position that the drive should move to in position profile mode using the current settings of motion control parameters such as velocity, acceleration, and *motion profile type* etc. It is given in position units.

Depending on the drive family, the position units can be:

- User defined position units for the MotionChip III family. The value is converted to position increments using the *position factor* (see **Chapter 5 Factor group**).
- Internal units (increments) for the MotionChip II family. No factor is applied.

If Control Word bit 6 = 0 (e.g. absolute positioning), represents the position to reach.

If Control Word bit 6 = 1 (e.g. relative positioning), represents the position displacement to do. When Control Word bit 14 = 0, the new position to reach is computed as: motor actual position (6064h) + displacement. When Control Word bit 14 = 1, the new position to reach is computed as: actual demand position (6062_h) + displacement.

Object description:

Index	$607A_h$
Name	Target position
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	Yes
Value range	-2^{31} … $2^{31}-1$
Default value	No

7.2.2. Object 6081h: Profile velocity

In a position profile, it represents the maximum speed to reach at the end of the acceleration ramp. The *profile velocity* is given in speed units.

Depending on the drive family, the speed units can be:

- User defined speed units for the MotionChip III family. The value is converted to internal units using the *velocity encoder factor* (see **Chapter 5 Factor group**).
- Internal units for the MotionChip II family. No factor is applied.

  65536 IU = 1 encoder increment / sample.

Object description:

Index	6081_h
Name	Profile velocity
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	-

7.2.3. Object 6083h: Profile acceleration

In position or speed profiles, represents the acceleration and deceleration rates used to change the speed between 2 levels. The same rate is used when *Quick Stop* or *Disable Operation* commands are received. The *profile acceleration* is given in acceleration units.

Depending on the drive family, the acceleration units can be:

- User defined acceleration units for the MotionChip III family. The value is converted to internal units using the *acceleration factor* (see **Chapter 5 Factor group**).
- Internal units for the MotionChip II family. No factor is applied.

  65536 IU = 1 encoder increment / sample^2.

Object description:

Index	6083_h
Name	Profile acceleration
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Value range	$0..(2^{32}-1)$
Default value	-

7.2.4. Object 6085h: Quick stop deceleration

The *quick stop deceleration* is the deceleration used to stop the motor if the *Quick Stop* command is received. The *quick stop deceleration* is given in acceleration units.

Object description:

Index	6085_h
Name	Quick stop deceleration
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW

PDO mapping	Possible
Value range	0..(2^{32}-1)
Default value	-

7.2.5. Object 2023h: Jerk time

In this object you can set the time to use for S-curve profile (jerk-limited ramp set in Object 6086h – Motion Profile Type)

Object description:

Index	2023_h
Name	Jerk time
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Value range	0 … 65535
Default value	-

7.2.6. Object 6086h: Motion profile type

Object description:

Index	6086_h
Name	Motion profile type
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RW
PDO mapping	Possible
Value range	INTEGER16
Default value	0

Data description:

Profile code	**Profile type**
-32768 … -1	Manufacturer specific (reserved)
0	Linear ramp (trapezoidal profile)
1,2	Reserved
3	Jerk-limited ramp (S-curve)
4 … 32767	Reserved

7.2.7. Object 6062h: Position demand value

This object represents the output of the trajectory generation. The *position demand values* is given in position units.

Object description:

Index	6062_h
Name	Position demand value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Value range	-2^{31} … $2^{31}-1$
Default value	-

7.2.8. Object 6063h: Position actual value* [32]

This object represents the actual value of the position measurement device in increments. It can be used as an alternative to *position actual value* (6064h) in order to save computation time.

Object description:

Index	6063_h
Name	Position actual value*
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Units	increments
Value range	-2^{31} … $2^{31}-1$
Default value	-

7.2.9. Object 6064h: Position actual value

This object represents the actual value of the position measurement device. The *position actual value* is given in position units.

Object description:

Index	6064_h

[32] Only available for MotionChip III family

Name	Position actual value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Yes
Value range	-2^{31} … $2^{31}-1$
Default value	-

7.2.10. Object 6065h: Following error window

This object defines a range of tolerated position values symmetrically to the *position demand value*, expressed in position units. If the *position actual value* is above the *following error window* for a period larger than the one defined in *following error time out*, a following error occurs. If the value of the *following error window* is $2^{32}-1$, the following control is switched off.

The maximum value allowed for the *following error window* parameter, expressed in increments, is 32767. When this object is written with a higher corresponding value, the *following error window* parameter will be set to its maximum value (32767)

Object description:

Index	6065_h
Name	Following error window
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	-

7.2.11. Object 6066h: Following error time out

See 6065h, *following error window*.

Object description:

Index	6066_h
Name	Following error time out
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Units	TU
Value range	0 … 65535
Default value	-

7.2.12. Object 6067h: Position window

The *position window* defines a symmetrical range of accepted positions relative to the *target position*. If the *position actual value* is within the *position window* for a time period defined inside the *position window time* object, this *target position* is regarded as reached. The *position window* is given in position units. If the value of the *position window* is 2^{32}-1, the position window control is switched off.

The maximum value allowed for the *position window* parameter, expressed in increments, is 32767.

Object description:

Index	6067_h
Name	Position window
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED32
Default value	-

7.2.13. Object 6068h: Position window time

See description of object 6067h, *position window*.

Object description:

Index	6068_h
Name	Position window time
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Units	TU

Value range	0 … 65535
Default value	-

7.2.14. Object 60F4h: Following error actual value[33]

This object represents the actual value of the following error, given in position units.

Object description:

Index	$60F4_h$
Name	Following error actual value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Value range	INTEGER32
Default value	-

7.2.15. Object 60FCh: Position demand value* [34]

This output of the trajectory generator in profile position mode is an internal value using increments as unit. It can be used as an alternative to *position demand value* (6062h) in order to save computation time.

Object description:

Index	$60FC_h$
Name	Position actual value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Units	Increments
Value range	-2^{31} … $2^{31}-1$
Default value	-

[33] Only available for the MotionChip III family

[34] Only available for the MotionChip III family

7.2.16. Object 2022h: Control effort

This object can be used to visualize the control effort of the drive (the reference for the current controller). It is available in internal units.

Object description:

Index	2022_h
Name	Control effort
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RO
PDO mapping	Possible
Value range	INTEGER16
Default value	-

Example: Execute an absolute trapezoidal profile with limited speed. First perform 4 rotations, wait motion complete and then set the target position of 16 rotations.

1. **Start remote node**. Send a NMT message to start the node id 6.

Send the following message:

COB-ID	**0**
Data	**01 06**

2. **Ready to switch on.** Change the node state from *Switch on disabled* to *Ready to switch on* by sending the shutdown command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**06 00**

3. **Switch on.** Change the node state from *Ready to switch on* to *Switch on* by sending the switch on command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**07 00**

4. **Enable operation.** Change the node state from *Switch on* to *Operation enable* by sending the enable operation command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**0F 00**

5. **Mode of operation.** Select position mode.

Send the following message (SDO access to object 6060_h, 8-bit value 1_h):

COB-ID	**606**
Data	**2F 60 60 00 01 00 00 00**

6. **Target position.** Set the target position to 4 rotations. By using and 500 lines incremental encoder the corresponding value of object $607A_h$ expressed in encoder counts is $1F40_h$.

Send the following message (SDO access to object $607A_h$ 32-bit value $00001F40_h$):

COB-ID	**606**
Data	**23 7A 60 00 40 1F 00 00**

7. **Target speed.** Set the target speed normally attained at the end of acceleration ramp to 500 rpm. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object 6081_h expressed in encoder counts per sample is $10AAAC_h$.

Send the following message (SDO access to object 6081_h, 32-bit value $0010AAAC_h$):

COB-ID	**606**
Data	**23 81 60 00 AC AA 10 00**

8. **Start the profile.**

Send the following message

COB-ID	**206**
Data	**1F 00**

9. **Wait movement to finish.**

10. **Reset the set point.**

Send the following message

COB-ID	**206**
Data	**0F 00**

11. **Target position.** Set the target position to 16 rotations. By using and 500 lines incremental encoder the corresponding value of object $607A_h$ expressed in encoder counts is $7D00_h$.

Send the following message (SDO access to object $607A_h$ 32-bit value $00007D00_h$):

COB-ID	**606**
Data	**23 7A 60 00 00 7D 00 00**

12. **Start the profile.**

Send the following message

COB-ID	**206**
Data	**1F 00**

13. **Wait movement to finish.**

14. **Check the value of motor actual position.**

Send the following message (SDO access to object 6064_h):

COB-ID	**606**
Data	**40 64 60 00 00 00 00 00**

15. **Check the value of position demand value.**

Send the following message (SDO access to object 6062_h):

COB-ID	**606**
Data	**40 62 60 00 00 00 00 00**

At the end of movement the motor position actual value should be equal with position demand value (plus or minus few encoder counts depending on your position tuning) and the motor should rotate 16 times.

Example: Execute an absolute Jerk-limited ramp profile.

1. **Start remote node**. Send a NMT message to start the node id 6.

Send the following message:

COB-ID	**0**
Data	**01 06**

2. **Ready to switch on.** Change the node state from *Switch on disabled* to *Ready to switch on* by sending the shutdown command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**06 00**

3. **Switch on.** Change the node state from *Ready to switch on* to *Switch on* by sending the switch on command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**07 00**

4. **Enable operation.** Change the node state from *Switch on* to *Operation enable* by sending the enable operation command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**0F 00**

5. **Mode of operation.** Select position mode.

Send the following message (SDO access to object 6060_h, 8-bit value 1_h):

COB-ID	**606**
Data	**2F 60 60 00 01 00 00 00**

6. **Motion profile type.** Select Jerk-limited ramp.

Send the following message (SDO access to object 6086_h, 16-bit value 3_h):

COB-ID	**606**
Data	**2B 86 60 00 03 00 00 00**

7. **Target position.** Set the target position to 5 rotations. By using and 500 lines incremental encoder the corresponding value of object $607A_h$ expressed in encoder counts is 2710_h.

Send the following message (SDO access to object $607A_h$ 32-bit value 00002710_h):

COB-ID	**606**
Data	**23 7A 60 00 10 27 00 00**

8. **Target speed.** Set the target speed to 150 rpm. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object 6081_h expressed in encoder counts per sample is 00050000_h.

Send the following message (SDO access to object 6081_h, 32-bit value 00050000_h):

COB-ID	**606**
Data	**23 81 60 00 00 00 05 00**

9. **Jerk time.** Set the time to use for Jerk-limited ramp. For more information related to this parameter, see the ESM help

Send the following message (SDO access to object 2023_h, 16-bit value $13B_h$):

COB-ID	**606**
Data	**2B 23 20 00 3B 01 00 00**

10. **Start the profile.**

Send the following message

COB-ID	**206**
Data	**1F 00**

11. **Wait movement to finish.**

12. **Check the value of motor actual position.**

Send the following message (SDO access to object 6064_h):

COB-ID	**606**
Data	**40 64 60 00 00 00 00 00**

13. **Check the value of position demand value.**

Send the following message (SDO access to object 6062_h):

COB-ID	**606**
Data	**40 62 60 00 00 00 00 00**

At the end of movement the motor position actual value should be equal with position demand value (plus or minus few encoder counts depending on your position tuning).

8. Interpolated Position Mode

8.1. Overview

The interpolated Position Mode is used to control multiple coordinated axles or a single axle with the need for time-interpolation of set-point data. The Interpolated Position Mode can use the time synchronization mechanism for a time coordination of the related drive units, based on the SYNC and the High Resolution Time Stamp messages (see object 1013 for details).

The Interpolated Position Mode allows a host controller to transmit a stream of interpolation data to a drive unit. The interpolation data is better sent in bursts because the drive supports an input buffer. The buffer size is the number of *interpolation data records* that may be sent to the drive to fill the input buffer.

The interpolation algorithm can be defined in the *interpolation sub mode select*. Linear (PT – Position Time) interpolation is the default interpolation method.

8.1.1. Internal States

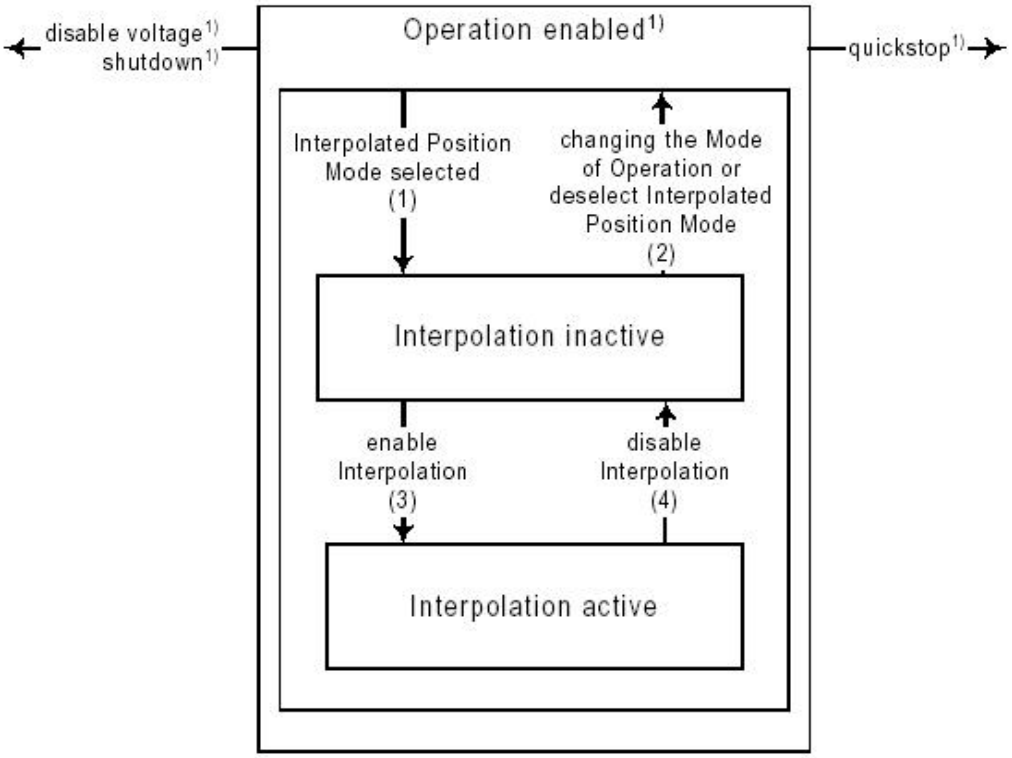


Figure 8.1 *Internal States for the Interpolated Position Mode*

[1] See state machine

***Interpolation inactive*:** This state is entered when the device is in state Operation enabled and the Interpolated Position Mode is selected. The drive will accept input data and will buffer it for interpolation calculations, but it does not move the motor.

***Interpolation active*:** This state is entered when a device is in state Operation enabled and the Interpolation Position Mode is selected and enabled. The drive will accept input data and will move the motor.

State Transitions of the Internal States

State Transition 1: NO IP-MODE SELECTED => IP-MODE INACTIVE

Event: Select ip-mode with *modes of operations* while inside Operation enable

State Transition 2: IP-MODE INACTIVE => NO IP-MODE SELECTED

Event: Select any other mode while inside Operation enable

State Transition 3: IP-MODE INACTIVE => IP-MODE ACTIVE

Event: Set bit *enable ip mode* (bit4) of the *control word* while in ip-mode and Operation enable

State Transition 4: IP-MODE ACTIVE => IP-MODE INACTIVE

Event: Reset bit *enable ip mode* (bit4) of the *control word* while in ip-mode and Operation enable

Control word in interpolated position mode

MSB LSB

See 6040h	Stop option	See 6040h	Halt	See 6040h	Abs / rel	Reserved	Enable ip mode	See 6040h
15 12	11	10 9	8	7	6	5	4	3 0

Table 8.1 *Control Word bits description for Interpolated Position Mode*

Name	Value	Description
Enable ip mode	0	Interpolated position mode inactive
	1	Interpolated position mode active
Abs / rel	0	Set position is an absolute value
	1	Set position is a relative value
Halt	0	Execute the instruction of bit 4
	1	Stop drive with (*profile acceleration*)
Stop option	0	On transition to inactive mode, stop drive immediately using *profile acceleration*
	1	On transition to inactive mode, stop drive after finishing the current segment.

Status word in interpolated position mode

MSB							LSB
See 6041h		Reserved	ip mode active	See 6041h	Target reached	See 6041h	
15	14	13	12	11	10	9	0

Table 8.2 *Status Word bits description for Interpolated Position Mode*

Name	Value	Description
Target reached	0	Halt = 0: Final position not reached Halt = 1: Drive decelerates
	1	Halt = 0: Final position reached Halt = 1: Velocity of drive is 0
ip mode active	0	Interpolated position mode inactive
	1	Interpolated position mode active

8.2. Interpolated Position Objects

8.2.1. Object 60C0h: Interpolation sub mode select

In the Interpolated Position Mode the drive supports two interpolation modes: PT (Position – Time) linear interpolation and PVT (Position – Velocity – Time) cubic interpolation. The interpolation mode is selected with Interpolation sub-mode select object. The sub-mode can be changed only when the drive is in Interpolation inactive state.

Each change of the interpolation mode will trigger the reset of the buffer associated with the interpolated position mode (because the physical memory available is the same for both the sub-modes, size of each data record is different).

Object description:

Index	$60C0_h$
Name	Interpolation sub mode select
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RW
PDO mapping	Possible
Value range	-2^{15} … $2^{15}-1$
Default value	0

Data description:

Profile code	Profile type
-32768 … -2	Manufacturer specific (reserved)
-1	PVT (Position – Velocity – Time)

	cubic interpolation
0	PT (Position – Time) Linear Interpolation
+1…+32767	Reserved

8.2.2. Object 60C1h: Interpolation data record

The **Interpolation Data Record** contains the data words that are necessary to perform the interpolation algorithm. The number of data words in the record is defined by the *interpolation data configuration.*

Object description:

Index	$60C1_h$
Name	Interpolation data record
Object code	ARRAY
Number of elements	2
Data Type	Interpolated Mode dependent

Entry description

Sub-index	01_h
Description	X1: the first parameter of ip function
Access	RW
PDO mapping	Possible
Value range	Interpolated Mode dependent
Default value	-

Sub-index	02_h
Description	X2: the second parameter of ip function
Access	RW
PDO mapping	Possible
Value range	Interpolated Mode dependent
Default value	-

Description of the sub-indexes:

X1 and X2 form a 64-bit data structure as defined below:

a) For PVT (Position – Velocity – Time) cubic interpolation:

There are 4 parameters in this mode:

Position – a 24-bit long integer value representing the target position (relative or absolute). Unit - position increments.

Velocity – a 24-bit fixed value representing the end point velocity (16 MSB integer part and 8 LSB fractional part). Unit - increments / sampling

Time – a 9-bit unsigned integer value representing the time of a PVT segment. Unit - position / speed loop samplings.

Counter – a 7-bit unsigned integer value representing an integrity counter. It can be used in order to have a feedback of the last point sent to the drive and detect errors in transmission.

In the example below Position[7…0] represents bits 0..7 of the position value.

Byte 0	Position [7...0]	
Byte 1	Position [15...8]	
Byte 2	Velocity [15...8]	
Byte 3	Position [23...16]	
Byte 4	Velocity [23...16]	
Byte 5	Velocity [31...24]	
Byte 6	Time [7...0]	
Byte 7	Counter[6…0]	Time[8]

b) For PT (Position –Time) linear interpolation:

There are 3 parameters in this mode:

Position – a 32-bit long integer value representing the target position (relative or absolute). Unit - position increments.

Time – a 16-bit unsigned integer value representing the time of a PT segment. Unit - position / speed loop samplings.

Counter – a 7-bit unsigned integer value representing an integrity counter. It can be used in order to have a feedback of the last point sent to the drive and detect errors in transmission.

In the example below Position[7…0] represents bits 0..7 of the position value.

Byte 0	Position [7...0]	
Byte 1	Position [15...8]	
Byte 2	Position [23...16]	
Byte 3	Position [31...24]	
Byte 4	Time [7...0]	
Byte 5	Time [15...8]	
Byte 6	Reserved	
Byte 7	Counter[6…0]	Reserved

8.2.3. Object 2072h: Interpolated position mode status

The object provides additional status information for the interpolated position mode.

Object description:

Index	2072_h
Name	Interpolated position mode status
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	Possible
Value range	UNSIGNED16
Default value	-

Table 8.3 *Interpolated position mode status bit description*

Bit	Value	Description
15	0	Buffer is not empty
	1	Buffer is empty – there is no point in the buffer.
14	0	Buffer is not low
	1	Buffer is low – the number of points from the buffer is equal or less than the low limit set using object 2074_h.
13	0	Buffer is not full
	1	Buffer is full – the number of points in the buffer is equal with the buffer dimension.
12	0	No integrity counter error
	1	Integrity counter error. If integrity counter error checking is enabled and the integrity counter sent by the master does not match the integrity counter of the drive.
11	0	Valid only for PVT (cubic interpolation): Drive has maintained interpolated position mode after a buffer empty condition (the velocity of the last point was 0).
	1	Valid only for PVT (cubic interpolation): Drive has performed a quick stop after a buffer empty condition because the velocity of the last point was different from 0
10 … 7		Reserved
6 … 0		Current integrity counter value

8.2.4. Object 2073h: Interpolated position buffer length

Through **Interpolated position buffer length** object you can change the default buffer length. When writing in this object, the buffer will automatically reset its contents and then re-initialize with the new length. The length of the buffer is the maximum number of interpolation data that can be queued, and does not mean the number of data locations physically available.

Remark: *It is NOT allowed to write a "0" into this object.*

Object description:

Index	2073_h
Name	Interpolated position buffer length
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Value range	UNSIGNED16
Default value	7

8.2.5. Object 2074h: Interpolated position buffer configuration

Through this object you can control more in detail the behavior of the buffer.

Object description:

Index	2074_h
Name	Interpolated position buffer configuration
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Value range	UNSIGNED16
Default value	-

Table 8.4 *Interpolated position buffer configuration*

Bit	Value	Description
15	0	Nothing
	1	Clear buffer and reinitialize buffer internal variables
14	0	Enable the integrity counter error checking
	1	Disable the integrity counter error checking

13	0	No change in the integral integrity counter
	1	Change internal integrity counter with the value specified in bits 0 to 6
12	0	If absolute positioning is set (bit 6 of *control word* is 0), the initial position is read from object 2079_h. It is used to compute the distance to move up to the first PVT point.
	1	If absolute positioning is set (bit 6 of *control word* is 0), the initial position is the current *position demand value*. It is used to compute the distance to move up to the first PVT point.
11 ... 8		New parameter for buffer low signaling. When the number of entries in the buffer is equal or less than buffer low value, bit 14 of object 2072_h will set.
7	0	No change in the buffer low parameter
	1	Change the buffer low parameter with the value specified in bits 8 to 11
6 … 0		New integrity counter value

8.2.6. Object 2079h: Interpolated position initial position

Through this object you can set an initial position for absolute positioning in order to be used to compute the distance to move up to the first point. It is given in position units.

Object description:

Index	2079_h
Name	Interpolated position initial position
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	No
Value range	INTEGER32
Default value	0

Example: Execute a PVT movement. Because this is a demo for a single axis the synchronization mechanism is not used here.

1. **Start remote node**. Send a NMT message to start the node id 6.

Send the following message:

COB-ID	**0**
Data	**01 06**

2. **Ready to switch on.** Change the node state from *Switch on disabled* to *Ready to switch on* by sending the shutdown command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**06 00**

3. **Switch on.** Change the node state from *Ready to switch on* to *Switch on* by sending the switch on command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**07 00**

4. **Enable operation.** Change the node state from *Switch on* to *Operation enable* by sending the enable operation command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**0F 00**

5. **Disable the RPDO2**. Write zero in object 1601_h sub-index 0, this will disable the PDO.

Send the following message (SDO access to object 1601_h sub-index 0, 8-bit value 0):

COB-ID	**606**
Data	**2F 01 16 00 00 00 00 00**

6. **Map the new objects**.
   a. Write in object 1601_h sub-index 1 the description of the interpolated data record sub-index 1:

Send the following message (SDO access to object 1601_h sub-index 1, 32-bit value $60C10120_h$):

COB-ID	**606**
Data	**23 01 16 01 20 01 C1 60**

   b. Write in object 1601_h sub-index 2 the description of the interpolated data record sub-index 2:

Send the following message (SDO access to object 1601_h sub-index 2, 32-bit value $60C10220_h$):

COB-ID	**606**
Data	**23 01 16 02 20 02 C1 60**

7. **Enable the RPDO2**. Set the object 1601_h sub-index 0 with the value 2.

Send the following message (SDO access to object 1601_h sub-index 0, 8-bit value 2):

COB-ID	**606**
Data	**2F 01 16 00 02 00 00 00**

8. **Mode of operation**. Select interpolation position mode.

Send the following message (SDO access to object 6060_h, 8-bit value 7_h):

COB-ID	**606**
Data	**2F 60 60 00 07 00 00 00**

9. **Interpolation sub mode select**. Select interpolation position mode.

Send the following message (SDO access to object $60C0_h$, 16-bit value $FFFF_h$):

COB-ID	**606**
Data	**2E C0 60 00 FF FF 00 00**

10. **Interpolated position buffer length**.

Send the following message (SDO access to object 2074_h, 16-bit value C_h):

COB-ID	**606**
Data	**2B 74 20 00 00 0C 00 00**

11. **Interpolated position initial position**. Set the initial position to 0.5 rotations. By using and 500 lines incremental encoder the corresponding value of object 2079_h expressed in encoder counts is $3E8_h$

Send the following message (SDO access to object 2079_h, 32-bit value $3E8_h$):

COB-ID	**606**
Data	**23 79 20 00 E8 03 00 00**

12. **Send the 1st PVT point**.

Send the following message:

COB-ID	**306**
Data	**10 27 55 00 03 00 F4 01**

13. **Send the 2nt PVT point**.

Send the following message:

COB-ID	**306**
Data	**10 27 55 00 03 00 F4 01**

14. **Send the 3nt PVT point**.

Send the following message:

COB-ID	**306**
Data	**10 27 55 00 03 00 F4 01**

15. **Send the last PVT point**.

Send the following message:

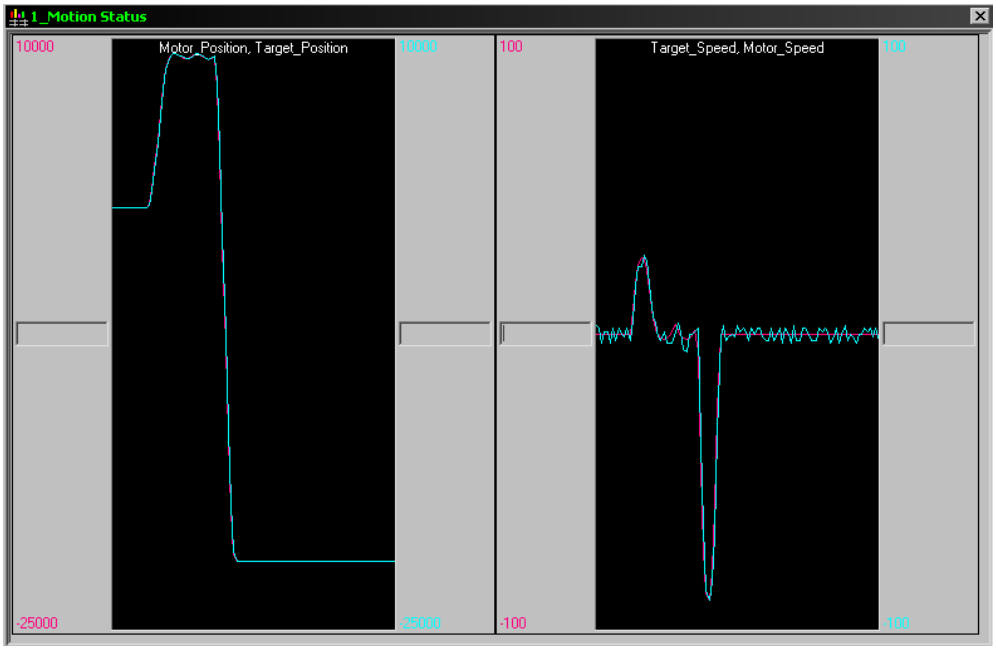
COB-ID	**306**
Data	**E0 B1 00 FF 00 00 F4 01**

16. **Start a relative motion**.

Send the following message:

COB-ID	**206**
Data	**5F 00**

After the sequences before are executed the motor should execute a movement as the one shown below.



9. Velocity Profile Mode

9.1. Overview

In the Velocity Profile Mode the drive performs speed control. The built-in reference generator computes a speed profile with a trapezoidal shape, due to a limited acceleration. The **Target Velocity** object (index $60FF_h$) specifies the jog speed (speed sign specifies the direction) and the **Profile Acceleration** object (index 6083_h) the acceleration/deceleration rate. While the mode is active, any change of the Target Velocity object by the CANopen master will update the drive's demand velocity enabling you to change on the fly the slew speed and/or the acceleration/deceleration rate. The motion will continue until the **Halt** bit from the Control Word is set. An alternate way to stop the motion is to set the jog speed to zero.

While the mode is active (profile velocity mode is selected in *modes of operation*), every time a write access is performed inside the object *target velocity*, the demand velocity of the drive is updated.

Control word in profile velocity mode

MSB **LSB**

See 6040h	Halt	See 6040h	reserved	See 6040h
15 9	8	7	6 4	3 0

Table 9.1 *Control Word bits for Velocity Profile Mode*

Name	Value	Description
Halt	0	Execute the motion
	1	Stop drive with *profile acceleration*

Status word in profile velocity mode

MSB **LSB**

See 6041h	Max slippage error	Speed	See 6041h	Target reached	See 6041h
15 14	13	12	11	10	9 0

Table 9.2 *Status word bits for Velocity Profile*

Name	Value	Description
Target reached	0	Halt = 0: *Target velocity* not (yet) reached Halt = 1: Drive decelerates
	1	Halt = 0: *Target velocity* reached Halt = 1: Velocity of drive is 0
Speed	0	Speed is not equal to 0

	1	Speed is equal to 0
Max slippage error	0	Maximum slippage not reached
	1	Maximum slippage reached

***Remark**: In case of the MotionChip III family, in order to set / reset bit 12 (speed), the object 606F$_h$, velocity threshold is used. If the actual velocity of the drive / motor is below the velocity threshold, then bit 12 will be set, else it will be reset.*

9.2. Velocity Mode Objects

9.2.1. Object 6069h: Velocity sensor actual value

This object describes the value read from the velocity encoder in increments.

Depending on the drive family, the velocity units can be:

- User defined speed units for the MotionChip III family. The value is converted to internal units using the *velocity factor*
- Internal units for the MotionChip II family. No factor is applied.

  65536 IU = 1 encoder increment / sample.

Object description:

Index	6069$_h$
Name	Velocity sensor actual value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Value range	INTEGER32
Default value	-

9.2.2. Object 606Bh: Velocity demand value

This object provides the output of the trajectory generator and is provided as an input for the velocity controller. It is given in velocity units.

Object description:

Index	606B$_h$
Name	Velocity demand value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Value range	INTEGER32
Default value	-

9.2.3. Object 606Ch: Velocity actual value

The *velocity actual value* is given in velocity units and is read from the velocity sensor.

Object description:

Index	$606C_h$
Name	Velocity actual value
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Yes
Value range	INTEGER32
Default value	-

9.2.4. Object 606Fh: Velocity threshold[35]

The *velocity threshold* is given in velocity units and it represents the threshold for velocity at which it is regarded as zero velocity. Based on its value, bit 12 of *status word* (speed) will be set or reset.

Object description:

Index	$606F_h$
Name	Velocity threshold
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED16
Default value	-

[35] Only available for MotionChip III family

9.2.5. Object 60FFh: Target velocity

The *target velocity* is the input for the trajectory generator and the value is given in velocity units.

Object description:

Index	$60FF_h$
Name	Target velocity
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	possible
Value range	INTEGER32
Default value	-

9.2.6. Object 60F8h: Max slippage[36]

The *max slippage* monitors whether the maximal slippage has actually been reached. The value is given in velocity units. When the *max slippage* has been reached, the corresponding bit 13 *max slippage error* in the *status word* is set.

Object description:

Index	$60F8_h$
Name	Max slippage
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	possible
Value range	INTEGER32
Default value	-

9.2.7. Object 2005h: Max slippage time out[37]

Time interval for *max slippage*.

Object description:

Index	2005_h
Name	Max slippage time out

[36] Only available for MotionChip III family. In case of MotionChip II family, the corresponding parameter can be set via the set-up or via indirect memory accesses.

[37] Same as note 31.

Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Value range	UNSIGNED16
Default value	-

Example: Execute a speed control with 600 rpm target speed.

1. **Start remote node**. Send a NMT message to start the node id 6.

Send the following message:

COB-ID	**0**
Data	**01 06**

2. **Ready to switch on.** Change the node state from *Switch on disabled* to *Ready to switch on* by sending the shutdown command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**06 00**

3. **Switch on.** Change the node state from *Ready to switch on* to *Switch on* by sending the switch on command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**07 00**

4. **Enable operation.** Change the node state from *Switch on* to *Operation enable* by sending the enable operation command via control word associated PDO.

Send the following message:

COB-ID	**206**
Data	**0F 00**

5. **Mode of operation.** Select speed mode.

Send the following message (SDO access to object 6060_h, 8-bit value 3_h):

COB-ID	**606**
Data	**2F 60 60 00 03 00 00 00**

6. **Target velocity.** Set the target velocity to 600 rpm. By using and 500 lines incremental encoder and 1ms sample rate for position/speed control the corresponding value of object $60FF_h$ expressed in encoder counts per sample is 140000_h.

Send the following message (SDO access to object $60FF_h$ 32-bit value 00140000_h):

COB-ID	**606**
Data	**23 FF 60 00 00 00 14 00**

7. **Check the motor actual speed.** It should rotate with 600 rpm.

10. Electronic Gearing Position (EGEAR) Mode[38]

10.1. Overview

In Electronic Gearing Position Mode the drive follows the position of an electronic gearing master with a programmable gear ratio.

The electronic gearing slave can get the position information from the electronic camming master in two ways:

Via CANbus communication channel, from another IDM680 drive set as electronic gearing master with object **Master Settings** (index 2010_h). The position is sent using TechnoCAN, an extension of the CANopen protocol, developed by Technosoft.

Via an external digital reference of type pulse & direction or quadrature encoder. Both options have dedicated inputs. The pulse & direction signals are usually provided by an indexer and must be connected to the pulse & direction inputs of the drive/motor. The quadrature encoder signals are usually provided by an encoder on the master and must be connected to the 2nd encoder inputs.

The reference type, i.e. the selection between the online reference received via communication channel and the digital reference read from dedicated inputs is done with object **External Reference Type** (index $201D_h$). The source of the digital reference (pulse & direction or second encoder inputs) is set during drive commissioning.

The drive set as slave in electronic gearing mode performs a position control. At each slow loop sampling period, the slave computes the master position increment and multiplies it with its programmed gear ratio. The result is the slave position reference increment, which added to the previous slave position reference gives the new slave position reference.

Remark: *The slave executes a relative move, which starts from its actual position*

The gear ratio is specified via **EGEAR multiplication factor** object (index 2013_h). EGEAR ratio numerator (sub-index 1) is a signed integer, while EGEAR ratio denominator (sub-index 2) is an unsigned integer. The EGEAR ratio numerator sign indicates the direction of movement: positive – same as the master, negative – reversed to the master. The result of the division between EGEAR ratio numerator and EGEAR ratio denominator is used to compute the slave reference increment.

The **Master Resolution** object (index 2012_h) provides the master resolution, which is needed to compute correctly the master position and speed (i.e. the position increment). If master position is not cyclic (i.e. the resolution is equal with the whole 32-bit range of position), set master resolution to 0x80000001.

You can smooth the slave coupling with the master, by limiting the maximum acceleration of the slave drive. This is particularly useful when the slave has to couple with a master running at high speed, in order to minimize the shocks in the slave. The feature is activated by setting ControlWord.5=1 and the maximum acceleration value in **Profile Acceleration** object (index 6083_h).

[38] Only available for MotionChip III family. Available also on request for MotionChip II family

Control word in electronic gearing position mode (slave axis)

MSB **LSB**

See 6040h	Halt	See 6040h	Reserved	Activate Acceleration Limitation	Enable Electronic Gearing Mode	See 6040h
15 9	8	7	6	5	4	3 0

Table 10.1 *Control Word bits for Electronic Gearing Position Mode*

Name	Value	Description
Enable Electronic Gearing Mode	0	Electronic gearing mode inactive
	1	Electronic gearing mode active
Activate Acceleration Limitation	0	Do not limit acceleration when entering electronic gear mode
	1	Limit acceleration when entering electronic gear mode to the value set in *profile acceleration* (object 6083_h)
Halt	0	Execute the instruction of bit 4
	1	Stop drive with *profile acceleration*

Status word in electronic gearing position mode

MSB **LSB**

See 6041h	Following error	Reserved	See 6041h	Target reached	See 6041h
15 14	13	12	11	10	9 0

Table 10.2 *Status Word bits for Electronic Gearing Position Mode*

Name	Value	Description
Target reached	0	Halt = 0: Always 0 Halt = 1: Drive decelerates
	1	Halt = 0: Always 0 Halt = 1: Velocity of drive is 0
Following error	0	No following error
	1	Following error occurred

10.2. Gearing Position Mode Objects

10.2.1. Object 2010h: Master settings

This object contains key settings for the master of EGEAR / ECAM mode. A master in EGEAR / ECAM mode is any IDM680 drive that controls a motor (irrespective of the control mode) and that will be designated to send the information about its position (demanded position or actual position) via communication to one or more slaves (programmed accordingly).

This object also allows setting the address of the slave to which the master will send its position, or, if there are more slaves to receive simultaneously the position from the master, the Group ID of these slaves.

Object description:

Index	2010_h
Name	Master settings
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	Possible
Units	-
Value range	0 … 65535
Default value	0

Table 10.3 *Master Settings bits description*

Bit	Value	Description
15	0	Master is not active – the master drive does not send any position values
	1	Master is active – the master drive starts sending its position to the slave axis
14 … 10	0	Reserved
9	0	The master sends its feedback (the position actual value)
	1	The master sends the demand position
8	0	Address is an axis ID
	1	Address is a group ID
7 … 0	x	Address of the slave drive(s)

10.2.2. Object 2012h: Master resolution

This object is used in order to set the master resolution in increments per revolution. This object is valid for the slave axis.

Object description:

Index	2012_h
Name	Master resolution
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW

PDO mapping	Possible
Units	Increments
Value range	0 … 2^{31}-1
Default value	80000001h (full range)

10.2.3. Object 2013h: EGEAR multiplication factor

In digital external mode, this object sets the gear ratio, or gear multiplication factor for the slaves. The sign indicates the direction of movement: positive – same as the master, negative – reversed to the master. The slave demand position is computed as the master position increment multiplied by the gear multiplication factor.

Example: if the gear ratio is Slave/Master = 1/3, the following values must be set: 1 in EGEAR ratio numerator (sub-index 1) and 3 in EGEAR ratio denominator (sub-index 2)

Object description:

Index	2013_h
Name	EGEAR multiplication factor
Object code	RECORD
Number of elements	2

Entry description:

Sub-index	1
Description	EGEAR ratio numerator (slave)
Object code	VAR
Data type	INTEGER16
Access	RW
PDO mapping	Possible
Value range	-32768 … 32767
Default value	1

Sub-index	2
Description	EGEAR ratio denominator (master)
Object code	VAR
Data type	UNSIGNED16
Access	RW
PDO mapping	Possible
Value range	0 … 65535
Default value	1

10.2.4. Object 2017h: Master actual position

The actual position of the master can be monitored through this object, regardless of the way the master actual position is delivered to the drive (on-line through a communication channel or from the digital inputs of the drive). The units are increments.

Object description:

Index	2017_h
Name	Master actual position
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RO
PDO mapping	Possible
Value range	-2^{31} … $2^{31}-1$
Default value	0

10.2.5. Object 2018h: Master actual speed

This object is used to inform the user of the actual value of the speed of the master, regardless of the way the master actual position is delivered to the drive (on-line through a communication channel or from the digital inputs of the drive). The units are increments / sampling.

Object description:

Index	2018_h
Name	Master actual speed
Object code	VAR
Data type	INTEGER16

Entry description:

Access	RO
PDO mapping	Possible
Value range	-32768 … 32767
Default value	0

10.2.6. Object 201Dh: External Reference Type

This object is used to set the type of external reference for use with electronic gearing position, electronic camming position, position external, speed external and torque external modes.

Object description:

Index	$201D_h$
Name	External Reference Type
Object code	VAR

Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Value range	UNSIGNED16
Default value	-

Table 10.4 *External Reference Type bit description*

Value	Description
0	Reserved
1	On-line. In case of External Reference Position / Speed / Torque Modes, select this option in order to read the reference from the object 201C, *External Online Reference* In case of Electronic Gearing and Camming Position Modes, select this option in order to read the master position received from a master drive via communication
2	Analogue. In case of External Reference Position / Speed / Torque Modes, select this option in order to read the reference from the dedicated analogue input.
3	Digital. In case of External Reference Position Modes, select this option in order to read the reference from the dedicated digital inputs as set in the setup made using EasySetup / EasyMotion Studio (either 2nd encoder or pulse & direction) In case of Electronic Gearing and Camming Position Modes, select this option in order to read master position from the dedicated digital inputs as set in the setup made using EasySetup / EasyMotion Studio (either 2nd encoder or pulse & direction)
4 … 65535	Reserved

11. Electronic Camming Position (ECAM) Mode[39]

11.1. Overview

In Electronic Camming Position the drive executes a cam profile function of the position of an electronic camming master. The cam profile is defined by a cam table – a set of (X, Y) points, where X is cam table input i.e. the position of the electronic camming master and Y is the cam table output i.e. the corresponding slave position. Between the points the drive performs a linear interpolation.

The electronic camming slave can get the position information from the electronic camming master in two ways:

1. Via CANbus communication channel, from another IDM680 drive set as electronic camming master with object **Master Settings** (index 2010_h). The position is sent using TechnoCAN, an extension of the CANopen protocol, developed by Technosoft.
2. Via an external digital reference of type pulse & direction or quadrature encoder. Both options have dedicated inputs. The pulse & direction signals are usually provided by an indexer and must be connected to the pulse & direction inputs of the drive. The quadrature encoder signals are usually provided by an encoder on the master and must be connected to the 2nd encoder inputs.

The reference type, i.e. the selection between the online reference received via communication channel and the digital reference read from dedicated inputs is done with object **External Reference Type** (index $201D_h$). The source of the digital reference (pulse & direction or second encoder inputs) is set during drive commissioning.

The electronic camming position mode can be: **relative** (if ControlWord.6 = 0) or **absolute** (if ControlWord.6 = 1).

In the relative mode, the output of the cam table is added to the slave actual position. At each slow loop sampling period the slave computes a position increment **dY = Y – Yold**. This is the difference between the actual cam table output Y and the previous one Yold. The position increment dY is added to the old demand position to get a new demand position. The slave detects when the master position rolls over, from 360 degrees to 0 or vice-versa and automatically compensates in dY the difference between **Ymax** and **Ymin**. Therefore, in relative mode, you can continuously run the master in one direction and the slaves will execute the cam profile once at each 360 degrees with a glitch-free transition when the cam profile is restarted.

When electronic camming is activated in relative mode, the slave initializes **Yold** with the first cam output computed: **Yold = Y = f(X)**. The slave will keep its position until the master starts to move and then it will execute the remaining part of the cam. For example if the master moves from X to Xmax, the slave moves with Ymax – Y.

In the absolute mode, the output of the cam table Y is the demand position to reach.

Remark: *The absolute mode must be used with great care because it may generate abrupt variations on the slave demand position if:*

- Slave position is different from Y at entry in the camming mode

[39] Only available for MotionChip III family. Available also on request for MotionChip II family

- Master rolls over and Ymax < Ymin

In the absolute mode, you can introduce a maximum speed limit to protect against accidental sudden changes of the positions to reach. The feature is activated by setting ControlWord.5=1 and the maximum speed value in object **Profile Velocity** (index 6081_h).

Typically, the cam tables are first downloaded into the EEPROM memory of the drive by the CANopen master or with EasyMotion Studio. Then using the object **CAM table load address** (index 2019_h) they are copied in the RAM address set in object **CAM table run address** (index $201A_h$). It is possible to copy more than one cam table in the drive/motor RAM memory. When the ECAM mode is activated it uses the CAM table found at the RAM address contained in **CAM table run address**.

A CAM table can be shifted, stretched or compressed.

Control word in electronic camming position mode

MSB **LSB**

See 6040h	Halt	See 6040h	Abs / Rel	Activate Speed Limitation	Enable Electronic Camming Mode	See 6040h
15 9	8	7	6	5	4	3 0

Table 11.1 *Control word bits for electronic camming position mode*

Name	Value	Description
Enable Electronic Camming Mode	0	Electronic camming mode inactive
	1	Electronic camming mode active
Activate Speed Limitation	0	Do not limit speed when entering absolute electronic camming mode
	1	Limit speed when entering absolute electronic camming mode at the value set in *profile velocity* (ONLY for absolute mode)
Abs / Rel	0	Perform relative camming mode – when entering the camming mode, the slave will compute the cam table relative to the starting moment.
	1	Perform absolute camming mode – when entering the camming mode, the slave will go to the absolute position on the cam table
Halt	0	Execute the instruction of bit 4
	1	Stop drive with *profile acceleration*

Status word in electronic camming position mode

MSB **LSB**

See 6041h	Following error	Reserved	See 6041h	Target reached	See 6041h
15 14	13	12	11	10	9 0

Table 11.2 *Status word bits for electronic camming position mode*

Name	Value	Description
Target reached	0	Halt = 0: Always 0 Halt = 1: Drive decelerates

	1	Halt = 0: Always 0 Halt = 1: Velocity of drive is 0
Following error	0	No following error
	1	Following error occurred

11.2. Electronic Camming Position Mode Objects

11.2.1. Object 2019h: CAM table load address

This is the **load address** of the CAM table. The CAM table is stored in EEPROM memory of the drive starting from the load address. The initialization of the electronic camming mode requires the CAM table to be copied from the EEPROM memory to the RAM memory of the drive, starting from the **run address**, set in object $201A_h$, for faster processing. The copy is made every time object 2019_h is written by SDO access.

Remark: *The **CAM table run address** object must be set before writing the object **CAM table load address** to assure a proper copy operation from EEPROM to RAM memory.*

Object description:

Index	2019_h
Name	CAM table load address
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	UNSIGNED16
Default value	4500h

11.2.2. Object 201Ah: CAM table run address

This is the run address of the CAM table e.g. the RAM address starting from which the CAM table is copied into the RAM during initialization of the electronic camming mode. (see also 2019_h).

Object description:

Index	$201A_h$
Name	CAM table run address
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RW

PDO mapping	No
Units	-
Value range	UNSIGNED16
Default value	E500h

11.2.3. Object 201Bh: CAM offset

This object may be used to shift the master position in electronic camming mode. The position actually used as X input in the cam table is not the master actual position (2017_h) but (master actual position – CAM offset) computed as modulo of master resolution (2012_h) The CAM offset must be set before enabling the electronic camming mode. The *CAM offset* is expressed in increments.

Object description:

Index	$201B_h$
Name	CAM offset
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Value range	0 … 2^{32}-1
Default value	0

11.2.4. Object 206Bh: CAM: input scaling factor

You can use this scaling factor in order to achieve a scaling of the input values of a CAM table. Its default value of 00010000h corresponds to a scaling factor of 1.0.

Object description:

Index	$206B_h$
Name	CAM input scaling factor
Object code	VAR
Data type	FIXED32

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	FIXED32
Default value	00010000_h

11.2.5. Object 206Ch: CAM: output scaling factor

You can use this scaling factor in order to achieve a scaling of the output values of a CAM table. Its default value of 00010000h corresponds to a scaling factor of 1.0.

Object description:

Index	$206C_h$
Name	CAM output scaling factor
Object code	VAR
Data type	FIXED32

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	FIXED32
Default value	00010000_h

12. External Reference Position Mode[40]

12.1. Overview

In this operating mode the drive performs position control with the demand position read from the external reference provided by another device.

There are 3 types of external references:

- Analogue – read by the drive via a dedicated analogue input (12-bit resolution)
- Digital – computed by the drive from:
  - Pulse & direction signals
  - Quadrature signals like A, B signals of an incremental encoder
- Online – received online via the CAN bus communication channel from the CANopen master in object **External On-line Reference** (index $201C_h$)

The reference type is selected with object **External Reference Type** (index $201D_h$). When the external reference is digital, the option for the input signals: pulse & direction or quadrature encoder is established during drive commissioning.

In external reference position mode with analogue or online reference, you can limit the maximum speed at sudden changes of the position reference and thus to reduce the mechanical shocks. This feature is activated by setting ControlWord.6=1 and the maximum speed value in object **Profile Velocity** (index 6081_h).

Control word in external reference position mode

MSB LSB

See 6040h	Halt	See 6040h	Reserved	Activate Speed Limitation	Enable External Position Mode	See 6040h
15 9	8	7	6	5	4	3 0

Table 12.1 *Control Word bit description for External Reference Position mode*

Name	Value	Description
Enable External Position Mode	0	External position mode inactive
	1	External position mode active
Activate Speed Limitation	0	Do not limit speed on the inactive to active mode transition
	1	Limit speed when enabling the External Position mode
Halt	0	Execute the instruction of bit 4
	1	Stop drive with *profile acceleration*

[40] Only available for MotionChip III family. Available also on request for MotionChip II family

In order to correctly set an external reference position mode, you have to set the way the reference is received (either on-line, analogue or digital), using the object 201Dh, *External Reference Type*.

Status word in external reference position mode

MSB **LSB**

See 6041h	Following error	Reserved	See 6041h	Target reached	See 6041h
15 14	13	12	11	10	9 0

Table 12.2 *Status Word bit description for External Reference Position mode*

Name	Value	Description
Target reached	0	Halt = 0: Always 0 Halt = 1: Drive decelerates
	1	Halt = 0: Always 0 Halt = 1: Velocity of drive is 0
Following error	0	No following error
	1	Following error occurred

12.2. External Reference Position Mode Objects

12.2.1. Object 201Ch: External On-line Reference

This is used to set the reference in case the *External Reference Type* (Object 201Dh) is set for *online*. The unit for this object is the internal unit defined for each drive mode (position / speed / torque).

Object description:

Index	$201C_h$
Name	External online reference
Object code	VAR
Data type	INTEGER32

Entry description:

Access	RW
PDO mapping	Possible
Units	Internal, operating mode dependant
Value range	INTEGER32
Default value	0

13. External Reference Speed Mode[41]

13.1. Overview

In this mode, the drive performs speed control with demand velocity read from the external reference provided by other devices.

There are 2 types of external references:

- Analogue – read by the drive via a dedicated analogue input (12-bit resolution)
- Online – received online via the CAN bus communication channel from the CANopen master in object **External On-line Reference** (index $201C_h$)

The reference type is selected with object **External Reference Type** (index $201D_h$). The option digital is not allowed in external reference speed mode.

In external reference speed mode, you can limit the maximum acceleration at sudden changes of the speed reference and thus to get a smoother transition. This feature is activated by setting ControlWord.5=1 and the maximum acceleration value in object Profile Acceleration (6083_h).

Control word in external reference speed mode

MSB							**LSB**
See 6040h	Halt	See 6040h	Reserved	Activate Acceleration Limitation	Enable External Speed Mode	See 6040h	
15 9	8	7	6	5	4	3	0

Table 13.1 *Control Word bit description for External Reference Speed Mode*

Name	**Value**	**Description**
Enable External Speed Mode	0	External speed mode inactive
	1	External speed mode active
Activate Speed Limitation	0	Do not limit acceleration on the inactive to active mode transition
	1	Limit acceleration when enabling the External Speed mode
Halt	0	Execute the instruction of bit 4
	1	Stop drive with *profile acceleration*

Status word in external reference speed mode

MSB							LSB
See 6041h	Max slippage error	Speed	See 6041h	Target reached	See 6041h		
15 14	13	12	11	10	9		0

[41] Only available for MotionChip III family. Available also on request for MotionChip II family

***Table 13.2** Status Word bit description for External Reference Speed Mode*

Name	Value	Description
Target reached	0	Halt = 0: Always 0 Halt = 1: Drive decelerates
	1	Halt = 0: Always 0 Halt = 1: Velocity of drive is 0
Speed	0	Speed is not equal to 0
	1	Speed is equal to 0
Max slippage error	0	Maximum slippage not reached
	1	Maximum slippage reached

***Remark:** In order to set / reset bit 12, the object from index 606F$_h$, velocity threshold from profile velocity mode will be used. If the actual velocity of the drive / motor is below the velocity threshold, then bit 12 will be set, else it will be reset.*

14. External Reference Torque Mode[42]

14.1. Overview

In this mode, the drive is controlled in torque mode and the external reference is interpreted as torque/current reference.

There are 2 types of external references:

- Analogue – read by the drive via a dedicated analogue input (12-bit resolution)
- Online – received online via the CAN bus communication channel from the CANopen master in object **External On-line Reference** (index $201C_h$)

The reference type is selected with object **External Reference Type** (index $201D_h$). The option digital is not allowed in external reference speed mode.

Control word in external reference torque mode

MSB						LSB
See 6040h	Halt	See 6040h	Reserved	Reserved	Enable External Torque Mode	See 6040h
15 9	8	7	6	5	4	3 0

Table 14.1 *Control Word bit description for External Reference Torque Mode*

Name	Value	Description
Enable External Torque Mode	0	External torque mode inactive
	1	External torque mode active
Halt	0	Execute the instruction of bit 4
	1	Stop drive – set torque reference to 0

Status word in external reference torque mode

MSB							LSB
See 6041h		Reserved	Reserved	See 6041h	Target reached	See 6041h	
15	14	13	12	11	10	9	0

Table 14.2 *Status Word bit description for External Reference Torque Mode*

Name	Value	Description
Target reached		Always 0

[42] Only available for MotionChip III family. Available also on request for MotionChip II family

15. Data Exchange between CANopen master and drives

15.1. Checking Setup Data Consistency

During the configuration phase, a CANopen master can quickly verify using the checksum objects and a reference **.sw** file whether the non-volatile EEPROM memory of the IDM680 drive contains the right information. If the checksum reported by the drive doesn't match the one computed from the **.sw** file, the CANopen master can download the entire **.sw** file into the drive EEPROM using the communication objects for writing data into the drive EEPROM.

In order to be able to inspect or to program any memory location of the drive, as well as for downloading of a new TML program (application software), three manufacturer specific objects were defined: Object 2064_h – Read/Write Configuration Register, 2065_h – Write Data at address specified in 2064_h, 2066_h – Read Data from address specified in 2064_h, 2067_h – Write data at specified address.

15.2. Image Files Format and Creation

An image file (with extension **.sw**) is a text file that can be read with any text editor. It contains blocks of data separated by an empty line. Each block of data starts with the block start address, followed by data values to place in ascending order at consecutive addresses: first data – to write at start address, second data – to write at start address + 1, etc. All the data are hexadecimal 16-bit values (maximum 4 hexadecimal digits). Each line contains a single data value. When less then 4 hexadecimal digits are shown, the value must be right justified. For example 92 represent 0x0092.

The **.sw** software files can be generated either from EasySetUp or from EasyMotion Studio.

In EasySetUp you create a **.sw** file with the command **Setup | EEPROM Programmer File…** The software file generated, includes the setup data and the drive/motor configuration ID with the user programmable application ID.

In EasyMotion Studio you create a **.sw** file with one of the commands: **Application | EEPROM Programmer File | Motion and Setup** or **Setup Only**. The option **Motion and Setup** creates a **.sw** file with complete information including setup data, TML programs, cam tables (if present) and the drive/motor configuration ID. The option **Setup Only** produces a **.sw** file identical with that produced by EasySetUp i.e. having only the setup data and the configuration ID.

The **.sw** file can be programmed into a drive:

- from a CANopen master, using the communication objects for writing data into the drive EEPROM
- using the EEPROM Programmer tool, which comes with EasySetUp but may also be installed separately. The EEPROM Programmer was specifically designed for repetitive fast and easy programming of **.sw** files into the Technosoft drives during production.

15.3. Data Exchange Objects

15.3.1. Object 2064h: Read/Write Configuration Register

Object Read/Write Configuration Register 2064_h is used to control the read from drive memory and write to drive memory functions. This object contains the current memory address that will be used for a read/write operation. It can also be specified through this object the type of memory used (EEPROM, data or program) and the data type the next read/write operation refers to. Additionally, it can be specified whether an increment of the memory address should be performed or not after the read or write operation. The auto-increment of the memory address is particularly important in saving valuable time in case of a program download to the drive as well when a large data block should be read from the device.

Object description:

Index	2064_h
Name	Read/Write configuration register
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	Possible
Units	-
Value range	0 … 2^{32}-1
Default value	0x84

Table 15.1 *Read/Write Configuration Register bit description*

Bit	Value	Description
31…16	x	16-bit memory address for the next read/write operation
15…8	0	Reserved (always 0)
7	0	Auto-increment the address after the read/write operation
	1	Do not auto-increment the address after the read/write operation
6…4	0	Reserved (always 0)
3,2	00	Memory type is program memory
	01	Memory type is data memory
	10	Memory type is EEPROM memory
	11	Reserved
1	0	Reserved (always 0)
0	0	Next read/write operation is with a 16-bit data
	1	Next read/write operation is with a 32-bit data

15.3.2. Object 2065h: Write 16/32 bits data at address set in Read/Write Configuration Register

The object is used to write 16 or 32-bit values using the parameters specified in object 2064_h – Read/Write Configuration Register. After the successful write operation, the memory address in object 2064_h, bits 31…16 will be auto-incremented or not, as defined in the same register. The auto-incrementing of the address is particularly useful in downloading a program (software application) in the drives memory.

Object description:

Index	2065_h
Name	Write data at address set in 2064_h (16/32 bits)
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	WO
PDO mapping	Possible
Units	-
Value range	0 … 2^{32}-1
Default value	No

The structure of the parameter is the following:

Bit	Value	Description
31…16	0	Reserved if bit 0 of object 2064_h is 0 (operation on 16 bit variables)
	X	16-bit MSB of data if bit 0 of object 2064_h is 1 (operation on 32 bit variables)
15…0	X	16 bit LSB of data

15.3.3. Object 2066h: Read 16/32 bits data from address set in Read/Write Configuration Register

This object is used to read 16 or 32-bit values with parameters that are specified in object 2064_h – Read/Write Configuration Register. After the successful read operation, the memory address in object 2064_h, bits 31…16, will be auto-incremented or not, as defined in the same register.

Object description:

Index	2066_h
Name	Read data from address set in 2064_h (16/32 bits)
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RO
PDO mapping	No
Units	-
Value range	UNSIGNED32
Default value	No

The structure of the parameter is the following:

Bit	Value	Description
31…16	0	Reserved if bit 0 of object 2064_h is 0 (operation on 16 bit variables)
	X	16-bit MSB of data if bit 0 of object 2064_h is 1 (operation on 32 bit variables)
15…0	X	16 bit LSB of data

15.3.4. Object 2067h: Write data at specified address

This object is used to write a single 16-bit value at a specified address in the memory type defined in object 2064_h – Read/Write Configuration Register. The rest of the bits in object 2064_h do not count in this case, e.g. the memory address stored in the Read/Write Control Register is disregarded and also the control bits 0 and 7. The object may be used to write only 16-bit data. Once the type of memory in the Read/Write Control Register is set, the object can be used independently. If mapped on a PDO, it offers quick access to any drive internal variable.

Object description:

Index	2067_h
Name	Write data at specified address
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	WO
PDO mapping	Possible
Units	-
Value range	UNSIGNED32
Default value	No

Bit	Value	Description
31…16	x	16-bit memory address
15…0	X	16 bit data value to be written

15.3.5. Object 2069h: Checksum configuration register

This object is used to specify a start address and an end address for the drive to execute a checksum of the E2ROM memory contents. The 16 LSB of this object are used for the start address of the checksum, and the 16 MSB for the end address of the checksum.

Note: The end address of the checksum must be computed as the start address to which you add the length of the section to be checked. The drive will actually compute the checksum for the memory locations between start address and end address.

The checksum is computed as a 16 bit unsigned addition of the values in the memory locations to be checked. When the object is written through SDO access, the checksum will be computed and stored in the read-only object $206A_h$.

Object description:

Index	2069_h
Name	Checksum configuration register
Object code	VAR
Data type	UNSIGNED32

Entry description:

Access	RW
PDO mapping	No
Units	-
Value range	UNSIGNED32
Default value	No

The structure of the parameter is the following:

Bit	Value	Description
31…16	X	16-bit end address of the checksum
15…0	X	16 bit start address of the checksum

15.3.6. Object 206Ah: Checksum read register

This object stores the latest computed checksum.

Object description:

Index	$206A_h$
Name	Checksum read register
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	RO
PDO mapping	No
Units	-

Value range	UNSIGNED16
Default value	No

15.4. Usage example for the data exchange objects

Check the integrity of the setup data on a drive and update it if needed.

Let's suppose that the setup data of an IDM680 drive is located at EEPROM addresses between 0x5E06 and 0x5EFF. Here are the steps to be taken in order to check the setup data integrity and to re-program the drive if necessary:

1. **Compute the checksum in the sw file**. Let's suppose that the computed checksum is 0x1234.
2. **Access object 2069_h in order to compute the checksum of the setup table located on the drive.** Write the value 0x5EFF5E06

Send the following message (SDO write to object 2069_h sub-index 0, 32-bit value $5EFF5E06_h$):

COB-ID	**60A**
Data	**23 69 20 00 06 5E FF 5E**

Following the reception of this message, the drive will compute the checksum of the EEPROM locations 0x5E06 to 0x5EFF. The result is stored in the object $206A_h$.

3. **Read the computed checksum from object $206A_h$**.

Send the following message (SDO read from object $206A_h$ sub-index 0):

COB-ID	**60A**
Data	**40 6A 20 00 00 00 00 00**

The drive returns the following message (Object $206A_h$ = 0x2345):

COB-ID	**58A**
Data	**4B 6A 20 00 45 23 00 00**

As the returned checksum (0x2345) does not match the checksum computed from the .sw file, the setup table has to be configured from the .sw file.

4. **Prepare the Read/Write Configuration Register for EEPROM write**. Write the value 0x5E060009 into the object 2064_h (write 32-bit data at EEPROM address 0x5E06 and auto-increment the address after the write operation).

Send the following message (SDO write to object 2064_h sub-index 0, 32-bit value $5E060009_h$):

COB-ID	**60A**
Data	**23 64 20 00 09 00 06 5E**

5. **Write the sw file data 32 bits at a time**. Supposing that the first 2 entries in the .sw file are 0x9876 and 0x3456, you have to write the value 0x34569876 into the object 2065_h.

Send the following message (SDO write to object 2065_h sub-index 0, 32-bit value 34569876_h):

COB-ID	**60A**
Data	**23 65 20 00 76 98 56 34**

6. **Repeat step 5 until all the setup data is written**
7. **Re-check the checksum (repeat steps 2 and 3). If ok, go to step 8**
8. **Reset the drive in order to activate the new setup.**

Send the following message (NMT message for reset node to drive with NodeID 10):

COB-ID	**0**
Data	**81 0A**

16. Advanced features

16.1. Overview

Due to its embedded motion controller, a Technosoft intelligent drive/motor offers many programming solutions that may simplify a lot the task of a CANopen master. This paragraph overviews a set of advanced programming features which can be used when combining TML programming at drive level with CANopen master control. All features presented below require usage of EasyMotion Studio as TML programming tool

Remark: *If you don't use the advanced features presented below you don't need EasyMotion Studio.*

16.2. Using TML Functions to Split Motion between Master and Drives

With Technosoft intelligent drives you can really distribute the intelligence between a CANopen master and the drives in complex multi-axis applications. Instead of trying to command each step of an axis movement, you can program the drives using TML to execute complex tasks and inform the master when these are done. Thus for each axis, the master task may be reduced at: calling TML functions (with possibility to abort their execution) stored in the drives EEPROM and waiting for a message, which confirms the finalization of the TML functions execution.

16.2.1. Build TML functions within EasyMotion Studio

The following steps describes how to create TML functions with EasyMotion Studio

1. **Define the TML functions.** Open the EasyMotion Studio project and select the Functions entry from the project tree. On the right side of the project panel add the TML functions executed by the drive. You may also remove, rename and change the functions download order.

Remark: *You can call up to 10 TML functions using the CANopen objects.*

2. **Add the TML code.** The added functions are listed in the project tree under the **Functions** entry. Select each function from the list and add the TML code that will be executed by the function.
3. **Download the TML functions into the drive memory**. Use the menu command **Application | Motion | Build** to create the executable code and the menu command **Application | Motion | Download Program** to download the TML code into the drive memory.

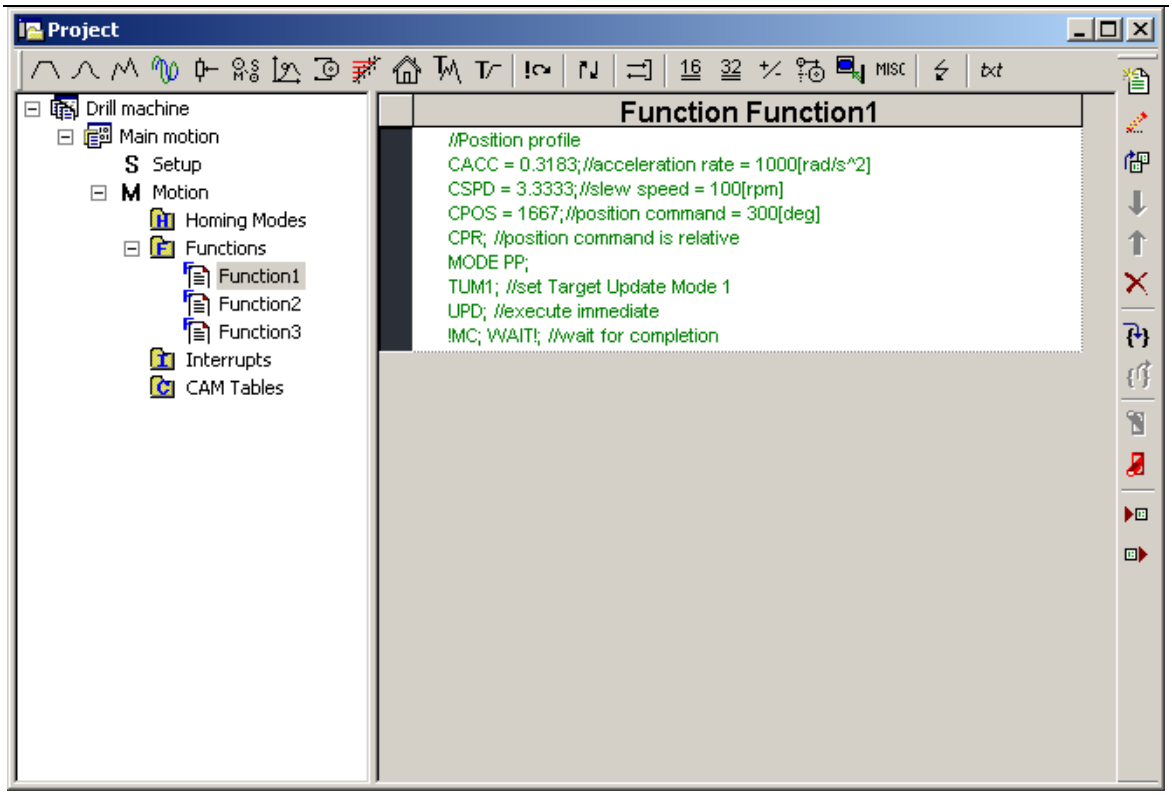


Figure 16.1 *EasyMotion Studio project window – functions edit view*

16.2.2. TML Function Objects

16.2.2.1. Object 2006h: Call TML Function

The object allows the execution of a previously downloaded TML function. When a write is performed to this object, the TML function with the index specified in the value provided is called. The TML function body is defined using EasyMotion Studio and saved in the EEPROM memory of the drive. The function index represents an offset in a predefined table of TML callable functions.

It is not possible to call another TML function, while the previous one is still running. In this case bits 7 (warning) from the Status Word and 14 (command error) from Motion Error Register are set, and the function call is ignored. The execution of any called TML function can be aborted by setting bit 13 in Control Word.

There are 10 TML functions that can be called through this mechanism (the first 10 TML functions defined using the EasyMotion Studio advanced programming environment). Any attempt to call another function (writing a number different from 1..10 in this object) will be signaled with an SDO abort code 0609 0030h (Value range of parameter exceeded). If a valid value is entered, but no TML function is defined in that position, an SDO abort code will be issued: 0800 0020h (Data cannot be transferred or stored to the application).

Object description:

Index	2006_h
Name	Call TML function
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Units	-
Value range	1...10
Default value	-

16.3. Executing TML programs

The distributed control concept can go on step further. You may prepare and download into a drive a complete TML program including functions, homing procedures, etc. The TML program execution can be started simply by writing a value in the dedicated object.

16.3.1. Object 2077h: Execute TML program

This object is used in order to execute the TML program from either EEPROM or RAM memory. The TML program is downloaded using the EasyMotion Studio software or by the CANopen master using the .sw file created in EasyMotion Studio.

Writing any value in this object (through the SDO protocol) will trigger the execution of the TML program in the drive. If no TML program is found on the drive, an SDO abort code will be issued: 0800 0020h (Data cannot be transferred or stored to the application).

Object description:

Index	2077_h
Name	Execute TML program
Object code	VAR
Data type	UNSIGNED16

Entry description:

Access	WO
PDO mapping	No
Value range	UNSIGNED16
Default value	-

16.4. Loading Automatically Cam Tables Defined in EasyMotion Studio[43]

Apart from the standard modes of operation of DSP-402, the Technosoft IDM680 drives offer others like: electronic gearing, electronic camming, external modes with analogue or digital reference etc. When electronic camming is used, the cam tables can be loaded in the following ways:

a) The master downloads the cam points into the drive active RAM memory after each power on;

b) The cam points are stored in the drive EEPROM and the master commands their copy into the active RAM memory

c) The cam points are stored in the drive EEPROM and during the drive initialization (transition to Ready to Switch ON status) are automatically copied from EEPROM to the active RAM

For the last 2 options the cam table(s) are defined in EasyMotion Studio and are included in the information stored in the EEPROM together with the setup data and the TML programs/functions.

Remark: *The cam tables are included in the* ***.sw*** *file generated with EasyMotion Studio. Therefore, the master can check the cam presence in the drive EEPROM using the same procedure as for testing of the setup data.*

16.4.1. CAM table structure

The cam tables are arrays of X, Y points, where X is the cam input i.e. the master position and Y is the cam output i.e. the slave position. The X points are expressed in the master internal position units, while the Y points are expressed in the slave internal position units. Both X and Y points 32-bit long integer values. The X points must be positive (including 0) and equally spaced at: 1, 2, 4, 8, 16, 32, 64 or 128 i.e. having the interpolation step a power of 2 between 0 and 7. The maximum number of points for one cam table is 8192.

As cam table X points are equally spaced, they are completely defined by two data: the **Master start value** or the first X point and the **Interpolation step** providing the distance between the X points. This offers the possibility to minimize the cam size, which is saved in the drive/motor in the following format:

- 1st word (1 word = 16-bit data):
  - Bits 15-13 – the power of 2 of the interpolation step. For example, if these bits have the binary value 010 (2), the interpolation step is $2^2 = 4$, hence the master X values are spaced from 4 to 4: 0, 4, 8, 12, etc.
  - Bits 12-0 – the length -1 of the table. The length represents the number of points (one point occupies 2 words)
- 2nd and 3rd words: the Master start value (long), expressed in master position units. 2nd word contains the low part, 3rd word the high part
- 4th and 5th words: Reserved. Must be set to 0

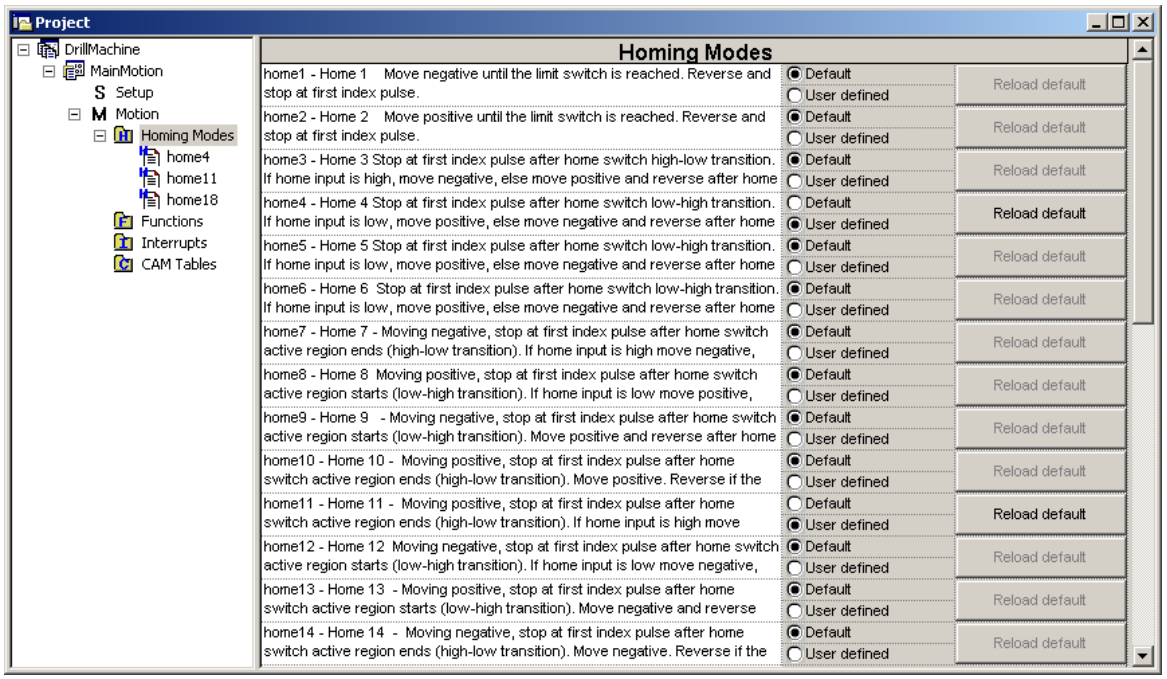
[43] Only available for the MotionChip III family. Available also upon request for MotionChip II family.

- Next pairs of 2 words: the slave Y positions (long), expressed in position units. The 1st word from the pair contains the low part and the 2nd word from the pair the high part

Last word: the cam table checksum, representing the sum modulo 65536 of all the cam table data except the checksum word itself.

16.5. Customizing the Homing Procedures[44]

The homing methods defined by the DSP-402 are highly modifiable to accommodate your application. If needed, any of these homing modes can be customized. In order to do this you need to select the Homing Modes from your EasyMotion Studio application and in the right side to set as “User defined” one of the Homing procedures. Following this operation the selected procedure will occur under Homing Modes in a sub tree, with the name *HomeX* where X is the number of the selected homing.



If you click on the *HomeX* procedure, on the right side you’ll see the TML function implementing it. The homing routine can be customized according to your application needs. Its calling name and method remain unchanged.

[44] Only available for MotionChip III family.

16.6. Customizing the Drive Reaction to Fault Conditions[45]

Similarly to the homing modes, the default service routines for the TML interrupts can be customized according to your application needs. However, as most of these routines handle the drive reaction to fault conditions, it is mandatory to keep the existent functionality while adding your application needs, in order to preserve the correct protection level of the drive. The procedure for modifying the TML interrupts is similar with that for the homing modes.

[45] Only available for MotionChip III family.

Appendix A Object Dictionary by Index

Index	Sub-index	Description	Available for MotionChip III family?	Available for MotionChip II family?
1000h	00h	Device type	YES	YES
1001h	00h	Error register	YES	YES
1002h	00h	Manufacturer status register	YES	YES
1003h		Predefined error field	YES	YES
	00h	Number of errors in history	YES	YES
	01h	Standard error field (history 1)	YES	YES
	02h	Standard error field (history 2)	YES	YES
	03h	Standard error field (history 3)	YES	YES
	04h	Standard error field (history 4)	YES	YES
	05h	Standard error field (history 5)	YES	YES
1005h	00h	COB-ID of the SYNC message	YES	YES
1006h	00h	Communication cycle period	YES	YES
1008h	00h	Manufacturer device name	YES	NO
100Ah	00h	Manufacturer software version	YES	NO
100Ch	00h	Guard time	YES	YES
100Dh	00h	Lifetime factor	YES	YES
1013h	00h	High resolution time stamp	YES	YES
1014h	00h	COB-ID Emergency object	YES	YES
1017h	00h	Producer heartbeat time	YES	YES
1018h		Identity Object	YES	YES
	00h	Number of entries	YES	YES
	01h	Vendor ID	YES	YES
1200h		Server SDO parameter	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID Client -> Server (rx)	YES	YES
	00h	Server SDO parameter	YES	YES
1400h		Receive PDO1 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO1	YES	YES
	02h	Transmission type	YES	YES
1401h		Receive PDO2 communication parameters	YES	YES

	00h	Number of entries	YES	YES
	01h	COB-ID RPDO2	YES	YES
	02h	Transmission type	YES	YES
1402h		Receive PDO3 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO3	YES	YES
	02h	Transmission type	YES	YES
1403h		Receive PDO4 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID RPDO4	YES	YES
	02h	Transmission type	YES	YES
1600h		RPDO1 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
1601h		RPDO2 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 6060h – modes of operation	YES	YES
1602h		RPDO3 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 607Ah – target position	YES	YES
1603h		RPDO4 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6040h – control word	YES	YES
	02h	2nd mapped object – 60FFh – target velocity	YES	YES
1800h		TPDO1 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO1	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1801h		TPDO2 communication parameters	YES	YES
	00h	Number of entries	YES	YES

	01h	COB-ID TPDO2	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1802h		TPDO3 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO3	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1803h		TPDO4 communication parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	COB-ID TPDO4	YES	YES
	02h	Transmission type	YES	YES
	03h	Reserved	YES	YES
	04h	Reserved	YES	YES
	05h	Event timer	YES	YES
1A00h		TPDO1 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
1A01h		TPDO2 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
	02h	2nd mapped object – 6061h – modes of operation display	YES	YES
1A02h		TPDO3 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 6041h – status word	YES	YES
	02h	2nd mapped object – 6064h – position actual value	YES	YES
1A03h		TPDO4 mapping parameters	YES	YES
	00h	Number of entries	YES	YES
	01h	1st mapped object – 606Bh – velocity demand value	YES	YES

	02h	2nd mapped object – 606Ch – velocity actual value	YES	YES
2000h	00h	Motion Error Register	YES	YES
2001h	00h	Motion Error Register mask	YES	YES
2002h	00h	Drive status mask	YES	YES
2004h	00h	COB-ID High resolution time stamp	YES	YES
2005h	00h	Max slippage time out	YES	NO
2006h	00h	Call TML function	YES	YES
2010h	00h	Master settings	YES	NO
2012h	00h	Master resolution	YES	NO
2013h		EGEAR multiplication factor	YES	NO
	00h	Number of entries	YES	NO
	01h	EGEAR ratio numerator (slave)	YES	NO
	02h	EGEAR ratio denominator (master)	YES	NO
2017h	00h	Master actual position	YES	NO
2018h	00h	Master actual speed	YES	NO
2019h	00h	CAM table load address	YES	NO
201Ah	00h	CAM table run address	YES	NO
201Bh	00h	CAM offset	YES	NO
201Ch	00h	External on-line reference	YES	NO
201Dh	00h	External reference type	YES	NO
2022h	00h	Control effort	YES	YES
2023h	00h	Jerk time	YES	YES
2025h	00h	Stepper current in open loop operation	YES	YES
2026h	00h	Stand-by current for stepper in open loop operation	YES	YES
2027h	00h	Timeout for stepper stand-by current	YES	YES
2040h	00h	Digital inputs status	NO	YES
2042h	00h	Digital inputs mask	YES	NO
2043h	00h	Digital outputs command	NO	YES
2045h	00h	Digital outputs status	YES	NO
2046h	00h	Analogue input: Reference	YES	NO
2047h	00h	Analogue input: Feedback	YES	NO
2050h	00h	Over current protection level	YES	YES
2051h	00h	Over current time out	YES	YES
2052h	00h	Motor nominal current	YES	YES
2053h	00h	I2t protection integrator limit	YES	YES

2054h	00h	I2t protection scaling factor	YES	YES
2055h	00h	DC-link voltage	YES	NO
2058h	00h	Drive temperature	YES	NO
2060h	00h	Software version of the TML application	YES	NO
2064h	00h	Read/Write configuration register	YES	YES
2065h	00h	Write data at address set in object 2064h (16/32 bits)	YES	YES
2066h	00h	Read data from address set in object 2064h (16/32 bits)	YES	YES
2067h	00h	Write data at specified address	YES	YES
2069h	00h	Checksum configuration register	YES	YES
206Ah	00h	Checksum read register	YES	YES
206Bh	00h	CAM input scaling factor	YES	NO
206Ch	00h	CAM output scaling factor	YES	NO
206Fh	00h	Time notation index	YES	NO
2070h	00h	Time dimension index	YES	NO
2071h		Time factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
2072h	00h	Interpolated position mode status	YES	YES
2073h	00h	Interpolated position buffer length	YES	YES
2074h	00h	Interpolated position buffer configuration	YES	YES
2075h		Position triggers	YES	NO
	00h	Number of entries	YES	NO
	01h	Position trigger 1	YES	NO
	02h	Position trigger 2	YES	NO
	03h	Position trigger 3	YES	NO
	04h	Position trigger 4	YES	NO
2076h	00h	Save current configuration	YES	YES
2077h	00h	Execute TML program	YES	YES
2078h	00h	Execute auto-tuning for Linear Halls config	YES	NO
2079h	00h	Interpolated position initial position	YES	YES
6007h	00h	Abort connection option code	YES	NO
6040h	00h	Control word	YES	YES
6041h	00h	Status word	YES	YES
605Eh	00h	Fault reaction option code	YES	NO

6060h	00h	Modes of operation	YES	YES
6061h	00h	Modes of operation display	YES	YES
6062h	00h	Position demand value	YES	YES
6063h	00h	Position actual value*	YES	NO
6064h	00h	Position actual value	YES	YES
6065h	00h	Following error window	YES	YES
6066h	00h	Following error time out	YES	YES
6067h	00h	Position window	YES	YES
6068h	00h	Position window time	YES	YES
6069h	00h	Velocity sensor actual value	YES	YES
606Bh	00h	Velocity demand value	YES	YES
606Ch	00h	Velocity actual value	YES	YES
606Fh	00h	Velocity threshold	YES	NO
607Ah	00h	Target position	YES	YES
607Ch	00h	Home offset	YES	YES
6081h	00h	Profile velocity	YES	YES
6083h	00h	Profile acceleration	YES	YES
6085h	00h	Quick stop deceleration	YES	YES
6086h	00h	Motion profile type	YES	YES
6089h	00h	Position notation index	YES	NO
608Ah	00h	Position dimension index	YES	NO
608Bh	00h	Velocity notation index	YES	NO
608Ch	00h	Velocity dimension index	YES	NO
608Dh	00h	Acceleration notation index	YES	NO
608Eh	00h	Acceleration dimension index	YES	NO
6093h		Position factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
6095h		Velocity encoder factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO
6097h		Acceleration factor	YES	NO
	00h	Number of entries	YES	NO
	01h	Numerator	YES	NO
	02h	Divisor	YES	NO

6098h	00h	Homing method	YES	YES
6099h	00h	Homing speeds	YES	YES
609Ah	00h	Homing acceleration	YES	YES
60C0h	00h	Interpolation sub mode select	YES	YES
60C1h		Interpolation Data Record	YES	YES
	00h	Number of entries	YES	YES
	01h	The first parameter	YES	YES
	0nh	The n-th parameter	YES	YES
60F4h	00h	Following error actual value	YES	NO
60F8h	00h	Max slippage	YES	NO
60FCh	00h	Position demand value*	YES	NO
60FDh	00h	Digital inputs	YES	NO
60FEh	00h	Digital outputs	YES	NO
60FFh	00h	Target velocity	YES	YES
6502h	00h	Supported drive modes	YES	YES

TECHNOSOFT