Contents

- **How to Read this Command Reference 3**
  - How to Read this Command Reference 3
  - Symbols and Conventions 4
  - Abbreviations 5
  - Definitions 5
- **Command Reference 7**
- **Appendix 135**
  - What's New in the Update Version starting with MCO 5.00? 135
  - Technical Reference 139
  - Illustrations 147
  - Index 150

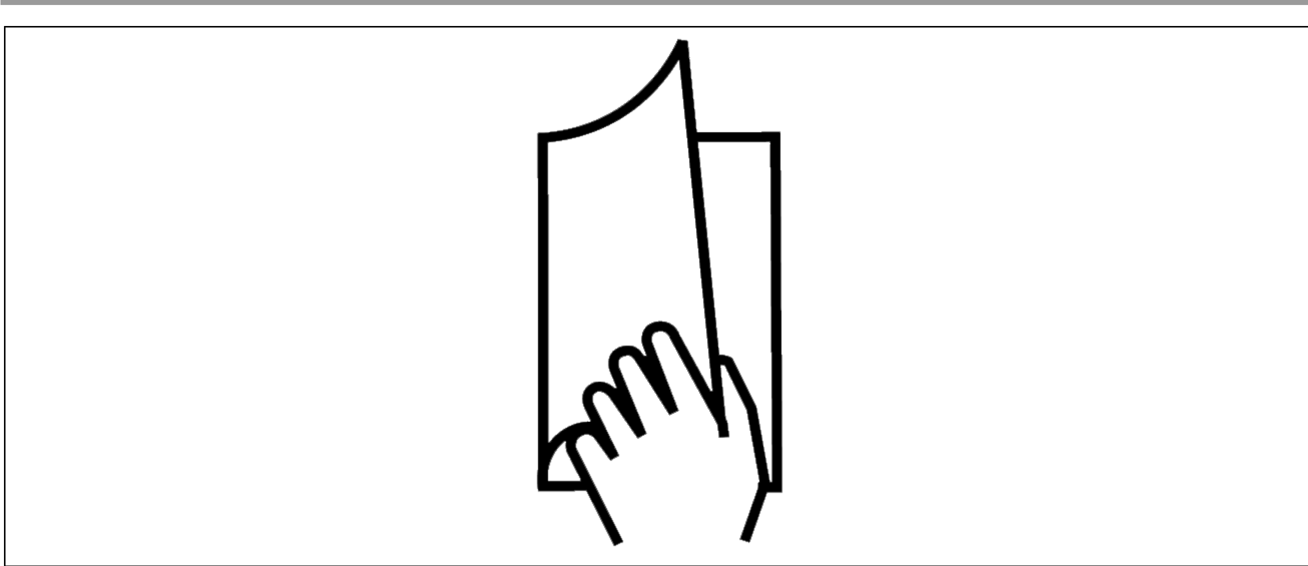
Copyright © Danfoss A/S, 2010

Trademarks VLT is a registered Danfoss trademark.

Hiperface® is a registered trademark of the Sick Stegmann GmbH, Max Stegmann GmbH Antriebstechnik-Elektronik.

Microsoft, Windows 2000, and Windows XP are either registered trademarks or trademarks of the Microsoft Corporation in the USA and other countries.

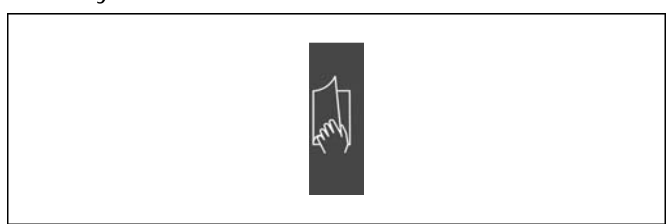
How to Read this Command Reference



☐ How to Read this Command Reference

This Command Reference completes the MCO 305 Design Guide with the with the detailed description of all commands. Please read also the Operating Instructions, in order to be able to work with the system safely and professionally, particularly observe the hints and cautionary remarks.

Chapter **How to Read this Command Reference** informs you about the symbols, abbreviations, and definitions used in this manual.

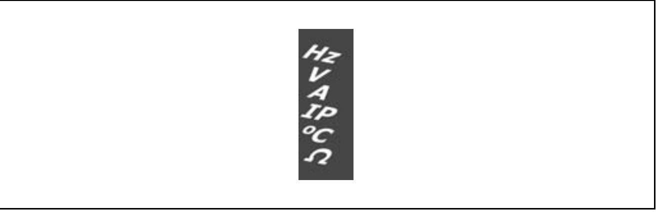


Page divider for 'How to Read this Command Reference'.

Chapter **Command Reference** provides a detailed description of all commands including syntax, parameters, as well as program samples.

Page divider for 'How to Program'.

Chapter **Appendix** gives a quick review of what has changed since previous releases in "What's new?". Experienced users will find detailed information in the technical reference material for example the "Array Structure of CAM Profiles". Plus, the manual ends with an index.



Page divider for 'Appendix'.

The Online Help provides in Chapter **Program Samples** almost 50 program samples which you can use to familiarize yourself with the program or copy directly into your program.

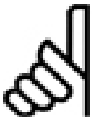
☐ Available Literature for FC 300, MCO 305, and MCT 10 Motion Control Tool

- The MCO 305 Operating Instructions provide the necessary information for built-in, set-up, and optimize the controller.
- The MCO 305 Design Guide entails all technical information about the option board and customer design and applications.
- This MCO 305 Command Reference completes the MCO 305 Design Guide with the detailed description of all commands.
- The VLT® AutomationDrive FC 300 Operating Instructions provide the necessary information for getting the drive up and running.
- The VLT® AutomationDrive FC 300 Design Guide entails all technical information about the drive and customer design and applications.
- The VLT® AutomationDrive FC 300 MCT 10 Operating Instructions provide information for installation and use of the software on a PC.

Danfoss Drives technical literature is also available online at www.danfoss.com/drives.

☐ Symbols and Conventions

Symbols used in this manual:



NB!:
Indicates something to be noted by the reader.

Indicates a general warning.

Conventions

The information in this manual follows the system and uses the typographical features described below to the greatest extent possible:

Menus and Functions

Menus and functions are printed italics, for example: *Controller → Parameters.*

Commands and Parameters

Commands and parameter names are written in capitals, for example: AXEND and KPROP; Parameters are printed in italics, for example: *Proportional factor.*

Parameter Options

Values for use to select the parameter options are written in brackets, e.g. [3].

Keys

The names of keys and function keys are printed in brackets, for example the control key [Ctrl] key, or just [Ctrl], the [Esc] key or the [F1] key.

Abbreviations

Ampere, Milliampere	A, mA
Control word	CTW
Digital Signal Processor	DSP
Frequency Converter	FC
Local Control Panel	LCP
Least significant bit	LSB
Main actual value	MAV
Motion Control Option	MCO
Motion Control Tool	MCT
Minute	min
Most significant bit	MSB
Main Reference	MRV
Master Unit	MU
Digital output switching to high side.	NPN

Parameter	par.
Position Control Loop	PID
Digital output switching to low side.	PNP
Pulses per Revolution	PPR
Quad-counts	qc
Reference	REF
Revolutions per Minute	RPM
Second, Millisecond	s, ms
Sample time	st
Status word	STW
User Unit	UU
Volts	V

Definitions

MLONG

An upper or lower limit for many parameters:

-MLONG = -1,073,741,824

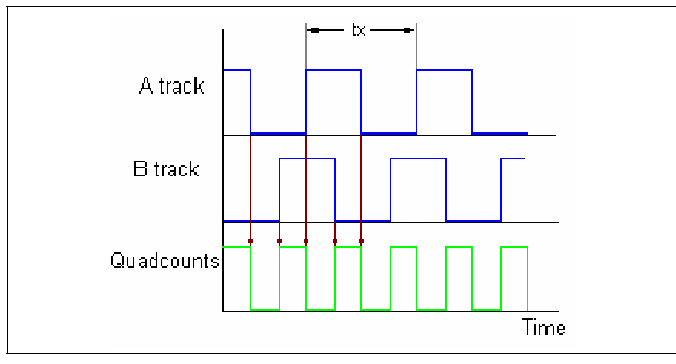
MLONG = 1,073,741,823

Quad-counts

Incremental encoders: 4 quad-counts correspond to one sensor unit.

Absolute encoders: 1:1 (1 qc correspond to one sensor unit).

Through edge detection, a quadrupling of the increments is produced by both tracks (A/B) of the incremental encoder. This improves the resolution.



Derivation of quad counts

☐ User Units

The units for the drive or the slave and the master, respectively, can be defined by the user in any way desired so that the user can work with meaningful measurements.

Starting with MCO 5.00 the factors SYNCFACTM / SYNCFACTS, POSFACT_Z / POSFACT_N are no longer limited to small values

Internally, it is act as follows: Whenever a value must be multiplied by the gear factor (i.e. master increments per ms), at first it is looked if a multiplication will result in an overflow. If so, a factor (64 bit) is used which consists of

SYNCFACTS/SYNCFACTM to multiply the delta_master.

If no overflow occurs, first it is multiplied by SYNCFACTS and then divided by SYNCFACTM.

Concerning the error we are dealing with, this means:

Normal case

Multiplying by SYNCFACTS has no error, but dividing by SYNCFACTM means that the result may be wrong by $1/2^{32}$. That means that (worst case) such an error occurs every ms, i.e. that after 1193 hours (49,71 days) we made an error of 1 qc (Slave).

Big factors

In that case, the used factor (SYNCFACTS/SYNCFACTM) itself could be wrong by $1/2^{32}$. This means that in the worst case an error of delta_master * $1/2^{32}$ occurs every ms. Assume that we have an encoder with 1000 counts (4000 qc) per revolution. Assume further, that we drive with 2000 rpm, i.e. we have a velocity of 133 qc/ms. This means we make an error of 133 * $1/2^{32}$ per ms. From this follows that in worst case (maximum error every ms always in same direction) we could have an error of 1 qc after 9 hours.

This should not be relevant in most applications.

User Units [UU]

All path information in motion commands are made in user units and are converted to quad-counts internally. These also have an effect on all commands for the positioning: e.g. APOS.

The user can also select meaningful units for the CAM control in order to describe the curve for the master and the slave, for example 1/100 mm, or 1/10 degrees in applications where a revolution is being observed.

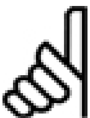
In the CAM control, the maximum run distance of the slave or the slave cycle length are indicated in User Units UU (qc).

The unit can be standardized with a factor. This factor is a fraction which consists of a numerator and denominator:

$$1 \text{ User Unit UU} = \frac{\text{par. 32-12 } \textit{User Unit Numerator}}{\text{par. 32-11 } \textit{User Unit Denomintor}}$$

par. 32-12 *User Unit Numerator* POSFACT_Z
par. 32-11 *User Unit Denominator* POSFACT_N

Scaling determines how many quad-counts make up a user unit. For example, if it is 50375/1000, then one UU corresponds to exactly 50.375 qc.



NB!:
When user units are transferred into qc, then they get truncated. When qc are transferred into user units, then they get rounded.

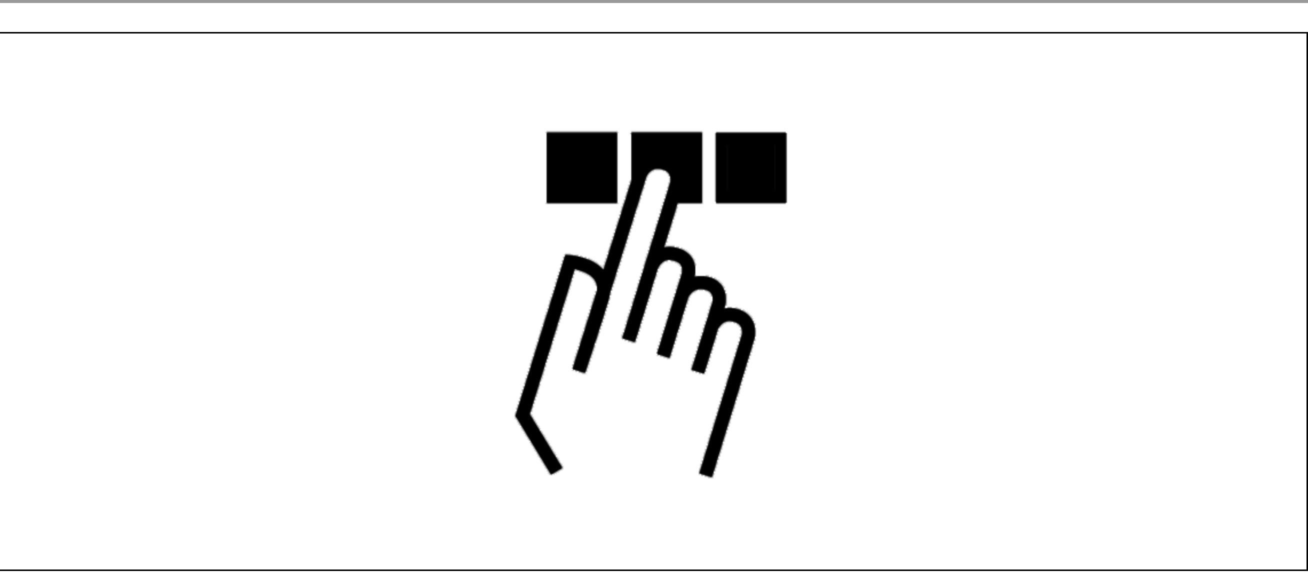
Master Units [MU]

A factor (fraction) is used for the conversion into qc, as with the user unit:

$$1 \text{ Master Unit MU} = \frac{\text{par. 33 – 10 } \textit{Synchronization Factor Master}}{\text{par. 33 – 11 } \textit{Synchronization Factor Slave}}$$

par. 33-10 *Synchronization Factor Master* SYNCFACTM
par. 33-11 *Synchronization Factor Slave* SYNCFACTS

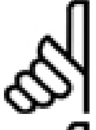
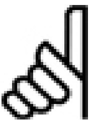
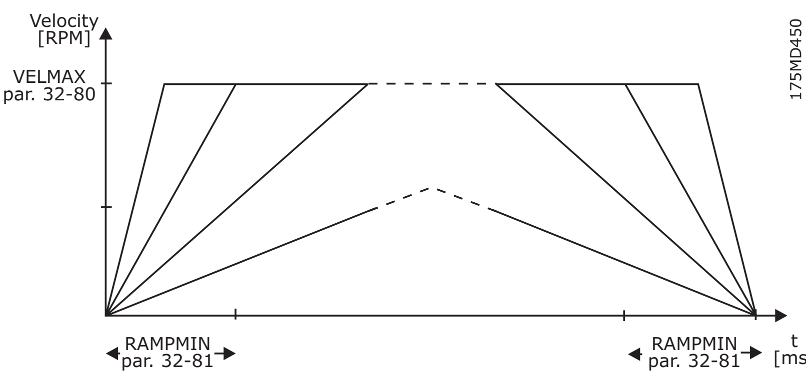
Command Reference



In the following section all commands are listed in alphabetical order and described in detail with syntax examples as well as short program samples. Please read also the section "Basics" in chapter "How to Progam" in the MCO 305 Design Guide.

MG.34.R1.02 – VLT® is a registered Danfoss trademark

☐ ACC

Summary	Setting acceleration for motion commands.
Syntax	ACC a
Parameter	a = acceleration
Description	The ACC command defines the acceleration for the next motion command (speed control, synchronizing or positioning). The value will remain valid until a new acceleration value is set, using the ACC command. The value is related to the parameters 32-81 *Shortest Ramp* and 32-80 *Maximum Velocity* as well as 32-83 *Velocity Resolution*.
	**NB!:** If acceleration has not been defined previous to a motion command, then the maximum acceleration will be the setting of par. 32-85 *Default Acceleration*.
	**NB!:** If the MCO 305 is used to control FC 300, then the ramps should always set via the option card and not in the FC 300. The FC 300 ramps must always be set to minimum.
Command Group	REL, ABS
Cross Index	DEC, VEL, POSA, POSR, Parameters: 32-81 *Shortest Ramp*, 32-80 *Maximum Velocity*, 32-83 *Velocity Resolution*
Syntax Example	ACC 10 /* Acceleration 10 */
Example	minimum acceleration time: 1000 ms maximum velocity: 1500 RPM (25 Rev./s) velocity resolution: 100
	
Program Sample	ACC_01.M

□ APOS

Summary Reads actual position

Syntax res = APOS

Return Value res = absolute actual position related to the actual zero point

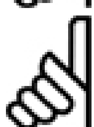
All path information in motion commands are made in user units and are converted to quad-counts internally. (See also Numerator and Denominator, parameters 32-12 and 32-11.)

The user unit (UU) corresponds in standard setting to the number of Quad counts.

$$\text{Parameter} = \frac{\text{par. 32 - 12 } \textit{User Unit Numerator}}{\text{par. 32 - 11 } \textit{User Unit Denomintor}} = 1$$

Description The APOS command can query the absolute position of the axis related to the actual zero point.

If a temporary zero point which has been set via SETORIGIN exists, then the position value is relative to this zero point.



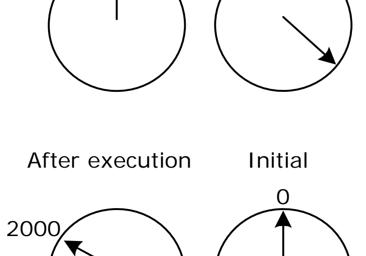
NB!:
The read out using APOS may or may not be equal to the target position or commanded position. The error or deviation may be due to the mechanics involved and truncated decimal values in the User Units.

APOS is affected by the parameters 32-12 and 32-11, and by commands SETORIGIN p, DEFORIGIN.

Example:

```
POSA 2000
PRINT "Actual Position Reached", APOS
```

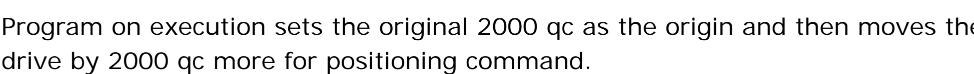
Output:
Actual Position Reached 2000
(depending on PID settings a small deviation may occur)



Example with SETORIGIN

```
SETORIGIN 2000
POSA 2000
PRINT "Actual Position", APOS
```

Output:
Actual Position 2000



Program on execution sets the original 2000 qc as the origin and then moves the drive by 2000 qc more for positioning command.

Command Group SYS

Cross Index CPOS, DEFORIGIN, SETORIGIN, POSA, POSR,
Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*

Syntax Example PRINT APOS /* display the actual position of axis on the PC */

Program Sample APOS_01.M, GOSUB_01.M, MOTOR_01.M

◻ APOSDIFF

Summary	Overflow handling of incremental encoders in applications.
Syntax	res = APOSDIFF oldpos
Return Value	oldpos = APOS at a previous time
Description	Returns difference between APOS and oldpos (res = APOS – oldpos) in UU. This command simplifies overflow handling of incremental encoders in applications. If, for example, the user stores an actual position in his program and wants to calculate the difference at a later time, then he normally has to account for overflow of the position. Instead this command can be used; see below. Internally those routines look if the difference is bigger than POS_LIMIT (0x3FFFFFFF). If so then it is assumed that an overflow happened and it is handled correctly.
	NB!: This will not solve the problem of overflowing if the application uses user units.
Portability	Command is available starting with MCO 5.00
Command Group	SYS
Cross Index	APOS

Syntax Example	oldpos = APOS
	.....
	diff = APOSDIFF oldpos
	// this function returns the difference between APOS and oldpos in user units
	// handling an overflow if necessary (diff = APOS – oldpos)

◻ AVEL

Summary	Queries actual velocity of axis.
Syntax	res = AVEL
Return Value	res = actual velocity of axis in UU/s, the value is signed
Description	This function returns the actual velocity of the axis in User Units per second. The accuracy of the values depends on the duration of the measurement (averaging). The standard setting is 20 ms, but this can be changed by the user with the _GETVEL command. It is sufficient to call up the command once in order to work with another measuring period from then on. Thus, the command: var = _GETVEL 100 sets the duration of the measurement to 100 ms, so that you have a considerably better resolution of the speed with AVEL and MAVEL, however, in condertrast, quick changes are reported with a delay of a maximum of 100 ms.
Command Group	SYS
Cross Index	MAVEL, APOS, _GETVEL
Syntax Example	PRINT AVEL /* queries actual velocity of axis and display on the PC */

☐ AXEND

Summary	AXEND reads info on status of program execution.
Syntax	res = AXEND
Return Value	res = axis status with the following meaning:

Value	Bit	
128	7	1 = Motor is reset, i.e. it is ready to start and is controlling again, e.g. after ERRCLR, MOTOR STOP, MOTOR ON
64	6	1 = axis controller is OFF, motor is off
	4 - 5	not in use
8	3	1 = motor is at STOP status
4	Bit 2	1 = speed mode is active
2	Bit 1	1 = positioning procedure is active
1	Bit 0	1 = target position reached; motor is not in motion

Description

The AXEND command gives the actual status of the axis or the status of the program execution.

This means for example that you can enquire when the "position is reached" and a positioning command (POSA, POSR) has actually been completed. When Bit 1 is set at [0] the positioning process is complete and the position reached.

If, however, the positioning command has been interrupted with MOTOR STOP and continued later with CONTINUE, then the following bits would be set at [1]:

the Bit 0 for "motor is at a standstill"
the Bit 1 for "positioning process active"
the Bit 3 for "motor is at STOP status"
the Bit 6 for "axis controller switched off"

The AXEND command is especially suitable for determining whether or not a movement in the NOWAIT ON condition is terminated.

Command Group SYS

Cross Index WAITAX, STAT, NOWAIT

Syntax Example

```
NOWAIT ON              // do not wait until position is reached
POSA 100000
WHILE (AXEND&2) DO
// as long as the positioning process is active, repeat loop
    IF IN1 THEN        // if input 01 is set
       VEL 100         // increase velocity
       POSA 100000
       WAIT IN1 OFF    // wait until input (key) is off
    ENDIF
ENDWHILE               // position reached
```

Syntax Example

```
IF (AXEND&64) THEN
    OUT 1 1        // set output 01, when axis controller is switched off
ELSE
    OUT 1 0
ENDIF
```

Program Sample AXEND_01.M

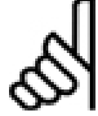
__ Command Reference __

◻ CANDEL

Summary	Deletes all or single CAN objects.
Syntax	CANDEL objno
Parameter	objno = object number, which is returned during the definition of the object = –1 deletes all objects (except the standard objects)
Description	With the CANDEL command CAN objects which were previously created with DEFCANIN or DEFCANOUT can be deleted. Standard objects, for the buffered input/output (OUTMSG or INMSG) cannot be deleted with this command. These cannot be created during initialization.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Syntax Example	CANDEL –1 /* all CAN objects are deleted */

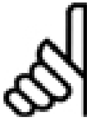
◻ CANIN

Summary	Reads an object via the CAN bus.
Syntax	status = CANIN objno time-out control varhi varlo
Parameter	objno = object number which is returned during the definition of the object. time-out = –1 does not wait for data = 0 waits until data arrives > 0 waits for the data in *time-out* [ms] control = 0 Checks whether the new data has arrived. The new data is subsequently copied to the variables. = 1 Sends a remote frame and waits for data in dependence on *time-out* varhi = Bytes 0 to 3 of the CAN object data varlo = Bytes 4 to 7 of the CAN object data
Return Value	Status = –1 no data has arrived = 0 o.k.
Description	The CANIN command copies the data (if present) of the CAN object 'objno' to the variables 'varhi' and 'varlo'. If 'control = 1', then the data is requested first. It is possible to gather all Transmit-PDOs of digital input modules or CAN-Drive status by using only one CAN telegram. This feature is restricted to the Master-bus. This feature must use the CAN-Object 15 internally which reduces the number of usable CAN objects on the master bus by 1. To enable this feature, the command CANINI 999 must be used This command enables reception of TxPDOs by interrupt and stores every received PDO in a buffer with a depth of 1. This works for all IDs from 1 to 127. That means if any IO-module in that range sends a TxPDO it is captured and stored it in the buffer. The next TxPDO from the same module of course overwrites the first one. To read out of the buffer, the following command can be used: result = CANIN (id * 100) timeout 0 hi lo This command returns -1 if no new information was in the buffer, otherwise it returns 0. You can use the timeout normally (-1 does not wait, 0 waits for ever for new information, n waits n ms).

	The result is returned in hi and lo, but we already arrange byte ordering so that they can be used as if you read them with PDO[]. That means if lo contains a 32-bit number then you can use it right away without reordering the bytes. Whenever you use a new CANINI command or you start a new program, this feature is disabled. Using this feature of course loads the processor if there is heavy PDO traffic on the BUS.
	**NB!:** It is not recommend this feature be used together with other CAN IO commands on the same bus. This may lead to unwanted results. If you use, for example, a CANopen digital I/O module with ID 3 on the master bus with an IN (3*256+1) or with CANINI 3 999, then this will result in a situation where the IN command will work but you would not get the PDOs with the above described CANIN command. This is because two CAN objects will exist with the same ID. In that case, the processor only serves the first one. As mentioned above, we use object 15 for reception of all PDOs, which is the last one.
Portability	Optional command and extended commands CANINI and CANIN to use only one CAN telegram are available starting with MCO>=5.00.
Command Group	CAN
Cross Index	CANOUT, CANDEL, DEFCANOUT, DEFCANIN, CANINI
Syntax Example	MSG = 0 temp = 0 /* Define variables */ rx1 = DEFCANIN 42 8 /* a RX object is created */ STAT = CANIN rx1 1000 0 MSG temp /* wait 1s for data */
Program Sample	CAN_sample.M, CANIN.M

☐ CANINI

Summary	Initializes the necessary objects (PDOs) for data exchange of CANopen nodes, or enables extended CANINI, CANIN function.
Syntax	CANINI no, no, no, ..., CANINI 999 CANINI no, no, 999, no is also possible, but only in the same command. The next new command would delete the previous parameters.
Parameter	no = guard * (busoffset * 1000 + id) guard = –1, +1 (without / with guarding) busoffset = 100000 , 0 (slave bus, master bus) Samples for CANINI values: 0 Deletes any objects, which were formerly assigned using the CANINI command. 999 enables reception of all TPDO1s (0x181-0x1FF) by interrupt and stores every received PDO in a buffer with a depth of 1. 1...127 CAN I/O with bus ID 1...127 with guarding -1...–127 CAN I/O with bus ID 1...127 without guarding
Description	The CANINI command establishes contact with CAN devices and creates permanent corresponding CAN objects in order to be able to communicate (using PDO) with these devices. The advantage is that these input modules can also be used for interrupt functions without permanent bus load due to status polling. If you do not need any interrupts, then you can accelerate the processing of the IN and INB commands with CANINI, since the corresponding devices send these information autonomously every time a change of state happens. It is strongly recommended to “guard” (i.e. CANINI > 0) any CAN devices, which are initialized using CANINI. That is the only way to make sure, that the device is still present and takes part in bus communication. If one device is no longer present, then an error 188 is triggered by the missing feedback of the GUARD object. An error routine, defined with ON ERROR can react to such an error state. When CANINI is executed for drives, the corresponding PDO is created and also the SYNC object if necessary. If guarding is started, a guarding telegram is sent every 20 ms to one device. If for example 4 devices are guarded, it takes 80 ms to check each device once. No response within 100 ms indicates an error 188 (guarding error). A maximum of 16 modules can be stored internally. Every new CANINI command reinitializes all objects which were assigned before with CANINI, i.e. guarding (GUARD) is stopped and also SYNC telegrams. This is not true, if there are permanent objects from par. 32-00 or 32-30 *Incremental Signal Type*. These objects will still remain, like the internal automatically created PDOs corresponding to the definition of parameters 32-00 and 32-30 *Incremental Signal Type*. The CANINI command starts all modules synchronously, i.e. for every module a NMT0 message is sent to set the module on the status “operational”. If a CANINI fails, it is possible to read the SYSVAR 67 [0x01220244] GuardErrorId to find out which id caused the problem.
Portability	Command is available starting with MCO 5.00.

	**NB!:** If the CANINI command is used on controls with multiple separated CAN bus circuits, GUARD and SYNC functionality is only supported on the so-called slave bus.
Command Group	CAN
Cross Index	IN, INAD, INB, OUT, OUTB, OUTDA, par. 32-00 and par. 32-00 *Incremental Signal Types*
Syntax Example	CANINI 1,2,3,4 /* Initialize the CAN modules with pre-set node number */
Program Sample	CAN_sample.M

☐ CANOUT

Summary	Sends message with an internal number.
Syntax	CANOUT no valhi vallo
Parameter	valhi Bytes 0 to 3 of the CAN object data vallo Bytes 4 to 7 of the CAN object data
Description	A CAN message is sent with this command from a sending object defined by DEFCANOUT. The values *hi* and *lo* are 4 bytes long
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	CANIN, CANDEL, DEFCANIN, DEFCANOUT
Syntax Example	valhi = 21 vallo = 41 tx1 = DEFCANOUT 500 8 /* TX object is defined with Id=500 and data length = 8 bytes */ CANOUT tx1 valhi vallo /* a CAN message is sent */
Program Sample	CANOUT.M

◻ COMOPTGET

Summary	Reads a Communication option telegram
Syntax	COMOPTGET no array
Parameter	array = the name of an array which must be at least the size of *no*
	no = number of words to be read
Description	COMOPTGET reads from the Communication option buffer the *no* words and writes them in the array 'array', starting with the first element.
Portability	Option
Communication Option Function	Parameters: Read and write parameters are not affected by the option board.
	NB!: The parameters 9-15 and 9-16 have additionally to be set with the correct values.
Control data:	The function of Control word (CTW) and Main Reference (MRV) depends on the setting of par. 33-82 *Drive Status Monitoring*; Status words (STW) and Main actual value (MAV) is always active:

	Parameter 33-82 "Enable MCO 305"	Parameter 33-82 "Disable MCO 305"
CTW/MRV	Disabled	Active
STW/MAV	Active	Active

Process data:	PCD's 1 – 4 of PPO type 2/ 4 and PCD's 1 – 8 of PPO type 5 are not assigned a parameter number by parameters 9-15 and 9-16 but are used as a free data area which can be used in a APOSS program. The command COMOPTGET is copying the data received on the communication option into an array, where each array element contains one data word (16 bit). The command COMOPTSEND is copying data from an array, where each array element contains one data work (16 bit) into the send buffer on the communication option, from where it is send via the network to the master.
Command Group	Communication option
Cross Index	COMOPTSEND
Program Sample	COM_OPT

◻ COMOPTSEND

Summary	Writes in the Communication option buffer
Syntax	COMOPTSEND no array
Parameter	array = the name of an array which must be at least the size of 'no'
	no = number of words to be sent
Description	COMOPTSEND writes in the Communication option buffer. In doing so the first 'no' values are sent from the 'array'.
Portability	With built-in Communication option.
Communication Option Function	See command COMOPTGET
Command Group	Communication option
Cross Index	COMOPTGET
Program Sample	COM_OPT

□ CONTINUE

Summary	Continues positioning from point of interrupted motion
Syntax	CONTINUE
Description	By using CONTINUE, positioning and speed motion commands which have been aborted via the MOTOR STOP command or an error condition or stopped via MOTOR OFF can be resumed. The CONTINUE command can be used especially in an error subroutine in connection with the ERRCLR command, to enable the correct continuation of a motion procedure following an error abort.
	NB!: However CONTINUE does not continue interrupted synchronization commands.
Command Group	CON
Cross Index	MOTOR STOP, ERRCLR, ON ERROR GOSUB
Syntax Example	CONTINUE /* continue interrupted motion procedure */
Program Sample	MSTOP_01.M

□ CPOS

Summary	Reads the actual command position of an axis
Syntax	res = CPOS
Return Value	res = Absolute commanded position in User Units (UU) related to the actual zero point
Description	The CPOS command queries the actual commanded position of the axis related to the actual zero point. The commanded position is understood to be the temporary set position which is re-calculated every millisecond by the positioning control during a positioning procedure or a movement in rotation mode. The command position can be queried independently of the operating condition (position control during standstill, positioning process, speed control or synchronization).
	NB!: If a set and active temporary zero point (set via SETORIGIN) exists, then the position value is relative to this zero point.
Command Group	SYS
Cross Index	APOS, DEFORIGIN, SETORIGIN, POSA, POSR, Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*
Syntax Example	PRINT CPOS /* actual command position of axis */
Program Sample	CPOS_01.M, GOSUB_01.M

☐ CPOSDIFF

Summary	Overflow handling of incremental encoders in applications.
Syntax	res = CPOSDIFF oldpos
Parameter	oldpos = CPOS at a previous time
Return Value	Returns difference between CPOS and oldpos (res = CPOS – oldpos) in UU.
Description	This command simplifies overflow handling of incremental encoders in applications. If, for example, the user stores an actual position in his program and wants to calculate the difference at a later time, then he normally has to account for overflow of the position. Instead this command can be used; see below. Internally those routines look if the difference is bigger than POS_LIMIT (0x3FFFFFFF). If so then it is assumed that an overflow happened and it is handled correctly.
	This will not solve the problem of overflowing if the application uses user units.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	CPOS

Syntax Example	oldpos = CPOS
	.....
	diff = CPOSDIFF x(1) oldpos
	// this function returns the difference between CPOS and oldpos in user units
	// handling an overflow if necessary (diff = CPOS – oldpos)

☐ CSTART

Summary	Starts the speed mode
Syntax	CSTART
Description	The CSTART command is starting the drive in speed control mode. Acceleration/deceleration, as well as the speed should be set via the ACC, DEC and CVEL commands prior to starting. CSTART does not contain the command MOTOR ON which turns on the motor control. When using CSTART an explicit calling up of MOTOR ON is necessary after previous use of MOTOR OFF.
	NB!: If no speed value has been defined via CVEL before the beginning of CSTART, then the default velocity 0 is used – this means that the motor will not rotate, but the PID controller is active. All CVEL commands following the start of speed mode will be carried out immediately, i.e. a corresponding speed change will take place immediately, with the defined acceleration or deceleration (ACC/DEC).
Command Group	ROT
Cross Index	ACC, DEC, CVEL, CSTOP
Syntax Example	CSTART /* rpm mode start */
Program Sample	CMODE_01.M

☐ CSTOP

Summary	Stops the drive in speed mode
Syntax	CSTOP
Description	Via the CSTOP command, the speed mode is terminated and the positioning mode is started, whereby a still rotating axis is stopped the deceleration defined with DEC and the motor is held in the stop position. **NB!:** A CSTOP command carried out in the positioning mode can also cause an abrupt termination of the positioning procedure.
Command Group	ROT
Cross Index	ACC, DEC, CVEL, CSTART
Syntax Example	CSTOP /* rpm mode stop */
Program Sample	CMODE_01.M

☐ CURVEPOS

Summary Retrieve slave curve position that corresponds to the current master position of the curve.

Syntax res = CURVEPOS

Return Value res = Slave position in CAM units (UU) absolute to the current zero point.

Description CURVEPOS returns the slave value which corresponds to the actual curve master position.

The position can be retrieved independently of the operating status (position control at standstill, positioning procedure, velocity control or synchronization).

CMASTERCPOS (SYSVAR) and CURVEPOS are now updated even if SYNCC is no longer active. The update of these values starts after a SETCURVE command (if par. 33-23 *Start Behavior for Sync* is < 2000) or after SYNCC and the first master marker (if par. 33-23 = 2000).

After the SYNCC command is stopped, we continue to update these values if *Start Behavior for Sync.* < 2000.

NB!:
The position is only defined if a SETCURVE has been set before.

NB!:
If a temporary zero point exists which has been set with SETORIGIN and is active, the position value will refer to this zero point.

NB!:
DEFMCPOS and DEFMORIGIN can still modify this position.

Command Group CAM

Cross Index APOS, DEFORIGIN, SETORIGIN, POSA, POSR, DEFMCPOS,
Parameters: 33-10 *Syncfactor Master*, 33-11 *Syncfactor Slave*

Syntax Example PRINT CURVEPOS // print actual slave position of the curve

Sample Fix points of a curve:

Master	Slave
0	0
500	500
700	300
1000	1200

Say the current master position is 800. Then the CURVEPOS returns the corresponding slave position of 450.

Case 1: Current Master Position is 800 and current slave position is 200. CURVEPOS will return the value 450.

Case 2: Current Master Position is 800 and current slave position is 700. CURVEPOS will return the value 450.

Hence CURVEPOS is independent of the slave position.

◻ CVEL

Summary	sets velocity for speed controlled motor movements
Syntax	CVEL v
Parameter	v = velocity value (negative value for reversing) $$\text{Command Velocity [RPM]} = V * \frac{\text{par. 32 - 80 Maximum Velocity}}{\text{par. 32 - 83 Velocity Resolution}}$$
Description	The velocity for the next speed controlled motor movement is set with the CVEL command. The value remains valid until a further CVEL sets a new velocity. The velocity value to be given will be related to the parameters 32-80 *Maximum Velocity* and 32-83 *Velocity Resolution*. **NB!:** CVEL commands which take place after CSTART will be carried out immediately i.e. the velocity will be adapted via the ACC/DEC set acceleration or deceleration to the new value of CVEL. If a velocity has not been defined before the start of speed control mode (CSTART), then the default velocity is 0, i.e. the motor will not turn, and a velocity input via CVEL will start the movement in speed control mode.
Command Group	ROT
Cross Index	ACC, DEC, CSTART, CSTOP, Parameter: 32-80 *Maximum Velocity*
Syntax Example	CVEL 100
Program Sample	CMODE_01.M

◻ DEC

Summary	sets deceleration
Syntax	DEC a
Parameter	a = deceleration
Description	The DEC command defines the deceleration for the next motion command (speed control synchronization or positioning). This value will remain valid until a new deceleration value is set with another DEC command. The value is related to the parameters 32-81 *Shortest Ramp* and 32-80 *Maximum Velocity* as well as 32-83 *Velocity Resolution*. **NB!:** If deceleration is not defined previous to the positioning command then deceleration will be the setting of parameter 32-85 *Default Acceleration*. **NB!:** If you work with the MCO 305 then you should always set the ramps via the option card and not in the FC 300. The FC ramps must always be set to minimum.
Command Group	REL, ABS
Cross Index	ACC, Parameters: 32-81 *Shortest Ramp*, 32-80 *Maximum Velocity*, 32-83 *Velocity Resolution*
Syntax Example	ACC 50 /* acceleration: 50, while braking 10 */ DEC 10
Example	minimum acceleration time: 1000 ms maximum velocity: 1500 RPM velocity resolution: 100

☐ DEFCANIN

Summary	Defines a receive object.
Syntax	objno = DEFCANIN id dlen
Parameter	id = CAN-identification number
	dlen = data length of the object in bytes (max. 8 bytes)
Return Value	objno
	A positive value means that the object was successfully created. This value is an internal number of the object and is used by other APOSS-CAN commands. A negative value means an error has occurred.
Description	This command defines an incoming communication object in the CAN-Controller.
	This command can also be used with the offset for the slave bus (telegram ID 100000).
	These objects can now be deleted one by one via the command CANDEL objno, where *objno* is the number returned by DEFCANIN or DEFCANOUT.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Program Sample	var1 = 0 /* declare variables */ var2 = 0 rx1= DEFCANIN 500 8 /* RX object with Id=500 and data length=8 bytes is defined */ CANIN rx1 0 0 var1 var /* a CAN message is read */

☐ DEFCANOUT

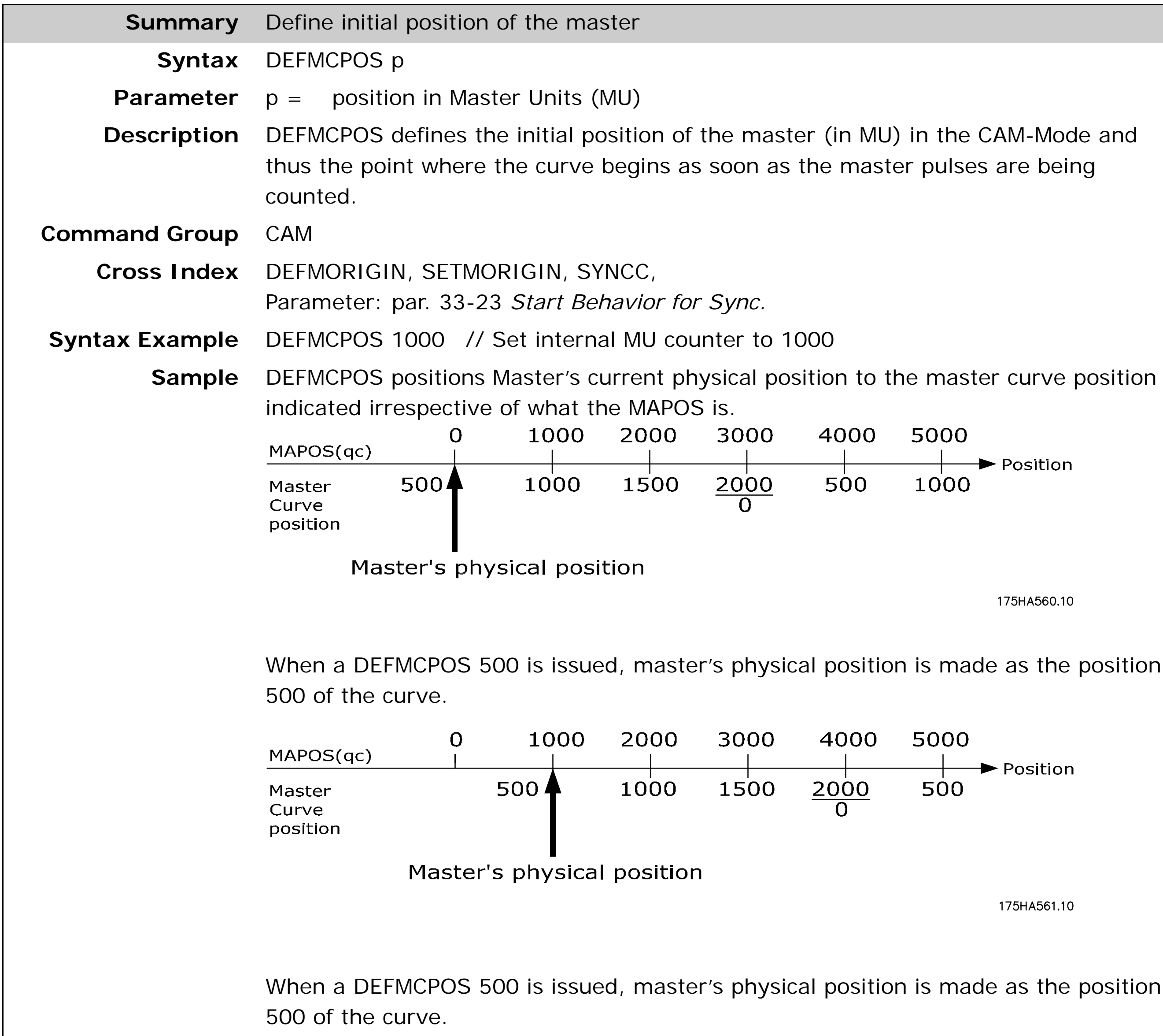
Summary	Defines a transmit object in the CAN controller.
Syntax	objno = DEFCANOUT id dlen
Parameter	id = CAN-identification number
	dlen = data length of the object in bytes (max. 8 bytes)
Return Value	objno
	A positive value means that an object was successfully created. This value is an internal number of the object and is used by other APOSS-CAN commands. A negative value indicates an error.
Description	This command defines an object in the CAN-Controller. This object is an outgoing object with a length of n bytes and the CAN identification of (id). Thus, objno is an internal number of the object Id and is used by the CANOUT command.
	This command can also be used with the offset for the slave bus (telegram ID 100000).
	So it is possible to define messages for slave and master bus. These objects can now be deleted one by one via the command CANDEL objno, where *objno* is the number returned by DEFCANIN or DEFCANOUT.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	DEFCANIN, CANOUT, CANIN
Syntax Example	no = DEFCANOUT 500 8 /* defines an object with Id=500 and length 8 bytes */ /* the function returns a 1 */ /* a message with Id=500 and a length of 8 bytes can now be sent */ /* with the command CANOUT 1 value1 value2 */ no = DEFCANOUT id len

□ DEFCORIGIN

Summary	Sets the command position as zero point.
Syntax	DEFCORIGIN
Description	DEFCORIGIN sets the command position as zero point. All absolute positioning commands (POSA etc.) refers to this zero point from now on. In doing so CPOS is set to zero and APOS is set in that way, that the difference is remained.
Portability	Command is available starting with MCO 5.00.
Command Group	INI
Cross Index	POSA, DEFORIGIN, CPOS
Syntax Example	POSA 80000 /* absolute positioning */ DEFCORIGIN X(1) /* define command position as zero point */

□ DEFMCPOS

Summary	Define initial position of the master
Syntax	DEFMCPOS p
Parameter	p = position in Master Units (MU)
Description	DEFMCPOS defines the initial position of the master (in MU) in the CAM-Mode and thus the point where the curve begins as soon as the master pulses are being counted.
Command Group	CAM
Cross Index	DEFMORIGIN, SETMORIGIN, SYNCC, Parameter: par. 33-23 *Start Behavior for Sync.*
Syntax Example	DEFMCPOS 1000 // Set internal MU counter to 1000
Sample	DEFMCPOS positions Master's current physical position to the master curve position indicated irrespective of what the MAPOS is.



When a DEFMCPOS 500 is issued, master's physical position is made as the position 500 of the curve.

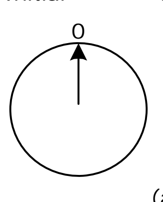
When a DEFMCPOS 500 is issued, master's physical position is made as the position 500 of the curve.

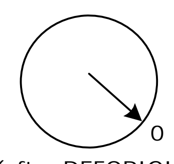
☐ DEFMORIGIN

Summary	Set the current master position as the zero point for the master.
Syntax	DEFMORIGIN
Description	DEFMORIGIN defines the current master position as the zero point for the master. The master position (MAPOS) refers to this zero point until a redefinition takes place using DEFMORIGIN or SETMORIGIN.
	NB!: The command DEFMORIGIN can not be used with absolute encoders (see par. 32-30 *Incremental Signal Type*).
Command Group	INI
Cross Index	MAPOS, SETMORIGIN
Syntax Example	DEFMORIGIN /* Set current position as the zero point for the master */

☐ DEFORIGIN

Summary	Sets the current position as zero point
Syntax	DEFORIGIN
Description	With the DEFORIGIN command the current position is set as the zero point. All absolute positioning commands (POSA) then refer to this zero point. The actual position reached in a positioning command is the Target position plus some error which is not compensated automatically while using a DEFORIGIN.
	NB!: The command DEFMORIGIN can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type*).
Command Group	INI
Cross Index	POSA
Syntax Example	POSA 80000 /* Absolute positioning */ DEFORIGIN /* define actual position as zero point */
Sample	POSA 2000 PRINT "Position before new origin", APOS DEFORIGIN PRINT "Position after defining new origin", APOS Output Position before new origin 2000, Position after defining new origin 0
Program Sample	DORIG_01.M, ORIG_01.M





◻ DEFSYNCORIGIN

Summary Defines master-slave relation for the next SYNCP or SYNCM command, or defines the start values for standard curves with SYNCC command.

Syntax DEFSYNCORIGIN mcpos spos

Parameter

mcpos	=	master reference position in qc, or master curve position in qc
spos	=	slave reference position, or
spos < SlaveCurveLenght	=	slave curve position
spos > SlaveCurveLength	=	the actual curve position has to be seen as the correct position within the curve corresponding to the master position *mcpos*

Description

This command defines how much distance ahead or behind should the slave be in relation to the master position. It allows defining the relation between master and slave for the next SYNCP or SYNCM command. It sets the internal slave command position (Mpcmd) to the slave value.

The master value is used for an internal MOVESYNCORIGIN. For that reason, a MOVESYNCORIGN will be overwritten by this command. Both actions are done at the moment, when the SYNC command is activated. So it is guaranteed, that master and slave will be synchronized at the above master-slave position.

With SYNCC the DEFSYNCORIGIN can be used to define the start values for standard curves as follows:

Here, *mcpos* tells the controller that the actual master position (MAPOS) corresponds to a master curve position of *mcpos*.

spos has two different meanings:

spos < SlaveCurveLenght	=	defines the actual position of the slave as the slave curve position *spos*.
spos > Slave CurveLength	=	the actual curve position has to be seen as the correct position within the curve corresponding to the master position *mcpos* (*spos* itself does not have any meaning, it just has to be greater than slave curve length).

Example: Assuming that the curve is a straight line going from 0,0 to 2000,4000 in terms of (master,slave).

Further assuming that the slave is at a position of 11000 qc and that the master has an actual position of 15000 qc. To make it simple, let us assume that the gear factors are 1 : 1.

If now a SETCURVE is done without any further commands, then the following would happen. The controller tries to find out where the slave is located within a curve. Therefore, it calculates the remainder of 11000 / 4000 which is 3000 and the remainder of 15000 / 2000 which is 1000. That means the result would be that the actual MCPOS (master curvepos) is 1000. So the corresponding slave curve position should be 2000. But at the moment the slave is at a curve position of 3000 instead. (The csstart position is the start position of the last curve, which would be 8000 in our case). If this curve is started, the slave would try correct the position and move to 2000.

If now the following command is used

```
DEFSYNCORIGIN 500 100000
```

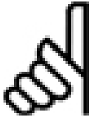
the following will happen:

The actual master position is defined as the master curve position of 500.

And because 100000 is bigger than the slave curve length of 4000, the actual slave position (of 11000 qc) equates to the slave curve position belonging to the master curve position of 500 which is 1000. So csstart will be set to 10000. If now the curve is started, it will be ok and just follow the curve when the master starts moving.

DEFSYNCORIGIN in conjunction with CU_GRAD curves (type 3)

In case of a CU_GRAD curve, the DEFSYNCORIGIN can be used to define the absolute end position of the curve in qc. In that case, the curve is started immediately and is calculated in such a manner that the end positions will be reached at the end of the curve.



NB!:
To avoid degenerate polynomials, the distances must be in a correct relation. If you start with velocity zero and want to end up with a velocity of 1 (slave has same velocity as master), then the master distance should be less than two times the slave distance. Otherwise, the polynomial will have extremes within the interval. This is more difficult to predict in other cases (start velocity not 0 or end velocity not 1). Therefore, you can check the PG_FLAG_CURVE_ERR to see if the last SETCURVE produced a curve with extremes (see STAT). Then you can read out the SYSVAR PFG_LASTERROR (see SDO dictionary) to decide what it was.

Portability Starting with MCO 5.00 start values for curves can be defined.

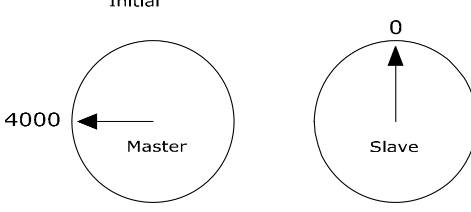
Command Group SYN

Cross Index MOVESYNCORIGIN

Sample Here when the master is in 2000 qc the slave should be in 4000 qc, i.e., slave should be ahead of master by 2000 qc.

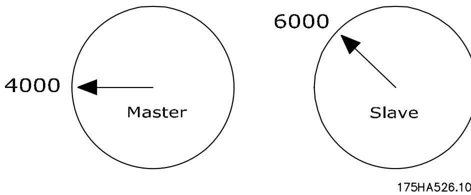
Also when the master is in 3000 qc the slave should be in 5000 qc.

Initial



Command; DEFSYNCORIGIN 2000 4000

The Slave is corrected to a position ahead of Master by 2000 qc



175HA526.10

◻ DELAY

Summary	Time delay
Syntax	DELAY t
Parameter	t = time delay in milliseconds (maximum MLONG)
Description	The DELAY command leads to a defined program delay. This parameter gives the delay time in milliseconds. If an interrupt occurs during the delay time, then following the processing of the interrupt procedure, the programmed delay will take place after the correct inclusion of the interrupt time. Thus, the DELAY command gives a constant delay time, independent of whether various interrupts have to be processed during the programmed delay time. If the interrupt requires more processing time than is available for the delay, then the interruption procedure will be carried out to the end, before the command following the DELAY instruction is commenced.
Command Group	CON
Cross Index	WAITT, WAITI, WAITAX
Syntax Example	DELAY 1000 /* 1 second delay */
Program Sample	DELAY_01.M

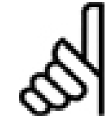
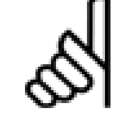
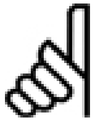
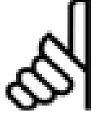
◻ DELETE ARRAYS

Summary	Delete all arrays in the RAM.
Syntax	DELETE ARRAYS
Description	With DELETE ARRAYS you can delete all arrays in the RAM without also deleting the parameters etc. This command has the same effect as the menu command *Controller → Reset → Arrays*.
	NB!: If you then execute a SAVE ARRAYS, the arrays in the EPROM are also overwritten!
	NB!: If DELETE ARRAYS is carried out after a DIM assignment in the program, it is then no longer possible to access the array elements.
	NB!: If a program contains a DELETE ARRAYS command, there are no more arrays in the RAM after the program is exited.
Command Group	INI

☐ DIM

Summary	Definition of an array
Syntax	DIM array [n]
Parameter	array = name of the array n = number of array elements
Description	Via a DIM instruction at the commencement of the program, it is possible to declare one or more arrays (= Variable fields). Arrays are valid for all programs. If arrays are not yet available in the MCO 305 memory, then the arrays are allocated via the DIM instructions. Arrays which are already available in the memory are checked to see if their size corresponds to the current DIM commands. If differences are found, then an error registration is made. If, additionally to the corresponding arrays, new arrays are declared, then these must also be added at the end of the DIM command. Each array element can later be accessed, similar to a variable, calculation results, characters or other information can be stored. An array element can be called up via the array name and an index. The indices are admissible from 1 to the defined size in the DIM allocation. An essential difference between variables and array elements consists in the fact that arrays are stored in the non-volatile memory, and their contents are permanent even when the power supply is switched off – insofar as it is saved with SAVEPROM or SAVE ARRAYS. In contrast to variables, arrays have a validity not only for one, but for all programs in the VLT unit flow. The only condition necessary is that the arrays must be accessible via a DIM command in the desired program which enables a data exchange between several programs. It is of no importance whether or not the array is identified with the same name in all the programs. What is important is the order of the array definitions. This means, for example, that the first defined array in all programs always refers to the first stored array in the memory, independent of the array name.
	NB!: The DIM command must be the first instruction in a program, and must appear before the subroutines. Indices from 1 to the defined size of the array are permissible. The array content will not be lost, even following switching off the power supply. A defined array size is valid for all programs, and cannot be altered. Only the order of the array definitions (not the names) determines which of the data-fields will be accessed. Array definitions can only be canceled via erasure of the entire memory.
Command Group	CON
Syntax Example	DIM xpos[100], ypos[100] /* define array XPOS and YPOS each with 100 elements */
Program Sample	DIM_01.M

☐ DISABLE ... interrupts

Summary	Locks the execution of interrupts.
Syntax	DISABLE inttyp
	**NB!:** DISABLE cannot be called up during interrupt procedures. (The system automatically switches back to enabled after an interrupt.)
Parameter	inttyp = ALL CANMSG COMBIT INT KEYPRESSED PARAM PERIOD position interrupts: ON APOS, ON IPOS, ON MAPOS, ON MCPOS, ON MIPOS STATBIT TIME
	**NB!:** The execution of error handling ON ERROR can not be locked with DISABLE. The error interrupt has the highest priority and it also interrupts other active interrupts.
Description	DISABLE switches off all or explicitly specified interrupts – apart from ON ERROR. If the function DISABLE ... is used in the main program, it can prevent interrupts of the corresponding type. This is particularly useful if a variable, which is set in an interrupt procedure, is used in the main program. To do this you should first switch off the corresponding (or all) interrupts in the main program DISABLE ... alter the variable and then switch the corresponding (or all) interrupts back on with ENABLE ...
	NB!: If an interrupt is disabled it still exist, but is not processed anymore (Exception: DISABLE ALL). The detection is still running in the background and the interrupt is captured in case of a non (!) edge-sensitive or a message-oriented interrupt (ON PERIOD, ON APOS, ON PARAM etc.). If the interrupt is ENABLEd again and there was a captured (non edge-sensitive) interrupt before, this interrupt is processed immediately. In case of edge-sensitive interrupts (e.g. ON INT, ON COMBIT, ON STATBIT), all interrupts, which take place during the DISABLEd phase are not processed, even not after switching on ENABLE again. These interrupts have no memory in case of DISABLEd state. Edge-sensitive interrupts which take place after the anew ENABLEing are still processed again.
	**NB!:** Exception: DISABLE ALL In opposite to the selective disabling of edge-sensitive interrupts (e.g. DISABLE INT) – these will be ignored and not executed after enabling – in case of DISABLE ALL the request is stored (edge-sensitive interrupts too) and the interrupt are still executed after enabling (ENABLE ALL)!
	DISABLE ALL in combination with selective DISABLE Please note, that an ENABLE ALL has no impact on simultaneous valid blockings defined by selective DISABLE commands (e.g. DISABLE INT). Thus a selective blocking must also be cleared by the corresponding selective ENABLE!

Interrupt handling within an Interrupt

During the execution of an interrupt subroutine at first a DISABLE ALL will automatically be done internally. This blocks the execution of all other interrupts, but keeps upcoming interrupt requests still in mind. At the end of the 'current' interrupt subroutine an ENABLE ALL will be again executed automatically. With the completion of the 'current' interrupt the upcoming stored interrupts will be executed yet. Therefore the execution of the commands DISABLE ALL and ENABLE ALL within an interrupt is not necessary and not meaningful, too.

The selective blocking of single interrupts within an interrupt subroutine can be necessary, depending on the application. Think of the following example: If the execution of an interrupt should lock the request and execution of other interrupt types, a selective DISABLE (e.g. DISABLE INT) can be done. In this case the selective interrupt blockage must be cleared (e.g. ENABLE INT) by the application program later on again. Typically this is done at the end of the current interrupt subroutine and enables the execution of corresponding interrupt requests in future again. All edge triggered interrupts, which were received between the corresponding selective DISABLE and ENABLE, will be ignored and not executed any longer (nor later). All interrupts, which were received before the selective blocking (e.g. DISABLE INT) or after the new selective release (e.g. ENABLE INT) will be processed after the completion of the "first" interrupt.

Command Group INT

Cross Reference ON INT, ON CANMSG, ON COMBIT, ON KEYPRESSED, ON STATBIT, ON PARAM, ON PERIOD, ON TIME, ENABLE .. Interrupts

Syntax Examples

```
DISABLE ALL        /* Switch off all interrupts */
DISABLE STATBIT    /* Switch off the interrupt for the status bit */
```

☐ ENABLE ... interrupts

Summary	Enables locked interrupts.
Syntax	ENABLE inttyp
Parameter	inttyp = ALL CANMSG COMBIT INT KEYPRESSED PARAM PERIOD position interrupts: ON APOS, ON IPOS, ON MAPOS, ON MCPOS, ON MIPOS STATBIT TIME
Description	ENABLE switches all or explicitly specified interrupts on again.
	NB!: ENABLE cannot be called up during interrupt procedures. (The system automatically switches back to enabled after an interrupt.)
	NB!: During the execution of an interrupt subroutine at first a DISABLE ALL and at the end an ENABLE ALL will automatically be done internally. Therefore the execution of the commands DISABLE ALL and ENABLE ALL within an interrupt is not necessary and not meaningful, too.
	NB!: Please see the command DISABLE .. interrupts for more details about interrupt blockings and how blocked interrupts are handled after the ENABLE command.
Command Group	INT
Cross Reference	ON INT, ON CANMSG, ON COMBIT, ON KEYPRESSED, ON STATBIT, ON PARAM, ON PERIOD, ON TIME, DISABLE .. Interrupts
Syntax Examples	ENABLE ALL /* Switch on all interrupts */
	ENABLE COMBIT /* Switch on the interrupt for the communication bit */

☐ ENCPOSOFFS

Summary	Syncs the incremental position counter with the absolute counter in the encoder.
Syntax	result = ENCPOSOFFS offset
Parameter	offset = Returns the difference between absolute and incremental position (absolute – incremental)
Return values	OK 0 The command was successful TIMEOUT -1 No answer has been received within 300ms BADFRAME -2 The received frame is not valid OVERFLOW -4 Received more bytes than the receive buffer can take
Description	The difference between the absolute encoder position and the incremental counter is determined and returned. For this, the incremental counter in the DSP is latched exactly at the moment where also the Hiperface encoder latches the absolute position which it sends back over RS485. With this difference, the user e.g. can set the position within APOSS to the absolute value with SETORIGIN. You can also use the MENCPOSOFFS command in case the Hiperface encoder is used as master signal instead of slave signal (see parameter 32-50). Motor feedback: The motor feedback signal is generated by the incremental signal. Often, the Hiperface encoders will be used with a PM motor. For PM motors, it is necessary to know how the absolute rotor position is. The rotor position relative to the absolute encoder position must be determined once during the commissioning of the system (or sometimes, it is also saved in the encoder). The offset will then be saved in a control card parameter (Par. 1-41). After a power cycle, the incremental signal (which is used for motor feedback) must be synced to the absolute position again (see program sample).
Command Group	SYS
Cross Reference	MENCPOSOFFS
Program Sample	(see below)

```
MOTOR OFF
SET ENCODERTYPE 0          // no incremental encoder is connected
SET ENCODERABSTYPE 1       // Hiperface encoder
SET ENCODEABSRES 4096      // Hiperface resolution
DELAY 1000
pos = 0
RSTORIGIN
offset = 0
PRINT "apos before: ", apos
retval = ENCPOSOFFS offset
PRINT "encposoffs returned: ", retval, " offset is: ", offset
SETORIGIN -offset
PRINT "apos afterwards: ", apos
WHILE(1) DO
PRINT apos                 // print incremental position
DELAY 500
ENDWHILE
```

ENCTGREAD

Summary	Reads a RS485 telegram from the encoder.
Syntax	result = ENCTGREAD array
Parameter	array = The user array where the received payload data should be put.
Return values	OK x (>0) TG has arrived with x bytes user data ACTIVE 0 The transmission is still ongoing TIMEOUT -1 No answer has been received within 300ms BADFRAME -2 The received frame is not valid OVERFLOW -4 Received more than the receive buffer can take
Description	After a telegram has been sent with ENCTGWRITE, the answer can be polled by this command. The return value will show if it has already arrived or if a timeout has occurred. You can also use the MENCTGREAD command in case the Hiperface encoder is used as master signal instead of slave signal (see parameter 32-50)
Command Group	SYS
Cross Reference	MENCTGREAD, ENCTGWRITE
Program Sample	see below

Program Sample

```
// Example program to receive absolute position
DIM sendbuffer[20]
#define HIPER_READ_POS 0x42
SET ENCODERTYPE 0           // no incremental encoder is connected
SET ENCODERABSTYPE 1        // hiperface encoder
SET ENCODEABSRES 4096       // hiperface resolution
DELAY 1000
pos = 0
WHILE(1) DO
sendbuffer[1] = HIPER_READ_POS
retval = ENCTGWRITE 1 sendbuffer   // send telegram
DELAY 1000
retval = ENCTGREAD sendbuffer      // receive answer
   // check if correct amount of bytes has been received
IF(retval == 7) then // 0x40 0x42 pos0 pos1 pos2 pos3 crc
pos.b4 = sendbuffer[3]
pos.b3 = sendbuffer[4]
pos.b2 = sendbuffer[5]
pos.b1 = sendbuffer[6]
PRINT "Pos = ", pos
ELSE
PRINT "-------- Transmission error ------------: ", retval
PRINT "1: ", sendbuffer[1]
PRINT "2: ", sendbuffer[2]
PRINT "3: ", sendbuffer[3]
PRINT "4: ", sendbuffer[4]
EXIT
ENDIF
DELAY 500
ENDWHILE
```

☐ ENCTGWRITE

Summary	Sends a RS485 telegram to the encoder.
Syntax	result = ENCTGWRITE length array
Parameter	length = The number of bytes (in the user array) to be sent.
	array = The user array containing the payload data to send to the encoder.
Return values	OK 0 Telegram has been sent
	BUSY -3 There is still another transmission ongoing and not timed out yet
Description	This command will send a RS485 telegram to the encoder with the ID "ENCODERID". The user has to fill the payload data into an array before. The command will then put this data into a regular RS485 frame and add CRC value to it.
	The command does not wait till the data has been sent or an answer is received, it returns immediately.
	The answer of the telegram has to be polled with ENCTGREAD
	You can also use the MENCTGWRITE command in case the Hiperface encoder is used as master signal instead of slave signal (see parameter 32-50).
Command Group	SYS
Cross Reference	ENCTGREAD, MENCTGWRITE
Program Sample	See program sample ENCTGREAD command.

☐ ERRCLR

Summary	Error cancellation
Syntax	ERRCLR
	The ERRCLR command should only be used in a subroutine for error handling (see ON ERROR GOSUB).
	NB!: ERRCLR contains the command MOTOR ON, which automatically turns on the control again. (The motor is position controlled at the current position.)
Description	An option card error can be cleared via the ERRCLR command. However, the cause of the error must be eliminated first; otherwise the same error alarm will occur again. If, in the meantime, another un-corrected error occurs, then only the first error will be canceled.
	ERRCLR also resets FC 300 alarms by means of Bit 7 of the control word.
Command Group	INI, CON
Cross Index	ON ERROR GOSUB, ERRNO, CONTINUE, MOTOR ON, Warnings and Error Messages
Syntax Example	ERRCLR /* erase actual error alarm */
Program Sample	ERROR_01.M, IF_01.M, INDEX_01.M

☐ ERRNO

Summary	System variable with the actual error code
Syntax	res = ERRNO
Description	ERRNO is a system variable which is available in all the programs, and contains the momentary error code. All error codes are explained in the chapter Troubleshooting. If, at the time of inquiry no error has occurred, then ERRNO will contain a 0.
Portability	Standard variable
Command Group	SYS
Cross Index	ON ERROR GOSUB, ERRCLR, Warnings and Error Messages
Syntax Example	PRINT ERRNO /* display actual error code */
Program Sample	ERROR_01.M, IF_01.M, INDEX_01.M

☐ EXIT

Summary	Premature program termination
Syntax	EXIT
Description	The EXIT command ends a program where active positioning procedures are being carried out to the end. The EXIT command is especially intended for use in an error treatment routine, and permits an unplanned program termination in the case of an un-correctable error occurrence. After an abort with EXIT, programs marked with *Autostart* will start up again automatically if SET PRGPAR = -1.
	NB!: A program should only be terminated in the case of a serious error, e.g. when reacting to a limit switch.
Command Group	CON
Cross Index	ON ERROR GOSUB, SET, Parameter: 33-80 *Activated Program Number* PRGPAR, Autostart
Syntax Example	EXIT /* Program termination */
Program Sample	EXIT_01.M, ERROR_01.M

◻ GET

Summary	Reads a parameter
Syntax	res = GET par
Parameter	par = parameter identification
Return Value	res = parameter value
Description	Reads the value of a parameter, a MCO 305 parameter, or an application parameter. Parameters are addressed with a code, for example KPROP for the *Proportional Factor,* or POSERR for the *Tolerated Position Error.* A complete list of the codes can be found in the Parameter Reference. Application parameters are addressed with a number of group 19-**. See also the parameter reference for details.
Command Group	PAR
Cross Index	SET, GETVLT, SETVLT, LINKGPAR, Parameter Reference
Syntax Example	PRINT GET POSLIMIT /* Print-out positive positioning limit */ posdiff = GET POSERR /* Read actual setting tolerated position error */ PRINT GET I_FUNCTION_9_4 /* reads input 4 for abort */
Program Sample	GETP_01.M

☐ GETVLT

Summary	Reads a VLT parameter
Syntax	res = GETVLT par
Parameter	par = parameter number
Return Value	res = parameter value
Description	GETVLT reads parameters and return the corresponding value. Thus, with GETVLT you have access to the operating data (e.g. motor current 1-24) or to the configurations (e.g. max. reference par. 3-03) of the FC 300. Since only integer values are transmitted, it is necessary to take the conversion index into consideration when evaluating the return value. Thus an LCP value of 50.0 Hz (par. 16-13 conversion index = –1) is equivalent to a return value of 500. The list of FC 300 parameters with their respective conversion index can be found in the FC 300 Operating Instructions.
	NB!: Use GETVLTSUB to read parameters with index numbers, for example FC 300 parameter 5-40.
Command Group	PAR
Cross Index	SETVLT
Syntax Example	PRINT GETVLT 413 /* reads par. 4-13 motor speed high limit */

☐ GETVLTSUB

Summary	Reads a VLT parameter with index number
Syntax	res = GETVLTSUB par indxno
Parameter	par = parameter number indxno = index number
Return Value	res = parameter value
Description	GETVLTSUB reads VLT parameters with index numbers, for example FC 300 parameter 5-40, and return the corresponding value. Since only integer values are transmitted, it is necessary to take the conversion index into consideration when evaluating the return value. Thus an LCP value of 50.0 Hz (par. 16-13 conversion index = –1) is equivalent to a return value of 500. The list of FC 300 parameters with their respective conversion index can be found in the FC 300 Operating Instructions.
Command Group	PAR
Cross Index	SETVLTSUB
Syntax Example	PRINT GETVLTSUB 540 0 // reads index 01 of the parameter 5-40 "Function Relay"

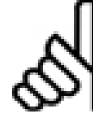
◻ GOSUB

Summary	Calls a subroutine
Syntax	GOSUB name
Parameter	name = subroutine name
Description	The GOSUB command will call up a subroutine, and the accompanying program will be carried out. The main program will be continued following the completion of the last subroutine command (RETURN). **NB!:** The subroutine must be defined at the beginning or end of a program within the SUBMAINPROG area.
Command Group	CON
Cross Index	SUBMAINPROG .. ENDPROG, SUBPROG .. RETURN, ON ERROR GOSUB .., ON INT n GOSUB
Syntax Example	GOSUB testup /* Call-up the subroutine testup */ Command line Command line SUBMAINPROG /* Subroutine testup must be defined */ SUBPROG testup Command line 1 Command line n RETURN ENDPROG
Program Sample	GOSUB_01.M, AXEND_01.M, INCL_01.M, STAT_01.M

◻ GOTO

Summary	Jump to a program label
Syntax	GOTO label
Parameter	label = identification of program target position
Description	The GOTO command enables an unconditional jump to the indicated program position and the program processing at this position will be carried out. The jumped-to position is identified with a label. A label can consist of one or more characters and may not be identical to a variable name or a command word. A label must also be unique, i.e. it may not be used for different program positions. It is therefore possible to program a continuous loop via the GOTO command. **NB!:** The label for the program target position must be followed by a colon (:).
Command Group	CON
Cross Index	LOOP
Syntax Example	endless: /* label to be jumped to */ command line 1 command line n GOTO endless /* jump command to label endless */
Program Sample	GOTO_01.M, EXIT_01.M, IF_01.M

◻ HOME

Summary	Move to device zero point (reference switch) and set as the real zero point.
Syntax	HOME
Description	The HOME command is moving the drive to the machine reference switch, which must be placed at the machine zero or reference position. Velocity and acceleration/deceleration for HOME positioning is defined in the parameters 33-03 *Velocity for Home Motion* and 33-02 *Ramp for Home Motion*. To achieve accurate positioning *Velocity for Home Motion* should not be higher than 10% of maximum velocity. The sign of par. 33-03 *Velocity for Home Motion* determines the direction in which the reference switch is searched. When the HOME position is reached, this position will be defined as 0. The reference switch can be approached in 4 different ways defined in par. 33-04 *Behavior during Home Motion*: 0 = Moves to reference switch, moves in opposite direction leaving the references switch and stops at the next index pulse (encoder zero pulse or external marker). 1 = Like 0 but without searching for the index pulse. 2 = Like 0 but leaving the switch without reversing the direction. 3 = Like 2 but without searching for the index pulse. If HOME is aborted via an Interrupt, HOME will not be continued automatically at the end of the interrupt routine function. Instead the program continues with the next command. This makes it possible for HOME to also be aborted after an error.
	**NB!:** The system must be fitted with a reference switch, when possible with an encoder with an index pulse.
	NB!: The HOME command will also be carried out to the end in the NOWAIT ON mode, before other program processing will be begun. Please note that ON PERIOD xx GOSUB xx must be disabled during homing. E.g. ON PERIOD n GOSUB x and the resetting after homing is completed.
	NB!: The command HOME can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type*).
Command Group	INI
Cross Index	INDEX, NOWAIT Parameters: 33-03 *Velocity for Home Motion*, 33-02 *Ramp for Home Motion*, 33-00 *Force HOME*
Syntax Example	HOME /* move to reference switch and index */
Program Sample	HOME_01.M

◻ IF ..THEN .., ELSEIF .. THEN .. ELSE .. ENDIF

Summary	Conditional single or multiple program branching. (When the conditions are fulfilled, then ..., else ...)
Syntax	IF condition THEN command ELSEIF condition THEN command ELSE command ENDIF
Parameter	condition = Branching criteria command = one or more program commands
Description	Conditional program branching can be realized with the IF..ENDIF construction. When the conditions following IF or ELSEIF are fulfilled, then the commands leading to the next ELSEIF, ELSE or ENDIF are carried out – and the program will be continued after the ENDIF instruction. When the conditions are not fulfilled, then the following ELSEIF branching will be checked and, in as much as the conditions are fulfilled, the corresponding program part will be carried out, and the program continued after ENDIF. The branching conditions that are checked after IF or ELSEIF can be made up of one or more comparison operations. Any number of ELSEIF branching can occur within an IF...ENDIF construction – however, only one ELSE instruction should be available. Following the ELSE instruction is a program part that must be carried out, in as much as none of the conditions are fulfilled. The ELSEIF and ELSE instructions can, but do not have to be, contained within an IF ENDIF construction.
	NB!: After a condition has been fulfilled, the appropriate program part will be carried out and the program following the ENDIF instruction continued. Further conditions will no longer be checked.
Command Group	CON
Cross Index	REPEA T .. UNTIL, WHILE .. ENDWHILE

Syntax Example	/* simple branch */
	IF (a == 1) THEN /* Variable a = 1, then */
	    command line 1
	    command line n
	ENDIF
	/* multiple branch */
	IF (a == 1 AND b != 1) THEN
	    command lines
	ELSEIF (a == 2 AND b != 1) THEN
	    command lines
	ELSEIF (a == 3) THEN
	    command lines
	ELSE
	    command lines
	ENDIF

Program Sample	IF_01.M, ERROR_01.M, EXIT_01.M, HOME_01.M, IN_01.M, ...

◻ IN

Summary	Reads status of digital input
Syntax	res = IN n
Parameter	n = input number 1 – 10 or 1 – 12 (option inputs) 18, 19, 27, 29, 32, 33 or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + input number (or input byte)
Return Value	res = input status 0 = Low-level or undefined 1 = High-level
Description	The status of a digital input can be read with the IN command. Depending on the signal level, a 0 or 1 will be given. The selection of the mode for input 11,12 is done by par. 33-60 IOMODE. The definition of a high or low level, as well as the input circuit, can be taken from the–"Operating Instructions", as well as from the FC 300 manual. The inputs 5 and 6 are also used as marker inputs for the master and slave encoders. CAN modules which fulfill the CAN OPEN specifications can also be addressed with the IN command via the corresponding number defined as follows: CAN-Bus + (Module-CAN-ID * 256) + input number (or input byte) When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules, but for the moment there are no objects there which are ready to receive, for example for interrupt functions. In order to execute interrupt functions you have to initialize the corresponding module with CANINI beforehand.
Portability	The parameters to use CANopen are available starting with MCO 5.00.
Command Group	I/O
Cross Index	INB, OUT, OUTB, CANINI Parameters: 33-60 *Terminal X59/1 and X59/2 Mode*, IOMODE, 33-50...59,61,62 *Terminal X57/n Digital Inputs*, I_FUNCTION_n

Syntax Example	in4 = IN 4 /* store condition input 4 in variable in4 */
	IF (IN 2) THEN /* if high level on terminal 2, set output 01 */
	OUT 1 1
	ELSE
	OUT 1 0
	ENDIF

Program Sample	IN_01.M

◻ INAD

Summary	Reads analog input or process data objects (PDO) of CAN objects
Syntax	res = INAD n
Parameter	n = a) number of the analog input: 53,54 b) with CAN applications: Module number * 256 + I/O number
Return Value	res = analog value a) Terminal 53/54: -1000 – 1000 = -10 V – 10 V Terminal 53/54: 0 – 10 V res = 0 – 100 b) The range of values depend on the analog input used.
Description	The INAD command reads the value of the analog inputs. CAN modules which fulfill the CAN open specifications can also be addressed with the INAD command via the corresponding Module number * 256 + the I/O number. When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules.
	NB!: The CAN commands operate with the pre-defined PDOs of CAN-OPEN. Do not change these default settings (minimum capability device), otherwise the CAN commands will not operate anymore.
Portability	The function Read PDO is available starting with MCO 5.00.
Command Group	I/O
Cross Index	CANINI, Operating Instructions MCO 305 and FC 300
Syntax Example	an1 = INAD 53 PRINT "Analog input 53 " ,an1

☐ INB

Summary	Reads one byte from digital inputs
Syntax	res = INB n
Parameter	n = input byte: 0 = input 1 (LSB) - 8 (MSB) 1 = input 33 (LSB) - 18 (MSB) 2 = input 9 - 10 (12) or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + input number (or input byte)
	NB!: Numbering of the bytes begins with 0; this is in contrast to the numbering of the individual inputs, which starts with 1.
Return Value	res = value of the input byte (0 - 255)
	The least significant bit corresponds to the condition of input 1/33.
Description	The condition of the digital inputs can be read as a byte via the INB command.
	The values reflect the condition of the individual inputs.
	The definition of the high and low level, as well as the input circuit, can be taken from the FC 300 manual.
	CAN modules which fulfill the CAN OPEN specifications can also be addressed with the INB command via the corresponding number, which is defined as follows:
	CAN-Bus + (Module-CAN-ID * 256) + input number (or input byte)
	When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules, but for the moment there are no objects there which are ready to receive, for example for interrupt functions. In order to execute interrupt functions you have to initialize the corresponding module with CANINI beforehand.
Command Group	I/O
Cross Index	IN, OUT, OUTB, FC 300 Operating Instructions
Syntax Example	in = INB 0 /* store the condition of the first 8 inputs */
Example	IN1 = low, IN2 = high, IN3 = high,
	all other inputs are low
	res = 2^1 + 2^2 = 6
Program Sample	INB_01.M, INB_02.M, OUTB_01.M

◻ INDEX

Summary	Move to index position of the encoder
Syntax	INDEX
Description	Movement to the index position of the encoder will be started via the INDEX command. The Index search takes places with the *Velocity for Home Motion* defined in par. 33-03. The *Velocity for Home Motion* sign determines the rotational direction in which the Index signal will be searched.
	NB!: The utilized encoder must have **! ! !** an index channel.
	NB!: Only encoders with a low active index pulse can be used. If an index pulse is not found within a complete revolution, then an alarm signal occurs. The INDEX command will also be carried out to the end in NOWAIT ON, before further program processing can be begun.
	NB!: The command INDEX can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type*).
Command Group	INI
Cross Index	HOME, POSA, DEFORIGIN, NOWAIT
Syntax Example	INDEX /* move to index */
Program Sample	INDEX_01.M

◻ INGLB

Summary	Reads a global CAN message via CAN bus.
Syntax	res = INGLB (p)
Parameter	p = time-out, defined ... p = 0 waits until a message arrives p > 0 waits a maximum of p milliseconds p < 0 does not wait for a message
Return Value	res = –1, if no message has come or bytes 2 and 3 of the CAN message, if a message has been received. The global variable MSGVAL then contains the upper bytes 4 to 7 of the CAN message.
Description	This command reads a global CAN message, i.e. a message which is sent to all CAN devices on the bus. These messages have the identifier 0 and thus have highest priority.
	NB!: This message is not buffered and thus will be written over when the next message arrives.
Portability	Standard command
Command Group	CAN
Cross Index	INMSG, OUTMSG

☐ INKEY

Summary Reads in a key signal.

Syntax INKEY (p)

Parameter p is the maximum waiting time, defined . . .

- $p = 0$ wait for key code
- $p > 0$ wait of max. p milliseconds
- $p < 0$ no wait for key code
  (a negative parameter must be given in brackets)

Return Value key code for the received character or –1 in case no character available

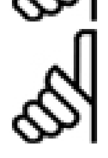
Following key codes are sent back, as long as the key is pressed. If more than one key were pressed simultaneously the corresponding sum of the values will be sent back:

key:	value:
[Main Menu]	1
[Quick Menu]	2
[Alarmlog]	4
[Status]	8
[OK]	16
[Cancel]	32
[Info]	64
[Back]	128
[→]-key / right	256
[↑]-key / up	512
[↓]-key / down	1024
[←]-key / left	2048
[Auto on]	4096
[Reset]	8192
[Hand on]	16384
[Off]	32768

Combinations send the corresponding values:

[OK] and [Cancel]	48
[Auto on] and [↑]-key	4608

NB!:
The keys keep their FC 300-functions, unless they are disabled in parameter 0-4*.



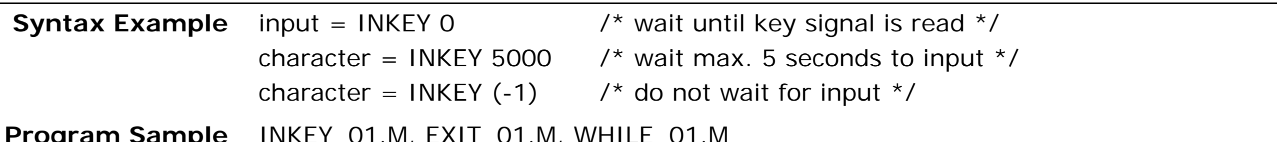
NB!:
NLCP is not covered at the moment.

Description With the INKEY command it is possible to read a key signal from the keypad of the FC 300 LCP. The parameter entered with INKEY determines whether the program waits unconditionally for a key signal, for a certain period of time or not at all.

One key signal is read in per successful INKEY command respectively. To input a string of characters it is necessary to repeat the INKEY command (p<>0) in a loop until no further key signals exist.

Command Group I/O

Cross Index PRINT



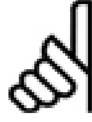
Syntax Example	input = INKEY 0 /* wait until key signal is read */ character = INKEY 5000 /* wait max. 5 seconds to input */ character = INKEY (-1) /* do not wait for input */
Program Sample	INKEY_01.M, EXIT_01.M, WHILE_01.M

◻ INMSG

Summary	Read CAN message from the buffer.
Syntax	intval = INMSG time-out
Parameter	time-out < 0 does not wait for data = 0 waits until data arrives > 0 waits for data in time-out [ms]
Return Value	INMSG returns –1, if no message has arrived or bytes 2 and 3 of the CAN message if a message has arrived. The global variable MSGVAL contains the upper bytes 4 to 7 of the CAN message.
Description	This command reads a message from the buffer, with *time-out* having an analog meaning such as with the INKEY command. The message has an Id, which is defined with the command "N_slaveno_baudraute" (see also $N command). The CAN identification number of the message is determined by the $N command. INMSG always reads objects which are 8 bytes long. Only bytes 2 to 7 are intended for the user; bytes 0 and 1 are reserved.
Portability	Standard command
Command Group	CAN
Cross Index	OUTMSG, ON CANMSG

Syntax Example	a = INMSG –1
	IF (a > –1) THEN
	b = MSGVAL
	ENDIF

□ IPOS

Summary	Queries last index or marker position of the slave
Syntax	res = IPOS
Return Value	res = last slave position (index or marker) absolute to actual zero point. The position input is made in user units (UU) and corresponds in the standard setting (parameter 32-12 *UU Numerator* and 32-11 *UU Denominator* = 1) to the number of qc.
Description	The command IPOS returns the last index or marker position of the slave absolute to the current zero point.
	NB!: If a temporary zero point, set and activated via SETORIGIN, exists, then the position is respective to this zero point
	The configuration of IPOS, that is whether the slave index- or marker position (= controlled drive) is returned, is done via the par. 33-20 *Slave Marker Type*.
	NB!: The trigger signal for the marker position has to be connected mandatory to the input 6.
	The position value in IPOS is accurate to +/- 1qc. In opposite to the position information in APOS, which is just updated in a controller cycle of typically 1 ms, the actual position value is hardware stored in real time a buffer (in an internal processor register), when the configured signal is high. Then it will be copied in the system variable IPOS. If simultaneously to the marker position an interrupt is initiated (ON INT 6 GOSUB ...) and within this interrupt it is operated with IPOS, you should use before IPOS reading a delay of 2 milliseconds (DELAY 2) within the interrupt subroutine. So it can be ensured, that the latched position value is already complete copied in the system variable IPOS and that not be taken an old value. See also sample.
	**NB!:** The command IPOS can not be used: – with absolute encoders (see par. 32-00 *Incremental Signal Type*) – when Parameter 32-50 is set to [3] – Motor Control.
Command Group	SYS
Cross Index	CPOS, DEFORIGIN, SETORIGIN, POSA, POSR, MIPOS, ON INT Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*, 33-20 *Slave Marker Type*
Syntax Example	PRINT IPOS /* queries last index position and display on PC */
Sample	ON INT 6 GOSUB slave_int // Definition interrupt handler SET SYNCMTYPS 2 // Definition of IPOS latching on positive edge at input 6 CVEL 10 // Start moving CSTART // Endless-Loop mainloop: // ... GOTO mainloop SUBMAINPROG

```
SUBPROG slave_int
    int_pos = APOS       // Latching APOS for testing, how exact it would be ...
    DELAY 2              // Wait 2 ms, to be sure, that IPOS is correct updated
    triggered_pos = IPOS        // Latching IPOS for a later handling etc.
                         // ....
                         // ...
    PRINT "Interrupt position: ",int_pos
    PRINT "Triggered position: ",triggered_pos
    RETURN
ENDPROG
```

□ IPOSDIFF

Summary	Overflow handling of incremental encoders in applications.
Syntax	res = IPOSDIFF oldpos
Parameter	oldpos = IPOS at a previous time
Return Value	Returns difference between IPOS and oldpos (res = IPOS – oldpos) in UU
Description	This command simplifies overflow handling of incremental encoders in applications. If, for example, the user stores an actual position in his program and wants to calculate the difference at a later time, then he normally has to account for overflow of the position. Instead this command can be used: see below. Internally those routines look if the difference is bigger than POS_LIMIT (0x3FFFFFFF). If so then it is assumed that an overflow happened and it is handled correctly.
	NB!: This will not solve the problem of overflowing if the application uses user units.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	IPOS

Syntax Example	oldpos = IPOS
	.....
	diff = IPOSDIFF oldpos
	// this function returns the difference between IPOS and oldpos in user units
	// handling an overflow if necessary (diff = IPOS – oldpos)

□ JERKFINVEL

Summary	Calculates the final velocity for a jerk-limited stop with maximum acceleration/deceleration.
Syntax	res = JERKFINVEL
Return Value	res = in percent (or VELRES units)
Description	The command calculates the final velocity which would be reached if the actual acceleration (or deceleration) stops under consideration of the given JERKMIN values. This also works if some movement other than a RAMPTYPE 2 movement is active. So you can calculate the final velocity if you want to leave your actual SYNCP, for example. The result is in percent (or VELRES units) so you can use it without conversion for a VEL or CVEL command.
	The result could also be negative (e.g. driving backwards, or driving near the velocity zero with a high deceleration).
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Syntax Example	Par. 32-86 *Acc. up for limited jerk* JERKMIN, par. 32-82 RAMPTYPE

□ JERKSTOPDIST

Summary	Calculates the necessary distance for a jerk-limited stop with maximum deceleration.
Syntax	res = JERKSTOPDIST dec
Parameter	dec = sets deceleration in % or VELRES units
Return Value	res = distance in UU
Description	The command calculates the distance that axis n will need to stop if it allows a maximum deceleration of DEC and uses the RAMPTYPE 2 with JERKMIN2 and JERKMIN4 (or JERKMIN). In the command, the deceleration DEC is given in percent (or VELRES units) and the result is given in User-Units. This command also works if you are not in a RAMPTYPE 2 movement. So you can calculate a RAMPTYPE 2 stop-ramp if you are in SYNCP, for example.
Portability	Command is available starting with version MCO 5.00.
Command Group	SYS
Cross Index	DEC, par. 32-83 VELRES, par. 32-86 *Acc. up for limited jerk*, par. 32-87 *Acc. down for limited jerk*, par. 32-89 *Dec. down for limited jerk*

◻ LINKGPAR

Summary	Links global parameter or parameter groups with LCP display
Syntax	LINKGPAR parno "text" min max option
Parameter	parno = LCP parameter number (group 19-00 to 19-99) text = descriptive text for display; only ASCII text (8-bit) is supported min = minimum value for this parameter max = maximum value for this parameter option = type of parameter 0 = offline, i.e. changes are only active after they have been confirmed with [OK]. 1 = online, that means changes via the LCP display are active at once.
Description	With LINKGPAR free internal application parameter can be linked with the LCP display. Subsequently it is possible to change this parameter via the LCP or read out the set value. When a linked parameter is changed with a SET command, the new value is also automatically transferred to the LCP, but is not changed in the default settings since the SET command only has a temporary effect. If the user changes a linked parameter on the LCP, the new value is executed. Only after the user has confirmed this value with OK is the new value saved permanently as an application parameter in the EPROM. The command LINKGPAR tests whether the value of the application parameter is within the specified range. If not, the corresponding limit is used and this value saved. This ensures that a display appears.
Command Group	PAR
Cross Index	SET, GET, Application parameter, Parameter Reference
Syntax Example	LINKGPAR 1901 "name" 0 100000 0 /* Link par. 19-01 with LCP display */

◻ LINKPDO

Summary Mapping of RxPDOs: Link the content of a RxPDO to elements of the internal system variable pseudo array SYSVAR. Each change of the RxPDO is copied automatically into the configured SYSVAR elements afterwards.

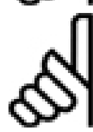
Syntax LINKPDO no len indx pdo

Parameter

no	=	rank order in the RxPDO, beginning with 1
len	=	number of the bits to be imported; requirement: length = a multiple of 8 (bit-by-bit), (e.g. 128 to copy 4 long values into the PDO, if PDO holds more than 8 bytes)
indx	=	index of the system variable SYSVAR
pdo	=	1 - 4 or 5 (“serial” PDO 5) All values differing from the valid range, are handled like value 1 (even '0'). This assures downward compatibility with old versions, which used 0 as default value.

NB!:

The rank order number (no) must – beginning with 1 – increase.

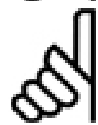


Multiple PDOs

The command can be used for any PDOs, i.e. that PDO 2 - PDO 5 are also supported, if an up-to-date firmware and compiler version is in use. The PDO number is defined by the last parameter (which was reserved by older firmware versions) of the LINKPDO command. Backward compatibility is given by the fact, that each PDO number out of range is handled like 1, i.e. that PDO 1 is default. This guarantees, that older code using 0 as the last parameter by default behaves the same and defaults to PDO 1 now.

CANopen PDO size

A CANopen PDO is always 8 bytes long; it can therefore hold a maximum of 8 objects.



PDO 5 (= “serial PDO”) Size

The mailbox size of the PDO 5 can be up to approx. 250 Bytes. The PDO 5 is also used by the oscilloscope tool of the APOSS development environment, therefore it is recommended to use this PDO not in applications, that shall be debugged using the oscilloscope tool later on.

Description The command LINKPDO links the content of RxPDOs to elements of the system variables pseudo array SYSVAR. This is called PDO mapping. Each change of RxPDO data is copied automatically into the configured SYSVAR elements.

The system variables pseudo array SYSVAR holds internal data and variables, as well as each SDO (according to the SDO object dictionary), which also means the first 250 elements of each application array are included. The RxPDO data can be directed to almost any variable, array or parameter by this. The SYSVAR index of a SDO can be calculated with the following formula:

0x01000000 + ("SDO index" << 8) + "SDO subindex"

Example 1: SDO 0x2300 / 12 (= SDO holding axis parameter KPROP of axis 1)
=> SYSVAR index = 0x0123000C

Example 2: SDO 0x2100 / 5 (= SDO with the first element of the first application array)
=> SYSVAR index = 0x01210001

	The RxPDO is also copied into the PDO array. The same data content can be accessed by reading the PDO array. Automatic PDO activation Just the PDO 1 (RxPDO = 0x200 + Node-ID / TxPDO = 0x180 + Node-ID) is enabled by default according to the CANopen specification, i.e. the "Valid" bit (0x1400 / Subindex 1) is set. If the mapping is configured for other PDOs using LINKPDO (or LINKSDO), then the "Valid" bits (0x1401-0x1404, Subindex 1) of these RxPDOs are also set automatically. CANopen versus APOSS mapping If the SYSVAR index refers to a SDO of the SDO object dictionary (i.e. SYSVAR index starts with 0x01...), then a pure CAN mapping is internally executed. When a corresponding object is changed by a SDO command, the PDO is also immediately rewritten. The mapping for the corresponding CAN object can be read-out by a supervisor control unit. If other SYSVAR indices are used, an APOSS mapping is carried out. This can be combined with the CAN Mapping, but does not conform to CANopen, because the correct map entries can not be read out by the CANopen mapping objects.
	NB!: The linking of internal system variables has to be accomplished very carefully und should only be done from experienced APOSS users. Thorough knowledge about the usage und meaning of the internal system variable is necessary, not to cause an incorrect system behavior.
Portability	Command is available starting with MCO 5.00.
Command Group	PAR
Cross Index	LINKSDO, SYSVAR, Parameter Reference, PDO
Syntax Example 1	// Link RxPDO 1 to user parameter 1 (= SDO 0x2201/01) LINKPDO 1 32 0x01220101 0

Syntax Example 2	// Link 8 bits of RxPDO 1 to digital outputs 1 - 8 (= SDO 0x2202/10)
	LINKPDO 1 8 0x0122020A 1
	// Link next 8 bits of RxPDO 1 to outputs 9 - 16 (= SDO 0x2202/11)
	LINKPDO 2 8 0x0122020B 1

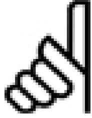
Syntax Example 3	// Link 16 bits of RxPDO 1 into DS402 control word (= SDO 0x6040/0) LINKPDO 1 16 0x01604000 1

☐ LINKSDO

Summary	Mapping of the TxPDOs: Link elements of the internal system variable pseudo array SYSVAR to a TxPDO. Each change of the corresponding SYSVAR element is forwarded to the TxPDO automatically afterwards.
Syntax	LINKSDO indx len no "text" pdo
Parameter	indx = Index of the system variable SYSVAR len = Length of the bits to be imported; requirement: length = a multiple of 8 (bit-by-bit) (e.g. 128 to copy 4 long values into the PDO, if PDO holds more than 8 bytes) no = Rank order in the PDO text = " " (has not yet been evaluated; however, can be uses as comment) pdo = values between 1 - 4 or 5 ("serial" PDO 5)

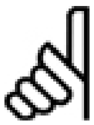
NB!:

The rank order number (no) must – beginning with 1 – increase.



Multiple PDOs

The command can be used for any PDOs, i.e. that PDO 2 - PDO 5 are also supported, if an up-to-date firmware and compiler version is in use. The PDO number is defined by the last parameter (which was reserved by older firmware versions) of the LINKSDO command. Backward compatibility is given by the fact, that each PDO number out of range is handled like 1, i.e. that PDO 1 is default. This guarantees, that older code using 0 as the last parameter by default behaves the same and defaults to PDO 1 now.



CANopen PDO size

A CANopen PDO is always 8 bytes long; it can therefore hold a maximum of 8 objects.

PDO 5 (= "serial PDO") Size

The mailbox size of the PDO 5 can be up to approx. 250 Bytes. The PDO 5 is also used by the oscilloscope tool of the APOSS development environment, therefore it is recommended to use this PDO not in applications, that shall be debugged using the oscilloscope tool later on.

Description

The command LINKSDO links the content of one or more elements of the system variables pseudo array SYSVAR to a TxPDO. This is called PDO mapping. Each change of a linked SYSVAR element is forwarded automatically into the defined bytes of the TxPDO.

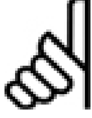
The system variables pseudo array SYSVAR holds internal data and variables, as well as each SDO (according to the SDO object dictionary), which also means the first 250 elements of each application array are included. The content of almost any variable, array or parameter can be forwarded to TxPDOs by the LINKSDO mapping configuration. The SYSVAR index of a SDO can be calculated with the following formula:

0x01000000 + ("SDO index" << 8) + "SDO subindex"

Example 1: SDO 0x2500 / 1 (= SDO holding the position value of axis 1)
=> SYSVAR index = 0x01250001

Example 2: SDO 0x2100 / 5 (= SDO with the first element of the first application array)
=> SYSVAR index = 0x01210001

The TxPDO is also copied into the PDO array. The same data content can be accessed by reading the PDO array.



NB!:

As standard, a changed PDO is automatically dispatched (asynchronous operating mode). If this is not desired, then, you can set the SDO-Index 0x1800 sub index 2 to another value (e.g. 254, instead of the standard 255). Thereby, dispatching no longer takes place automatically, but instead the PDO has to be collected per remote frame.

Automatic PDO activation

Just the PDO 1 (RxPDO = 0x200 + Node-ID / TxPDO = 0x180 + Node-ID) is enabled by default according to the CANopen specification, i.e. the "Valid" bit (0x1800 / Subindex 1) is set. If the mapping is configured for other PDOs using LINKSDO (or LINKPDO), then the "Valid" bits (0x1801-0x1804, Subindex 1) of these TxPDOs are also set automatically.

	CANopen versus APOSS mapping If the SYSVAR index refers to a SDO of the SDO object dictionary (i.e. SYSVAR index starts with 0x01...), then a pure CAN mapping is internally executed. When a corresponding object is changed by a SDO command, the PDO is also immediately rewritten. The mapping for the corresponding CAN object can be read-out by a supervisor control unit. If other SYSVAR indices are used, an APOSS mapping is carried out. This can be combined with the CAN Mapping, but does not conform to CANopen, because the correct map entries can not be read out by the CANopen mapping objects.
	NB!: The linking of internal system variables has to be accomplished very carefully und should only be done from experienced APOSS users. Thorough knowledge about the usage und meaning of the internal system variable is necessary, not to cause an incorrect system behavior.
Portability	Command is available starting with MCO 5.00; with the same version the#DEBUG command has been replaced by the Debug mode.
Command Group	PAR
Cross Index	LINKPDO, SYSVAR, Parameter Reference, Debugging Commands, PDO
Syntax Example 1	// Link current position error (= SDO 0x0x2500/6) to TxPDO 1 LINKSDO 0x01250006 32 1 " " 1

Syntax Example 2	// Link digital inputs 1 - 8 (= SDO 0x2202/10) to TxPDO 1
	LINKSDO 0x01220202 8 1 " " 1
	// Link digital inputs 9 - 16 (= SDO 0x2202/10) to next 8 bits of TxPDO 1
	LINKSDO 0x01220203 8 2 " " 1

Syntax Example 3	// Link DS402 status word (= SDO 0x6041/0) into TxPDO 1
	LINKSDO 0x01604100 16 1 " " 1
	// Link DS402 "Mode of operation display" (= SDO 0x6061/0) into TxPDO 1
	LINKSDO 0x01606100 8 2 " " 1

☐ LINKSYSVAR

Summary	Link system variable with LCP display
Syntax	LINKSYSVAR indx parno "text"
Parameter	indx = Index of the system variable SYSVAR parno = LCP-Parameter number 19-00 to 19-99 text = descriptive text for display
Description	The command LINKSYSVAR links the system variable SYSVAR[indx] with the FC 300 Parameter (19-00 to 19-99) and the display "text". This means that you can link internal values on the display without using LINKGPAR.
	NB!: The parameter is updated every 40 ms. Therefore, if five parameters are linked in this way, it takes at least 200 ms until the same parameter is updated.
Command Group	PAR
Cross Index	LINKGPAR, SYSVAR, Application parameter, Parameter Reference
Syntax Examples	LINKSYSVAR 33 1990 "internal line number" LINKSYSVAR 30 1991 "Motor voltage"

◻ LOOP

Summary	Defined loop repetition
Syntax	LOOP n label
Parameter	n = number of loop repetitions label = identification of target program position
Description	A single or multiple repetition of a certain program part can be realized by using the LOOP command. The number of loop repetitions can be given as either an absolute value or in the form of a variable. The program position to be jumped to is identified via a label. A label can be made up of one or more characters, and must not be identical with a variable name or a command word. A label must also be unique, i.e. the same label may not be used more than once for different program positions. **NB!:** The label on the target program position must be followed by a colon (:). Because the internal loop counter monitors only at the end of the loop and then decreases by one, the commands within the loop will be carried out with one more sequence than keyed in (keyed in loop repetitions 10 = 11 real repetitions).
Command Group	CON
Cross Index	GOTO, WHILE .. ENDWHILE, REPEAT .. UNTIL
Syntax Example	next_in: /* jump to label */ command line 1 command line n LOOP 9 next_in /* repeat loop contents 10 times */
Program Sample	LOOP_01.M, APOS_01.M, IN_01.M, MOTOR_01.M, NOWAI_01.M

◻ MAPOS

Summary	Queries current actual position of the master
Syntax	res = MAPOS
Return Value	res = master position to absolute actual zero point in qc
Description	With the MAPOS command it is possible to query the actual master position (absolute to the actual zero position).
Command Group	SYS
Cross Index	CPOS, DEFORIGIN, SETORIGIN, POSA, POSR, Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*
Syntax Example	PRINT MAPOS /* queries actual master position and print to PC */

☐ MAPOSDIFF

Summary	Overflow handling of incremental encoders in applications.
Syntax	res = MAPOSDIFF oldpos
Parameter	oldpos = MAPOS at a previous time
Return Value	Returns difference between MAPOS and oldpos (res = MAPOS – oldpos) in UU
Description	This command simplifies overflow handling of incremental encoders in applications. If, for example, the user stores an actual position in his program and wants to calculate the difference at a later time, then he normally has to account for overflow of the position. Instead this command can be used, see below. Internally those routines look if the difference is bigger than POS_LIMIT (0x3FFFFFFF). If so then it is assumed that an overflow happened and it is handled correctly.
	NB!: This will not solve the problem of overflowing if the application uses user units.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	MAPOS

Syntax Example	oldpos = MAPOS
	..
	diff = MAPOSDIFF oldpos
	// this function returns the difference between MAPOS and oldpos in user units
	// handling an overflow if necessary (diff = MAPOS – oldpos)

☐ MAVEL

Summary	Queries actual velocity of the master
Syntax	res = MAVEL
Return Value	res = actual velocity of the master in qc/s, the value is signed
Description	This function returns the actual velocity of the master drive in qc/s, with qc referring to the master encoder. The accuracy of the values depends on the duration of the measurement (averaging). The standard setting is 20 ms, but this can be changed by the user with the _GETVEL command. It is sufficient to call up the command once in order to work with another measuring period from then on. Thus, the command: var = _GETVEL 100 sets the duration of the measurement to 100 ms, so that you have a considerably better resolution of the speed with AVEL / MAVEL, however, in contrast, quick changes are reported with a delay of a maximum of 100 ms.
Command Group	SYS
Cross Index	AVEL
Syntax sample	PRINT MAVEL /* queries actual velocity of the master and print to PC */

☐ MENCPOSOFFS

Summary	Syncs the incremental position counter with the absolute counter in the encoder.
Syntax	result = MENCPOSOFFS offset
Parameter	offset = Returns the difference between absolute and incremental position (absolute –incremental)
Return values	OK 0 The command was successful TIMEOUT -1 No answer has been received within 300ms BADFRAME -2 The received frame is not valid OVERFLOW -4 Received more bytes than the receive buffer can take
Description	The difference between the absolute encoder position and the incremental counter is determined and returned. For this, the incremental counter in the DSP is latched exactly at the moment where also the Hiperface encoder latches the absolute position which it sends back over RS485. With this difference, the user e.g. can set the position within APOSS to the absolute value with SETMORIGIN. You can also use the ENCPOSOFFS command in case the Hiperface encoder is used as slave signal instead of master signal (see parameter 32-52)
Command Group	SYS
Cross Reference	ENCPOSOFFS
Program Sample	See program sample ENCPOSOFFS command.

☐ MENCTGREAD

Summary	Reads a RS485 telegram from the encoder.
Syntax	result = MENCTGREAD array
Parameter	array = The user array where the received payload data should be put.
Return values	OK x (>0) TG has arrived with x bytes user data ACTIVE 0 The transmission is still ongoing TIMEOUT -1 No answer has been received within 300ms BADFRAME -2 The received frame is not valid OVERFLOW -4 Received more than the receive buffer can take
Description	After a telegram has been sent with MENCTGWRITE, the answer can be polled by this command. The return value will show if it has already arrived or if a timeout has occurred. You can also use the ENCTGREAD command in case the Hiperface encoder is used as slave signal instead of master signal (see parameter 32-52)
Command Group	SYS
Cross Reference	ENCTGREAD, MENCTGWRITE
Program Sample	See program sample ENCTGREAD command.

◻ MENCTGWRITE

Summary	Sends a RS485 telegram to the encoder.
Syntax	result = MENCTGWRITE length array
Parameter	length = The number of bytes (in the user array) to be sent.
	array = The user array containing the payload data to send to the encoder.
Return values	OK 0 Telegram has been sent
	BUSY -3 There is still another transmission ongoing and not timed out yet
Description	This command will send a RS485 telegram to the encoder with the ID "MENCODERID". The user has to fill the payload data into an array before. The command will then put this data into a regular RS485 frame and add CRC value to it.
	The command does not wait till the data has been sent or an answer is received, it returns immediately.
	The answer of the telegram has to be polled with MENCTGREAD
	You can also use the ENCTGWRITE command in case the Hiperface encoder is used as slave signal instead of master signal (see parameter 32-52).
Command Group	SYS
Cross Reference	MENCTGREAD, ENCTGWRITE
Program Sample	See program sample ENCTGREAD command.

□ MIPOS

Summary	Query last index or marker position of the master
Syntax	res = MIPOS
Return Value	res = last index or marker position of the master absolute to actual zero point in qc
Description	The command MIPOS returns the last index or marker position of the master absolute to the current zero point. The configuration of MIPOS, that is whether master-encoders index- or marker position (= controlled drive) is returned, is done with the par. 33-19 *Master Marker Type*. **NB!:** The trigger signal for the marker position has to be connected mandatory to the input 5. The position value in MIPOS is accurate to +/- 1qc. In opposite to the position information in MAPOS, which is just updated in a controller cycle of typically 1 ms, the actual position value is hardware stored in real time a buffer (in an internal processor register), when the configured signal is high. Then it will be copied in the system variable MIPOS. If simultaneously to the marker position an interrupt is initiated (ON INT 5 GOSUB ...) and within this interrupt it is operated with MIPOS, you should use before reading of MIPOS a delay of 2 milliseconds (DELAY 2) within the interrupt subroutine. So it can be ensured, that the latched position value is already complete copied in the system variable MIPOS and that not be taken an old value. – See also sample. **NB!:** The command MIPOS can not be used: – with absolute encoders (see par. 32-30 *Incremental Signal Type)* – when Parameter 32-50 is set to [3] – Motor Control.
Command Group	SYS
Cross Index	CPOS, DEFORIGIN, SETORIGIN, POSA, POSR, ON INT Par.: 32-12 *UU Numerator*, 32-11 *UU Denominator*, 33-19 *Master Marker Type*
Syntax Example	PRINT MIPOS /* print to the PC the last index position of the master */
Sample	see below

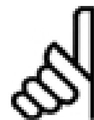
```
// Definition Interrupt-Handler
ON INT 5 GOSUB master_int
    // Definition of IPOS-Latching on positive edge at input 5
SET SYNCMTYPM 2
CVEL 10                 // Start moving
CSTART                  // Endless-Loop
mainloop:               // ...
GOTO mainloop
SUBMAINPROG
SUBPROG master_int
    int_mpos = MAPOS
    // Latching MAPOS for testing, how exact it would be ...
    DELAY 2             // Wait 2 ms, to be sure, that MIPOS is correct updated
    triggered_mpos = MIPOS    // Latching IPOS for a later handling etc.
                        // ...
                        // ...
    PRINT "Interrupt master position: ",int_mpos
    PRINT "Triggered master position: ",triggered_mpos
    RETURN
ENDPROG
```

☐ MIPOSDIFF

Summary	Overflow handling of incremental encoders in applications.
Syntax	res = MIPOSDIFF oldpos
Parameter	oldpos = MIPOS at a previous time
Return Value	Returns difference between MIPOS and oldpos (res = MIPOS – oldpos) in UU
Description	This command simplifies overflow handling of incremental encoders in applications. If, for example, the user stores an actual position in his program and wants to calculate the difference at a later time, then he normally has to account for overflow of the position. Instead this command can be used; see below. Internally those routines look if the difference is bigger than POS_LIMIT (0x3FFFFFFF). If so then it is assumed that an overflow happened and it is handled correctly.
	NB!: This will not solve the problem of overflowing if the application uses user units.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	MIPOS

Syntax Example	oldpos = MIPOS
	..
	diff = MIPOSDIFF oldpos
	// this function returns the difference between MIPOS and oldpos in user units
	// handling an overflow if necessary (diff = MIPOS – oldpos)

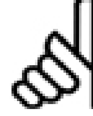
☐ MOTOR OFF

Summary	Turns off motor control
Syntax	MOTOR OFF
Description	The motor control can be disabled by using the MOTOR OFF command. After MOTOR OFF, the drive axis can be moved freely, as long as there is no motor brake. A monitoring of the actual position will continue to take place, i.e. the actual position (APOS) can still be queried after MOTOR OFF.
	**NB!:** For a restart of a motion process after MOTOR OFF the command MOTOR ON must be used. Only the command ERRCLR automatically activates MOTOR ON.
Command Group	INI
Cross Index	MOTOR ON
Syntax Example	MOTOR OFF /* switch off controller of the axis */
Program Sample	MOTOR_01.M, POS_01.M

□ MOTOR ON

Summary	Turns on motor control
Syntax	MOTOR ON
Description	The motor control can be enabled again, following a previous MOTOR OFF, by use of the MOTOR ON command. When carrying out the MOTOR ON, the commanded position is set to the actual position, i.e. the motor remains at the actual position. Thus the positioning error is reset at the execution of MOTOR ON. **NB!:** The MOTOR ON command is not suitable for re-activation of position control following an error. For this purpose, the ERRCLR command is to be used.
Command Group	INI
Cross Index	MOTOR OFF
Syntax Example	MOTOR ON /* switch on controller of the axis */
Program Sample	MOTOR_01.M, POS_01.M

□ MOTOR STOP

Summary	Stops the drive
Syntax	MOTOR STOP
Description	By using the MOTOR STOP command, a drive in positioning, speed or synchronizing mode can be decelerated with programmed acceleration and arrested at the momentary position. A drive arrested with this command can, at a later point, via the CONTINUE command, resume its original motion. (Exception: CONTINUE does not continue an interrupted synchronization command.)  **NB!:** If MOTOR STOP is executed in a subprogram or if NOWAIT is set to ON, then the next lines in the program are already processed while MOTOR STOP is being processed; the braking process runs in the background. Therefore, in order to slow the drive down to a speed of zero it is necessary to ascertain that no new positioning command is given during braking.
Command Group	CON
Cross Index	POSA, POSR, CSTART, CONTINUE, CSTOP, NOWAIT
Syntax Example	MOTOR STOP /* interrupt motion of the axis */
Program Sample	MSTOP_01.M

◻ MOVESYNCORIGIN

Summary	Relative shifting of the origin of synchronization
Syntax	MOVESYNCORIGIN mvalue
Parameter	mvalue = Relative offset in relation to the Master in qc Value range: (–MLONG / par. 33-11 SYNCFACTS) – (MLONG / par. 33-11 SYNCFACTS)
Description	The command shifts the origin of synchronization in relation to the master. While SET SYNCPOSOFFS sets the offset position absolutely, MOVESYNCORIGIN always relates to the last one and shifts the offset position relatively. If you have to shift the offset position continually, you can prevent too large numbers or an overflow in this way.
	NB!: Valid for position synchronization SYNCP and position synchronization with marker correction SYNCM.
Command Group	SYN
Cross Index	SET, Parameters: 33-11 *Synchronization Factor Slave,* SYNCFACTS, 33-12 *Position Offset for Synchronization,* SYNCPOSOFFS
Syntax Example	MOVESYNCORIGIN 1000

◻ MSGVAL

Summary	Contains the second part of the last read CAN message.
Syntax	longval = MSGVAL
Return Value	longval = bytes 4 to 7 of the last CAN message to be read
Description	MSGVAL is a variable which returns the long value of the CAN message. The CAN message must have been previously read with INMSG or INGLB.
	This value is only valid as long as no new INMSG or INGLB command has been executed.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	INMSG, INGLB, ON CANMSG

Syntax Example	a = INMSG –1
	IF (a > –1) THEN b = MSGVAL
	ENDIF

◻ NOWAIT

Summary Wait / Do not wait after a POSA/POSR command

Syntax NOWAIT s

Parameter s = condition:

ON = continue program execution while going to target position
OFF = hold program execution until target position is reached

Description The NOWAIT command defines the program flow for positioning commands.

There are two conditions NOWAIT ON and NOWAIT OFF:

NOWAIT ON This will allow the system to simultaneously position and to process the following instructions as well.

175HA528.10

```
NOWAIT ON
POSA 2000
WAIT 1000
OUT 11
POSA 1000
```

Position / WAIT 1000 / OUT 11 / 2000 / 1000 / 1000 2000 3000 Time / 0 1000 1550 2000

When starting a positioning command with NOWAIT ON, further command procedures are continued and the positioning work takes place in the background (so to say). In the NOWAIT ON condition it is thus possible to query the momentary position, or to alter the velocity or the target position during the positioning procedure.

NOWAIT OFF Allows the execution of the program line by line, i.e., the positioning takes place and the processor waits till it is over and then does the following instructions.

175HA527.10

```
NOWAIT OFF
POSA 2000
WAIT 1000
OUT 11
POSA 1000
```

Position / WAIT 1000 / OUT 11 / 2000 / 1000 / 1000 2000 3000 Time / 0 1000 2000

Note: The dotted arrows mean the movement positions.

NB!:
The default condition is NOWAIT OFF, i.e. if no NOWAIT instruction is contained within a program, then the positioning procedure is carried out in its entirety before the processing of the next command is begun.

If, when in NOWAIT condition during an active positioning procedure, a further positioning command follows, then the new target position will be tracked without interruption.

The HOME as well as the INDEX commands will be processed to the end in the NOWAIT ON condition, before the next command can be begun.

Command Group CON

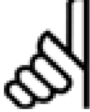
Cross Index WAITAX, AXEND, POSA, POSR, HOME, INDEX

Syntax Example

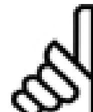
```
NOWAIT ON     /* no waiting after POS-commands */
NOWAIT OFF    /* wait after POS-commands till target reached */
```

Program Samples NOWAI_01.M,MSTOP_01.M, OUT_01.M, VEL_01.M

◻ ON CANINPUT

Summary	Call up a subprogram when a CAN telegram type 'id' arrives.
Syntax	ON CANINPUT id GOSUB name
Parameter	id = 0
	name = name of the subroutine
Return Value	–
Description	The instruction ON CANINPUT calls up a subroutine, when a global CAN telegram with the identifier '0' arrives. This is the same input telegram, which can be read out with INGLB and MSGVAL. This global telegram is also used for the program break. Therefore be aware that byte 0 and 1 may not be used and must set to 0. In contrast the bytes 2 up to 7 can be freely used. Additional it can be react on a received zero telegram with ON CANINPUT.
	**NB!:** The instruction should be located at the beginning of the program so that it is valid for the entire program. The subroutine to be called up must be defined within the program area marked with SUBMAINPROG and ENDPROG.
Portability	Command is available starting with MCO 5.00.
Command Group	INT
Cross Index	ON ERROR, ON INT, ON PERIOD, ENABLE ..., DISABLE ...
Syntax-Example	ON CANINPUT 0 GOSUB break /* interrupt procedure is defined */

◻ ON CANMSG GOSUB

Summary	Calls up subroutine.
Syntax	ON CANMSG GOSUB name
Parameter	name = name of the subroutine
Description	Calls up a subroutine when there is a message in the buffer. Subroutine *name* is called up when at least one message is in the CAN receive buffer.
	**NB!:** – The instruction ON CANMSG GOSUB should be located at the beginning of the program so that it is valid for the entire program. – The subroutine to be called up must be defined within the program area marked with SUBMAINPROG and ENDPROG.
Portability	Command is available starting with MCO 5.00.
Command Group	INT
Cross Index	ON ERROR, ON INT, ON PERIOD, ENABLE ..., DISABLE ...
Syntax Example	ON CANMSG GOSUB CAN PROC /* interrupt procedure is defined */

☐ ON COMBIT .. GOSUB

Summary	Call up a subprogram when Bit n of the communication buffer is set.
Syntax	ON COMBIT n GOSUB name
Parameter	n = Bit n of communication buffer –32 <= n<=32, n!= 0
	name = name of subprogram
	NB!: ON COMBIT refers to the first 32 Bits of the process data memory.
Description	The instruction ON COMBIT is used to call up a subprogram when Bit n of the communication buffer is set.
	NB!: – The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. – During the execution of an ON COMBIT subroutine NOWAIT is set to ON.
Priority	If a number of interrupts occur simultaneously, the subprogram assigned to the lowest bit is worked through first. The other interrupts will be processed afterwards. If, during an interrupt subroutine, the same interrupt occurs (exception: error interrupt), then it will be ignored and thus lost.
Portability	In the case of COMOPTGET and COMOPTSEND, the offset of 2 Word is retained for compatibility reasons.
Command Group	INT
Cross Index	SUBPROG .. RETURN, COMOPTGET, COMOPTSEND, Priorities of Interrupts, NOWAIT
Syntax Example	ON COMBIT 5 GOSUB test // set interrupt on field bus bit 5

☐ ON DELETE .. GOSUB

Summary	Deletes a position interrupt.
Syntax	ON DELETE pos GOSUB name
Parameter	pos = value name = name of subprogram
Description	The command can be used to delete position interrupt, e.g. ON APOS, which is defined as follows: ON sign APOS xxx GOSUB name The parameter 'pos' of this command can hold any value, e.g. 0. It is not checked and has no relevance for the deletion of the interrupt. The main importance belongs to the parameter 'name', which has to hold the name of the subprogram that was formerly defined in the ON APOS command. So, the 'ON DELETE pos GOSUB name' command deletes any (!) position interrupt, which belongs to the subprogram identified by the given name. Please see sample 1.
	NB!: Only position interrupts are deleted, but no other type of interrupt.
	Re-routing of an ON … APOS … GOSUB It is possible to 're-route' a position interrupt to another subprogram. This does not define a new interrupt, but just modifies the subprogram, which has to be executed in case of interrupt detection. The command syntax is the same like for the ON APOS command: ON sign APOS xxx GOSUB newname The parameters 'sign' and 'xxx' have to be exactly the same like within the original definition. The position which is concerned is identified by these two parameters. The parameter 'newname' has to hold the updated name of the subprogram, which has to be called up in case of the interrupt, takes place. Please see sample 2.
	NB!: Only position interrupts can be re-routed, but no other type of interrupt .
Command Group	INT
Cross Index	ON posint GOSUB, ON INT ..
Syntax Example 1	ON – APOS 20000 GOSUB hitinfo // Interrupt #1 ON – APOS 10000 GOSUB hitinfo // Interrupt #2 ON + APOS 10000 GOSUB hitinfo // Interrupt #3 ON + APOS 0 GOSUB hitzero // Interrupt #4 ON – APOS 0 GOSUB hitzero // Interrupt #5 ON INT 3 GOSUB hitinfo // Interrupt #6 … ON DELETE 0 GOSUB hitinfo … ON + APOS 99999 GOSUB hitinfo // New defined position interrupt

	Result: All the position interrupts (#1, #2, #3) belonging to the subprog hitinfo are deleted as soon as 'ON DELETE 0 GOSUB hitinfo' is executed. These interrupts don't count anymore for the maximum number of available interrupts and can not be enabled or disabled anymore. All other non-position interrupts, even the ones belonging to the same subprogram (e.g. ON INT 3) are still valid! As soon as the command line 'ON + APOS 99999 GOSUB hitinfo' is executed, this defines a new position interrupt, which is "linked" to the given subprogram (that has been already in use before).
Syntax Example 2	ON – APOS 10000 GOSUB hitinfo // Interrupt #1 ON + APOS 10000 GOSUB hitinfo // Interrupt #2 ... ON + APOS 10000 GOSUB hitposdir // Re-routed interrupt #2 Result: As soon as the second definition of the 'ON + APOS 10000 ...' is executed, the interrupt #2 is "re-routed" to the newly defined subprogram 'hitposdir'. It is still the same interrupt (i.e. not an additional one), which calls up another subprogram now. The "old" definition of interrupt #1 'ON – APOS 10000 GOSUB hitinfo' is still valid without any modification.

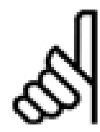
☐ ON DELETE .. SETOUT

Summary	Deletes all interrupts which set or reset an output.
Syntax	ON DELETE sign inttype SETOUT outno
Parameter	sign + = rising edge – = falling edge inttype = APOS IPOS MAPOS MCPOS MIPOS outno = output number
Description	This command deletes all interrupts which set or reset the output *outno*. If the *outno* is positive in the above command, then only interrupts are deleted where the outno is set. If *outno* is negative, then only interrupts are deleted where the output is reset. So if you use both types of interrupt definitions, you must also have two delete commands for that.
Portability	Command is available starting with MCO 5.00.
Command Group	INT
Cross Index	ON INT .. GOSUB, ON posint GOSUB
Syntax Example	SET SYNCMPULSS 20000 // distance between two markers ON +ipos 500 SETOUT 1 ON +ipos 1000 SETOUT -1 ... (program) ... ON DELETE 0 SETOUT 1 ON DELETE 0 SETOUT -1

☐ ON ERROR GOSUB

Summary	Definition of an error subroutine
Syntax	ON ERROR GOSUB name
Parameter	name = name of the subroutine
Description	By using the ERROR GOSUB instruction, a subroutine will be defined, which can be called up in case of error. If an error occurs after the definition, then an automatic program abort will not take place – instead, the defined subroutine will be called up. Within this subroutine, it is possible to target the re-action to the error, to wait for user intervention via ERRCLR (clear error) or, in the case of non-correctable errors, to abort the program via the EXIT instruction. If the program is not aborted, then the processing will continue from the point where the interruption occurred. By using the CONTINUE command, it is possible to continue the error-interrupted motion. (Exception: synchronization commands)
	NB!: The ON ERROR GOSUB instruction should be at the start of a program, so that it has validity for the entire program. The subroutine to be called up must be defined within the identified SUBMAINPROG and ENDPROG program. The identification of an error condition and the call up of the corresponding sub-routine requires a maximum of 2 milliseconds.
	NB!: – Error subroutines cannot be interrupted through any other interrupts. – During the execution of an error routine NOWAIT is automatically set to ON. If the error subroutine is exited with the error still active because e.g. ERRCLR was not carried out or another error has occurred, then a new call takes place.
	NB!: The ON ERROR GOSUB xx routine does not terminate the HOME and INDEX command. This means they will be executed after the error has been cleared. To prevent this an ON TIME 1 can be included in the error routine.
Command Group	INT
Cross Index	SUBPROG...RETURN, ERRCLR, ERRNO, CONTINUE, EXIT, Priorities of Interrupts, ON TIME, NOWAIT
Syntax Example	ON ERROR GOSUB errhandle /* definition of an error subroutine */ command lines 1 ... n SUBMAINPROG /* subroutine errhandle must be defined */ SUBPROG errhandle command lines 1 ... n RETURN ENDPROG
Program Sample	ERROR_01.M, IF_01.M, INDEX_01.M

☐ ON INT .. GOSUB

Summary	Defining an interrupt input
Syntax	ON INT n GOSUB name
Parameter	n = number of the input to be monitored; reaction to the rising edge (input area 1 ... 8 and FC 300 inputs 18 ... 33) -n = number of the input to be monitored, reaction to the falling edge (input area –8 ... –1 and FC 300 inputs –33 ... –18) = or with CAN applications > 256: Module number * 256 + I/O number name = subroutine name
Description	By using the ON INT GOSUB instruction, a subroutine must be defined which will be called up when an edge is detected at the monitored input. A maximum of one subroutine per input can be defined. The ON INT command allows the assignment of a positive interrupt <u>and</u> a negative interrupt for an input <u>at the same time</u>: ON INT 1 GOSUB posedge ON INT -1 GOSUB negedge This definition can take place at any time. If, following this definition, a corresponding interrupt occurs, then the accompanying subroutine is called up and processed. After the last subroutine command (RETURN), the program will continue from the point of interrupt. If interrupt functions are set to CAN modules, they have to be initialized with CANINI.
	NB!: The ON INT GOSUB instruction should be at the start of the program, so that it has validity for the entire program. The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. The identification of an interrupt and the call up of the corresponding subroutine requires a maximum of 2 milliseconds. Interrupt from FC 300 input add additional 2 ms, in worst case. A minimal signal length of 1 ms is necessary for the sure identification of a level change! The chapter input/output terminal contains more information concerning the input circuit and input technical data.
	NB!: – The instruction for ON INT GOSUB is edge and not level triggered. – During the execution of a subroutine called by an interrupt NOWAIT is automatically set to ON.
Priority	If a number of interrupts occur simultaneously, the subprogram assigned to the lowest bit is worked through first. The other interrupts will be processed afterwards. If, during an interrupt subroutine, the same interrupt occurs (exception: error interrupt), then it will be ignored and thus lost.
Portability	Starting with MCO 5.00 a positive <u>and</u> a negative interrupt for an input at the same time can be assigned.
	The CAN commands operate with the pre-defined PDOs of CANopen. Do not change these default settings (minimum capability device), otherwise the CAN commands will not operate anymore.
Command Group	INT

Cross Index
SUBPROG..RETURN, ON ERROR .. GOSUB, WAITI, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts, NOWAIT, CANINI

Syntax Example
```
ON INT 4 GOSUB posin      /* Definition of Input 4 (positive edge) */
ON INT –5 GOSUB negin     /* Definition of input 5 (negative edge) */
command line 1
command line n
SUBMAINPROG               /* subroutine must be defined */
    SUBPROG posin
        command line 1
        command line n
    RETURN
    SUBPROG negin
        command lines 1 ... n
    RETURN
ENDPROG
```

Program Sample
ONINT_01.M, DELAY_01.M

☐ ON KEYPRESSED GOSUB

Summary	Interrupt when a key pressed or released.
Syntax	ON KEYPRESSED GOSUB name
Parameter	name = name of subroutine
Description	The instruction ON KEYPRESSED can be used to respond, when a key of the LCP panel is pressed and or released. The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.
	NB!: During the execution of subroutine called by an interrupt NOWAIT is automatically set to ON.
Command Group	INT
Cross Index	SUBPROG .. RETURN, INKEY

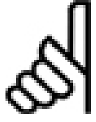
Syntax Example
```
ON KEYPRESSED GOSUB keyhandler
WHILE(1) DO          // endless loop
ENDWHILE
//////////////////////////////////
SUBMAINPROG
SUBPROG keyhandler
    key = INKEY(-1)      // don't wait for key

    PRINT key
RETURN
ENDPROG
```

◻ ON PARAM .. GOSUB

Summary	Call up a subprogram when a parameter is altered.
Syntax	ON PARAM n GOSUB name
Parameter	n = parameter number name = subroutine name

Description	The instruction ON PARAM can be used to respond when parameters are altered via the LCP display and to call up a subprogram.
	All parameters (32-xx, 33-xx) and all application parameters (19-xx) as well as parameters of general interest (e.g. 8-02, 5-00) can be used.
	The ON PARAM for array elements (e.g. ON PARAM 0x01210005) also works if the array is mapped into a receive PDO with a LINKPDO command, but it is limited. Only one array element can be used in an ON PARAM per linked array.
	Example:
	DIM test[100]
	LINKPDO 1 320 0x01210005 0
	// the first ten longs of the PDO are linked into
	// the first ten elements of array test
	ON PARAM 0x01210007 GOSUB test
	// if the third element of array test changes
	If another ON PARAM is added like
	ON PARAM 0x01210009 GOSUB testsub
	// if the fifth element of array test changes
	then this will overwrite the first ON PARAM, because only one per LINK-Array. can be handled
	The ON PARAM will become active if the array element is written by an incoming PDO or by an SDO write to the SDO 0x012100ss with the correct subindex.
	It will not become active if the array is written in the program by test[nn] = value. And it will not become active if the whole array gets overwritten by an array write using the 0x23FF SDO.

	**NB!:** – A maximum of 10 ON PARAM functions are possible. – The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. – During the execution of a subroutine called by an interrupt NOWAIT is automatically set to ON.
Priority	If, during an interrupt subroutine, the same interrupt occurs (exception: error interrupt), then it will be ignored and thus lost.
Command Group	INT
Cross Index	SUBPROG..RETURN, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts, NOWAIT
Syntax Example 1	ON PARAM 3267 GOSUB poserr // when position error is changed SUBMAINPROG SUBPROG poserr PRINT "New position error: ", GET POSERR RETURN
Syntax Example 2	// Trigger an interrupt each time the CANopen DS402 "Mode of operation" changes ON PARAM 0x01606000 GOSUB OpModeUpdate

```
SUBMAINPROG
SUBPROG OpModeUpdate
    PRINT "New DS402 Operation Mode: ",sysvar[0x01606000]
    // Execute an action depending on the new operating mode
    SWITCH (sysvar[0x01604000])
        CASE 1:
            PRINT "Profile position mode"
            // Action ....
            BREAK
        CASE 2:
            PRINT "Velocity mode"
            // Action ....
            BREAK
        // CASE ....
            // ...
            // BREAK
        DEFAULT:
            PRINT "Operation Mode is not supported"
    ENDSWITCH
RETURN
ENDPROG
```

Syntax Example 3

```
// Link the RxPdo with the user parameter 1 ...
LINKPDO 1 32 0x01220101 0
// and configure an interrupt each time it changes
ON PARAM 0x01220101 GOSUB UserParamUpdate
SUBMAINPROG
SUBPROG UserParamUpdate
    PRINT "User Parameter 1 updated: ",sysvar[0x01220101]
RETURN
ENDPROG
```

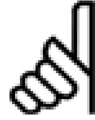
◻ ON PERIOD

Summary	Calls up a subroutine at regular intervals.
Syntax	ON PERIOD n GOSUB name
Parameter	n > 20 ms = time in ms, after which the subroutine is called up again
	n = 0 = switch off the function
	name = subprogram name
Description	With ON PERIOD it is possible to call up a subprogram at regular intervals (time triggered). ON PERIOD works like an interrupt. Is checked every 20 ms.
	NB!:
	– The precision with which the time is depends on the remaining program. Typically the precision is ±1 ms.
	– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program
	– During the execution of an ON PERIOD subroutine NOWAIT is set to ON.
Command Group	INT
Cross Index	ON TIME, GOSUB, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts, NOWAIT

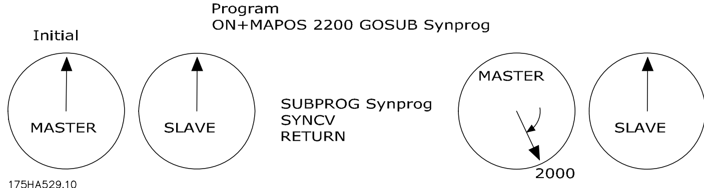
☐ ON posint .. GOSUB

Summary	Call up a subprogram when a position interrupt occurs.
Syntax	ON sign postype position GOSUB name
Parameter	sign + = rising edge or when the position is passed in positive direction – = falling edge or when the position is passed in negative direction 100 xxx 0 <–––––– positive direction ––––––> negative direction postype = APOS IPOS MAPOS MCPOS MIPOS position = depending on the command in user units [UU], or master user units [MU], or curve units [CU] name = subprogram name

Description	If an ON xPOS command is used and a position is given which lies behind an overflow of the encoder, then internally this is handled automatically.
	If e.g. no POSFACTs are set, then the following is now handled correctly.
	testpos = 0x3FFFFFF0 // position short before overflow
	newpos = testpos + 200 // new test position
	ON +APOS newpos GOSUB myprog
	The newpos is internally handled by a correct entry into the interrupt list.
	The same is true if the POSFACT_Z and POSFACT_N is set and the user value will cause an overflow of the internal qc positions.
	All or single position interrupts can be deleted with the command ON DELETE .. GOSUB.

	**NB!:** – The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. – During the execution of subprograms triggered by an interrupt, NOWAIT ON is set automatically.
ON APOS .. GOSUB	Call up a subprogram when the slave position xxx (UU) has been passed in positive or negative direction. The instruction can be useful for positioning and synchronization controls, as well as for CAM controls and CAM boxes. For example, in order to replace the increasing slave position in the case of open curves after each cycle by a recurring reference point. Sample: CSTART ON +apos 2000 GOSUB stop SUBMAINPROG SUBPROG stop CSTOP RETURN ENDPROG Initial 0 After Positioning 0 As per the program above the drive stops once it reached the position 2000.

ON IPOS .. GOSUB	This position interrupt looks at the distance between the last marker position and the actual position. It is important that the SYNCMPULSS is set correctly. This information is used for detection of overflow as well as for backward driving. In the case of backward driving, the information used is (SYNCMPULSS + (APOS – IPOS)) instead of (APOS – IPOS).
ON MAPOS .. GOSUB	Call up a subprogram when the master position xxx (MU) has been passed in positive or negative direction. For example, in order to set an output at any point in the case of a linear drive (slave) with a traversing range from 0 to 10000 UU.



Here according to the program Velocity Synchronization starts after Master has reached 2200 qc in positive direction. Then the slave and master move in SYNCV.

ON MCPOS .. GOSUB	Call up a subprogram when the master position xxx (MU) is passed.
	It is possible to call up a subprogram with the instruction ON MCPOS which is typical for cam controls if a specific master position (MU) has been passed in positive or negative direction. This allows not only the realization of CAM boxes, but also the execution of tasks that are much more complex. For example, one could change parameters online depending on the position.

NB!:
A DEFMCPOS or a SETCURVE must always be placed in front of the command ON MCPOS .. GOSUB, since otherwise the curve position is not known.

ON MIPOS .. GOSUB	Call up a subprogram when the distance between two markers is reached.
	This position interrupt looks at the distance between the last marker position and the actual position. It is important that the SYNCMPULSM is set correctly. This information is used for detection of overflow as well as for backward driving. In the case of backward driving, the information used is (SYNCMPULSM + (MAPOS – MIPOS)) instead of (MAPOS – MIPOS).
Portability	ON POSINT handle overflow is available starting with MCO 5.00
Command Group	INT
Cross Index	SUBPROG .. RETURN, DISABLE .., ENABLE .., Priorities of Interrupts, ON DELETE .. GOSUB, APOS, MAPOS, MIPOS, IPOS

Syntax Example 1

```
ON –apos 800 GOSUB name
    // Call up the subroutine name when slave position 800
    // is passed in negative direction
```

Syntax Example 2

```
SET SYNCMPULSS 20000   // distance between two markers
ON +ipos 5000  GOSUB prog1
ON +ipos 15000 GOSUB prog2
```

In this example, two markers have a distance of 20000qc. Let us assume that the first marker is at position 0. Then prog1 will be called at 5000, 25000, 45000, and so on, and prog2 will be executed at 15000, 35000, and so on.

Syntax Example 3

```
ON +mapos 1200 GOSUB name
    // Always call up subprogram at position 1200
```

Syntax Example 4

```
SET SYNCMPULSM 20000   // distance between two markers

ON +mipos 5000  GOSUB prog1
ON +mipos 15000 GOSUB prog2
```

	In this example two markers have a distance of 20000qc. Let us assume that the first marker is at position 0. Then prog1 will be called at 5000, 25000, 45000, and so on, and prog2 will be executed at 15000, 35000, and so on.
Syntax Example 5	Cardboard boxes are transported irregularly on a conveyor belt. By setting an output, the slave will always be started when the position xxx is reached. SUBMAINPROG // set subprogram output SUBPROG output OUT 3 1 // 03 on RETURN ENDPROG ON +MCPOS 4500 GOSUB output // call subprogram output always on position 4500

◻ ON posint .. SETOUT (TOIN)

Summary	Simulate a cam box (all types of POSINTs)
Syntax	ON +/- type position SETOUT outno ON +/- type position SETOUT outno TOIN inno
Parameter	type = any POSINT APOS IPOS MAPOS MIPOS position = depending on the command in user units [UU], or master user units [MU], or curve units [CU] outno = could be any valid output number (or the negative output number) inno = could be any valid input number (or the negative input number)
Description	All position interrupt functions can use this feature which simulates a cam box. This is possible with all types of POSINTs. In the first case, the output *outno* is either set or reset, depending on whether the *outno* is positive or negative. In the second form, the output is set to the value of the input *inno*. (Or to the opposite of the input value, if either the *inno* or the *outno* is negative.). If both *outno* and *inno* are negative, that is the same as if they both were positive. The advantage of those commands is that they are handled in the background and do not interrupt the application program. They are also faster than calling a subroutine which in turn then sets an output. Typical reaction time is below 1 ms.
Portability	Command is available starting with MCO 5.00
Command Group	INT
Cross Index	SUBPROG .. RETURN, APOS, IPOS, MAPOS, MCPOS, MIPOS
Syntax Example	SET SYNCMPULSS 20000 // distance between two markers on +ipos 500 setout 1 toin 2 on +ipos 1000 setout -1 In this example, the output 1 is set to the value of input 2 at a position of 500 user units after the marker. Then the output 1 is set to 0 again at a position of 1000 qc after the marker.

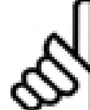
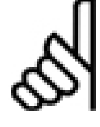
◻ ON STATBIT .. GOSUB

Summary	Call up a subprogram when bit n of the FC 300 status is set.
Syntax	ON STATBIT n GOSUB name
Parameter	n = Bit n of the status word Byte 1 + 2 Status word of the FC300 (see FC3xx manual) Byte 3 Bit 17 1 = MOVING Bit 18 1 = Overflow Slave Encoder Bit 19 1 = Overflow Master Encoder Bit 20 1 = POSFLOAT active *) Byte 4 SYNCSTAT Bit 25 1 = SYNCREADY Bit 26 1 = SYNCFAULT Bit 27 1 = SYNCACCURACY Bit 28 1 = SYNCMMHIT Bit 29 1 = SYNCSMHIT Bit 30 1 = SYNCMMERR Bit 31 1 = SYNCSMERR name = subroutine name *) Explanation: i.e. the axis is within the tolerance range of the control window par. 32-71 REGWMAX / par. 32-72 REGWMIN. As soon as the control window is set, the axis controller is switched on again.
Description	The instruction ON STATBIT is used to call up a subprogram when bit n of FC 300 status is set. These 32 bits of the FC 300 status consist of the FC 300 status word, the byte 3 of the internal status (e.g. MOVING) and the bit n of SYNCSTAT.
	NB!: – The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. – During the execution of a subroutine called by an interrupt NOWAIT is automatically set to ON.
Priority	If a number of interrupts occur simultaneously, the subprogram assigned to the lowest bit is worked through first. The other interrupts will be processed afterwards. If, during an interrupt subroutine, the same interrupt occurs (exception: error interrupt), then it will be ignored and thus lost.
Command Group	INT
Cross Index	SUBPROG ..RETURN, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts
Syntax Example	ON STATBIT 30 GOSUB markererror /* Interrupt, if error flag Master */ SUBMAINPROG SUBPROG markererror SYNCSTATCLR 32 /* clear error flag SYNCMMERR */ /* use value 32 of Parameter SYNCSTATCLR, not the bit-number! */ RETURN ENDPROG

☐ ON TIME

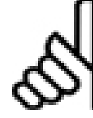
Summary	One-time access of a subroutine.
Syntax	ON TIME n GOSUB name
Parameter	n = time in ms, after which the subroutine is called up (maximum MLONG)
	name = name of the subroutine
Description	After expiration of the time set the corresponding subroutine is called up. In the meantime the program flow continues normally.
	NB!: – The precision with which the time is kept depends on the hardware used and the remaining program. Typically the precision is ±1 ms. – In General: The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program. – During the execution of an ON TIME subroutine NOWAIT is set to ON.
Command Group	INT
Cross Index	ON PERIOD, GOSUB, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts
Syntax Example	OUT 1 1 /* light on */ ON TIME 200 GOSUB off1 /* light off again after 200 ms */ SUBMAINPROG SUBPROG off1 OUT 1 0 RETURN ENDPROG

◻ OUT

Summary	Set or re-set digital outputs.
Syntax	OUT n s or OUT X/n s
Parameter	n = output number MCO 305: 1 – 8 (6) FC 312 outputs: 27 Relay outputs: 21 (relay 1) and 22 (relay 2), Note: FC 312 < 11 kW only has relay 1. MCB 105, relay outputs: X34/1 (relay 7), X34/5 (relay 8) and X34/10 (relay 9). or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) X/n = terminal block / pin number s = condition 0 = OFF 1 = ON
Description	The 8 (6) digital outputs of the MCO 305 option, the digital and relay outputs of FC 300, and the relay outputs of MCB 105 can be set and re-set by using the OUT command. The selection of the mode for output 7,8 is done by par. 33-60 IOMODE. If the outputs are used NPN or PNP depends on the selection for the standard FC 300 outputs set in par. 5-00. CAN modules which fulfill the CAN OPEN specifications can also be addressed with the IN command via the corresponding number, which is defined as follows: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules.
	**NB!:** If an illegal combination or a pin number is used for X/n, which can not be set, then error 171 will be reported. But there is no check if an input is used instead of an output or vice versa.
	NB!: – After switching on the system, all outputs are OFF. These outputs, which have pre-defined functions according to the I/O parameter settings, will also be influenced by the OUT commands! – The actual output status remains as it is, even after program end or program abort. – The output circuit and maximum load current can be taken from MCO Operating Instructions and the FC 300 Operating Instructions.
	**NB!:** The CAN commands operate with the pre-defined PDOs of CANopen. Don't change these default settings (minimum capability device), otherwise the CAN commands will not operate anymore.
Portability	Set or reset CAN modules is available starting with MCO 5.00.
Command Group	I/O

Cross Index	OUTB, IN, INB, Parameters: 33-60 *Terminal X59/1 and X59/2 Mode*, IOMODE, 33-63...70 *Terminal X59/n Digital Output*, O_FUNCTION_n
Syntax Example	OUT 3 1 // set output 3 on MCO 305 to 1 OUT 27 1 // set output 27 on FC 300 main board to 1 OUT X59/3 1 // set output 3 on MCO 305 to 1 OUT X34/1 1 // set first relay on relay option (relay 7) to 1
Program Sample	OUT_01.M

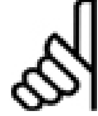
◻ OUTAN

Summary	Sets speed reference.
Syntax	OUTAN v
Parameter	v = bus reference range: –0X4000 – 0X4000 = –100 % – 100 %
Description	The OUTAN command can set FC 300 bus reference (speed or torque reference depending on setting of FC 300 par. 1-00). With OUTAN it is also possible to turn off the controller in OPEN LOOP using MOTOR OFF and to operate the FC 300 without feedback as a pure frequency converter. In this manner you can use APOSS to directly output set values, to read inputs, etc.
	**NB!:** The command MOTOR OFF must be executed previously. Thus, monitoring of the position error is no longer active.
Command Group	I/O
Cross Index	MOTOR OFF, MCO 305 Operating Instructions, FC 300 Design Guide
Syntax Example	MOTOR OFF /* turn off controller */ OUTAN 0X2000 /* set speed reference 50% */

☐ OUTB

Summary	Alteration of the condition of a digital output byte
Syntax	OUTB n v
Parameter	n = output byte 0 = 1 – 8 1 = 27,29 or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) v = value (0 ... 255)
	NB!: Numbering of the bytes begins with 0; this is in contrast to the numbering of the individual inputs, which starts with 1.
Description	With the OUTB command the condition of the digital outputs can be changed byte-by-byte. The byte value transferred determines the condition of the individual outputs. The bit with the lowest value in the byte corresponds to the set condition of output 1. CAN modules which fulfill the CAN OPEN specifications can also be addressed with the IN command via the corresponding number, which is defined as follows: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules.
	NB!: After switching on the system, all outputs are OFF. Outputs which have pre-defined functions according to the I/O parameter settings will also be influenced by the OUTB command! The actual output status remains as it is even after program end or program aborted. Output circuit and maximum load current see MCO Operating Instructions.
	NB!: The CAN commands operate with the pre-defined PDOs of CAN-OPEN. Don't change these default settings (minimum capability device), otherwise the CAN commands will not operate anymore.
Portability	The command OUTB for CAN modules is available starting with MCO 5.00.
Command Group	I/O
Cross Index	OUT, IN, INB, Parameters: 33-63...70 *Terminal X59/n Digital Output*, O_FUNCTION_n
Syntax Examples	OUTB 0 10 // switch through outputs 2 and 4, disable other outputs OUTB 0 245 // disable outputs 2 and 4, switch through all other outputs OUTB 0 128 // switch through output 8 only, disable others OUTB 256 1 // set output 1 to CAN module 1
Program Sample	OUTB_01.M

☐ OUTDA

Summary	Sets FC 300 analog output.
Syntax	OUTDA n v
Parameter	n = output number (42) or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) v = value (0 – 100000)
	NB!: Parameter 6-50 must be set to "MCO controlled".
Description	With the OUTDA command it is possible to control the analog output of the FC 300 control card. FC 300 has one analog output. It is configured via parameter 6-50. A FC 300 control card output can only be controlled from the application program when it is configured as option output in the appropriate parameter. CAN modules which fulfill the CAN OPEN specifications can also be addressed with the IN command via the corresponding number, which is defined as follows: CAN-Bus + (Module-CAN-ID * 256) + output number (or output byte) When executing such a command the corresponding CAN objects are created temporarily, evaluated and subsequently released. Thus, it is possible to address any number of modules.
	**NB!:** The CAN commands operate with the pre-defined PDOs of CAN-OPEN. Don't change these default settings (minimum capability device), otherwise the CAN commands will not operate anymore.
Portability	Set output for CAN modules is available starting with MCO 5.00.
Command Group	I/O
Cross Index	FC 300 Design Guide, parameter 6-50
Syntax Example	/* condition: parameter 650 is set to "MCO controlled" */ OUTDA 42 50000 /* set FC 300 output to 10 mA */

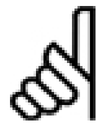
◻ OUTMSG

Summary	Sends a CAN message.
Syntax	OUTMSG intval longval
Parameter	intval Bytes 2 and 3 of the CAN-message longval Bytes 4 and 7 of the CAN-message
Return Value	–
Description	Sends a CAN message (buffered). The CAN Id (CAN identification number) results from the settings of the 'slaveno'. OUTMSG always deals with objects which are 8 bytes long. Only bytes 2 to 7 are intended for the user. Bytes 0 and 1 are reserved.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	INMSG, ON CANMSG, INAD
Syntax Example	temperature = INAD 1 OUTMSG 20 temperature

◻ PCD

Summary	Pseudo array for direct access to the field bus data area	
Syntax	PCD[n]	
Parameter	n = index	
Description	You can directly access the field bus data area with the command PCD without an additional command COMOPTGET or COMOPTSEND. The communications memory is written or read word by word (16-Bit).	
	NB!: The parameters 9-15 and 9-16 must additionally to be set with the correct values.	
Command Group	Communication option	
Cross Index	COMOPTGET, COMOPTSEND, SYSVAR	
Syntax Example	Variable = PCD[1]	// Word 1
	Variable = PCD[1].2	// Bit 2 of Word 1
	Variable = PCD[2].b1	// Byte 1 of Word 2
	PCD[1] = Variable	
	PCD[1].3 = Variable	
Syntax Example	_IF (PCD[2]= = 256) THEN	// compare value
	_IF (PCD[3].2) THEN	// is bit 2 of PCD3 high?

☐ PDO

Summary	Pseudo array for direct access to the CANopen PDOs.
Syntax	PDO[n]
Parameter	n = 1001 for 1. PDO (first 4 bytes), 1002 for 1. PDO (next 4 bytes) 2001 for 2. PDO (first 4 bytes), 2002 for 2. PDO (next 4 bytes) 3001 for 3. PDO (first 4 bytes), 3002 for 3. PDO (next 4 bytes) 4001 for 4. PDO (first 4 bytes), 4002 for 4. PDO (next 4 bytes) 5001 for 5. PDO (Serial PDO supported) Also supported out of compatibility reasons: n = 1, 2 (1. PDO, first and second 4 Bytes) The offset to read other PDOs is always 1000. The next 4 bytes of a PDO are accessed by an offset of +1. Depending on the system in use PDOs might hold even more than 8 bytes. The access to all consecutive bytes is done in the same manner, e.g. 1001, 1002, 1003, 1004, etc. The same scheme is valid for the "serial" PDO 5 (e.g. accessed by USB, RS232).
	PDO Activation (enable / disable) Just the PDO 1 (RxPDO = 0x200 + Node-ID / TxPDO = 0x180 + Node-ID) is enabled by default according to the CANopen specification. If other PDOs are required, these have to be enabled by direct setting the corresponding "Valid" bit (0x1400 - 0x1404 resp. 0x1800 - 0x1804, subindex 1) or by the configuration of mapped objects using the commands LINKSDO or LINKPDO, which set the "Valid" bit of the corresponding PDO automatically.
	CANopen PDO size A CANopen PDO is always 8 bytes long; it can therefore hold a maximum of 8 objects.
	PDO 5 (= "serial PDO") Size The mailbox size of the PDO 5 can be up to approx. 250 Bytes. The PDO 5 is also used by the oscilloscope tool of the APOSS development environment, therefore it is recommended to use this PDO not in applications, that shall be debugged using the oscilloscope tool later on.
Description	The command PDO enables direct access to the CANopen data process objects. During reading of a PDO (reaction to an incoming PDO) there are three possibilities: 1. After the command LINKPDO, the incoming telegram will always be diverted in the appropriate system variable. The "Valid" bit of the PDO is set automatically. 2. The program itself accesses and reads the PDO array. 3. Using ON COMBIT, a function is called up, as soon as the bit n of the PDO changes. In the case of outgoing PDO, you should differentiate between two processes: 1. After the command LINKSDO the PDO will be written in as soon as the variable changes. Cyclical update of the parameter every 10 ms. This default value can be changed via SDO entry 0x1800 - 0x1804 subindex 5 in accordance with the CAN Open specification. The "Valid" bit of the PDO is set automatically. 2. The command PDO writes directly in the outgoing PDO.

The first PDO array element (e.g. PDO[1001]) contains the data bytes 1 - 4 of the maximum possible 8 data bytes of the PDO in case of a CANopen PDO. The second PDO array element (e.g. PDO[1002]) contains the data bytes 5 - 8. More PDO array elements are just available for special systems or enhanced interfaces, e.g. the "serial" PDO 5.

Every PDO array element (e.g. PDO[1002, PDO[1002]) contains a 32-bit value. But the byte order within the 32 bit value must be attended when evaluating. The best way to check the corresponding location is with byte- (.b) or word-wise (.w) access to the PDO array element.

PDO[1].b1 -> Byte 1 of PDO
PDO[1].b2 -> Byte 2 of PDO
PDO[1].b3 -> Byte 3 of PDO
PDO[1].b4 -> Byte 4 of PDO

PDO[2].b1 -> Byte 5 of PDO
PDO[2].b2 -> Byte 6 of PDO
PDO[2].b3 -> Byte 7 of PDO
PDO[2].b4 -> Byte 8 of PDO

NB!:
The commands LINKPDO and LINKSDO link an internal system variable of the control unit directly to a PDO and set the "Valid" bit of this PDO.

Even when mapping was configured (by LINKPDO or LINKSDO), you can still access the PDO by the PDO array elements. However, you must take into consideration that only bytes should be directly written into the PDO-array, which are not in use by the mapping configuration. Otherwise the integrity of data is not guaranteed!

NB!:
As standard, a changed PDO content is automatically dispatched (asynchronous operating mode). If this is not desired, then the SDO Index 0x1800 - 0x1804 subindex 2 can be set to another value (e.g. 254, instead of the standard 255). Thereby, active sending no longer takes place automatically, but the PDO has to be collected per remote frame instead.

Of course, the PDO mapping can also be configured by a master device (e.g. PLC, PC) using the standard CANopen procedures and the mapping objects 0x1600 – 0x1604 resp. 0x1A00 – 0x1A04. In that case only SDOs can be mapped, but none of the internal control data, which can just be accessed by internal SYSVAR numbers.

In the same manner, it is also possible to enable or disable the PDOs 1 - 5 by modifying the "Valid" bit of the subindex 1 of the objects 0x1400 – 0x1404 resp. 0x1800 – 0x1804.

The subindices 2-5 of 0x1800 can be used to define transmission type, inhibit and event time of the TxPDOs. The defined transmission type of the TxPDO is also used for the RxPDO by MCO option units.

RxPDOs do not use extra CAN memory, but TxPDOs need one object per TxPDO, if they are enabled, i.e. the "Valid" bit is set.

Portability Command is available starting with MCO 5.00.

Command Group CAN

Cross Index SYSVAR

Syntax Example	Variable = PDO[1]	// PDO data byte 1 - 4
	Variable = PDO[1].2	// Bit 2 of PDO data byte 1 - 4
	Variable = PDO[2].b1	// Byte 1 of PDO data byte 5 - 8
	PDO[1] = Variable	
	PDO[1].3 = Variable	
Program Sample	IF (PDO [1] ==256) THEN	// compare value
	IF (PDO [2].2) THEN	// Is Bit 1 of PDO data byte 5 - 8 high?

PID

Summary	Calculates PID filter.
Syntax	u(n) = PID e(n)
Parameter	e(n) = actual deviation (error) for which the PID filter should be used
Return Value	u(n) = result of the PID calculation
Description	A PID filter can be calculated with this function. The PID filter works according to the following formula: u(n) = (KP * e(n) + KD *(e(n)-e(n–1)) + KI*∑e(n)) / timer where the following is true: e(n) error occurring at time n KP proportional factor of the PID control KD *Derivative Value* KI *Integral Factor* (limited by Integration Limit) timer controller sample time e(n) → KPROP / KINT → KILIM / KDER → u(n) The corresponding factors can be set with the following commands: SET PID KPROP 1 /* set KP 1 */ SET PID KDER 1 /* set KD 1 */ SET PID KINT 0 /* set KI 0 */ SET PID KILIM 0 /* Integration limit 0 */ SET PID TIMER 1 /* Sample time = 1 */ The following syntax example also show the default allocation of the factors.
Command Group	SYS
Syntax Example	e = INAD 53 u = PID e PRINT "input = ",e, "output = ",u

POSA

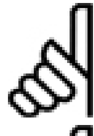
Summary	Positioning in relation to actual zero point.
Syntax	POSA p
Parameter	p = Position in user units (UU) absolute to the actual zero point; the UU corresponds in the standard setting the number of Quadcounts.
Description	The axis can be moved to a position absolute to the actual zero position. When the POSA command exceeds the *Negative or Positive Software End Limit* (parameters 33-41 or 33-42) the program continue with the next command after an error. **NB!:** If a temporary zero point, set via SETORIGIN, exists and is active, then the position result refers to this zero point. **NB!:** If an acceleration and/or velocity have not been defined at the time of the POSA command, then the procedure will take place with the values of parameters 32-84 *Default Velocity* and 32-85 *Default Acceleration*.
Command Group	ABS
Cross Index	VEL, ACC, POSR, HOME, DEFORIGIN, SETORIGIN Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*
Syntax Example	POSA 50000 /* move axis to position 50000 */
Program Sample	POS_01.M

☐ POSA CURVEPOS

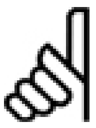
Summary Move slave to the curve position corresponding to the master position

Syntax POSA CURVEPOS

Description This command acts like POSA and moves the slave to the corresponding position on the curve, which is given by the actual master position.



NB!:
If a temporary zero point, set via SETORIGIN, exists and is active, then the position result refers to this zero point.



NB!:
If an acceleration and/or velocity have not been defined at the time of the POSA command, then the procedure will take place with the values of parameters 32-84 *Default Velocity* and 32-85 *Default Acceleration*.

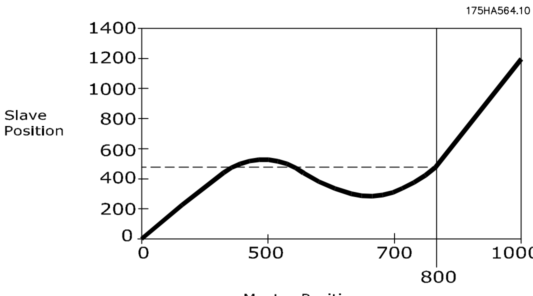
Command Group ABS, CAM

Cross Index CURVEPOS, SETORIGIN

Syntax Example POSA CURVEPOS
// Move slave to the curve position corresponding to the master position

Sample Fix points of a curve:

Master	Slave
0	0
500	500
700	300
1000	1200



Say the current master position is 800 (the darkened vertical line).

Case 1: Current Master Position is 800 and current slave position is 200. POSA CURVEPOS will move the slave to 450.

Case 2: Current Master Position is 800 and current slave position is 700. POSA CURVEPOS will move the slave to 450.

☐ POSR

Summary Positioning in relation to actual position

Syntax POSR d

Parameter d = distance to actual position in user units (UU); this corresponds in the standard setting to the number of Quadcounts.

Description The axis can be moved to a position relative to the actual position by use of the POSR command.

NB!:
If acceleration and/or velocity has not been defined at the time of the POSR command, then the procedure will take place with the values of parameters 32-84 *Default Velocity* and 32-85 *Default Acceleration*.

Command Group REL

Cross Index VEL, ACC, POSA,
Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*

Syntax Example POSR 50000 /* move axis relative 50000 UU */

Program Sample POS_01.M

☐ PRINT

Summary	Information output
Syntax	PRINT i or PRINT i;
Parameter	i = information, for example, variables, text, CHR (n) separated by commas. The CHR command returns the ASCII characters corresponding to a certain number.
Description	Calculation results, variables contents and text information can be displayed on the connected PC via the RS485 communication interface by use of the PRINT command, if the APOSS software is open and the connection active. To obtain multiple data with a single PRINT command, the individual elements must be separated with a comma (,). Text information must be given in quotation marks ("). A line feed is normally created following each PRINT instruction. This automatic line feed can be suppressed with a semi-colon (;) after the last output element.
Command Group	SYS
Cross Index	INKEY
Syntax Example	PRINT "Information is important !" /* print text information */ PRINT "Information is important !"; /* print information without line feed */ variable = 10 PRINT variable /* print contents of variables */ PRINT APOS /* print returned value function */ PRINT "Variable", variable,"Pos.:",APOS /* print mixed information */
Program Sample	Uses – see all Program Samples.

☐ PRINTDEV

Summary	Stops information output
Syntax	PRINTDEV nn printlist
Parameter	nn = number for the print device 0 = Standard output -1 = no output after that line 1 = CAN bus 2 = serial printlist = normal argument for a print command
Description	PRINTDEV can be used to disable all prints in a program without commenting them out one by one. **NB!:** The instruction (-1) defines the standard device new and is immediately effective for all PRINT commands, that do not hold a DEV.
Command Group	SYS
Cross Index	PRINT, INKEY
Syntax Example	PRINTDEV –1 "start" ... PRINT "normal print " ... PRINTDEV 0 "now print again info"

☐ PULSACC

Summary	Set acceleration for the virtual master.
Syntax	PULSACC a
	NB!: Changes in the acceleration in PULSACC are only valid after the next PULSVEL command.
Parameter	a = acceleration in Hz/s
Description	With PULSACC it is possible to set the acceleration/deceleration for the virtual master (encoder output). The virtual master signal simulates an encoder signal. To calculate the pulse acceleration PULSACC the parameter *Encoder Resolution*, the master velocity and the ramp times must be taken into consideration. The signals generated are evaluated simultaneously as master input so that MAPOS, MIPOS, etc. function as they would in an external master. The virtual encoder signal can just be outputted, if a control unit with an encoder output is in use. The actual port number of the encoder output depends on the hardware. The configuration and activation of the virtual encoder (i.e. PULSACC unequal 0) automatically configures the encoder port as an output. PULSACC = 0 is the condition for switching off the virtual master mode, provided it is followed by a PULSVEL command.
Command Group	SYN
Cross Index	PULSVEL
Example	The virtual master signal should correspond to an encoder signal of 1024 counts/revolution. The maximum speed of 25 encoder revolutions/s should be achieved in 1 s. $\text{PULSACC} = \frac{\Delta \text{ pulse velocity (PULSVEL) [Hz]}}{\Delta \text{ t [s]}}$ $= \frac{25 \text{ counts/s} * 1024 \text{ counts/revolution}}{1 \text{ s}}$ $= 25600 \text{ counts/s}^2 = 25600 \text{ Hz/s}$

PULSVEL

Summary	Set the velocity for the virtual master.
Syntax	PULSVEL v
Parameter	v = velocity in pulses per second (Hz)
Description	With PULSVEL it is possible to set the velocity for the virtual master (encoder output). The virtual master signal simulates an encoder signal. To calculate the pulse velocity the parameters *Encoder resolution* and master velocity must be taken into consideration.
Command Group	SYN
Cross Index	PULSACC
Example	The virtual master signal should correspond to an encoder signal of 2048 counts /revolution within an encoder speed of 50 revolutions/s. $\text{PULSVEL} = \text{encoder counts per turn} * \frac{\text{turns}}{\text{s}}$ $= 2048 * 50\,\text{Hz} = 102400\,\text{Hz}$

REPEAT .. UNTIL ..

Summary	Conditional loop with end criteria (Repeat … until condition fulfilled)
Syntax	REPEAT UNTIL Condition
Parameter	condition = Abort criteria
Description	The REPEAT..UNTIL construction enables any number of repetitions of the enclosed program section, dependent on abort criteria. The abort criteria consist of one or more comparative procedures and are always checked at the end of a loop. As long as the abort criteria are not fulfilled, the loop will be processed repeatedly. **NB!:** Because the abort criteria are checked at the end of the loop, the commands within the loop will be carried out at least once. To avoid the possibility of an endless loop, the processed commands within the loop must have a direct or indirect influence on the result of the abort monitoring.
Command Group	CON
Cross Index	LOOP, WHILE .. DO .. ENDWHILE
Syntax Example	REPEAT /* start loop */ command line 1 command line n UNTIL (A != 1) /* Abort condition */
Program Sample	REPEA_01.M, DIM_01.M, ONINT_01.M, OUT_01.M, INKEY_01.M

RSTORIGIN

Summary	Erase temporary zero point
Syntax	RSTORIGIN
Description	A previously with SETORIGIN set temporary zero point can be erased by use of the RSTORIGIN command. This means that all the following absolute positioning commands (POSA) again refer to the real zero point.
Command Group	INI
Cross Index	SETORIGIN, DEFORIGIN, POSA
Syntax Example	RSTORIGIN /* reset temporary zero point */
Program Sample	TORIG_01.M, OUT_01.M, VEL_01.M

◻ SAVE part

Summary	Save arrays or parameters in the EPROM
Syntax	SAVE part part = ARRAYS, AXPARS, GLBPARS, or USRPARS
Description	If array elements or parameters are altered while the program is running the altered values can be saved individually in the EPROM with these commands: SAVE GLBPARS — saves the range of global parameters (group 30-5* and group 33-8*), and application parameters (group 19-**) in the EPROM. SAVE AXPARS — saves all other axes parameters. SAVE USRPARS — saves only application parameters (group 19-**).
	NB!: The EPROM can only handle execution of this command up to 10000 times.
Command Group	INI
Cross Index	DELETE ARRAYS, SAVEPROM

◻ SAVEPROM

Summary	saves memory in EPROM
Syntax	SAVEPROM
Description	When changing array elements or application parameters (group 19-**) while the program is running SAVEPROM offer the possibility of saving the values which have been changed. This must be done by triggering the command SAVEPROM explicitly. SAVEPROM triggers the same process, which can also be started in the menu *Controller*. If you want to save only array elements or only global and application parameters, use the corresponding commands SAVE .. ARRAY, GLBPAR, or USRPARS.
	NB!: The execution time of SAVEPROM depends on the amount of data to be saved. It can be up to 4 seconds.
	NB!: Please note that the MCO parameters (group 32-** and 33-**) are not saved by SAVEPROM. To do this you must use the command SAVE AXPARS.
	NB!: The EPROM can only handle execution of this command up to 10000 times.
Command Group	INI
Syntax Example	PRINT "please wait" SAVEPROM PRINT "Thanks"

◻ SDOREAD

Summary	Reads SDO of a connected CANopen device.
Syntax	val = SDOREAD id index sub
Parameter	id = CAN id (1...127) -id = executes the command without waiting for the answer
	index = index of object (0x0000...0xFFFF)
	sub = sub index (0x00 – 0xFF)
Return Value	value of the SDO with index and sub index
Description	This command allows reading SDO of a connected CANopen device.
	After doing the SDO read, the value is given back in *val*. In case of problems an APOSS error will be reported.
	It is possible to call SDOREAD with negative CAN-Id numbers. Then the command is executed but it will not wait for the answer. However, SDOREAD does not return a meaningful result in such a case. Therefore use SDOSTATE to check the result of an active communication.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	SDOWRITE, SDOSTATE

☐ SDOREADSEG

Summary	Segmented read of SDOs (unpacked).
Syntax	res = SDOREADSEG id, index, subindex, arrayname
Parameter	id = CAN id number -id = führt den Befehl ohne Warten auf eine Antwort aus index = 0x2000 subindex = parameter number arrayname = name of a existing array
Return Value	value in the array given as parameter: 1 byte in 1 array element
Description	The command allows segmented read of unpacked SDOs. This is especially useful for strings or binary data. SDOREADSEG does a segmented read (if possible) and delivers the result in the array given as parameter. This array contains one byte of the segmented data (character) in one array element. See the example below of an SDOREAD with waiting and unpacked. The command can be used with waiting for result and producing an error if something goes wrong, or without waiting and not producing errors. In the second case, the result must be checked by the SDOSTATE command. For results, see there.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	SDOREAD, SDOSTATE
Example	see below

```
DIM test[20]
id = 3
// Routine to read a string
// thereby the array contains only one character per element
long printstring (long[] arr, long len)
{
   long ind
   ind = 1
   WHILE(ind <= len) DO
      PRINT chr(arr[ind]);
      ind++
   ENDWHILE
   PRINT " "
}

// Test segmented SDO unpacked with waiting
PRINT ""
PRINT "Test segmented SDO unpacked with waiting "
subindex = 1549  // Sw Version
value = SDOREADSEG id, 0x2000, subindex, test
PRINT "number characters ",value
printstring(test,value)
```

☐ SDOREADSEGP

Summary	Segmented read of SDOs (packed).
Syntax	res = SDOREADSEGP id, index, subindex, arrayname
Parameter	id = CAN id number -id = führt den Befehl ohne Warten auf eine Antwort aus index = 0x2000 subindex = parameter number arrayname = name of a existing array
Return Value	value in the array given as parameter: 4 bytes in 1 array element
Description	The command allows segmented read of unpacked SDOs. This is especially useful for strings or binary data. SDOREADSEGP does a segmented read (if possible) and delivers the result packed in the array given as parameter. The command SDOREADSEGP packs the bytes into the array elements. That means every array element contains four bytes. See the example below of a packed read without waiting. The command can be used with waiting for result and producing an error if something goes wrong, or without waiting and not producing errors. In the second case, the result must be checked by the SDOSTATE command. For results, see there.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	SDOREAD, SDOSTATE
Example	see below

```
DIM test[20]
vltid = 3
// Routine to dump a packed string
// thereby each element contains max 4 characters
long printpackedstring (long[] arr, long len)
{
   long ind,cc,rel
   ind = 1
   cc  = 1
   WHILE(cc <= len) DO
      ind = ((cc-1)  % 4) + 1
      rel = ((cc-1) mod 4) + 1
      PRINT chr(arr[ind].b rel);
      cc++
   ENDWHILE
   PRINT " "
}
// Test of the segmented SDO packed without waiting
PRINT ""
PRINT "Segmented SDO with SdoReadSegP packed without waiting"
para = 1549
res = 0
value = SDOREADSEGP -vltid, (0x2000 + para), 0, test
WHILE(value == 0) DO
   value = SDOSTATE vltid res
ENDWHILE
IF(value > 0) THEN
   PRINT "Parameter ",para," has the length ",res
   printpackedstring(test,res)
ELSE
   PRINT "Read of ",para," failed - error ",value
ENDIF
```

☐ SDOSTATE

Summary Checks the result of an active communication.

Syntax res = SDOSTATE id value

Parameter

id	=	CAN id
value	=	additional return value, whose meaning depends of the return value “res”

Return Value

0	=	id is busy, waiting for answer
1	=	answer arrived, result available in value
2	=	segmented read complete, data is in the array, number of bytes received is in value
-2	=	timeout occurred while waiting for answer value = (index << 8 + subindex)
-12	=	CAN-Error occurred (bus error) (will be reset internally)
-xx	=	internal errors in firmware
-33	=	id not in use (id is free for new communication)
-50	=	SDO was aborted by slave, SDO abort code is available in value
-51	=	array was too small for the segmented read (minimum size in value)
-52	=	toggle bit error in segmented transfer
-53	=	too much data came in (more than stated in the SDOREAD)
-54	=	not enough data received when done was detected.
-55	=	array write error in segmented read

Description It is possible to call SDOREAD or SDOWRITE with negative Can-Id numbers. In such a case, the command is executed but it will not wait for the answer. However, SDOREAD does not return a meaningful result in such a case.

Therefore SDOSTATE allows a check for the result of an active communication. SDOSTATE returns 0 as long as communication is busy. It returns negative results in the case of an error (TIMEOUT, SDO-Abort, ..). If SDOSTATE returns a positive result, the last SDO command was completed successful. If the last command was an SDOREAD, the SDOSTATE will return the result in the parameter value.

It is possible to start several transactions (up to 5) to different IDs in parallel. (Of course not with the same ID). For example, three SDOREADS can be send out to IDs 1,2,3 and then poll for the results.

Portability Command is available starting with MCO 5.00.

Command Group CAN

Cross Index SDOWRITE, SDOREAD

Syntax Example

```
id = 1
value = 0
res = SDOREAD -id idx subidx
WHILE(erg == 0) DO
   res = SDOSTATE id value
ENDWHILE
IF(res > 0) THEN
   // use value as result of the SDOREAD
ELSE
   // handle the error case
ENDIF
```

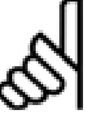
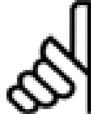
☐ SDOWRITE

Summary	Sets SDO of a connected CAN-open device.
Syntax	SDOWRITE id index sub val
Parameter	id = CAN id (1...127) -id = executes the command without waiting for the answer index = index of object (0x0000...0xFFFF) sub = sub index (0x00 ... 0xFF) val = parameter value
Return Value	–
Description	This command allows writing a SDO to a connected CAN-open device. After doing the SDO write, the value in 'val' is written to the corresponding object. In case of problems an APOSS error will be reported. It is possible to call SDOWRITE with negative CAN-Id numbers. Then the command is executed but it will not wait for the answer. However, SDOWRITE does not return a meaningful result in such a case. Therefore use SDOSTATE. to check the result of an active communication.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Cross Index	SDOREAD, SDOSTATE.
Syntax Example	SDOWRITE 127 0x2300 2 3000 // sets max. velocity to 3000 rpm

☐ SET

Summary	sets a parameter
Syntax	SET par v
Parameter	par = Parameter identification v = parameter value
Description	With the SET command parameters can be temporarily changed while the program is running. The parameter codes permitted can be found in chapter "Parameter Reference" in the MCO 305 Design Guide. **NB!:** The parameter alterations are only valid while the program is running. After program end or abort, the original parameter values are valid again. The parameter alterations can be made permanent by using the command SAVEPROM.
Command Group	PAR
Portability	SET I_xxx commands will still work and automatically be transferred into the new I_FUNCTION parameters.
Cross Index	GET, Parameter Reference
Syntax Examples	SET POSLIMIT 100000 /* set positive positioning limit */ SET KPROP 150 /* change proportional factor */ SET PRGPAR 2 /* change activated program number */ SET I_FUNCTION_9_n /* previously the command SET I_BREAK */

☐ SETCURVE

Summary	Sets CAM curve.
Syntax	SETCURVE array
Parameter	array = name of the array or of the curve
Description	SETCURVE defines the actual used curve, which is described in 'array'. This command has to be used, before the commands CURVEPOS, SYNCCxx, SYNCCSTART, or SYNCCSTOP can be used. When this command is executed, the necessary pre calculations are done. See CAM extensions and new curve types in section Curve Arrays and Curve Types.
	**NB!:** The DIM instruction with the name of the curve or array and the number of array elements must stand in front of the command SETCURVE or at the beginning of the program. If there are several arrays or curves in the zbc (or cnf) file, then the order in the DIM instruction must match the order of the arrays in the zbc-file.
	**NB!:** <u>If SYNCC is not active:</u> If SETCURVE is used while SYNCC is not active, then SETCURVE will reset the curve master position depending on the actual master position. That means, CMASTERCPOS (SYSVAR 4230) is calculated out of MAPOS. This position is not longer reset by SYNCC. This Position can only be reset by a DEFMCPOS or by a new SETCURVE outside of SYNCC-mode. <u>If SYNCC is active:</u> If SETCURVE is used while SYNCC is active, the CMASTERCPOS will not be changed. All other parameters like 32-11 *User Unit Denominator*, 32-12 *UU Numerator*, 33-23 *Start Behavior for Sync.*, 33-15 and 33-16 *Marker Number for Master* and *for Slave*, 33-17 and 33-18 *Master* and *Slave Marker Distance*, 33-21 and 33-22 *Master and Slave Marker Tolerance Window*, and all Curve-Array information will be updated, after the next restart of the curve. While SYNCC is active, the only way to influence the CMASTERCPOS is a DEFMCPOS (which is executed with next restart of curve) or MOVESYNCORIGN which is executed immediately. CMASTERCPOS (SYSVAR) and CURVEPOS are now updated even if SYNCC is no longer active. The update of these values will be started after a SETCURVE command (if SYNCMSTART is < 2000) or after SYNCC and the first master marker (if SYNCMSTART = 2000).
	NB!: Transferring the array to the DSP may take some ms. A curve array of 900 values will take around 40 ms. For that reason the maximum array size is 2000. (Most curves have not more than some hundred values.) See also illustration Curve Array in chapter Technical Reference.
Portability	CAM enhancements and new types of fix points are available starting with MCO 5.00. With MCO 5.00 no interpolation points are used anymore.
Command Group	PAR
Cross Index	DIM, CMASTERCPOS (see axis process data), CURVEPOS, Curve Arrays and Curve Types in „Array Structure of CAM Profiles" in chapter Technical Reference
Syntax Example	DIM curve [280] // See number of elements in the title bar of the CAM-Editor SETCURVE curve

☐ SETMORIGIN

Summary	Set any position as the zero point for the master.
Syntax	SETMORIGIN value
Parameter	value = absolute position
Description	With the SETMORIGIN command you can set any position as the new zero point for the master.
	NB!: The command SETMORIGIN cancels the command DEFMORIGIN.
	NB!: Thus, to alter the zero point for the master again, you have to reset it with SETMORIGIN or DEFMORIGIN. RSTORIGIN does not have any effect on the zero point for the master.
Command Group	INI
Cross Index	DEFMORIGIN, MAPOS
Syntax Example	SETMORIGIN 10000 /* Set the zero point for the master at 10000 */

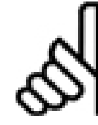
☐ SETORIGIN

Summary	Set absolute position as temporary zero point
Syntax	SETORIGIN p
Parameter	p = absolute position in relation to the real zero point
Description	Any absolute position can temporarily be set as a new reference point for absolute positioning command (POSA) by use of the SETORIGIN command. This position is called temporary zero point. In combination with the command CURVEPOS, one can fix in this way that the current slave position matches the corresponding value of the curve.
	NB!: It is possible to carry out several SETORIGIN commands without carrying out a previous RSTORIGIN. The absolute position value always refers to the real zero point. The last carried out SETORIGIN command therefore determines the position of the temporary zero point in relation to the real zero point.
Command Group	INI
Cross Index	RSTORIGIN, DEFORIGIN, POSA, CURVEPOS
Syntax Example	SETORIGIN 50000 /* set temporary zero point to 50000 */
Syntax Example	SETORIGIN (–CURVEPOS) // Set temporary zero to the beginning of the curve
Program Sample	TORIG_01.M, OUT_01.M, VEL_01.M

☐ SETVLT

Summary	Sets a FC 300 parameter
Syntax	SETVLT par v
Parameter	par = parameter number v = parameter value
Description	With the SETVLT command FC 300 parameters can be changed temporarily and thus the configuration of the FC 300 can also be changed temporarily. Since only integer values can be transmitted the parameter value to be transmitted must be adjusted with the associated conversion index. A list of the FC 300 parameters with the corresponding conversion index can be found in the FC 300 manual.
	NB!: The parameter alterations are only stored in RAM. After power down the original parameter values are restored.
Command Group	PAR
Cross Index	GETVLT
Syntax Example	/* change par. 3-03 "maximum reference" high to 60 Hz */ /* -Conversion index = -3 (Multiplied with 10^3 during transmission) */ SETVLT 303 60000

☐ SETVLTSUB

Summary	Sets a FC 300 parameter with index number.
Syntax	SETVLTSUB par indxno v
Parameter	par = parameter number indxno = index number v = parameter value
Description	With the SETVLT commands FC 300 parameters can be changed temporarily and thus the configuration of the FC 300 can also be changed temporarily, in this case parameters with index numbers too. Since only integer values can be transmitted the parameter value to be transmitted must be adjusted with the associated conversion index. A list of these parameters with the corresponding conversion index can be found in the FC 300 manual.
	**NB!:** The parameter alterations are only stored in RAM. After power down the original parameter values are restored.
Command Group	PAR
Cross Index	GETVLTSUB
Syntax Example	SETVLT 025 1 100 // sets index 1 of the par. 0-25 "Quick menu" to 100 "configuration"

☐ STAT

Summary	Query axis and control status.
Syntax	res = STAT
Return Value	res = Axis and Control status (4-Byte value): Byte 3 MSB Bit 0 1 = MOVING Bit 1 1 = OVERFLOW Slave Encoder Bit 2 1 = OVERFLOW Master Encoder Bit 3 1 = POSFLOAT active *) Byte 2 Status byte of axis control Bit 7 1 = axis control switched off Bit 2 1 = position reached Bit 0,1,3-6 has no meaning Byte 1 not used Byte 0 LSB Bit 7 1 = limit switch active Bit 6 1 = Reference switch active Bit 5 1 = Start switch active Bit 2 1 = axis control switched off Bit 0,1,3,4 not in use *) Explanation: i.e. the axis is within the tolerance range of the control window REGWINMAX / REGWINMIN. As soon as the control window is set, the axis controller is switched on again. SYSVAR 4258 PFG_LASTERROR gives more information about the type of error.
Description	The STAT command reports the actual status, of the axis control unit as well as that of the axis. For example, whether the axis controller shuts down, ends the motion or the end switch is active. The status of the program execution cannot be called up with STAT, but only with AXEND.
Command Group	SYS
Cross Index	AXEND
Syntax Example	PRINT STAT /* print status word */
Program Sample	STAT_01.M

□ SUBMAINPROG .. ENDPROG

Summary	Subroutine section definition
Syntax	SUBMAINPROG ENDPROG
Description	The code word SUBMAINPROG begins the subroutine section, and the code word ENDPROG ends this specific program. The term subroutine means command sequences that, via the GOSUB instructions, can be called up and executed from various program positions. All necessary subroutines must be contained within the subroutine section. It is possible to insert a subroutine anywhere within a main program; however, for reasons of clarity, it is advisable to insert it either at the beginning or end of a program.
	NB!: Only one subroutine area may be inserted within a program.
Command Group	CON
Cross Index	SUBPROG .. RETURN, GOSUB, ON ERROR GOSUB, ON INT n GOSUB
Syntax Example	SUBMAINPROG /* Begin the subroutine section */ subroutine 1 subroutine n ENDPROG /* End the subroutine section */
Program Sample	GOSUB_01.M, AXEND_01.M, ERROR_01.M, INCL_01.M, STAT_01.M

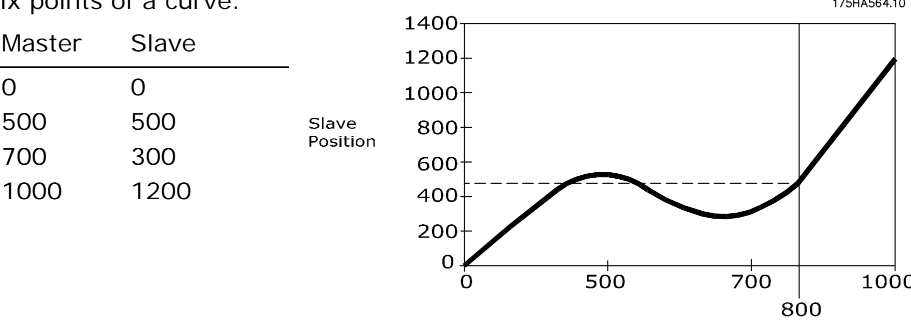
□ SUBPROG name .. RETURN

Summary	Subroutine definition
Syntax	SUBPROG name RETURN
Parameter	name = subroutine name
Description	The instruction SUBPROG identifies the beginning of a subroutine. The name of the subroutine must directly follow SUBPROG code word. The name can be made up of one or more characters, and must be unique, i.e. only one subroutine may have that name. A subroutine can be called up and executed at any time by use of a GOSUB instruction. A subroutine can have any number of command lines and can refer to all program variables. The last command in each subroutine must be the RETURN instruction, which permits exiting the subroutine and continuing the program with the command following the GOSUB instruction.
	NB!: All subroutines must be contained within the SUBMAINPROG and ENDPROG defined areas. It is not admissible to declare a second subroutine within an existing subroutine.
Command Group	CON
Cross Index	SUBMAINPROG .. ENDPROG, GOSUB, ON ERROR GOSUB, ON INT .. n GOSUB
Syntax Example	SUBMAINPROG /* begin SP-section */ SUBPROG sp1 /* begin sp1 */ command line 1 command line n RETURN /* end sp1 */ ENDPROG /* end SP-section */
Program Sample	GOSUB_01.M, AXEND_01.M, ERROR_01.M, IF_01.M, STAT_01.M

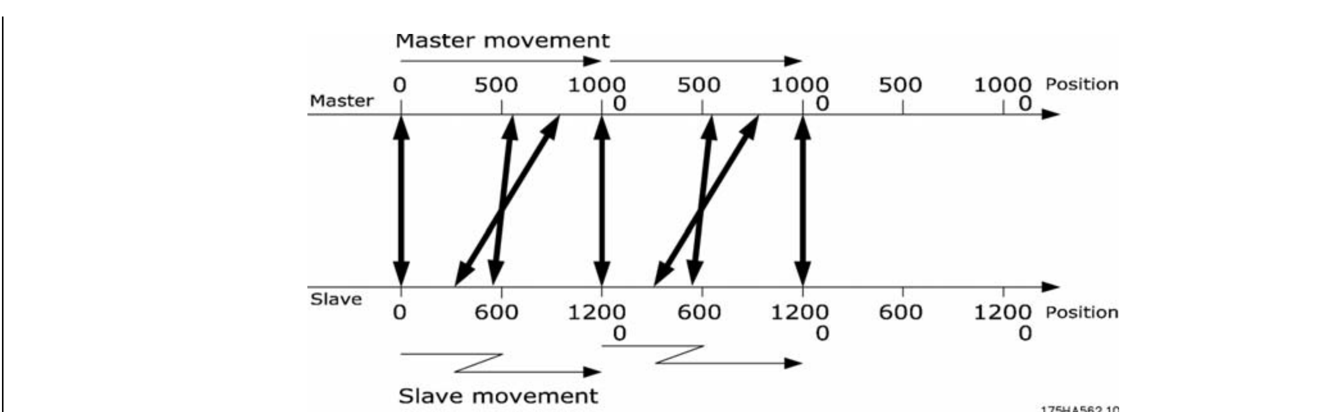
◻ SYNCC

Summary	Synchronization in CAM-Mode
Syntax	SYNCC num
Parameter	num = number of curves to be processed; 0 = the drive remains in CAM-Mode until another mode is started with commands such as MOTOR STOP, CSTART, POSA, etc. < 0 = starts SYNCC without resetting the markers
Description	The command SYNCC starts the CAM-Mode (CAM control). From this moment, the curve positions of the master are counted depending on the actual master positions and the defined starting behavior in par. 33-23 *Start Behavior for Sync*: Where and when counting is started. With the parameter SYNCMSTART = 2000, the curve positions of the master are only counted after the next master marker. **NB!:** SYNCC does not start the slave drive nor does it interrupt on-going motions (e.g. CVEL), only SYNCCSTART does. **NB!:** The drive remains in CAM-Mode until *num* curves have been processed successfully. If the synchronization (after *num* curves) is being closed normally, the start stop point pair 2 will be used – if no SYNCCSTOP with a corresponding point pair is defined – in order to stop the drive. It will then come to a stop at the position *slavepos* (see parameters). SYNCC can be started with a negative number. This will start SYNCC without resetting the markers. The negative number will then be reduced by one before the absolute value is used as Count. This continues to allow the usage of counts.
Portability	Using with negative numbers to start is available starting with MCO 5.00.
Command Group	CAM
Cross Index	SYNCCSTART, CAM-Editor
Syntax Example	DIM curve [280] // see number of elements in the title bar of the CAM-Editor SETCURVE curve // Set curve SYNCC // Synchronization in CAM-Mode
Sample	Fix points of a curve:

Master	Slave
0	0
500	500
700	300
1000	1200

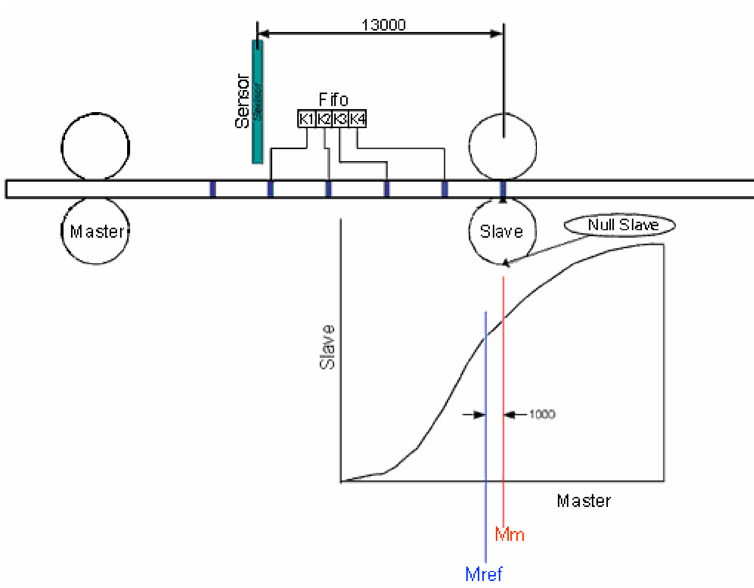


Hence in SYNCC 1 command thus locks the slave and master position as per the array.


Master movement
Master
0
500
1000
0
500
1000
0
500
1000
0
Position
Slave
0
600
1200
0
600
1200
0
600
1200
0
Position
Slave movement
175HA562.10

SYNCCMM

Summary	Synchronization in CAM-Mode with master marker correction
Syntax	SYNCCMM num
Parameter	num = number of curves to be processed; 0 = the drive remains in CAM-Mode until another mode is started with commands such as MOTOR STOP, CSTART, POSA, etc.
Description	Like SYNCC, the command SYNCCMM brings about synchronization in CAM-Mode, but beyond that it also performs a marker correction (only if the master moves forward). In order to save the distance between sensor and processing point, the par. 33-17 *Master Marker Distance* is used. It allows the correction of the marker position without changing the curve. Also, larger sensor distances than the actual curve length are possible. In this case, a FIFO is used for the marker correction (see example). The marker can be the zero pulse of the encoder or an external 24 V signal. **NB!:** SYNCCMM does not start the slave drive nor does it interrupt on-going motions (e.g. CVEL), only SYNCCSTART does. **NB!:** The drive remains in CAM-Mode until 'num' curves have been processed successfully. If the synchronization (after 'num' curves) is being closed normally, the start stop point pair 2 will be used – if no SYNCCSTOP with a corresponding point pair is defined – in order to stop the drive. It will then come to a stop at the position *slavepos* (see parameters).
Command Group	CAM
Cross Index	par. 33-17 *Master Marker Distance*
Syntax Example	SETCURVE curve SYNCCMM 1 // Synchronize 1 x in CAM mode with marker correction
Sample	If for example curve length is 3000 and distance of sensor to working point is 13000, we will have a FIFO with 4 Register and an offset of 1000 which has to be concerned. See the following diagram



☐ SYNCCMS

Summary	Synchronization in CAM-Mode with slave marker correction.
Syntax	SYNCCMS num
Parameter	num = number of curves to be processed; 0 = the drive remains in CAM-Mode until another mode is started with commands such as MOTOR STOP, CSTART,, POSA, etc.
Description	Like SYNCC, the command SYNCCMS brings about a synchronization in CAM-Mode, but beyond that it also performs a marker correction of the slave. Here, the slave position is corrected, not the curve position. In contrast to SYNCCMM, no FIFO is created. The marker can be the zero pulse of the encoder or an external 24 V signal. **NB!:** SYNCCMS does not start the slave drive nor does it interrupt on-going motions (e.g. CVEL), only SYNCCSTART does. **NB!:** The drive remains in CAM-Mode until 'num' curves have been processed successfully. If the synchronization (after 'num' curves) is being closed normally, the start stop point pair 2 will be used – if no SYNCCSTOP with a corresponding point pair is defined – in order to stop the drive. It will then come to a stop at the position 'slavepos' (see parameters).
Command Group	CAM
Cross Index	Par. 33-18 *Slave Marker Distance*
Syntax Example	SETCURVE curve SYNCCMS 0 // Synchronization in CAM-Mode with slave marker correction

☐ SYNCCSTART

Summary	Start slave for synchronization in CAM-Mode
Syntax	SYNCCSTART pnum
Parameter	pnum = Start stop points number pnum > 0 Engaging begins when the corresponding point A is reached, provided the master moves in positive direction; the engage curve is finished at point B. If point A and B are identical, the slave will be engaged with the set maximum velocity – i.e. without curve – as soon as the master has reached this point. pnum = 0 The slave will be engaged immediately with the set maximum velocity. It does not matter in what direction the master moves or whether it moves at all. pnum < 0 Again, the corresponding point pair is used, however, engaging begins at point B and is finished at point A, i.e. in negative direction.
Description	The command starts the movement of the slave. With *pnum*, the point pair is selected that determines in which master position the synchronization begins and where it should be finished. When moving forward, the synchronization begins at point A and is finished up to point B. When moving backward, it begins at point B and is finished up to point A.
Command Group	CAM
Cross Index	SETCURVE, Start-Stop-Points
Syntax Example	SETCURVE curve SYNCC 0 // CAM mode synchronization SYNCCSTART 1 // Engage slave at point A from start stop point pair 1

◻ SYNCCSTOP

Summary	Stop slave after the CAM synchronization
Syntax	SYNCCSTOP pnum slavepos
Parameter	pnum = Start stop points number
	pnum > 0 — Engaging begins when the corresponding point A is reached, provided the master moves in positive direction; the engage curve is finished at point B. If point A and B are identical, the slave will be engaged with the set maximum velocity – i.e. without curve – as soon as the master has reached this point.
	pnum = 0 — The slave will be engaged immediately with the set maximum velocity. It does not matter in what direction the master moves or whether it moves at all.
	pnum < 0 — Again, the corresponding point pair is used, however, engaging begins at point B and is finished at point A, i.e. in negative direction.
	slavepos = — Position where the slave is supposed to stand after disengaging.
	NB!: When moving forward, disengaging begins at point A and is finished at point B; vice versa when moving backward.
	NB!: If the program is closed without SYNCCSTOP command, disengaging occurs by default with the second point pair and a stop occurs at the *Slave Stop Position* defined in the *Curve Data*.
Description	This command stops synchronization without leaving SYNCC mode. The slave will be disengaged according to the point pair defined in *pnum*. Only then will the slave actually be stopped. When the stop point has been reached, the slave must be at *slavepos*.
Command Group	CAM
Cross Index	Slave-Stop-Position
Syntax Example	SETCURVE curve SYNCC 0 // Synchronize in CAM-Mode SYNCCSTART 1 // Start slave with start point pair 1 SYNCCSTOP 2 0 // Stop slave with stop point pair 2 at the slave position 0 or 3600
Sample	(see figure below)

Slave positions [degrees]
stamp beginning 120°, end 240°

Stamping roller = Slave

Conveyor belt = Master

0 1500 2500 4000

Master positions [1/10 mm]

Area where master and slave must be in sync

e75MD001.10

☐ SYNCERR

Summary Queries actual synchronization error of the slave

Syntax res = SYNCERR

Return Value res = actual synchronization error of the slave in UU and

a) as an absolute value when the value of the accuracy window is defined with a plus sign in the parameter SYNCACCURACY;

b) with polarity sign when in SYNCACCURACY the value of the window is defined with a minus sign.

Description SYNCERR returns the actual synchronization error in User Units UU. This is the distance between the actual master position (converted with drive factor and offset) and the actual position of the slave.

If the par. 33-13 SYNCACCURACY is defined by a minus sign, you can also determine whether the synchronization is running ahead (negative result) or running behind (positive result).

NB!:
Up to option card version < 5.00: SYNCERR only functions in synchronization mode. As soon as you exit SYNCM or SYNCP, the pulses are no longer counted. SYNCERR is only updated within a SYNC command.

With option card software 5.00 onwards the SYNCERR is also updated when SYNCP or SYNCM are not longer active, e.g. after a MOTOR STOP.

Command Group I/O

Cross Index TRACKERR, MAPOS, APOS,
Parameters: 33-12 *Position Offset for Synchronization* (SYNCPOSOFFS), 33-10 and 33-11 *Synchronization Factor Master* and *Slave*, 33-13 *Accuracy Window for Position Sync.* (SYNCACCURACY)

Syntax Example

```
PRINT SYNCERR        /* query actual synchronization error of the slave */
```

Samples SYNCACCURACY = 1000

Here the SYNCERR returns the absolute value 700.

SYNCACCURACY = 1000

SYNCERR will display the absolute value 700 even though the slave is ahead of master.

Here the SYNCERR returns the value of –700 showing that the slave is ahead of master.

◻ SYNCM

Summary Angle/position synchronization with the master with marker correction

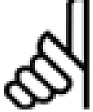
Syntax SYNCM

Description The SYNCM command functions just like the SYNCP command by making an angle/position synchronization with the master, but also makes a marker correction. Thus, during the starting of synchronization the program is synchronized to the next marker calculated. In this manner it is possible to compensate for differing running behaviors, for example slippage.

If RAMPTYPE >= 2, then synchronization is started with limited jerk. This only concerns the start; reaching the master velocity still looks like a trapezoid. This helps smoothen the start procedure in the case of heavy loads or fragile mechanics.

Once synchronization has been completed, deviation is determined at every marker (or every n-th marker if the number of markers is not identical for the master and slave). This is input into the control as the new offset and the program immediately attempts to compensate for this. However, in doing so the values set for velocity VEL, and acceleration ACC or DEC are not exceeded.

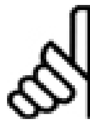
NB!:
In addition to the parameters used by SYNCP, par. 33-25 SYNCREADY, and par. 33-24 SYNCFAULT are also of significance.



NB!:
Since the following parameters could lead to overdefinition, it is important to ensure that these values are logical, match one another, and are consistent with the information on the gear factors.

par. 33-15, 33-16 *Marker Number for Master* and *for Slave*
par. 33-17, 33-18 *Master Marker* and *Slave Marker Distance*
par. 33-19, 33-20 *Master* and *Slave Marker Type*

NB!:
SYNCM should only be called up once since the synchronizing continues until the next motion or stop command. All additional SYNCM commands cause the synchronization to start over again from the beginning and this is not normally intended, as you reset the actual SYNCERR.



NB!:
When defined in par. 33-23 *Start Behavior for Sync.*, the system waits for the first evaluation of the marker pulses on starting SYNCM and only then the offset par. 33-12 *Position Offset for Synchronization* is applied.

Marker Signal The marker can be the zero pulse from the encoder or an external 24 volt signal (I5 = master; I6 = slave).

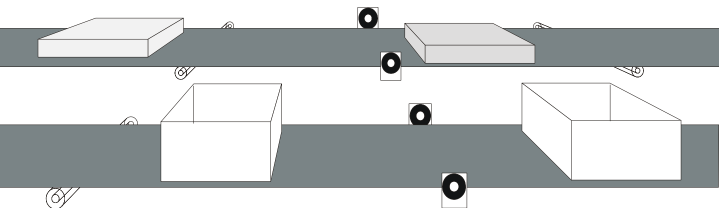
Portability Start behavior if RAMPTYPE >= 2 is available starting with MCO 5.00.

Command Group SYN

Syntax Example

```
SYNCM          /* synchronization of the position with marker correction */
```

Example



Even when both belts are running synchronously the lids may not be aligned with the boxes at the right time. With SYNCM the difference between master and slave is detected by means of the external markers and the possible position deviation is corrected.

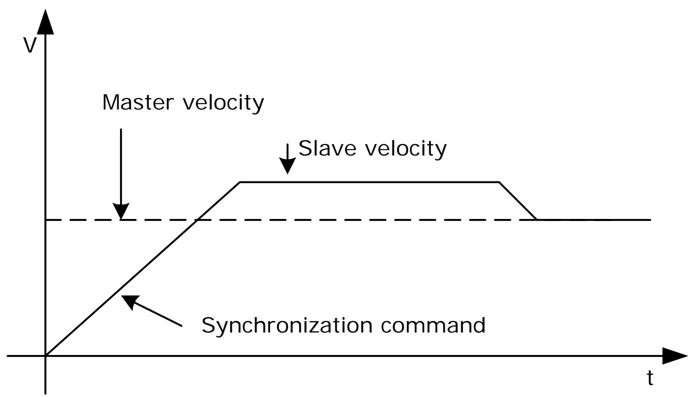
MG.34.R1.02 – VLT® is a registered Danfoss trademark

SYNCMARKERSTART

Summary	Resets a marker or resets marker handling.
Syntax	SYNCMARKERSTART restarttype
Parameter	restarttype = 0 or 1
Return Value	–
Description	This command replaces the SYNCMFPAR 64 functionality. In newer applications this command should be used instead of parameter SYNCMFPAR 64. **restarttype = 0** This command does a marker reset which means – Clears the flags (PG_FLAG_SYNCMMERR, PG_FLAG_SYNCSMERR) – Resets the SyncWindow, the correction values, the gear correction, the faked counts – Resets illegal counts, filtered marker distances, deviation values and all Mfilters. – Sets two Flags (SYN_ACCEPTNXTM, SYN_ACCEPTNXTS) in SYNCSTART This will lead to the following result. – Next master and next slave marker are accepted in all cases. – Correction is calculated corresponding to the actual value of SYNCMSTART – The resulting correction will be handled as a start correction – The n:m relation of the markers should be kept (if possible). – Marker Windows are reset corresponding to the parameter values – All filters restart with default values. This command also starts marker handling if not already active. This command can be used to start marker handling outside of SYNCM. (Former SYNCMFPAR – 64 functionality). It also can be used to restart a running SYNCM without loosing n:m relation. **restarttype = 1** A real Reset of the marker handling will be done. That means that as well as the things described in Marker Restart, we also reset the following values Markercounts – that means a n:m relation will be lost SYNCMSTART is reevaluated. This will possibly result in a new start sequence if SYNCM is active. SYNCSTART is set to start condition. That means all flags are set like SYN_START + SYN_NOSLAVE + SYN_NOMASTER + SYN_NOMVEL + SYN_ACCEPTNXTM + SYN_ACCEPTNXTS This feature was created to allow a complete marker restart outside of SYNCM. It does exactly the same as if you do a SYNCM command. So the next markers will be accepted and there will be no faking if no markers are seen. That is the main difference to the normal restart function.
Portability	Command is available starting with MCO 5.00.
Command Group	SYN
Cross Index	par. 33-28 SYNCMFPAR, par. 33-23 SYNCMSTART, SYNCM SYSVAR 4209 PFG_SYNCSTART see Axis Process Data

☐ SYNCP

Summary	Angle/position synchronization with the master
Syntax	SYNCP
Description	The command completes an angle/position synchronization with the master. In doing so the position according to the gear factors to the master is kept synchronous, that means after an external disturbance the program subsequently tries to recover the corresponding stretch.



However, in doing so the values set for velocity VEL, and acceleration ACC or DEC are not exceeded.

The following parameters affect the behavior:
par. 33-10 *Syncfactor Master* and
par. 33-11 *Syncfactor Slave* (gear factors)
par. 33-12 *Position Offset for Synchronization*
par. 33-13 *Accuracy Window for Position Sync.* (accuracy for flag)
par. 32-68 *Reverse Behavior for Slave*

During synchronization the program proceeds as follows:

When the SYNCP command is started, the actual master position is determined and is retained. From the master velocity, and in consideration of the acceleration allowed, the necessary slave velocity is calculated in order to reach the master position. The slave is accelerated so long until the calculated position has been reached or until it is close enough to the reference position to reach it.

If RAMPTYPE >= 2, then synchronization is started with limited jerk. This only concerns the start; reaching the master velocity still looks like a trapezoid. This helps smoothen the start procedure in the case of heavy loads or fragile mechanics.

NB!:
As soon as the deviation between the position of the slave and master is less than par. 33-13 SYNCACCURACY, the ACCURACY flag is set.

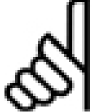
If par. 32-68 REVERS is set so that it is not possible to drive in reverse, but for some reason the slave is further than the master (e.g. because only the master has moved in reverse) then the slave will wait at velocity 0.

In doing so the slave takes its own acceleration time into consideration and will start moving if necessary before the correct position has been reached if the master already has a higher velocity.

Instead of using this catch-up procedure it is also possible to move the slave with CVEL to approximately the same velocity as the master and then trigger SYNCP.

A change in the par. 33-12 *Position Offset for Synchronization* during the synchronization leads to a new synchronization procedure with ramps (see above).

MG.34.R1.02 – VLT® is a registered Danfoss trademark



NB!:
SYNCP should only be called up once since the synchronizing continues until the next motion or stop command. All additional SYNCP commands cause the synchronization to start over again from the beginning and this is not normally intended, as you reset the actual SYNCERR.

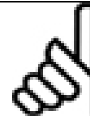
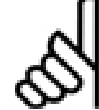
Portability	Start behavior if RAMPTYPE >= 2 is available starting with MCO 5.00.
Command Group	SYN
Syntax Example	SYNCP /* normal synchronization of the position */ CVEL 50 /* achieve velocity before synchronization */ CSTART WAITT 500 SYNCP

☐ SYNCSTAT

Summary	Flag to query synchronization status.
Syntax	res = SYNCSTAT
Return Value	res = synchronization status with the following meaning:

	Value	Bit
Par. 33-25 SYNCREADY	1	0
Par. 33-24 SYNCFAULT	2	1
Par. 33-13 SYNCACCURACY	4	2
SYNCMMHIT	8	3
SYNCSMHIT	16	4
SYNCMMERR	32	5
SYNCSMERR	64	6

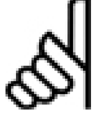
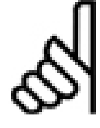
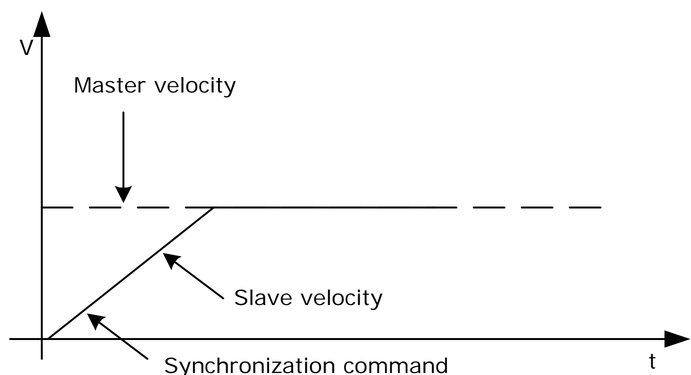
Description	The following flags are defined and can be queried with SYNCSTAT: READY, FAULT, ACCURACY and MHIT and MERR for both the master and the slave.
SYNCACCURACY	The controller checks whether SYNCERR < par. 33-13 SYNCACCURACY is true every ms. If this is true, then SYNCACCURACY is set, otherwise the flag is deleted. This check is made for both SYNCP and SYNCM. This flag is not used with SYNCV. When executing a SYNCP or SYNCM command the flag is deleted.
SYNCFAULT / SYNCREADY	For every SYNCP or SYNCM command these flags are deleted. Subsequently the program checks whether SYNCACCURACY is set or not at every marker pulse of the slave (SYNCP) or when a marker pulse of the master and a marker pulse of the slave exist (SYNCM). If it is set the ready counter is increased and the fault counter is set to 0, otherwise the fault counter is increased and the ready counter set to 0. If the ready counter is greater than the value determined by the par. 33-25 SYNCREADY, then the flag SYNCREADY is set. Otherwise it is deleted. If the fault counter is greater that the value determined by the par. 33-24 SYNCFAULT then the flag SYNCFAULT is set. Otherwise it is deleted.
SYNCMMHIT / SYNCSMHIT	SYNCMMHIT and SYNCSMHIT are set, if the master marker or the slave marker is occurred. These flags are deleted for every SYNCM command. Subsequently the flag SYNCMMHIT is set after the first occurrence of a master marker pulse or after the n-th marker pulse (par. 33-15 *Marker Number for Master*). The same is true for SYNCSMHIT with the slave.

	**NB!:** This flag is no longer deleted unless SYNCM is started again or explicitly deleted with SYNCSTATCLR.
SYNCMMERR / SYNCSMERR	If in the *Marker Windows* par. 33-21 SYNCMWINM or par. 33-22 SYNCMWINS a tolerance range is defined, then SYNCMMERR or SYNCSMERR are set as soon as the maximum distance allowed has been achieved and no marker was identified. Example: Distance between two master markers par. 33-17 = 30000 *Master Marker Tolerance Window* par. 33-21 SYNCMWINM = 1000 The flag is set at 31000 if no marker is identified. These flags are deleted for every SYNCM command. If the *Master Marker Tolerance Window* is 0 and thus no tolerance range is defined, the program checks every marker pulse (or after every n-th pulse) whether the distance between the two last markers registered is less than 1.8 times the value defined by the par. 33-17 *Master Marker Distance*. If not the corresponding flag is set. The same applies analogously for SYNCSMERR in the slave.
	**NB!:** These flags are automatically reset: during the next successful marker correction and in the event of a new start of SYNCM or through the command SYNCSTATCLR.
Command Group	SYN
Cross Index	SYNCSTATCLR
Syntax Example	IF (SYNCSTAT & 4) THEN OUT 1 1 /* If ACCURACY then set output */ ENDIF

□ SYNCSTATCLR

Summary	Resetting of the flags MERR and MHIT
Syntax	SYNCSTATCLR value
	The SYNCSTATCLR command should only be used in a subprogram for dealing with errors. (see ON ERROR GOSUB).
Parameter	value = 8 = SYNCMMHIT 16 = SYNCSMHIT 32 = SYNCMMERR 64 = SYNCSMERR
Description	The corresponding bits can be reset in SYNCSTAT with SYNCSTATCLR *value* thus resetting the error flags MERR and the HIT flags MHIT. None of the other flags can be altered.
Command Group	SYN
Cross Index	ON STATBIT, ON ERROR GOSUB, ERRNO, CONTINUE, MOTOR ON
Syntax Example	SYNCSTATCLR 32 /* clear current error message */

☐ SYNCV

Summary	Velocity synchronization with the master
Syntax	SYNCV
	**NB!:** SYNCV should only be called up once since the synchronizing continues until the next motion or stop command. All additional SYNCV commands cause the synchronization to start over again from the beginning and this is not normally intended, as you reset the actual synchronization error.
	**NB!:** Track and synchronization errors are not monitored in SYNCV mode, it is therefore recommendable to use the hardware encoder monitor.
Description	With SYNCV the velocity synchronization with the master is completed, for example after an external disturbance. In doing so only the velocity is taken into consideration and the controller does not attempt to recover the position.  For synchronization and during the synchronization process neither the pre-set velocity, VEL, nor the pre-set acceleration, ACC or DEC, are exceeded. The parameters of the gear factors used for synchronization are: par. 33-10 *Synchronizing Factor Master*, par. 33-11 *Synchronizing Factor Slave*. Furthermore, the speeds are not simply determined on the basis of the difference between the current position minus the last position (master/slave), rather the values are filtered according to the settings in par. 33-26 *Velocity Filter* (SYNCVFTIME). This means the filter for the slave is determined by the maximum speed. In other words: VELMAX * 5 corresponds to the encoder resolution for the filter table, where VELMAX is the speed in qc/ms. (The formula is a result of the assumption that the filter table for the encoder resolution was made with a maximum speed of 3000 RPM.) During the transition from the speed controller to the position controller this is done as smoothly as possible. In addition the new set position is defined in such a manner that the following is true: command_pos = actual_pos + error old_error, cvel, avel are maintained.
Command Group	SYN
Cross Index	Parameters of the AXS group

☐ SYSVAR

Summary	System variable (Pseudo array) reads system values.
Syntax	SYSVAR[n] n = index
Description	With the system variable SYSVAR – a prepared pseudo array – it is possible to read system and process data. This index can also be used to link the system variable using LINKPDO or LINKSDO or specify recording data with TESTSETP or TESTSETINDEX. This index can also be used, if you link the system variable with the LCP display using LINKSYSVAR. For compatibility reasons it is still possible to use these SYSVAR numbers. But it is recommended to use the SDO dictionary number for addressing: 0x01nnnnss = Can Index nnnn, Subindex ss see SDO Dictionary
	NB!: The values of the system variables are internal, hardware-dependent values which may change.
Portability	CANopen
Command Group	CON
Cross index	LINKPDO, LINKSDO, TESTSETP, TESTSETINDEX, Axis Parameters (CAN-SDO Number 0x2300), see SDO Dictionary in the "Appendix" in the online help..

System Process Data

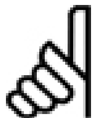
Index	Description System Process Data
1	Input Byte 0 (inputs 1..8 from MCO 305)
2	Input Byte 1 (inputs 18..33 from CC)
3	Input byte 2 (inputs 9..10 / 12 from MCO 305)
9	Output Byte 0
17	Top 2 bytes which are provided by the APOSS command STAT
22	Internal millisecond counter. Value which is also supplied by the APOSS command TIME
28	Actual motor current [1/100 Amp.]; (parameter 16-14)
30	Motor voltage [1/10 V]; (parameter 16-12)
31	FC 300 status (parameter 16-03)
32	Main actual value (parameter 16-05)
33	Current line number of the APOSS program in case of #DEBUG NOSTOP
34	Motor frequency (parameter 16-13)
35	Motor torque (parameter 16-16)

☐ TESTSETDEST

Summary Specifies the memory section for saving recorded data.

Syntax TESTSETDEST arrayname

Parameter arrayname = name of the array used for the recording or keyword DYNMEM for recording data into free memory.



NB!:
Please make sure that the size of the array or available memory (DYNMEM) is sufficient for the recording. You need 10 elements for the header describing the data recording configuration, plus the number of test series (if the array holds more than one), plus the number of configured system variables multiplied with the number of samples. Thus for 100 samples of 6 system variables (i.e. TESTSTART 100) you need an array size of 611 elements or in case of DYNMEM usage, the free memory has to be 611 x 4 bytes= 2'444 bytes.

The structure of the recorded data is as follows:
(All data values have a length of 4 bytes.)

Designation	Content	Meaning
version	999	Identification mark and version of the data structure
ms	1	Interval between two measurements in ms
no. of indices	3	Number of system variables defined (always 3, if configured by command TESTSETP)
svi 1	i	Index of the 1. recorded system variable (sysvar)
svi 2	i	Index of the 2. recorded system variable (sysvar)
...	i	Index of the ....
svi n	i	Index of the n. recorded system variable (sysvar)
no. of samples	nn	Number of recorded samples
first entry count	0	Relative count of the first sample in the recorded data: In case of one time recording: 0 In case of cyclic recording: where to start reading
stop time	...	Internal system time, when recording was stopped
data	...	Recorded data, total no of values = no. of samples * no. of variables (svi 1 ... svi n)
no. of test series	0	Number of test series (if more than one is present) 0 or missing, if there are no more test series
more test series	...	(see above)

NB!:
If the given memory section (array or DYNMEM) does not have sufficient space for the defined number of measurements (parameter no > 0), the error 171 "Array too small" or error 194 "DYNMEM too small" is triggered during program execution. It is recommended to use TESTSTART 0, which fills up the given memory in maximum.

Description TESTSETDEST defines, where the recorded data has to be saved. There are the following two possibilities:
- A (big enough) array, which was defined at program start (DIM ...).
- The free memory, which is indicated by the keyword DYNMEM.

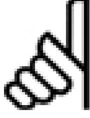
If a TESTSETP is executed after a TESTSETDEST command, then the destination given by the TESTSETP command is valid. Always the last setting before the TESTSTART is the one, which is in use. If no TESTSETDEST or TESTSETP command is executed at all, the DYNMEM is used as default destination.

Portability	Command is available starting with MCO 5.00. To read the resulting data with APOSS, a current version of APOSS should be used. Current versions of APOSS provide a simple way to select between arrays and DYNMEM. Older versions of APOSS can be used if the array number is set to the special value of 65535.
Command Group	SYS
Cross Index	TESTSETP, TESTSETINDEX, TESTSETTIME, TESTSETTYPE, TESTSTART, TESTSTOP

Syntax Example	DIM testdata[6011] // Define an array (used for data recording later on)
	TESTSETDEST testdata // Define an array as the destination
	// Define the system variables to be recorded:
	// Slave actual pos., Slave command pos., Master pos.,
	// Tracking error, Synchronization error, Slave velocity
	TESTSETINDEX 4096, 4097, 4105, 4101, 4207, 4186
	TESTSTART 1000 // Start recording of 1000 samples
	SYNCP // Start position synchronization

Syntax Example	// Define the system variables to be recorded:
	// Slave actual pos., Slave command pos., Tracking error, Slave velocity
	TESTSETINDEX 4096, 4097, 4101, 4186
	TESTSETDEST DYNMEM // Define free memory as the destination
	NOWAIT ON // Do not wait until a position command is finished
	DEFORIGIN // Set position value to 0
	VEL 100 // Maximum velocity
	TESTSTART 0 // Start recording (until TESTSTOP or DYNMEM filled up)
	POSA 100000 // Start positioning
	DELAY 500 // Wait 500 ms
	VEL 20 // Set reduced velocity
	POSA 100000 // Apply new velocity but keep old target
	NOWAIT OFF // Wait until end of positioning
	DELAY 200 // Wait 200 ms
	TESTSTOP 0 0 // Stop recording

☐ TESTSETINDEX

Summary	Specify system variables for data recording.
Syntax	TESTSETINDEX svi1, svi2, svi3, ..., svin
Parameter	svi1...svin: indices of maximum 20 system variables (sysvar[...]) to be recorded.
	NB!: Maximum 20 system variables can be defined for simultaneous data recording. If more variables are listed as part of the TESTSETINDEX command, an error 195 “Too many indices for TESTSETINDEX” comes up at program runtime.
	NB!: Please notice, that the size of the required memory for data storage is mainly depending on the number of defined system variables multiplied by the number of samples.
	**NB!:** If more than 3 system variables are configured for data recording, it is a must that at least a APOSS-IDE for MCO 5.00 is in use for reading out and visualizing the data using the *Tools → Oscilloscope (TESTSETP).*
	NB!: If a recording was stopped (by TESTSTOP or reaching the defined number of samples) and another TESTSETINDEX and a following TESTSTART is executed, then the new data recording fills up the configured memory (array or DYNMEM) right from the beginning again. This means, that all old data recordings in the same memory area (array or DYNMEM) are overwritten and lost.
Description	TESTSETINDEX configures the system variables (sysvar[...]), which have to be captured during data recording. The TESTSTART command defines the number of test samples and triggers the start of data recording. The recorded data can be read out and visualized by *Tools → Oscilloscope (TESTSETP) afterwards.* The TESTSETINDEX command is a newer APOSS enhancement and offers a higher flexibility than the former (but still existing) TESTSETP command. Therefore it is recommended to use TESTSETINDEX instead of TESTSETP. The TESTSETINDEX command offers the possibility to define more than three system variables for data recording. A complete configuration of a data recording uses the commands TESTSETINDEX, TESTSETDEST, TESTSETTYPE, TESTSETTIME. *Tools → Oscilloscope (Single Shot)* can be used to set up a TESTSETINDEX configuration online without the need for TESTSET... commands and definitions within the APOSS program code. It is also possible to sample data in a 1 ms period. The difference and advantage of the implementation of TESTSET... commands directly in the program code is, that the recording starts at a defined line of code then. *Tools → Oscilloscope (Free Run)* can be used to record system and program variables online during a program run. The sample rate of the Free Run Oscilloscope is limited by some factors, like the type of communication interface in use, the number of data to be recorded during each sample or the firmware version of the controller. The minimum sample rate is 10 ms if you use the onboard RS485.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	SYSVAR, TESTSTART, TESTSETDEST, TESTSETTIME, TESTSETTYPE, TESTSTOP, TESTSETP Oscilloscope

Syntax Example

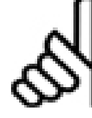
```
// Configure data recording of the following system variables:
// Slave actual pos., Slave command pos., Master position,
// Tracking error, Synchronization error, Slave velocity
TESTSETINDEX 4096, 4097, 4105, 4101, 4207, 4186
TESTSETTIME 20          // Record data every 20 ms
TESTSETDEST DYNMEM      // Use free memory for data recording
TESTSETTYPE 1           // Use cyclic recording (if memory is too small)
TESTSTART 0             // Start recording (until TESTSTOP)
SYNCP                   // Start position synchronization
DELAY 5000              // Run 5 seconds in synchronized mode
CVEL 10                 // Set 10% of maximum velocity
CSTART X(1)             // Run in continuous velocity mode
DELAY 2000              // Use velocity mode for 2 seconds
MOTOR STOP X(1)         // Stop motor
DELAY 100               // Wait 100 ms
TESTSTOP 0 0            // Stop recording
```

☐ TESTSETP

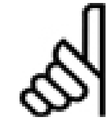
Summary	Specify recording data for test run
Syntax	TESTSETP ms vi1 vi2 vi3 *arrayname*
Parameter	ms = interval in milliseconds between two measurements vi 1–3 = indices of the three values to be recorded. The agreements for the system array apply. Three values are always recorded. array name = Name of the array used for the recording
Array Format	The values are stored as follows within the array (all values 4 Byte):

Designation	Content	Meaning
version	000	Version of the data structure
ms	1	Interval between two measurements in ms
no. of indices	3	Number of indices defined (always 3 for TESTSETP)
svi 1	i	Index of the 1. recorded system variable (SYSVAR)
svi 2	i	Index of the 2. recorded system variable (SYSVAR)
svi 3	i	Index of the 3. recorded system variable (SYSVAR)
no. of samples	nn	Number of recorded samples
first entry count	0 ...	Relative count of the first sample in the recorded data: In case of one time recording: 0 In case of cyclic recording: where to start reading
stop time	...	Internal system time, when recording was stopped
data	...	Recorded data, total no of values =
...	...	no. of samples * no. of variables (svi 1 ... svi n)
no. of test series	0 ...	Number of test series (if more than one is present) 0 or missing, if there are no more test series in the same array
more test series	...	(see above)

NB!:
Please make sure that the size of the array or available memory (DYNMEM) is sufficient for the recording. You need 10 elements for the header describing the data recording configuration, plus the number of test series (if the array holds more than one), plus 3 elements for each sample. Thus for 100 samples (i.e. TESTSTART 100) you need an array size of 311 elements or in case of DYNMEM usage, the free memory has to be 311 x 4 bytes= 1'244 bytes.



NB!:
Please notice, that firmware versions (< MCO 5.00) just had a header using 7 elements. This meant, that the array size just required an overhead of 7 elements (or 8, if more than one test series was included). For new firmware versions (>= MCO 5.00) there has to be at least an overhead of 11 elements taken into account for the calculation of the array size.



NB!:
If the keyword DYNMEM is used, and the data should be save in the free memory, than the version of the controller has to be at least MCO 5.00.

Description TESTSETP configures the parameters of a data recording, i.e. which data has to recorded, how often and in which memory section. The sample rate can be set down to a value of 1 ms. The TESTSTART command defines the number of test samples and triggers the start of the recording. At least the commands TESTSETP and TESTSTART have to be part of the APOSS application program for that. The recorded data can be read out and visualized by *Tools → Oscilloscope (TESTSETP)* afterwards.

The data can be recorded into a specified array or into the so-called DYNMEM memory section. The keyword DYNMEM stands for the free memory, which is not used by the program or variables and available for data recording. If the available DYNMEM size is not big enough to handle the recorded data, an error message is returned. It is recommended to use TESTSTART 0, if DYNMEM is in use.

In this case the maximum available DYNMEM is used without the risk to generate an overflow and get an error message. The usage of the DYNMEM requires a version starting with MCO 5.00.

The TESTSETINDEX command is a newer APOSS enhancement and offers a higher flexibility than the TESTSETP command. Therefore it is recommended to use TESTSETINDEX instead of TESTSETP. The TESTSETINDEX command offers the possibility to define more than three system variables and is available starting with MCO 5.00. A complete configuration can use the commands TESTSETINDEX, TESTSETDEST, TESTSETTYPE, TESTSETTIME. These commands can also be used in combination with the TESTSETP configuration. But it has to be kept in mind that these commands overwrite the corresponding configuration settings, which have been defined by a former TESTSETP. The recording is always based on the last defined settings.

Tools → Oscilloscope (Single Shot) can be used to set up a TESTSETINDEX configuration online without the need for TESTSET... commands and definitions within the APOSS program code. It is possible also to sample data in a 1 ms period. The difference and advantage of the implementation of TESTSET... commands directly in the program code is, that the recording starts at a defined line of code then.

Tools → Oscilloscope (Free Run) can be used to record system and program variables online during a program run. The sample rate of the Free Run Oscilloscope is limited by some factors, like the type of communication interface in use, the number of system variables to be recorded during each sample or the firmware version of the controller. The minimum sample rate is 10 ms if you use the onboard RS485.

Tools → Oszilloscope (Tune) can be used to trigger a test run which records actual and command position, speed, acceleration and current and whose result can be seen in the test run graphic. So, this is a kind of a predefined, menu driven data recording of a specified motor movement.

Portability Starting with MCO 5.00 the TESTSETP command has been replaced by four new commands with many more capabilities. The previous TESTSETP still works as before but it is recommended that the new commands be used.

Command Group SYS

Cross References TESTSTART, DIM, SYSVAR,
TESTSETINDEX, TESTSETDEST, TESTSETTIME, TESTSETTYPE,
Tools → Oscilloscope (TESTSETP)

Syntax Example

```
DIM tstrunarray[311]              // Array with 311 elements
// Configure data recording of slave pos., master pos, tracking error each 3ms
TESTSETP 3 0X1001 0X1009 0X1005 tstrunarray
TESTSTART 100                     // Start recording of 100 samples
POSR 10000                        // Start positioning
```

Syntax Example

```
DIM tstrunarray[311]                   // Array with 311 elements
TESTSETP 3 4096 4105 4101 tstrunarray    // same example, but decimal index
```

Syntax Example

```
TESTSETP 3 4096 4105 4101 DYNMEM  // same example, but using DYNMEM
```

☐ TESTSETTIME

Summary	Defines the sampling period for data recording.
Syntax	TESTSETTIME ms
Parameter	ms = Sampling period in milliseconds If this command is not used before a TESTSTART, then the default (1 ms) is used.
Description	This command defines the sampling period for data recording. (This was the first parameter of the previous TESTSETP.) According to the given period in milliseconds the values of the configured system variables are saved into the configured memory section (array or DYNMEM). The minimum sampling period is 1 ms. This means that each millisecond the data of all configured system variables is recorded. If a TESTSETP is executed after a TESTSETTIME command, then the period given by the TESTSETP command is valid. Always the last setting before the TESTSTART is the one, which is in use. If no TESTSETTIME and no TESTSETP is executed before the TESTSTART then the default value of 1 ms is in use.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	TESTSETP, TESTSETINDEX, TESTSETDEST, TESTSETTYPE, TESTSTART, TESTSTOP

Syntax Example	TESTSETDEST DYNMEM // Use DYNMEM for data recording
	TESTSETTYPE 0 // Configure one time recording
	TESTSETTIME 5 // Sample data just every 5 milliseconds
	// Configure system variables used for data recording:
	// Slave Pos., Tracking error, Input 1-8, Output 1-8
	TESTSETINDEX 4096, 4101, 0x01220202, 0x0122020A
	DEFORIGIN // Set position to 0
	TESTSTART 0 // Start recording (until TESTSTOP or DYNMEM is filled up)
	VEL 10 // 10% of maximum velocity
	ACC 10 // 10% of maximum acceleration
	DEC 50 // 50% of maximum deceleration
	POSA 100000 // Start Positioning (and wait until finished)
	DELAY 200 // Wait 200 ms
	TESTSTOP 0 0 // Stop recording

□ TESTSETTYPE

Summary	Defines the type of data recording, i.e. one time recording or cyclic recording.
Syntax	TESTSETTYPE type
Parameter	Type = 0 = one time recording (= default setting) 1 = cyclic recording If no type is set before a TESTSTART is executed, then the default (one time recording) is used. **NB!:** A cyclic recording even proceeds when a program is finished or stopped by an error. The recording just stops, if the command TESTSTOP is executed or an user break (= pressing [Esc]) is taking place. **NB!:** A cyclic recording always overwrites the given memory as soon as the memory limit or the defined number of samples is reached. The actually stored number of samples depends thereby on the size of the memory (TESTSTART 0) or the defined number of samples (TESTSTART value > 0).
Description	This command specifies, if a configured data recording is executed just once or runs endless in a cyclic mode in the background until a TESTSTOP is executed. If cyclic recording is activated and TESTSTART 0 is defined, all the available memory is used and overwritten again and again. If a defined number of samples is specified (i.e. TESTSTART value > 0) , just the corresponding memory section is in use and overwritten again and again. A one time recording (TESTSETTYPE 0) stops, when … … the defined number of samples is recorded (TESTSTART value > 0) … the specified memory (array or DYNMEM) is filled up (TESTSTART 0) … the command TESTSTOP is executed … the program execution is stopped by the user by pressing the [Esc] key A cyclic recording stops (TESTSETTYPE 1), when … … the recording is stopped by the command TESTSTOP … the program execution is stopped by the user by pressing the [Esc] key The cyclic recording can be used very well for debugging purposes. Especially in case of machine misbehavior, which does not take place very often. The usage of cyclic recording, DYNMEM and TESTSTART 0 gives the power to track a maximum number of information for an endless period of time. When an error happens, a defined event takes place or an external interrupt comes up, the recording can be stopped by the TESTSTOP command. The information recorded before, can then be read out, evaluated and saved to disk using *Tools → Oscilloscope (TESTSETP)*.
Portability	Command is available starting with MCO 5.00.
Command Group	SYS
Cross Index	TESTSETP, TESTSETINDEX, TESTSETDEST, TESTSETTIME, TESTSTART, TESTSTOP

Syntax Example

```
SYNCP                      // Synchronization of the position
// Configure system variables used for data recording:
// Slave pos., Master pos., Tracking error, Sync. error,
// Slave velocity, Scaled difference of master and slave velocity
TESTSETINDEX 4096, 4105, 4101, 4207, 4185, 4236
TESTSETTYPE 1              // Configure cyclic recording
WAITI 1 ON                 // Wait until input 1 is present
TESTSTART 0                // Start recording (until TESTSTOP)
WAITI 1 OFF                // Wait until input 1 signal is removed
TESTSTOP 0 0               // Stop recording
```

Syntax Example

```
// Define an error handler
ON ERROR GOSUB errhandler
// Define all necessary data recording settings:
TESTSETDEST DYNMEM         // Use DYNMEM for data recording
TESTSETTYPE 1              // Configure cyclic recording
// Configure system variables used for data recording:
// Slave Pos., Tracking error, Input 1-8, Output 1-8
TESTSETINDEX 4096, 4101, 0x01220202, 0x0122020A
TESTSTART 0                // Start recording (until TESTSTOP)

// Program main loop
endless:
    // Execute the application code
GOTO endless

// Subprogram section
SUBMAINPROG
SUBPROG errhandler         // Error handler
    TESTSTOP 0 0           // Stop recording
    EXIT                   // Quit program
RETURN                     // End of error handler
ENDPROG                    // End of subprogram section
```

☐ TESTSTART

Summary Start the recording of a test run.

Syntax TESTSTART n

Parameter

n = 0 The maximum number of samples is recorded, which fits into the defined memory section (array or DYNMEM).

n > 0 number of samples to be recorded

If the given memory section (array or DYNMEM) does not have sufficient space for the defined number of samples (parameter no > 0), the error 171 "Array too small" or error 194 "DYNMEM too small" is triggered during program execution. Therefore it is recommended to use TESTSTART 0, which fills up the given memory in maximum.

If a recording was stopped (by TESTSTOP, or [Esc] is pressed, or in case of a single shot recording, or reaching the defined number of samples) and another TESTSETINDEX and a following TESTSTART is executed, then the new data recording fills up the configured memory (array or DYNMEM) right from the beginning again. This means, that all old data recordings in the same memory area (array or DYNMEM) are overwritten and lost.

If a next TESTSTART is executed without a TESTSETINDEX definition in between, then the new data is appended to the last data recording. The configured memory (array or DYNMEM) holds more than one recording then.

Description This command is used to start data recording according to the configuration made by the last TESTSETP, TESTSETINDEX, TESTSETDEST, TESTSETTIME, TESTSETTYPE commands. The recorded data can be read out and visualized afterwards using *Tools → Oscilloscope (TESTSETP)*.

If not all parameters are completely specified before the TESTSTART command is executed, the following default settings are in use for data recording:

- Default sampling period: 1 ms
- Default type of recording: one time
- Default memory section for data storage: DYNMEM
- Default system variables for recording: 4096 (ACTPOS), 4097 (COMPOS), 4324 (ACTCURR)

It is also possible to gather more than one recording in one array. For example, it is possible to start a recording of 1000 samples, then stop the recording and then start another 500 samples. If then the data are read out with APOSS, two recordings will be seen, that you can look at. If a restart from scratch is desired, then a new TESTSETINDEX command must be executed.

Portability Extension concerning the Oscilloscoope functions are available starting with MCO 5.00.

Command Group SYS

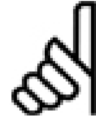
Cross References *Tools → Oscilloscope (TESTSETP)*
TESTSETP, TESTSETINDEX, TESTSETDEST, TESTSETTIME, TESTSETTYPE, TESTSTOP

Syntax Example

```
SYNCP                    // Synchronization of the position
// Configure system variables used for data recording:
// Slave pos., Master pos., Tracking error, Sync. error,
TESTSETINDEX 4096, 4105, 4101, 4207
WAITI 1 ON               // Wait until input 1 detects a signal
TESTSTART 200            // Start recording (200 measurements)
```

Syntax Example	NOWAIT ON // Do not wait until the position is reached
	DEFORIGIN // Do not wait until the position is reached
	// Configure system variables for recording:
	// Actual pos., Command pos., Tracking Error, Velocity, Indexpos., Inp. 1-8
	TESTSETINDEX 4096, 4097, 4101, 4186, 4098, 0x01220202
	TESTSTART 0 // Start recording (until TESTSTOP or DYNMEM is filled up)
	VEL 50 // 50% of maximum velocity
	POSA 100000 // Start positioning
	WAITP 20000 // Wait until position 20000 is reached
	VEL 100 // Increase velocity to 100%
	POSA 100000 // Use new velocity for the same target
	NOWAIT OFF // Wait until positioning is finished
	DELAY 200 // Wait 200ms
	TESTSTOP 0 0 // Stop recording

□ TESTSTOP

Summary	Stops the data recording.
Syntax	TESTSTOP method param
Parameter	method — method, how stop should take place. 0 = stops immediately No more methods are implemented up to now param — parameter of the stopping method. Not in use for the implemented stopping methods up-to-now
	**NB!:** Due to the fact, that just method 0 is implemented up to now, the only valid command and parameter combination look like this: TESTSTOP 0 0
Description	This command stops the data recording in the configured way.
	NB!: If a recording was stopped by TESTSTOP and another TESTSETINDEX and a following TESTSTART is executed, then the new data recording fills up the configured memory (array or DYNMEM) right from the beginning again. This means, that all old data recordings in the same memory area (array or DYNMEM) are overwritten and lost.
	NB!: For applications, which use cyclic recording, it is recommended to insert a TESTSTOP command at the end of the program or if the program quits in case of an error. Otherwise the recording still runs on even if the program is not executing anymore.
Portability	Command is available starting with MCO 5.00. If values with a newer APOSS version are read out, the recording is displayed correctly (time wise). Since MCO 5.00, sampling is also stopped when [Esc] is pressed in APOSS.
Command Group	SYS
Cross Index	TESTSETP, TESTSETINDEX, TESTSETDEST, TESTSETTIME, TESTSETTYPE, TESTSTART

Syntax Example	// Define an error handler
	ON ERROR GOSUB errhandler
	// Configure system variables used for data recording:
	// Slave pos., Master pos., Tracking error, Sync. error,
	// Slave velocity, Scaled difference of master and slave velocity
	TESTSETINDEX 4096, 4105, 4101, 4207, 4185, 4236
	TESTSETTYPE 1 // Configure cyclic recording

```
WAITI 1 ON              // Wait until input 1 is present
SYNCP                   // Synchronization of the position
TESTSTART 0             // Start recording (until TESTSTOP)
WAITI 1 OFF             // Wait until input 1 signal is removed
MOTOR STOP              // Stop motor
TESTSTOP 0 0            // Stop recording

// Subprogram section
SUBMAINPROG
SUBPROG errhandler      // Error handler
    TESTSTOP 0 0        // Make sure, that recording is stopped
    EXIT                // Quit program
RETURN                  // End of error handler
ENDPROG                 // End of subprogram section
```

Syntax Example

```
NOWAIT ON               // Do not wait until the position is reached
DEFORIGIN               // Set current position to 0
// Configure system variables for recording:
// Actual pos., Command pos., Tracking Error, Velocity, Indexpos., Inp. 1-8
TESTSETINDEX 4096, 4097, 4101, 4186, 4098, 0x01220202
TESTSTART 0             // Start recording (until TESTSTOP or DYNMEM is filled up)
VEL 20                  // Use 20% of maximum velocity
POSA 100000             // Start positioning with velocity 50%
WAITP 50000             // Wait until position 50000 is reached
VEL 100                 // Increase velocity to 100%
POSA 100000             // Start positioning with new velocity
NOWAIT OFF              // Wait until positioning is finished
DELAY 200               // Wait 200ms
TESTSTOP 0 0            // Stop recording
```

Sample

This example gathers data continuously until a special condition is reached and recording is stopped. If you then read out the data, you will get the last sampled data before the stop occurred. The amount of data is dependent on the available dynamic memory.

```
#define PFG_ACTPOS        0x01250001
#define PFG_COMPOS        0x01250002
#define PFG_VCMDSIGNED    0x012500B5
...
TESTSETINDEX PFG_ACTPOS, PFG_COMPOS, PFG_VCMDSIGNED
TESTSETDEST DYNMEN      // select dynamic memory for recording
TESTSETTIME 5           // select a sampling period of 5 ms
TESTSETTYPE 1           // select cyclic recording
TESTSTART 0      // start recording and use the whole available dynamic memory
...
IF(whatever) THEN
    TESTSTOP 0 0        // stop recording
ENDIF
```

This way of recording could also be used without changing your program at all by using the oscilloscope feature of APOSS. (See Single Shot Oscilloscope).

TIME

Summary	Reads system-time
Syntax	res = TIME
Return Value	res = system-time in milliseconds after switching on
	NB!: Please note that after counting up to MLONG the value will change to –MLONG.
Description	The internal system-time can be read out using the TIME command. The TIME command is most suitable for calculating the execution time of a command sequence or device cycle time.
Command Group	SYS
Syntax Example	PRINT TIME /* print current system-time */ timestop1 = TIME /* store current system-time */
Program Sample	ACC_01.M, DELAY_01.M, EXIT_01.M, GOSUB_01.M

TRACKERR

Summary	Queries actual position error of the axis
Syntax	res = TRACKERR
Return Value	res = actual trailing of the axis in UU
Description	Queries the difference between the CPOS and APOS, i.e., it tracks the error occurring. It is to be noted that the actual position need not be the same as the commanded position and they are not automatically compensated.

Position
175HA520.10
600
500
400
300
200
100
0
CPOS in UU
APOS in UU
0 10 20 30 40 50 60 70 80 Time

Command Group	SYS
Cross Index	APOS, CPOS, par. 32-67 *Max. Tolerated Position Error*
Syntax Example	PRINT TRACKERR /* query actual position error of the axis */
Sample	see below

```
POSA 500
WHILE(1) DO
{
PRINT "Command Position", CPOS
PRINT "Actual Position", APOS
PRINT "Error", TRACKERR
WAITT 10
}
```

	ENDWHILE Output: Command Position 100 Actual Position 98 Error 2 Command Position 200 Actual Position 199 Error 1 Command Position 300 Actual Position 297 Error 3 ... Command Position 500 Actual Position 500 Error 0 ... and so on The gray shaded region between the CPOS and the APOS at the time interval can be thus readout using TRACKERR. To read all the error throughout the positioning, TRACKERR should be in a loop to actually track to the error.

◻ USRSTAT

Summary	Sets the user status (long) which can be queried via the CAN bus.
Syntax	USRSTAT val
Parameter	val = value to be set
Description	Sets the user status (long) which can be queried via the CAN bus.
Portability	Command is available starting with MCO 5.00.
Command Group	CAN
Syntax Example	USRSTAT 5 /* sets user status to 5 */

☐ VEL

Summary	Set velocity for relative and absolute motion.
Syntax	VEL v $$\text{Command Velocity [RPM]} = V * \frac{\text{par. 32 - 80 Maximum Velocity}}{\text{par. 32 – 83 Velocity Resolution}}$$
Parameter	v = scaled velocity value
Description	The velocity for the next absolute and relative positioning procedure is determined with the VEL The velocity for the next absolute and relative positioning procedures and the maximum allowed velocity for synchronizing procedures are determined with the VEL command. The value remains valid until a new velocity is set via another VEL command. The new velocity value will be set in reference to the parameters 32-80 *Maximum Velocity* and 32-83 *Velocity Resolution*. If the velocity value equals the *Velocity Resolution*, then the RPM value set in *Maximum Velocity* will be used. NOTE: Slave velocity in synchronizing mode is also limited by the VEL command. **NB!:** If no velocity has been set prior to a positioning or synchronizing command, then the value of par. 32-84 *Default Velocity* will be used. If the velocity needs to be altered during positioning, it is possible in the NOWAIT ON mode, when the VEL command is followed by another POSA command targeting to the desired position. The maximum speed allowed can be changed at any time with the command VEL, if a SYNCV, SYNCP, or SYNCM follows the VEL command.
Command Group	REL, ABS
Cross Index	ACC, POSA, POSR, NOWAIT Parameter: 32-80 *Maximum Velocity*
Syntax Example	VEL 100 /* Velocity 100 */
Program Sample	VEL_01.M

☐ VLTALARMSTAT

Summary	Returns if an alarm is active or not.
Syntax	VLTALARMSTAT
Description	The command VLTALARMSTAT returns if an alarm is active or not. There are two possible values, 1<<3 or 1<<6 depending if the alarm can be reset or not. bit 3 = Alarm(s) present bit 6 = Trip Lock Alarm(s) present
Command Group	CON
Cross Index	VLTERRCLR

Syntax Example	If (VLTALARMSTAT) then
	Print " alarm active ", VLTALARMSTAT
	VLTERRCLR
	Endif

☐ VLTCONTROL

Summary	Sets the VLT control word in MOTOR OFF state
Syntax	VLTCONTROL control word value
Parameter	value
Description	VLTCONTROL allows setting the VLT control word while the system is in MOTOR OFF state. This command can be used to set the control word to every value. The user is responsible for the correct values (specially the bit DATA VALID). The commands behave as follows. When using VLTCONTROL the first time, it will be changed into User-Control mode. In this mode the control word is not influenced at all. The user is responsible for the control word. As long as the system is in that mode, OUTAN only influences reference, but not control word. It can be only returned to normal mode by using a MOTOR ON command, or by starting a new APOSS program.
Command Group	CON
Cross Index	MOTOR OFF, MOTOR ON, OUTAN
Syntax Example	MOTOR OFF ... VLTCONTROL 0x047C // activates drive OUTAN 0x1000 ... MOTOR ON // disables user control of control word

☐ VLTERRCLR

Summary	Clears a VLT-alarm
Syntax	VLTERRCLR
Description	The command VLTERRCLR clears a VLT-alarm without doing anything with an existing option error. This command can be used everywhere in an APOSS program.
Command Group	CON
Cross Index	VLTALARMSTAT

Syntax Example	If (VLTALARMSTAT) then
	Print " alarm active ", VLTALARMSTAT
	VLTERRCLR
	Endif

☐ WAITAX

Summary	Wait till target position is achieved
Syntax	WAITAX
Description	The WAITAX command has been designated for use with an active NOWAIT mode. By use of this command in NOWAIT ON condition, it is possible to wait for further program processing after a positioning command, until the axis has achieved its set position.
Command Group	CON
Cross Index	NOWAIT ON/OFF, POSA, POSR, AXEND, STAT, WAITI
Syntax Example	WAITAX /* Wait till the axis has ended motion */ WAIT AX /* Alternative form */
Program Sample	WAIT_01.M, VEL_01.M

WAITI

Summary	Wait for defined input condition
Syntax	WAITI n s
Parameter	n = input number: 1 – 8 or 16 – 33 s = expected condition: ON = High-Signal, OFF = Low-Signal
Description	The WAITI command waits before continuing the processing until the specified input has got the desired signal level. **NB!:** If the expected input condition does not occur, then the program will remain 'stuck' at this point. A minimal signal length is required for the sure identification of a signal condition! Please see the FC 300 Operation Instructions and FC 300 Design Guide for information about the circuit and technical data for the inputs.
Command Group	CON
Cross Index	ON INT .. GOSUB, DELAY, WAITT, WAITAX
Syntax Example	WAITI 4 ON /* Wait till high level reached input 4 */ WAITI 4 1 /* 3 alternative forms */ WAIT I 4 ON WAIT I 4 1
Syntax Example	WAITI 6 OFF /* Wait till Low level reached input 6 */ WAITI 6 0 /* 3 alternative forms */ WAIT I 6 OFF WAIT I 6 0
Program Sample	WAIT_01.M

WAITNDX

Summary	waits until the next index position is reached
Syntax	WAITNDX t
Parameter	t = time-out in ms
Description	Waits for the index while checking time-out. The program waits until either the index of the axis is found or the time (time-out) is exceeded. **NB!:** If the time is exceeded then an error is triggered which can be evaluated with a ON ERROR function. **NB!:** The command WAITNDX can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type).*
Command Group	CON
Cross Index	WAITI, WAITP, INDEX
Syntax Example	CVEL 1 CSTART WAITNDX 10000 /* Waits a maximum of 10 s for the axis to reach the index position */ OUT 1 1

WAITP

Summary	waits until a certain position is reached
Syntax	WAITP p
Parameter	p = absolute position being waited for
Description	The WAITP command causes the program to wait until position p is reached. If, from the speed and the current position, it follows that the point p has already been exceeded then the command is also terminated. **NB!:** Active ON INT or ON PERIOD commands can affect the precision and reproducibility.
Command Group	CON
Cross Index	DELAY, WAITI, WAITAX
Syntax Example	NOWAIT ON POSA 10000 WAITP 5000 /* wait for position 5000 */ OUT 1 1 /* set output 1 */ NOWAIT OFF

Output 1 5000 10000 Position

e75MD052

WAITT

Summary	Time delay
Syntax	WAITT t
Parameter	t = delay time in milliseconds (maximum MLONG)
Description	The WAITT command is suitable for achieval of a defined program time delay. The inputted parameter shows the delay time in milliseconds. **NB!:** If an interrupt occurs during the delay time, then the entire delay procedure will be re-begun following the processing of the interrupt. The DELAY command is preferable to the WAITT command, because of its constant time behavior.
Command Group	CON
Cross Index	DELAY, WAITI, WAITAX
Syntax Example	WAITT 5000 /* wait 5 seconds */ WAIT T 5000 /* alternative form */
Program Sample	WAIT_01.M

☐ WHILE .. DO .. ENDWHILE

Summary	Conditional loop with start criteria. (While condition is fulfilled, repeat ...)
Syntax	WHILE (condition) DO ENDWHILE
Parameter	condition = abort criteria
Description	By using the WHILE .. ENDWHILE construction you can repeat the enclosed program area one or more times, dependent on any criteria. The loop critera are made up of one or more comparison operations, and are always monitored at the start of a loop. When a negative result already appears at the first monitoring, this can cause an omission of the commands within the loop, and the program will continue after the ENDWHILE instruction.
	NB!: Depending on the loop criteria, it can happen that the contents of the loop will never be processed. To avoid an endless loop, the processed commands within the loop must have a direct or indirect influence on the result of the abort check.
Command Group	CON
Cross Index	LOOP, REPEAT .. UNTIL

Syntax Example	WHILE (A != 1 AND B == 0) DO
	    command line 1
	    command line n
	ENDWHILE

Program Sample	WHILE_01.M, INKEY_01.M

☐ _GETVEL

Summary	Changes sample time for AVEL and MAVEL
Syntax	var = _GETVEL t The values are displayed in UU/s for AVEL or qc/s for MAVEL.
Parameter	t = sample time in ms
Description	With the _GETVEL command it is possible to change the sampling time for AVEL and MAVEL. AVEL and MAVEL usually work with a sampling time of 20 ms. With this sampling time the resolution is better. However, a new measurement is only sampled every 20 ms. The command _GETVEL lasts exactly as long as the assigned value, e.g. _GETVEL 200 ca. 200 ms.
Command Group	SYS
Cross Index	AVEL, MAVEL
Syntax Example	var = _GETVEL 200 Thus, the measurement resolution is considerably better; however changes are only seen after a delay of 200 ms.

□ #INCLUDE

Summary	Inclusion of file contents in the indicated program position
Description	This #INCLUDE instruction has been replaced by the #include instruction of the preprocessor.
Portability	The syntax of the #include statement for versions of the APOSS IDE prior to version MCO 5.00 is different. Prior to version MCO 5.00, the statement contained no quotes. If an “old” program is opened using the updated APOSS IDE, then quotes will be automatically added.
Command Group	Preprocessor
Program Sample	INCL_01.M + INCSTA01.M + INCPOS01.M + INCIN01.M

MG.34.R1.02 – VLT® is a registered Danfoss trademark

Appendix

Hz
V
A
IP
°C
Ω

What's New in the Update Version starting with MCO 5.00?

Significant new functionality has been added to the APOSS IDE and much of the existing functionality has been greatly enhanced. The MCO CAN Interface is supported. The major areas of change are described below.

APOSS Tools

Oscilloscope Tool

A new *Oscilloscope* tool has been added to APOSS. This is a graphical display tool that provides powerful and comprehensive functions to help optimize controller parameters and debug executing programs. It allows the user to "watch" any internal controller parameter, variable, or state while the controller is executing. The Oscilloscope tool replaces the "Testrun" functionality which is now obsolete and has been removed from APOSS.

Array Editor Tool

A new *Array Editor* tool has been added to APOSS. The Array Editor allows the user to conveniently view and update all parameters and arrays in the controller using a list format. This list can be customized by the user so that it can be used to configure user applications. The Array Editor can be used either from within APOSS or as a stand-alone application.

Enhanced CAM Editor

The *CAM Editor* tool has been enhanced as follows:

- The default file extension for configuration files has been changed from ".cnf" to ".zbc". However, ".cnf" files are still accepted. This change was done to avoid conflicts with the Microsoft-reserved file extension of ".cnf". Using ".zbc" allows APOSS to properly define a file type icon for Windows Explorer displays and allows configuration files to be opened automatically in the standard way by double-clicking on them.
- Basic window handling has been improved so that CAM Editor windows behave like normal windows. For example, menus and toolbars work in the standard way and CAM Editor windows will "go to the background" when other APOSS windows are selected. Configuration files are opened and saved using the standard File menu items and toolbar buttons.
- The color of lines and points has been changed so that their types can be more easily identified. Tangent lines and points are displayed in green while curve lines and points are display in red.

- A new curve type (GRAD - 3) has been added. This allows the user to specify the start and end gradients of a curve.
- Two new segment types have been added: Trapezoid and 3rd order segments. These segment types can be used to connect adjacent tangent segments.

Watch Window

The “Watch Window” has been enhanced as follows:

- The Watch Window is no longer implemented as a separate window (“Show Watch”). It is now embedded directly in the APOSS Window as a splitter window and is available at all times. It can be controlled using the menu and toolbar buttons in the standard way.
- Watched variables are maintained in the user's program file and will be re-added to the Watch Window when a program is opened.
- ”double” variables and 2-dimensional arrays are now supported.
- Several limitations of the old Watch Window (e.g. a limit of 10 variables) have been removed.

□ Enhanced APOSS Compiler

The APOSS compiler has been enhanced as follows:

- Compilations are now optimized for the specific controller for which they are compiled. This results in a reduced command execution time and faster program cycle execution.
- The stack size can be defined in the *Settings → Compiler* dialog.
- SWITCH, CASE, BREAK constructs are supported.
- Two dimensional arrays are supported.
- Array copy is supported.
- Variables can be declared as “double” and “constant”.
- Post-increment (++) and post-decrement (--) operators are supported.
- Function declaration and function call is supported.
- Functions can have parameter lists and return a value.
- Local variables are supported
- Casting (long <-> double) of variable types is supported.
- ROUND and FABS commands are enhanced for variables of type “double”.
- New floating point commands for variables of type “double” are available:
  sqrt(), sin(), grad(), rad(), ln(), exp, pi

□ APOSS User Interface Changes

Among the more significant changes to the APOSS user interface, are the following:

- All the various tools (i.e. “CAM Editor”, “Oscilloscope”, etc.) have been moved under a new *Tools* menu. These can also be accessed using toolbar buttons.
- All sample programs are now built into APOSS. They can be opened using the File → Sample menu command.
- The *Edit → Find in Files* function will search for a string in files on disk.
- *Next Bookmark, Previous Bookmark,* and *Toggle Bookmark* commands have been added to the Edit menu. This will allow the bookmark functionality within the editor to be used more easily.
- Single programs can now be deleted on the controller (using the *Controller → Programs* menu item).
- “Teach in” has been removed from the *Command List* [F12]. It is no longer supported.

☐ New Commands

The following new commands are supported in the APOSS programming language:

Command	Content
APOSDIFF	Overflow handling of incremental encoders in applications.
CANDEL	Deletes all or single CAN objects.
CANIN	Reads an object via the CAN bus.
CANINI	Initializes the necessary objects (PDOs) for data exchange of CANopen nodes, or enables extended CANINI, CANIN function.
CANOUT	Sends message with an internal number.
CPOSDIFF	Overflow handling of incremental encoders in applications.
DEFCANIN	Defines a receive object.
DEFCANOUT	Defines a transmit object in the CAN controller.
INGLB	Reads a global CAN message via CAN bus.
INMSG	Reads CAN message from the buffer.
IPOSDIFF	Overflow handling of incremental encoders in applications.
JERKFINVEL	Calculates the final velocity for a jerk-limited stop with maximum acceleration/deceleration.
JERKSTOPDIST	Calculates the necessary distance for a jerk-limited stop with max deceleration.
LINKPDO	Link the system variable with RxPDO and copy in the internal parameters, or link part of an array into the PDO.
LINKSDO	Link TxPDO with internal system variable, or link part of an array out of the PDO.
MAPOSDIFF	Overflow handling of incremental encoders in applications.
MIPOSDIFF	Overflow handling of incremental encoders in applications.
MSGVAL	Contains the second part of the last read CAN message.
ON CANINPUT	Calls up a subprogram when a CAN telegram type 'id' arrives.
ON CANMSG GOSUB	Calls up subroutine.
ON DELETE..SETOUT	Deletes all interrupts which set or reset an output.
ON KEYPRESSED GOSUB	Call up a subprogram when a LCP key is pressed or released.
ON posint .. GOSUB	Call up a subprogram when a position interrupt occurs.
ON posint .. SETOUT (TOIN)	Simulate a cam box (all types of POSINTs).
OUTMSG	Sends a CAN message.
PDO	Pseudo array for direct access to the CANopen PDOs.
SDOREAD	Reads SDO of a connected CANopen device.
SDOREADSEG	Segmented read of SDOs (unpacked).
SDOREADSEGP	Segmented read of SDOs (packed).
SDOSTATE	Checks the result of an active communication.
SDOWRITE	Sets SDO of a connected CAN-open device.
SYNCMARKERSTART	Resets a marker or resets marker handling.
SWAPMENC	This command is not supported anymore. The function is realized with SET ENCODERTYPE and SET MENCODERTYPE, see par. 32-00 and 32-30.
TESTSETDEST	Defines the memory section for saving recorded data

Command	Content
TESTSETINDEX	Specifies system variables for data recording
TESTSETTIME	Defines the sampling period for data recording
TESTSETTYPE	Defines “one time” or “cyclic” recording

☐ New and Extended Parameters

The following new axis parameters are supported:

Parameter		Content
32-13 Enc.2 Control	ENCCONTROL	Configuration of position evaluation after a change of encoder source.
32-14 Enc.2 node ID		Encoder node ID
32-15 Enc.2 Guard		Encoder CAN Guard
32-43 Enc.1 Control	MENCCONTRO L	Configuration of master position evaluation after a change of encoder source.
32-44 Enc.1 node ID		Encoder node ID
32-45 Enc.1 Guard		Encoder CAN Guard
32-73 Integral limit filter time	KILIMTIME	Time (ms) which is used to increase or decrease the integral limit of the position control loop up to KILIM.
32-74 Position error filter time	POSERRTIME	Time frame [ms] for triggering position error state.
32-86 Acc. up for limited jerk	JERKMIN	Minimum time required before reaching the maximum acceleration.
32-87 Acc. down for limited jerk	JERKMIN2	Acceleration ramp-down constant.
32-88 Dec. up for limited jerk	JERKMIN3	Deceleration ramp-up constant.
32-89 Dec. down for limited jerk	JERKMIN4	Deceleration ramp-down constant.
33-32 Feed Forward Speed Adaptation	SYNCFFVEL	Velocity feed forward [per mill of VCMD] for synchronization modes.
33-33 Velocity Filter Window	SYNCVFLIMIT	Sync error window [qc] for automatic deactivation of SYNCVFTIME.
33-90 X62 MCO CAN node ID	CANNR	CAN node ID
33-91 X62 MCO CAN baud rate	CANBAUD	CAN Baud rate
33-94 X60 MCO RS485 Serial termination		RSTERMINATION
33-95 X60 MCO RS485 serial baud rate		RSBAUDRATE

Along with the new axis parameters, the description of the following parameters has been amended:

32-01 *Incremental Resolution ENCODER*, 32-60 P*roportional Factor KPROP*, 32-61 *Derivative Value for PID Control* KDER, 32-62 *Integral Factor* KINT, 32-69 *Sampling Time for PID control* TIMER und 32-63 *Limit Value for Integral Sum* KILIM.

Technical Reference

This section documents data structures and compiler details which are only required in exceptional cases by the user. For example, if an automatically generated programming is to be modified like a curve profile.

Array Structure of CAM Profiles

Header

The header contains general information like

- Identification for curve array
- Version number for curve structure
- Type of curve
- Name of curve
- Index to curve information section
- Index to start/stop point section
- Index to fixed point section
- Index to interpolation point section
- Index to start/stop point indices (in interpolation section)
- Index to start/stop velocities (times 100000)
- Index to startpath interpolation points
- Index to stoppath interpolation points

Curve Information Section

This section of the array contains all information about the type of curve like

- Length of curve (master)
- Length of curve (slave)
- Number of fix points
- Number of Interpolation points (this gives the resolution)
- Type of interpolation
- Slave stop point, point where slave is positioned, when synchronization is stopped
- Correction start point (only valid for marker synchronization)
- Correction end point (only valid for marker synchronization)
- Maximum correction which is allowed (only valid for marker synchronization)
- Maximum start/stop path length (Size of start/stop path area) (min. 2)
- No of start/stop point pairs
- Maximum number of cycles per minute (Application information)

Curve Start/Stop Point Section

This section contains the start/stop points. Because the use of this point is up to the user, we just speak of a path, which can be a start or a stop sequence. Every path consists of 2 points. If we are moving forward, the path starts (start or stop) with the a-point and ends with the b-point. If we are moving backward, the path starts with the b-point and ends with the a-point. So the user is able to tell us in the program, which pair of points to use for starting or stopping, when he uses a STARTCURVE or STOPCURVE command.

- Path 1 (a – point)
- Path 1 (b – point)
- Path 2 (a – point)
- Path 2 (b – point), ...

These points have to lie on interpolation points, so possibly the PC software has to adjust them according to the interpolation resolution. This should not be a real restriction, because the interpolation points are normally very dense. So for example if we have rotating master which makes one revolution per cycle and we choose a cycle length of 3600 MU (1 MU = 1/10 degree). Let us further assume, that we choose the number of interpolation points as 1200, than you have a resolution of 3 MU = 3/10 degree for defining your start and stop points.

Fixed Point Section

This section contains the fix points, which were the basis for the interpolation calculation. These points always consist of the following triple

– Master coordinate
– Slave coordinate
– Type of point (tangent, curve)

These points are defined by the user in MU units (see internal description). If you want to avoid, that the real interpolation curve misses your fix points, you have to choose them in such a manner that they lay on an interpolation point (see above). This can be forced through a snap function within the PC software.

Interpolation Point Section

This section contains a list of slave coordinates. They belong to master coordinates which are of equal distance, given by the interpolation resolution.

Indices of Start/Stop Points

Here we have the indices of the start/stop points (see above) within the interpolation array. These are necessary for the ease of start and stop recognition. We are waiting until start index for example equals the actual index and direction of movement is correct. If both is true, we start synchronization. The same is true for stopping.

Start Stop Velocities

To be able to calculate an appropriate starting or stopping path, we need the velocity we have to reach at end (start) or we will have at the beginning (stop) in UU/MU units (Slave units per Master units).

Start / Stop Paths

This is the place for the interpolation points of the actual start and stop path. These points are calculated when a SYNCCSTART or SYNCCSTOP command is executed, but we have to reserve the room right now.

◻ CAM Array Definition

Index	Name	Unit	Value	Description
General				
1	Identification	(dec)	999.000.001	Number to identify array
2	VersionNumber	(dec)	100	Version as decimal (1.00 = 100)
3	CurveType	(dec)	0	0 = symmetrical; 1 = compatible
4	CurveName 1	(4char)	Nona	Name of curve total 16 char.
5	CurveName 2	(4char)	meCu	default is:
6	CurveName 3	(4char)	rve0	NonameCurve00001
7	CurveName 4	(4char)	0001	
8	IndexCIF	(dec)	16	Index to Curve Information Part
9	IndexSTP	(dec)	27	Index to Start/Stop point Part
10	IndexFIP	(dec)	IndexSTP +	Index to Fix point Part

MG.34.R1.02 – VLT® is a registered Danfoss trademark

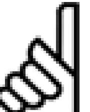
Index	Name	Unit	Value	Description
			STPno*2	
11	IndexINP	(dec)	IndexFIP + FixPointNo * 3	Index to Interpolation Point Part
12	IndexSTPInd	(dec)	IndexINP + InterpolPointNo	Index to StartStop Interpolation Indices
13	IndexSTPVel	(dec)	IndexSTPInd +STPno*2	Index to StartStop Velocities
14	IndexSTIP	(dec)	IndexSTPVel +STPno*2	Index to Startpath interpolation points
15	IndexSTPIP	(dec)	IndexSTIP + MaxStartStopLen	Index to Stoppath interpolation points
Curve Information				
1	MasterCycleLen	MU	-	Length of Curve in CurveMaster units
2	SlaveCycleLen	UU	-	Slave max. travel distance in CurveSlave units
3	FixPointNo	(dec)	4	Number of fix points (minimum 4)
4	InterpolPointNo	(dec)	-	Number of interpolation points (including first and last, which correspond to the same location)
5	InterpolType	(dec)	0	0 = cubic spline, 1 = periodic cubic spline
6	SlaveStopPosition	UU	0	Position, where slave stands after stopping
7	CorrectionStartPoint	MU	0	Position, where Correction may start
8	CorrectionStopPoint	MU	MasterCycleLen	Position, where Correction has to be finished
9	MaximumCorrection	UU	-	Maximum Correction which is allowed in one cycle
10	MaxStartStopLen	(dec)	0	Maximum length of start/stop path (no of int. points)
11	StartStopNo	(dec)	0	Number of start stop point pairs (n) (see below)
12	MMaxCycles	(dec)	0	Max. number of cycles per minute (application info)
13	MMarkerPos	CM	0	Master Marker Position in curve
14	SMarkerPos	CS	0	Slave Marker Position in curve
Start/Stop Point				
1	STPoint_1.a	MU	0	Start (forward) / Stop (backward) point no. 1
2	STPoint_1.b	MU	0	Stop (forward) / Start (backward) point no. 1
3	STPoint_2.a	MU	0	Start (forward) / Stop (backward) point no. 2
4	STPoint_2.b	MU	0	Stop (forward) / Start (backward) point no. 2
5	...	MU	0	
6	...	MU	0	
2*n-1	STPoint_n.a	MU	0	Start (forward) / Stop (backward) point no. n
2*n	STPoint_n.b	MU	0	Stop (forward) / Start (backward) point no. n

Index	Name	Unit	Value	Description
Fix Point				
1	FixPoint_1.master	MU	0	Fix point no. 1 - master coordinate
2	FixPoint_1.slave	UU	-	Fix point no. 1 - slave coordinate
3	FixPoint_1.type	(dec)	C	Fix point no. 1 - type of point (C = Curve Point, T = Tangent Point
4	...			
5	...			
6	...			
3*n-2	FixPoint_n.master	MU	MasterCycleLen	Fix point no. n - master coordinate
3*n-1	FixPoint_n.slave	UU	-	Fix point no. n - slave coordinate
3*n	FixPoint_n.type	(dec)	C	Fix point no. n - type of point (C = Curve Point, T = Tangent Point)
Interpolation Point				
1	IntPoint_1	UU	0	Interpolation Point no. 1 - slave coordinate
...				
n	IntPoint_n	UU	-	Interpolation Point no. n - slave coordinate
StartStop Indices				
1	STPoint_1.a-index	(dec)	0	Index in Interpolation Array, corresponding to Start point
2	STPoint_1.b-index	(dec)	0	Index in Interpolation Array, corresponding to Start point
3	..			
StartStop Velocities				
1	STPoint_1.a-veloc.	(dec)	(*100000)	Velocity (UU/MU * 100000) in Start point
2	STPoint_1.b-veloc.	(dec)	(*100000)	Velocity (UU/MU * 100000) in Start point
...				
StartPath Interpolation Points				
1	StartPoint_1	UU	0	Interpolation Point no. 1 - for start path
...				
n				
StopPath Interpolation Points				
1	StopPoint_1	UU	0	Interpolation Point no. 1 - for stop path
...				
n				

☐ Curve Arrays and Curve Types

Starting with MCO 5.00 no interpolation points are used anymore. So only the fix points are relevant for the curve. When a SETCURVE is executed (or when the curve is really started), the coefficients for the corresponding polynomials are calculated. Then, when the curve is running, the polynomials are calculated on the fly, while driving.

This procedure allows the user to modify a curve on the fly within the application program. This can be done by overwriting some of the values within the curve array. (See description of curve array in Illustrations.). After that a SETCURVE must be executed to activate the modified curve. This curve is then started as soon as the active curve ends, or immediately by replacing the active curve, if the new curve is of type INTRPT_GRAD (see array description).



NB!:
The curve array is used by the internal SYNCC procedures as long as the curve is running. So you should never modify the curve array of a running curve. To solve this problem you must have two arrays which are used alternatively. That means while one curve is active, the next one can be prepared and started. As soon as the new one is active (PFG_CWRAP changes), the old curve can be modified again.

Type of Fix points

Starting with MCO 5.00 there are new types of fix points. The way tangent points are used is changed. Now the first fix point always tells what type of segment is following. So the last point is always of same type as the first point.

There are new points like Poly3 and Trapezoidal which allow special segments within the curve to be used.

For new curves only the bold point types should be used.

#define CU_CPOINT 1 // Curve point (next segment is 3'rd or 5'th order polynomial).

#define CU_T1POINT 2 // Tangent start point (replaced by CU_TPOINT).

#define CU_T2POINT 3 // Tangent end point (replaced by CU_CPOINT).

#define CU_TPOINT 4 // Next segment is a tangent segment.

#define CU_ZPOINT 5 // Next segment is a trapezoidal segment.

#define CU_3POINT 6 // Next segment is a 3'rd order polynomial.

So you can, for example, create sequences like .. 4 – 5 – 4 – 5 which means that you will have two straight lines (tangents) which are connected by two parabolas. Those two parabolas meet each other in the middle between the fix points and at all three points (start, middle, end) of the segments, the velocity is the same as the adjacent segment.

The following new curve types can be used by changing the array.

New Curve type 3 – CU_GRAD

Starting with MCO 5.00 another type of curve (3) is supported. This curve consists of only 2 fix points and is calculated as a polynomial of 5th order. Therefore, the following values were added at the end of the fix point area in the curve array (G_CFPIdx is the index of the fix point area):

```
RestartCurve[3] =              3                      // Curve type
RestartCurve[G_CFPIdx+0] =     0                      // Master start coordinate
RestartCurve[G_CFPIdx+1] =     0                      // Slave start coordinate
RestartCurve[G_CFPIdx+2] =     1                      // Fix Point Type = curvepoint
RestartCurve[G_CFPIdx+3] = G_ShingleDistance * 4      // Master end coordinate
RestartCurve[G_CFPIdx+4] = G_ShingleDistance * 2      // Slave end coordinate
RestartCurve[G_CFPIdx+5] =     1                      // Fix Point Type = curvepoint
RestartCurve[G_CFPIdx+6] =     0                      // Velocity v0
RestartCurve[G_CFPIdx+7] =     1                      // Divisor v0
RestartCurve[G_CFPIdx+8] =     0                      // Acceleration a0
```

```
RestartCurve[G_CFPIdx+9] =     1                  // Divisor a0
RestartCurve[G_CFPIdx+10] =    0     // Jerk j0
RestartCurve[G_CFPIdx+11] =    1     // Divisor j0
RestartCurve[G_CFPIdx+12] =    1     // Velocity v1
RestartCurve[G_CFPIdx+13] =    1     // Divisor v1
RestartCurve[G_CFPIdx+14] =    0     // Acceleration a1
RestartCurve[G_CFPIdx+15] =    1     // Divisor a1
RestartCurve[G_CFPIdx+16] =    0     // Jerk j1
RestartCurve[G_CFPIdx+17] =    1     // Divisor j1
```

That means we have the possibility to define start and end gradients and acceleration for the polynomial. (Jerk is ignored at the moment. Designed for future use.)

In this array, the start coordinates are only for display purposes because they are replaced by the actual values when the curve is started. (see below)

As a result of this behavior of type 3 curves (predefined end values and calculated start values), they normally cannot be continued when they reach the end (since typically the velocities do not match). Therefore, they are normally continued by another standard curve. If for any reason they are not continued by another curve, they will just try to continue with the actual velocity. This is done with a poly5 looking more or less like a straight line.

NB!:
This continuation overwrites the original curve array with this continuation curve.

Continuation did not work for curves with more than 2 fix points prior to 6.7.11. In those versions, there was also an error when CU_GRAD curves with more than 2 fix points where used as a continuation curve.

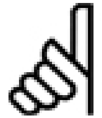
New Curve type 4 CU_GRAD_INTRPT

Type 4 is available starting with MCO 5.00. This type is nearly identical to type 3 (CU_GRAD).

The big difference with curves of type CU_GRAD_INTRPT is that they are started immediately when the SETCURVE is executed. When this is done, the actual values for velocity and acceleration are used for the calculation. The actual values of the MCPOS and CURVEPOS are subtracted from the end coordinates of the curve before it is calculated (curves must always internally start at 0,0). This guarantees, that the original end coordinates are absolute to the start of the interrupted curve.

For example, assume that a curve is running which starts at 0,0 and ends at 2000,2000 (master,slave). Now we define a curve of type CU_GRAD_INTRPT, which starts anywhere and ends at 4000,4000. If this curve is now set by SETCURVE at the moment when the original curve passes 1500,1800, for example, then the new curve is calculated in such a manner that it starts at this point (1500,1800) and ends at 4000,4000. To realize this, it uses the velocity and acceleration in the actual point, sets MCPOS and CURVEPOS to 0 and reduces the end coordinates to (4000-1500, 4000-1800) = (2500,2200). It will have the defined velocity (v1) and acceleration (a1) defined in the curve array.

These types of curve are used for processes where the standard curve looks more or less like a straight line (SYNCP / SYNCM behavior) and where the poly5 curves are used to align start or stop or restart processes to defined points.



NB!:
The responsibility for the correctness of poly5 curve lies with the user / application. The firmware does not do any plausibility test.

To allow readability by CAM-Editor, the CurveVersion (index 2) should be 102. Otherwise, the CAM editor will not accept those new curves.

Curve type 3 – CU_GRAD with SYNCCSTART

Such curves (Poly 5) can now also be started with SYNCCSTART. This means the curve starts immediately and does not wait for next marker. (In previous versions, it was only possible to start such a curve with SYNCMSTART 2001.)

If such a curve is started now with SYNCCSTART, the user is responsible for the correct setting of the endpoints. Startpoint is 0 which will be internally set to the actual command position. Hence, the curve must be defined from 0 .. endpoint.

Example:

```
G_CFPIdx = StartCurve[10]
  // Array index, where fix point definition starts
  // 1. Fix Point must be 0/0,
  // because curve always has to start at this position
StartCurve[G_CFPIdx+0]  = 0    // Master start coordinate
StartCurve[G_CFPIdx+1]  = 0    // Slave start coordinate
StartCurve[G_CFPIdx+2]  = 1    // Fix Point Type = curvepoint
StartCurve[G_CFPIdx+3]  = G_ShingleDistance * 4
     // Master end coordinate
StartCurve[G_CFPIdx+4]  = G_ ShingleDistance * 2
     // Slave end coordinate
StartCurve[G_CFPIdx+5]  = 1    // Fix Point Type = curvepoint
StartCurve[G_CFPIdx+6]  = 0    // Velocity v0
StartCurve[G_CFPIdx+7]  = 1    // Divisor v0
StartCurve[G_CFPIdx+8]  = 0    // Acceleration a0
StartCurve[G_CFPIdx+9]  = 1    // Divisor a0
StartCurve[G_CFPIdx+10] = 0    // Jerk j0
StartCurve[G_CFPIdx+11] = 1    // Divisor j0
StartCurve[G_CFPIdx+12] = 1    // Velocity v1
StartCurve[G_CFPIdx+13] = 1    // Divisor v1
StartCurve[G_CFPIdx+14] = 0    // Acceleration a1
StartCurve[G_CFPIdx+15] = 1    // Divisor a1
StartCurve[G_CFPIdx+16] = 0    // Jerk j1
StartCurve[G_CFPIdx+17] = 1    // Divisor j1

SETCURVE StartCurve
….
SYNCC 0
SYNCCSTART 0
```

In such a case, when SYNCCSTART is executed, the 0 point of the curve is mapped onto the actual command position. Also, the actual velocity is calculated (and start acceleration is set to zero).

Curve type 3 – CU_GRAD with SYNCCSTART and DEFSYNCORIGIN

In addition to the above mentioned possibility, you have the option to define the endpoints of the curve with absolute values. So the start of such a curve could look like

```
mendpos = MIPOS + mdistqc
  // apos must be in qc
sendpos = (sdistqc % G_SlaveQcProProdukt) * G_SlaveQcProProdukt + SYSVAR[4098]
defsyncorigin mendpos sendpos
  // define target position for master and slave (in qc)
SYNCC 0
SYNCCSTART 0
```

In this case, the curve is again started immediately and the actual command position will be the Curve zero position. However, the end position of master and slave are calculated in such a way that the curve will end at the given absolute positions.

Final velocity is given by the curve (1 in our example above) and start velocity is taken from actual values.

Curve types CU_GRAD with stability check

Whenever the new curve types are calculated, a check for extremes within a poly5 is also done if possible. (This only works if start acceleration is 0.) If an extreme within the interval is found, the curve error flag PG_FLAG_CURVE_ERR is set and the error number is stored in the PFG_G_LastError which allows the user to detect this situation. At the same time, the PFG_G_CPOLYMAXVEL [4288] and PFG_G_CPOLYMINVEL [4289] (SU/MU) are stored. Writing to PFG_G_LastError [4258] clears the flag PG_FLAG_CURVE_ERR (Bit 64 <<24) in the STAT.

So a sample could look like.

```
IF((STAT x(1))&(64<<24)) THEN // error bit set
              switch(SYSVAR[4258])
               case 5 :  PRINT " Minimum in interval ",SYSVAR[4289] * 100 % 128
                         break
               case 6 :  PRINT " Maximum in interval ",SYSVAR[4288] * 100 % 128
                         break
               case 7 :  PRINT " Minimum in interval ",SYSVAR[4289] * 100 % 128
                         PRINT " Maximum in interval ",SYSVAR[4288] * 100 % 128
                         break
               default : PRINT " Other curve error ",SYSVAR[4258]
              endswitch
ENDIF
```

□ Illustrations

Bin File Map (Compiler >= 6.7.0)

Offset (Bytes) CodeVersion >= 5

Offset (Bytes) CodeVersion < 5

Offset (CodeVersion >= 5)	Offset (CodeVersion < 5)	Content
0	0	COD_VERSION (0)
2	1	CodeVersion
4	2	COD_PRGHDR (119 / 0x77) - (0x77 0x00) (CodeVersion >= 5)
6	3	Size of Header information in Bytes
8	5	MaxVars (number of Variables defined in Aposs + 20)
10	7	Stacksize (size of stack defined in Aposs)
12	9	First command of the application program
14	10	.... more code ....
...	...	....
n	n	COD_ENDE (127 / 0x7F) - (0x00 0x7F) (CodeVersion >= 5)
0	0	Filename[0] - 1st character of file name (s = strlen)
1	1	Filename[1] - 2nd character of filename
..	..	more characters
s-1	s-1	Filename[s-1] - last character of fname
s	s	Filename[s] - terminating \0 character
?	?	Extra \0 if length of filename was odd
0	0	FirmwareVersion (long - 4 bytes)
4	4	Minfw (Minimum Firmware Version) (long 4 - bytes)
8	8	Cmpvers (Compiler Version) (long 4 - bytes)
12	12	filtime.dwHighDateTime (long 4 - bytes)
16	16	filtime.dwLowDateTime (long 4 - bytes)
20	20	InfoSize (in bytes) - l(long 4 bytes -- PC notation, LSB first)
24	24	0x55FF44EE - marks end of information section (LSB first)

CodeVersion is as follows
2 - firmware < 6.04.00 exesize had to be smaller than 65000
3 - firmware < 6.06.00
4 - new fp commands and optimizing commands
5 - firmware < 6.06.00 - commands and parameters are always 2 bytes , all jumps are 4 bytes.
6 - firmware >= 6.06.00 - firmware supports new posints and posintio (axes specific) OnStatbit does support axes now.

This section is only present (for compatibility with old versions) if MaxVars > 112. This number is calculated as 92 + 20, where 92 is the number of variables in old controllers and 20 is the number of temporary registers. This number was fix in old controllers. And those controllers do not understand the PRGHDR command.

Size of program header in bytes including command and sizeinfo.
7 - for CodeVersion < 5
8 - for CodeVerision >= 5 (all commands are now 2 bytes)

Since version 6.06.00 (Codeversion >= 5) all commands and parameters have at least 2 bytes. Therefore the end of a program looks like 0x7f 0x00 now.

The complete filename is appended. The size of this block is always even. There may be two \0 at the end if necessary.

a number like 60650 means 6.06.50

minimum firmware version required for binary program

Compiler version used to produce binary code (i.e. 60520 means 6.05.20)

"File time" is the number of 100-nanosecond intervals that have elapsed since 12:00AM, January 1, 1601 (UTC). This can be converted back to a "nice" local time using the functions FileTimeToSystemTime() and SystemTimeToTzSpecificLocalTime().

Size of info section - at the moment this is (s+1+20 or s+2+20)

All longs in the info section are in PC notation which means LSB first.

All longs in other sections are MSB first, for Codeversion < 7 which means firmware < 6.7.0
For firmware >= 6.7.01 Codeversion 7 is used and the Integers are ordered corresponding to the CPU.
That means MSB first for Motorola and LSB first for DSP.

BinFileMap_6_5.vsd
2009-08-31-bi

CurveArray (Long orientiert)

		Field	Description
1	1	Identification (999.000.001)	
2	2	Version (101, 102)	
3	3	CurveType	Range is 0 .. 4
4	4	CurveName1 (4char)	
5	5	CurveName2 (4char)	
6	6	CurveName3 (4char)	
7	7	CurveName4 (4char)	
8	8	IndexCIF - CurveInformation -	Default = 17
9	9	Index STP - Start-Stop Points	Default = 31 (32 for Version 102)
10	10	Index FIP - Fixpoint Part	Default STP+STPno*2
11	11	Index INP - Interpolation Part	Default FIP + FixPointNo * 3
12	12	Index STPI - StartStopInerpInd	Default INP + InterPolPointNo
13	13	Index STPV - StartStopVel	Default STP + STPno *2
14	14	Index STIP - StartPathInterpol	Default STPV + STPno*2
15	15	Index STPIP - StopPathInterpol	Default STIP + MaxStartStopLen
16	16	Extra Info Index (CU_GRAD / LBLINF - Optional Label Info)	
17	1	MasterCycleLen (MU)	Length of Curve in Master Units
18	2	SlaveCycleLen (UU) -	Max. Slave travel dist in UU
19	3	FixPoint number	Number of fix points (minimum 2)
20	4	InterpolPointNo	Number of interpol points
21	5	Interpolation Type	(0 = open, 1 = periodic, 2+3 spec)
22	6	SlaveStopPosition	position slave has to be after stop
23	7	CorrectionStartPoint	Pos. where correction may start
24	8	CorrectionStopPoint	Pos. where correction has to stop
25	9	Maximum Correction	maxmal allowed correction
26	10	MaxStartStopLen	max length of start stop path
27	11	StartStopNo	Number of start stop point pairs
28	12	MMaxCycles	Max cycles per minute (info only)
29	13	MMarkerPos	Master Marker Pos. in curve
30	14	SMarkerPos	Slave Marker Pos in curve (cmd)
31	15	ExtraDataSize	Size of extra Data (Version>=102)
32	1	STPoint_1a	Start Point Pair 1 - Point A
		STPoint_1b	Start Point Pair 1 - Point B
		....	
		FixPoint_1.master	Master Coordinate
		FixPoint_1.slave	Slave Coordinate
		FixPoint_1.type	Type (C = curve, T = Tangent)
		....	
		FixPoint_n.master	Master Coordinate
		FixPoint_n.slave	Slave Coordinate
		FixPoint_n.type	Type (C = curve, T = Tangent)
		StartVelocityNum	used for Poly5 and Splines
		StartVelocityDen	
		StartAccelerationNum	(not used for Splines)
		StartAccelerationDen	(not used for Splines)
		StartJerkNum	(not used at the moment)
		StartJerkDen	(not used at the moment)
		EndVelocityNum	used for Poly5 and Splines
		EndVelocityDen	
		EndAccelerationNum	(not used for Splines)
		EndAccelerationDen	(not used for Splines)
		EndJerkNum	(not used at the moment)
		EndJerkDen	(not used at the moment)
		Interpolatio nSection (Interpolation Points, Start Stop Indices, StartStop Velocities, StartPath Interpolation Points, StopPath Interpolation Points)	

defines handling of start / end velocities.
0 = start and end velocity is average,
1 = end velocity is set to start velocity (Hauser compatible).
2 = start and end gradients are set to 0 (5th order).
3 = start and end gradients are user defined (CU_GRAD)
4 = start and end gradients user defined (CU_GRAD_INTRPT) (starts immediately)

not used any more, because curves are calculated on the fly now

optional Label Info, only used if Interpolation Type (see 21) is >= 2
Since Version 102 used if CurveType(see 3) is CU_GRAD (3) or CU_GRAD_INTRPT(4). In that case the extra info contains start and stop gradients.

internally overwritten by FixPoint_n.master - FixPoint_1.master

internally overwritten by (FixPoint_n.slave- FixPoint_1.slave)

maximum is 100 at the moment

not used any more, because curves are calculated on the fly now

Interpolation Types are interpreted as follows:
0 = open (only relevant for CAM - editor)
1 = periodic (only relevant for CAM - editor)
2 = Labeling - old version with precalculation
3 = Labeling - actual development version

wird im Moment nicht übergeben !! Stop ohne CURVESTOP wird wohl nicht funktionieren

next 3 only relevant for marker correction

not used any more, because curves are calculated on the fly now

next 2 only relevant for marker correction

At the moment length is 12

values are given in MU. Point A is where the stopping begins and B is where it has to be finished. If driving backward, it is vice versa.

Fixpoints are give in MasterUnits (master coordinate) or UserUnits (slave coordinate). The type can be either
C = CurvePoint
T = TangentPoint
More then two curvepoints are interpolated by cubic spline functions. That means, that a 3rd order Polynom connects two curvepoints in that way, that the position, velocity and acceleration are identical in intermediate Curvepoints.
If there is only one CurvePoint followed by an TangentPoint ore a TangentPoint followed by only one CurvePoint (endpoint) or two tangents following each other, then we use 5th order polynoms to connect these points.

Velocity is used, if the CurveType is 3 or 4 (CU_GRAD or CU_GRAD_INTRPT) and curve does not start with a tangent. If curve starts with a tangent, all other values are ignored too.

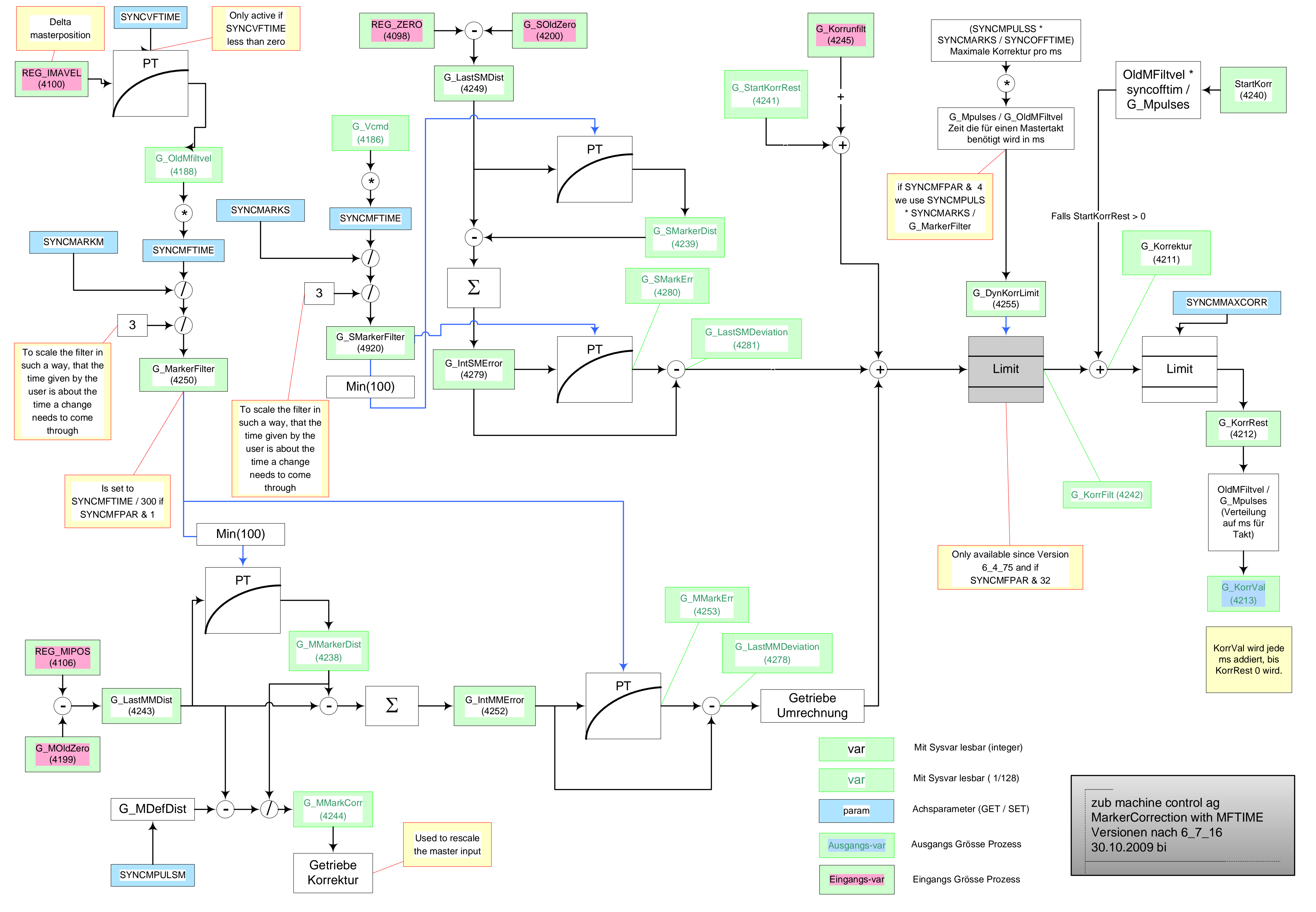
Acceleration is used, if the CurveType is 3 or 4 (CU_GRAD or CU_GRAD_INTRPT) and curve starts with a Poly5.

Velocity is used, if the CurveType is 3 or 4 (CU_GRAD or CU_GRAD_INTRPT) and curve does not end with a tangent. If curve ends with a tangent, all other values are ignored too.

Acceleration is used, if the CurveType is 3 or 4 (CU_GRAD or CU_GRAD_INTRPT) and curve ends with a Poly5.

not used any more, because curves are calculated on the fly now

CurveArray_6_6_x.vsd
2008-05-24 bi



Delta masterposition
SYNCVFTIME
Only active if SYNCVFTIME less than zero
REG_IMAVEL (4100)
PT
G_OldMfiltvel (4188)
SYNCMARKM
SYNCMFTIME
3
To scale the filter in such a way, that the time given by the user is about the time a change needs to come through
G_MarkerFilter (4250)
Is set to SYNCMFTIME / 300 if SYNCMFPAR & 1
SYNCMARKS
G_Vcmd (4186)
SYNCMFTIME
3
G_SMarkerFilter (4920)
Min(100)
To scale the filter in such a way, that the time given by the user is about the time a change needs to come through
REG_ZERO (4098)
G_SOldZero (4200)
G_LastSMDist (4249)
PT
G_SMarkerDist (4239)
Σ
G_IntSMError (4279)
PT
G_SMarkErr (4280)
G_LastSMDeviation (4281)
G_StartKorrRest (4241)
G_Korrunfilt (4245)
(SYNCMPULSS * SYNCMARKS / SYNCOFFTIME) Maximale Korrektur pro ms
G_Mpulses / G_OldMFiltvel Zeit die für einen Mastertakt benötigt wird in ms
if SYNCMFPAR & 4 we use SYNCMPULS * SYNCMARKS / G_MarkerFilter
G_DynKorrLimit (4255)
Limit
Only available since Version 6_4_75 and if SYNCMFPAR & 32
G_KorrFilt (4242)
OldMFiltvel * syncofftim / G_Mpulses
StartKorr (4240)
Falls StartKorrRest > 0
G_Korrektur (4211)
SYNCMMAXCORR
Limit
G_KorrRest (4212)
OldMFiltvel / G_Mpulses (Verteilung auf ms für Takt)
G_KorrVal (4213)
KorrVal wird jede ms addiert, bis KorrRest 0 wird.
Min(100)
PT
REG_MIPOS (4106)
G_MOldZero (4199)
G_LastMMDist (4243)
G_MMarkerDist (4238)
Σ
G_IntMMError (4252)
PT
G_MMarkErr (4253)
G_LastMMDeviation (4278)
Getriebe Umrechnung
G_MDefDist
SYNCMPULSM
G_MMarkCorr (4244)
Getriebe Korrektur
Used to rescale the master input
var
Mit Sysvar lesbar (integer)
var
Mit Sysvar lesbar (1/128)
param
Achsparameter (GET / SET)
Ausgangs-var
Ausgangs Grösse Prozess
Eingangs-var
Eingangs Grösse Prozess
zub machine control ag
MarkerCorrection with MFTIME
Versionen nach 6_7_16
30.10.2009 bi

Index

#

#INCLUDE 134

[

[Esc] key 122

_

_GETVEL 133

A

Abbreviations 5
ACC 8
APOS 9
APOSDIFF 10
APOSS Compiler enhancements 136
APOSS Tools 135
APOSS User Interface Changes 136
Array Structure of CAM Profiles 139
AVEL 10
AXEND 11

C

CAM Array Definition 140
CANDEL 12
CANIN 12
CANINI 14
CANOUT 15
COMOPTGET 16
COMOPTSEND 16
CONTINUE 17
CPOS 17
CPOSDIFF 18
CSTART 18
CSTOP 19
Curve Arrays 143
Curve Types 143
CURVEPOS 20
CVEL 21

D

DEC 21
DEFCANIN 22
DEFCANOUT 22
DEFCORIGIN 23
DEFMCPOS 23
DEFMORIGIN 24
DEFORIGIN 24
DEFSYNCORIGIN 25
DELAY 27
DELETE ARRAYS 27
DIM 28
DISABLE ... interrupts 29

E

ENABLE ... interrupts 31
ENCPOSOFFS 32
ENCTGREAD 33
ENCTGWRITE 34
ERRCLR 34
ERRNO 35
EXIT 35

G

GET 36
GETVLT 37
GETVLTSUB 37
GOSUB 38
GOTO 38

H

HOME 39

I

IF .. THEN .. , ELSEIF .. THEN .. ELSE .. ENDIF 40
Illustrations 147
IN 41
INAD 42
INB 43
INDEX 44
INGLB 44
INKEY 45
INMSG 46
IPOS 47
IPOSDIFF 48

J

JERKFINVEL 49
JERKSTOPDIST 49

L

LINKGPAR 50

LINKPDO ... 51
LINKSDO ... 52
LINKSYSVAR ... 54
Literature ... 4
LOOP ... 55

M

MAPOS ... 55
MAPOSDIFF ... 56
Master Units [MU] ... 6
MAVEL ... 56
MENCPOSOFFS ... 57
MENCTGREAD ... 57
MENCTGWRITE ... 58
MIPOS ... 59
MIPOSDIFF ... 60
MLONG ... 5
MOTOR OFF ... 60
MOTOR ON ... 61
MOTOR STOP ... 61
MOVESYNCORIGIN ... 62
MSGVAL ... 62

N

New and Extended Parameters ... 138
New Commands ... 137
NOWAIT ... 63

O

ON CANINPUT ... 64
ON CANMSG GOSUB ... 64
ON COMBIT .. GOSUB ... 65
ON DELETE .. GOSUB ... 66
ON DELETE .. SETOUT ... 67
ON ERROR GOSUB ... 68
ON INT .. GOSUB ... 69
ON PARAM .. GOSUB ... 71
ON PERIOD ... 72
ON posint .. GOSUB ... 73
ON posint .. SETOUT (TOIN) ... 75
ON STATBIT .. GOSUB ... 76
ON TIME ... 77
OUT ... 78
OUTAN ... 79
OUTB ... 80
OUTDA ... 81
OUTMSG ... 82

P

PCD ... 82
PDO ... 83
PID ... 86
POSA ... 86
POSA CURVEPOS ... 87
POSR ... 87
PRINT ... 88
PRINTDEV ... 88
PULSACC ... 89
PULSVEL ... 90

Q

Quad-counts ... 5

R

REPEAT .. UNTIL ... 90
RSTORIGIN ... 90

S

SAVE part ... 91
SAVEPROM ... 91
SDOREAD ... 92
SDOREADSEG ... 93
SDOREADSEGP ... 94
SDOSTATE ... 95
SDOWRITE ... 96
SET ... 96
SETCURVE ... 97
SETMORIGIN ... 98
SETORIGIN ... 98
SETVLT ... 99
SETVLTSUB ... 99
STAT ... 100
SUBMAINPROG .. ENDPROG ... 101
SUBPROG name .. RETURN ... 101
Symbols ... 4
SYNCC ... 102
SYNCCMM ... 104
SYNCCMS ... 105
SYNCCSTART ... 105
SYNCCSTOP ... 106
SYNCERR ... 107
SYNCM ... 108
SYNCMARKERSTART ... 109
SYNCP ... 110
SYNCSTAT ... 111
SYNCSTATCLR ... 112
SYNCV ... 113
System Process Data ... 114
SYSVAR ... 114

T

Technical Reference ... 139

TESTSETDEST 115
TESTSETINDEX 117
TESTSETP 119
TESTSETTIME 121
TESTSETTYPE 122
TESTSTART 124
TESTSTOP 125
TIME 127
TRACKERR 127

U

User Units [UU] 6
USRSTAT 128

V

VEL 129
VLTALARMSTAT 129
VLTCONTROL 130
VLTERRCLR 130

W

WAITAX 130
WAITI 131
WAITNDX 131
WAITP 132
WAITT 132
WHILE .. DO .. ENDWHILE 133