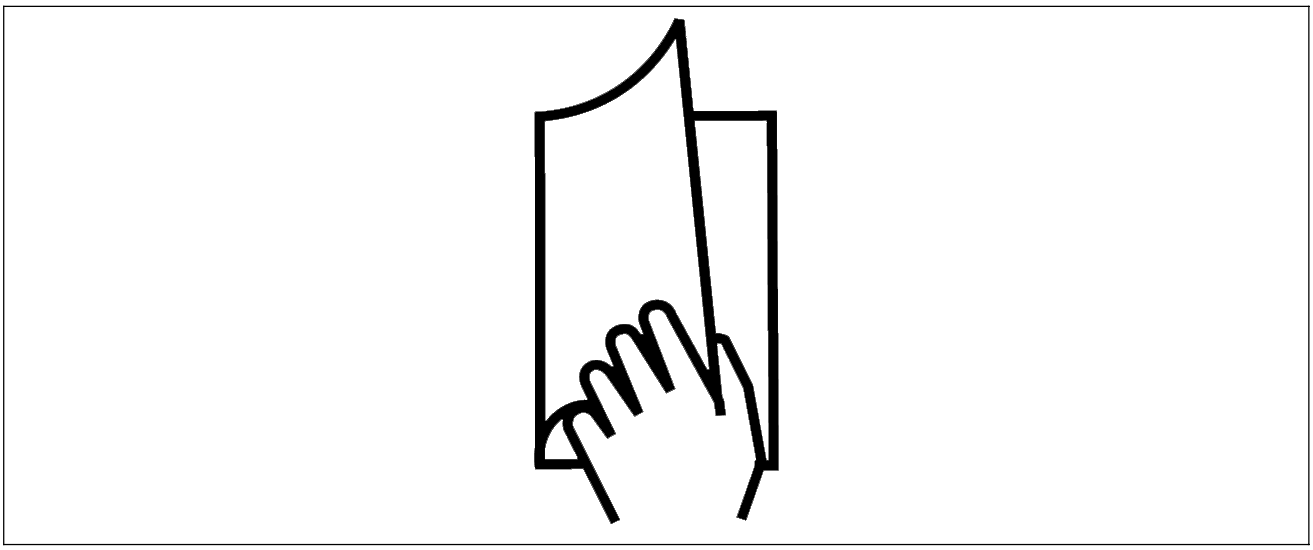
## Contents

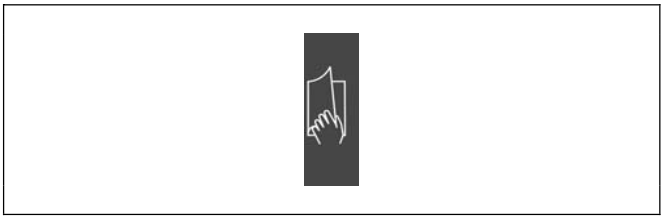
# How to Read this Design Guide



## How to Read this Design Guide

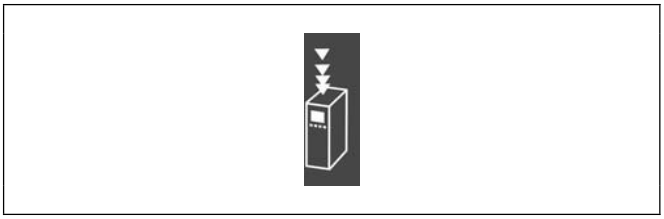
This Design Guide will introduce all aspects of your MCO 305. Please read also the Operating Instructions, in order to be able to work with the system safely and professionally, particularly observe the hints and cautionary remarks.

Chapter **How to Read this Design Guide** introduces the design guide and informs you about the symbols, abbreviations, and definitions used in this manual.



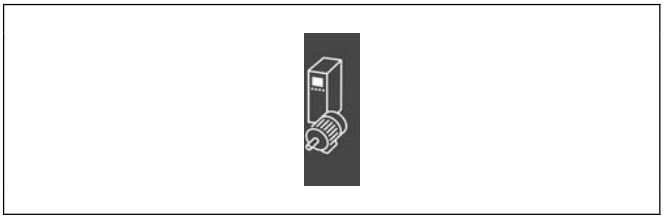
**Page divider for 'How to Read this Design Guide'.**

Chapter **Introduction to MCO 305** informs you about the functionality and features of the MCO 305, gives a system overview including configuration examples, and informs you about some basic topics like encoder and program execution.



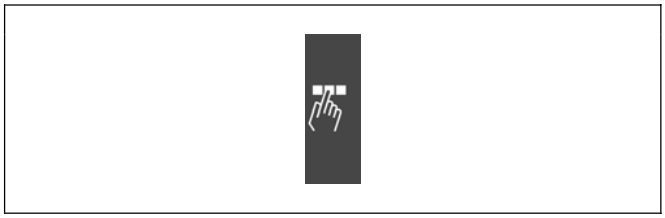
**Page divider for 'Introduction'.**

Chapter **Functions and Examples** guides you through some applications examples from simple positioning to different synchronizations as well as CAM controls. Setting the parameters, programming of controls, and editing of curves can be reconstructed in detail with these examples.



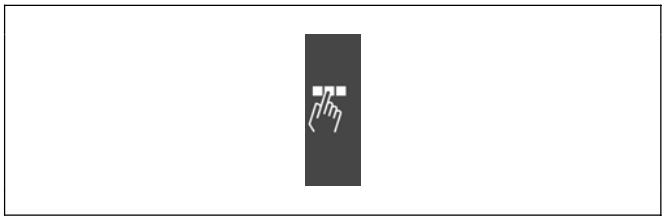
**Page divider for 'Functions and Examples'.**

Chapter **PC Software Interface** informs you about the APOSS specific menus and functions mainly the CAM-Editor for editing curve profiles. Click on → *Help* in the APPOS menu bar for more details.



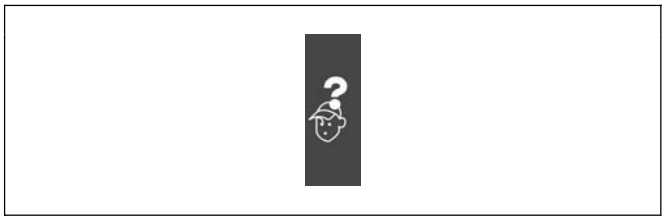
**Page divider for 'PC Software Interface'.**

Chapter **How to Program** shows you how to program controls for the Frequency Converter using MCO 305. This chapter provides a detailed description of all commands and all parameters in the sections Software and Parameter Reference.



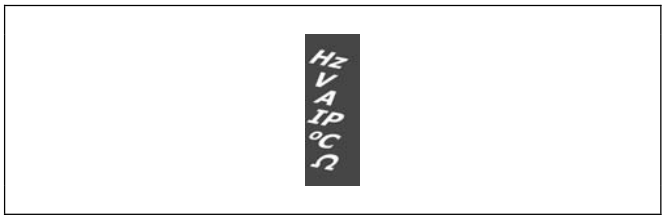
**Page divider for 'How to Program'.**

Chapter **Troubleshooting** assists you in solving problems that may occur when using the frequency converter with MCO 305. The next section explains the most important messages from the PC user interface.



**Page divider for 'Troubleshooting'.**

Chapter **Appendix** gives a quick review of what has changed since previous releases in "What is 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 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 Design Guide:

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

Indicates a general warning.

Indicates a high-voltage warning.

***** Indicates default setting.

**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 [Cntl] key, or just [Cntl], the [Esc] key or the [F1] key.

## □ Abbreviations

| | |
|---|---|
| Automatic Motor Adaptation | AMA |
| Control word | CTW |
| Direct Current | DC |
| 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 |

| | |
|---|---|
| Switch normally closed | nc |
| Switch normally open | no |
| 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

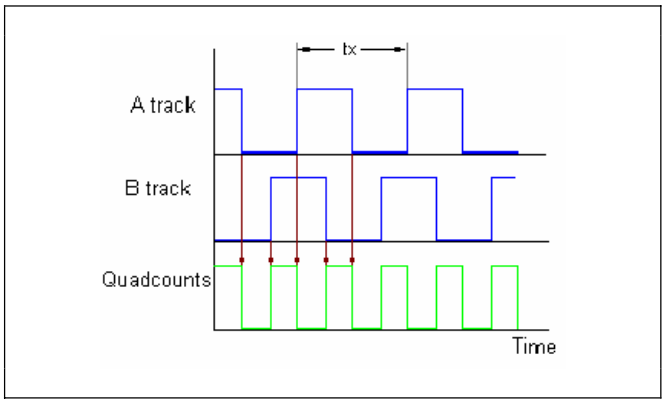
### □ Online / Offline Parameters

Changes to online parameters are activated immediately after the data value is changed. Changes to offline parameters are not activated until you enter [OK] on the LCP.

### □ 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**

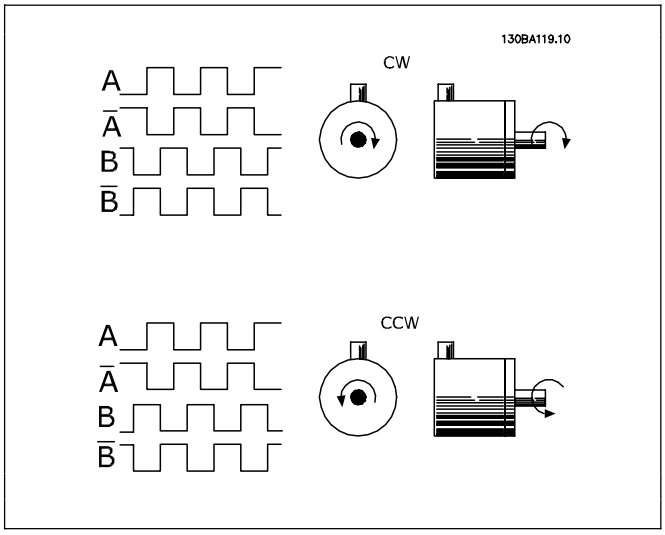
### □ Encoder Direction

The direction of encoder is determined by which order the pulses are entering the drive.

Clockwise direction means channel A is 90 electrical degrees before channel B.

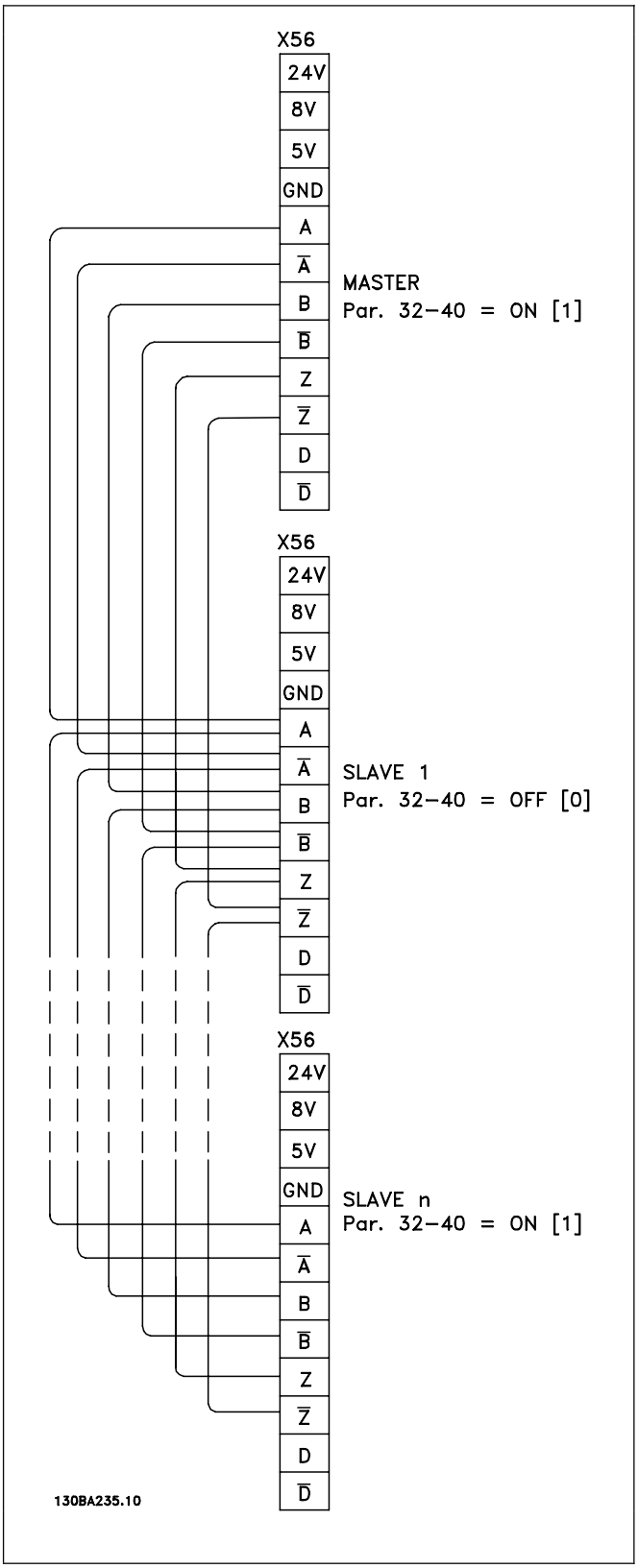
Counter Clockwise direction means channel B is 90 electrical degrees before A.

The direction determined by looking into the shaft end.



### □ Virtual Master

Virtual master is an encoder simulation which serves as a common master signal for synchronization of up-to 32 axes.



### □ 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.

#### 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, POS.

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).

You can standardize the unit with a factor. This factor is a fraction which consists of a numerator and denominator:

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

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:

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

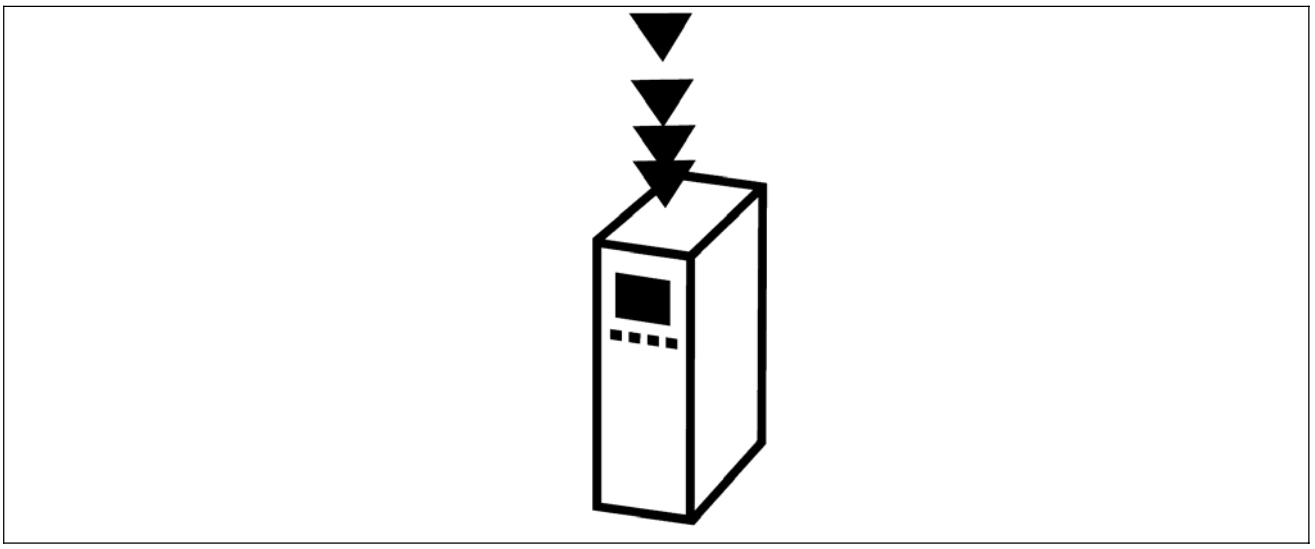
par. 33-10 *Synchronization Factor Master* SYNCFACTM
par. 33-11 *Synchronization Factor Slave* SYNCFACTS

### □ Open Loop vs. Closed Loop

Open loop is control without feedback. Closed loop control compares velocity or position feedback with the commanded velocity or position and generates a modified command to make the error smaller. The error is the difference between the required speed and the actual speed.

Open loop can be used on systems where motor velocity is not critical, and where accurate positioning is not necessary. Applications such as fan and blower control, pump control, and some low-end home appliances are examples.

# Introduction to VLT Motion Control Option MCO 305



## What is VLT Motion Control Option MCO 305?

MCO 305 is an integrated programmable Motion Controller for VLT Automation Drive FC 301 and FC 302; it adds functionality and flexibility to the already very comprehensive standard functionality of these drives.

FC 301 and FC 302 with MCO 305 is an intelligent drive offering highly accurate and dynamic motion control featuring, Synchronization (electronic shaft), Positioning and electronic CAM control. In addition the programmability offers the possibility to implement a variety of application functions such as monitoring and intelligent error handling.

Development of application programs for MCO 305 and configuration/commissioning is done via easy to use PC software tools integrated in VLT Motion Control Tools MCT 10. The PC software tools includes programming editor with program examples, CAM profile editor as well as "test-run" and "scope" function for controller optimizing. MCO 305 is based on event controlled programming using a structured text programming language developed and optimized for this application.

FC 301 and FC 302 can be delivered as an "all-in-one" drive with the MCO 305 module preinstalled or MCO 305 can be delivered as option module for field installation.

Basic Features and specifications:

- Home function.
- Absolute and relative positioning.
- Software and Hardware end limits.
- Velocity, Position and Marker synchronizing.
- CAM control.
- Virtual master function for synchronizing of multiple slaves.
- On-line adjustable gear-ratio.
- On-line adjustable offset.
- Definition of application parameters accessible via FC 300 local control panel.
- Read/Write access to all FC 300 parameters.
- Sending and receiving data via Field-bus interface (requires Field-bus option).
- Interrupts triggered by various events: Digital input, position, Field-bus data, parameter change, status change and time.
- Calculation, comparison, bit manipulation and logical gating.
- Conditional and unconditional jumps.
- Graphical PID optimizing tool.
- Debugging tools.
- Supported encoder types: 5V Incremental RS422 and SSI absolute single- and multi-turn, Gray code, adjustable clock frequency and data length.
- 3 supply voltages: 5V, 8V and 24V.

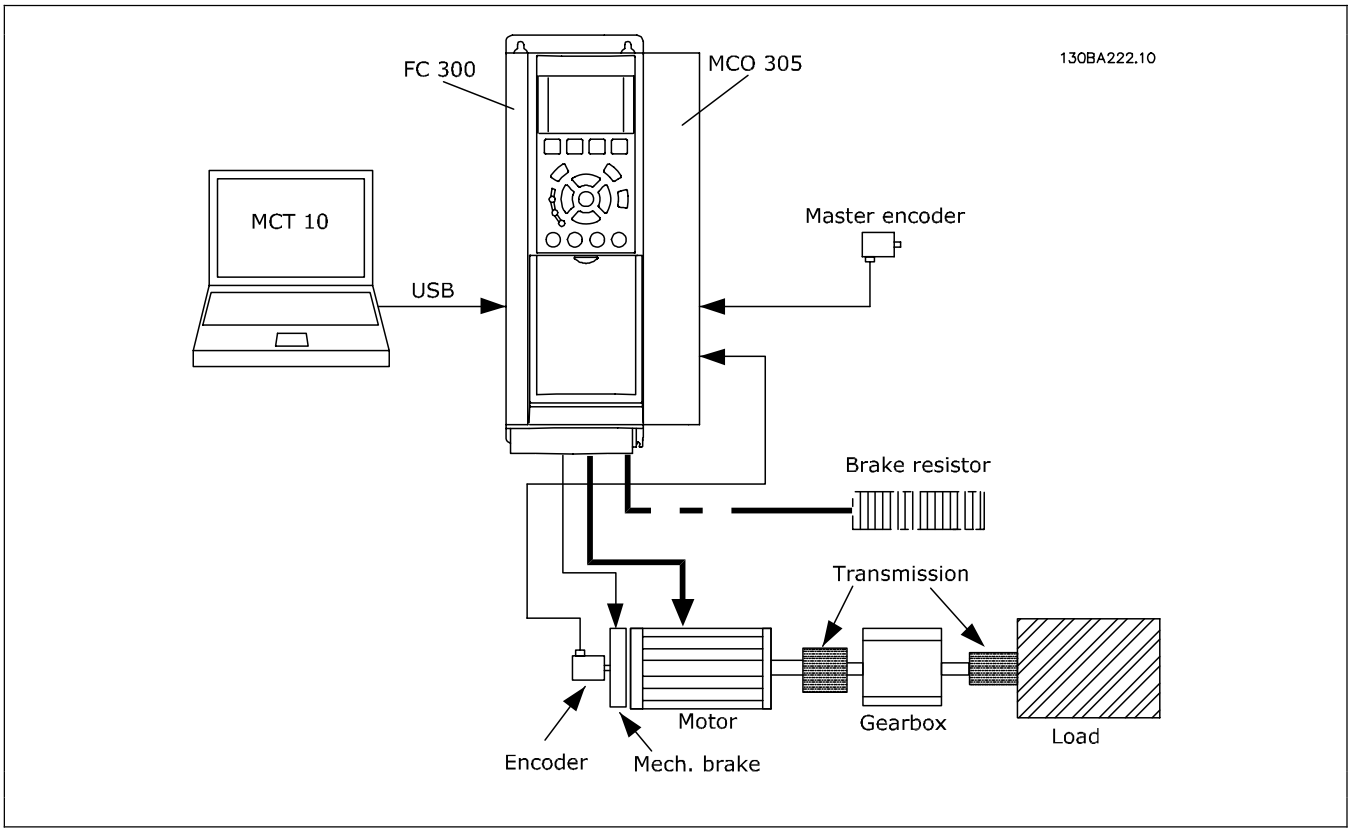
## System Overview

The MCO 305 system includes at least the following elements:

- FC 300.
- MCO 305 module.
- Motor/geared motor.
- Feedback encoder. Encoder must be mounted on motor shaft when operating FC 300 in Flux closed loop, feedback encoder for positioning and synchronizing can be mounted anywhere in the application. See "Encoders in applications" for more details.
- Master encoder (only for synchronizing).
- PC with MCT 10 for programming.

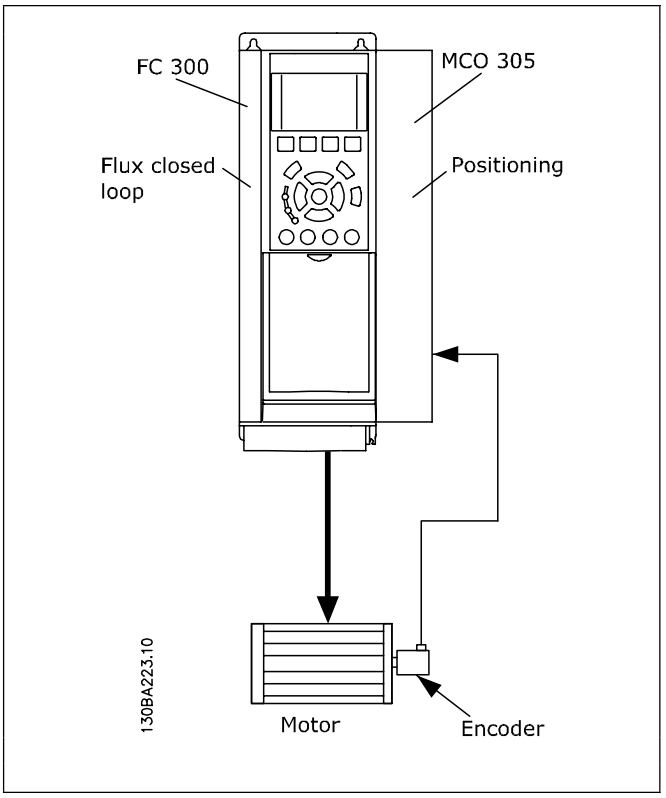
The following might also be required:

- Brake resistor for dynamic braking.
- Mechanical brake.

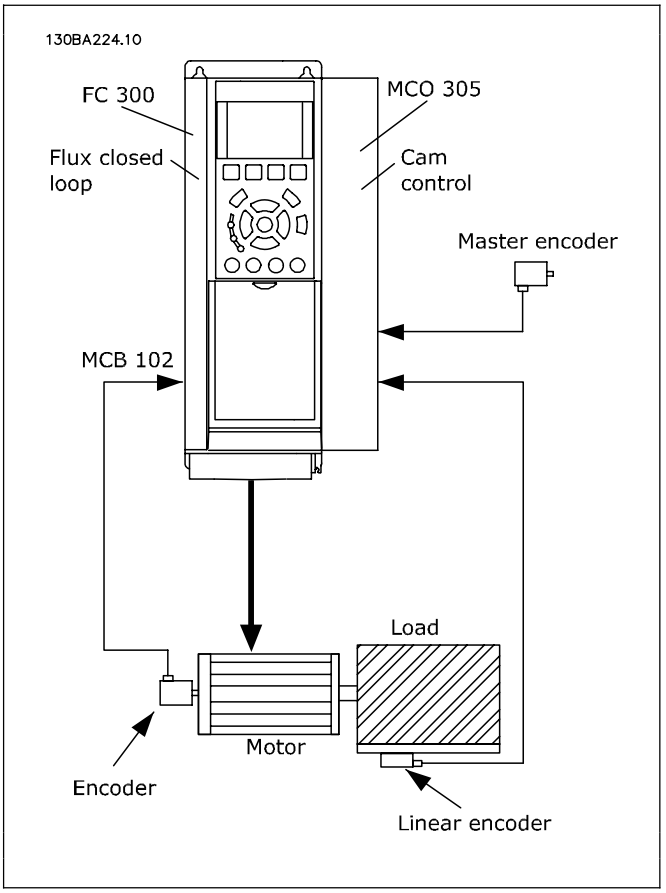


## □ Configuration Examples

One encoder used as motor feedback for closed loop Flux control as well as position feedback.




One encoder used as motor feedback for closed loop Flux control (connected via encoder option MCB 102), linear encoder used as slave position feedback and a third encoder as master.



## □ Interface between MCO 305, FC 300 and other Option Modules

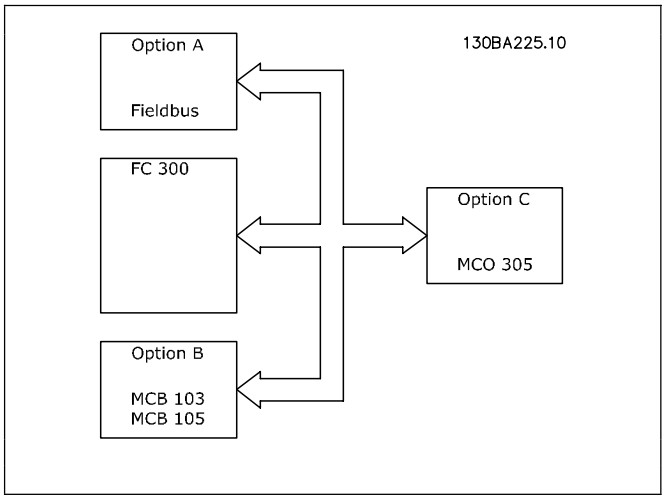
The Interface between MCO 305 and the FC 300 control card provides read/write access to all parameters as well as reading status of all inputs and the possibility to control all outputs. In addition various process data such as status word and actual motor current can be read by the MCO 305 application program.

MCO 305 is controlling FC 300 via the speed/torque reference; see section "Control loops" for further details.

Field-bus interface (e.g. PROFIBUS and DeviceNet): MCO 305 has read/write access to data received/send via the various Field-bus interfaces (requires optional Field-bus option module).

Relay option MCB 105: The relay outputs of MCB 105 can be controlled by the MCO 305 application program.

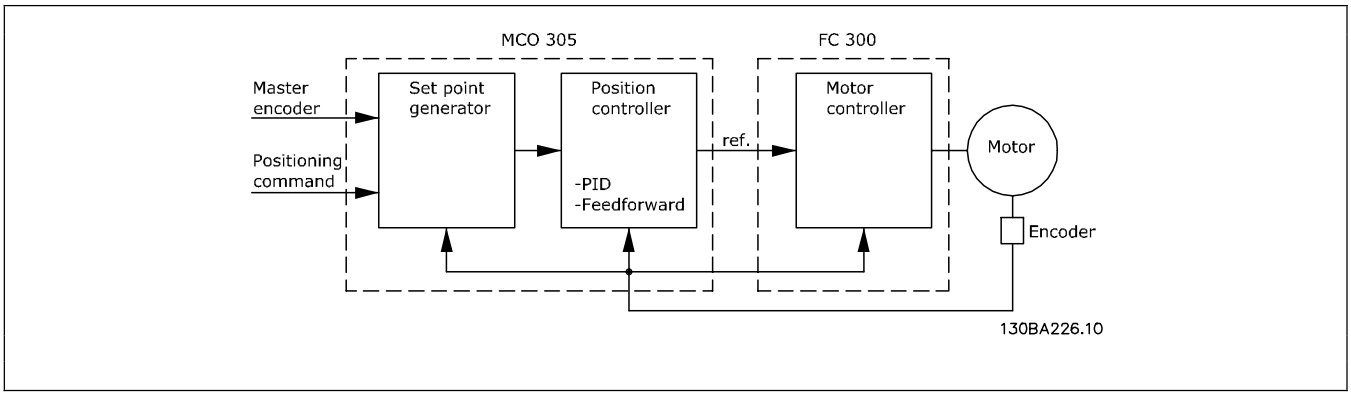
General purpose I/O option MCB 103: Status of inputs can be read and outputs can be controlled via the MCO 305 application program.




Up-/download of MCO 305 application programs and configuration data is done via the FC 300 interfaces (RS485 or USB) or via PROFIBUS DPV1 (requires optional PROFIBUS module). The same applies for the on-line PC software functions such as test-run and debugging.

## □ Control Loops

MCO 305 has a PID (Proportional, Integral, Derivative) controller for position control based on actual position (encoder feedback) and commanded position (calculated). The MCO 305 PID is controlling the position in all modes of operation except velocity synchronizing where the velocity is controlled instead. FC 300 is an "amplifier" in the MCO 305 control loop and it must therefore be optimized for the connected motor and load before the MCO 305 PID can be set-up. FC 300 can be operated in open loop or closed loop within the MCO 305 control loop, see example below:




Guideline for optimizing MCO 305 PID can be found in MCO 305 Operating Instructions.

Guideline for optimizing FC 300 can be found in FC 300 Operating Instructions.

## Encoder

MCO 305 supports various encoder types:

- Incremental encoder with RS422 signal type.
- Incremental encoder with sine–cosine signal type.
- Absolute encoder with SSI interface.

Master and feedback/slave encoder type can be selected independently; encoders can be rotary or linear. Selection of encoder type depends on application requirements and general preferences. Attention must however be paid to the resolution of the selected encoder. There are 3 important selection criteria:

- Maximum position accuracy is +/- 1 encoder increment.
- To ensure stable and dynamic control a minimum of 20 encoder increments per PID controller sample period (default is 1 millisecond) is needed at the minimum application velocity.
- Maximum frequency of the MCO 305 encoder inputs must not be exceeded at maximum velocity.

The feedback encoder can be mounted directly on the motor shaft or behind gearboxes and/or other types of transmissions. There are how ever some important issues to be aware of when mounting the encoder:

- There should be a firm connection between motor and encoder. Slip, backlash, and elasticity will reduce control accuracy and stability.
- When the encoder is running at a low speed it must have a high resolution in order to meet the above requirement (minimum 20 encoder increments per controller sample).

## Program Execution

MCO 305 can store multiple programs, up-to 90. Only one of these programs can be executed at a time, there are three ways to control which program to execute:

- Via parameter 33-80 *Activated Program Number*.
- Via digital inputs (parameters 33-50 through 33-59, 33-61 and 33-62).
- Via PC software.

One program must be defined as *Autostart* program, the Autostart program is automatically executed after power up. Without Autostart program it is only possible to execute a program via PC software.

The Autostart program is always executed first, if the Autostart program is terminated (no loop or by EXIT command) the following can happen:

1. When parameter 33-80 (*Activated Program Number)* = -1 and no input (parameters 33-50 through 33-59, 33-61 and 33-62) is selected as *Start program execution* ([13] or [14]). The Autostart program will restart.
2. When parameter 33-80 (*Activated Program Number)* ≠ -1 and no input (parameters 33-50 through 33-59, 33-61 and 33-62) is selected as *Start program execution* ([13] or [14]). The selected program (par. 33-80) will be executed.
3. When an input (parameters 33-50 through 33-59, 33-61 and 33-62) is selected as *Start program execution* ([13] or [14]) and one or more inputs are selected as *Program select* ([15]). The selected program (Program select inputs) will be executed when the *Start program execution* input is activated.

The active program can be aborted via a digital input when defining an input as *Break program execution* (Option [9] or [10] in 33-50 through 33-59, 33-61 and 33-62). The aborted program can be restarted via a digital input when defining an input as *Continue program execution* (Option [11] or [12] in 33-50 through 33-59, 33-61 and 33-62).

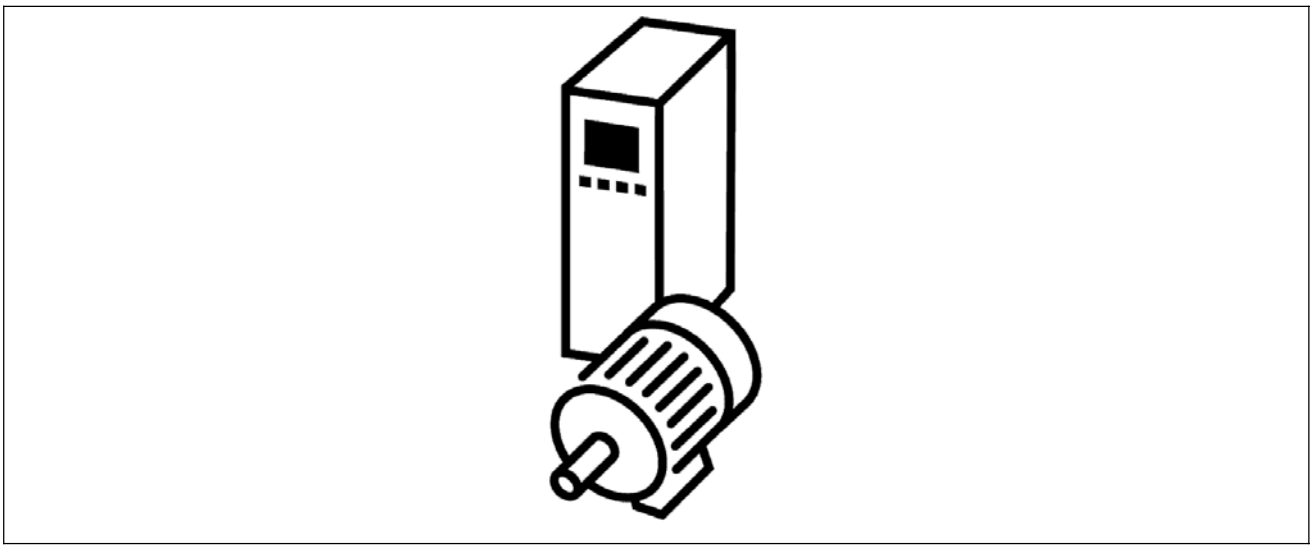
Starting the Autostart program after power-up can be avoided by pressing the [Cancel] key of the FC 300 LCP during power-up. The key must be pressed until the message User abort (error 119) appears in the display.

A temporary program can be executed from the program editor (MCT10/APOSS), temporary programs are only stored in RAM and are thus lost at power-down. The temporary program can also be executed in a special Debug mode where it is possible to influence the program execution as well as reading out data and variables, see on-line help of APOSS for further details.

When connecting a PC with MCT 10 to the drive, the active program might be aborted e.g. when downloading a new program or when working with the program editor ([Esc] will abort program execution).

**NB!:**
In case of an error the active program will be terminated if no error handler (ON ERROR GOSUB xxxx) is defined and the program will not be restarted.

# Functions and Examples



## Positioning

Basically the term positioning in connection with drives means moving the shaft to a specific position. In order to obtain accurate positioning it is necessary to use a closed loop system to control the actual position, based on position feedback from an encoder.

A positioning procedure with a closed loop positioning controller requires the following: Set velocity, acceleration, deceleration and a target position; a velocity profile is calculated based on the actual position of the shaft as well as the before mentioned parameters; the shaft is moved according to the velocity profile until the target position is reached.

Typical applications where accurate positioning is required:

- Palletizers, for example stacking boxes on a pallet.
- Index tables, for example filling material into trays on a rotating table.
- Conveyors, for example when cutting material to length.
- Hoists, for example a lift stopping at different levels.

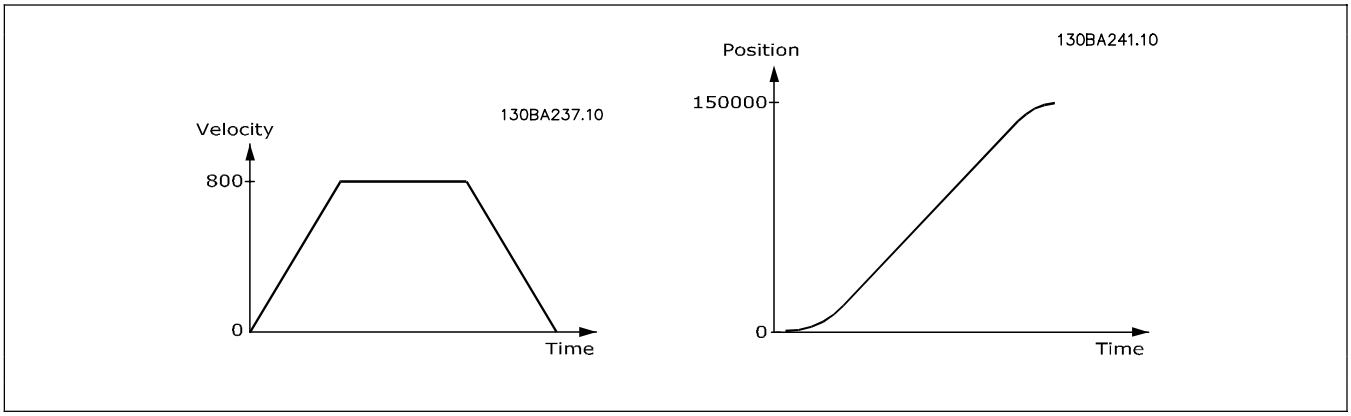
MCO 305 offers three main positioning types

- Absolute
- Relative
- Touch Probe

### Absolute Positioning

Absolute positioning always relates to the absolute zero point of the system, this also means that the absolute zero point must be defined before an absolute positioning procedure can be conducted. When using incremental encoders the zero point is defined by means of a Home function, where the drive approaches a reference switch, stops and defines the actual position as zero. When using absolute encoders the zero point is given by the encoder.

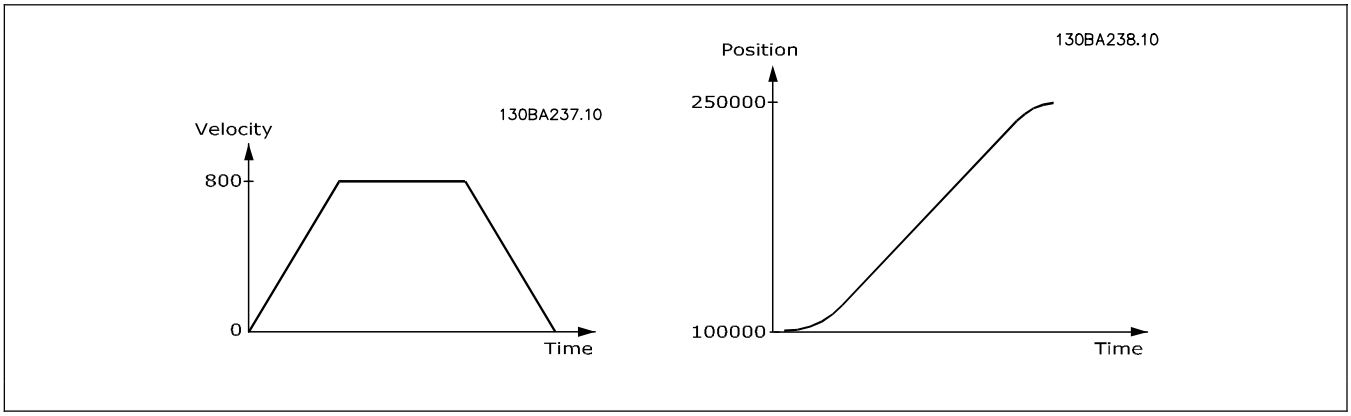
If the starting position is 0 and with an absolute positioning to 150.000 the target position is 150.000, the drive will thus move a distance of 150.000. If on the other hand the starting position is 100.000 and with an absolute positioning to 150.000 the target position is still 150.000 but the drive will only move a distance of 50.000 because it moves to position 150.000 related to the zero point.




**Relative Positioning**

Relative positioning is always relating to the actual position, it is therefore possible to execute a positioning procedure without defining the absolute zero point.

If the starting position is 100.000, with a relative positioning to 150.000 the target position is 250.000 (100.000 + 150.000), the moving distance is thus 150.000.

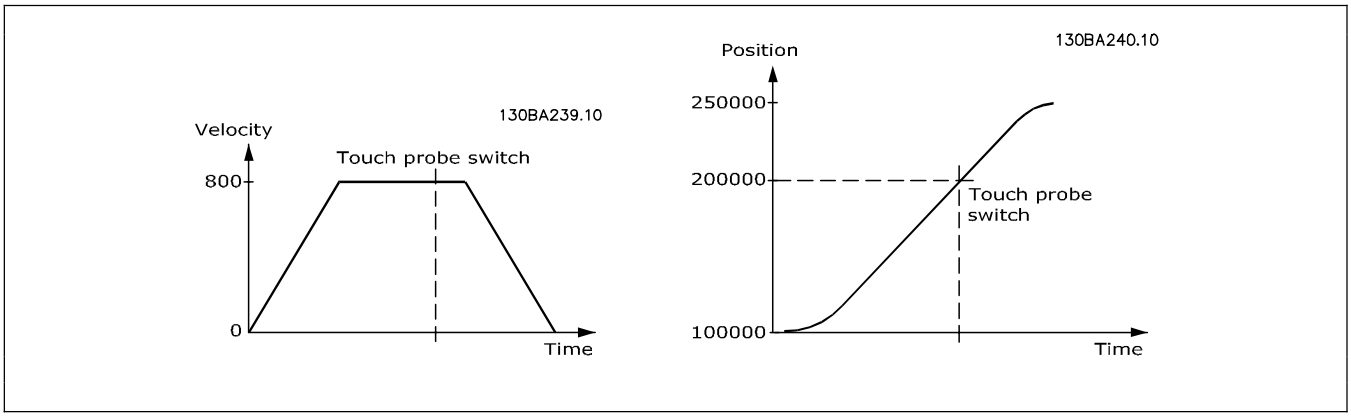



**Touch Probe Positioning**

With touch probe positioning, the positioning is related to the actual position when the touch probe input is activated, that means the target position is the position of the touch probe + the positioning distance. Touch probe positioning is thus relative positioning relating to a touch probe instead of the actual starting position.

The touch probe is a sensing device; it can be a mechanical switch, a proximity sensor, an optical sensor or the like. Once the touch probe is activated, for example by a box moved on a conveyor belt, the reference for the positioning is set.

With touch probe positioning to position 50.000 the drive is running until the touch probe is activated for example at position 200.000 and it continues to a target position of 250.000 (200.000 + 50.000). Touch probe positioning is also called “marker related positioning”.

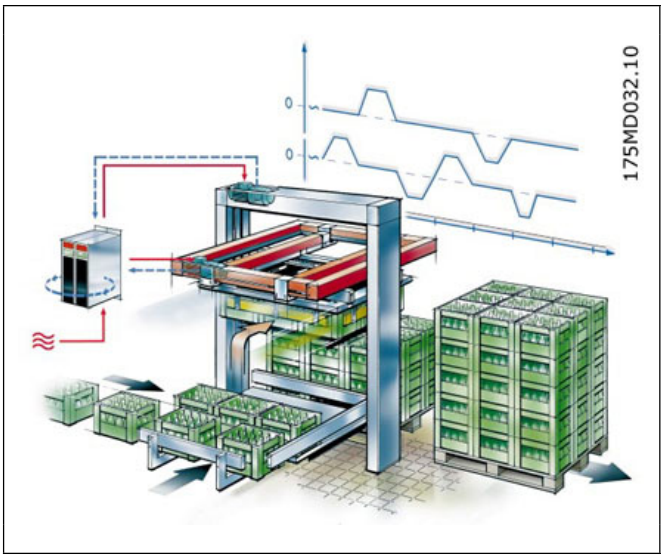


## Application Example: A Bottle Box Palletizer

The following samples show a palletizer stacking boxes with bottles. The boxes are unloaded using a pack gripper. The three positioning modes are used in this sample and explained in three steps.

NOTE: The following are just examples and the presented settings and programs might not cover the complete functionality required by a real application.

It is assumed that the motor and encoder connections are checked and that all basic parameter settings such as motor data, encoder data and PID controller are done. Instructions for setting up parameters can be found in FC 300 Operating Instructions and MCO 305 Operating Instructions.




## Absolute Positioning

Absolute Positioning is explained with following function of the palletizer: The horizontal axis has two fixed target positions; one is above the pick-up and the other one is above the pallet. The horizontal axis is controlled with absolute positioning between the pick-up position and the deliver position.

### Parameter Settings and Commands for Palletizer Application

The following MCO 305 parameters are relevant for absolute positioning:



| Command | Description | Syntax | Parameter |
|---|---|---|---|
| **Absolute Positioning (ABS)** | | | |
| ACC | Sets acceleration | ACC a | a = acceleration |
| DEC | Sets deceleration | DEC a | a = deceleration |
| HOME | Move to device zero point (reference switch) and set as the real zero point. | HOME | – |
| POSA | Positions axis absolutely | POSA p | p = position in UU |
| VEL | Sets velocity for relative and absolute motions and set maximum allowed velocity for synchronizing | VEL v | v = scaled velocity value |

### ☐ Program Example: Absolute Positioning for Palletizer Application

```
/********************** Absolute Positioning program sample **************************/
//   Inputs:   1       Go to pick up position
//             2       Goto deliver position
//             3       Home switch
//             8       Clear Error
//  Outputs:   1       In pick up position
//             2       In deliver position
//             8       Error
/******************************** Interrupts ***********************************/
ON ERROR GOSUB errhandle
      // In case of error go to error handler routine, this must always be included
/*****************************  Basic settings  *********************************/
VEL 80          // Sets positioning velocity related to par. 32-80 Maximum velocity
ACC 100         // Sets positioning acceleration related to par. 32-81 Shortest ramp
DEC 100         // Sets positioning deceleration related to par. 32-81 Shortest ramp
/********************** Define application parameters ****************************/
LINKGPAR 1900 "Pick up Position" 0 1073741823 0
LINKGPAR 1901 "Deliver Position" 0 1073741823 0
/******************  Define Home(0) position after power up  *********************/
SET I_FUNCTION_3 1      // Define input 3 as Home switch input
HOME                    // Go to Home and set position to 0
/******************************** main loop ************************************/
MAIN:
IF (IN 1 == 1) AND (IN 2 == 0) THEN       // Go to pick up position when only input 1 is high
   OUT 2 0              // Reset "In deliver position" output
   POSA (GET 1900)      // Go to position
   OUT 1 1              // Set "In pick up position" output
ELSEIF (IN 1 == 0) AND (IN 2 == 1) THEN       // Go to deliver position when only input 2 is high
   OUT 1 0              // Reset "In pick up position" output
   POSA (GET 1901)      // Go to position
   OUT 2 1              // Set "In deliver position" output
ELSE
   MOTOR STOP           // Stop if both inputs are low or high.
ENDIF
GOTO MAIN
/********************** Sub programs start ***************************************/
SUBMAINPROG
/************************ Error handler ******************************************/
SUBPROG errhandle
   err = 1              // Set error flag to remain in error handler until error is reset.
   OUT 8 1              // Set error output
   WHILE err DO         // Remain in error handler until the reset is received
   IF IN 8 THEN         // Reset error when Input 8 is high.
      ERRCLR            // Clear error
      err=0             // Reset error flag
   ENDIF
   ENDWHILE
   OUT 8 0              // Reset error output
RETURN
/*********************************************************************************/
ENDPROG
/*************************  End of program ***************************************/
```

## □ Relative Positioning

Relative Positioning is explained with following function of the palletizer: When leaving the deliver position the vertical axis just needs to move up one box height so that it is clear of the stack before the horizontal axis can move back to the pick-up position. This is done by relative positioning to "box height" in the "up-direction".

### □ Parameter Settings and Commands for Palletizer Application (Relative Positioning)

The following MCO 305 parameters are relevant for relative positioning:


| | | |
|---|---|---|
| 32-0* | Encoder 2 – Slave | page 183 |
| 32-6* | PID-Controller | page 189 |
| 32-8* | Velocity & Acceleration | page 191 |


| Command | Description | Syntax | Parameter |
|---|---|---|---|
| **Relative Positioning (REL)** | | | |
| ACC | Sets acceleration | ACC a | a = acceleration |
| DEC | Sets deceleration | DEC a | a = deceleration |
| POSR | Positioning relative to the actual position | POSR d | d = distance to actual position in UU |
| VEL | Sets velocity | VEL v | v = scaled velocity value |

### □ Program Example: Relative Positioning for Palletizer Application

```
/********** Relative positioning sample program for palletizer application example **********/
//          Inputs:       1     Go to position
//                        8     Clear Error
//          Outputs:      1     In position
//                        8     Error
/***************************** Interrupts ****************************************/
ON ERROR GOSUB errhandle
// In case of error go to error handler routine, this must always be included
/************************* Define flags ****************************************/
flag = 0
/**********************    Basic settings    ***********************************/
VEL 80        // Sets positioning velocity related to par. 32-80 Maximum velocity.
ACC 100       // Sets positioning acceleration related to par. 32-81 Shortest ramp.
DEC 100       // Sets positioning deceleration related to par. 32-81 Shortest ramp.
/******************* Define application parameters ********************************/
LINKGPAR 1900 "Box high" 0 1073741823 0
/**************************  main loop ****************************************/
MAIN:
IF (IN 1 == 1) AND (flag == 0) THEN     // Go to position once (ensured by flag) when input 1 is high.
   OUT 1 0                  // Reset "In position" output.
   POSR (GET 1900)          // Go to position.
   OUT 1 1                  // Set "In position" output.
   flag = 1                 // Set "flag" to ensure that distance is only traveled once.
ELSE
   MOTOR STOP // Stop if input is low.
   flag = 0                 // Reset "flag" to enable new positioning.
ENDIF
GOTO MAIN
/**********************  Sub programs start  ***********************************/
SUBMAINPROG
/************************ Error handler ****************************************/
```

```
SUBPROG errhandle
err = 1                 // Set error flag to remain in error handler until error is reset.
   OUT 8 1              // Set error output.
   WHILE err DO         // Remain in error handler until the reset is received.
      IF IN 8 THEN      // Reset error when Input 8 is high.
         ERRCLR         // Clear error.
         err=0          // Reset error flag.
      ENDIF
   ENDWHILE
   OUT 8 0              // Reset error output.
   flag = 0             // Reset "flag" to enable new positioning.
RETURN
/*****************************************************************************/
ENDPROG
/********************  End of program  ***********************************/
```

## □ Touch Probe Positioning

Touch Probe is explained with following function of the palletizer:

When the horizontal axis is in the deliver position the vertical axis has numerous target positions depending on the height of the already stacked boxes, which again depends on the box height and the number of layers of boxes. This is controlled with Touch probe positioning where the touch probe detects the top of the stack in order to calculate the deliver position on top of the stack.

### □ Parameter Settings and Commands for Touch Probe Application

The following MCO 305 parameters are relevant for touch probe positioning:


| | | |
|---|---|---|
| 32-0* | Encoder 2 – Slave | page 183 |
| 32-6* | PID-Controller | page 189 |
| 32-8* | Velocity & Acceleration | page 191 |
| 33-4* | Limit Handling | page 204 |


| Command | Description | Syntax | Parameter |
|---|---|---|---|
| **Touch Probe** | | | |
| ON INT | Defining an interrupt input. | ON INT n GOSUB name | n = number of the input to be monitored<br>1 - 8 = reaction to the rising edge<br>–1 - 8 = reaction to the falling edge<br>name = subroutine name |
| ACC | Sets acceleration. | ACC a | a = acceleration |
| DEC | Sets deceleration. | DEC a | a = deceleration |
| POSR | Positioning relative to the actual position. | POSR d | d = distance to actual position in UU |
| CVEL | Sets velocity for speed controlled motor movements. | CVEL v | v = velocity value (negative value for reversing) |
| CSTART | Starts the speed mode. | – | – |

### □ Program Example: Touch Probe Positioning for Palletizer Application

```
/******** Touch probe positioning sample program for palletizer application example *********/
//      Inputs:     1     Go to position
//                  2     Touch probe
//                  8     Clear Error
//      Outputs:    1     In position
//                  8     Error
/********************************** Interrupts ***********************************/
ON ERROR GOSUB errhandle       // In case of error go to error handler routine, this must always be included
ON INT 2 GOSUB tp_handler      // Call touch probe handler on positive edge of input 2.
/*********************************  Define flags  ********************************/
flag = 0
tp_active = 0
/*****************************   Basic settings    ********************************/
VEL 80       // Sets positioning velocity related to parameter 32-80 Maximum velocity.
ACC 100      // Sets positioning acceleration related to parameter 32-81 Shortest ramp.
DEC 100      // Sets positioning deceleration related to parameter 32-81 Shortest ramp.
/************************** Define application parameters **************************/
LINKGPAR 1900 "Touch probe distance" 0 1073741823 0
/********************************* main loop ***************************************/
MAIN:
IF (IN 1 == 1) AND (flag == 0) THEN        // Start movement once (ensured by flag) when input 1 is high.
   OUT 1 0                  // Reset "In position" output.
   CVEL 80                  // Set constant velocity.
   CSTART                   // Start with constant velocity.
   tp_active = 0            // Reset "tp_active" to enable new touch probe positioning.
   flag = 1                 // Set "flag" to ensure that distance is only traveled once.
ELSE
   MOTOR STOP               // Stop if input is low.
   flag = 0                 // Reset "flag" to enable new start.
ENDIF
GOTO MAIN
/******************************** Sub programs start ********************************/
SUBMAINPROG
/******************************** Touch probe handler *******************************/
SUBPROG tp_handler
   IF (tp_active == 0) THEN
      POSR (GET 1900)    // Go to touch probe target position
      WAITAX             // Halt program execution until position is reached (This is necessary as
                         // NOWAIT ON is automatically set in a subroutine called by interrupt).
      OUT 1 1            // Set "In position" output.
      tp_active = 1      // Set "tp_active" to ensure that touch probe positioning is done only once
   ENDIF
RETURN
/********************************** Error handler *********************************/
SUBPROG errhandle
   err = 1                  // Set error flag to remain in error handler until error is reset.
   OUT 8 1                  // Set error output.
   WHILE err DO             // Remain in error handler until the reset is received.
      IF IN 8 THEN          // Reset error when Input 8 is high.
         ERRCLR             // Clear error.
         err=0              // Reset error flag.
      ENDIF
   ENDWHILE
   OUT 8 0                  // Reset error output.
   flag = 0                 // Reset "flag" to enable new positioning.
RETURN
/***************************************************************************************/
ENDPROG
/******************************** End of program ***********************************/
```

## □ Synchronizing

Synchronizing is used in applications where 2 or more shafts need to follow each other in velocity or position. It can be a simple master-slave system where a slave is following the velocity or position of a master or it can be a multi-axis system where multiple slaves are following the velocity or position of a common master signal. Electronic synchronization is very flexible compared to a mechanical shaft, belt or chain as the gear-ratio and position offset can be adjusted during operation. Velocity and position of the slave drive is controlled based on a master encoder signal, a feedback encoder signal as well as the set gear-ratio.

During synchronization the slave is always restricted by maximum velocity and acceleration/ deceleration (parameter group 33-8*). In addition the allowed deviation between master and slave velocity can be restricted by parameter 33-14, e.g. 33-14 = 5% means that the slave can only be 5% faster or slower than actual master velocity when doing position corrections.

MCO 305 offers 3 basic types of synchronization:

For synchronous operation of two or more drives you can use

- Velocity synchronization
- Position synchronization
- Marker synchronization

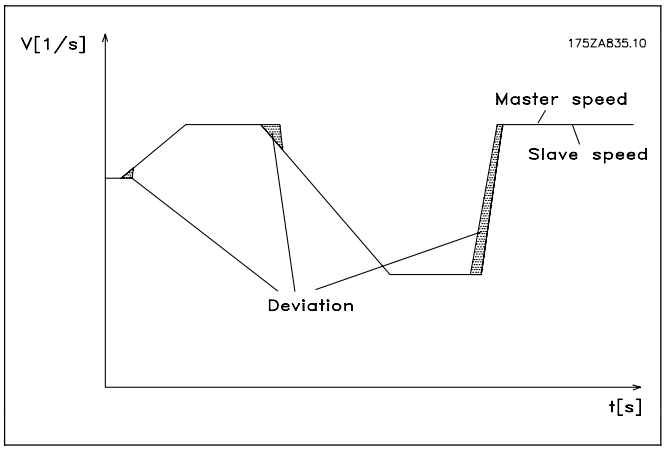
## □ Velocity Synchronization (SYNCV)

Velocity synchronization (SYNCV) is closed loop velocity control where the set-point is the master velocity multiplied by the gear-ratio and the actual velocity is measured by the slave encoder; position deviations will not be corrected. Note however that using the Integral part of the PID controller will lead to some level of position correction as the integral sum of velocity is equal to position.

The slave must be at least as fast and dynamic as the master in order to maintain accurate synchronization, i.e. the slave must be able to match the maximum velocity, acceleration and deceleration of the master. Already during the design phase it is thus important to consider making the least dynamic shaft the master as this shaft will anyway be the limiting factor for the system performance.

Typical applications are:

- Synchronizing of two or more conveyors
- Material stretching
- Mixing

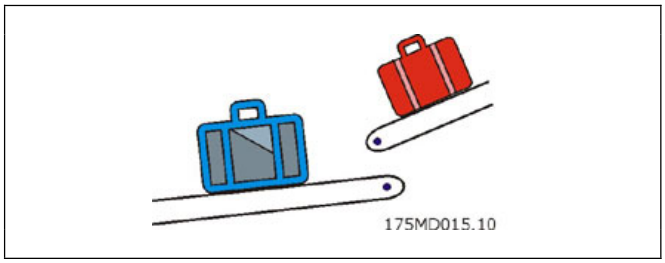



**Control behavior with velocity synchronization.**

## □ Application Example: Suit Case Conveyor Belt

Two or more conveyor belts must run at the same speed in order to get a smooth transfer of the suit case from one conveyor belt to the next.

In addition to start and stop of velocity synchronizing the sample program has a manual mode with the possibility to increase and decrease the velocity via digital inputs.




NOTE: The following is just an example and the presented settings and programs might not cover the complete functionality required by a real application.

It is assumed that the motor and encoder connections are checked and that all basic parameter settings such as motor data, encoder data and PID controller are done. Instructions for setting up parameters can be found in FC 300 Operating Instructions and MCO 305 Operating Instructions.

### ☐ Parameter Settings and Commands for Conveyor Belt Application

The following MCO 305 parameters are relevant for velocity synchronization:


| | | |
|---|---|---|
| 32-0* | Encoder 2 – Slave | page 183 |
| 32-3* | Encoder 1 – Master | page 186 |
| 32-6* | PID-Controller | page 189 |
| 32-8* | Velocity & Acceleration | page 191 |
| 33-1* | Synchronization | page 194 |


| Command | Description | Syntax | Parameter |
|---|---|---|---|
| SYNCV | Synchronization of velocity | SYNCV | – |
| ON ERROR GOSUB | Definition of an error subroutine | ON ERROR GOSUB name | name = name of the subroutine |

### ☐ Program Example: Velocity Synchronizing

```
/************************ Velocity synchronizing sample program ***********************/
// Inputs:      1      Start/stop synchronization
//              2      Start manual mode
//              3      Increase manual velocity
//              4      Decrease manual velocity
//              8      Clear Error
// Outputs:     1      In synchronizing mode
//              2      In manual mode
//              8      Error
/********************************** Interrupts ************************************/
ON ERROR GOSUB errhandle       // In case of error go to error handler routine, this must always be included
/*****************************   Basic settings  **********************************/
VEL 100                 // Sets maximum slave velocity related to parameter 32-80 Maximum velocity
ACC 100                 // Sets slave acceleration related to parameter 32-81 Shortest ramp
DEC 100                 // Sets slave deceleration related to parameter 32-81 Shortest ramp
/************************** Define application parameters ***************************/
LINKGPAR 1900 "Manual velocity" 0 100 0
LINKGPAR 1901 "Velocity step" 0 10 0
/************************** Define flags and variables ******************************/
sync_flag = 0
done = 0
err = 0
man_vel = 0
/********************************** main loop *************************************/
MAIN:
IF (IN 1 == 1) AND (sync_flag == 0) THEN        // Start synchronizing once when input 1 is high
   SYNCV                    // Start velocity synchronizing mode
   sync_flag = 1            // Set sync_flag to ensure synchronizing is only started once.
   OUT 1 1                  // Set "In synchronizing mode" output
ELSE
   MOTOR STOP               // Stop if input 1 is low.
   sync_flag = 0            // Reset sync_flag after stop
   OUT 1 0                  // Reset "In synchronizing mode" output
ENDIF
IF (IN 2 == 1) AND (sync_flag == 0) THEN        // Start manual mode if input 2 high and not synchronizing
   OUT 2 1                  // Set "In manual mode" output
   man_vel = GET 1900       // Set manual velocity to parameter 1900
   CVEL man_vel
   CSTART                           // Start constant velocity mode
   WHILE (IN 2 == 1) DO             // Stay in manual mode while input 2 is high
      CVEL man_vel                  // Update manual velocity
      IF (IN 3 == 1) AND (done == 0) THEN       // Increase manual velocity by one step when input 3 is set
```

```
        man_vel = man_vel + GET 1901
        done = 1
    ELSEIF (IN 4 == 1) AND (done == 0) THEN      // Decrease manual velocity by 1 step when input 3 is set
        man_vel = man_vel - GET 1901
        done = 1
    ELSE
        done = 0
    ENDIF
  ENDWHILE
  CSTOP                  // Stop when leaving manual mode
  OUT 2 0                // Reset "In manual mode" output when leaving manual mode
ENDIF
GOTO MAIN
/****************************** Sub programs start *********************************/
SUBMAINPROG
/******************************** Error handler ***********************************/
SUBPROG errhandle
  err = 1                // Set error flag to remain in error handler until error is reset.
  OUT 8 1                // Set error output
  OUT 1 0                // Reset "In synchronizing mode" output in case of error
  OUT 2 0                // Reset "In manual mode" output in case of error
  WHILE err DO           // Remain in error handler until the reset is received
    IF (IN 8) AND NOT (IN 1) AND NOT (IN 2) THEN
                         // Reset error when Input 8 is high and input 1+2 is low.
      ERRCLR             // Clear error
      err=0              // Reset error flag
    ENDIF
  ENDWHILE
  OUT 8 0                // Reset error output
  sync_flag = 0          // Reset sync_flag after an error
  done = 0               // Reset done flag after an error
RETURN
/*******************************************************************************/
ENDPROG
/****************************** End of program *********************************/
```

## Position/Angle Synchronization (SYNCP)

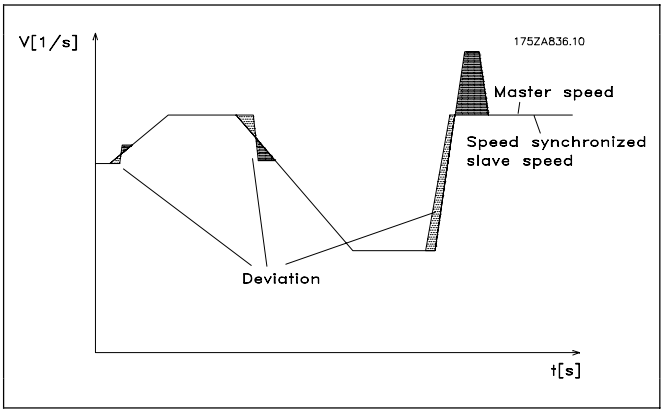
Position synchronization (SYNCP) is closed loop position control with a moving target where the set-point (commanded position) is the master position multiplied by the gear-ratio also considering any position off-set. The slave position is controlled based on this set-point and the actual position feedback from the slave encoder. Any position deviation will continuously be corrected considering maximum velocity, acceleration and deceleration of the slave. The gear-ratio is set as a fraction (numerator and denominator) to avoid rounding errors e.g. when using prime numbers. The gear-ratio must be 100% correct, even the smallest rounding error will lead to position drifting over time.

When starting position synchronizing the actual slave position will be locked to the actual master position, it is thus necessary to bring the slave into the right physical position with respect to the physical position of the master. This can be done manually or be means of an automatic homing procedure (requires external reference switches or absolute encoders).

The slave must be faster and more dynamic than the master in order to maintain accurate synchronization both at maximum master velocity and during acceleration/deceleration. I.e. the slave must be able to exceed the maximum velocity, acceleration and deceleration of the master to allow it to catch up if getting behind the master. Already during the design phase it is thus important to consider making the least dynamic shaft the master as this shaft will anyway be the limiting factor for the system performance.

Typical applications are:

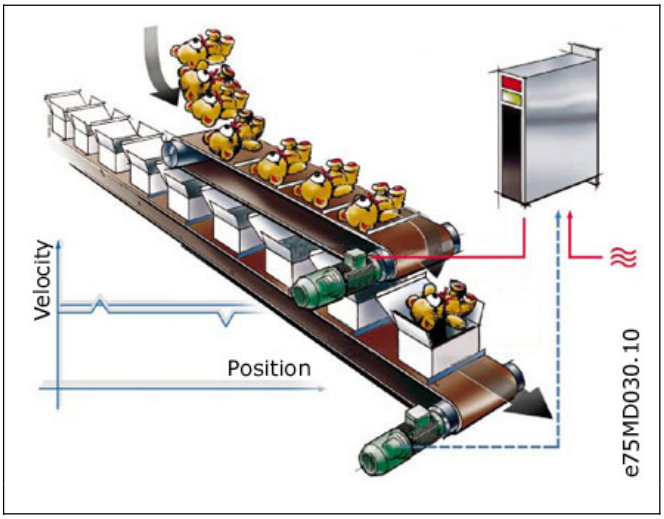
- Bottle washing machines.
- Foil wrapping.
- Packaging machines.
- Conveyors.
- Multi axis hoists.
- Filling.
- Printing.
- Cut on the fly.




**Control behavior with position synchronization**

## □ Application Example: Packaging with Fixed Product Distances

This application consists of two conveyors, 1 carrying empty boxes another one carrying teddy bears; the purpose of the machine is to put teddies into the boxes. Both boxes and teddies come with a fixed distance and it is ensured that there is no slip between the encoders and the boxes and teddies. It is thus sufficient to position synchronize based on the encoders. At start it is ensured that the master (box conveyor) it always in the same position, the teddy conveyor needs to be homed prior to starting synchronization. There are 3 ways to ensure that the teddies are aligned with the boxes when starting:




- Adjust the physical position of the home/reference switch.
- Adjust home offset in parameter 33-01.
- Adjust position offset for synchronization in parameter 33-12.

NOTE: The following is just an example and the presented settings and programs might not cover the complete functionality required by a real application.

It is assumed that the motor and encoder connections are checked and that all basic parameter settings such as motor data, encoder data and PID controller are done. Instructions for setting up parameters can be found in FC 300 Operating Instructions and MCO 305 Operating Instructions.

### □ Parameter Settings and Commands for Packaging Application

The following MCO 305 parameters are relevant for position synchronization:

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| DEF SYNCORIGIN | Defines master-slave relation for the next SYNCP or SYNCM command | DEFSYNCORIGIN master slave | master = reference position in qc<br>slave = reference position |
| MOVESYNCORIGIN | Relative shifting of the origin of synchronization | MOVESYNCORIGIN mvalue | mvalue = Relative offset |
| PULSACC | Sets acceleration for master simulation | PULSACC a | a = acceleration in Hz/s |
| PULSVEL | Sets velocity for master simulation | PULSVEL v | v = velocity in pulses per second (Hz) |
| SYNCP | Synchronization of angle/position | SYNCP | – |
| SYNCSTAT | Queries flag for synchronization status | res = SYNCSTAT | – |
| SYNCERR | Queries actual synchronization error of the slave | res = SYNCERR | – |

### ☐ Position Synchronizing Sample Program

```
/************************** Position Synchronizing Sample Program ***********************/
// Inputs:   1      Start/stop synchronization
//           2      Start homing
//           3      Home switch
//           4      Increase offset
//           5      Decrease offset
//           8      Clear Error
// Outputs:  1      Within synchronizing accuracy, set accuracy window in par. 33-13
//           2      Homing done
//           8      Error
/****************************** Interrupts ************************************/
ON ERROR GOSUB errhandle
   // In case of error go to error handler routine, this must always be included
/**************************   Basic settings   ***********************************/
VEL 100             // Sets maximum slave velocity related to par. 32-80 Maximum velocity
ACC 100             // Sets slave acceleration related to par. 32-81 Shortest ramp
DEC 100             // Sets slave deceleration related to par. 32-81 Shortest ramp
/********************** Define application parameters ******************************/
LINKGPAR 1900 "Offset step" 0 10000 0
LINKGPAR 1901 "Offset type" 0 1 0
/********************** Set parameters and flags  *********************************/
SET I_FUNCTION_3 1              // Define input 3 as Home switch input
next_step = 0
home_done = 0
new_offset = 0
/************************************ main loop ***********************************/
MAIN:
IF (IN 2 == 1) THEN             // Start homing if input 2 high
   HOME                         // Go to Home and set position to 0
   home_done = 1                // Set home_done flag
   OUT 2 1                      // Set home done output
ENDIF
IF (IN 1 == 1) AND  (home_done == 1) THEN       // Start synchronizing when input 1 = 1 and homing done.
```

```
   SYNCP                                        // Start position synchronizing mode
   old_offset = GET SYNCPOSOFFS
   WHILE (IN 1 == 1) DO                         // Stay in synchronizing mode while input 1 = 1
      IF (IN 4 == 1) THEN
         GOSUB increase_offset
      ELSEIF (IN 5 == 1) THEN
         GOSUB decrease_offset
      ENDIF
      IF (SYNCSTAT & 4) THEN
         OUT 1 1
      ELSE
         OUT 1 0
      ENDIF
   ENDWHILE
   MOTOR STOP                   // Stop if input 1 is low.
   home_done = 0                // Reset home_done flag after stop
   OUT 1 0
   OUT 2 0                      // Reset home done output after stop
   IF (new_offset != old_offset) AND (GET 132 == 0) THEN       // Save absolute offset if changed
      SAVE AXPARS               // NOTE: Saving more that 10000 times can damage flash PROM
   ENDIF
ENDIF
GOTO MAIN
/***************************** Sub programs start *********************************/
SUBMAINPROG
/****************************** Increase offset **********************************/
SUBPROG increase_offset
   IF (Next_step) THEN                  // Check if next offset step is enabled
      IF (GET 1901 == 0) THEN           // Absolute offset
         new_offset = old_offset + GET 1900      // Read existing offset and add step value
         SET SYNCPOSOFFS new_offset              // Set new position offset
      ELSE                                       // Relative offset
         MOVESYNCORIGIN GET 1900                 // Execute relative offset with offset step
      ENDIF
   ENDIF
   Next_step=0                          // Disable next offset step
   ON TIME 500 GOSUB Enb_Step           // Enable next offset step after 500 ms
RETURN
/****************************** Decrease offset **********************************/
SUBPROG decrease_offset
   IF (Next_step) THEN                  // Check if next offset step is enabled
      IF (GET 1901 == 0) THEN           // Absolute offset
         new_offset = GET SYNCPOSOFFS - GET 1900
         // Read existing offset and subtract step value
         SET SYNCPOSOFFS new_offset              // Set new position offset
      ELSE                                       // Relative offset
         MOVESYNCORIGIN (- GET 1900)             // Execute relative offset with - offset step
      ENDIF
   ENDIF
   Next_step=0                          // Disable next offset step
   ON TIME 500 GOSUB Enb_Step           // Enable next offset step after 500 ms
RETURN
/************************  Enable new offset step ******************************/
```

```
SUBPROG Enb_step
   Next_step = 1          // Enable next offset step
RETURN
/***************************** Error handler ***********************************/
SUBPROG errhandle
   err = 1                // Set error flag to remain in error handler until error is reset.
   OUT 8 1                // Set error output
   OUT 2 0                // Reset home done output in case of error
   WHILE err DO                          // Remain in error handler until the reset is received
      IF (IN 8) AND NOT (IN 1) THEN      // Reset error when Input 8 is high and input 1 low.
         ERRCLR                          // Clear error
         err=0                           // Reset error flag
      ENDIF
   ENDWHILE
   OUT 8 0                // Reset error output
   home_done = 0          // Reset home_done flag after an error
RETURN
/*******************************************************************************/
ENDPROG
/***************************** End of program **********************************/
```

## □ Marker Synchronization (SYNCM)

Marker synchronization (SYNCM) is extended position synchronizing where an additional position correction is done to align a slave marker with a master marker. Master and slave marker signals can be the encoder zero pulse or external sensors connected to digital inputs. As with position synchronizing it is possible to adjust gear-ratio and offset, in addition a marker ratio can be set e.g. 1 master marker to 3 slave markers which mean that every master marker will be aligned with every 3rd slave marker.

The marker signals can be monitored by defining a position window, only one marker (the first one) will be accepted within the marker window and any marker signal outside the marker window will be ignored. Without marker windows every marker signal including noise pulses and jitter will be accepted and used to correct the slave position. The first master marker and first slave marker after starting are not monitored as the system does not know where the first marker is supposed to be. After detecting the first marker the anticipated position of the following markers is known as the marker distance must be specified by parameter individually for master and slave.

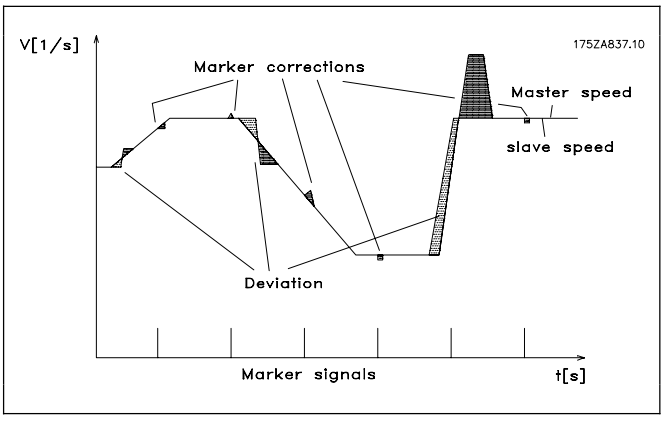
When starting marker synchronization the initial behavior will be like position synchronization but when the first set of markers has been detected marker correction will start. Which markers to use for the first marker correction can be defined by parameter 33-23, by defining the start behavior it is possible to define whether the slave must always wait for the master, catch up with the master or make the smallest correction, see parameter 33-23 for a detailed description of the available options. Homing procedures are thus not necessary prior to starting as the marker correcting will automatically align the slave with the master.

The slave must be faster and more dynamic than the master in order to make marker correction and maintain accurate synchronization both at maximum master velocity and during acceleration/deceleration. I.e. the slave must be able to exceed the maximum velocity, acceleration and deceleration of the master to allow it to make marker correction and to catch up if getting behind the master. Already during the design phase it is thus important to consider making the least dynamic shaft the master as this shaft will anyway be the limiting factor for the system performance.

Typical applications are:

Basically the same application types as position synchronizing but where one or more of the following is true:

- Automatic alignment after start is required.
- Gear-ratio cannot be set 100% correct.
- There is slip somewhere between the encoder and the part the must be synchronized.
- Varying distance between products.




**Control behavior with marker synchronization**

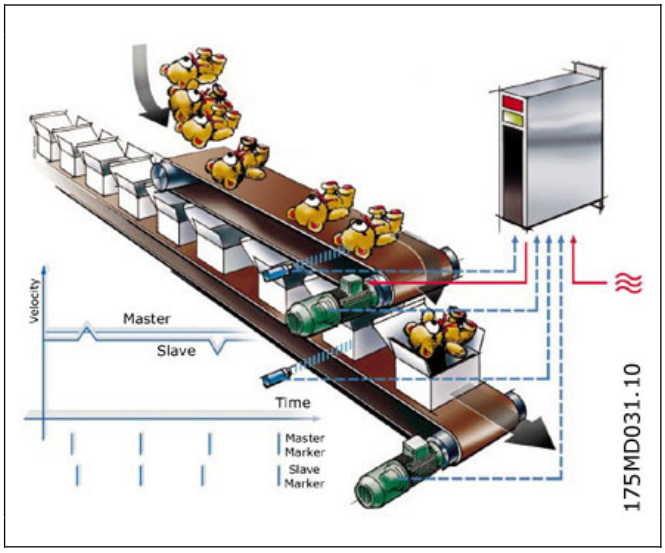
## □ Application Example: Packaging with Varying Product Distance and Slip

This application consists of two conveyors, 1 carrying empty boxes and another one carrying teddy bears, the purpose of the machine is to put teddies into the boxes. Both boxes and teddies are moved by friction and can thus move on the conveyor belt. It means that there is no fixed relationship between the encoders and the position of box and teddy bear and the distance can vary. It is therefore necessary to use external marker detection on both boxes (master) and teddies (slave) in order to synchronize the teddy bear position to the box position. Alignment can be obtained by adjusting the physical position of the marker detectors or by adjusting the position offset in parameter 33-12.

In addition to start and stop of marker synchronizing the sample program has measuring of both *Master* and *Slave Marker Distance*. The average distance between the detected markers is calculated and the marker distance parameters (33-17 and 33-18) are automatically set.

NOTE: The following is just an example and the presented settings and programs might not cover the complete functionality required by a real application.

It is assumed that the motor and encoder connections are checked and that all basic parameter settings such as motor data, encoder data and PID controller are done. Instructions for setting up parameters can be found in FC 300 Operating Instructions and MCO 305 Operating Instructions.




## □ Parameter Settings and Commands for Packaging Application

The following MCO 305 parameters are relevant for marker synchronization:



| Command | Description | Syntax | Parameter |
|---|---|---|---|
| DEF SYNCORIGIN | Defines master-slave relation for the next SYNCP or SYNCM command. | DEFSYNCORIGIN master slave | master = reference position in qc<br>slave = reference position |
| MOVE SYNCORIGIN | Relative shifting of the origin of synchronization. | MOVESYNCORIGIN mvalue | mvalue = Relative offset |
| PULSACC | Sets acceleration for master simulation. | PULSACC a | a = acceleration in Hz/s |
| PULSVEL | Sets velocity for master simulation. | PULSVEL v | v = velocity in pulses per second (Hz) |
| SYNCM | Synchronization of angle/position with marker correction. | SYNCM | – |
| SYNCSTAT | Queries flag for synchronization status. | res = SYNCSTAT | |
| SYNCERR | Queries actual synchronization error of the slave. | res = SYNCERR | – |
| IPOS | Queries last index or marker position of the slave. | res = IPOS | – |
| MIPOS | Queries last index or marker position of the master. | res = MIPOS | – |

### □ Program Example: Marker Synchronization

```
/********************** Marker synchronizing sample program ************************/
// Inputs:      1       Start/stop synchronization
//              2       Measure slave marker distance
//              3       Measure master marker distance
//              5       Master marker
//              6       Slave marker
//              8       Clear Error
// Outputs:     1       Within synchronizing accuracy, set accuracy window in parameter 33-13
//              2       Marker measurement active.
//              8       Error
***************************** Interrupts ***************************************/
ON ERROR GOSUB errhandle
   // In case of error go to error handler routine, this must always be included
/*******************************    Basic settings    ********************************/
VEL 100          // Sets maximum slave velocity related to parameter 32-80 Maximum velocity
ACC 100          // Sets slave acceleration related to parameter 32-81 Shortest ramp
DEC 100          // Sets slave deceleration related to parameter 32-81 Shortest ramp
/************************ Define application parameters ****************************/
LINKGPAR 1900 "Measuring velocity" 0 100 0
/************************* Set parameters and flags  ******************************/
SET SYNCMTYPM 2            // Set master marker type to external
SET SYNCMTYPS 2            // Set slave marker type to external
sync_flag = 0
/****************************** main loop ***************************************/
MAIN:
IF (IN 1 == 1) AND (sync_flag == 0) THEN      // Start synchronizing once when input 1 is high.
   SYNCM                                      // Start marker synchronizing mode
   sync_flag = 1                              // done flag
ELSE
   MOTOR STOP                          // Stop when input 1 is low.
   sync_flag = 0                       // Reset sync_flag after stop.
ENDIF
IF (IN 2 == 1) AND (sync_flag == 0) THEN          // Start measuring slave marker distance
   GOSUB slave_measure                            // Note: Slave motor will rotate
ELSEIF (IN 3 == 1) AND (sync_flag == 0) THEN      // Start measuring slave marker distance,
   GOSUB master_measure                           // master must run.
ENDIF
GOTO MAIN
/************************** Sub programs start ********************************/
SUBMAINPROG
/********************** Measure slave marker distance ***************************/
SUBPROG slave_measure
   OUT 2 1                                 // Set "Marker measurement active" output
   CVEL GET 1900                           // Set measuring velocity
   CSTART                                  // Start constant velocity mode
   old_ipos = IPOS                         // Read "old" marker position
   marker_number = 0                       // Reset variable
   total_dist = 0                          // Reset variable
   skip_first = 0                          // Reset variable
   WHILE (IN 2 == 1) DO                    // Stay in measuring mode while input 2 high
      new_ipos = IPOS                      // Read "new" marker position
      IF (new_ipos != old_ipos) THEN                // Check if a new marker was detected
         marker_distance = new_ipos - old_ipos      // Calculate marker distance
         IF (marker_distance < 0) THEN// Change sign if negative
            marker_distance = (marker_distance * -1)
         ENDIF
```

```
        IF (skip_first == 0) THEN       // Do not use first value as it might be invalid
            skip_first = 1
        ELSE
            marker_number = marker_number + 1   // Increment counter
            total_dist = total_dist + marker_distance // Summarize marker distances
        ENDIF
        old_ipos = new_ipos             // Set "old" marker position to "new" marker position
    ENDIF
  ENDWHILE
  CSTOP                                 // Stop when leaving slave marker measurement
  SET SYNCMPULSS (total_dist rnd marker_number)       // calculate average marker distance and set par.
  OUT 2 0                               // Reset "Marker measurement active" output
RETURN
/********************** Measure master marker distance ***************************/
SUBPROG master_measure
  OUT 2 1                               // Set "Marker measurement active" output
  old_mipos = MIPOS                     // Read "old" marker position
  marker_number = 0                     // Reset variable
  total_dist = 0                        // Reset variable
  skip_first = 0                        // Reset variable
  WHILE (IN 2 == 1) DO                  // Stay in measuring mode while input 2 high
    new_mipos = MIPOS                   // Read "new" marker position
    IF (new_mipos != old_mipos) THEN                  // Check if a new marker was detected
      marker_distance = new_mipos - old_mipos         // Calculate marker distance
      IF (marker_distance < 0) THEN                   // Change sign if negative
        marker_distance = (marker_distance * -1)
      ENDIF
      IF (skip_first == 0) THEN       // Do not use first value as it might be invalid
        skip_first = 1
      ELSE
        marker_number = marker_number + 1   // Increment counter
        total_dist = total_dist + marker_distance // Summarize marker distances
      ENDIF
      old_mipos = new_mipos           // Set "old" marker position to "new" marker position
    ENDIF
  ENDWHILE
  SET SYNCMPULSM (total_dist rnd marker_number)       // calculate average marker distance and set par.
  OUT 2 0                               // Reset "Marker measurement active" output
RETURN
/********************************** Error handler ***********************************/
SUBPROG errhandle
  err = 1                   // Set error flag to remain in error handler until error is reset.
  OUT 8 1                   // Set error output
  OUT 2 0                   // Reset "Marker measurement active" output in case of error
  WHILE err DO                          // Remain in error handler until the reset is received
    IF (IN 8) AND NOT (IN 2) THEN       // Reset error when Input 8 is high and input 2+3 low
      ERRCLR                            // Clear error
      err=0                             // Reset error flag
    ENDIF
  ENDWHILE
  OUT 8 0                   // Reset error output
  sync_flag = 0             // Reset sync_flag after error
RETURN
/*****************************************************************************/
ENDPROG
/******************************** End of program *********************************/
```

## □ CAM Control

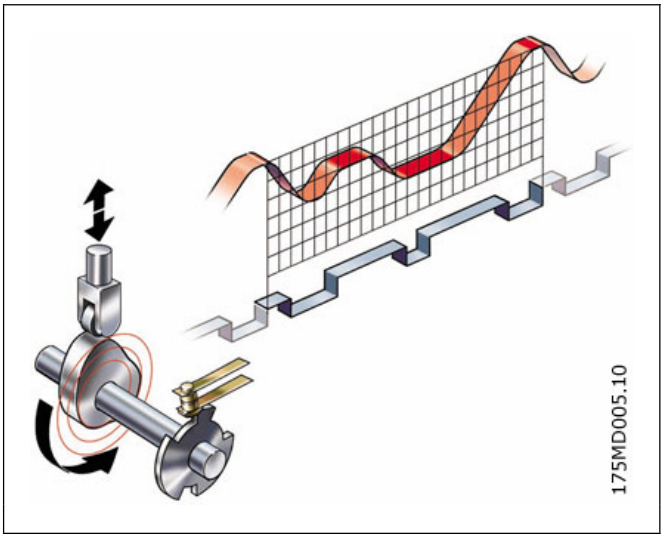
In order to realize CAM control, you need – depending on the application – at least one curve which describes the slave position in relation to the master position, as well as the engaging and disengaging behavior. Of course, many additional parameters are required for a CAM control which, together with the Fix points of the curve, produces a curve profile.

The synchronization in CAM-Mode (SYNCC command) can also be performed with marker correction (SYNCCMM and SYNCCMS). This would be required, for example, if the products are transported irregularly on a conveyor belt, or if adding errors need to be compensated for.

Diagram of the principle: The mechanical cam disc and the mechanical camshaft are shown on the left, the curves for the electronic CAM control and the electronic CAM box are shown on the right:

In order to create the curve profile, use the → *CAM-Editor*. Then you set the Fix points of the curve and define the parameters required for your application.

You can enter all values in physical or user-defined units under a Windows interface. You can constantly control the curve profile graphically; in this way, you can check velocity and acceleration of the slave axis.




### □ Interpolation, Tangent Points, Accuracy, and Array

Interpolation

The *CAM-Editor* calculates the curve from Fix points with the help of a spline interpolation. This is optimized to a minimum torque. In order to prevent speed leaps in the case of repeated curve cycles, the velocity at the beginning and the end is equated. You can choose between three types of curve for this calculation. In any type, the interpolation takes account of the gradient of the curve at the beginning and the end. Either the gradient at the beginning and at the end is averaged or the gradient at the beginning of the curve is also used for the end of the curve, or [0] is used for both the beginning and end gradient.

Tangent Points for Straight Sections

For areas where the velocity must be constant and the acceleration = 0, you need to use tangent points. A straight line will be drawn instead of a spline between these points.

Accuracy

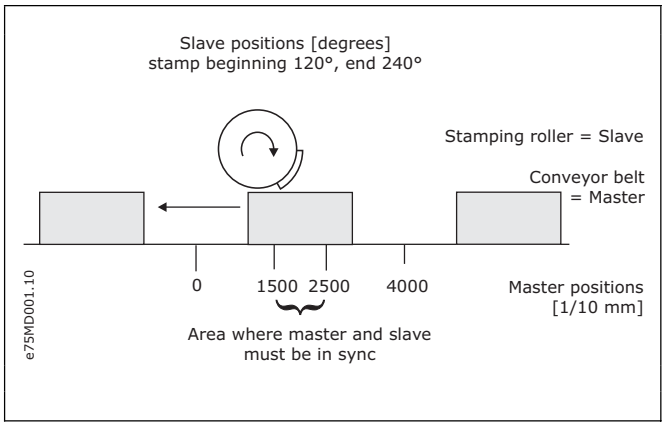
The Fix points are used directly as interpolation points provided that this is permitted by the interval distance. The *CAM-Editor* performs a linear interpolation between the interpolation points. If a fixpoint is not hit by the selected interval distance, then the corresponding slave reference value does not exist in the interpolation table. You can avoid such deviations if you have activated → ☑ *Snap on Grid.*

Internal realization as array

The curve profiles are realized internally as arrays which you can call up by means of a DIM instruction and the SETCURVE command.

## □ Stamping of Boxes with Use-by Date

The following example explains step by step how to edit the curve for this application of the CAM control and then how to incorporate it into your control program. A roller is supposed to stamp an inscription with a length of 10 cm on cardboard boxes. The stamp corresponds to a roller section of 120 degrees. 60 cardboard boxes are transported on the conveyor belt per minute. The cardboard boxes are always transported on the conveyor belt at exactly the same distance from each other (e.g. by means of a mechanical set pattern). During the printing process, the stamping roller and the cardboard box must run in sync:




### □ How to Edit the Curve Step by Step

1. Set the FC 300 with the required parameters.
2. Select this CNF file, APOSS will then automatically open the selected file as well as the *CAM-Editor*.
3. Determine the gearing factor of the master in MU units.

   The input should be possible in 1/10 mm resolution.

   The drive is connected with the conveyor belt by means of a gearing of 25:11; i.e. the motor makes 25, the drive pulley 11 revolutions.

   Gearing factor = 25/11

   Incremental encoder directly on the master drive; encoder resolution = 4096

   The drive pulley has 20 teeth/revolution, 2 teeth correspond to 10 mm, thus 1 revolution corresponds to = 100 mm conveyor belt feed or 1000/10 mm.
   Thus, the scaling factor is 1000.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} qc = 1 \text{ MU}$$

$$\frac{25/11 * 4096 * 4}{1000} qc = \frac{25 * 4096 \times * 4}{1000 \times 11} qc = 1 \text{ MU}$$

$$= \frac{2048}{55} qc = 1 \text{ MU} = \frac{\text{par. 33 - 10 } \textit{Syncfactor Master}}{\text{par. 33 - 11 } \textit{Syncfactor Slave}}$$

   Enter these values in the index card → *Synchronization* (the selected units should always be whole numbers):

   par. 33-10 *Syncfactor Master* = 2048
   par. 33-11 *Syncfactor Slave* = 55

4. Enter the gearing factor of the slave in UU units:

   Gearing factor = 5/1
   Encoder resolution (increm. encoder) = 500

   One revolution of the roller is 360 degrees. We are going to work with a resolution of 1/10 degrees. This means that we are dividing one revolution of the roller into 3600 work units:
   Scaling factor = 3600

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} qc = 1 \text{ UU}$$

$$\frac{5/1 * 500 * 4}{3600} qc = \frac{5 * 500 * 4}{3600} qc = 1 \text{ UU}$$

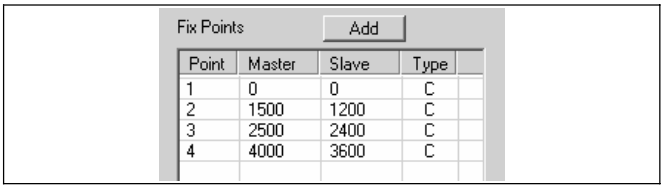
$$= \frac{25}{9} qc = 1 \text{ UU} = \frac{\text{par. 32 - 12 } \textit{User Unit Numerator}}{\text{par. 32 - 11 } \textit{User Unit Denomintor}}$$

   Enter these whole number values in the index card → Encoder:

   par. 32-12 *User Unit Numerator* = 25
   par. 32-11 *User Unit Denominator* = 9

5. Determine a whole number factor for the intervals in the index card → *Curve Data* so that the Fix points are on the interpolation points. Set this using the "Set" button.
   A complete cycle length of the master is 400 mm; this corresponds to 4000 MU.
   The → *Number of Intervals* = 40 results in a reasonable interval time of 25 ms.

6. Define → *Fix points* for the conveyor belt (master) and the roller (slave). The function → *Snap on Grid* should be activated.

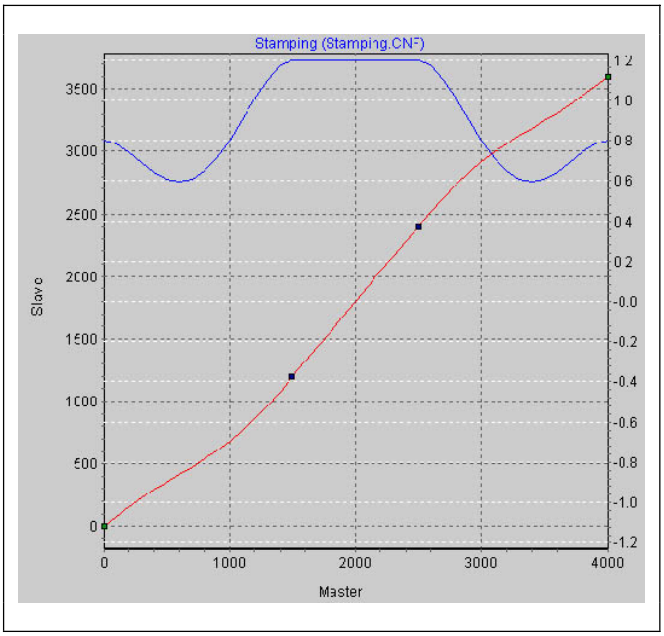



7. Master and slave must run synchronously with the same velocity between the position 1500 and 2500. This requires a straight line that is determined by two tangent points.

   Double-click in the column → *Type* for the fixpoint at position 2500.

   Alternatively, you can move the cursor to the fixpoint 2500, click on the right mouse button and select → *Change Type* in the subsequent context menu. Since two tangent points are always required, the previous one (at 1500) will also be changed at the same time.

8. Activate the diagram of the → ☑ *Velocity* to see the corresponding velocity curve:
9. Enter the → *Cycles/min Master* = 60 in the index card → *Curve Info*. This is the (maximum) number of cardboard boxes that are processed per minute.
10. Verify whether the acceleration of the slave is within the limit. For this purpose, you must activate the illustration of the → ☑ *Acceleration* and of the → ☑ *Acc. Limit.*
11. In order to load the curve into your control system, you must first save the file as a CNF file by clicking the "Save CNF As" button. You will see the name of the curve and the number of array elements in the title bar. You will need the latter for the DIM instruction during programming.




12. Load the CNF file with the modified parameters and the – automatically generated – curve arrays into the FC 300 by means of *Parameters → Restore from file.*

### □ Program Example: Stamping of Boxes with Use-by Date

Since the curve is stored internally as an array, the DIM instruction must appear first in your program:

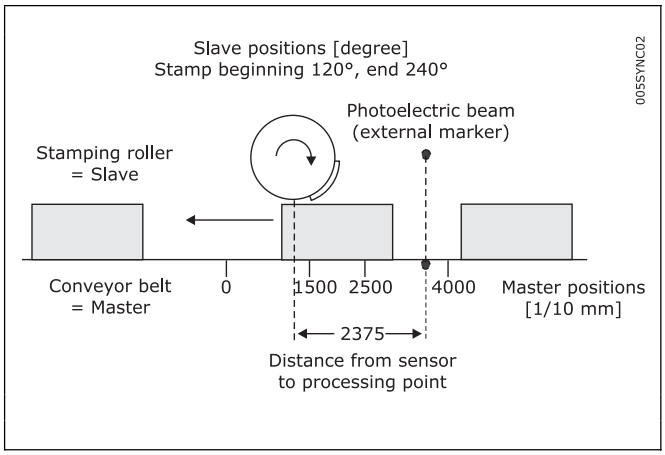
```
DIM stamping[92]        // Number of elements from the title bar of the CAM-Editor
HOME                    // Slave axis performs a home run (switch for zero position on top)
  // Afterwards, the slave will be in the zero position (0 degrees)
  // (Omitted if an absolute encoder is used)
SETCURVE stamping       // Set stamp curve
  // Assumption: A box is positioned at the processing point with its front edge and the master stands idle
DEFMCPOS 1000           // 1000 corresponds to this position (front edge of box)
POSA CURVEPOS           // Bring slave to the curve position that corresponds to the master position
SYNCC 0                 // Change into and remain in CAM-Mode
SYNCCSTART 0            // Engage roller immediately with the set maximum velocity
  // This does not cause any movement since the master is idle and in the correct position
  // Now the master can be started
start:      // Empty main loop so that program will not be terminated
            // Additional processing could take place here
GOTO start
```

## □ Printing of Cardboard Boxes with Marker Correction

If the boxes are not always transported at the exact same distance from each other, markers will be required that can recognize a box and correct the synchronization.

The following section describes how you can adapt the curve of the previous example to this application.

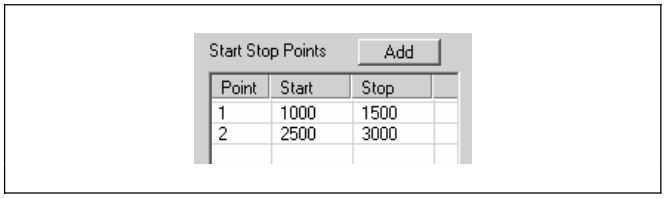
Again, a roller is supposed to stamp an inscription with a length of 10 cm on cardboard boxes. A maximum of 60 boxes are transported on the conveyor belt per minute. During the printing operation, both the stamping roller and the box must run in sync.




### □ How to Edit a Curve for Synchronization with Marker

1. Steps 1 to 9 are the same as in the previous example.

10. Define the point pairs for the engaging and disengaging in the list → *Start-Stop Points*. We make the assumption that engaging takes place at the beginning of the box and disengaging is to take place until the end of the box.




11. In the index card → *Curve Data*, determine the position in which the roller should stop if no other *Slave Stop Position* is defined in the program.

    The roller always needs to return to the position 0 degrees: → *Slave Stop Position* = 0

12. The photoelectric beam (external marker) has a distance of 237.5 mm from the processing point (= stamp touches the cardboard box) and detects the beginning of the box (corresponding to master position 1000). The marker distance is therefore 2375. Enter this value in the index card → *Synchronization* and define the permitted tolerance for the appearance of the markers and the external marker type = 2 for the master.

    | | | |
    |---|---|---|
    | par. 33-17 | *Master Marker Distance* | = 2375 |
    | par. 33-21 | *Master Marker Tolerance Window* | = 200 |
    | par. 33-19 | *Master Marker Type* | = 2 |

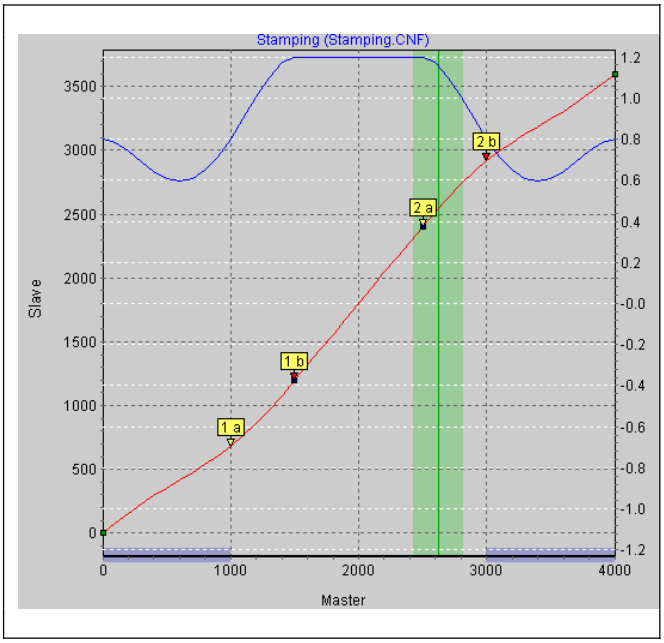
    Enter the master position in the index card → *Curve Data*:

    | | |
    |---|---|
    | Master-Marker-Position | = 100 |

13. Take a look at the curve profile in order to determine when the correction of the synchronization may begin at the earliest and when it must be finished. The vertical green line indicates the master position where the marker is recognized, the light green area shows the tolerance window for the appearance of the master marker.

    At the earliest, the correction may begin when the printing of a box has been completed, since any change of velocity during the printing process would damage the box. The correction must be finished completely when the next cardboard box reaches the processing point. In this example, the master positions at the end and beginning of a box are quite suitable:

    Correction Start = 3000
    Correction End = 1000




    Enter the values in the index card → *Curve Data*; the depiction of the area is shaded in blue in the curve profile.

14. Verify whether the velocity and acceleration of the slave remain within the limit. For this purpose, you must activate the illustration of the → ☑ *Velocity* and of the → ☑ *Vel.-Limit* and then the illustration of the → ☑ *Acceleration* and of the → ☑ *Acc.-Limit.*
15. Click the "Save CNF As" button to save the CNF file, for example "marker".
16. Load the CNF file with the modified parameters and the – automatically generated – curve arrays into the FC 300 by means of *Parameters* → *Restore from file.*

### □ Program Example: Synchronization with Marker

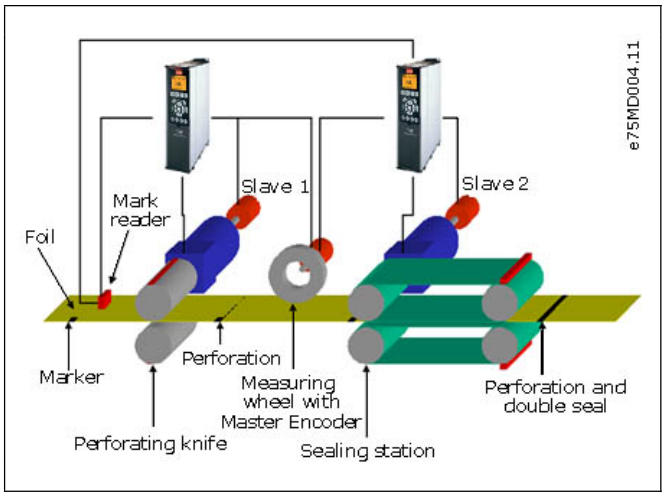
Since the curve is stored internally as an array, the DIM instruction must appear first in your program:

```
/******** Printing of Cardboard Boxes with Marker Correction (Synchronization with Marker) ******/
DIM marker[112]              // Number of elements from the title bar of the CAM-Editor
HOME                         // Slave axis performs a home run (switch for zero position on top)
      // Afterwards, the slave will be in the zero position (0 degrees)
      // (Omitted if an absolute encoder is used)
SETCURVE marker              // Set stamp curve with marker
dist = GET SYNCMPULSM        // Distance to sensor
DEFMCPOS (1000-dist)         // This is the location that corresponds to the sensor signal
SET SYNCMSTART 2000          // Counting of the master pulse does not begin
      // until the next edge comes from the sensor
SYNCCMM 0                    // Synchronize in CAM-Mode until motor stop
SYNCCSTART 1                 // Engage roller with start point pair 1
      // Synchronous operation
WAITI 4 ON                   // Wait for input signal when conveyor belt is being switched off
SYNCCSTOP 2 0                // Disengage roller with stop point pair 1 and stop at position 0 degrees
```

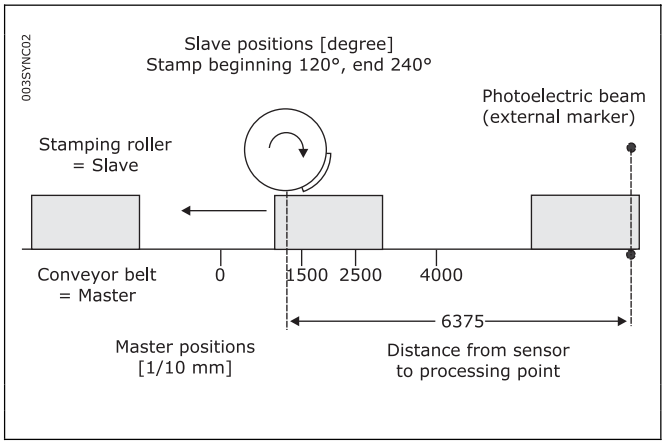
## □ If the Sensor Distance is Larger than one Master Cycle Length

In many applications, the marker cannot be placed within one master cycle length, for example in the case of the following equipment for the production of plastic bags:

Since no markers can be installed between the slaves here, there is only a marker reader in this application; the welding station is much farther away than one master cycle length. Since the distance of the sensor is larger than one master cycle length, a buffer is provided for the marker deviation. When the marker appears, the value is entered in the buffer and read out upon the appearance of the next marker: see figure.



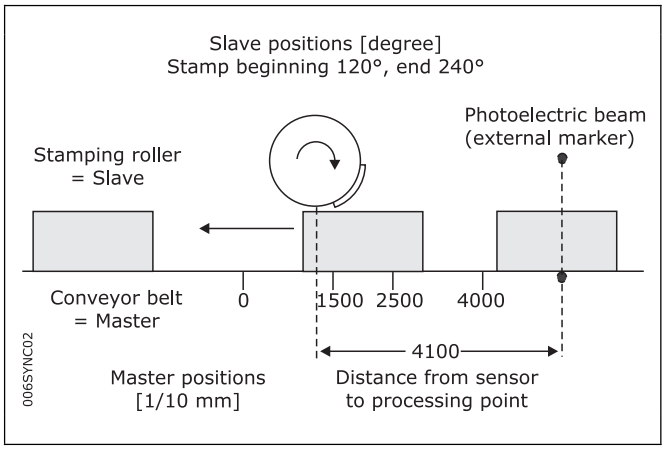

In order to assess in what area corrections may be made, you should subtract the master cycle length as often as necessary until the value is < 1 master cycle length. This is the maximum permitted distance for making corrections. In this example, it is 6375 – 4000 = 2375 and thus the same correction range as in the previous example.



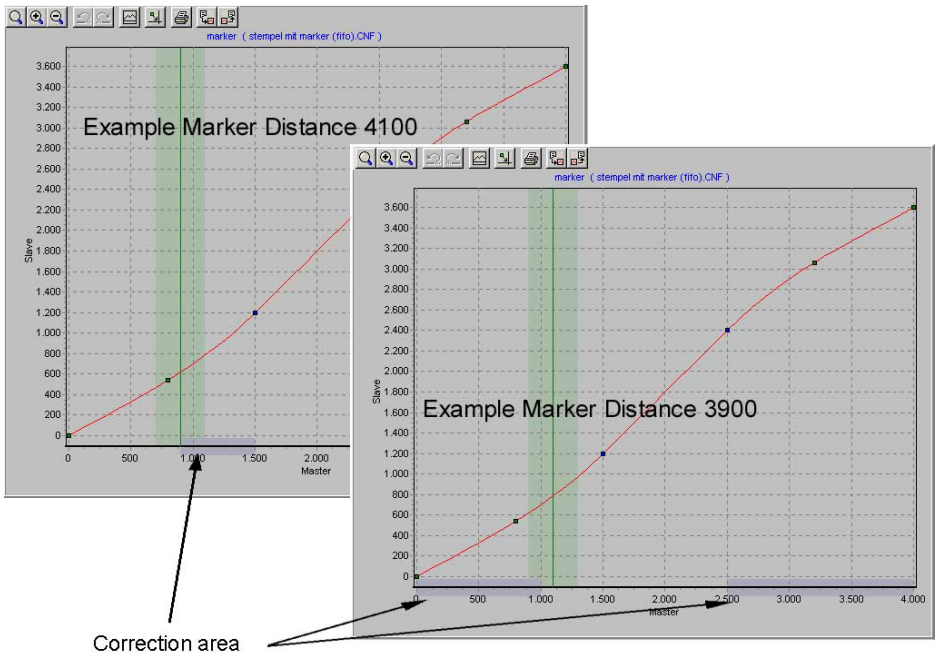

## □ Problematic Situations in the Determination of the Marker Distance

If the marker is mounted so close to the processing point that there is no time left to correct the synchronization after the marker has been detected, then you can remedy the problem only through mechanical modification of the marker.

On the other hand, the same effect might also occur if the marker distance is larger than the master cycle length and if an insufficient distance likewise remains after the subtraction of this value, for example:




The value is entered into the buffer when the marker appears. The buffer is read out only when the next marker is recognized. This means that the marker is only "recognized" at the master position 900 and that there is little time left in our example to correct the error. It is the same effect as if the sensor had been mounted at the value (distance – master cycle length) or (4100 – 4000), respectively, i.e. only 10 mm in front of the processing point.




Thus, it would be better to install the sensor in such a way that the distance to the processing point is either smaller or substantially larger than one master cycle length; here, for example, at a distance of 3900. Then it is possible to correct from 2500 to 1000.

Alternatively, the sensor could be installed further away, for example at a distance of 7900; this has the same effect as if the sensor had been installed at distance – master cycle length (7900 – 4000), i.e. 3900 in front of the processing point. This allows enough time to correct the synchronization.

If this cannot be done mechanically, then the values must be manipulated somewhat in order to avoid the solution with the buffer. Please proceed as follows:

Subtract a value x from the actual distance so that the distance becomes < than the master cycle length, for example 4100 – 200 = 3900. You also subtract the value x from the master position, i.e. 1000 – 200 = 800.

Enter both values in the index cards: → *Synchronization* and → *Curve Data*:

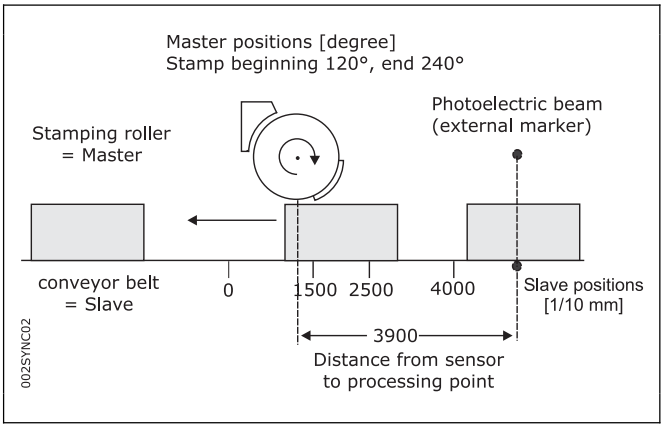
| | |
|---|---|
| par. 33-17 *Master Marker Distance* | = 3900 |
| Master Marker Position | = 800 |

Since no buffer is generated now, it would be possible to correct from 2500 to 800, for example

## □ Slave Synchronization with Marker

In the following example, the conveyor belt is the slave and the stamp roller the master, since the take-up and delivery of the dye must be continuous for a uniform printing process. A maximum of 20 cardboard boxes are transported on the conveyor belt per minute. The distance of the boxes is not larger than one master cycle length. Stamp roller and box must run in sync during the printing operation: see figure.

In contrast to the synchronization with master marker correction, here the slave position is being corrected instead of the curve.



### ☐ How to Edit a curve for the Slave Synchronization

1. Set the FC 300 with the required parameters and save these parameters with *Parameters* → *Save to file* with the extension "CNF".
2. This CNF file must be open in the *CAM-Editor*.
3. Determine the gearing factor of the master in MU units.

   Gearing factor = 5/1
   Encoder resolution (Incremental encoder) = 500

   One revolution of the roller is 360 degrees. We are going to work with a resolution of 1/10 degrees. This means that we are dividing one revolution of the roller into 3600 work units:
   Scaling factor = 3600

   $$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{qc} = 1 \text{ MU}$$

   Enter these whole number values in the index card → Synchronization:

   Par. 33-10 *Syncfactor Master* = 25
   Par. 33-11 *Syncfactor Slave* = 9

4. Enter the gearing factor of the slave in UU units:

   The input should be possible in 1/10 mm resolution.

   The drive is connected with the conveyor belt by means of a gearing of 25:11; i.e. the motor makes 25, the drive pulley 11 revolutions.
   Gearing factor = 25/11

   Incremental encoder directly on the master drive; encoder resolution = 4096

   The drive pulley has 20 teeth/revolution, 2 teeth correspond to 10 mm, thus 1 revolution corresponds to = 100 mm conveyor belt feed or 1000/10 mm.
   Thus, the scaling factor is 1000

   $$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{qc} = 1 \text{ UU}$$

   Enter these values in the index card → *Encoder*:

   Par. 32-12 *User Unit Numerator* = 2048
   Par. 32-11 *User Unit Denominator* = 55

5. Determine a whole number factor for the intervals in the index card → *Curve Data* so that the Fix points are on the interpolation points.

   A complete cycle length of the master is 360 degrees; this corresponds to 3600 MU.
   For a master cycle length of 3600, the → *Number of Intervals* = 36 produces a reasonable interval time of 27.7 ms. Set these two values using the "Set" button in the *Curve Data* index card.

6. Define → *Fix points* for the roller (slave) and the conveyor belt (master). The function → *Snap on Grid* should be activated.

Fix Points — Add

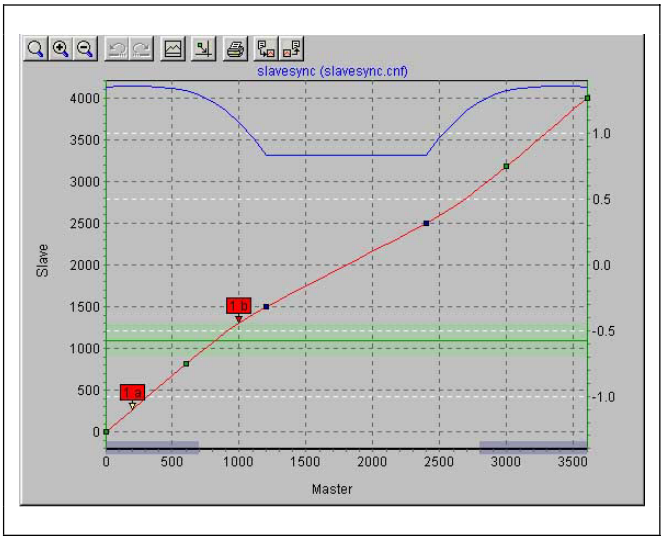
| Point | Master | Slave | Type |
|---|---|---|---|
| 1 | 0 | 0 | C |
| 2 | 1200 | 1500 | C |
| 3 | 2400 | 2500 | C |
| 4 | 3600 | 4000 | C |

7. Master and slave must run synchronously with the same velocity between the master positions 1200 to 2400. For this, you need to have a straight line that is determined with two tangent points. Double-click in the column → *Type* for the fixpoint at position 2400.

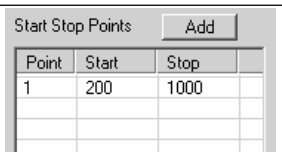
Fix Points — Add

| Point | Master | Slave | Type |
|---|---|---|---|
| 1 | 0 | 0 | C |
| 2 | 1200 | 1500 | T |
| 3 | 2400 | 2500 | T |
| 4 | 3600 | 4000 | C |

Activate the diagram of the → ☑ *Velocity* to see the corresponding velocity curve as it is shown in this figure:




8. Enter the → *Cycles/min Master* = 20 in the index card → *Curve Info*. This is the (maximum) number of cardboard boxes that can be processed per minute.
9. Verify whether the acceleration of the slave is within the limit. For this purpose, you must activate the illustration of the → ☑ *Acceleration* and of the → ☑ *Acc. Limit.*




10. Define in the list → *Start-Stop Points* with some safe distance in order to start the synchronization at the beginning. Engaging should take place between 20 and 100 degrees because it must be completed at 120 degrees.

11. In the index card → *Curve Data*, define the position where the conveyor belt should stop if no other *Slave Stop Position* is being defined in the program:

    The conveyor belt should always stop in position 0:
    Slave Stop Position = 0
12. The photoelectric beam (external marker) has a distance of 390 mm from the processing point (= stamp touches the box) and detects the beginning of the box (corresponds to slave position 1000). Thus, the marker distance is 3900. Enter this value in the index card → *Synchronization* and define the permitted tolerance for the appearance of the markers and the external marker type = 2 for the slave:

    | | | |
    |---|---|---|
    | Par. 33-18 | *Slave Marker Distance* | = 3900 |
    | Par. 33-22 | *Slave Marker Tolerance Window* | = 200 |
    | Par. 33-20 | *Slave Marker Type* | = 2 |

    Enter the master position in the index card → *Curve Data*:
    Slave-Marker-Position = 1000
13. Take a look at the curve profile in order to determine when the correction of the synchronization may begin at the earliest and when it must be finished. The green horizontal line indicates the master position where the marker is recognized, the light green area shows the tolerance window for the appearance of the master marker.

    At the earliest, the correction may begin when the printing of a cardboard box has been completed, since any change of velocity during the printing process would damage the printing stamp and/or the box. Also, the correction must have been completed in its entirety when the next box arrives at the processing point. In this example, the slave positions at the end and beginning of a box are quite suitable. Enter the values in the index card → *Curve Data*:

    | | |
    |---|---|
    | Correction Start | = 2800 |
    | Correction End | = 750 |
14. Verify whether the velocity and acceleration of the slave remain within the limit. For this purpose, you must activate the illustration of the → ☑ *Velocity* and of the → ☑ *Vel.-Limit* and then the illustration of the → ☑ *Acceleration* and of the → ☑ *Acc.-Limit*.
15. Click the "Save CNF As" button to save the CNF file.

16. Load the CNF file with the modified parameters and the – automatically generated – curve arrays into the FC 300 by means of *Parameters → Restore from file.*

### □ Program Example: Slave Synchronization with Marker

In order to determine the master position, a switch on the master is required that indicates the zero position. In order to put the slave into the correct position, it will be moved forward to the photoelectric beam. This corresponds to the beginning of the box = 1000. Then the slave will be moved further by 2900 (= marker distance 3900–1000); thus, the slave is exactly in front of the processing point with the beginning of the cardboard box 1000, i.e. at slave position 0.

```
DIM slavesync[108]          // Number of elements from the title bar of the CAM-Editor
HOME                // Slave axis performs a home run (switch for zero position on top)
                    // afterwards, the slave will be in the zero position (0 degrees)
                    // (Omitted if an absolute encoder is used)
DEFMCPOS 0                  // Curve starts at master position 0
SET SYNCMSTART 2000         // Counting of the master pulse does not begin
                            // until the next edge comes from the sensor
SETCURVE slavesync          // Set curve for the slave synchronization
CSTART                      // Go to start
CVEL 10                     // Go forward slowly until photoelectric beam appears
oldi = IPOS                 // oldi = last marker position of the slave
WHILE (oldi == IPOS) DO     // Wait until box is detected
ENDWHILE
POSA (IPOS + 2900)          // Move box forward by 2900
SYNCCMS 0                   // Synchronize in CAM-Mode
SYNCCSTART 1                // Engage with start-stop point pair 1
```

## □ CAM Box

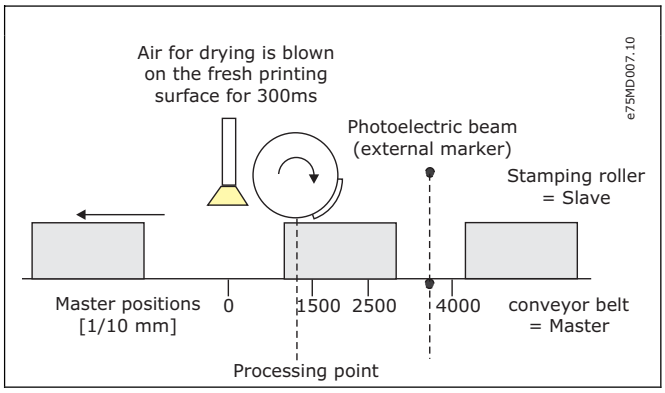
The mechanical camshaft is also reproduced by one (or more) curves. In order to realize a CAM box, it must be possible to engage and disengage the slave at specific master positions over and over again.

This is possible with APOSS with the interrupt command ON MAPOS .. GOSUB and ON APOS .. GOSUB. It is possible to call up a subprogram whenever a defined master position has been passed (both in the positive or negative direction, to be precise).

It is possible to realize many applications that are typical for the packaging industry in connection with a curve profile in which several have been defined for engaging and disengaging.

### □ Program Example of a CAM Box

After a cardboard box has been printed, the fresh print is to be dried immediately in the air stream:

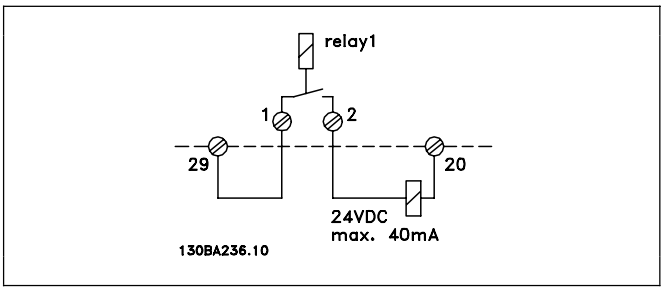


```
ON MCPOS 2500 GOSUB drier
      // Call up a subprogram when the master position 2500 is passed in positive direction
SUBMAINPROG
   SUBPROG drier
      OUT 1 1              // Turn on drier
      DELAY 300            // Dry for 300 ms
      OUT 1 0              // Turn off drier
   RETURN
ENDPROG
```

## Mechanical Brake Control

In applications controlled by MCO 305 involving an electro-mechanical brake it normally makes sense to control the brake from the MCO 305 application program to avoid situations when the position controller of MCO 305 attempts to move the motor while the brake is still engaged.

Controlling the brake from the MCO 305 application program can be combined with the Mechanical brake control of FC300 by using 2 outputs in series e.g. Digital output 29 set to Mechanical brake control (par. 5-31) and relay output 1 set to MCO controlled (par. 5-40 [0]) and connect the brake as shown below:




### Program Example for Relative Positioning with Mechanical Brake

```
/*********************************************************************/
      Inputs:        1        Go to position
                     8        Clear Error
      Outputs:       1        In position
                     8        Error
                     11       Relay output for mechanical brake
/************************** Interrupts ********************************/
ON ERROR GOSUB errhandle
   // In case of error go to error handler routine, this must always be included
/************************  Define flags  ******************************/
flag = 0
/************************   Basic settings   ***************************/
VEL 80          // Sets positioning velocity related to par. 32-80 Maximum velocity.
ACC 100         // Sets positioning acceleration related to par. 32-81 Shortest ramp.
DEC 100         // Sets positioning deceleration related to par. 32-81 Shortest ramp.
/******************* Define application parameters *********************/
LINKGPAR 1900 "Box height" 0 1073741823 0
LINKGPAR 1901 "Brake close delay" 0 1000 0
LINKGPAR 1902 "Brake open delay" 0 1000 0
/********************   Initialize drive to safe state  ***************************/
GOSUB engage       // Ensure that mechanical brake is closed after power up.
/****************************** main loop ****************************/
MAIN:
IF (IN 1 == 1) AND (flag == 0) THEN
   // Go to position once (ensured by flag) when input 1 is high.
   GOSUB disengage              // Open mechanical brake before start.
   OUT 1 0                      // Reset "In position" output.
```

```
    POSR (GET 1900)              // Go to position.
    OUT 1 1                      // Set "In position" output.
    flag = 1            // Set "flag" to ensure that distance is only traveled once.
ELSEIF (IN 1 == 0) AND (flag == 1) THEN       // Stop once when input 1 is low
    MOTOR STOP                   // Stop if input is low.
    flag = 0                     // Reset "flag" to enable new positioning.
    GOSUB engage                 // Close mechanical brake after stop.
ENDIF
GOTO MAIN
/********************** Sub programs start ********************************/
SUBMAINPROG
/******************* Engage mechanical brake ******************************/
SUBPROG engage
    OUT 11 0                     // Close mechanical brake.
    DELAY (GET 1901)
    // Wait to ensure that brake is engaged before releasing the motor.
    MOTOR OFF                    // Stop position control and coast the motor.
RETURN
/***************** Disengage mechanical brake ****************************/
SUBPROG disengage
    MOTOR ON                     // Enable drive and start position control.
    DELAY (GET 1902)
    // Wait to ensure that motor is magnetized before opening the brake.
    OUT 11 1                     // Open mechanical brake.
RETURN
/********************** Error handler ************************************/
SUBPROG errhandle
    OUT 11 0  // Close mechanical brake in case of error.
    err = 1     // Set error flag to remain in error handler until error is reset.
    OUT 8 1   // Set error output.
    WHILE err DO     // Remain in error handler until the reset is received.
        IF IN 8 THEN                    // Reset error when Input 8 is high.
            ERRCLR                      // Clear error.
            err=0                       // Reset error flag.
        ENDIF
    ENDWHILE
    OUT 8 0                             // Reset error output.
    flag = 0                            // Reset "flag" to enable new positioning.
RETURN
/*****************************************************************************/
ENDPROG
/************************* End of program ********************************/
```

## Limited-Jerk

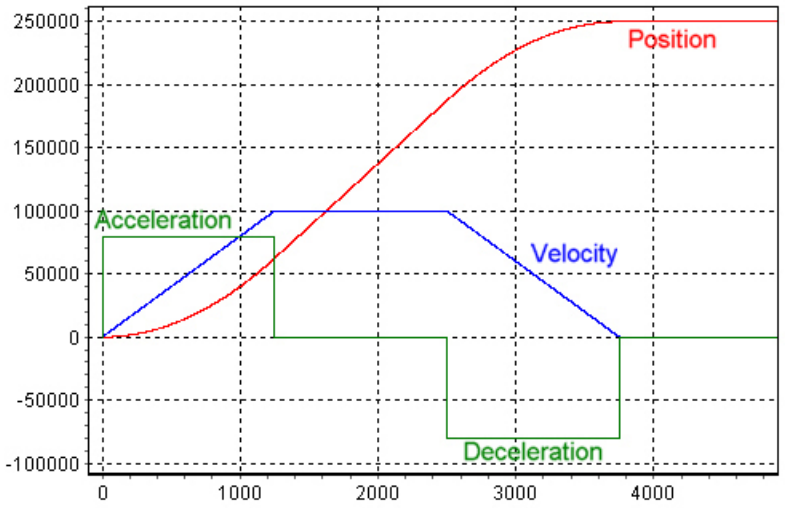
### Understanding Limited-Jerk Movements

Limited-jerk movements are similar to normal Trapezoidal movements except that the user may control the "gentleness" of the acceleration and deceleration ramps. This allows the user to limit the "jerk" caused by the "instantaneous" acceleration of a Trapezoidal movement.

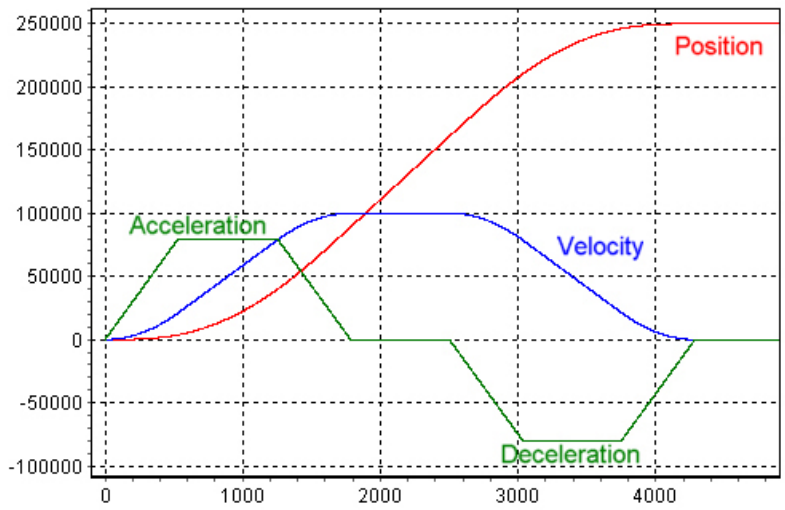
Typical applications where limited jerk is required:

- lift
- movement of heavy loads

As an example, the following chart shows the acceleration, velocity, and position curves for a Trapezoidal movement from one position to another position. The sharp changes in acceleration cause the motor to experience a "jerk" at the beginning and end of each velocity ramp.




The following chart shows the same movement done using a Limited-jerk movement. Note that the acceleration changes are no longer "instantaneous" and that the "corners" of the velocity curve have become rounded. This will result in a smoother motor movement. It will also require slightly longer to get to the desired target position because the motor takes longer to accelerate to maximum acceleration.



The user can control the "gentleness" of the acceleration ramp using 4 parameters:

**Parameter Limited Jerk**

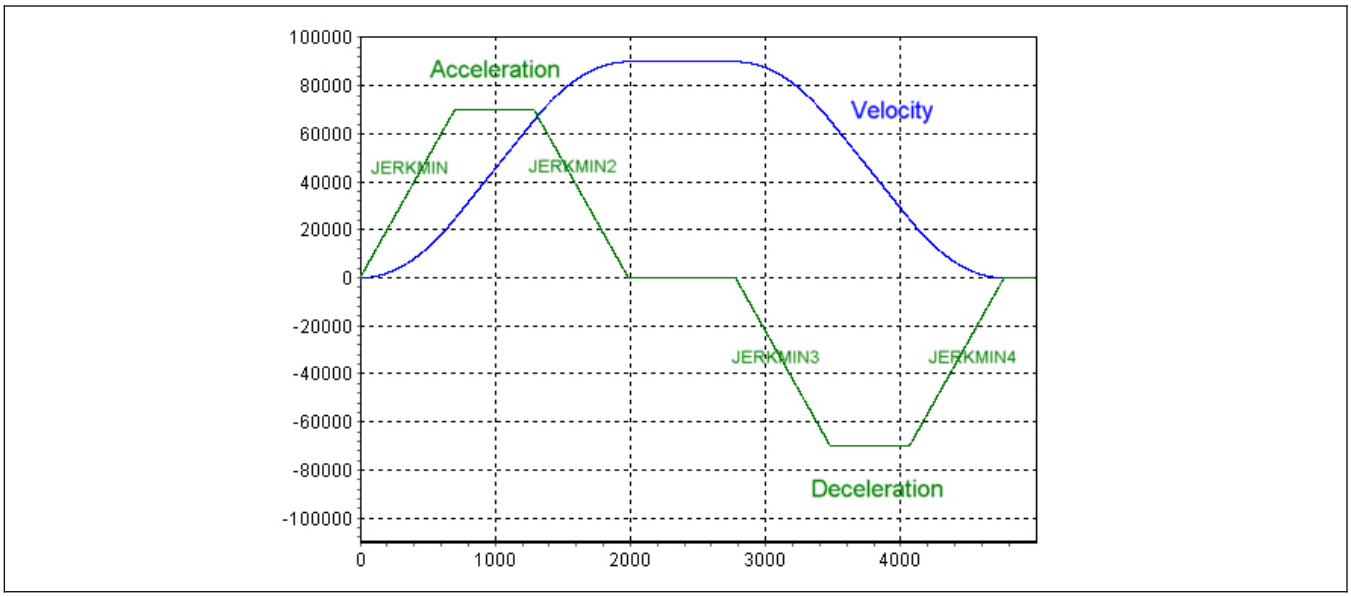
JERKMIN: Acceleration ramp-up constant. This specifies the number of milliseconds required to ramp the acceleration up from 0 to maximum acceleration.

JERKMIN2: Acceleration ramp-down constant. This specifies the number of milliseconds required to ramp the acceleration *down* from maximum acceleration to 0 (i.e. normally to constant maximum velocity). If set to 0, this will default to the same value as JERKMIN.

JERKMIN3: Deceleration ramp-up constant. This specifies the number of milliseconds required to ramp the deceleration *up* from 0 to maximum deceleration. If set to 0, this will default to the same value as JERKMIN.

JERKMIN4: Deceleration ramp-down constant. This specifies the number of milliseconds required to ramp the deceleration down from maximum deceleration to 0 (i.e. normally to 0 velocity). If set to 0, this will default to the same value as JERMIN.

These constants correspond to the "slopes" of the different portions of the acceleration curve. This is shown in the chart below. When set to larger and larger numbers, the acceleration and/or deceleration will become gentler and gentler as the ramps become longer and longer. Note that the acceleration slope determined by the JERKMIN acceleration ramp-up constant will be used whenever the acceleration is being ramped-up. It is not used only to ramp-up from 0 to maximum acceleration. Similarly, JERKMIN2 is used whenever the acceleration is being ramped-down, JERKMIN3 is used whenever the deceleration is being ramped-up, and JERKMIN4 is used whenever the deceleration is being ramped-down.




Limited-jerk movements will not normally exceed the velocity and acceleration limits set for the controller (e.g. limits set by the VEL, ACC, DEC, etc., commands). In the above chart, these limits can be recognized by the "plateaus" in the acceleration curve. If the current velocity and/or acceleration are outside these limits when the Limited-jerk movement is started, then the movement is accelerated or decelerated as necessary to bring the movement within the set limits.

It is important to understand that for Limited-jerk movements, "acceleration" is defined as "speeding up" in either direction (i.e. either forwards or backwards) and "deceleration" is defined as "slowing down" in either direction. A result of this is that maximum velocity, maximum acceleration, maximum deceleration, and the four JERKMIN values are all independent of the direction of movement. This can have important consequences when a Limited-jerk movement must "reverse" the direction of the motor, particularly when maximum deceleration differs from maximum acceleration. In this case, the Limited-jerk movement will ensure that the *deceleration* ramp flows smoothly into an *acceleration* ramp at exactly 0 velocity when the direction changes and without exceeding either the deceleration or acceleration limits.

A Limited-jerk movement can be used in three different situations:

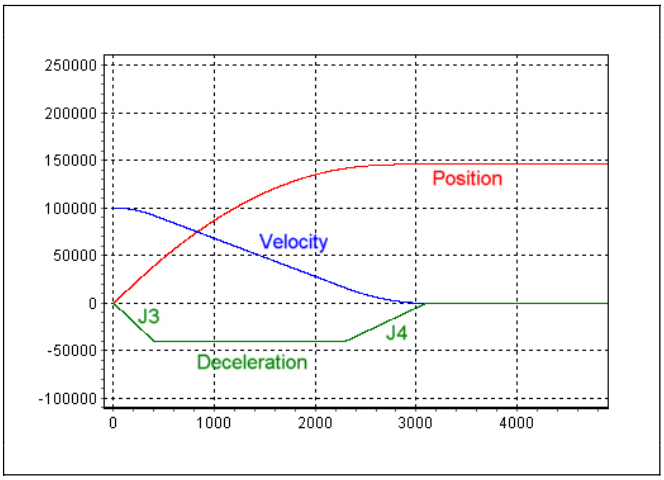
1. Stopping from the current velocity and acceleration (where the final position is not important).
2. Changing from the current velocity and acceleration to a specified constant velocity (where the positions are not important).
3. Moving from the current position (and the current velocity and acceleration) and stopping at a specified position.

### □ Examples

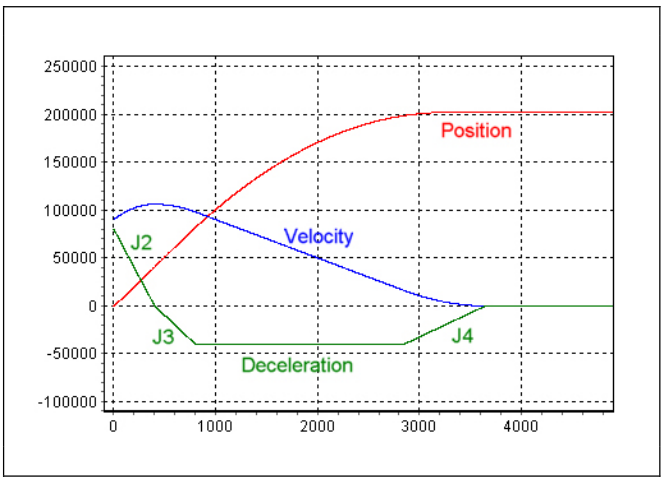
In the following examples, maximum acceleration has been set to a value higher than maximum deceleration so the motor can speed up faster than it can slow down. As well, JERKMIN has been set smaller than JERKMIN2, JERKMIN2 smaller than JERKMIN3, and JERKMIN3 smaller than JERKMIN4. This was done so that the various segments of the curves can be visually distinguished in the charts. The four JERKMIN values have been labeled simply J1, J2, J3, and J4.

#### Stopping

This chart shows a stopping movement that begins from a positive constant velocity. The curve consists of a deceleration ramp-up segment (using JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to 0 velocity (using JERKMIN4).

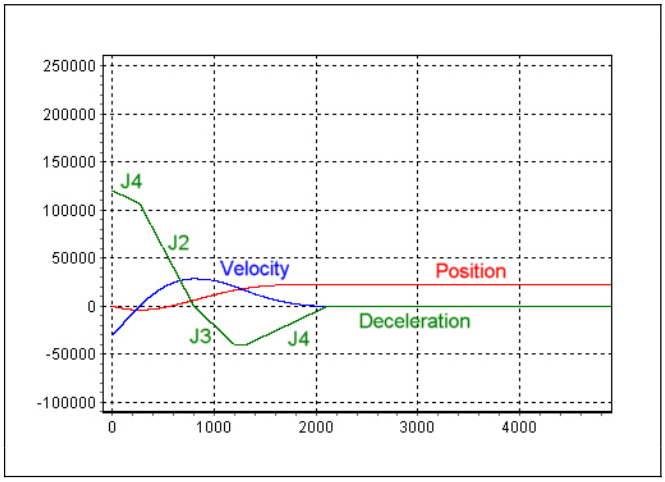



This chart shows a stopping movement that begins from a positive velocity and a positive acceleration. Since the initial acceleration is positive, the curve must start with an acceleration ramp-down segment to 0 acceleration (using JERKMIN2). This will then be followed by a deceleration ramp-up segment (using JERKMIN3), a constant deceleration segment, and a deceleration ramp-down segment to 0 velocity (using JERKMIN4).



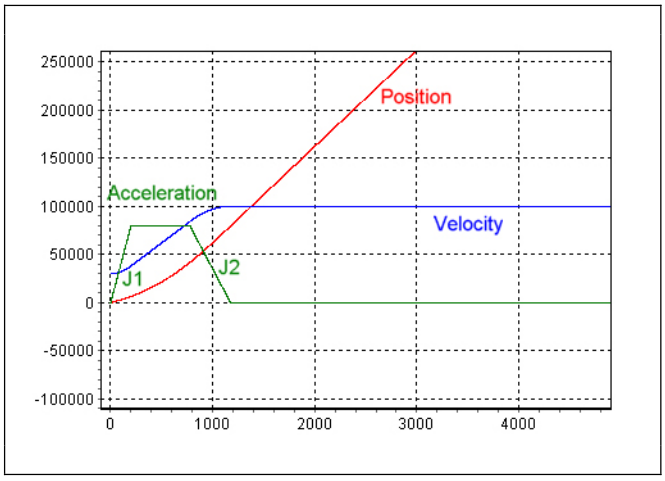
The chart below shows a stopping movement that begins from a *negative* velocity and a very high deceleration. (It is a deceleration because the speed is slowing down.) However, because the initial deceleration is so high, the motor is unable to stop without overshooting 0 velocity and 'coming back'.

So the curve starts with a deceleration ramp-down segment (using JERKMIN4) to slow the deceleration as much as possible before reaching 0 velocity. At 0 velocity, the "deceleration" becomes an 'acceleration'" because the direction has changed. Hence, the curve continues by ramping-down the acceleration (using JERKMIN2) until it gets to 0 acceleration. The motor is now at a constant positive velocity and so the curve will finish in the normal way with a deceleration ramp-up segment (using JERKMIN3), a constant deceleration segment (very short in this example), and a deceleration ramp-down segment to 0 velocity (using JERKMIN4).

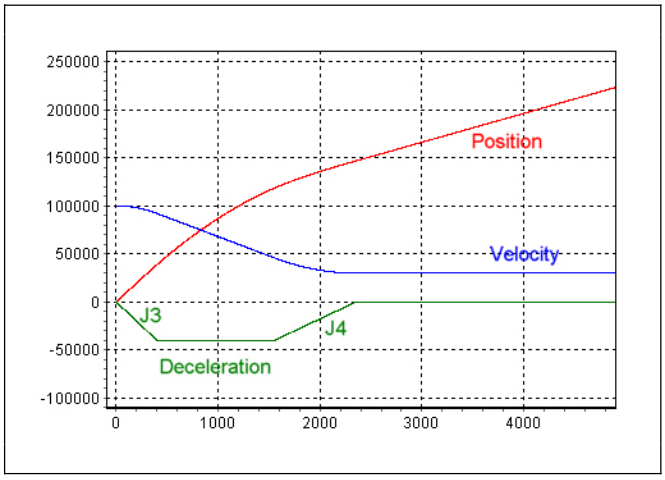



### Changing to a Constant Velocity

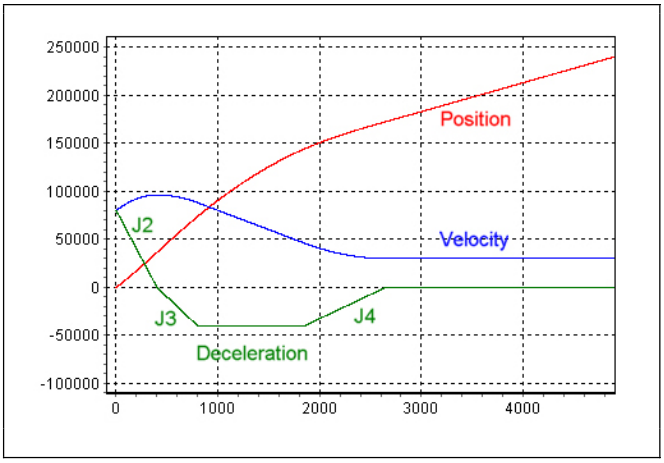
This chart shows a movement that begins from a positive constant velocity and increases its speed to a higher positive constant velocity. This curve consists of an acceleration ramp-up segment (using JERKMIN), followed by a constant acceleration segment (at maximum acceleration), and finally an acceleration ramp-down segment to constant velocity (using JERKMIN2). Note that the deceleration JERKMIN3 and JERKMIN4 values are not used because there is never any deceleration.




This chart shows a movement that begins from a high positive constant velocity and decreases its speed to a lower positive constant velocity. This curve consists of a deceleration ramp-up segment (using JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to constant velocity (using JERKMIN4). Note that the acceleration JERKMIN and JERKMIN2 values are not used because there is never any acceleration.

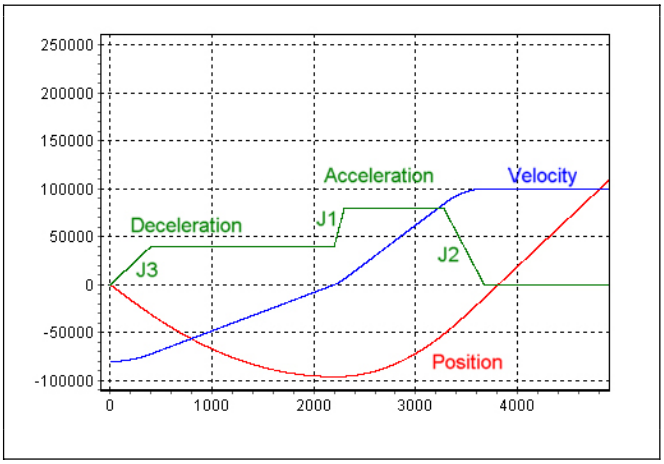


This chart is similar to the previous example except that it begins with a positive acceleration. In this case, the curve must start with an acceleration ramp-down segment (using JERKMIN2). Once the acceleration reaches 0, then the curve can continue as before.




The chart below shows a movement that begins with a *negative* constant velocity and changes direction to a positive constant velocity. The curve must start by slowing down the speed so that it can 'turn around'. Hence the curve begins with a deceleration ramp-up segment (using JERKMIN3) until it reaches maximum deceleration.

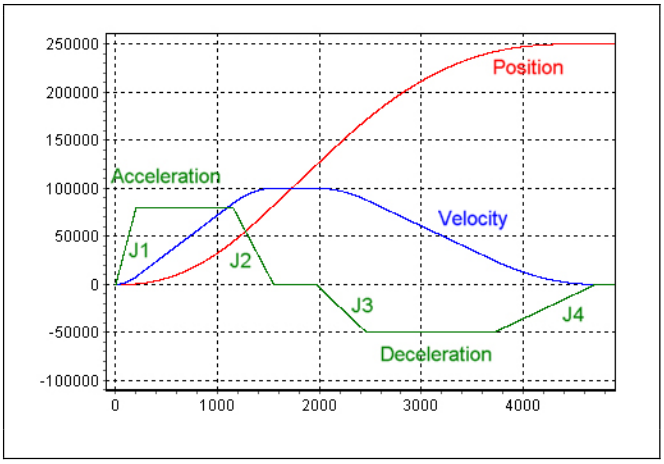
The deceleration continues at maximum deceleration until it reaches 0 velocity. Note that there is no deceleration ramp-down segment because the movement is not stopping. At exactly 0 velocity, the direction reverses and the movement is now accelerating in the other direction. But since maximum acceleration is higher than maximum deceleration for this example, the curve is now able to include an acceleration ramp-up segment (using JERKMIN). The curve finishes in the normal way with a constant acceleration segment and an acceleration ramp-down segment to constant velocity (using JERKMIN2).



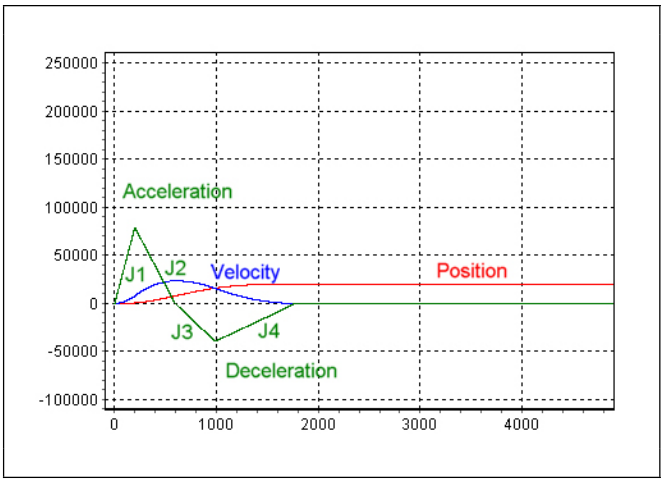

### Moving to a Specified Position

The chart below shows a 'normal' movement from being stopped at one position and then moving forward to stop at another position. The curve starts with an acceleration ramp to maximum velocity. This portion of the curve will be similar to the first of the examples shown in "Changing to a Constant Velocity". The curve is simply changing to a constant velocity where the constant velocity is the maximum velocity.

Hence, the curve will consist of an acceleration ramp-up segment (using JERKMIN), a constant acceleration segment at maximum acceleration, and then an acceleration ramp-down segment to maximum velocity (using JERKMIN2). The movement then proceeds at maximum velocity until it needs to start the deceleration ramp that will stop the movement at the desired position. The deceleration ramp is identical to the first of the examples shown in "Stopping". The curve will consist of a deceleration ramp-up segment (JERKMIN3), followed by a constant deceleration segment (at maximum deceleration), and finally a deceleration ramp-down segment to 0 velocity (using JERKMIN4) stopping at the desired position.

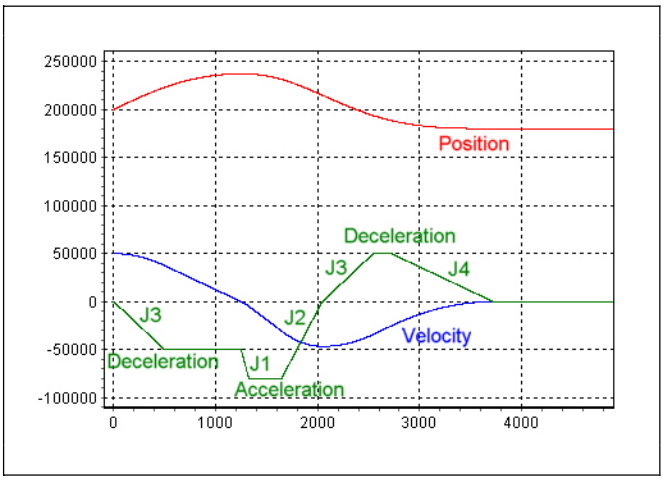


This chart shows a typical 'short' movement where maximum velocity cannot be reached. In this case, the curve ramps-up the acceleration (using JERKMIN) for as long as possible. This may or may not reach maximum acceleration, depending on how far away the target position is. There will then be an acceleration ramp-down (using JERKMIN2), followed immediately by a deceleration ramp-up (using JERKMIN3). Again, depending on the target position, there may or may not be a constant deceleration segment. The curve ends with a deceleration ramp-down to 0 velocity at the target position.

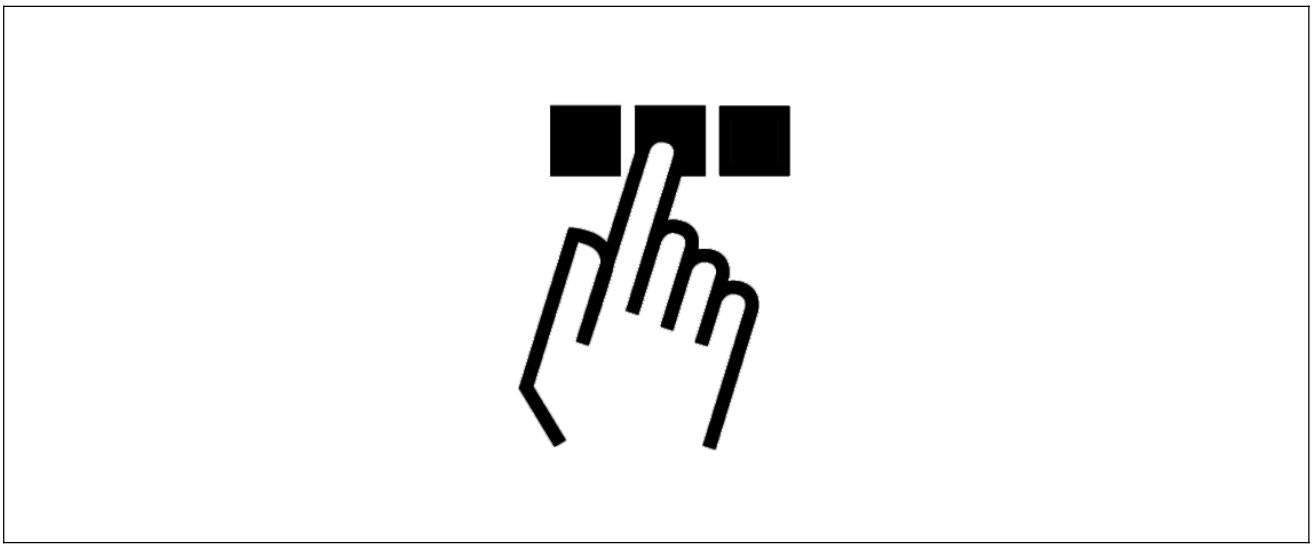



The chart below shows an example where the motor is initially moving in the "wrong" direction and must turn around and 'go back' to the target position. Since it must "turn around", the curve starts with a deceleration ramp-up segment (using JERKMIN3) to maximum deceleration.

This will slow the speed down until it turns around. Deceleration continues at maximum deceleration until 0 velocity is reached and the direction reverses. At exactly this point, the motor is now 'speeding up'" but in the other direction. From this point, the curve is similar to a normal movement to a target position, except that the whole curve is inverted because the direction has changed. The curve will have an acceleration ramp-up segment (going backwards), it may or may not have a constant acceleration segment, it will have an acceleration ramp-down segment, it may or may not have a constant velocity segment, it will have a deceleration ramp-up segment, it may or may not have a constant deceleration segment, and finally it will have a deceleration ramp-down to the target position.



# PC Software Interface



## Specifics of the User Interface

You should be familiar with the basic functions and terminology of the Microsoft Windows interface, because this manual does not explain the basics, but the specifics of the PC User Interface.

For programming the MCO 305 the VLT® Motion Control Tool MCT 10 is used. With that you also start the integrated APOSS software for developing control programs and for editing curves.

*Projects* can be programmed off line or by means of *Networking* online.

- Online: When MCT 10 has a connection established to the drive, then APOSS will use the drive connection that MCT 10 has already established.
- Offline: All the features that allow to switch drives or connect to multiple drives or to read current parameters are enabled.

The selection of operating mode is done by MCT 10 at start-up of APOSS and can not be changed while APOSS is running.

When APOSS is started by MCT 10, then APOSS will connect to only a single drive. Hence, all the features that allow APOSS to switch drives or connect to multiple drives are disabled.

When neither MCT 10 is used online nor offline APOSS will be used in a Stand-alone Mode.

## The APOSS Window

Each opened window represents an APOSS program which can be connected with a FC 300. Thus, you can open at least the same number of edit windows as the number of controllers you have selected.

Title Bar

Displays number and name of the connected FC 300. In the event of an error the error number is also shown in the title bar of the controller which triggered the error.

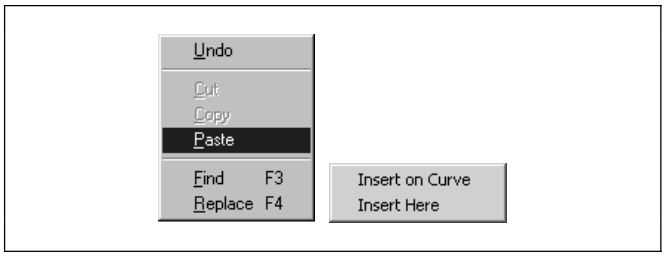
Tool Bar

The tool bar offers beside of the standard functions *New file*, *Open file*, *etc.*, some more: From *Info* to right:
*Select Controller*, *Close Interface*, and *CAM-Editor*.


Context Menus

Context menus are provided at certain program locations if you click on the right mouse button. For example: the context menu in the *Edit* window or a selection menu in *CAM-Editor* for inserting or deleting Fix points.

The context menus are closed automatically when the selected function is being executed or if you click on any other location on the screen with the left mouse button.




Edit Window

In this window you can write your program with the assistance of the functions in the *Edit* menu just like in a text editor. Different colors are used to distinguish between comments, program sections, operators, numbers, etc. You can alter the colors using *Settings* → *Colors Editor*.

Communication Window

The Communication Window is the under part of the Edit Window. It shows the messages from the controller, including the programmed PRINT commands and messages from the compiler.

### □ Keyboard

All keys except the [Esc] key are used exactly as in standard Windows applications, for example arrows, directions keys, etc.

[Esc] key

In addition to the standard functions of the [Esc] key, you can also use this key at any time to abort a program running under APOSS.

**NB!:**
A rotating drive will slow down with the maximum allowed deceleration!

### □ Short Cuts

Keys are often used as shortcuts with other keys either as a key combination or as a key sequence. For a key combination you have to hold the first key depressed while you press the second key, e.g. [Shift] + [Insert], in order to insert the contents of the notepad. For key sequences you can press the keys one after the other, e.g. [Alt] + [O] in order to open the *Edit* menu.

Cut, Copy and Paste

… follows the Windows specifications, for example copying with [Cntl] + [Insert], or [Cntl] + [C].

Cursor Positioning

… follows the Windows specifications also, for example "Go to document end" with [Cntl] + [End], or "Go to line n" with [Cntl] **+** [G].

Expand the Text Marked

… follows the Windows specification closely, for example "Text marked down one line" with [Shift] + [↓] key.

Undo function

You can use [Alt] + [Backspace], or [Cntl] + [Z] to undo the last action.

**NB!:**
*File* → *Save* deletes the Undo memory.

Record Macro

The following short cut could be instrumental in editing: [Cntl] + [Shift] + [R].

### □ Function Keys

Frequently used functions are allocated to the function keys, e.g. with [F12] you can call-up the *Command List* for comfortable programming. Or with [F1] you can access the on-line help. The other function keys will be mentioned at the corresponding situation.

## □ File Menu

The *File* menu contains all the commands to close, save, print, and exit a program. All commands can be used with a mouse click or with the key combination [Alt] and the underlined letter, as it is usual.

*File → New*

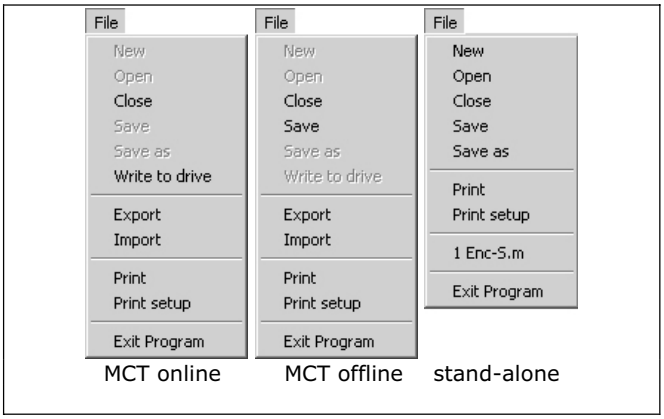
New files must be created via MCT 10.

*File → Open*

Select the file via the MCT 10. APOSS and with that the file is opened automatically.

*File → Save as*

Please use the features of the MCT 10 to rename a program file (*.m) or to copy it..




*Write to drive*

Clicking on → *Write to drive* will compile the current file being edited, make a connection to the drive and download the compiled file to temporary memory on the controller.

If the download was successful, then the program is saved in permanent memory. If MCT 10 requests it, then the source code is also downloaded to the drive.

*Export / Import*

The *Export*/*Import* buttons allows a directly accessing to the .m files in MTC 10 online mode:

Clicking on *File → Export* will display the "Save As" dialog and allow saving the .m file in a directory that the user chooses.

Clicking on *File → Import* will display the "Open File" dialog and allow the user to import a previously saved .m file.

**NB!:**
The imported file will overwrite the incore version of the existing .m file being edited, i.e. it will delete everything that is currently editing and replace it with the contents of the imported file. Use "Quit", if you do not want to overwrite; then the original file, that you edited is still unchanged.

*Exit Program*

The APOSS program can be ended by clicking on → *Exit Program* or on the ☒ icon. If you have not yet saved a new file or changes to an old file then you will have the chance to do this.

**NB!:**
However, *Exit Program* does not end a program running in the controller. You can only abort or end a program with [Esc]. In order to do this the file which is linked with the controller must be open or re-opened.

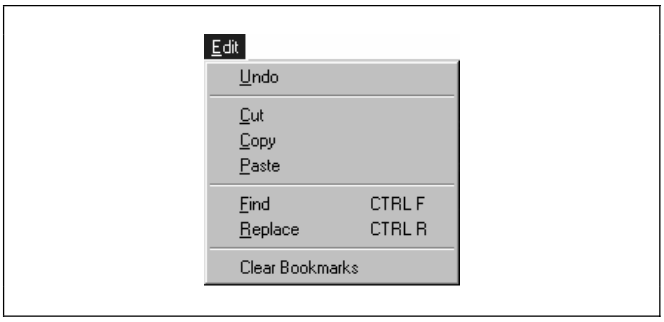
**NB!:**
However, if the controller stops at *File → Exit Program* then this can be due to the fact that the controller is sending PRINT commands which can no longer be displayed by the communications window.

## Edit Menu

The *Edit* menu offers the necessary editing help for programming. Most of these commands can also be reached via certain keys and key combinations, as is usual in Windows.

Some editing help can only be accessed using combinations of keys, e.g.

| | |
|---|---|
| Line-by-line deletion | [Cntl] + [Y] |
| Go to line n | [Cntl] + [G] |
| Line open above | [Cntl] + [Shift] + [N] |




Tabulators

Use the tabulators and the different colors to visually structure your program. The tab increments are permanently set.

Line number

Within your program you can use the line numbers for orientation purposes. For example, the syntax check not only places the cursor in the corresponding line, but also names the line number containing the incorrect command.

The current line number can be found in the status bar, for example 13:1. This means that the cursor is located in line 13 at position 1.

### Find and Replace

Find and replace is realized in accordance with the Windows conventions and supplemented by some useful functions.

Click on *Edit* → *Find* or press [Cntl] **+** [F] and enter the term searched. Use [F3] to jump from one site found to the next one.

Click on → *Mark All* instead of → *Find* and all sites found are immediately marked with a blue triangle at the left margin. You can then jump back and forth with [F2] between the sites found.

Regular Expression

The regular expression implementation in search and replace functionality handles the following syntax, see table:

| | |
|---|---|
| Wildcards | ? (for any character), + (for one or more of something), * (for zero or more of something). |
| Sets of characters | Characters enclosed in square brackets will be treated as an option set. Character ranges may be specified with a – (e.g. [a-c]). |
| Logical OR | Subexpressions may be ORed together with the \| pipe symbol. |
| Parenthesized subexpressions | A regular expression may be enclosed within parentheses and will be treated as a unit. |
| Escape characters | Sequences such as \t, etc. will be substituted for an equivalent single character. \\ represents the backslash. |

### Clear Bookmarks

If bookmarks have been used in the editor, then they are saved and restored along with the program file.

Click on → *Clear Bookmarks* to clear all existing bookmarks from the editor.

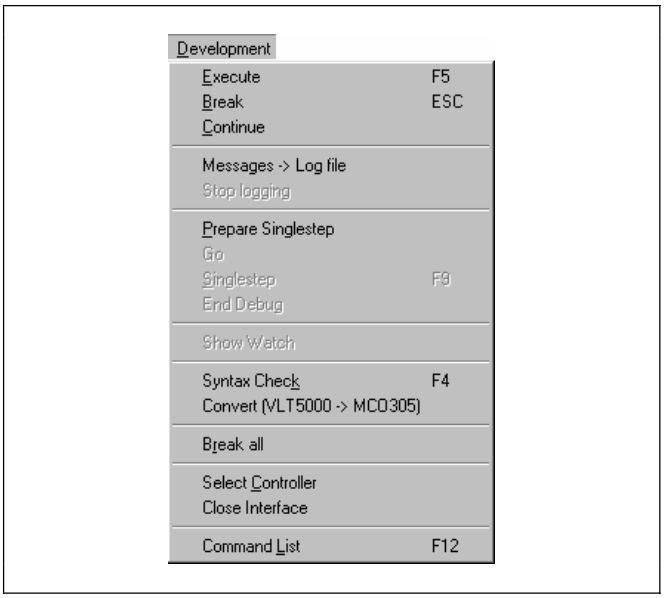
## □ Development Menu

This menu provides functions to run, break, continue, or to run the programs step-by-step for debugging. The debug mode and the possibility to change the variables during program execution make programming easier.

However, before you begin it is always necessary to select a controller or the FC 300.

Several helpful functions can be found in the *Command List*. First of all, this lists a general overview of all APOSS commands, which can be inserted to the edit window immediately. Secondly, you can work here with the teach-in programming.

In offline mode, all functions which require access to the drive can not be used. Most of the *Development* menu items are disabled. APOSS will use the drive connection that MCT 10 has already established.




### □ Execute [F5]

The program that is opened and displayed in the editor will be started.

In doing so the program is compiled and loaded into the FC 300. At the same time the program is loaded into the temporary sector of the RAM; this is overwritten with each subsequent execute command. Thus, when programming you have a quick and uncomplicated process to test the programs.

**NB!:**
However, it is neither possible to move a compiled program back to the computer, nor to edit the source file on the PC again. Thus, in general, you should also save all programs on the hard drive of the computer.

### □ Break [Esc] and Break all

Click on *Development* → *Break* or press [Esc] in order to immediately abort the program. When doing so, it is possible that active motion processes could be ended prematurely.

**NB!:**
Braking is done with the maximum deceleration permitted.

If the program run in several controllers use *Development* → *Break all,* to abort the programs running.

### □ Continue Program

Click on *Development* → *Continue*, in order to continue a program which was just aborted. In doing so any motion processes which were interrupted will be completed.

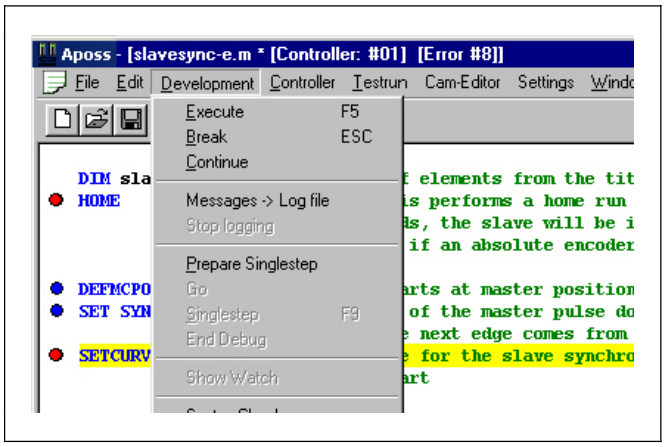
If a program with an error message was aborted, you can → *Continue* it again with this function once you have removed the error and/or erased the error message.

### □ Messages -> Log file

This can be used to start the logging of messages to a file. Note that *Stop logging* will be enabled if logging has been started.

## ☐ Debugging

Single step processing (Tracing) is particularly suitable for testing newly developed programs and can be helpful when searching for errors.




### Prepare Singlestep

Click on *Development* → *Prepare Singlestep* and the program opened is prepared for the debug mode: It is compiled and a debug file is produced, the program is loaded into the FC 300 and all executable program lines are marked by blue dots. Now, also the respective menu items are activated.

Set Breakpoints

By double-clicking you can set a breakpoint before every program line marked with a blue dot. This will be highlighted in red. The program execution will then stop before this program line – being highlighted in yellow – is executed.

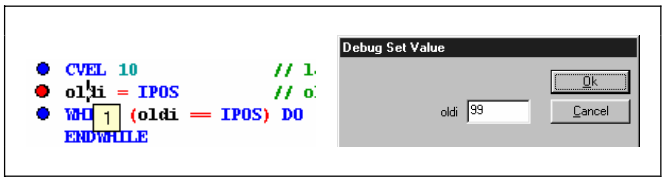
By further double-clicking the red breakpoints are changed again into blue markings for the program lines which shall be skipped while tracing, i.e. no break in debug mode.

**NB!:**
Depending on the speed of the program execution and communication the number of breakpoints should be limited to a reasonable amount. A maximum of 10 breakpoints are allowed.

**NB!:**
ON PERIOD functions should be deactivated during debugging, since the internal timer doesn't stop with *Singlestep*. The program tries to recover the ON PERIOD features later on and this could be problematic.

### Read Variables or Change Variables Online

In the debug mode the current value of the variables after the program execution can be read. Click on the variable with the → left mouse button and the value is displayed until you move the mouse cursor again.




In the debug mode the variables can be changed during program execution. Please observe that such a change should also be considered for the program. Click on the variable with the → right mouse button and set the desired value in the following field.

### Go (Debug) and Singlestep

The program execution stops at the first breakpoint and waits for your input:

Click on *Development* → *Singlestep* or press [F9] in order to execute the next program line.

Or click on *Development* → *Execute* or press [F5] in order to process the program until the next breakpoint.

By pressing [F9] the program will then stop before the next program line, by pressing [F5] before each breakpoint.

Interrupt Program Execution in the Debug Mode

Click on *Development* → *Break* or press [Esc] in order to immediately abort the program execution. When doing so, it is possible that active motion processes could be ended prematurely.

**NB!:**
Braking is done with the maximum deceleration permitted.
Then, the cursor is in the program line which should be executed next. You can continue with *Development* → *Execute* [F5] or → *Singlestep* [F9].

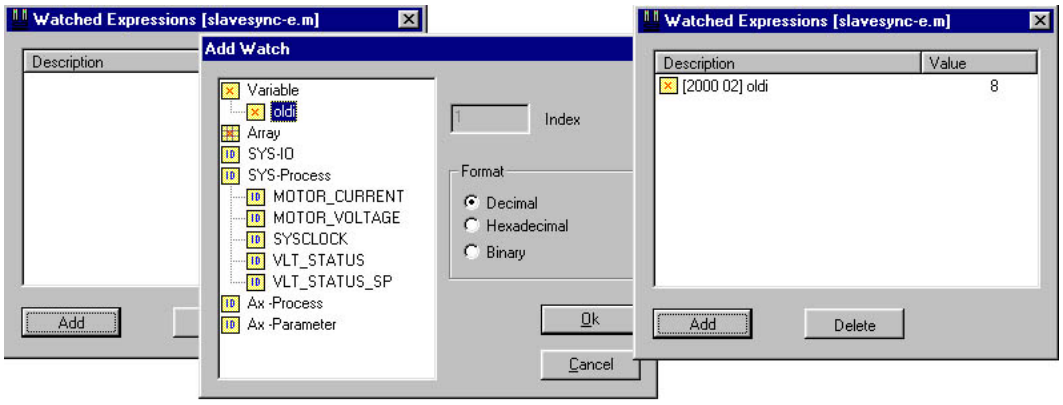
### End Debug

With *Development* → *End Debug* the program execution is ended immediately and you exit the debug mode. The marking of the program lines is removed, but the breakpoints are still displayed so that they can be used again with the next debugging. That means that if you insert program lines, the breakpoints wander along with it.

### □ Show Watch

This function enables online monitoring of the variables, arrays, system and axis processing data (according to the SYSVAR indices) and axis parameters.

Click on *Development* → *Show Watch* and in the following dialog window on → *Add*. In the next dialog window you can choose between the variables, arrays and parameters.




By double-clicking on the desired type, e.g. variable, you can choose between all variables used in the program. Mark the expression to be monitored and select the format in which it should be displayed (decimal, hexadecimal, binary). Then click on *OK*.

You can add more expressions for monitoring → *Add* and, of course, delete again → *Delete*. You can monitor a maximum of 10 expressions simultaneously.

**NB!**
The monitoring window is updated constantly. Thus, depending on the speed of the program execution and communication the number of the monitored expressions should be limited to a reasonable amount.

**NB!**:
The array watching is limited to the first 250 elements.

Change Watch Window

If you want to change the size of the monitoring window (watched expressions), move the cursor to the lower edge of the dialog field and – as soon as the cursor has changed its shape – click and pull the window in the desired direction.

Close Watch window

Click on *Development* → *Close Watch* or on the close symbol in the dialog window. If you open it again later, the previously selected expressions are monitored online and displayed.

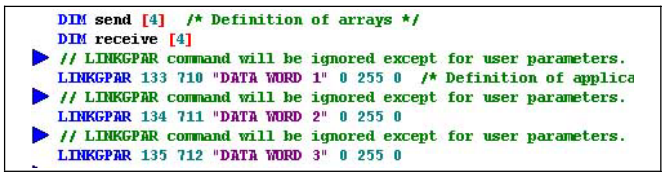
### □ Syntax Check [F4]

The program will be aborted as soon as a faulty command is found. The line number and an error description are outputted to the communications window. The cursor is automatically placed at the exact position of the syntax error and the program stops at this point.

The *Syntax Check* produces a debug file in addition to checking the syntax. This file will be called "temp.ad$".

### □ Convert (VLT5000 -> MCO 305)

This converter checks your previous programs, gives a summery of the required changes, and adds comments where changes have to be done. Note: It does not change your program automatically, see sample with LINKGPAR command.



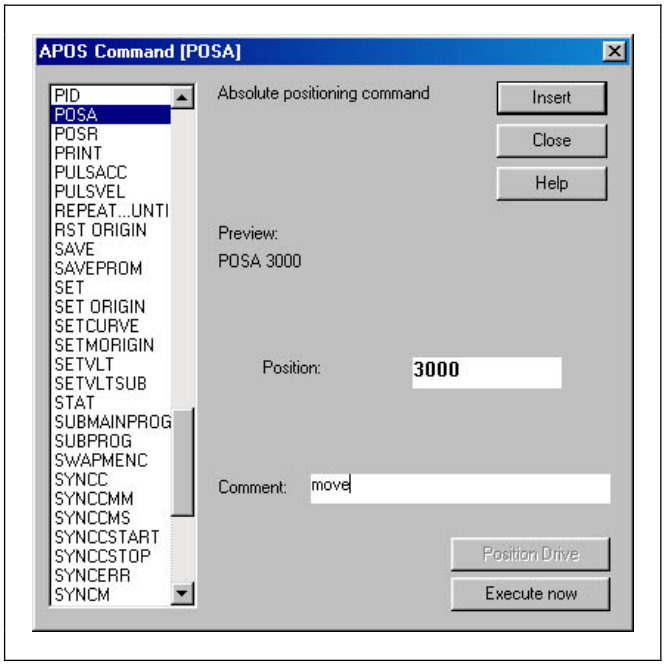

### □ Command List [F12]

The *Command List* offers all commands with their syntax, which you can simply *Insert* into your program. It is also possible to program your controller with teach-in programming.

You can find detailed information on all commands in this dialog field. Simply click on *Help* or press [F1] and you get information to the marked command.

In operation mode offline it is not possible to execute any of the commands directly or to move the drive to position.

Enter the position for the axis in the field. The preview shows you the exact syntax of the command. Now you can select from three alternatives which you can mix randomly to program the FC 300.




**NB!:**
In general, the values entered during programming are not tested whether they are within the permissible range. Due to the multitude of possible applications and various motor sizes this is not possible nor is it desirable.

Insert or Execute now

Move the cursor in the edit window to the position where you want to insert one or more new commands, click on *Development* → *Command List* and select the necessary command in the dialog field, e.g. POSA, complement the value or a comment and click on → *Insert*.

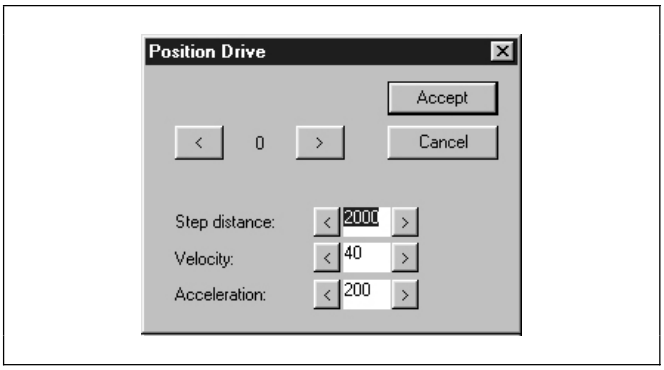
Or click on → *Execute now* and test this command before inserting it in your program.

**NB!:**
Enabled drives start up.

Position Drive

Or you can use the teach-in function and click on *Position drive*: in the dialog field the actual position of the axis is displayed. Click on the forwards > or on the backwards < symbol and move the drive to the position desired. This can be done either step-by-step, with individual mouse clicks, or with continuous movement by holding the mouse button depressed.

Once the drive has reached the desired position, click on *Accept* and the value is entered in the dialog field for the axis.



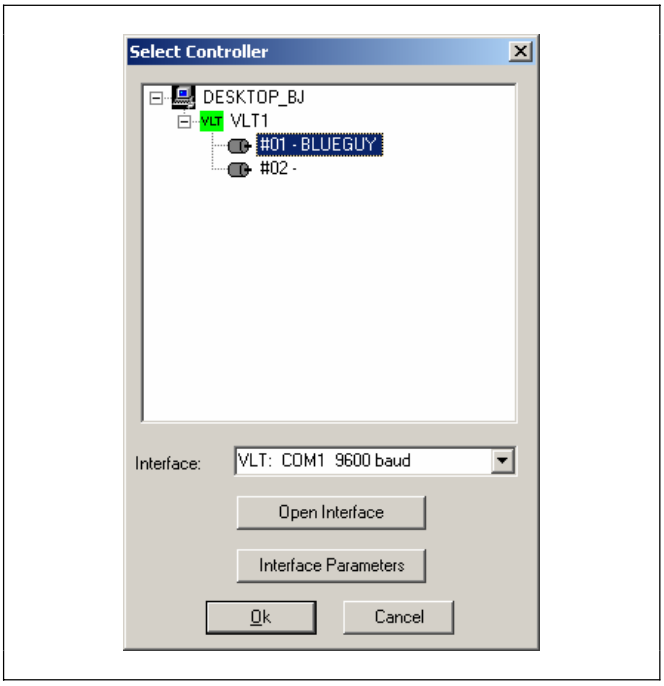

### □ Select Controller

If you have configured more than one FC 300, then use *Development* → *Select controller* to select the FC 300 that you want to use to load and start the programs. All currently available controllers will be displayed in a tree view. Select the desired controller and click on OK to connect to that controller. If no controllers are shown or the desired interface is not present, then select the desired interface from the Interface dropdown box and click on → *Open Interface*.

Run programs in several FC 300s

If you want to load the program into several controllers, link the program with the corresponding FC 300 and click on → *Execute.*

If you want to load a different program in each controller, open a different edit window for each FC 300, then open the desired program file and connect it to the FC 300 with → *Select controller.* Then start each program, one after the other, with → *Execute*.



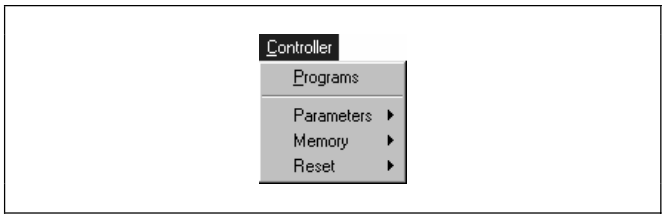

### □ Close Interface

When this menu item is selected, if there is a currently open motor controller interface, then it is closed.

If there is no open interface, then the menu item has no effect.

## Controller Menu

With these functions you can manage your programs: You can save or delete the programs in the option EEPROM and can mark a program for *Autostart*.




### Programs

Click on *Controller* → *Programs* and the dialog shows all the connected controllers. The controller marked is the FC 300 which is currently linked to the program displayed in the edit window. Naturally, you can also mark and edit another FC 300.

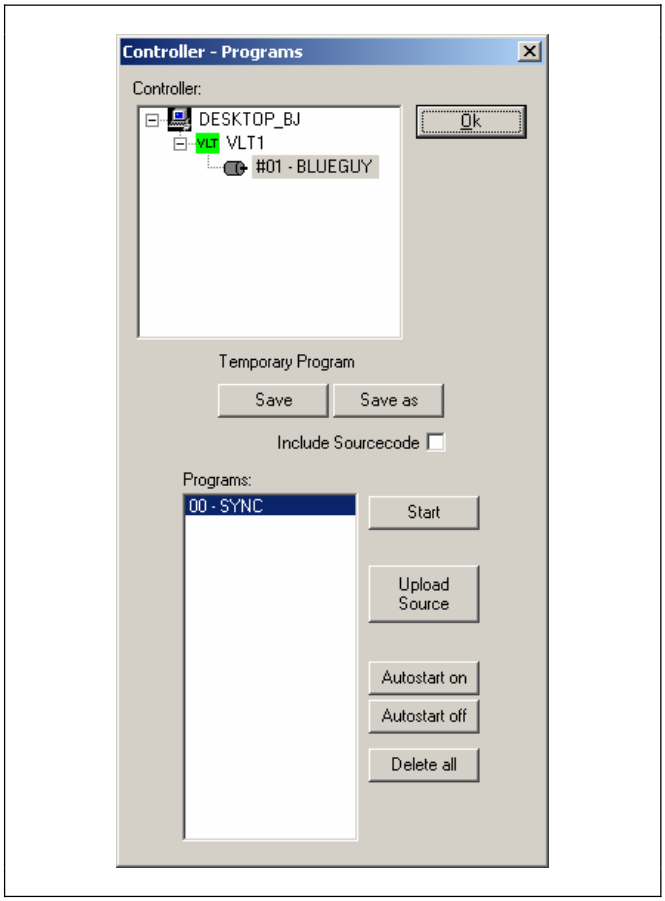
#### Save a Temporary Program

Whenever you execute a program, it is loaded into a temporary sector in the RAM, which is over-written with each subsequent run. Here you can permanently → *Save* the last temporary program executed.

**NB!:**
It is recommended that the non-compiled program file is also saved on the PC hard drive since you can no longer edit the compiled source file in the FC 300.

Click on → *Save* and enter a name in the sub-sequent dialog field or confirm the file name suggested. The program number will be assigned automatically.

**NB!:**
When an attempt is made to save a new program and a program is already active, then the new program cannot be saved. In this case, a dialog is presented that will allow a *Break* to be sent to the currently active program. The new program will then be saved.




#### Programs Save as

Click on → *Save as* and you can also assign the program number (0 to 90) yourself, in addition to the program name.

Using this program number any program can also be started over the inputs, for example from a PLC. For this all inputs are to be set accordingly with *Controller* → *Parameters* → *Global*.

Include Sourcecode

When the check box is activated, the source code is protected in FC 300, in addition to directly executable and compiled program files. This can be re-accessed when necessary and saved in a file on a PC.

When program source code is downloaded, then *Include* files within the source code are expanded and downloaded with the source code. This allows a complete program, rather than only a partial program, to be stored on the controller.

Click on → *Save as* and enter a name in the subsequent dialog field or confirm the file name. The suggested filename contains the date and time to be safe from overwriting actual files in case of uploading.

The source coding is saved in Flash EPROM. If insufficient space is available for the source coding in Flash EPROM a message is displayed and other program files must be erased before saving the new program file.

All programs saved using source coding are marked with a '+' sign.

**Start Programs**

In this dialog field you can select a program and *Start* it directly.

**Upload Source**

All programs marked with '+' can be read out of the control in source coding format and can be filed on your PC for subsequent use.

Select the desired program and click on *Upload Source*. You can edit or duplicate the file for other FC 300's.

**Autostart**

With *Autostart* you can mark a program that, in the future, is to be started immediately after the FC 300 is turned on. Select the desired program and click on *Autostart on*. The program selected will then be marked with a *****.

If you want to remove an *Autostart* command once it has been assigned, click on *Autostart off* or simply mark another program.

If you want to have more than one program run with *Autostart*, use the par. 33-80 *Activated Program Number*. This allows you to determine which program should be started after the conclusion of the program run in *Autostart*.

If nothing else has been determined in the parameters 33-80 PRGPAR, 33-5* I_FUNCTION_n_13, or I_FUNCTION_n_14, the program marked with *Autostart* will always be started.

A pre-set Autostart has the Following Effect:

If no error is registered after a cold start (exceptions: tolerated position error is exceeded, end switch error and SW end switch error) the corresponding *Autostart* program will be started.

If the *Autostart* program is aborted by the user (APOSS) it will not start again unless a new cold start is made. In this case no programs are started due to inputs or par. 33-80 PRGPAR.

If the *Autostart* program is aborted due to an error (since no ON ERROR routine was defined) or ended normally, then the program subsequently checks whether a start is planned through inputs or if the par. 33-80 PRGPAR is set. If so, then the corresponding program is executed or the program waits for the start input. If not, the *Autostart* program starts over again from the beginning. It follows that:

One-time Execution of the Autostart Program

If it is planed in principle to start the programs via the par. 33-80 PRGPAR or via the inputs, then the *Autostart* program is only executed once (for example for HOME functions).

Repeated Execution of the Autostart Program

In all other instances the *Autostart* program is started repeatedly.

Thus, it is also possible to simply start a program over again with an EXIT command. This is useful if, in an error situation (ON ERROR), you do not wish to continue with RETURN, but rather, for example, wish to force a repeated home run. However, it is important to make sure that an error has not occurred (except for a tolerated position error is exceeded, end switch errors and SW end switch errors), since otherwise the *Autostart* program will not be started again.

Linking Autostart Programs

Naturally, starting via the par. 33-80 PRGPAR can also be used for linking purposes: After a program has been started the program number defined by par. 33-80 PRGPAR can be converted. Thus, it is possible to determine which program is to be executed next.

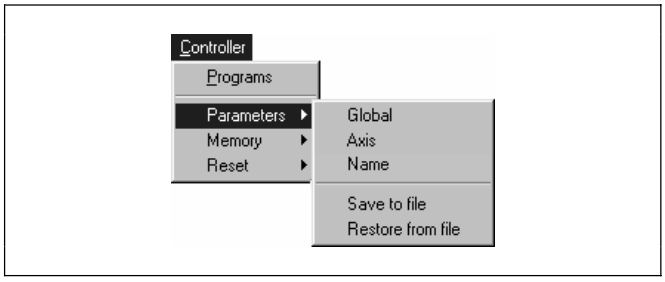
**NB!:**
If no *Autostart* program has been defined, then it is not possible to start a program via par. 33-80 PRGPAR; this always requires a terminated *Autostart* program.

### Delete all Programs

Click on *Delete all* if you want to delete all the programs in the FC 300. Make sure beforehand that you have saved the programs on the PC or in the archive for safety reasons.

### □ Controller > Parameters

The parameters in the *Controller* menu are divided into two groups: the global parameters and the axis parameters. Please see for more details, parameter values, and the default settings the chapter Parameter Reference.
Or simply press [F1] when the mouse cursor is in one of the input fields and you get information about the corresponding parameter.




### Global and Axis Parameters

The global parameters include the input/output functions and the standard parameters which are grouped in 33-5* and 33-8*.

The axis parameters are always valid for all the programs belonging to one controller.

Mark the FC 300 you wish to edit in the dialog field. You can change each pre-set value individually. Click on → *OK* to load the changes in the FC 300.

With *Reset* → *Parameter* in the *Controller* menu can reset all the default settings; however in doing so all parameters, including the axis parameters, are reset to default settings set in the factory.

There are two possibilities to set or change the parameters:

Set or Change Axis Parameters Online

Click on *Controller* → *Parameters* → *Axis* and mark the controller the parameters of which you want to view or change in the subsequent dialog field. Also select the type in the field *Parameters*:

| | |
|---|---|
| Slave Encoder | par. group 32-0* |
| Master Encoder | par. group 32-3* |
| Home | par. group 33-0* |
| Inputs/Outputs (including Limit Handling) | par. group 33-4* and 33-5* |
| PID-Controller | par. group 32-6* |
| Synchronization | par. group 33-1* |
| Velocity | par. group 32-8* |

You can change each parameter and re-load it in the FC 300 by clicking on *OK*. But you can also immediately select another FC 300, change the parameters and then load all the changes into the FC 300 simultaneously with *OK*.

How to Change the Parameters of a Configuration File

Beside the possibility to change online the parameters, you can change all parameter settings of a saved configurations file, too. For this case open the → *CAM Editor* and change the parameter in the corresponding index cards.

**NB!:**
These changes only relate to the CNF-file, but not to the parameter in the controller. To accept the changed settings of the CNF-file into the controller, you have to load the CNF-file into the controller: *Controller → Parameter → Restore from file.*

Reset Parameter

If you want the standard settings for all parameters, simply click on *Controller* → *Reset* → *Parameters*.

**NB!:**
The global and I/O parameters will also be reset to the default settings if you do this.

**Parameters > Name**

You can also enter a name for each FC 300 in addition to the number or change an existing name with this function. Click on *Controller* → *Parameters* → *Name* and select the FC 300, enter a name in the field *Name* (no longer than 8 characters) or overwrite the existing name and click on *OK.*

**Parameters > Save to File and > Restore from File**

Save with *Parameters* → *Save to file* the parameters including the arrays in a file with the extension ".CNF".

This way you can quickly load the parameters in another FC 300 or re-load them in the FC 300 at any subsequent time, for example after deleting the EEPROM.

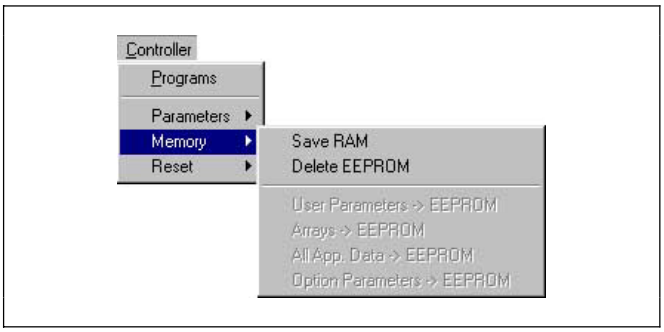
Select the controller and then click on → *Save.* Enter a name in the subsequent dialog field or confirm the file name. The suggested filename contains the date and time to be safe from overwriting actual files in case of restoring from file. If the parameters of multiple controllers are to be saved, then simply select another controller and → *Save* again.

Click on *Parameters* → *Restore from file* and select the file containing the data to be loaded. In the subsequent dialog, select the controller in which the data should be loaded and then click the → *Restore* button. The parameters saved, including the arrays, are immediately loaded in the controller.

If the same data is to be loaded into more than one controller, then simply select another controller and click *Restore* again.

**□ Controller > Memory**

Additional to → *Save RAM* and → *Delete EEPROM* there are functions to save individually data in the LCP memory for example → *Option Parameters.*




**Memory > Save RAM**

The function *Save RAM* is usually not needed since the programs and parameters are automatically saved. But with → *Save RAM* you can also save the current array values in the EEPROM. Save RAM corresponds to the command SAVEPROM, since all programs, parameters and arrays are saved.

**Memory > Delete EEPROM**

Delete the EEPROM in the LCP if you either want to undo the array definition or want to re-set all parameters to the default settings.

**NB!:**
When you delete the EEPROM all parameters are reset to the factory settings. However, this is only done after the FC 300 has been turned off.

**NB!:**
Remember the following when you delete the EEPROM:

1. Check whether you have saved all the necessary programs on the computer so that you can load these into the FC 300 again once the EEPROM has been deleted.
2. Check whether you have saved the parameters for all the FC 300s connected in a file on the computer.
3. Click on *Memory* → *Delete EEPROM*.
4. Re-load the parameters and the necessary programs in the controller or controllers.

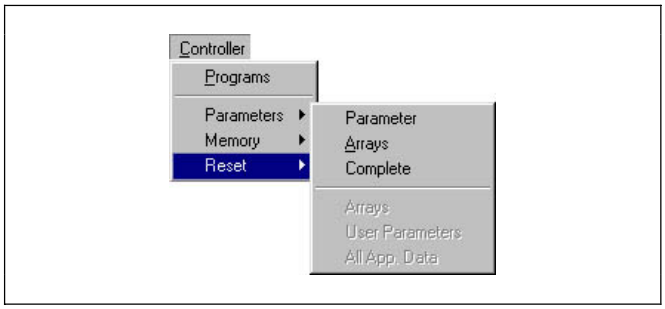
**Memory > Save Part in LCP EEPROM**

Use the corresponding function to save → *User Parameters*, → *Arrays*, → *All Application Data* (contain additional to user parameters and arrays the application programs), or → *Option Parameters* (MCO 305 parameters) individually in the EEPROM of the LCP.

**□ Controller > Reset Parameters, Arrays, or Complete**

With *Reset* → *Parameters* all global parameters and all axis parameters in the MCO are reset to the factory settings.

With *Reset* → *Arrays* you can delete all array definitions in the RAM without deleting the parameters etc. This command has the same effect as the command DELETE ARRAYS.



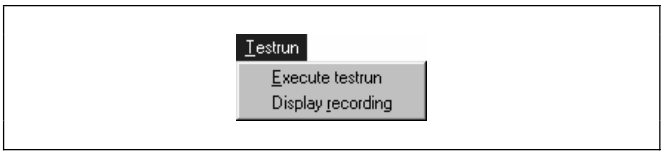

**NB!:**
If you then carry out SAVE ARRAYS, the arrays in the EEPROM are also overwritten!

With *Reset* → *Complete* not only all parameters, but also the programs and arrays are erased and the MCO 305 is reset to the basic default setting ...

**NB!:**
**...** and this happens immediately – not only after the controller has been turned on and off as is the case when you delete EEPROM.

## □ Testrun Menu

The *Testrun* menu offers the functions → *Execute Testrun* from the entry of the test run parameters up to the graphic representation of the test run results.



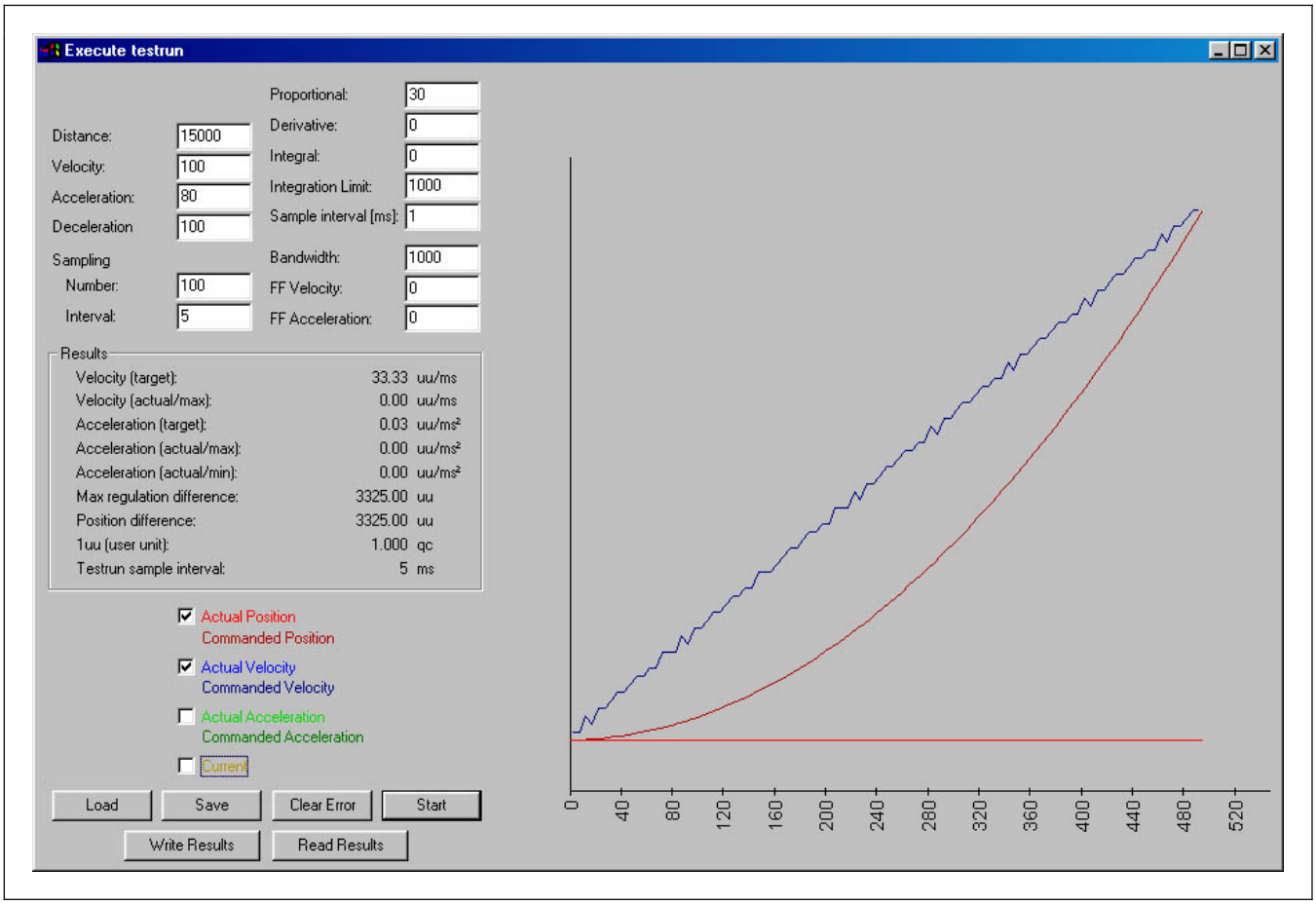

If you have used TESTSETP to define a test run with different parameters, you can also graphically display these results after execution (TESTSTART) with *Testrun → Display Recording*.

In order to correctly set the controller parameters it is important to choose the correct test parameters.

### □ Setting Parameters for a Testrun

Click on *Testrun → Execute Testrun* and enter the test run parameters in the dialog field.

If possible only change one parameter per measurement and check the effect this has.




#### Distance

Set the path of motion in user units and utilize the entire duration of the measurement as well as possible:

The number of measurement points multiplied by the time difference between two measurements result in the entire duration of measurement and thus determines the graph. For an optimal evaluation of the figures the path of motion should be selected so that the end position is reached after approximately 80% of the entire recording time. Thus vibrations at the target position are easily recognizable.

Example: 50 measurements in intervals of 30 ms = 1.5 s recording time

#### Velocity, Acceleration, and Deceleration

The test run parameters of velocity, acceleration and deceleration are entered in percent of the respective maximum value.

Complete the measurements with the values most often needed for the controller for velocity, acceleration and deceleration.

In order to be able to evaluate the vibration behavior of the final velocity you should try to achieve a trapeze-shaped velocity course. For this it may be necessary to increase the sampling interval or reduce the final velocity.

**NB!:**
Before you start to optimize the control behavior, check whether the maximum velocity and the maximum acceleration have been achieved.

#### Sampling Number

The number of readings and the sample interval determine the entire duration of the measurement. 50 to 100 measurement points are sufficient for an optimal graph.

The number of maximum possible measurement points is limited by the internal memory of the MCO 305 and also by any programs that are stored there. If the memory is not sufficient for the desired number of samples then it is necessary to delete the programs stored in the option board with *Controller* → *Programs* → *Delete all*.

#### Sampling Interval

Select a data sampling interval which is suitable for the system and for the frequency converter, for example 20 to 30 ms.

For dynamic applications the sampling interval can be decreased to 1 ms; for servo-motors the shortest possible sampling interval should be set.

Naturally, to record slower or very long motion processes the time difference in milliseconds can be increased, however the maximum is 255 milliseconds.

**NB!:**
The sampling interval mentioned here is the internal between measurements, not the controller sample interval.

### □ Execute Testrun

#### Before Starting a Testrun: Safety Notes

Move the drive to the starting position; it is important to do this before you open the test window.

**NB!:**
While optimizing the controller with the function *Testrun* the drive is automatically returned to the starting position after reaching the target position.

If run in reverse is not allowed, then the par. 32-68 *Reverse Behavior for Slave* must be set to "no reversing" = [2].

Click on *Controller* → *Parameters* → *Axis* and change the setting in the parameter group *PID Controller*.

**NB!:**
Make sure that any and all brakes are released and that there are not any obstacles in the positioning path.

Improperly adjusted controller parameters can cause the motor and the mechanism to be damaged. For this reason never optimize controls without having an EMERGENCY STOP button.

**Starting the Testrun**

Click on *Testrun* → *Execute testrun* and the dialog field with the test run parameters is opened. The test run parameters last saved and the current axis parameters are already entered.

Begin the test series with "stable" controller parameters: If the standard set controller parameters cause the drive to vibrate heavily in the starting position, select a low *Proportional* (32-60) and *Derivative Value* (32-61) (ca. 20) and set the *Integral Factor* (32-62) to zero. Then, starting from these values, optimize the controller.

If it is at all possible you should optimize the controller first with the motor and the drive until the readings are no longer in the critical range. Then connect the mechanical load and complete the fine optimization.

Click on → *Start*: The test run is executed; the current position values, etc. are saved and, at the end of the test run, transferred to the computer for evaluation.

**NB!:**
Observe the behavior and temperature of the motor: if there are strong vibrations or an excessive increase in the motor temperature the motion process must be aborted prematurely with the EMERGENCY STOP button and different controller parameters must be selected.

After making an EMERGENCY STOP you must move the drive back to the starting position before making another test run. Reduce the par. 32-60 *Proportional Factor*, and if necessary also the par. 32-61 *Derivative Value,* before starting the next test run or measurement.

After the measurement is complete all measurement data is automatically transferred to the computer and, during this process, the drive returns to the starting position at a reduced velocity. The figures are plotted automatically.

Clear Error

Clicking this button will clear any currently pending error conditions in the motor controller.

If there are pending error conditions, then the *Start* button will be grayed and a *Testrun* cannot be started.

Saving and Loading Testrun Parameters

Click on *Save*, to save the test run parameters in the FC 300. If, during additional test runs, you receive poorer control results then you can → *Load* the parameters previously used again.

Write and Read Results

Use these function to save the testrun result as ASCII text file. This allows previously written testrun result files to be re-read and displayed.

**□ Display Recording**

If you have used TESTSETP to define a test drive with different parameters, you can also graphically display these results after execution (TESTSTART) with *Testrun* → *Display Recording*.

Insofar as this is possible with the parameters you have selected, i.e. that the result can actually be displayed with the four graphics.

These four diagrams or seven curves are used as follows:

(1) The actual position curve shows the values of index w1 (see TESTSETP),

(2) the set position curve the values of index w2,

(3) and the power curve the values of index w3.

(4) The actual speed curve shows the difference of the recorded data to the values of w1.
In the case that position data is being recorded this means the change in the position in ms = speed.

(5) The set speed curve shows the difference to the values of w2; in the case that position data is being recorded this means the change in the position in ms = speed.

(6) The actual acceleration curve shows the difference to the values of the actual speed (see 4); in the case that position data is being recorded this means the change in the speed in ms = acceleration.

(7) The set acceleration curve shows the difference to the values of the set speed (see 5); in the case that position data is being recorded this means the change in the speed in ms = acceleration.

### ☐ Evaluating Motion Figures

Check the maximum position, the maximum velocity, the number of "overshoots", and the duration of the building-up process.

For each graph the most important test run parameters, the corresponding maximum values and actual values are displayed:

- velocity in user units/ms,
- acceleration in user units/ms²,
- the maximum regulation difference which occurred during the process are shown,
- the actual position difference an the target,
- user units in qc,
- data sample interval in ms.

Click on the corresponding input field to view one of the other graphs, for example for velocity. You can also view two or even all four graphs at once. However, then no units are shown on the x-axis.

#### Testrun Graphs

The Positioning graph shows the set positions (dark or brown curve) and the positions actually achieved (light or red curve).

Velocity graph: The yellow (light) curve shows the achieved velocity path, the brown (dark) curve shows the desired trapeze-shaped set curve.

In special instances the trapeze-shaped velocity curve can degenerate to a triangle shape. This effect occurs when the positioning distance is too short to achieve the maximum velocity at the desired acceleration.

Acceleration graph: The light green curve shows the actual course of acceleration, the dark curve shows the desired trapeze-shaped set curve during acceleration and deceleration.

Current graph: The blue line shows the actual motor current.

## ☐ CAM-Editor Menu

The curve profiles for any CAM controls are realized with the *CAM-Editor*. The individual curves are defined by Fix points, parameters for the engage and disengage motion, as well as parameters for the synchronization with the marker. There are *index cards* in the *CAM-Editor* for this input and for other curve data. The curves and parameters are illustrated in the diagram; in addition, you can also enter and manipulate the Fix points interactively.

### ☐ How to Start the CAM-Editor

You can open configurations files (*.CNF) with MCT 10. This will start the APOSS CAM-Editor.

The file is expected to contain at least one curve. If the file does not contain a curve, then the user is prompted to add a curve to the file. Also, the user may not delete the last curve in the file.

When the CAM-Editor is closed, APOSS is also closed and the user is returned to MCT 10.

The “New CNF”, “Load CNF”, and “Save CNF As” buttons are disabled since all file handling is done by MCT 10.

Before you Begin to Edit a Curve

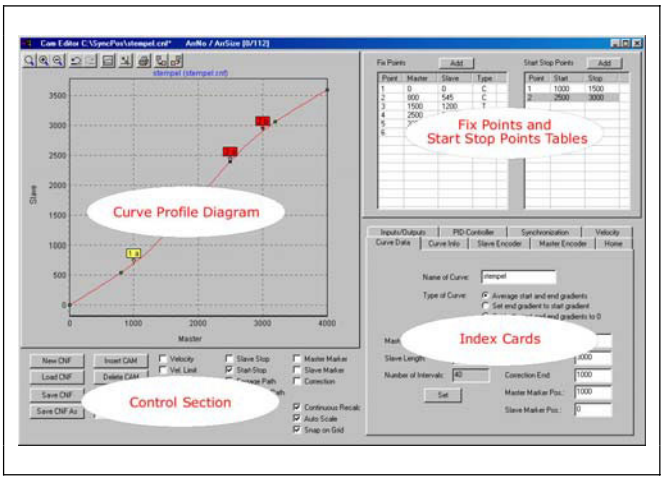
… you should load the parameters of the control as a CNF file into the curve editor because then the parameters will already be entered properly into the corresponding index card. You can get this file using MCT 10 . During this operation, existing arrays, if any, will also be output.

If you do not load a CNF file, the FC 300 will be set to the default settings.

### ☐ CAM-Editor Window

The *CAM-Editor* window is divided into four sectors:

- Curve profile diagram along with the Curve profile toolbar,
- Control section containing checkboxes for displaying and hiding various attributes of the Curve profile diagram and buttons for managing the CNF file as a whole.
- Table of the *Fix points* and *Start Stop Points*,
- Index cards: *Curve Data, Curve Info* and all parameters according to the dialog fields of the APOSS program: *Encoder*, *Home*, *Inputs/Outputs*, *PID-Controller*, *Synchronization* and *Velocity*.




The title bar displays the name of the CNF file and the complete path. The right side of the title bar displays the number (the location) of the array in the CNF file and the number of array elements which you need for the DIM instruction in the APOSS program, for example "Arr.Nr/ArrSize [0/350]".

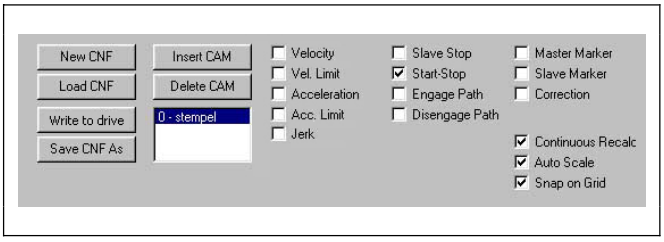
The entire *CAM-Editor* window can be enlarged or reduced as desired by resizing the window in the normal way.

To close the CAM-Editor window, click the normal "Close window" button at the top right corner of the window.

### ☐ CAM Controls

The CAM controls section of the CAM-Editor window contains checkboxes for displaying and hiding various attributes of the Curve profile diagram and buttons for managing the CNF file as a whole.

Please use the online help for more details.




#### Continuous Recalc, Auto-Scale, and Snap on Grid

Continuous Recalc: If this flag is checked, then the curve profiles will be continuously recalculated and redisplayed, as Fix points are dragged in the curve profile diagram.

Auto Scale: If this flag is checked, then the curve profile diagram will be automatically re-scaled so that the entire curve is always visible.
Note that if the diagram has been manually zoomed or scrolled, then this flag will be automatically cleared so that the user's area of interest will remain visible.

Snap on Grid: If this flag is checked, then Fix points will be snapped to the nearest interpolation grid point whenever they are moved. It is recommended that this flag always be checked. Fix points which do not fall on interpolation points, will not lie on the curve profile.
Note that the last fix point is always snapped to the interpolation grid, regardless of this flag, so that the proper relationship can be maintained between the master cycle length and the number of interpolation points.

#### Online Mode: Managing the CNF File

Use the functions of MCT 10 Motion Control Tool for *New*, *Load* and *Save CNF as*.

Write to drive: Click on → *Write to drive* to download the new CNF values (specifically the CAM arrays) to the drive. The new values will also be saved in the MCT 10 database.

### Offline Mode: Managing the CNF File

Use the functions of MCT 10 Motion Control Tool for *New*, *Load* and *Save CNF as*.

Save CNF: The current data is saved as a CNF file, overwriting the previous version of the file (i.e. stores them back into the MCT 10 database).

## □ CAM Profile

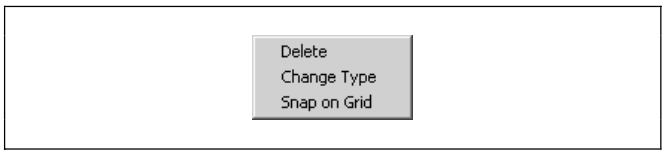
The curve profile section of the CAM-Editor window gives a graphic illustration of the curve, parameters and other information in the curve profile. You are also able to interactively modify and insert the Fix points with the mouse and to enlarge the illustration. A toolbar is available in the upper-left portion of the window and provides other curve-related functions.

A blue header shows the curve name and the file name of the displayed curve.

Fix points can be inserted, deleted, or moved interactively with the mouse using a context menu that will display by double click. Please use the online help for more details.

### How to Change the Point Type in the Diagram

In the curve profile diagram, curve points are shown in green and tangent points in blue. Move the mouse cursor to a fixpoint until the hand icon appears and click on the right mouse button. In the following context menu, select *Change Type*:




If the selected point is a curve point, then both it and the previous point (to the left) are changed to tangent points. If the selected point is a tangent point, then both it and the point at the other end of the tangent segment are changed to curve points.

### How to Zoom or Scroll CAM Profiles

The function follows the Windows specification. Please read the online help for more details.

### Curve Profile Toolbar

The Curve Profile window contains the following toolbar: From left to right:
 Zoom Reset,  Zoom in / out,  Undo / Redo,  Full size,  Snap on Grid,  Print,
  Import / Export ASCII (Please read the online help for more details.)

## □ Using Fix Points

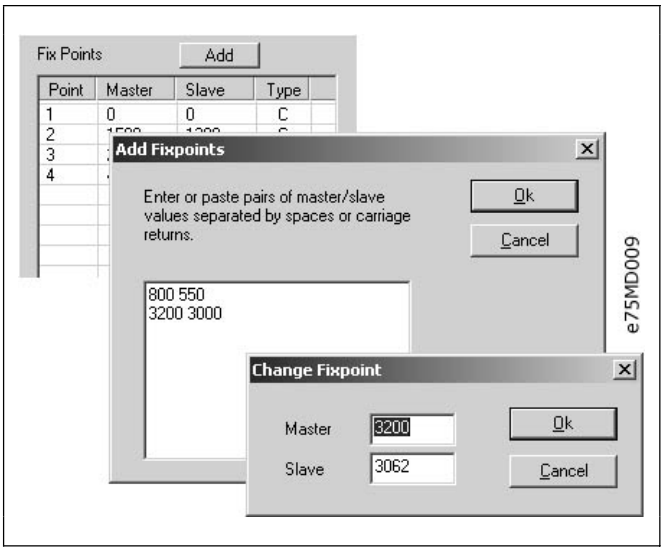
Fix points are used to define the basic shape of the curve over the length of the master cycle. The CAM-Editor will fit a mathematical spline through these Fix points in order to create a smooth curve. This spline is then approximated using many small straight line segments and these line segments are the final result which the controller will use. The line segments are created using a uniform interpolation grid that is determined by the user-specified number of interpolation points.

The master cycle needs to fit exactly onto the interpolation grid so that a whole number of interpolation intervals can be used. Hence, the master cycle length needs to be a multiple of the number of interpolation intervals. Otherwise, numerical errors may cause unexpected behavior. It is recommended that the master cycle length be a value of at least four digits.

The number of interpolation intervals must be large enough so that the curve can be approximated with reasonable accuracy. However, it must also not be so large that the controller begins to have performance problems as a result too many very small segments. The "Interval Time" (found on the Curve Info index card) should not be smaller than 20-30 ms.

When a new curve is created, it will be initially being created with only two Fix points. Additional Fix points can then be added either using the Fix points table or using the mouse in the curve profile diagram. Fix points should always lie on the interpolation grid, so you should always activate → *Snap on Grid*, if possible.

### How to Insert or Delete Fix Points in the Table




Fix Points can be added by clicking the "Add" button in the Fix Points table. This will display the dialog for adding points. Points must be specified as pairs of master and slave values and multiple pairs can be defined at the same time. The Fix points can be entered in any order; they will be automatically ordered as they are being added to the curve.
Note that if *Snap on Grid* has been activated, then the master values will be snapped onto the interpolation grid as they are being added.

Fix points can be deleted by clicking on desired point in the Fix Points table to select it. Then press the [Del] key.

Fix points can be modified by double clicking on the desired point in the Fix Points table. This will display the dialog for modifying points.

### Type: Curve and Tangent Points

The curve will be calculated as spline interpolation between curve points. For sectors where the velocity must be constant and the acceleration = 0, you should use the tangent points. A straight line instead of a spline will be placed between these points.

The CAM-Editor will always try to make a smooth transition from one sector to the next. For this reason, at least one curve point is required between adjacent tangent sectors. This allows the CAM-Editor to generate a spline between the two tangents. Hence, tangent points will always only come in pairs.

To change a sector from a curve to a tangent, simply double click on the "Type" column in the Fix Points table of the second Fix point of the tangent. This point and the previous point will then be changed to tangent points. To change a tangent sector back to a curve, double click on the "Type" column of either of the two tangent Fix points.

You can execute the same action in the diagram: Move the cursor to the point until the hand icon appears. Click on the right mouse button and select → *Change Type*. Here, both points are changed immediately, too.

## □ Using Start Stop Points

Start Stop Points are used to define point pairs for engaging and disengaging the slave during the synchronization. You need one point pair to determine the master position where the synchronization should start and where the engaging should take place. You can determine with an additional point pair from what point the disengaging should be made and where the synchronization should be stopped.

You can define several point pairs (a maximum of 25), for example for multiple starts and stops in a cycle in order to take account of different situations during the start. With the commands SYNCCSTART *pnum* and SYNCCSTOP *pnum slavepos*, you can determine in your program which point pair is to be used.

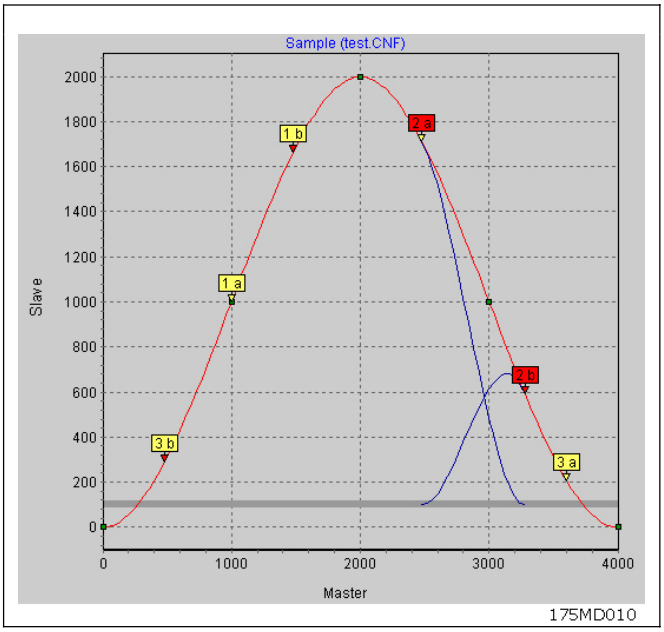
If the Start and Stop points are identical, the slave will be engaged with the set maximum velocity, i.e. without curve, as soon as the master has reached this point.

If the Start point is greater than the Stop point, then the engaging/disengaging operation will "wrap around" the master cycle.

If no start stop points have been defined, the slave will be engaged with the set maximum velocity in the case of SYNCCSTART.

The order of the points in each pair is important and will automatically be taken into account by the run direction: When moving forward, the synchronization begins at the Start point and is finished at the Stop point. When moving backward, it begins at the Stop point and is finished at the Start point.

If the program is closed without the explicit command SYNCCSTOP *pnum slavepos*, the second point pair will always be used for disengaging.

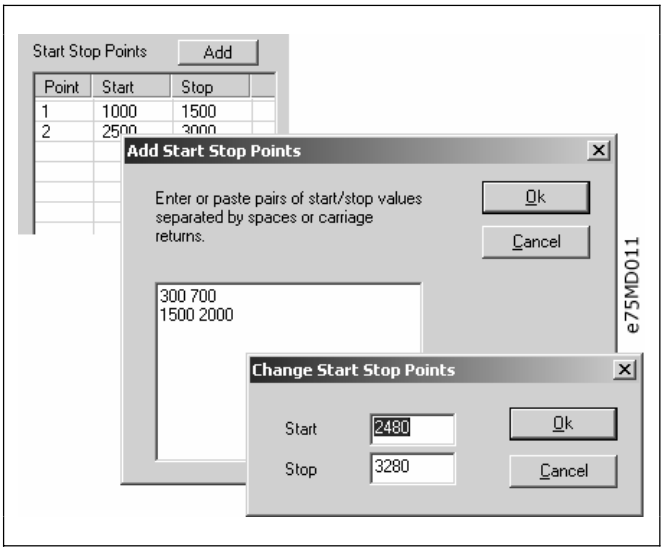



### Adding, Deleting, and Displaying Start Stop Points and Progression

Start Stop points can be added by clicking the "Add" button in the Start Stop Points table. This will display the dialog for adding points. Points must be specified as pairs of start and stop values and multiple pairs can be defined at the same time. Unlike Fix points which must be ordered, Start Stop pairs can be entered in any order.

Start Stop points can be deleted by clicking on desired point in the Start Stop Points table to select it. Then press the [Del] key.

Start Stop points can be modified by double clicking on the desired point in the Start Stop Points table. This will display the dialog for modifying points:




You can visually display the Start Stop points and the engage and disengage curves in the curve profile. Activate → ☑ *Start Stop*. Yellow flags indicate the point pairs for the engaging and disengaging in the synchronization. The Start point will be labeled with "a" and have a yellow arrow. The Stop point will be labeled with "b" and have a red arrow. If a Start Stop pair has been selected by clicking on it in the Start Stop Points table, then that pair will be displayed with red flags.

In order to display the engage and disengage curves activate either → *Engage Path,* or → *Disengage Path,* or both. Then select the Start Stop pair of interest by clicking on it in the Start Stop Points table.

**NB!:**
When the master moves forward engaging is shown by the engage path and disengaging is shown by the disengage path.
When the master moves backward, engaging is shown by the *Disengage* path and disengaging is shown by the *Engage* path!

### □ Index cards Curve Data, Curve Info, and Parameter

Before you edit a curve, you should always first load the parameters of your control into the *CAM-Editor*. You can save the parameters including the arrays into a CNF-file with MCT 10.

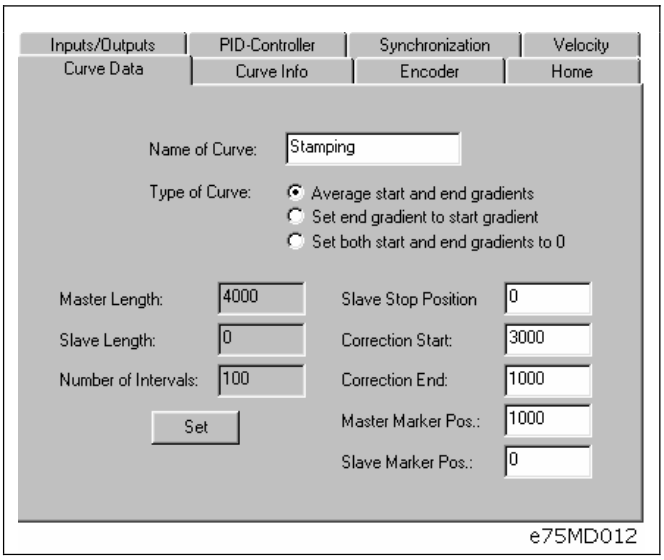
If you do not load any parameters, you will find that the FC 300 default settings have been entered.

If you change the parameters in the course of creating the curve, they will also be saved in the CNF file and can be loaded into the control with MCT 10 or in Stand-alone Mode with *Parameter → Restore from file*.

### □ Index Card Curve Data

You can determine important key data of your curve in the index card → *Curve Data*:

If you edit several curves, you can give meaningful names to the curves for your own information in *Name of Curve*.




#### Type of Curve

In order to prevent velocity leaps in the case of repeated curve cycles, you can choose between three curve types. In each case, the interpolation takes account of the gradient of the curve at the beginning and end.

Select the curve type:

- The gradient of the curve at the beginning and end is averaged.
- The gradient at the beginning of the curve is also used for the end of the curve.
- The gradient at both the beginning and end are set to 0.

#### Setting the Master Length and Number of Intervals

The master length and number of intervals can be set in three ways:

1. Changing the last fixpoint by double clicking on it in the Fix Points table.
2. Moving the last Fix point using the mouse in the curve profile diagram.
3. Pressing the "Set" button in the Curve Data index card.

Both of the first two methods change the master cycle length directly. When changed in this way, the number of interpolation points is changed automatically so that the interpolation interval (the distance between interpolation points) remains unchanged. The last Fix points is always snapped to the interpolation grid in order to avoid a fractional number of interpolation points.

If the "Set" button in the *Curve Data* index card is pressed, then a dialog is displayed that allows both the master cycle length and the number of interpolation intervals to be changed simultaneously. If the change is made this way, then the length of the interpolation interval is recalculated.
Note that this may result in some Fix points ending up off-grid, so it is recommended that the *Snap on Grid* toolbar button also be pressed.

Note that the user is prevented from shortening the master cycle length beyond certain limits. The last fixpoint cannot be moved to a position prior to

- the previous fixpoint,
- any Start Stop point,
- either of the correction start or end points,
- or the master marker position.

If it is necessary to move the last fixpoint beyond these limits, then it is necessary to first change the limiting points/positions and then to move the last fixpoint.

### Slave Stop Position

Determine the position where the slave should run to and stop if no SYNCCSTOP *pnum slavepos* command with the variable *slavepos* was set in the program.

This position will also be used if SYNCC starts with a specific number of cycles and does not use a SYNCCSTOP command.

A grey line indicates this position in the curve profile. Activate → ☑ *Slave Stop* for this purpose.

### Correction Start and Correction End

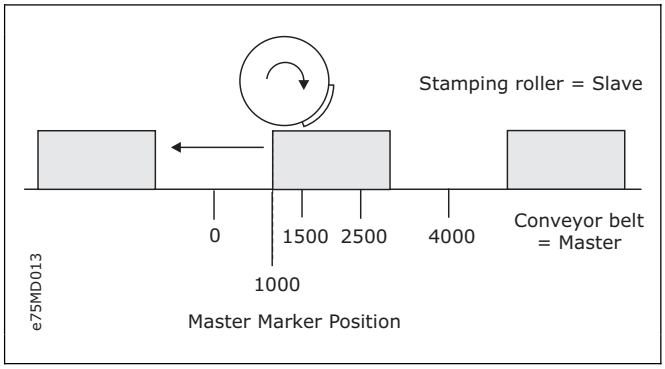
Enter the master positions where the master correction is supposed to begin and where it is supposed to end. Be careful to leave enough time to correct the synchronization before the processing point is reached.

The correction area is shown in blue in the curve profile. Activate → ☑ *Correction* for this purpose.

### Master Marker Position and Slave Marker Position

Enter the master position (or the slave position in the case of a slave synchronization with marker) for which the marker has been set, here for example the beginning of a cardboard box.

The position of the curve where the marker is detected is calculated from the master marker position and the marker distance. This position is shown as a green line in the curve profile and allows you to fix the correction area. Activate → ☑ *Master Marker* or *Slave Marker* for this purpose.




### Master Length and Slave Length

Information about the cycle length of the master or slave set in the table of Fix points.

**NB!:**
Slave length must be positive, this can be ensured by defining *Rotational Direction* in par. 32-10.

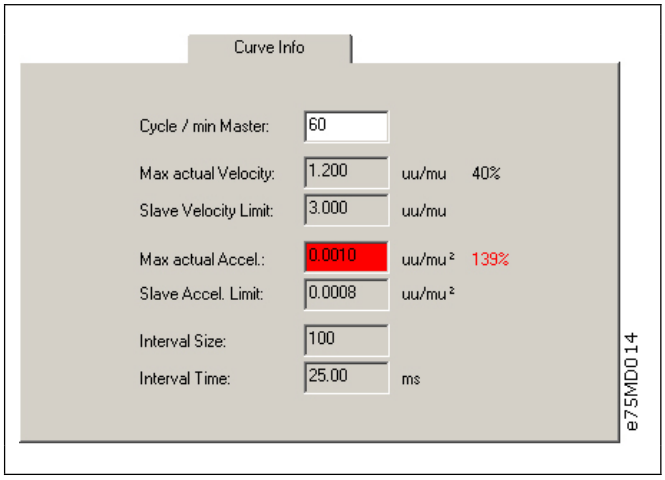
## ☐ Index Card Curve Info

In this index card, you can determine the number of → *Cycles/min Master* in the input field. In the other fields, you can find curve information calculated from the parameters and the curve application.

If the velocity or acceleration exceeds the limit, then it is shown in red as it is shown in this sample.

You can graphically display the velocity and acceleration in the curve profile. Activate the corresponding checkbox, for example → ☑ *Velocity.*

Please see the online help for more details.




### Cycles/min Master

Enter the number of cycles of the master per minute. In most cases, this will be the (maximum) number of products that are processed per minute.

**Interval Size and Interval Time (ms)**

The Interval Size is derived from the number of intervals per master cycle length.

The time in (ms) for an interval is also derived from the number of intervals per master cycle length. It should not be smaller than 30 ms. (30 to 100 ms are suitable values.) Thus, you should make changes in *Curve Data → Number of Intervals* in order to get a reasonable value.

### □ Index Cards Parameters

There are index cards for the parameters *Encoder, Home, Inputs/Outputs, PID-Controller, Synchronization*, and *Velocity*.

Below just the special features of the latter two are explained. Please read the chapter Parameter Reference for more information about the content, units, value range and default setting, or select an input field and press [F1].

**Index Card Synchronization**

Syncfactor Master and Slave

The two parameters 33-10 *Syncfactor Master* and 33-11 *Syncfactor Slave* are used to determine the MU units in the CAM control.

Marker Distance

Enter the distance of the sensor to the processing point here; in the case of master markers in par. 33-17 and in the case of slave markers in par. 33-18.

The position of the curve where the marker is detected is calculated from the master marker position and the marker distance. This position is shown as a green line in the curve profile and allows you to fix the correction area. Activate → ☑ *Master Marker* or *Slave Marker* for this purpose.

Tolerance

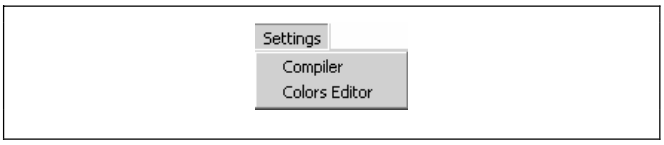
Tolerance window for the appearance of the Master Markers (Marker monitoring) or of the Slave Markers (parameters 33-21 and 33-22). The tolerance window is shown as a green area in the curve profile. Activate → ☑ *Master Marker* or *Slave Marker* for this purpose.

**Index Card Velocity**

In addition, the maximum velocity and acceleration reached in the current application are calculated here in qc/scan time. The display in the curve profile occurs in units. Activate → ☑ *Velocity or Acceleration* for this purpose.

## □ Settings Menu

This menu offers various options and settings depending on operating mode.

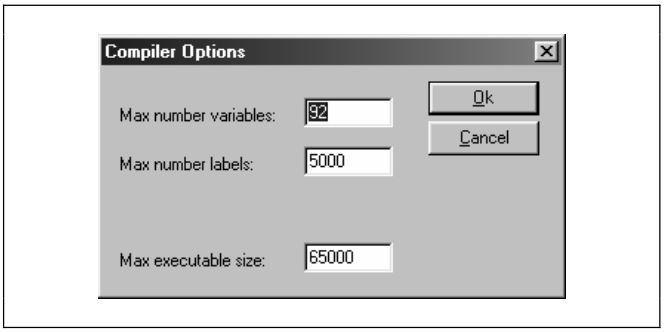



### □ Compiler

The default values for the compiler options are set accordingly for most applications. Thus, they do not require too much memory and, at the same time, they allow the necessary input to be made.

Maximum number of variables

The number of variables has a direct effect on the amount of memory available in the option. It is important to remember that an array also occupies the space of a variable. Increase this number if you need more than 92 variables (incl. arrays).



Maximum number of labels

The maximum number of labels determines how much memory is available for internal hyper-links. Internal hyper-links are automatically created for all branches of the program (GOTO, IF, LOOP, REPEAT, WHILE, GOSUB) during compiling. The recommended range is between 100 and 500 internal labels.

Increase the maximum values permitted if the number of labels is not sufficient for text input.

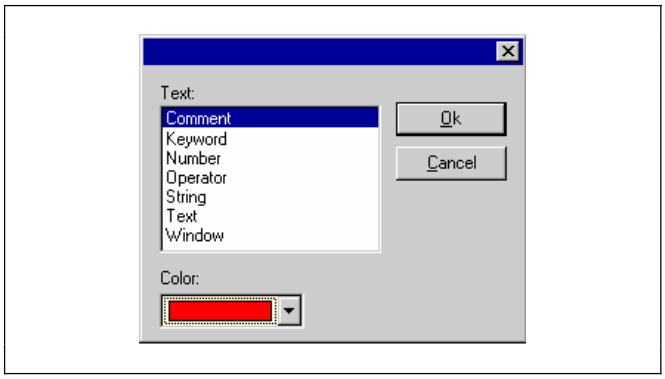
If there is not sufficient memory reduce the maximum number of variables or the labels or increase the maximum values permitted if the number of labels is not sufficient for text input.

Maximum executable size

The maximum executable size specifies the maximum size, in bytes, of a program after it is compiled and ready for download to the controller. Note that some older controllers do not support executable sizes larger than 65,000 bytes.

### □ Colors Editor

In order to provide greater clarity, different colors can be assigned to the various program sections such as comment, key word, number, etc. Select the type, e.g. Comment, and select the desired color. Click OK to store the new settings.




### □ Window and Help Menu

The features of the *Window* menu follows the Windows standards e.g. → *Cascade*, → *Tile vertically* (alongside each other) or *Tile horizontally* (on top of each other) in the *Windows* menu**.**

Help menu

The representation and functionality of the online-help varies, depending on the operating system used, but follows in every system the standard specifications, e.g. → *Index.* A full text search engine is also available.

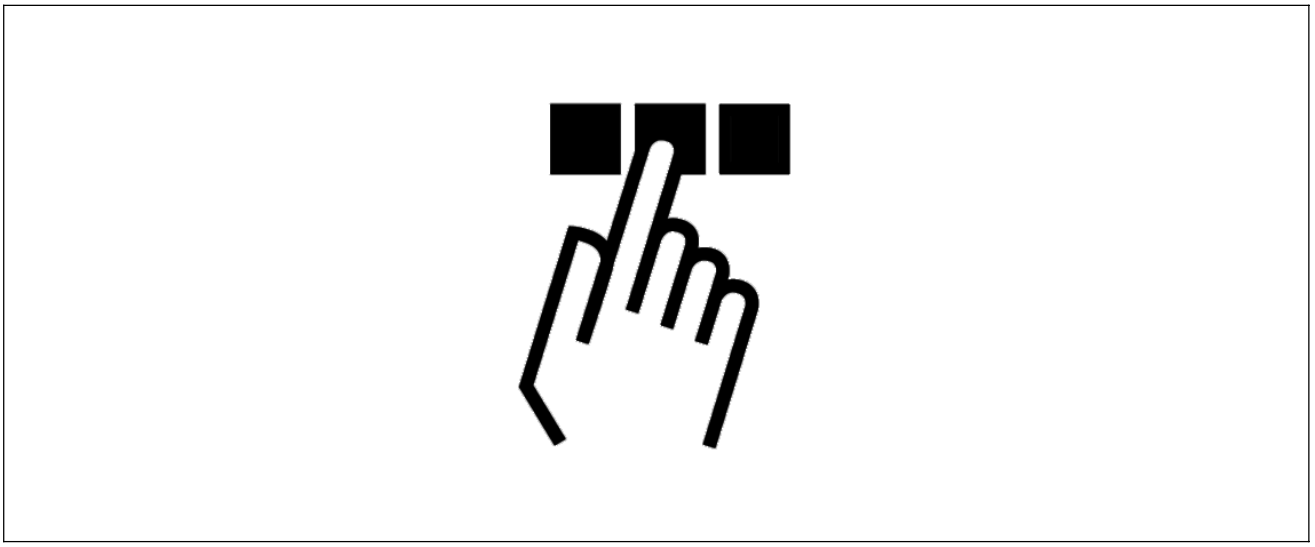
Context-sensitive help

The *Command list* and the parameter dialog fields in the menu *Controller* as well as the *CAM-Editor* offers a direct access to the online help. Mark a command in the *Command list* or select one of the input fields of a parameter and press [F1]. You will get the corresponding information.

About program

Here you can find the version numbers of the APOSS program, the program library, and the compiler.

# How to Program



## Programming the MCO with the APOSS Macro-language

The following chapters describe how to program the FC 300 with MCO 305 using APOSS. Beginners should read the basic explanations on the programming language APOSS, i.e. program layout, command structure, interrupt, elements of the programming language, and arithmetic. Experienced users should inform themselves about the APOSS-specific basic principles, e.g. user units, or parameters.

The *Edit* menu is used to create and comment on the programs. It is particularly easy to write a program by using the *Command List* menu. Once a command is selected, all the necessary input fields are immediately opened. After entering the values the syntax is automatically formed and you can insert the entire command in your program.

With teach-in programming you simply move the axis can be simply moved to the desired position and store the position which has been reached. In this manner you can quickly program the most complicated adjustments and sequences of movements.

All commands are described in the Software Reference, first in a general overview and then alphabetically ordered, in detail and complemented with short examples. You can reconstruct as many as 50 programs with assistance of the information in the program samples in the corresponding chapter in the online-help.

And in chapter Parameter Reference all the parameters are described, first in general and then in detail.

## Program Layout

Usually a program starts with the definitions of arrays, interrupts, and application parameters. For example:

```
DIM send[12], receive[12]     // Array

ON ERROR GOSUB errhandle  // Interrupts
ON INT -1 GOSUB stopprog
ON PERIOD 500 GOSUB calc
ON TIME 10000 GOSUB break

LINKGPAR 1990 "Offset [qc]" 0 100000 0
                  // Application parameters
```

Next step is initializing: setting parameters, flags, and variables. For example:

```
SET POSERR 100000000      // Parameters
SET 1990 10000
SETVLT 205 50

offset = 0                // Flags/variables
sync_flag = 0

VEL 100                   // System parameters
ACC 100
DEC 100
```

Followed by the main program loop:

```
main:
…
GOTO main
```

```
main:
IF (IN 3 == 1) THEN
/* Go into synchronizing mode, if input 3 = 1 */
   GOSUB syncprog
ELSE    /* If input 3 not = 1, run in speed mode */
   GOSUB speedprog
GOTO main
```

Sub program areas are defined as follows:

```
SUBMAINPROG
   SUBPROG name
   …
   RETURN
ENDPROG
```

```
SUBMAINPROG
    SUBPROG syncprog
        IF (sync_flag == 0) THEN
        /* synchronize, if not already synchronized */
            SYNCP
            sync_flag = 1
        ENDIF
    RETURN
    SUBPROG errhandle
        WAITI 18 on
        /* waiting for digital input 18, clear the error */
            sync_flag = 0
        ERRCLR
    RETURN
ENDPROG
```

### □ Sequential Command Processing

In general a command is processed to the end before a new command is begun. This means that for position commands the program waits until the target position has been reached.

Exception: If NOWAIT has been set to ON.

### □ Command Run Times

Timing of the GETVLT and SETVLT commands is dependent on the reading and writing capability of the FC 300. The GETVLT commands are typically 20 ms or faster. The SETVLT command can be rather slow and timing depends on what is going on in the FC 300.

It is not possible to give a maximum time for a read or write command concerning FC 300 parameters. (It may take up to 100, or 500 ms, or even more.)

If the command execution times are critical in an application it is possible to measure the run times of a command sequence under the different operating conditions using the command TIME.

### ◻ Tips for Increasing Program Readability

Use of capital and small initial letters (i.e. all commands capital letters, all variables small).

Placement of spacing between command parts.

Place comments in your program. The comments are between
/* … */ or after //…
/* Begin COMMENT End */ or
// Begin COMMENT End
Inadmissible is: Nesting comments (/* **...** /***...***/ **...** */)

Use of line identification within the loop.

## ◻ Command Structure

All instructions consist of: COMMAND WORD + possible *Parameter***.** A variable can also be used as parameter instead of an absolute number.

| | |
|---|---|
| Example | POSA 10000 |
| or | pos = 10000<br>POSA pos |

### ◻ Input Values

As in other programming languages the values inputted are not tested. Thus, it is the programmer's responsibility to ensure that extreme values do not lead to problems. When searching for such potential problems use the debug mode.

## ◻ Error Handling

Possible errors like timeout, position error, or an emergency stop must be considered and error handling subroutines must be programmed for that. Otherwise the program would be aborted without any possibility to clear the error.

The program sample evaluates an error status by the error number and defines a subroutine (error handler), called if an error occurs.

```
ON ERROR GOSUB errhandle
PRINT "Please create a temporary error status"
PRINT " by pushing a limit switch."
endless:       /* Endless loop */
GOTO endless
/****  SUBROUTINE AREA  ****/
SUBMAINPROG
    SUBPROG errhandle
    /* Evaluate error number and re-set error status */
        PRINT "Current error number: ",ERRNO
        IF (ERRNO == 25) THEN
        /* limit switch or control stop */
           PRINT "HW end limit activated"
        ELSE
          PRINT "It's o.k., although it wasn't a limit switch"
        ENDIF
        PRINT "You can re-set the cause of the error";
        PRINT "within the next 10 seconds"
        DELAY 10000          /* Wait 10 seconds */
        PRINT "Re-set error status ..."
        ERRCLR               /* Re-set error status */
        PRINT "... and program terminated."
        EXIT                 /* program termination */
    RETURN
ENDPROG
```

## □ Debugging

*Development → Messages -> Log file* can be used to start the logging of messages to a file. Note that "Stop logging" will be enabled if logging has been started.

Click on *Development → Syntax Check*. The program will be aborted as soon as a faulty command is found. The line number and an error description are outputted to the communications window. The cursor is automatically placed at the exact position of the syntax error and the program stops at this point.

The *Syntax Check* produces a debug file in addition to checking the syntax. This file will be called "temp.ad$".

Click on *Development → Prepare Singlestep* and the program opened is prepared for the debug mode: It is compiled and a debug file is produced, the program is loaded into the FC 300 and all executable program lines are marked by blue dots. Now, also the respective menu items are activated. Please see the section Debugging on page 58 for more details.

## □ Interrupts

In general there are four types of interrupts:

| | |
|---|---|
| ON INT | Interrupt at the edges of an input |
| ON PERIOD / ON TIME | Interrupt after a certain period of time |
| ON COMBIT / ON STATBIT | Interrupt when Bit n is set |
| ON PARAM | Interrupt when a parameter n is changed |

### □ General Processing of Interrupt Procedures

After every internal APOSS command a query is made whether an interrupt event has occurred. It is important to remember that with every internal APOSS command the compiler creates a command in APOSS machine code.

Thus, for example, a simple command such as:

```
POSA (target + 1000)
```

is broken down into the following APOSS machine code:

```
MOVE target to register 101
MOVE immediate 1000 to register 102
ADDREG register 102 plus register 101 to register 101
POSA axis to register 101
```

Furthermore, for commands which take longer (such as DELAY or WAITAX) the program constantly checks whether an interrupt event has occurred. If this is the case, the command is interrupted and continued once the interrupt has been processed.

**NB!:**
Do not use WAITT in connection with interrupts since the waiting process starts again after the interruption.

### □ Use of Variables within Interrupt Procedures

The example above with the "APOSS machine code" clearly shows that it is necessary to use the utmost testcare when assigning variables within interrupt procedures.

If, for example, in the main program the following assignment is made:

```
target = target + value - 1000
```

this is broken down into a series of APOSS machine code commands and the intermediate results are stored in temporary registers. Only at the end of the sequence is the result stored in "target".

If during the execution of this command an interrupt is triggered and in the corresponding procedure the following command is executed:

target = 0

then, in this instance, problems will arise. This is because after processing the interrupt procedure the program jumps back to the main program and then the intermediate result which still exists is stored in target: Thus, the 0 in *target* is overwritten once again.

### □ ON PERIOD within Interrupt Procedures

In contrast, for ON PERIOD functions the time when the next call instruction should take place is calculated at the start of such a function, thus

START_TIME = TIME + PERIOD.

As soon as this time has been reached the function is executed and subsequently the next start time is calculated with the following formula

START_TIME = START_TIME + PERIOD.

This ensures that the call intervals are really the same since the execution time does not influence the calculation. But this means that the user must make sure that the period of time is actually longer than the execution time as otherwise a "jam" is created. That means that actually only the ON PERIOD function is executed.

### □ Response Times

The existence of an interrupt is checked in a special function which is also used as a watch dog control. For this reason this is generally called up in any procedure which could last somewhat longer and in all loops, etc.

This procedure checks every 1 ms whether such an event exists and, if necessary, sets a corresponding flag. At the latest this flag is detected and evaluated after the current APOSS machine code has been processed.

The response time is the maximum run time of the machine code or 1 ms, whichever is greater.

One exception is the time interrupt (ON TIME / ON PERIOD). This checks whether the time has elapsed every 20 ms. Thus, it is not logical to define an ON PERIOD with less than 20 ms.

**NB!:**
Furthermore, in general, it is important to make sure that interrupt functions do not last too long. Particularly for ON PERIOD functions it is important to ensure that the function does not last longer than the period since otherwise a "jam" of function procedure calls will be created.

### □ Priorities of Interrupts

If two interrupt events should occur simultaneously then the processing is prioritized as follows:

ON INT comes before

ON APOS, ON MAPOS, ON MCPOS before

ON COMBIT before

ON STATBIT before

ON PARAM before

ON TIME / PERIOD, but the other events are not lost.

Within the individual types of interrupts the following is true:

ON INT / ON COMBIT / ON STATBIT

If two (input) interrupts occur simultaneously, then the one with the lower number is executed first, however the other is not lost. After the interrupt procedure is completed the other is called up accordingly.

If the same input or interrupt occurs again while the procedure is being executed this is noted again and subsequently executed.

Thus, an interrupt can only be lost if it occurs twice during the execution of an interrupt procedure.

ON TIME / ON PERIOD

As described above the execution time for every temporal function is stored in an internal structure. For simultaneous execution times the procedure that is first on the list will be executed first. The priority is thus determined by the sequence of the ON PERIOD commands.

ON PARAM

If several of these interrupts occur simultaneously, they are processed according to the sequence of the ON PARAM commands in the program.

### □ Interrupt Nesting

It is not possible for one interrupt to be suspended by another. Accordingly, while one interrupt is being processed a second interrupt cannot be processed. The only exception is the ON ERROR function, which is also possible during the processing of interrupts.

However, an ON ERROR function cannot be suspended by an interrupt.

### □ NOWAIT in Interrupts

In general, during an interrupt NOWAIT is set to ON, that means that the program does not wait for the completion of POSA commands.

This is necessary since otherwise a POSA command cannot be suspended by an interrupt procedure, since this would immediately wait for the arrival of the axis. Thus, if you wish to wait for the arrival of an axis during an interrupt procedure, this must be done explicitly with WAITAX.

## □ Elements of the Programming Language

### □ APOSS Number Formats

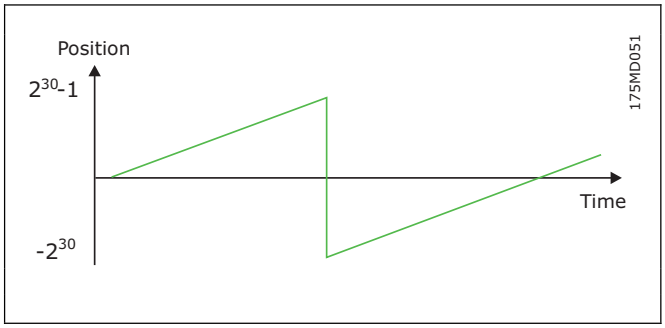
Word (16 bit): $2^{16}$ - 1 = 65535

Long word (32 bit): $-2^{31}$ to $+2^{31}-1$ = -2147483648 to + 2147483647

Positions (32 bit, where 1bit is used to handle overflow):

$-2^{30}$ to $+2^{30}-1$ corresponding to -1073741823 to + 1073741823

Jumping from position 1 billion to position –1 billion when positioning or synchronizing is handled bumpless by the controller and no pulses are lost.



### □ Constants

Constants can be used anywhere where parameters or values are expected. Constants are usually entered in integral numbers, for example: value = 5000

Constants …

- are integer number values between –2 to +2 billion,
- are valid within the entire program (they are global),
- can be entered as a decimal, hexadecimal (0x + hexadecimal number), octal (0 + octal number) or in ASCII (between apostrophes), for example:
  value = 5000 = decimal 5000
  value = 0x7F = decimal 127
  value = 0100 = decimal 64
  value = 'A' = decimal 65
- Hexadecimal and ASCII entries, in particular, avoid many conversions and make the program more readable, for example:
  key = 'A'

The advantage of constants is that they don't need own storage capacity.

### □ Variables

Variables

- can only be used for intermediate data storage of inquiry and calculation results.
- occur via the allocation of a value.
- must not be defined separately.
- are valid within the whole program, (i.e. they are global)
- contain integer number values between –2 to +2 billion.
- can be used within commands, instead of constant values.
- must be allocated a value (4 Byte) before use in a command.

Variable identification names

- can be of any length
- can consist of letters, numerals and underlines
- must not contain any country-specifics, such as “ä”, “é”
- must begin with a letter
- can be written in small or capital letters (no difference!)
- may not be identical to a command name

Special variables: ERRNO = A system variable, which contains the relevant error number

### □ Arrays

Writing programs with dialog requires user input or positions to be stored for a longer period of time, for example, even after the VLT has been turned off. Usually such input consists of several values which are best stored in fields or arrays.

Arrays are stored in the memory area of the user program and are defined globally, that means they are independent of the current program. The user can determine how many arrays are defined and how large the individual arrays should be. This determination is made with the DIM statement and is then fixed and cannot be changed (except by erasing memory). Each program that is intended to use arrays must contain a corresponding DIM statement which corresponds to the original definition. Otherwise an error will be indicated.

#### DIM Statement

The DIM statement has to be the first statement in the program and must appear before the subroutine area.

The DIM statement specifies the arrays to be subsequently used. If no arrays have been previously created then they will be created now. If arrays had been previously defined then it is important that the information corresponds with the original definition.

Program sample
DIM target1[20], target2[20], target3[20], plant_offset[50]
DIM parameter[10]

With these commands a total of 5 arrays have been defined with their corresponding sizes. If this program is executed once then the arrays listed above will be created in the option board. If, when the program is re-started, it is determined that the definition of the arrays differs from the arrays in the option board then this is indicated as an error. However, it is correct if a second program only contains the following line:

Program sample
DIM target1[20], target2[20], target3[20]

However, the sequence of definition must always be the same since the option board does not store the names of the arrays but only their position in the DIM statement. Thus the following program line is also correct and the xpos array is identical to the target1 array.

Program sample
DIM xpos[20], ypos[20], zpos[20], offs[50]

#### Indexes

The elements of an array are designated by a corresponding index in square brackets: xpos[5]. Indexes are allowed from 1 to the size of the array defined. Thus, in the above case for xpos from 1 to 20. If an attempt is made to access elements before or after this array then an error message is generated since this could lead to data overrun and destruction of the array.

#### Reading and Writing Arrays

Access to the arrays thus defined is made analog to the use of variables. Thus all of the following statements in the program sample are correct:

```
xpos [1] = 10000
xpos [2] = 20000
xpos [3] = 30000
i = 1
WHILE (i<20) DO
    ypos [i] = i*1000
    i = i+1
ENDWHILE
zpos [1] = APOS
POSA xpos [1]
offs [1] = (xpos[2]) % 20
```

#### Arrays versus Variables

In general arrays can be used everywhere variables are also permitted. Furthermore, an array only occupies the location of an internal variable and thus only reduces the number of maximally permitted variables by one. The maximum number of variables can be set in the menu *Settings* → *Compiler*.

## □ Arithmetic, Operators

The compiler offers the following commands and parameters:

| | |
|---|---|
| Operators | plus, minus, times, divided by, XOR, Modulo, Division, Absolute amount |
| Bit operators | and, or, invert, left shift, rightshift, bit, byte, word, long |
| Comparison Operations | greater than, less than, greater than or equal to, less than or equal to, the same as, not equal |
| Logical Operations | and, or, not |

Inform yourself about the type of assignment operation which is structured in accordance with the Bit/Byte commands and about the priorities of the operators and the operations.

**NB!:**
All arithmetical operations are integer number operations.

### □ Operators

| Symbol | Meaning | Syntax / Example | Description |
|---|---|---|---|
| + | plus | 3 + 3 = 6 | Addition |
| – | minus | 9 – 3 = 6 | Subtraction |
| * | times | 2 * 3 = 6 | Multiplication |
| % | divided by | 19 % 3 = 6 | Division (result is truncated) |
| ^ | XOR | expr1 ^ expr2<br>127 ^ 255 = 128 | Exclusive Or (binary operation) |
| mod | Modulo | expr1 mod expr2<br>250 mod 16 = 10 | Mathematic modulo (rest of an integer division) |
| rnd | Division | expr1 rnd expr2<br>250 rnd 16 = 16 | Division with round-off (opposite to truncating) |
| abs | Absolute amount | Abs(expr)<br>abs (-5) = 5 | Absolute amount of the expression |

### □ Bit Operators

| Symbol | Meaning | Syntax / Example | Description | Value range |
|---|---|---|---|---|
| & | and | 7 & 6 = 6 | bit-by-bit relationship | |
| \| | or | 2 \| 4 = 6 | bit-by-bit relationship | |
| ~ | invert | ~(–7) = 6 | bit-by-bit inversion | |
| << | left shift | 3 << 1 = 6 | bit-by-bit shift to the left | |
| >> | right shift | 12 >> 1 = 6 | bit-by-bit shift to the right | |
| . | Bit | expr1.expr2<br>7.1 = 1<br>7.3 = 1<br>7.4 = 0 | Returns the Bit expr2 from expr1 | 1 - 32 |
| .b | Byte | expr1.b expr2<br>0x027F.b1 = 127<br>0x027F.b2 = 2 | Returns the Byte expr2 from expr1 | 1 - 4 |
| .w | Word | expr1.w expr2<br>0x0010FFFF.w2 = 16 | Returns the Word expr2 from expr1 | 1 - 2 |
| .l | Long | expr1.l expr2 | Returns the Long expr2 from expr1 (standard) | |

### ☐ Comparison Operations and Logical Operations

| Comparison operations | |
|---|---|
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| == | the same as |
| != | not equal |

| Logical operations | |
|---|---|
| AND | and |
| OR | or |
| NOT | not |

### ☐ Assignment Operation

| Assignment | | Description | Value range |
|---|---|---|---|
| Value | = 0 | Standard assignment to a variable. | |
| Field[1] | = 0 | Standard assignment to an array value. | |
| Value.3 | = 1 | Bit 3 is set at 1, value = 4 | 1 - 32 |
| Field[1].8 | = 1 | Bit 8 is set at 1, field[1] = 128 | 1 - 32 |
| Value.b1 | = 72 | The lowest byte of value is set at 72<br>Value = 72 | 1 - 4 |
| Value.b2 | = 128 | Second byte of value is set at 128<br>Value = 0x00008048 | 1 - 4 |
| Value.w2 | = 15 | Second word of value is set at the value 15.<br>Value = 0x000F8048 | 1 - 2 |

### ☐ Priority of the Operators and the Operations

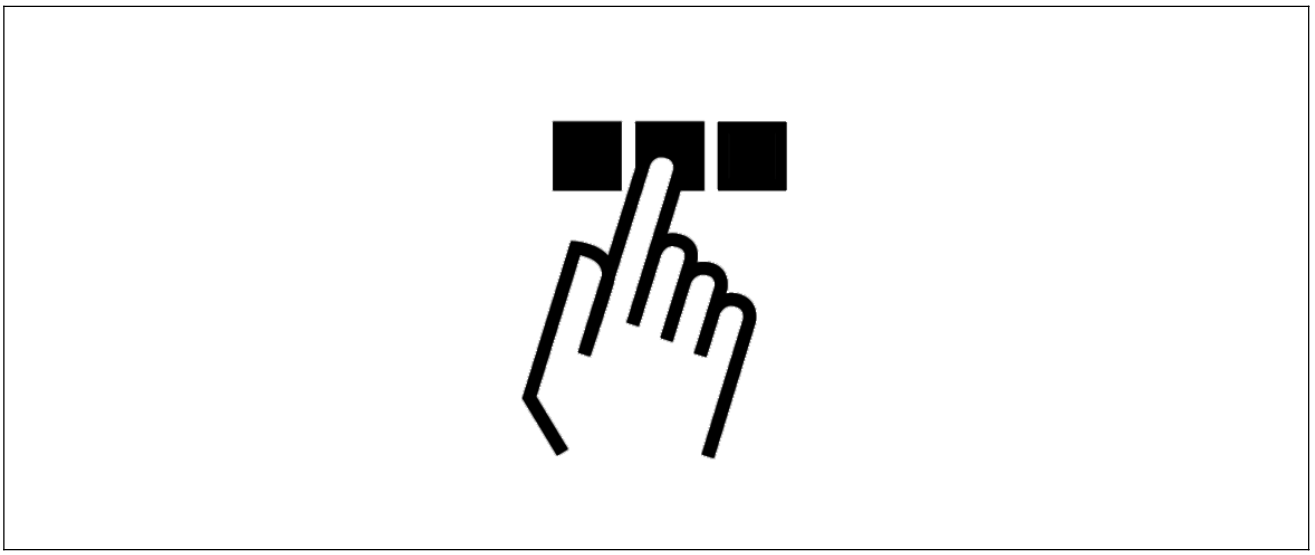
Operators in the same line have the same priority, thus they are completed one after the other.

The priorities are described in decreasing order:

| | | | |
|---|---|---|---|
| * | % | (multiplicative) | |
| + | – | (additive) | |
| >> | << | (bit-by-bit shifting) | |
| >= | <= > < | (relation) | |
| == | != | (equality) | |

| | |
|---|---|
| & | (bit-by-bit and) |
| \| | (bit-by-bit inclusive or) |
| AND | (logical and) |
| OR | (logical or) |

# Software Reference



## Command Overview

All commands are described in a general overview in groups and subsequent alphabetically ordered, in detail and complemented with short examples.

## Initialization Commands

Commands to initialize the axes and MCO 305 start up and define the zero point(s). (Group INI)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| DEFMORIGIN | sets the current master position as the zero point for the master | DEFMORIGIN | – |
| DEF ORIGIN | sets the actual position as zero point | DEF ORIGIN | – |
| DELETE ARRAYS | deletes all arrays in the RAM | DELETE ARRAYS | – |
| ERRCLR | clears error | ERRCLR | – |
| HOME | moves to machine zero point | HOME | – |
| INDEX | moves to next index position | INDEX | – |
| MOTOR OFF | turns off motor control | MOTOR OFF | – |
| MOTOR ON | turns on motor control | MOTOR ON | – |
| RST ORIGIN | resets temporary zero point | RST ORIGIN | – |
| SETMORIGIN | set the current position as the zero point for the master | SETMORIGIN value | value = absolute position |
| SET ORIGIN | sets temporary zero point | SET ORIGIN p | p = absolute position |
| SAVE ARRAYS | save arrays in the EEPROM | SAVE ARRAYS | – |
| SAVE AXPARS | save current axis parameters in the EEPROM | SAVE AXPARS | – |
| SAVE GLBPARS | save current global parameters in the EEPROM | SAVE GLBPARS | – |
| SAVEPROM | saves memory in EEPROM | SAVEPROM | |
| SWAPMENC | swap master and slave encoder internally | SWAPMENC s | s = condition ON / OFF |

## □ Control Commands

Commands for controlling the program flow and for structuring the programs. (Group CON)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| CONTINUE | continues positioning from point of interruption, e.g. following a motor-stop | CONTINUE | – |
| DELAY | time delay | DELAY t | t = delay in ms |
| DIM | declaration of a global array | DIM array [n] | array = name<br>n = number of elements |
| EXIT | desired, premature program termination | EXIT | – |
| GOSUB | calling up a subroutine | GOSUB name | name = subroutine name |
| GOTO | jumping within a program | GOTO label | label = target position |
| IF THEN | conditional program execution | IF condition THEN command | condition = branching criteria |
| … ELSE IF THEN | … with conditional alternative branching | ELSEIF condition THEN command | command = one or more program commands |
| … ELSE | … with alternative branching | ELSE command | |
| … ENDIF | end of the conditional program execution | ENDIF | |
| LOOP | repeats loop | LOOP n label | n = number of repetitions<br>label = target position |
| MOTOR STOP | motor-stop with a programmed delay (with ramp) | MOTOR STOP | – |
| NOWAIT | on/off switch for waiting for the command execution | NOWAIT s | s = condition ON / OFF |
| REPEAT | beginning of repeat loop | REPEAT | |
| REPEAT… UNTIL | conditional loop, with an end criteria | UNTIL condition | condition = abort criteria |
| SUBMAINPROG | commencement of the subroutine definition area | SUBMAINPROG | – |
| … ENDPROG | end of the subroutine definition area | ENDPROG | – |
| SUBPROG | begins a subroutine | SUBPROG name | name = subroutine name |
| … RETURN | ends a subroutine | RETURN | – |
| SYSVAR | system variable (pseudo array) reads system values | SYSVAR [n] | n = index |
| WAITAX | waits until target position is reached | WAITAX | – |
| WAITI | waits for input | WAITI n s | n = input number<br>s = expected ON / OFF |
| WAITNDX | waits until the next index position is reached | WAITNDX t | t = timeout in ms |
| WAITP | waits until a certain position is reached | WAITP p | p = absolute position |
| WAITT | time delay in milliseconds | WAITT t | t = delay in ms |
| WHILE … DO | conditional loop with commencement criteria | WHILE condition DO | condition = abort criteria |
| ENDWHILE | end of the loop | ENDWHILE | – |
| #INCLUDE | compiler directive: embedding further data | #INCLUDE file | file = name of the file to be included |

## □ Input/Output Commands

Commands for setting and re-setting the outputs, reading the inputs, reading movement information, reading system data, and entering and outputting user information. (Group I/O)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| APOS | reads actual position | res = APOS | – |
| AVEL | queries actual velocity of an axis | res = AVEL | – |
| AXEND | reads info on status of program execution | res = AXEND | – |
| CPOS | reads set position | res = CPOS | – |
| ERRNO | reads error number | res = ERRNO | – |
| IN | reads input bit-by-bit (individually) | res = IN n | n = input number |
| INB | reads input by bytes (units of 8) | res = INB n | n = input number |
| INAD | reads analog input | res = INAD n | n = number of analog input |
| INKEY | reads key codes of FC 300 | INKEY p | p = 0 (wait for key code)<br>p > 0 (wait for max. p ms)<br>p < 0 (no wait for key code) |
| IPOS | queries last index or marker position of the slave | res = IPOS | – |
| MAPOS | queries actual position of the master | res = MAPOS | – |
| MAVEL | queries actual velocity of the master | res = MAVEL | – |
| MIPOS | queries last index or marker position of the master | res = MIPOS | – |
| OUT | sets digital outputs bit-by-bit (single) | OUT n s | n = output number<br>s = condition ON / OFF |
| OUTAN | sets FC 300 reference | OUTAN v | v = bus reference |
| OUTB | sets digital outputs by bytes (units of 8) | OUTB n v | n = output byte<br>v = value |
| OUTDA | sets FC 300 analog outputs | OUTDA n v | n = output number<br>v = value |
| PID | completes PID calculation | u(n) = PID e(n) | e(n) = actual deviation |
| PRINT | output (display) texts and variables | PRINT i or PRINT i; | i = information |
| PRINT DEV | stops information output | PRINT DEV nn printlist | nn = no for the device<br>0 = standard<br>-1 = no output after that<br>printlist = argument |
| STAT | reads axis status | res = STAT | – |
| SYNCERR | queries actual synchronization error of the slave | res = SYNCERR | – |
| TESTSETP | specifies recording data for test run | TESTSETP ms vi1 vi2 vi3 arrayname | ms = interval in ms<br>vi 1 … 3 = indices of the values to be recorded<br>arrayname = array used for recording |
| TESTSTART | starts the recording of a test run | TESTSTART no | no = number of measurements to be carried out |
| TRACKERR | queries actual position error of an axis | res = TRACKERR | – |

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| TIME | reads internal controller time | res = TIME | – |
| _GETVEL | changes sample time for AVEL and MAVEL | var = _GETVEL t | t = sample time in ms |

## □ Interrupt Functions (INT)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| DISABLE … | locks the execution of interrupts | DISABLE inttyp | inttyp = INT, COMBIT, … |
| ENABLE … | enables locked interrupts | ENABLE inttyp | inttyp = INT, COMBIT, … |
| ON APOS .. | call up a subprogram when the slave position xxx is passed. | ON sign APOS xxx GOSUB name | sign = passing direction<br>xxx = slave position [UU]<br>name = subprogram |
| ON COMBIT .. | call up a subprogram when Bit n of the communication buffer is set | ON COMBIT n GOSUB name | n = bit n of communication buffer<br>name = subprogram |
| ON DELETE .. | deletes a position interrupt: ON APOS, ON MCPOS, or ON MAPOS | ON DELETE pos GOSUB name | pos = value<br>name = subprogram name |
| ON ERROR .. | calls subroutine in the event of an error | ON ERROR GOSUB name | name = subprogram name |
| ON INT . . | calls subroutine depending on input signal | ON INT n GOSUB name | n = input to be monitored<br>name = subprogram |
| ON MAPOS .. | call up a subprogram when the master position xxx [MU] is passed. | ON sign MAPOS xxx GOSUB name | sign = passing direction<br>xxx = master position<br>name = subprogram |
| ON MCPOS .. | call up a subprogram when the master position xxx [MU] is passed. | ON sign MCPOS xxx GOSUB name | sign = passing direction<br>xxx = master position<br>name = subprogram |
| ON PARAM .. | calls up a subprogram when a parameter is altered | ON PARAM n GOSUB name | n = parameter number<br>name = subprogram name |
| ON PERIOD .. | calls subroutine at regular intervals | ON PERIOD n GOSUB name | n > 20 ms (time for re-call)<br>n = 0 (switch off function)<br>name = subprogram |
| ON STATBIT.. | call up a subprogram when bit n of the FC 300 status is set | ON STATBIT n GOSUB name | n = bit n of the FC status<br>name = subprogram |
| ON TIME.. | calls subroutine after single timing | ON TIME n GOSUB name | n = time for re-call<br>name = subprogram |

## □ Commands for Parameter Handling

All parameters with a parameter code can be set and read with the following commands. Parameter code see overview in Parameter Reference. (Group PAR)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| GET | reads parameter values (MCO 305 and application parameters) | res = GET par | par = parameter identification |
| GETVLT | reads FC 300 parameter values | res = GETVLT par | par = parameter number |
| GETVLTSUB | reads FC 300 parameter values with index number | res = GETVLTSUB par indxno | par = parameter number<br>indxno = index number |
| LINKGPAR | links global parameter with LCP display | LINKGPAR parno "text" min max type | parno = LCP par. number<br>text = ASCII text<br>min = min. value<br>max = max. value<br>type = online / offline par. |
| LINKSYSVAR | links system variable with LCP display | LINKSYSVAR indx parno "text" | indx = SYSVAR index<br>parno = LCP par. number<br>text = display text |
| SET | sets parameter values (MCO 305 and application parameter) | SET par v | par = par. identification<br>v = parameter value |
| SETVLT | sets FC 300 parameter values | SETVLT par v | par = parameter number<br>v = parameter value |
| SETVLTSUB | sets FC 300 parameter values with index number | SETVLTSUB par indxno v | par = parameter number<br>indxno = index number<br>v = parameter value |

## □ Communication Option Commands

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| COMOPTSEND | writes in the Communication option buffer | COMOPTSEND no array | no = number of words to be send<br>array = name of an array (at least the size of no) |
| COMOPTGET | reads a Communication option telegram | COMOPTGET no array | no = number of words to be read<br>array = name of an array (at least the size of no) |
| PCD | pseudo array for direct access to the field bus data area | PCD[n] | n = index |

## □ Speed Control Commands

Commands to obtain a permanently driving axis with constant speed. (Group ROT)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| CSTART | starts the continuous movement in RPM mode | CSTART | – |
| CSTOP | stops the drive in speed mode | CSTOP | – |
| CVEL | sets the velocity for speed regulation | CVEL v | v = velocity value |

## □ Positioning Commands

Commands for the absolute and relative positioning of the axis. (Group ABS and REL)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| **Absolute Positioning (ABS)** | | | |
| ACC | Sets acceleration | ACC a | a = acceleration |
| DEC | Sets deceleration | DEC a | a = deceleration |
| POSA | Positions axis absolutely | POSA p | p = position in UU |
| VEL | Sets velocity for relative and absolute motions and set maximum allowed velocity for synchronizing. | VEL v | v = scaled velocity value |
| **Relative Positioning (REL)** | | | |
| ACC | Sets acceleration | ACC a | a = acceleration |
| DEC | Sets deceleration | DEC a | a = deceleration |
| POSR | Positioning relative to the actual position. | POSR d | d = distance to actual position in UU |
| VEL | Sets velocity | VEL v | v = scaled velocity value |

## □ Synchronizing Commands

Commands to synchronize the slave with the master or the master simulation. (Group SYN)

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| DEF SYNCORIGIN | Defines master-slave relation for the next SYNCP or SYNCM command | DEF SYNCORIGIN master slave | master = reference position in qc<br>slave = reference position |
| MOVE SYNCORIGIN | Relative shifting of the origin of synchronization | MOVE SYNCORIGIN mvalue | mvalue = Relative offset |
| PULSACC | Sets acceleration for master simulation | PULSACC a | a = acceleration in Hz/s |
| PULSVEL | Sets velocity for master simulation | PULSVEL v | v = velocity in pulses per second (Hz) |
| SYNCV | Synchronization of velocity | SYNCV | – |
| SYNCP | Synchronization of angle/position | SYNCP | – |
| SYNCM | Synchronization of angle/position with marker correction | SYNCM | – |
| SYNCSTAT | Queries flag for synchronization status | res = SYNCSTAT | |
| SYNCSTATCLR | Resetting of the flags MERR and MHIT | SYNCSTATCLR value | value =<br>8 = SYNCMMHIT<br>16 = SYNCSMHIT<br>32 = SYNCMMERR<br>64 = SYNCSMERR |

## CAM Commands

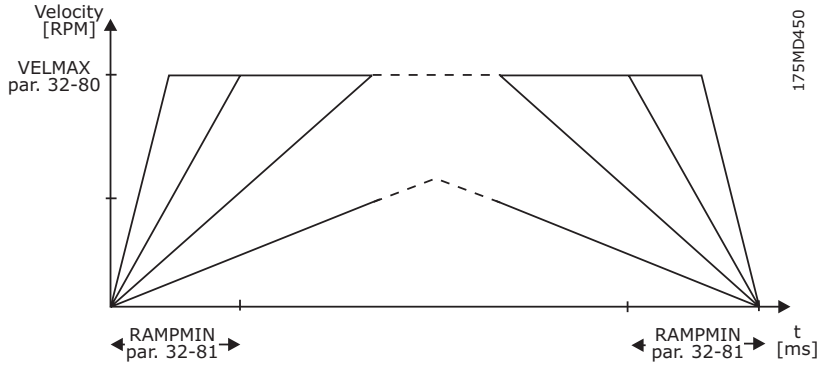
Commands for the synchronization in CAM-Mode (CAM control).

| Command | Description | Syntax | Parameter |
|---|---|---|---|
| CURVEPOS | Retrieve slave curve position that corresponds to the current master position of the curve | res = CURVEPOS | – |
| DEFMCPOS | define initial position of the master | DEFMCPOS p | p = position in MU |
| POSA CURVEPOS | move slave to the curve position corresponding to the master position | POSA CURVEPOS | – |
| SETCURVE | set CAM curve | SETCURVE array | array = array or curve name |
| SYNCC | synchronization in CAM-Mode | SYNCC num | num = number of curves to be processed (0 = drives remains in CAM mode) |
| SYNCCMM | synchronization in CAM-Mode with master marker correction | SYNCCMM num | as above |
| SYNCCMS | synchronization in CAM-Mode with slave marker correction | SYNCCMS num | as above |
| SYNCCSTART | start slave for synchronization in CAM-Mode | SYNCCSTART pnum | pnum = start stop points number |
| SYNCCSTOP | stop slave after the CAM synchronization | SYNCCSTOP pnum slavepos | pnum = start stop points number<br>slavepos = slave position after disengaging |

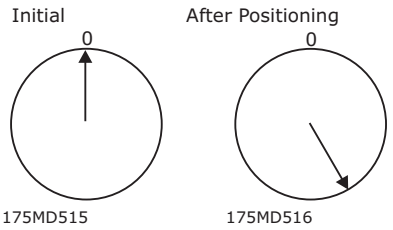
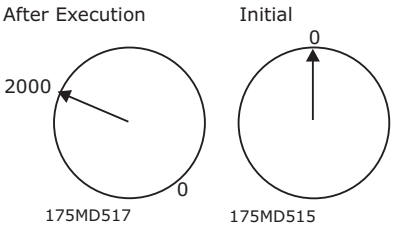
## □ All Commands from ACC to #INCLUDE

In the following section all commands are listed in alphabetical order and described in detail with syntax examples as well as short program samples.

### □ 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,<br>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<br>maximum velocity: 1500 RPM (25 Rev./s)<br>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<br>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.)<br>The user unit (UU) corresponds in standard setting to the number of Quad counts.<br>$\text{Parameter} = \frac{\text{par. 32 - 12 } \textit{User Unit Numerator}}{\text{par. 32 - 11 } \textit{User Unit Denomintor}} = 1$ |
| **Description** | The APOS command can query the absolute position of the axis related to the actual zero point.<br>If a temporary zero point which has been set via SET ORIGIN exists, then the position value is relative to this zero point.<br>**NB!:**<br>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.<br>APOS is affected by the parameters 32-12 and 32-11, and by commands SET ORIGIN p, DEF ORIGIN.<br>Example:<br>POSA 2000<br>PRINT "Actual Position Reached", APOS<br>Output:<br>Actual Position Reached 2000<br>(depending on PID settings a small deviation may occur)<br>  <br>Example with SET ORIGIN<br>SET ORIGIN 2000<br>POSA 2000<br>PRINT "Actual Position", APOS<br>Output:<br>Actual Position 2000<br>  <br>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** | I/O |
| **Cross Index** | CPOS, DEF ORIGIN, SET ORIGIN, POSA, POSR,<br>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 |

### ☐ 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:<br>var = _GETVEL 100<br>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 contrast, quick changes are reported with a delay of a maximum of 100 ms. |
| **Command Group** | I/O |
| **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** I/0

**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

### □ 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*<br>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!:**<br>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.<br>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).<br>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'<br>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.<br>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!:**<br>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.<br>The command position can be queried independently of the operating condition (position control during standstill, positioning process, speed control or synchronization). |
| |  **NB!:**<br>If a set and active temporary zero point (set via SET ORIGIN) exists, then the position value is relative to this zero point. |
| **Command Group** | I/O |
| **Cross Index** | APOS, DEF ORIGIN, SET ORIGIN, POSA, POSR,<br>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 |

### □ 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.<br>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.<br>  <br>**NB!:**<br>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.<br>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 |

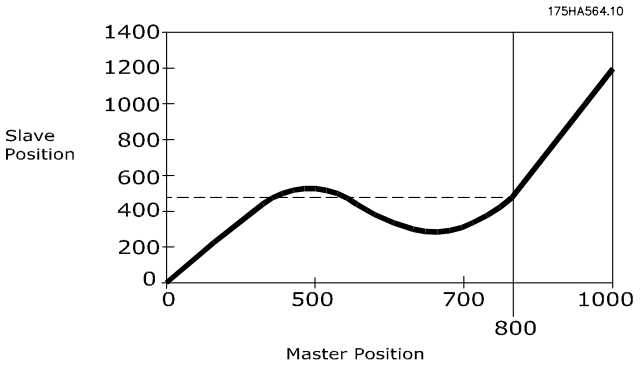
### □ CSTOP

| | |
|---|---|
| **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.<br>  <br>**NB!:**<br>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.<br>The position can be retrieved independently of the operating status (position control at standstill, positioning procedure, velocity control or synchronization).<br>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).<br>After the SYNCC command is stopped, we continue to update these values if *Start Behavior for Sync.* < 2000.<br>**NB!:**<br>The position is only defined if a SETCURVE has been set before.<br>**NB!:**<br>If a temporary zero point exists which has been set with SET ORIGIN and is active, the position value will refer to this zero point.<br>**NB!:**<br>DEFMCPOS and DEFMORIGIN can still modify this position. |
| **Command Group** | CAM |
| **Cross Index** | APOS, DEF ORIGIN, SET ORIGIN, POSA, POSR, DEFMCPOS,<br>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

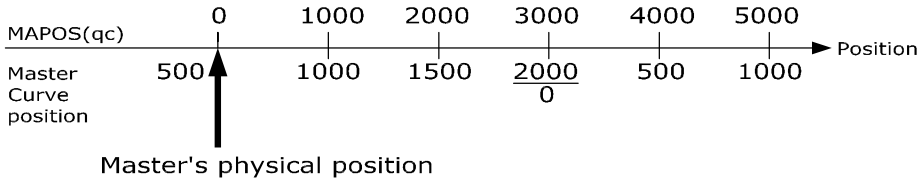
| | |
|---|---|
| **Summary** | sets velocity for speed controlled motor movements |
| **Syntax** | CVEL v |
| **Parameter** | v = velocity value (negative value for reversing)<br>$\text{Command Velocity [RPM]} = V * \frac{\text{par. 32 - 80 Maximum Velocity}}{\text{par. 32 - 83 Velocity Resolution}}$ |
| **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.<br>The velocity value to be given will be related to the parameters 32-80 *Maximum Velocity* and 32-83 *Velocity Resolution*.<br>**NB!:**<br>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.<br>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,<br>Parameter: 32-80 *Maximum Velocity* |
| **Syntax Example** | CVEL 100 |
| **Program Sample** | CMODE_01.M |

### ☐ DEC

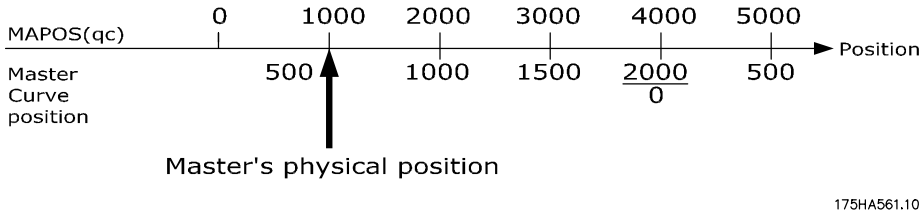
| | |
|---|---|
| **Summary** | sets deceleration |
| **Syntax** | DEC a |
| **Parameter** | a = deceleration |
| **Description** | The DEC command defines the deceleration for the next motion command (speed control synchronization or positioning). This value will remain valid until a new deceleration 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*.<br>**NB!:**<br>If deceleration is not defined previous to the positioning command then deceleration will be the setting of parameter 32-85 *Default Acceleration*.<br>**NB!:**<br>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,<br>Parameters: 32-81 *Shortest Ramp*, 32-80 *Maximum Velocity*, 32-83 *Velocity Resolution* |
| **Syntax Example** | ACC 50 /* acceleration: 50, while braking 10 */<br>DEC 10 |
| **Example** | minimum acceleration time: 1000 ms<br>maximum velocity: 1500 RPM<br>velocity resolution: 100 |

### □ 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,<br>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.



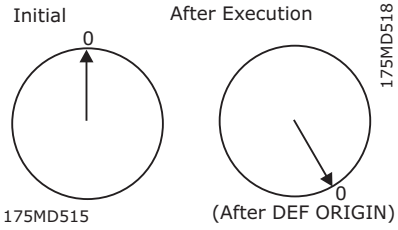

When a DEFMCPOS 500 is issued, master's physical position is made as the position 500 of the curve.

### □ 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!:**<br>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 */ |

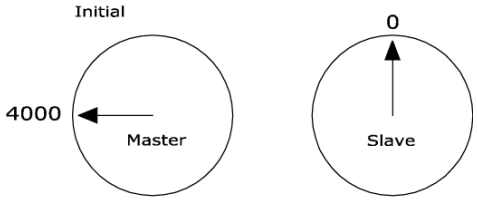
### □ DEF ORIGIN

| | |
|---|---|
| **Summary** | Sets the current position as zero point |
| **Syntax** | DEF ORIGIN |
| **Description** | With the DEF ORIGIN command the current position is set as the zero point. All absolute positioning commands (POSA) then refer to this zero point.<br>The actual position reached in a positioning command is the Target position plus some error which is not compensated automatically while using a DEF ORIGIN.<br>**NB!:**<br>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 */<br>DEF ORIGIN /* define actual position as zero point */ |
| **Sample** | POSA 2000<br>PRINT "Position before new origin", APOS<br>DEF ORIGIN<br>PRINT "Position after defining new origin", APOS<br>Output<br>Position before new origin 2000,<br>Position after defining new origin 0 |
| **Program Sample** | DORIG_01.M, ORIG_01.M |



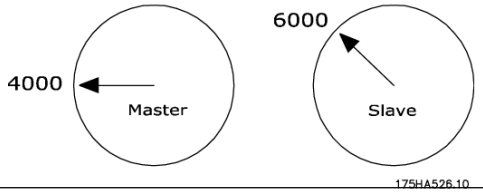

### □ DEF SYNCORIGIN

| | |
|---|---|
| **Summary** | Defines master-slave relation for the next SYNCP or SYNCM command |
| **Syntax** | DEFSYNCORIGIN master slave |
| **Parameter** | master = reference position in qc<br>slave = reference position |
| **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 to the slave value.<br>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.<br>So it is guaranteed, that master and slave will be synchronized at the above master-slave position. |
| **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.<br>Also when the master is in 3000 qc the slave should be in 5000 qc. |




Command; DEFSYNCORIGIN 2000 4000

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



### □ 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.<br>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.<br>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*.<br>  <br>**NB!:**<br>If you then execute a SAVE ARRAYS, the arrays in the EEPROM are also overwritten!<br>  <br>**NB!:**<br>If DELETE ARRAYS is carried out after a DIM assignment in the program, it is then no longer possible to access the array elements.<br>  <br>**NB!:**<br>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<br>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).<br>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.<br>Each array element can later be accessed, similar to a variable, calculation results, characters or other information can be stored.<br>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.<br>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.<br>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.<br>  <br>**NB!:**<br>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.<br>The array content will not be lost, even following switching off the power supply.<br>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.<br>Array definitions can only be canceled via erasure of the entire memory. |
| **Command Group** | CON |
| **Syntax Example** | DIM xpos[100], ypos[100]<br>/* 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!:**<br>DISABLE cannot be called up during interrupt procedures. (The system automatically switches back to enabled after an interrupt.) |
| **Parameter** | inttyp = ALL, INT, COMBIT, STATBIT, PARAM, PERIOD, TIME, POSINT |
| | **NB!:**<br>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.<br>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!:**<br>If an interrupt is disabled it still exist, but is not processed anymore (Exception: DISABLE ALL).<br>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.<br>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!:**<br>Exception: DISABLE ALL<br>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)! |
| | <u>DISABLE ALL in combination with selective DISABLE</u><br>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 COMBIT, 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 |
| | **NB!:**<br>ENABLE cannot be called up during interrupt procedures. (The system automatically switches back to enabled after an interrupt.) |
| **Parameter** | inttyp = ALL<br>INT<br>COMBIT<br>STATBIT<br>PARAM<br>PERIOD<br>TIME<br>POSINT (ON APOS, ON MAPOS, ON MCPOS) |
| **Description** | ENABLE switches all or explicitly specified interrupts on again. |
| | **NB!:**<br>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. |
| | 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 COMBIT, ON STATBIT, ON PARAM, ON PERIOD, ON TIME, DISABLE …interrupts |
| **Syntax Examples** | ENABLE ALL /* Switch on all interrupts */<br>ENABLE COMBIT /* Switch on the interrupt for the communication bit */ |

### ☐ ERRCLR

| | |
|---|---|
| **Summary** | Error cancellation |
| **Syntax** | ERRCLR<br>The ERRCLR command should only be used in a subroutine for error handling (see ON ERROR GOSUB). |
| | **NB!:**<br>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.<br>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,<br>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.<br>If, at the time of inquiry no error has occurred, then ERRNO will contain a 0. |
| **Portability** | Standard variable |
| **Command Group** | I/O |
| **Cross Index** | ON ERROR GOSUB, ERRCLR,<br>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.<br>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.<br>After an abort with EXIT, programs marked with *Autostart* will start up again automatically if<br>SET PRGPAR = -1. |
| |  <br>**NB!:**<br>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,<br>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.<br>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.<br>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,<br>Parameter Reference |
| **Syntax Example** | PRINT GET POSLIMIT /* Print-out positive positioning limit */<br>posdiff = GET POSERR /* Read actual setting tolerated position error */<br>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.<br>Since only integer values are transmitted, it is necessary to take the conversion index into consideration when evaluating the return value.<br>Thus an LCP value of 50.0 Hz (par. 16-13 conversion index = –1) is equivalent to a return value of 500.<br>The list of FC 300 parameters with their respective conversion index can be found in the FC 300 Operating Instructions. |
| | **NB!:**<br>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 4-13 /* 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<br>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.<br>Since only integer values are transmitted, it is necessary to take the conversion index into consideration when evaluating the return value.<br>Thus an LCP value of 50.0 Hz (par. 16-13 conversion index = –1) is equivalent to a return value of 500.<br>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<br>// 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.<br>The main program will be continued following the completion of the last subroutine command (RETURN).<br>**NB!:**<br>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 */<br>Command line<br>Command line<br>SUBMAINPROG /* Subroutine testup must be defined */<br>SUBPROG testup<br>Command line 1<br>Command line n<br>RETURN<br>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.<br>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.<br>It is therefore possible to program a continuous loop via the GOTO command.<br>  <br>**NB!:**<br>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 */<br>command line 1<br>command line n<br>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*.<br><br>To achieve accurate positioning *Velocity for Home Motion* should not be higher than 10% of maximum velocity.<br><br>The sign of par. 33-03 *Velocity for Home Motion* determines the direction in which the reference switch is searched.<br><br>When the HOME position is reached, this position will be defined as 0.<br><br>The reference switch can be approached in 4 different ways defined in par. 33-04 *Behavior during Home Motion*:<br><br>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).<br><br>1 = Like 0 but without searching for the index pulse.<br><br>2 = Like 0 but leaving the switch without reversing the direction.<br><br>3 = Like 2 but without searching for the index pulse.<br><br>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.<br><br>  <br>**NB!:**<br>The system must be fitted with a reference switch, when possible with an encoder with an index pulse.<br><br>  <br>**NB!:**<br>The HOME command will also be carried out to the end in the NOWAIT ON mode, before other program processing will be begun.<br><br>Please note that ON PERIOD xx GOSUB xx must be disabled during homing.<br>E.g. ON PERIOD n GOSUB x and the resetting after homing is completed.<br><br>  <br>**NB!:**<br>The command HOME can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type*). |
| **Command Group** | INI |
| **Cross Index** | INDEX, NOWAIT<br>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.<br>(When the conditions are fulfilled, then **...**, else **...**) |
| **Syntax** | IF condition THEN command<br>ELSEIF condition THEN command<br>ELSE command<br>ENDIF |
| **Parameter** | condition = Branching criteria<br>command = one or more program commands |
| **Description** | Conditional program branching can be realized with the IF..ENDIF construction.<br>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.<br>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.<br>The branching conditions that are checked after IF or ELSEIF can be made up of one or more comparison operations.<br>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.<br>The ELSEIF and ELSE instructions can, but do not have to be, contained within an IF ENDIF construction. |
|  | **NB!:**<br>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 */<br>IF (a == 1) THEN /* Variable a = 1, then */<br>command line 1<br>command line n<br>ENDIF<br>/* multiple branch */<br>IF (a == 1 AND b != 1) THEN<br>command lines<br>ELSEIF (a == 2 AND b != 1) THEN<br>command lines<br>ELSEIF (a == 3) THEN<br>command lines<br>ELSE<br>command lines<br>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<br>1 – 10 or 1 – 12 (option inputs)<br>18, 19, 27, 29, 32, 33 |
| **Return Value** | res = input status<br>0 = Low-level or undefined<br>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.<br>The selection of the mode for input 11,12 is done by par. 33-60 IOMODE.<br>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.<br>The inputs 5 and 6 are also used as marker inputs for the master and slave encoders. |
| **Command Group** | I/O |
| **Cross Index** | INB, OUT, OUTB<br>Parameters: 33-60 *Terminal X59/1 and X59/2 Mode*, IOMODE,<br>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 */<br>IF (IN 2) THEN /* if high level on terminal 2, set output 01 */<br>OUT 1 1<br>ELSE<br>OUT 1 0<br>ENDIF |
| **Program Sample** | IN_01.M |

### ☐ INAD

| | |
|---|---|
| **Summary** | Reads analog input |
| **Syntax** | res = INAD n |
| **Parameter** | n = number of the analog input: 53,54 |
| **Return Value** | res = analog value<br>Terminal 53/54: -1000 – 1000 = -10 V – 10 V<br>Terminal 53/54: 0 – 10 V<br>res = 0 – 100 |
| **Description** | The INAD command reads the value of the analog inputs. |
| **Command Group** | I/O |
| **Syntax Example** | an1 = INAD 53<br>PRINT "Analog input 53 " ,an1 |

### □ INB

| | |
|---|---|
| **Summary** | Reads one byte from digital inputs |
| **Syntax** | res = INB n |
| **Parameter** | n = input byte:<br>0 = input 1 (LSB) - 8 (MSB)<br>1 = input 33 (LSB) - 18 (MSB)<br>2 = input 9 - 10 (12) |
| **Return Value** | res = value of the input byte (0 - 255)<br>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.<br>The values reflect the condition of the individual inputs.<br>The definition of the high and low level, as well as the input circuit, can be taken from the FC 300 manual. |
| **Command Group** | I/O |
| **Cross Index** | IN, OUT, OUTB |
| **Syntax Example** | in = INB 0 /* store the condition of the first 8 inputs */ |
| **Example** | IN1 = low, IN2 = high, IN3 = high,<br>all other inputs are low<br>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.<br>  <br>**NB!:**<br>The utilized encoder must have **! ! !** an index channel.<br>  <br>**NB!:**<br>Only encoders with a low active index pulse can be used.<br>If an index pulse is not found within a complete revolution, then an alarm signal occurs.<br>The INDEX command will also be carried out to the end in NOWAIT ON, before further program processing can be begun.<br>  <br>**NB!:**<br>The command INDEX can not be used with absolute encoders (see par. 32-00 *Incremental Signal Type*). |
| **Command Group** | INI |
| **Cross Index** | HOME, POSA, DEF ORIGIN, NOWAIT |
| **Syntax Example** | INDEX /* move to index */ |
| **Program Sample** | INDEX_01.M |

### □ INKEY

| | |
|---|---|
| **Summary** | Reads in a key signal. |
| **Syntax** | INKEY (p) |
| **Parameter** | p is the maximum waiting time, defined ...<br>p = 0 wait for key code<br>p > 0 wait of max. p milliseconds<br>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<br>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.<br>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 */<br>character = INKEY 5000 /* wait max. 5 seconds to input */<br>character = INKEY (-1) /* do not wait for input */ |
| **Program Sample** | INKEY_01.M, EXIT_01.M, WHILE_01.M |

### ☐ 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.<br>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.<br>  <br>**NB!:**<br>If a temporary zero point, set and activated via SET ORIGIN, exists, then the position is respective to this zero point<br>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*.<br>  <br>**NB!:**<br>The trigger signal for the marker position has to be connected mandatory to the input 6.<br>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.<br>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.<br>  <br>**NB!:**<br>The command IPOS can be used with absolute encoders (see par. 32-00 *Incremental Signal Type*) only in connection with an external marker (see par. 33-20 *Slave Marker Type*). |
| **Command Group** | I/O |
| **Cross Index** | CPOS, DEF ORIGIN, SET ORIGIN, POSA, POSR, MIPOS, ON INT<br>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<br>SET SYNCMTYPS 2 // Definition of IPOS latching on positive edge at input 6<br>CVEL 10 // Start moving<br>CSTART // Endless-Loop<br>mainloop: // ...<br>GOTO mainloop<br>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
```

### ☐ 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)<br>text = descriptive text for display; only ASCII text (8-bit) is supported<br>min = minimum value for this parameter<br>max = maximum value for this parameter<br>option = type of parameter<br>0 = offline, i.e. changes are only active after they have been confirmed with [OK].<br>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.<br>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.<br>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 EEPROM.<br>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 */ |

### □ LINKSYSVAR

| | |
|---|---|
| **Summary** | Link system variable with LCP display |
| **Syntax** | LINKSYSVAR indx parno "text" |
| **Parameter** | indx = Index of the system variable SYSVAR<br>parno = LCP-Parameter number 19-00 to 19-99<br>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.<br>If, for example, you are compiling with #DEBUG NOSTOP, link the internal line number with the FC 300 parameter 19-90. Then you can selectively observe the program execution.<br>  **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 19-90 "internal line number"<br>LINKSYSVAR 30 19-91 "Motor voltage" |

### □ LOOP

| | |
|---|---|
| **Summary** | Defined loop repetition |
| **Syntax** | LOOP n label |
| **Parameter** | n = number of loop repetitions<br>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.<br>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.<br>  The label on the target program position must be followed by a colon (:).<br>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 */<br>command line 1<br>command line n<br>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** | I/O |
| **Cross Index** | CPOS, DEF ORIGIN, SET ORIGIN, POSA, POSR,<br>Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator* |
| **Syntax Example** | PRINT MAPOS /* queries actual master position and print to PC */ |

### ☐ 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.<br>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:<br>var = _GETVEL 100<br>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** | I/O |
| **Cross Index** | AVEL |
| **Syntax sample** | PRINT MAVEL /* queries actual velocity of the master and print to PC */ |

### ☐ 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.<br>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*.<br>**NB!:**<br>The trigger signal for the marker position has to be connected mandatory to the input 5.<br>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.<br>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.<br>**NB!:**<br>The command MIPOS can be used with absolute encoders (see par. 32-30 *Incremental Signal Type)* in connection with an external marker (see par. 33-19 *Master Marker Type*). |
| **Command Group** | I/O |
| **Cross Index** | CPOS, DEF ORIGIN, SET ORIGIN, POSA, POSR, ON INT<br>Parameters: 32-12 *User Unit Numerator*, 32-11 *User Unit Denominator*, 33-19 *Master Marker Type* |
| **Syntax Example** | PRINT MIPOS /* print to the PC the last index position of the master */ |
| **Sample** | // Definition Interrupt-Handler<br>ON INT 5 GOSUB master_int<br>// Definition of IPOS-Latching on positive edge at input 5<br>SET SYNCMTYPM 2<br>CVEL 10 // Start moving<br>CSTART // Endless-Loop<br>mainloop: // ...<br>GOTO mainloop<br>SUBMAINPROG<br>SUBPROG master_int<br>int_mpos = MAPOS<br>// Latching MAPOS for testing, how exact it would be …<br>DELAY 2 // Wait 2 ms, to be sure, that MIPOS is correct updated<br>triggered_mpos = MIPOS // Latching IPOS for a later handling etc.<br>// ....<br>// ...<br>PRINT "Interrupt master position: ",int_mpos<br>PRINT "Triggered master position: ",triggered_mpos<br>RETURN<br>ENDPROG |

### ☐ 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

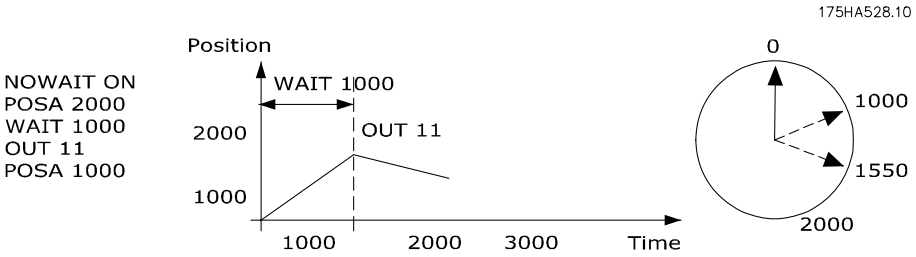
| | |
|---|---|
| **Summary** | Stops the drive |
| **Syntax** | MOTOR STOP |
| **Description** | By using the MOTOR STOP command, a drive in positioning, speed or synchronizing mode can be decelerated with programmed acceleration and arrested at the momentary position.<br>A drive arrested with this 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.<br>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<br>Value range:<br>(–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!:**<br>Valid for position synchronization SYNCP and position synchronization with marker correction SYNCM. |
| **Command Group** | SYN |
| **Cross Index** | SET,<br>Parameters: 33-11 *Synchronization Factor Slave,* SYNCFACTS<br>33-12 *Position Offset for Synchronization,* SYNCPOSOFFS |
| **Syntax Example** | MOVESYNCORIGIN 1000 |

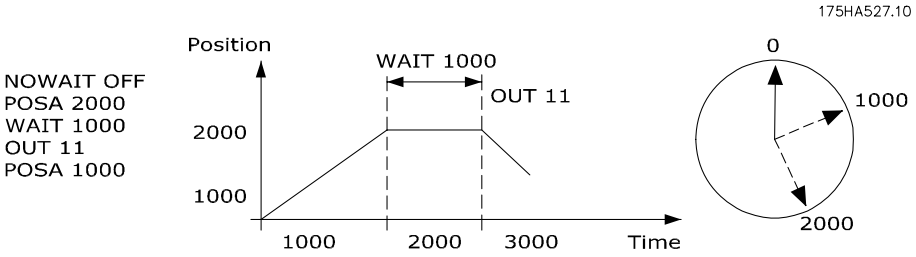
## □ NOWAIT

| | |
|---|---|
| **Summary** | Wait / Do not wait after a POSA/POSR command |
| **Syntax** | NOWAIT s |
| **Parameter** | s = condition:<br>ON = continue program execution while going to target position<br>OFF = hold program execution until target position is reached |
| **Description** | The NOWAIT command defines the program flow for positioning commands.<br>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. |




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. |




Note: The dotted arrows mean the movement positions.

**NB!:**

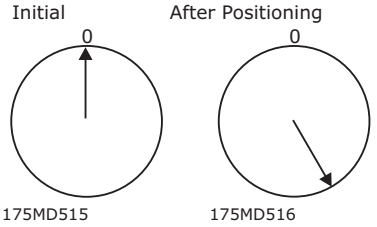
The default condition is NOWAIT OFF, i.e. if no NOWAIT instruction is contained within a program, then the positioning procedure is carried out in its entirety before the processing of the next command is begun.

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, NOWAIT |
| **Syntax Example** | NOWAIT ON /* no waiting after POS-commands */<br>NOWAIT OFF /* wait after POS-commands till target reached */ |
| **Program Samples** | NOWAI_01.M,MSTOP_01.M, OUT_01.M, VEL_01.M |

### □ ON APOS .. GOSUB

| | |
|---|---|
| **Summary** | Call up a subprogram when the slave position xxx is passed. |
| **Syntax** | ON sign APOS xxx GOSUB name |
| **Parameter** | sign = + = when the slave position xxx is passed in positive direction<br>– = when the slave position xxx is passed in negative direction<br>xxx = slave position in UU<br>name = name of the subprogram<br>100 xxx 0<br><–––––– positive direction<br>––––––> negative direction |
| **Description** | It is possible to call up a subprogram with the instruction ON APOS, if a specific slave position (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.<br>**NB!:**<br>– An ON APOS interrupt can be deleted with the command ON DELETE .. GOSUB<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– During the execution of subprograms triggered by an interrupt, NOWAIT ON is set automatically. |
| **Command Group** | INT |
| **Cross Index** | SUBPROG .. RETURN, DISABLE, ENABLE …, Priorities of Interrupts, ON DELETE .. GOSUB, NOWAIT |
| **Syntax Example** | ON –APOS 800 GOSUB name<br>// Call up the subroutine name when slave position 800<br>// is passed in negative direction |
| **Sample** | CSTART<br>ON +APOS 2000 GOSUB STOP<br>SUBMAINPROG<br>SUBPROG STOP<br>CSTOP<br>RETURN<br>ENDPROG<br>  <br>As per the program above the drive stops once it reached the position 2000. |

### □ 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<br>−32 <= n<=32, n!= 0<br>name = name of subprogram |
| | 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!:**<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– 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 ON APOS, ON MAPOS, or ON MCPOS |
| **Syntax** | ON DELETE pos GOSUB name |
| **Parameter** | pos = value<br>name = name of subprogram |
| **Description** | The command can be used to delete an ON APOS interrupt, which is defined as follows:<br>ON sign APOS xxx GOSUB name<br>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!:**<br>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 APOS .., ON MAPOS .., ON MCPOS .., 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.

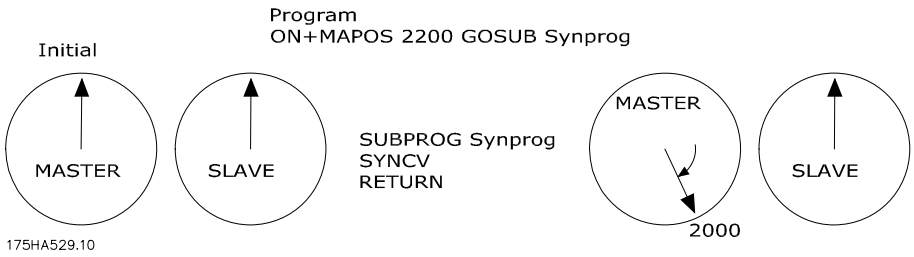
### □ ON ERROR GOSUB

| | |
|---|---|
| **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.<br>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.<br>If the program is not aborted, then the processing will continue from the point where the interruption occurred.<br>By using the CONTINUE command, it is possible to continue the error-interrupted motion. (Exception: synchronization commands)<br>  <br>**NB!:**<br>The ON ERROR GOSUB instruction should be at the start of a program, so that it has validity for the entire program.<br>The subroutine to be called up must be defined within the identified SUBMAINPROG and ENDPROG program.<br>The identification of an error condition and the call up of the corresponding sub-routine requires a maximum of 2 milliseconds.<br>  <br>**NB!:**<br>– Error subroutines cannot be interrupted through any other interrupts.<br>– During the execution of an error routine NOWAIT is automatically set to ON.<br>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.<br>  <br>**NB!:**<br>The ON ERROR GOSUB xx routine does not terminate the HOME and INDEX com-mand. 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 */<br>command lines 1 … n<br>SUBMAINPROG /* subroutine errhandle must be defined */<br>SUBPROG errhandle<br>command lines 1 … n<br>RETURN<br>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; (input area –8 … 8 and FC 300 inputs 18 … 33 and –33 … –18)<br>positive input numbers (1 … 8) = reaction to the rising edge<br>negative input numbers (–1 … 8) = reaction to the falling edge<br>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.<br>A maximum of one subroutine per input can be defined. It is not possible to define an interrupt for the falling and the rising edges of the same input<br>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.<br>  <br>**NB!:**<br>The ON INT GOSUB instruction should be at the start of the program, so that it has validity for the entire program.<br>The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>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.<br>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.<br>  <br>**NB!:**<br>**–** The instruction for ON INT GOSUB is edge and not level triggered.<br>– 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, ON ERROR .. GOSUB, WAITI, DISABLE interrupts, ENABLE interrupts, Priorities of Interrupts, NOWAIT |
| **Syntax Example** | ON INT 4 GOSUB posin /* Definition of Input 4 (positive edge) */<br>ON INT –5 GOSUB negin /* Definition of input 5 (negative edge) */<br>command line 1<br>command line n<br>SUBMAINPROG /* subroutine must be defined */<br>SUBPROG posin<br>command line 1<br>command line n<br>RETURN<br>SUBPROG negin<br>command lines 1 … n<br>RETURN<br>ENDPROG |
| **Program Sample** | ONINT_01.M, DELAY_01.M |

### □ ON MAPOS .. GOSUB

| | |
|---|---|
| **Summary** | Call up a subprogram when the master position xxx (MU) is passed. |
| **Syntax** | ON sign MAPOS xxx GOSUB name |
| **Parameter** | sign = + = when the master position xxx is passed in positive direction<br>– = when the master position xxx is passed in negative direction<br>xxx = Master Position in MU<br>name = name of the subprogram<br>100 xxx 0<br><–––––– positive direction<br>––––––> negative direction |
| **Description** | It is possible to call up a subprogram with the instruction ON MAPOS, if a specific master position (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.<br>**NB!:**<br>– An ON MAPOS Interrupt can be deleted with the command ON DELETE .. GOSUB ...<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– During the execution of subprograms triggered by an interrupt, NOWAIT ON is set automatically |
| **Command Group** | INT |
| **Cross index** | SUBPROG .. RETURN, DISABLE …, ENABLE …, Priorities of Interrupts, ON DELETE .. GOSUB, NOWAIT |
| **Syntax Example** | ON +MAPOS 1200 GOSUB name<br>// Always call up subprogram at position 1200 |
| **Sample** |  <br>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

| | |
|---|---|
| **Summary** | Call up a subprogram when the master position xxx (MU) is passed. |
| **Syntax** | ON sign MCPOS xxx GOSUB name |
| **Parameter** | sign = + = when the master position xxx is passed in positive direction<br>– = when the master position xxx is passed in negative direction<br>xxx = Master Position in MU<br>name = name of the subprogram<br>100 xxx 0<br><–––––– positive direction<br>––––––> negative direction |
| **Description** | 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.<br>**NB!:**<br>– 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.<br>– An ON MCPOS Interrupt can be deleted with the command ON DELETE .. GOSUB ...<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– During the execution of a subroutine called by an interrupt NOWAIT is automatically set to ON. |
| **Command Group** | INT |
| **Cross Index** | SUBPROG .. RETURN, DISABLE …, ENABLE …, Priorities of Interrupts, ON DELETE .. GOSUB, NOWAIT |
| **Syntax Example** | ON +MCPOS 1200 GOSUB parameter<br>// Always call up subprogram at position 1200 |
| **Syntax Example** | 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.<br>ON +MCPOS 4500 GOSUB output<br>// call subprogram output always on position 4500<br>SUBMAINPROG // set subprogram output<br>SUBPROG output<br>OUT 3 1 // 03 on<br>RETURN<br>ENDPROG |

### ☐ ON PARAM .. GOSUB

| | |
|---|---|
| **Summary** | Call up a subprogram when a parameter is altered. |
| **Syntax** | ON PARAM n GOSUB name |
| **Parameter** | n = parameter number<br>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.<br>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.<br>**NB!:**<br>The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>A maximum of 10 ON PARAM functions are possible.<br>**NB!:**<br>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 Sample** | ON PARAM 32-67 GOSUB poserr // when position error is changed<br>SUBMAINPROG<br>SUBPROG poserr<br>PRINT "New position error: ", GET POSERR<br>RETURN |

### ☐ 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<br>n = 0 = switch off the function<br>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.<br>**NB!:**<br>– The precision with which the time is depends on the remaining program. Typically the precision is ±1 ms.<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program<br>– 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 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<br>Byte 1 Status byte of the FC 300<br>Bit 1,2,4-7 no meaning<br>Bit 3 1 = moving finished<br>Bit 8 1 = FC 300 switched off<br>Byte 2<br>Bit 9 1 = MOVING<br>Bit 10 1 = Overflow Slave Encoder<br>Bit 11 1 = Overflow Master Encoder<br>Bit 12 1 = POSFLOAT active *)<br>Byte 3 not used<br>Byte 4 SYNCSTAT<br>Bit 25 1 = SYNCREADY<br>Bit 26 1 = SYNCFAULT<br>Bit 27 1 = SYNCACCURACY<br>Bit 28 1 = SYNCMMHIT<br>Bit 29 1 = SYNCSMHIT<br>Bit 30 1 = SYNCMMERR<br>Bit 31 1 = SYNCSMERR<br>name = subroutine name<br>*) 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 2 of the internal status (e.g. MOVING) and the bit n of SYNCSTAT.<br>**NB!:**<br>– The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– 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 */<br>SUBMAINPROG<br>SUBPROG markererror<br>SYNCSTATCLR 32 /* clear error flag SYNCMMERR */<br>/* use value 32 of Parameter SYNCSTATCLR, not the bit-number! */<br>RETURN<br>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)<br>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.<br>  <br>**NB!:**<br>– The precision with which the time is kept depends on the hardware used and the remaining program. Typically the precision is ±1 ms.<br>– In General: The subroutine to be called up must be defined within the SUBMAINPROG and ENDPROG identified program.<br>– 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 */<br>ON TIME 200 GOSUB off1 /* light off again after 200 ms */<br>SUBMAINPROG<br>SUBPROG off1<br>OUT 1 0<br>RETURN<br>ENDPROG |

### □ OUT

| | |
|---|---|
| **Summary** | Set or re-set digital outputs. |
| **Syntax** | OUT n s or<br>OUT X/n s |
| **Parameter** | n = output number<br>MCO 305: 1 – 8 (6)<br>FC 301 outputs: 27<br>Relay outputs: 21 (relay 1) and 22 (relay 2), Note: FC 301 < 11 kW only has relay 1.<br>MCB 105, relay outputs: X34/1 (relay 7), X34/5 (relay 8) and X34/10 (relay 9).<br>X/n = terminal block / pin number<br>s = condition<br>0 = OFF<br>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.<br>The selection of the mode for output 7,8 is done by par. 33-60 IOMODE.<br>If the outputs are used NPN or PNP depends on the selection for the standard FC 300 outputs set in par. 5-00.<br>**NB!:**<br>If an illegal combination or a pin number is used for X/n, which can not be set, then error 171 will be reported.<br>But there is no check if an input is used instead of an output or vice versa.<br>**NB!:**<br>After switching on the system, all outputs are OFF.<br>These outputs, which have pre-defined functions according to the I/O parameter settings, will also be influenced by the OUT commands!<br>The actual output status remains as it is, even after program end or program abort.<br>The output circuit and maximum load current can be taken from MCO Operating Instructions and the FC 300 Operating Instructions. |
| **Command Group** | I/O |
| **Cross Index** | OUTB, IN, INB,<br>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<br>OUT 27 1 // set output 27 on FC 300 main board to 1<br>OUT X59/3 1 // set output 3 on MCO 305 to 1<br>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<br>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).<br>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.<br>  <br>**NB!:**<br>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,<br>MCO 305 Operating Instructions, FC 300 Design Guide |
| **Syntax Example** | MOTOR OFF /* turn off controller */<br>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<br>0 = 1 – 8<br>1 = 27,29<br>v = value (0 ... 255) |
| **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.<br>  <br>**NB!:**<br>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.<br>Output circuit and maximum load current see MCO Operating Instructions. |
| **Command Group** | I/O |
| **Cross Index** | OUT, IN, INB,<br>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<br>OUTB 0 245 // disable outputs 2 and 4, switch through all other outputs<br>OUTB 0 128 // switch through output 8 only, disable others |
| **Program Sample** | OUTB_01.M |

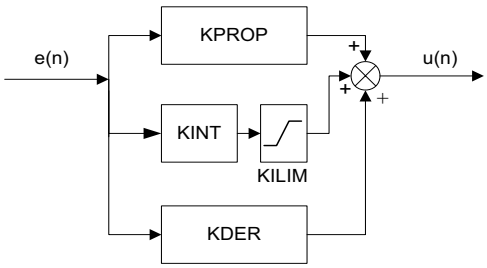
### ☐ OUTDA

| | |
|---|---|
| **Summary** | Sets FC 300 analog output. |
| **Syntax** | OUTDA n v |
| **Parameter** | n = output number (42)<br>v = value (0 – 100000) |
| | **NB!:**<br>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.<br>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. |
| **Command Group** | I/O |
| **Cross Index** | FC 300 Design Guide, parameter 6-50 |
| **Syntax Example** | /* condition: parameter 650 is set to "MCO controlled" */<br>OUTDA 42 50000 /* set FC 300 output to 10 mA */ |

### ☐ 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!:**<br>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<br>Variable = PCD[1].2 // Bit 2 of Word 1<br>Variable = PCD[2].b1 // Byte 1 of Word 2<br>PCD[1] = Variable<br>PCD[1].3 = Variable |
| **Syntax Example** | _IF (PCD[2]= = 256) THEN // compare value<br>_IF (PCD[3].2) THEN // is bit 2 of PCD3 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:<br>u(n) = ( KP * e(n) + KD *(e(n)-e(n–1)) + KI*∑e(n) ) / timer<br>where the following is true:<br>e(n) error occurring at time n<br>KP proportional factor of the PID control<br>KD *Derivative Value*<br>KI *Integral Factor* (limited by Integration Limit)<br>timer controller sample time<br>  <br>The corresponding factors can be set with the following commands:<br>SET PID KPROP 1 /* set KP 1 */<br>SET PID KDER 1 /* set KD 1 */<br>SET PID KINT 0 /* set KI 0 */<br>SET PID KILIM 0 /* Integration limit 0 */<br>SET PID TIMER 1 /* Sample time = 1 */<br>The following syntax example also show the default allocation of the factors. |
| **Command Group** | I/O |
| **Syntax Example** | e = INAD 53<br>u = PID e<br>PRINT "input = ",e, "output = ",u |

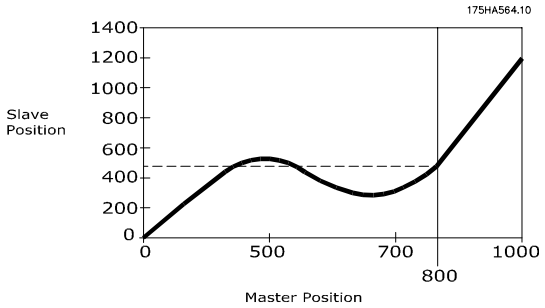
### □ POSA

| | |
|---|---|
| **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.<br>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.<br>**NB!:**<br>If a temporary zero point, set via SET ORIGIN, exists and is active, then the position result refers to this zero point.<br>**NB!:**<br>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, DEF ORIGIN, SET ORIGIN,<br>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 SET ORIGIN, 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, SET ORIGIN |
| **Syntax Example** | POSA CURVEPOS<br>// 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,<br>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.<br>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.<br>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 (").<br>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** | I/O |
| **Cross Index** | INKEY |
| **Syntax Example** | PRINT "Information is important !" /* print text information */<br>PRINT "Information is important !"; /* print information without line feed */<br>variable = 10<br>PRINT variable /* print contents of variables */<br>PRINT APOS /* print returned value function */<br>PRINT "Variable", variable,"Pos.:",APOS /* print mixed information */ |
| **Program Sample** | Uses – see all Program Samples. |

### □ PRINT DEV

| | |
|---|---|
| **Summary** | Stops information output |
| **Syntax** | PRINT DEV nn printlist |
| **Parameter** | nn = number for the device<br>0 = Standard output<br>-1 = no output after that line<br>printlist = normal argument for a print command |
| **Description** | PRINT DEV can be used to disable all prints in a program without commenting them out one by one.<br>The instruction (-1) defines the standard device new and is immediately effective for all PRINT commands, that do not hold a DEV. |
| **Command Group** | I/O |
| **Cross Index** | PRINT, INKEY |
| **Syntax Example** | PRINT DEV –1 "start" |

### □ 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).<br>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.<br>The signals generated are evaluated simultaneously as master input so that MAPOS, MIPOS, etc. function as they would in an external master.<br>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).<br>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<br>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.<br>  <br>**NB!:**<br>Because the abort criteria are checked at the end of the loop, the commands within the loop will be carried out at least once.<br>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 */<br>command line 1<br>command line n<br>UNTIL (A != 1) /* Abort condition */ |
| **Program Sample** | REPEA_01.M, DIM_01.M, ONINT_01.M, OUT_01.M, INKEY_01.M |

### □ RST ORIGIN

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

### □ SAVE part

| | |
|---|---|
| **Summary** | Save arrays or parameters in the EEPROM |
| **Syntax** | SAVE part<br>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 EEPROM with these commands:<br>SAVE GLBPARS saves the range of global parameters (group 30-5* and group 33-8*), and application parameters (group 19-**) in the EEPROM.<br>SAVE AXPARS saves all other axes parameters.<br>SAVE USRPARS saves only application parameters (group 19-**).<br>  <br>**NB!:**<br>The EEPROM can only handle execution of this command up to 10000 times. |
| **Command Group** | INI |
| **Cross Index** | DELETE ARRAYS, SAVEPROM |

### □ SAVEPROM

| | |
|---|---|
| **Summary** | saves memory in EEPROM |
| **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.<br>SAVEPROM triggers the same process, which can also be started in the menu *Controller*.<br>If you want to save only array elements or only global and application parameters, use the corresponding commands SAVE .. ARRAY, GLBPAR, or USRPARS. |
| |  **NB!:**<br>The execution time of SAVEPROM depends on the amount of data to be saved. It can be up to 4 seconds. |
| |  **NB!:**<br>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!:**<br>The EEPROM can only handle execution of this command up to 10000 times. |
| **Command Group** | INI |
| **Syntax Example** | PRINT "please wait"<br>SAVEPROM<br>PRINT "Thanks" |

### □ SET

| | |
|---|---|
| **Summary** | sets a parameter |
| **Syntax** | SET par v |
| **Parameter** | par = Parameter identification<br>v = parameter value |
| **Description** | With the SET command parameters can be temporarily changed while the program is running.<br>The parameter codes permitted can be found in the Parameter Reference. |
| |  **NB!:**<br>The parameter alterations are only valid while the program is running. After program end or abort, the original parameter values are valid again.<br>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 */<br>SET KPROP 150 /* change proportional factor */<br>SET PRGPAR 2 /* change activated program number */<br>SET I_FUNCTION_9_n /* previously the command SET I_BREAK */ |

### □ SETCURVE

| | |
|---|---|
| **Summary** | Set 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.<br>When this command is executed, the necessary pre calculations are done.<br>  <br>**NB!:**<br>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 cnf-file, then the order in the DIM instruction must match the order of the arrays in the cnf-file.<br>  <br>**NB!:**<br><u>If SYNCC is not active:</u><br>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.<br><u>If SYNCC is active:</u><br>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.<br>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.<br>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).<br>  <br>**NB!:**<br>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.) |
| **Command Group** | PAR |
| **Cross Index** | DIM, CMASTERCPOS (SYSVAR), CURVEPOS, |
| **Syntax Example** | DIM curve [280]<br>// See number of elements in the title bar of the CAM-Editor<br>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. RST ORIGIN 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 */ |

### □ SET ORIGIN

| | |
|---|---|
| **Summary** | Set absolute position as temporary zero point |
| **Syntax** | SET ORIGIN 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 SET ORIGIN command. This position is called temporary zero point.<br>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 SET ORIGIN commands without carrying out a previous RST ORIGIN. The absolute position value always refers to the real zero point. The last carried out SET ORIGIN command therefore determines the position of the temporary zero point in relation to the real zero point. |
| **Command Group** | INI |
| **Cross Index** | RST ORIGIN, DEF ORIGIN, POSA, CURVEPOS |
| **Syntax Example** | SET ORIGIN 50000 /* set temporary zero point to 50000 */ |
| **Syntax Example** | SET ORIGIN (–CURVEPOS)<br>// 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<br>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.<br>Since only integer values can be transmitted the parameter value to be transmitted must be adjusted with the associated conversion index.<br>A list of the FC 300 parameters with the corresponding conversion index can be found in the FC 300 manual.<br>  <br>**NB!:**<br>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 */<br>/* -Conversion index = -3 (Multiplied with $10^3$ during transmission) */<br>SETVLT 3-03 60000 |

**□ SETVLTSUB**

| | |
|---|---|
| **Summary** | Sets a FC 300 parameter with index number. |
| **Syntax** | SETVLTSUB par indxno v |
| **Parameter** | par = parameter number<br>indxno = index number<br>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.<br>Since only integer values can be transmitted the parameter value to be transmitted must be adjusted with the associated conversion index.<br>A list of these parameters with the corresponding conversion index can be found in the FC 300 manual.<br>  <br>**NB!:**<br>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 0-25 1 100<br>// 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):<br>Byte 3 MSB<br>Bit 0 1 = MOVING<br>Bit 1 1 = OVERFLOW Slave Encoder<br>Bit 2 1 = OVERFLOW Master Encoder<br>Bit 3 1 = POSFLOAT active *)<br>Byte 2 Status byte of axis control<br>Bit 7 1 = axis control switched off<br>Bit 2 1 = position reached<br>Bit 0,1,3-6 has no meaning<br>Byte 1 not used<br>Byte 0 LSB<br>Bit 7 1 = limit switch active<br>Bit 6 1 = Reference switch active<br>Bit 5 1 = Start switch active<br>Bit 2 1 = axis control switched off<br>Bit 0,1,3,4 not in use<br>*) 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. |
| **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** | I/O |
| **Cross Index** | AXEND |
| **Syntax Example** | PRINT STAT /* print status word */ |
| **Program Sample** | STAT_01.M |

### □ SUBMAINPROG .. ENDPROG

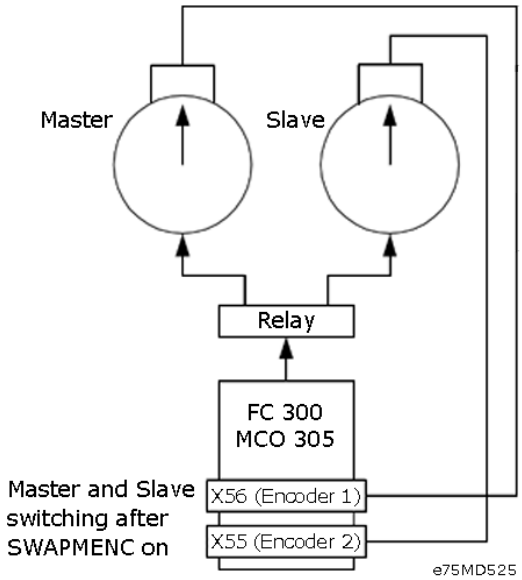
| | |
|---|---|
| **Summary** | Subroutine section definition |
| **Syntax** | SUBMAINPROG<br>ENDPROG |
| **Description** | The code word SUBMAINPROG begins the subroutine section, and the code word ENDPROG ends this specific program. The term subroutine means command sequences that, via the GOSUB instructions, can be called up and executed from various program positions.<br>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.<br>  **NB!:**<br>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 */<br>subroutine 1<br>subroutine n<br>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<br>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.<br>A subroutine can be called up and executed at any time by use of a GOSUB instruction.<br>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.<br>  **NB!:**<br>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 */<br>SUBPROG sp1 /* begin sp1 */<br>command line 1<br>command line n<br>RETURN /* end sp1 */<br>ENDPROG /* end SP-section */ |
| **Program Sample** | GOSUB_01.M, AXEND_01.M, ERROR_01.M, IF_01.M, STAT_01.M |

## □ SWAPMENC

| | |
|---|---|
| **Summary** | Swap master and slave encoder internally. |
| **Syntax** | SWAPMENC s |
| **Parameter** | s = condition<br>ON = master encoder input is feedback input<br>OFF = slave encoder input is feedback input |
| **Description** | This command allows swapping of the master and slave encoders. This is particularly meaningful if one wishes to alternately use two motors with one control.<br>Prior to the command SWAPMENC ON/OFF, a MOTOR OFF must always be executed in order to avoid a tolerated position error exceeded. Also, the controller parameters or axis parameters must be changed if both motors are different.<br>The motor leads can be switched via relay. |




**NB!**:
In this changeover, no positions are lost, even if the motors are moved by hand while the other motor is controlled. It is possible to always access the uncontrolled motor through MAPOS as well.

| | |
|---|---|
| **Command Group** | INI |
| **Cross Index** | MAPOS |
| **Syntax Example** | SWAPMENC ON // Swap slave encoder internally with master encoder |
| **Sample** | MOTOR OFF<br>OUT 1 1 // switch motor leads<br>SET KPROP ... // change axis parameters<br>SWAPMENC ON // swap encoder internally<br>MOTOR ON // turn on control again<br>POSA 10000<br>// move the motor which is connected to the master encoder<br>MOTOR OFF<br>OUT 1 0 // Switch motor leads<br>SET KPROP ... // Change axis parameters<br>SWAPMENC OFF // Swap encoder<br>MOTOR ON // Turn on control again<br>POSA 0<br>// Move the motor again which is connected to the slave encoder<br>Normally the upper socket X56 reads signal from the Master drive and X55 reads signals from the Slave drive. After a SWAPMENC ON, the master makes a positioning of 1000.<br>And then after a SWAPMENC OFF then slave makes the positioning.<br>The two drives need not necessarily be Master and Slave. They can just be two different motors with one FC 300 / MCO 305. |

## □ SYNCC

| | |
|---|---|
| **Summary** | Synchronization in CAM-Mode |
| **Syntax** | SYNCC num |
| **Parameter** | num = number of curves to be processed;<br>0 = the drive remains in CAM-Mode until another mode is started with commands such as MOTOR STOP, CSTART, POSA, etc. |
| **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).

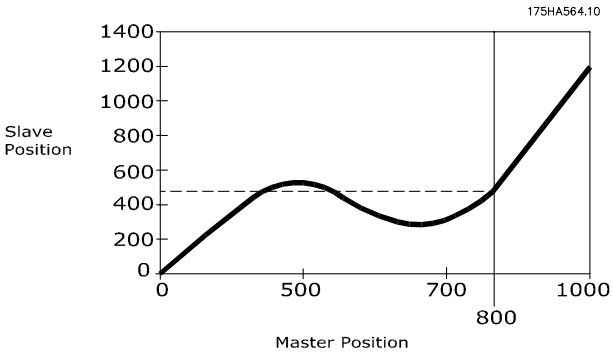
| | |
|---|---|
| **Command Group** | CAM |
| **Cross Index** | SYNCCSTART |

**Syntax Example**

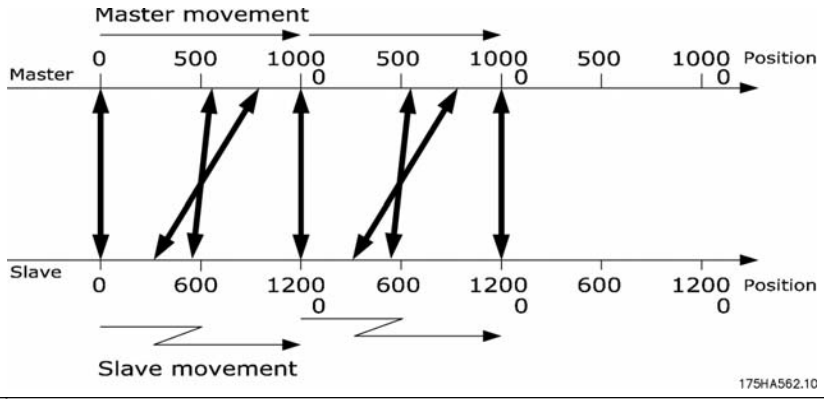
```
DIM curve [280]   // see number of elements in the title bar of the CAM-Editor
SETCURVE curve    // Set curve
SYNCC             // Synchronization in CAM-Mode
```

**Sample** Fix points of a curve:

| Master | Slave |
|---|---|
| 0 | 0 |
| 500 | 500 |
| 700 | 300 |
| 1000 | 1200 |

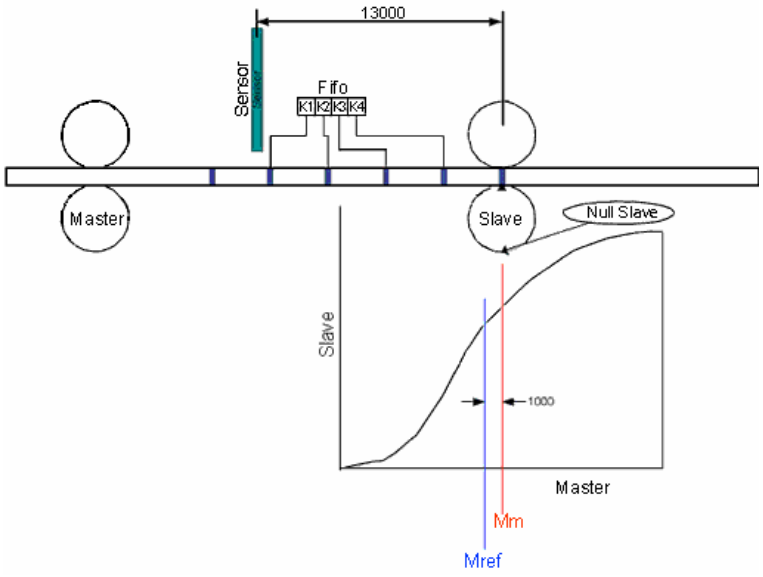



Hence in SYNCC 1 command thus locks the slave and master position as per the array.



## □ SYNCCMM

| | |
|---|---|
| **Summary** | Synchronization in CAM-Mode with master marker correction |
| **Syntax** | SYNCCMM num |
| **Parameter** | num = number of curves to be processed;<br>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).<br>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).<br>The marker can be the zero pulse of the encoder or an external 24 V signal.<br>**NB!:**<br>SYNCCMM does not start the slave drive nor does it interrupt on-going motions (e.g. CVEL), only SYNCCSTART does.<br>**NB!:**<br>The drive remains in CAM-Mode until 'num' curves have been processed successfully.<br>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<br>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

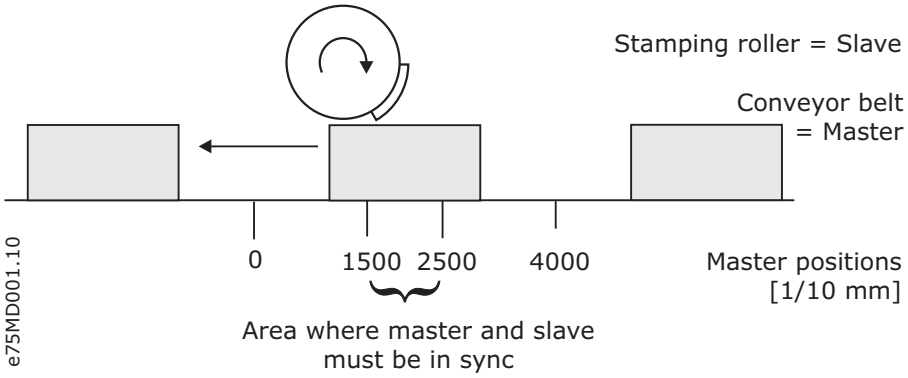
| | |
|---|---|
| **Summary** | Synchronization in CAM-Mode with slave marker correction. |
| **Syntax** | SYNCCMS num |
| **Parameter** | num = number of curves to be processed;<br>0 = the drive remains in CAM-Mode until another mode is started with commands such as MOTOR STOP, CSTART,, POSA, etc. |
| **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.<br>In contrast to SYNCCMM, no FIFO is created.<br>The marker can be the zero pulse of the encoder or an external 24 V signal.<br>  <br>**NB!:**<br>SYNCCMS does not start the slave drive nor does it interrupt on-going motions (e.g. CVEL), only SYNCCSTART does.<br>  <br>**NB!:**<br>The drive remains in CAM-Mode until 'num' curves have been processed successfully.<br>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<br>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<br>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.<br>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.<br>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.<br>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<br>SYNCC 0 // CAM mode synchronization<br>SYNCCSTART 1 // Engage slave at point A from start stop point pair 1 |

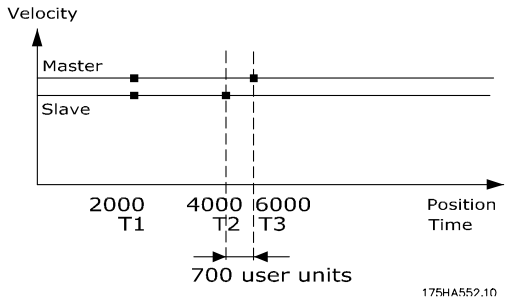
### □ SYNCCSTOP

| | |
|---|---|
| **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.<br>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!:**<br>When moving forward, disengaging begins at point A and is finished at point B; vice versa when moving backward. |
| | **NB!:**<br>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<br>SYNCC 0 // Synchronize in CAM-Mode<br>SYNCCSTART 1 // Start slave with start point pair 1<br>SYNCCSTOP 2 0<br>// Stop slave with stop point pair 2 at the slave position 0 or 3600 |
| **Sample** | Slave positions [degrees]<br>stamp beginning 120°, end 240° |



### □ SYNCERR

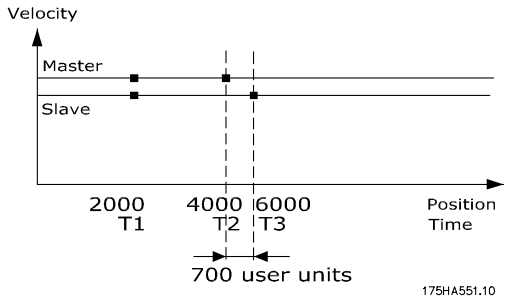
| | |
|---|---|
| **Summary** | Queries actual synchronization error of the slave |
| **Syntax** | res = SYNCERR |
| **Return Value** | res = actual synchronization error of the slave in UU and<br>a) as an absolute value when the value of the accuracy window is defined with a plus sign in the parameter SYNCACCURACY;<br>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.<br>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).<br>**NB!:**<br>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.<br>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,<br>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<br>Here the SYNCERR returns the absolute value 700. |



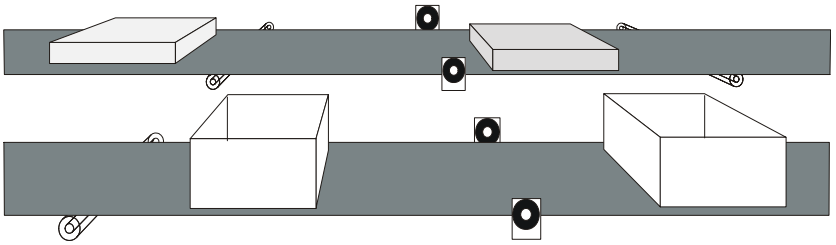

SYNCACCURACY = 1000

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

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

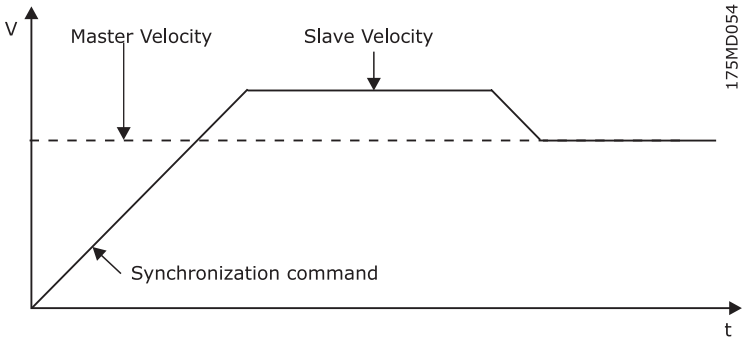


### □ SYNCM

| | |
|---|---|
| **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.<br><br>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.<br><br>**NB!:**<br>In addition to the parameters used by SYNCP, par. 33-25 SYNCREADY, and par. 33-24 SYNCFAULT are also of significance.<br><br>**NB!:**<br>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.<br><br>par. 33-15, 33-16 *Marker Number for Master* and *for Slave*<br>par. 33-17, 33-18 *Master Marker* and *Slave Marker Distance*<br>par. 33-19, 33-20 *Master* and *Slave Marker Type*<br><br>**NB!:**<br>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.<br><br>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). |
| **Command Group** | SYN |
| **Syntax Example** | SYNCM /* synchronization of the position with marker correction */ |
| **Example** |  <br>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. |

## □ 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.

**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).

**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.

| | |
|---|---|
| **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.<br>This flag is not used with SYNCV.<br>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).<br>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.<br>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.<br>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*).<br>The same is true for SYNCSMHIT with the slave. |
| | **NB!:**<br>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. |

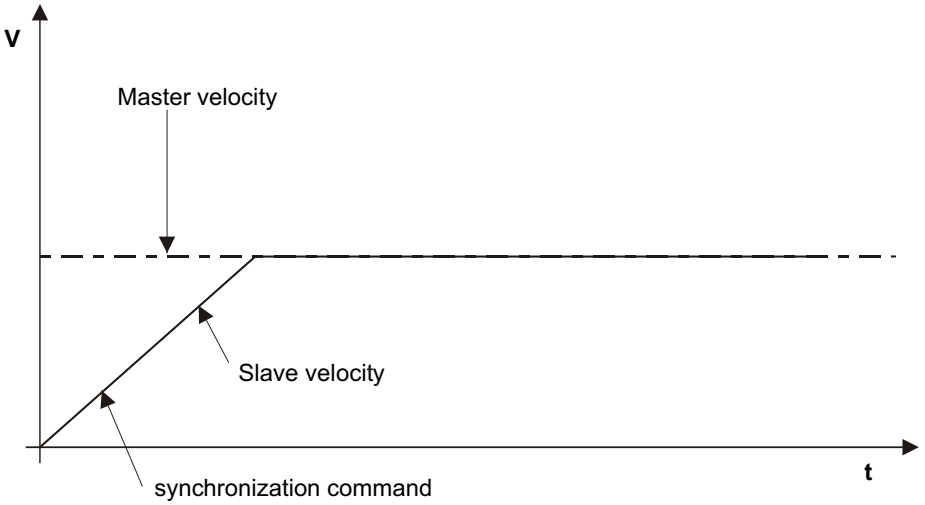
| | |
|---|---|
| | Example:<br>Distance between two master markers par. 33-17 = 30000<br>*Master Marker Tolerance Window* par. 33-21 SYNCMWINM = 1000<br>The flag is set at 31000 if no marker is identified.<br>These flags are deleted for every SYNCM command.<br>If the *Master Marker Tolerance Window* 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.<br>The same applies analogously for SYNCSMERR in the slave.<br>  **NB!:**<br>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 */<br>ENDIF |

### □ SYNCSTATCLR

| | |
|---|---|
| **Summary** | Resetting of the flags MERR and MHIT |
| **Syntax** | SYNCSTATCLR value<br>The SYNCSTATCLR command should only be used in a subprogram for dealing with errors. (see ON ERROR GOSUB). |
| **Parameter** | value = 8 = SYNCMMHIT<br>16 = SYNCSMHIT<br>32 = SYNCMMERR<br>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!:**<br>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!:**<br>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 Filte*r (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]<br>n = index |
| | **NB!:**<br>The values of the system variables are internal, hardware-dependent values which may change. |
| **Description** | The system variable SYSVAR – a prepared pseudo array – provides detailed system information. You also need this index if you link the system variable with the LCP display using LINKSYSVAR or specify recording data of a *Test run* with TESTSETP. |
| **Command Group** | CON |
| **Cross index** | LINKSYSVAR, TESTSETP |

**System and Axis 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) |

| Index | Description Axis Process Data |
|---|---|
| 4096 | Current position slave [qc] (see APOS [UU]) |
| 4097 | Set position slave [qc] (see CPOS [UU]) |
| 4098 | Last slave index position [qc] (see IPOS [UU]) |
| 4099 | Actual velocity in qc/st, where st is the sample time set by _GETVEL. |
| 4100 | Current velocity Master (as above) |
| 4101 | Current position error in qc |
| 4102 | Contains the number of revolutions of the encoder after the first overflow of the absolute encoder, provided the graduation per revolution is entered correctly in par. 32-01 *Incremental Resolution* (ENCODER). |
| 4103 | As above for the master |
| 4105 | Current master position [qc] (see MAPOS) [UU]) |
| 4106 | Last Master index position [qc] (see MIPOS) [UU]) |

| Index | Description Axis Process Data |
|---|---|
| 4107 | Internal current velocity (ACTPOS – last ACTPOS) (qc/1 ms) |
| 4108 | Internal Master velocity (see above) |
| 4109 | Current frequency of the master simulation (1/1000 Hz) (see PULSVEL) |
| 4110 | Determines whether the master simulation is active or not (1 or 0) |
| 4111 | Actual set value which is output by the control through the position controller (between –FFFFF and FFFFF or –1048575 and 1048575 decimal) |
| 4113 | Current used timer for the PID loop (TIMER). |
| 4114 | Current used Timer for the profile generator (PROFTIME). |
| 4115 | Specifies, whether negative references should be read (!=0) out or not (=0) . |
| 4116 | Specifies, whether reference with 0-10V and direction output should be read out (>0). If so these parameter contains the output number, which is used (1–8). |
| 4117 | Actual acceleration of the virtual master. |
| 4118 | Target frequency for virtual master (unit see above). |
| 4119 | Vlamode (abs / relative ?) |
| 4120 | Number of qc between index-pulses |
| 4121 | Type of z pulse SYNCMTYPM |
| 4122 | User reference value given by OUTAN , scaling see REG_REFERENCE |
| 4123 | Internal parameter: slave encoder type (ENCDODERTYPE = 0..2) |
| 4124 | Internal parameter: master encoder type (MENCODERTYPE = 0..6) |
| 4125 | Slave encoder resolution |
| 4126 | Master encoder resolution |
| 4127 | Defines whether amplifier is set to stop mode (!=0) or to idle (==0) in case of motor off. |
| 4128 | Internal parameter: delivers same information as MAVEL does, but always gives information of real encoder, even if master is simulated (MENCODERTYPE == 6). |

| Index | Description Axis Process Data, Profile Generator Values | | |
|---|---|---|---|
| 4218 | Returns all 32 flags of the profile generator. These are: | | |
| | PG_FLAG_BUSY | 1L | // Flag for busy information |
| | PG_FLAG_COMMANDERR | 2L | // Flag for Command Error occurred (not used) |
| | PG_FLAG_POSREACHED | 4L | // Flag for Position reached |
| | PG_FLAG_INDEX_HIT | 8L | // Flag for Index observed |
| | PG_FLAG_WRAP_OCC | 16L | // Flag for Wraparound occurred (not used) |
| | PG_FLAG_POS_ERR | 32L | // Flag for Position Error occurred |
| | PG_FLAG_BRKPT_RCHD | 64L | // Flag for Breakpoint reached (not used) |
| | PG_FLAG_FLOATING | 128L | // Flag for MOTOR OFF |
| | | | |
| | PG_FLAG_MOVING | 1L << 8 | // Flag for axes is moving |
| | PG_FLAG_OVERFLOWS | 2L << 8 | // Flag for overflow of slave position |
| | PG_FLAG_OVERFLOWM | 4L << 8 | // Flag for overflow of master position |
| | PG_FLAG_POSFLOAT | 8L << 8 | // Flag for floating in pos control |
| | PG_FLAG_INTERNTST | 64L << 8 | // Flag for internal use |
| | PG_FLAG_SYNCREADY | 1L << 24 | // Flag for Synchronization ready |
| | PG_FLAG_SYNCFAULT | 2L << 24 | // Flag for Synchronization failed |
| | PG_FLAG_SYNCACCUR | 4L << 24 | // Flag for Accuracy reached (syn) |

| Index | Description Axis Process Data, Profile Generator Values | | |
|---|---|---|---|
| | PG_FLAG_SYNCMMHIT | 8L << 24 | // Flag for master marker hit |
| | PG_FLAG_SYNCSMHIT | 16L << 24 | // Flag for slave marker hit |
| | PG_FLAG_SYNCMMERR | 32L << 24 | // Flag for master marker distance exceeded |
| | PG_FLAG_SYNCSMERR | 64L << 24 | // Flag for slave marker distance exceeded |
| | PG_FLAG_TESTFLAG | 128L << 24 | // Flag for internal use |

| Index | | Description Axis Process Data / CAM profile |
|---|---|---|
| 4220 | CINDEX | Actual index into the curve interpolation area (number of actual interpolation point 0..Intno-1) |
| 4221 | CVINDEX | Actual value index used. Is equal to CINDEX if not starting or stopping is active. In that case the CVINDEX points into the start or stop path data. |
| 4222 | CMAXINDEX | Maximum index allowed (Intno – 1) |
| 4223 | CIMPS | Curve position within the actual interpolation interval (integer part of 64 bit value) |
| 4224 | CMILEN | Length of interpolation interval in MU units (integer part of 64 bit value) |
| 4225 | CWRAP | Actual curve counter (1 .. CCOUNTER); will be reset after every re-start of SYNCC. |
| 4226 | CSSTART | Slave start offset of actual curve in qc (for closed curves always 0) |
| 4227 | CCOUNTER | Demanded value of number of curves to do (last SYNCC command) |
| 4228 | CCURVEPOS | Slave curve position in UU units (updated in SETCURVE and when SYNCCxx active). |
| 4229 | CSLAVECPOSQ | Actual slave curve position in qc (relative to CSSTART). |
| 4230 | CMASTERCPOS | Actual master curve position in MU units (initialized in SETCURVE and updated after SYNCCxx); see also CURVEPOS. |
| 4231 | GETCMDVEL | Gives command velocity (intgr part * 128) with sign (see also 4186) |
| 4240 | PFG_G_STARTKORR | Contains the first correction value after SYNCM start. It specifies, how many marker faults must be compensate from the start-up. (Will be filtered, if SYNCOFFTTIME and par. 33-29 *Filter Time for Marker Correction* are set). |
| 4241 | PFG_G_START KORRREST | Contains the carryover of the start correction value, which has to be executed yet. Scaled with PFG_G_SCALESHIFT. |
| 4242 | PFG_G_KORRFILT | Contains the filtered correction value, scaled with PFG_G_SCALESHIFT. |
| 4243 | PFG_G_LASTMMDIST | Contains the last measured distance between two master markers (qc master). |
| 4244 | PFG_G_MMARK CORR | Contains the gear correction factor, which was calculated; scaled with PFG_G_SCALESHIFT. |
| 4245 | PFG_G_KORRUNFILT | Contains the last unfiltered correction value (qc - slave). |
| 4246 | PFG_G_MDISTMARK | Contains the actual master position distance of the last master marker in % of the nominal marker distance. |
| 4247 | PFG_G_SDISTMARK | Contains the actual slave position distance of the last slave marker in % of the nominal marker distance. |
| 4248 | PFG_G_START KORRVAL | STARTKORRVAL is the value with that the start correction value will be relieved (reduced) at every marker correction. |
| 4249 | PFG_G_LASTSM DIST | Last measured distance between two slave marker in qc - slave. |
| 4250 | PFG_G_MARKER FILTER | Tau for the PT filter used to calculate medium marker distance. |
| 4251 | PFG_G_KORRTAU | Tau for the PT filter used to calculate PFG_G_KORREKTUR |
| 4252 | PFG_G_INTMM ERROR | Sum of all errors of the marker distance (actual – medium distance) |
| 4253 | PFG_G_MMARKERR | Filtered Sum of all errors (scaled value) |
| 4275 | PFG_G_JSTATE | Contains the state of the Limited Jerk Movement state machine. These could be: |

| Index | | Description Axis Process Data / CAM profile | | |
|---|---|---|---|---|
| | | PGS_JRK_ACC_S | 1 | // Acceleration start (ramp up towards Amax). |
| | | PGS_JRK_ACC_C | 2 | // Acceleration constant (at Amax towards Vmax). |
| | | PGS_JRK_ACC_E | 3 | // Acceleration end (ramp down to Vmax). |
| | | PGS_JRK_CONST | 4 | // Constant velocity (at Vmax). |
| | | PGS_JRK_DEC_S | 5 | // Deceleration start (ramp up towards -Amax). |
| | | PGS_JRK_DEC_C | 6 | // Deceleration constant |
| | | PGS_JRK_DEC_E | 7 | // Deceleration end (ramp down to 0 velocity). |
| | | PGS_JRK_END | 8 | // Movement is complete (0 velocity). |
| | | PGS_JRK_START | 9 | // Starting limited jerk movement. |
| | | PGS_JRK_ACC_HIGH | 10 | // Acceleration is too high. |
| | | PGS_JRK_ACC_LOW | 11 | // Acceleration is too low (i.e. too negative). |
| | | PGS_JRK_VEL_HIGH_S | 12 | // Velocity is too high, starting ramp to Vmax. |
| | | PGS_JRK_VEL_HIGH_C | 13 | // Velocity is too high, constant deceleration towards Vmax. |
| | | PGS_JRK_VEL_HIGH_E | 14 | // Velocity is too high, ending ramp to Vmax. |
| | | PGS_JRK_REVERSE | 15 | // Reverse the algorithm orientation. |
| 4276 | PFG_G_VCMDSIGNED | Signed reference velocity [qc/st], identical to SYSVAR[4186], but with sign | | |
| 4277 | PFG_G_JERKSTOPPATH | Delivers the length of the stop path assuming that a motor stop with the actual velocity and actual acceleration would be done. For this calculation the currently active values for VEL, ACC, DEC, and JERKMIN are used. This length of the stop path of course varies when i.e. the velocity or par. *Jerk Duration* JERKMIN or any other relevant parameter is changed.<br>This calculation can also be used when a RAMPTYPE other than 2 is used. In that case the value is calculated as if RAMPTYPE 2 is active.<br>Be aware that the value can change unexpected if the system is in the process of doing a limited jerk movement and is in an overshoot situation. This will occur if a POSA is done with a position that can't be reached (given the current actual velocity) without going past the position, turning around, and coming back to the position. It happens because the limited jerk movement is designed to reverse direction without the need to come to a standstill. | | |

### □ TESTSETP

| | |
|---|---|
| **Summary** | Specify recording data for test run |
| **Syntax** | TESTSETP ms vi1 vi2 vi3 *arrayname* |
| **Parameter** | ms = interval in milliseconds between two measurements<br>vi 1–3 = indices of the three values to be recorded. The agreements for the system array apply. Three values are always recorded.<br>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 |
| vi1 | i | Value, recorded at point 1 (Index) |
| vi2 | i | Value, recorded at point 2 (Index) |
| vi3 | i | Value, recorded at point 3 (Index) |
| number | nn | Specifies how many measurements follow |
| data | … | Measurement data |
| … | … | (total nn*3) |
| number | 0-mm | Number of measurements (if others are present) |
| data | … | (see above) |

Please make sure that the size of the array is sufficient for the recording. You need 6 elements for the header, 1 element for the number, and 3 elements for each measurement. Thus for 100 measurements you need 307 elements.

**Description**

The menu *Testrun* can be used to trigger a test run which records desired and current position, speed, acceleration and current and whose result can be seen in the test run graphic.

With the two commands TESTSETP and TESTSTART you can record other or additional parameters, for example the master position. And, in contrast to a test run, you can record this data while executing the program.

With TESTSETP you determine the parameters of the recording (which parameters are to be recorded, how often and in which array) and with TESTSTART you then start recording.

**Command Group** I/O

**Cross References** TESTSTART, DIM, SYSVAR, Testrun, Display Recording

**Syntax Example**

```
DIM tstfahrtarray[307]          // Array with 307 elements
TESTSETP 3 0X1001 0X1009 0X1005 tstrunarray
   // Record current position, current position of the master
   // and current tolerated position error
... Start positioning run ...
TESTSTART 100                   // Start recording
```

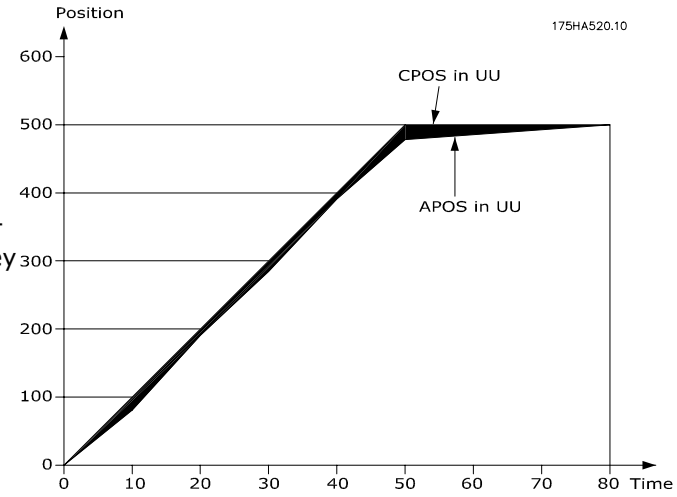
### □ TESTSTART

| | |
|---|---|
| **Summary** | Start the recording of a test run. |
| **Syntax** | TESTSTART no |
| **Parameter** | no = number of measurements to be carried out |
| | If an array does not have sufficient space for no. measurements, the error Error 171 "Array too small" is triggered. |
| **Description** | This command is used to start the recording of a test run with the contents as defined in TESTSETP. The recorded data can then – as required – be graphically represented using *Testrun* → *Display recording*. There are four graphics: Position, Speed, Acceleration and Current available for this. |
| **Command Group** | I/O |
| **Cross References** | TESTSETP, Testrun, Display Recording |
| **Syntax Example** | SYNCP // Synchronization of the position<br>WAITI 1 ON // When the key is pressed<br>TESTSTART 200 // Start recording (200 measurements) |
| **Syntax Example** | NOWAIT ON // Do not wait until the position is reached<br>VEL 50<br>POSA 100000 // Start positioning with velocity 50%<br>WHILE (APOS<50000) DO // Wait until position 50000 is reached<br>ENDWHILE<br>VEL 100 // Increase velocity to 100%<br>TESTSTART 200 // Start recording (200 measurements)<br>DELAY 20 // Wait 20 ms<br>POSA 100000 // Start positioning with new velocity<br>NOWAIT OFF // Wait until positioning is finished |

### □ TIME

| | |
|---|---|
| **Summary** | Reads system-time |
| **Syntax** | res = TIME |
| **Return Value** | res = system-time in milliseconds after switching on |
| | **NB!:**<br>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** | I/O |
| **Syntax Example** | PRINT TIME /* print current system-time */<br>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.<br>It is to be noted that the actual position need not be the same as the commanded position and they are not automatically compensated.<br>  |
| **Command Group** | I/O |
| **Cross Index** | APOS, CPOS, par. 32-67 *Max. Tolerated Position Error* |
| **Syntax Example** | PRINT TRACKERR /* query actual position error of the axis */ |
| **Sample** | POSA 500<br>WHILE(1) DO<br>{<br>PRINT "Command Position", CPOS<br>PRINT "Actual Position", APOS<br>PRINT "Error", TRACKERR<br>WAITT 10<br>}<br>ENDWHILE<br>Output:<br>Command Position 100<br>Actual Position 98<br>Error 2<br>Command Position 200<br>Actual Position 199<br>Error 1<br>Command Position 300<br>Actual Position 297<br>Error 3 …<br>Command Position 500<br>Actual Position 500<br>Error 0<br>… and so on<br>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. |

### ☐ VEL

| | |
|---|---|
| **Summary** | Set velocity for relative and absolute motion. |
| **Syntax** | VEL v<br>$\text{Command Velocity [RPM]} = V * \frac{\text{par. 32 - 80 Maximum Velocity}}{\text{par. 32 – 83 Velocity Resolution}}$ |
| **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.<br>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.<br>NOTE: Slave velocity in synchronizing mode is also limited by the VEL command.<br>**NB!:**<br>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.<br>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.<br>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<br>Parameter: 32-80 *Maximum Velocity* |
| **Syntax Example** | VEL 100 /* Velocity 100 */ |
| **Program Sample** | VEL_01.M |

### ☐ 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 */<br>WAIT AX /* Alternative form */ |
| **Program Sample** | WAIT_01.M, VEL_01.M |

### □ WAITI

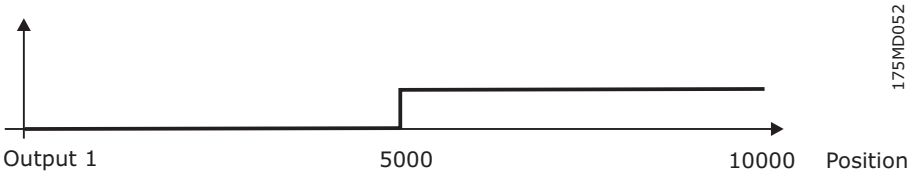
| | |
|---|---|
| **Summary** | Wait for defined input condition |
| **Syntax** | WAITI n s |
| **Parameter** | n = input number 1 – 8 or 16 – 33<br>s = expected condition: ON = High-Signal<br>OFF = Low-Signal |
| **Description** | The WAITI command waits before continuing the processing until the specified input has got the desired signal level.<br>  <br>**NB!:**<br>If the expected input condition does not occur, then the program will remain 'stuck' at this point.<br>A minimal signal length is required for the sure identification of a signal condition!<br>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 */<br>WAITI 4 1 /* 3 alternative forms */<br>WAIT I 4 ON<br>WAIT I 4 1 |
| **Syntax Example** | WAITI 6 OFF /* Wait till Low level reached input 6 */<br>WAITI 6 0 /* 3 alternative forms */<br>WAIT I 6 OFF<br>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.<br>  <br>**NB!:**<br>If the time is exceeded then an error is triggered which can be evaluated with a ON ERROR function.<br>  <br>**NB!:**<br>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<br>CSTART<br>WAITNDX 10000<br>/* Waits a maximum of 10 s for the axis to reach the index position */<br>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<br>POSA 10000<br>WAITP 5000 /* wait for position 5000 */<br>OUT 1 1 /* set output 1 */<br>NOWAIT OFF |




### ☐ 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.<br>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 */<br>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<br>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 criteria is made up of one or more comparison operations, and is 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.<br>  <br>**NB!:**<br>Depending on the loop criteria, it can happen that the contents of the loop will never be processed.<br>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<br>command line 1<br>command line n<br>ENDWHILE |
| **Program Sample** | WHILE_01.M, INKEY_01.M |

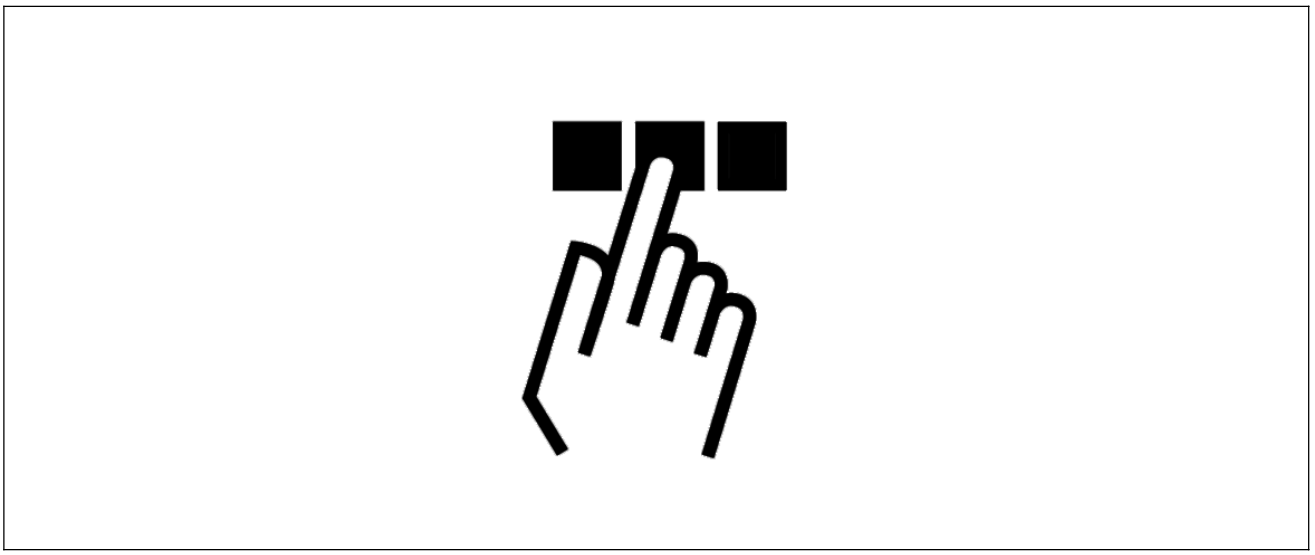
### ☐ _GETVEL

| | |
|---|---|
| **Summary** | Changes sample time for AVEL and MAVEL |
| **Syntax** | var = _GETVEL t<br>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.<br>The command _GETVEL lasts exactly as long as the assigned value, e.g. _GETVEL 200 ca. 200 ms. |
| **Command Group** | I/0 |
| **Cross Index** | AVEL, MAVEL |
| **Syntax Example** | var = _GETVEL 200<br>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 |
| **Syntax** | #INCLUDE file |
| **Parameter** | file = complete name of the file to be included<br>(path commands inadmissible) |
| **Description** | The #INCLUDE instruction tells the Compiler to include the contents of the given file in the corresponding program position during the translation of the program. The INCLUDE instruction is therefore not a genuine command that causes a reaction within the MCO 305, but an instruction for the translation program – a Compiler Directive.<br>The #INCLUDE instruction can be placed in any program position, as many times as desired within a program. However, attention must be given to the fact that the data to be included contains commands that may be used in the momentary program position, and that the command syntax is correct.<br>The #INCLUDE instruction is especially suitable for storing frequently used subroutines in separate files and to include them in the application within the SUBMAINPROG.. ENDPROG area. |
|  | **NB!:**<br>The file to be included must be in the current directory.<br>The given data names must end with ".m".<br>The commands within the file to be included must have correct syntax. |
| **Command Group** | CON |
| **Syntax Example** | #INCLUDE INC_UP01.M /* Include contents from file INC_UP01.M */ |
| **Program Sample** | INCL_01.M + INCSTA01.M + INCPOS01.M + INCIN01.M |

# Parameter Reference



## □ FC 300, MCO 305, and Application Parameters

Basically there are these main parameter types: FC 300 parameters, MCO 305 parameters, and application parameters (group 19-**):

**– FC 300 and MCO 305 Parameters**

The parameters concerning the Frequency Converter are described in FC 300 Design Guide. The following section describes all the parameters which are necessary or helpful using the MCO 305.

Default Setting and Reset

All parameters has a default setting ex factory which can be reset by manual initialization of the FC 300 or by means of the setup copy function (see in FC 300 operating instruction for further details).

The parameters can also be reset to default settings in menu *Controller* → *Reset* → *Parameters* or → *Complete*. Deleting the entire memory in menu *Controller* → *Memory* → *Delete EEPROM* will reset the parameters to default setting as well.

**– Application Parameters**

The application parameters 19-00 to 19-99 are defined in a APOSS program using the command LINKGPAR and will be displayed in the LCP.

The parameters 19-90 to 19-99 are read only parameters which can be used for data read out on the LCP in display line 1 or 2. You can select which parameters are displayed at which position on the FC 300 by means of par. 0-2* LCP Display.

## □ Parameter Access

There are three methods to access parameters:

- LCP
- PC Software MCT 10
- Field bus

### □ Initialization to Default Settings

Initialize the frequency converter to default setting in two ways:

Recommended initialization (via par. 14-22):

1. Select par 14-22
2. Press [OK]
3. Select "Initialization"
4. Press [OK]
5. Cut off the mains supply and wait until the displays turns off.
6. Reconnect the mains supply – the frequency converter is now reset.

**NB!**
MCO 305 programs and arrays are not affected.

| Par. 14-22 initializes all except: | |
|---|---|
| 14-50 | *RFI 1* |
| 8-30 | *Protocol* |
| 8-31 | *Address* |
| 8-32 | *Baud Rate* |
| 8-35 | *Minimum Response Delay* |
| 8-36 | *Max Inter-char Delay* |
| 15-00 to 15-05 | Operating Data |
| 15-20 to 15-22 | Historic log |
| 15-30 to 15-32 | Fault log |

Manual initialization:

1. Disconnect from mains and wait until the display turns off.
2. Press [Status] – [Main Menu] – [OK] at the same time:
3. Reconnect mains supply while pressing the keys.
4. Release the keys after 5 s.
5. The frequency converter is now programmed according to default settings.

| This method initializes all except: | |
|---|---|
| 15-00 | *Operating Hours* |
| 15-03 | *Power-up's* |
| 15-04 | *Over temp's* |
| 15-05 | *Over volt's* |

**NB!**:
When you carry out manual initialization, you also reset serial communication and fault log settings.

And all MCO 305 programs and arrays are deleted!

### □ Reading and Writing Parameters

From the application program there is read access to all FC 300 parameters including MCO 305 parameters and application parameters (group 19-**). There are two commands for reading parameters:

- GET is used to read all MCO 305 related parameters i.e. groups 19-**, 32-**, 33-**, and 34-**.
- GETVLT is used to read all other FC 300 parameters.

There is also write access to FC 300 parameters but with some restrictions: Parameter groups 16-** and 34-** are read only and can thus not be changed. Some FC 300 parameters can only be changed while the drive is stopped and can thus not be changed during operation, see FC 300 design guide for a complete description of all parameters.

There are two commands for writing parameters:

- SET is used to write to all MCO 305 related parameters i.e. groups 19-**, 32-**, and 33-**.
- SETVLT is used to write to all other FC 300 parameters.

Overview

| | Parameters | Command | Example |
|---|---|---|---|
| Read | 32-**, 33-** and 34** | GET *name* | var = GET ENCODER |
| | 19-** | GET *number* | var = GET 1900 |
| | All other FC 300 parameters | GETVLT *number* | var = GETVLT 1610 |
| Write | 32-** and 33-** | SET *name* | SET ENCODER 1024 |
| | 19-** | SET *number* | SET 1900 555 |
| | All other FC 300 parameters | SETVLT *number* | SETVLT 303 1500 |

**NB!**:
Parameter changes made by SET or SETVLT are only stored in RAM and will be lost at power down, exception: The application parameters (group 19-**) are automatically stored at power down. The other MCO 305 parameters (group 32-** and 33-**) can be saved by means of the SAVE AXPARS command.

### □ Parameter Changes and Storage

Parameters changed via the LCP or via menu *Controller* → *Parameters* → *Axis* are saved in an EEPROM and thus retained at power down.

Parameters changed from the APOSS application program with the command SETVLT are only stored in RAM and thus lost at power down.

Parameters changed from the APOSS application program with the command SET are only active while the application program is running. These parameters can be saved in an EEPROM and thus retained at power down by means of the command SAVEPROM or press the [OK] key on the FC 300 display.

**NB!:**
Please note that an EEPROM has limited lifetime; but in can be reprogrammed approximately 10000 times.

## □ FC 300 Parameters Overview

In addition to adding new parameters (groups 19-**, 32-**, 33-** and 34-**), some existing FC 300 parameters are modified when MCO 305 is installed, some parameters will get additional selections, some will get a different default value. The following is an overview of the parameters in question:

| Par. number | New selections | New default |
|---|---|---|
| 0-20<br>0-21<br>0-22<br>0-23<br>0-24 | [1990] - [1999]<br>[3400] - [3410]<br>[3421] - [3430]<br>[3440] - [3441]<br>[3450] - [3462] | - |
| 1-02 | [4] MCO Encoder 1<br>[5] MCO Encoder 2 | - |
| 1-62 | - | 0% |
| 3-15<br>3-16<br>3-17 | - | [0] No function |

**∗ default setting  [ ] value for use in communication via serial communication port**

| Par. number | New selections | New default |
|---|---|---|
| 3-41<br>3-42<br>3-51<br>3-52<br>3-61<br>3-62<br>3-71<br>3-72 | - | 0.01 s |
| 4-10 | - | "Both Directions" |
| 5-10<br>5-11<br>5-12<br>5-13 | - | [0] No operation |
| 5-30<br>5-31<br>5-32<br>5-33 | [51] MCO Controlled | [51] MCO Controlled |
| 5-40 | [51] MCO Controlled | [51] MCO Controlled |
| 5-60<br>5-63 | [51] MCO Controlled | [51] MCO Controlled |
| 6-50 | [52] MCO 0-20 mA | [52] MCO 0-20 mA |
| 6-60 | [53] MCO 4-20 mA | |
| 7-00 | [4] MCO Encoder 1<br>[5] MCO Encoder 2 | - |
| 8-02 | [5] Option C0 | [5] Option C0 |
| 9-15 | [3401] - [3410] | Index [0] 3401<br>Index [1] 3402<br>Index [2] 3403<br>Index [3] 3404<br>Index [4] 3405<br>Index [5] 3406<br>Index [6] 3407<br>Index [7] 3408<br>Index [8] 3409<br>Index [9] 3410 |
| 9-16 | [3421] - [3430] | Index [0] 3401<br>Index [1] 3402<br>Index [2] 3403<br>Index [3] 3404<br>Index [4] 3405<br>Index [5] 3406<br>Index [6] 3407<br>Index [7] 3408<br>Index [8] 3409<br>Index [9] 3410 |

**NB!:**
In general it is very important to optimize the FC 300 parameters to the motor, i.e. by using AMA, in order to obtain a good control behavior.

Maximum Reference par. 3-00 must be set in accordance with par. 32-80 *Maximum Velocity* before the controller parameters are optimized.

## □ Application Settings

### □ 19-** Application Parameters

#### 19-00 ...19-89 Application Parameters

**Range**

-2147483648 – 2147483647

(The real range seen at the LCP is defined by LINKGPAR)

**Function**

The application parameters can be used to input application specific data to the application program. Application parameters are created via the LINKGPAR command whereby it is possible to specify parameter name as well as minimum and maximum setting limits. See also description of LINKGPAR.

Note: Application parameters are only visible and accessible via LCP when created/defined in the application program.

**Syntax Example**

```
LINKGPAR 1901 "name" 0 100000 0
/* Link par. 19-01 with LCP */
```

#### 19-90 .. 19-99 Read only Application Parameters

**Range**

-2147483648 – 2147483647

(Range depends on the data linked to the read-out parameter)

**Function**

The read only application parameters can be used to read out additional internal process data and application specific data from the application program. Read only application parameters are created via:

- LINKSYSVAR command for internal process data.
- LINKGPAR command for application specific data;

whereby it is possible to specify the parameter name. See also description of LINKSYSVAR and LINKGPAR.

Parameters 19-90 through 19-99 can be selected as LCP display read outs via parameters 0-20 through 0-24.

Note: Read only application parameters are only visible at LCP when created/defined in the application program.

## □ MCO Parameters

For a better overview the MCO Parameters are divided into groups:

| | | |
|---|---|---|
| **32-**** | **MCO Basic Settings** | |
| **32-0*** | Encoder 2 - Slave | page 183 |
| 32-3* | Encoder 1 - Master | page 184 |
| 32-5* | Feedback Source | page 188 |
| 32-6* | PID controller | page 189 |
| 32-8* | Velocity & Acceleration | page 191 |

| | | |
|---|---|---|
| **33-**** | **MCO Advanced Settings** | |
| 33-0* | Home Motion | page 193 |
| 33-1* | Synchronization | page 194 |
| 33-4* | Limit Handling | page 204 |
| 33-5* | I/O Configuration | page 205 |

| | | |
|---|---|---|
| **34-**** | **MCO Data Readouts** | page 211 |
| 34-0* | PCD Write Parameters | page 211 |
| 34-2* | PCD Read Parameters | page 211 |
| 34-4* | Inputs & Outputs | page 212 |
| 34-5* | Process Data | page 212 |

### □ General Information on the Parameter Values

Some limiting values are listed at 1 billion to make them more easily readable. However, the exact value is 1,073,741,823. (= MLONG).

### □ Input Range

Whether the input range listed is exceeded is not checked by the program, since due to the large domain there are no suitable test possibilities.

**NB!:**
Even within the areas indicated illogical inputs can result from the large differences in performance of the motors and the wide variety of possible applications. Therefore, it is the responsibility of the programmer and the user to observe the performance ranges of the drive and of the system.

**NB!:**
If the value of the parameter is out of the range defined by the minimum and maximum value the command is not displayed and executed correct.

## □ MCO Basics Settings



### □ 32-0* Encoder 2 - Slave

Following parameters configures the interface for the Encoder 2:

#### 32-00 Incremental Signal Type

ENCODERTYPE

**Option**

| | | |
|---|---|---|
| | None | [0] |
| ∗ | RS422 (TTL/line driver) | [1] |
| | Sinusoidal 1Vpp | [2] |

**Function**

Specifies type of the incremental encoder connected to Encoder 2 interface (X55).

Select *None* [0] if no incremental encoder is connected.

Select *RS422 (TTL/Line driver)* [1] if a digital incremental encoder with an interface according to RS422 is connected.

Select *Sinusoidal 1Vpp* [2] if an analog incremental encoder with 1 V peak-peak signal is connected.

#### 32-01 Incremental Resolution

ENCODER

**Range [Unit]**

1 – MLONG [PPR] ∗ 1024

**Function**

The encoder resolution is used to calculate velocity in RPM (rounds per minute) as well as timeout for detection of the zero pulse in connection with HOME and INDEX.

Set the resolution of the incremental encoder connected to Encoder 2 interface (X55):

– Digital incremental encoder (32-00 = [1]): The resolution must be set in Pulses Per Revolution.

– Analog incremental encoder (32-00 = [2]): The resolution must be set in sinusoidal signal periods per revolution.

Encoder resolution can be found on encoder name-plate or datasheet.

Note: Maximum frequency of the encoder signal must not exceed 410 kHz.

Note: Parameter only visible when par. 32-00 ≠ 0

#### 32-02 Absolute Protocol

ENCODERABSTYPE

**Option**

| | | |
|---|---|---|
| ∗ | None | [0] |
| | SSI | [4] |
| | SSI with filter | [5] |

**Function**

Specifies type of the absolute encoder connected to Encoder 2 interface (X55).

Select *None* [0] if no absolute encoder is connected.

Select *SSI* [4] if an absolute encoder with SSI interface is connected.

Select *SSI with filter* [5] if an absolute encoder with SSI interface is connected and the communication/signal is unstable.

A leap in the position data is detected if it is larger than the encoder resolution/2. The correction is made by means of an artificial position value which is calculated from the last velocity. If the error continues for more than 100 read-outs (> 100 ms), there will be no further correction which will then indeed lead to a "position error" (error 108).

The total number of errors will be saved in an internal variable which can be read out via SYSVAR[16].

**NB!:**
The following commands cannot be used with absolute encoders:
DEF ORIGIN, HOME, INDEX, and WAITNDX.

**NB!:**
The IPOS command can only be used with absolute encoder if an external marker is used.

### 32-03 Absolute Resolution

ENCODERABSRES

**Range**

1 … MLONG ∗ 8192

**Function**

The encoder resolution is used to calculate velocity in RPM (rounds per minute).

Set the resolution of the absolute encoder connected to Encoder 2 interface (X55) in positions per revolution. Encoder resolution can be found on encoder nameplate or datasheet.

Note: Parameter only visible when par. 32-02 ≠ 0

### 32-05 Absolute Encoder Data Length

ENCODERDATLEN

**Range [Unit]**

8 – 37 [Bit] ∗ 25

**Function**

Specify the number of data bit's for the connected absolute encoder, see encoder data sheet. This is required for MCO 305 to generate the correct number of clock bit's.

Note: Parameter only visible when par. 32-02 ≠ 0

### 32-06 Absolute Encoder Clock Frequency

ENCODERFREQ

**Range [Unit]**

78.125 – 2000.000 [kHz] ∗ 262.000

**Function**

Specifies the frequency of the absolute encoder clock signal generated by MCO 305. Set a frequency appropriate for the connected encoder.

Note: Parameter only visible when par. 32-02 ≠ 0

### 32-07 Absolute Encoder Clock Generation

ENCODERCLOCK

**Option**

| | | |
|---|---|---|
| | Off | [0] |
| ∗ | On | [1] |

**Function**

Select whether encoder 0 shall generate an absolute encoder clock signal or not.

Select *Off* [0] if more MCO 305's are connected to the same absolute encoder. Only one device (MCO 305) is allowed to generate the clock signal and only one device (encoder or MCO 305) is allowed to generate the data signal when multiple MCO 305's are interconnected.

Select *On* [1] if the MCO 305 is connected to just one absolute encoder.

Note: Parameter only visible when par. 32-02 ≠ 0

### 32-08 Absolute Encoder Cable Length

ENCODERDELAY

**Range [Unit]**

0 – 300 m ∗ 0

**Function**

The absolute encoder (SSI) clock and data signals will be out of synchronization if the signal delay caused by the encoder cable is too long. MCO 305 is automatically compensating the cable delay when the cable length is known. The cable delay compensation is based on a cable delay of approximately 6ns ($6 * 10^{-9}$seconds) per meter.

Specify the total cable length (in meters) between MCO 305 and the absolute encoder.

Note: Parameter only visible when par. 32-02 ≠ 0

### 32-09 Encoder Monitoring

ENCODERMONITORING

**Option**

| | | |
|---|---|---|
| ∗ | Off | [0] |
| | On | [1] |

**Function**

Monitoring of open- and short-circuit of the encoder inputs can enabled or disabled.

Select *Off* [0] if hardware monitoring not is required.

Select *On* [1] if hardware monitoring is required.

An encoder error will issue error code 192.

## 32-10 Rotational Direction

POSDRCT

### Option

| | | |
|---|---|---|
| * | No action | [1] |
| | Reference reversed | [2] |
| | User units reversed (–1) | [3] |
| | UU and Reference reversed (–2) | [4] |

### Function

Normally a positive reference value brings about a positive change of the position. If this is not the case, the reference value can be reversed internally. The following options are available:

1 = No change, i.e. positive reference values produce positive encoder values.

2 = The sign of the reference value is reversed internally (plus becomes minus and vice versa). This is equal to a reversal of the motor leads, or a transposition of the A and B tracks on the encoder.

3 = The sign of the user unit is reversed. Thus, positive reference values produce positive encoder values which are indicated as negative values, however. This applies to all outputs (APOS, CPOS, …), all user inputs (POSA, POSR, …), and all synchronization factors as well as the velocities (CVEL, par. 33-03 *Velocity for Home Motion*).

4 = Same as [2], i.e. the sign of the reference value is reversed internally; in addition, the sign of the user unit is negated as in [3].

The direction of rotation (relation to master) can be turned by negative par. 33-10 *Synchronizing Factor Master*.

## 32-11 User Unit Denominator

POSFACT_N

### Range

1 – MLONG * 1

### Function

All path information in motion commands is made in user units and are converted to quad-counts internally. By choosing these scaling units correspondingly it is possible to work with any technical measurement unit (for example mm).

This factor is a fraction which consists of a numerator and denominator.

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

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.

In CAM mode, the parameter is used to determine the unit for the slave drive so that it is possible to work with meaningful units in the CAM-Editor. See prerequisites of the formula and example under par. 32-12 *User Unit Numerator*.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}}\ \text{qc} = 1\ \text{UU}$$

In addition, it is possible to compress or expand the curves with this factor without having to define new curves each time. The use of numerator and denominator for the gearing factor leads to a very precise result since transmission ratios can be represented as a fraction in virtually all cases.

## 32-12 User Unit Numerator

POSFACT_Z

### Range

1 – MLONG/max. position (UU) * 1

### Function

All path information in motion commands are made in user units and are converted to quad-counts internally. By choosing these scaling units correspondingly it is possible to work with any technical measurement unit (for example mm).

This factor is a fraction which consists of a numerator and denominator.

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

Scaling determines how many quad-counts make up a user unit.

In CAM-Mode, the parameter is used to fix the unit for the slave drive so that it is possible to work with meaningful units in the CAM-Editor. See example 2.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}}\ \text{qc} = 1\ \text{UU}$$

provided that: $\text{Gearing Factor} = \frac{\text{Motor Revolutions}}{\text{Revolutions on Output}}$

Encoder = Incremental encoder (in the case of absolute encoders, the multiplier 4 is omitted)

Scaling factor = Number of user units UU (qc) that correspond to one revolution at the drive

In addition, it is possible to compress or expand the curves with this factor without having to define new curves each time. The use of numerator and denominator for the gearing factor leads to a very precise result since transmission ratios can be represented as a fraction in virtually all cases.

**Example 1**

Shaft or spindle

25 motor revolutions result in 1 spindle revolution; gearing factor = 25/1

Encoder resolution (incremental encoder) = 500

Spindle gradient = 1 revolution of the spindle = 5 mm

Scaling factor in case of working with 1/10 mm resolution = 5 * 10 = 50

$$\frac{25/1 * 500 * 4}{50} \text{qc} = \frac{25 * 10 * 4}{1} \text{qc} = \frac{1000}{1} \text{qc} = 1 \text{UU}$$

Par. 32-12 *User Unit Numerator* = 1000
Par. 32-11 *User Unit Denominator* = 1

**Example 2**

Cylinder:

Gear factor = 5/1

Encoder resolution (incremental encoder) = 500

One revolution of the cylinder is 360 degrees. It is requested to work with a resolution of 1/10 degrees. This means that one revolution of the cylinder is divided into 3600 units.

Scaling factor = 3600

$$\frac{5/1 * 500 * 4}{3600} \text{qc} = \frac{5 * 500 * 4}{3600} \text{qc} = 1 \text{UU}$$

$$= \frac{25}{9} \text{qc} = 1 \text{UU} = \frac{\text{par. 32 - 12 } \textit{User Unit Numerator}}{\text{par. 32 - 11 } \textit{User Unit Denomintor}}$$

par. 32-12 *User Unit Numerator* = 25
par. 32-11 *User Unit Denominator* = 9

## □ 32-3* Encoder 1 - Master

Following parameters configures the interface for the encoder 1:

### 32-30 Incremental Signal Type

MENCODERTYPE

**Option**

| | | |
|---|---|---|
| | None | [0] |
| * | RS422 (TTL/line driver) | [1] |

**Function**

Specifies type of the incremental encoder connected to Encoder 1 interface (X56).

Select *None* [0] if no incremental encoder is connected.

Select *RS422 (TTL/Line driver)* [1] if a digital incremental encoder with an interface according to RS422 is connected.

### 32-31 Incremental Resolution

MENCODER

**Range [Unit]**

| | |
|---|---|
| 1 – MLONG [PPR] | * 1024 |

**Function**

Set the resolution of the incremental encoder connected to Encoder 1 interface (X56):

Digital incremental encoder (32-00 = [1]): The resolution must be set in Pulses Per Revolution.

Encoder resolution can be found on encoder nameplate or datasheet.

Note: Maximum frequency of the encoder signal must not exceed 410 kHz.

Note: Parameter only visible when par. 32-30 ≠ 0

### 32-32 Absolute Protocol

MENCODERABSTYPE

**Option**

| | | |
|---|---|---|
| * | None | [0] |
| | SSI | [4] |
| | SSI with filter | [5] |

**Function**

Specifies type of the absolute encoder connected to Encoder 1 interface (X56).

Select *None* [0] if no absolute encoder is connected.

Select *SSI* [4] if an absolute encoder with SSI interface is connected.

Select *SSI with filter* [5] if an absolute encoder with SSI interface is connected and the communication/signal is unstable.

Virtual master: It is possible to simulate a master by means of APOSS command with the encoder type [5], for example when the master position is read via the bus. The simulated master positions are set and read with the system variable SYSVAR[4105].

**NB!:**
The MIPOS command can only be used with absolute encoder if an external marker is used.

### 32-33 Absolute Resolution

MENCODERABSRES

**Range [Unit]**

1 – MLONG [PPR] * 8192

**Function**

Note: Parameter only visible when par. 32-32 ≠ 0

### 32-35 Absolute Encoder Data Length

MENCODERDATLEN

**Range [Unit]**

8 – 37 [Bit] * 25

**Function**

Specify the number of data bit's for the connected absolute encoder, see encoder data sheet. This is required for MCO 305 to generate the correct number of clock bit's.

Note: Parameter only visible when par. 32-32 ≠ 0

### 32-36 Absolute Encoder Clock Frequency

MENCODERFREQ

**Range [Unit]**

78.125 – 2000.000 [kHz] * 262.000

**Function**

Specifies the frequency of the absolute encoder clock signal generated by MCO 305. Set a frequency appropriate for the connected encoder.

Note: Parameter only visible when par. 32-32 ≠ 0

### 32-37 Absolute Encoder Clock Generation

MENCODERCLOCK

**Option**

| | | |
|---|---|---|
| | Off | [0] |
| * | On | [1] |

**Function**

Select whether encoder 1 shall generate an absolute encoder clock signal or not.

Select *Off* [0] if more MCO 305's are connected to the same absolute encoder or if it is connected to another MCO 305 where the absolute virtual master is active. Only one device (MCO 305) is allowed to generate the clock signal and only one device (encoder or MCO 305) is allowed to generate the data signal when multiple MCO 305's are interconnected.

Select *On* [1] if the MCO 305 is connected to just one absolute encoder.

Note: Parameter only visible when par. 32-32 ≠ 0

### 32-38 Absolute Encoder Cable Length

MENCODERDELAY

**Option**

0 – 300 m * 0

**Function**

The absolute encoder (SSI) clock and data signals will be out of synchronization if the signal delay caused by the encoder cable is too long. MCO 305 is automatically compensating the cable delay when the cable length is known. The cable delay compensation is based on a cable delay of approximately 6 ns ($6 * 10^{-9}$ seconds) per meter.

Specify the total cable length (in meters) between MCO 305 and the absolute encoder.

Note: Parameter only visible when par. 32-32 ≠ 0

### 32-39 Encoder Monitoring

MENCODERMONITORING

**Option**

| | | |
|---|---|---|
| * | Off | [0] |
| | On | [1] |

**Function**

Monitoring of open- and short-circuit of the encoder inputs can enabled or disabled.

Select *Off* [0] if hardware monitoring not is required.

Select *On* [1] if hardware monitoring is required.

An encoder error will issue (error 192).

### 32-40 Encoder Termination

MENCODERTERM

**Option**

| | | |
|---|---|---|
| | Off | [0] |
| * | On | [1] |

**Function**

Termination resistors can be switched on or off for encoder 1.

Select *Off* [0] if high input impedance is required when:

- One encoder is connected to multiple MCO 305
- The virtual master output of one MCO 305 is connected to multiple MCO 305.

Select *On* [1] when the encoder is only connected to this MCO 305.

See MCO 305 Operating Instructions for wiring diagram.

## □ 32-5* Feedback Source

### 32-50 Source Slave

**Option**

| | | |
|---|---|---|
| * | Enc 2 | [2] |
| | Motor Control | [3] |

**Function**

Choose the feedback source for MCO. Choose [2] for Enc2. When using motor control principle "Flux with motor feedback" (Par. 101) it is possible to use the flux feedback source (Par. 102) for the MCO 305. Choose [3] for MCO 305 feedback from feedback source given in Par. 102. This can be internal 24V encoder, Encoder option, or resolver option.

### □ 32-6* PID-Controller

Optimize the controller using the following control parameters:

#### 32-60 Proportional Factor

KPROP

**Range**

0 – 100000 ∗ 30

**Function**

The *Proportional Factor* KPROP indicates the linear correction factor with which the deviation between the current set and actual position is evaluated and a corresponding correction of the motor speed is made.

Rule of Thumb:
KPROP greater = Drive will become 'stiffer'
KPROP too high = Tendency to overswing

#### 32-61 Derivative Value for PID Control

KDER

**Range**

0 – 100000 ∗ 0

**Function**

The *Derivative Value* is the correction factor with which the changing speed of a motor position error is evaluated.

The *Derivative Value* works against the tendency to overswing due to a high P-share and 'dampens' the system. However, if the *Derivative Value* selected is too large this will lead to a 'nervous' drive.

#### 32-62 Integral Factor

KINT

**Range**

0 – 100000 ∗ 0

**Function**

The *Integral Factor* KINT is the weighting factor, with which at time n the sum of all motor position errors are evaluated.

The *Integral Factor* of the PID filter causes a corresponding corrective motor torque which increases over time. Through the integral share a static position error is reduced to zero, even if a constant load is affecting the motor.

However, an *Integral Factor* which is too large leads to a 'nervous' drive.

#### 32-63 Limit Value for Integral Sum

KILIM

**Range**

0 – 1000 ∗ 1000

0 = Integral off

**Function**

This parameter limits the integral sum in order to avoid instability and PID wind-up in case of feedback error.

#### 32-64 PID Bandwidth

BANDWIDTH

**Range [unit]**

0 – 1000 [1/10 %] ∗ 1000

0 = PID off

**Function**

The value 1000 means that the PID filter can output the full command value. For a *Bandwidth* of 500 only 50 % of the set value is output. Thus, values less than 1000 limit the P-share accordingly.

The bandwidth in which the PID controller should function can be limited, for example to avoid the built-up of a vibration in case of a system which could be jeopardized by vibrations.

However, then it is necessary to enter considerably higher values for the parameters 32-65 *Velocity* and 32-66 *Acceleration Feed-forward* in order to achieve the corresponding control. A system adjusted in such a manner is not as dynamic as it could be, but is considerably more stable and tends to experience less uncontrolled vibrations.

#### 32-65 Velocity Feed-forward

FFVEL

**Range**

0 – 100000 ∗ 0

**Function**

When a control has a limited *Bandwidth* then a base velocity must be set so that it can be ruled out that the control will entirely prevent the drive from running due to the limit set.

*Velocity Feed-forward* indicates the value with which the velocity forward feed is completed.

When working with a normal PID algorithm the FFVEL must always be the same as the D factor in order to achieve typical dampening D.

### 32-66 Acceleration Feed-forward

FFACC

**Range**

| | |
|---|---|
| 0 – 100000 | ∗ 0 |

**Function**

Set the base acceleration whenever you have limited the bandwidth. Thus you will prevent the control from not accelerating at all due to the limit set. *Acceleration Feed-forward* indicates the value with which the acceleration forward feed is completed.

For a normal PID algorithm this value is equal to 0.

### 32-67 Max. Tolerated Position Error

POSERR

**Range [unit]**

| | |
|---|---|
| 1 – MLONG [qc] | ∗ 20000 |

**Function**

The *Maximum Tolerated Position Error* defines the tolerance allowed between the current actual position and the calculated command position. If the value defined with POSERR is exceeded then the position control is turned off and a position error is triggered.

The *Position Error* does not affect the positioning accuracy, but merely determines how precisely the theoretically calculated path of motion must be followed, without an error being triggered.

**NB!:**
For safety reasons the *Position Error* selected should not be too large since this could be dangerous for both the machine and its operator.

**NB!:**
On the other hand, if the values for the *Maximum Tolerated Position Error* are too small this could result in frequent errors. As a guideline, it is wise to set the quadruple of encoder counts per revolution. This corresponds to one encoder rotation.

### 32-68 Reverse Behavior for Slave

REVERS

**Option**

| | | |
|---|---|---|
| ∗ | Reversing allowed | [0] |
| | Reversing only allowed when master is reversed | [1] |
| | Reversing blocked | [2] |

**Function**

REVERS determines the behavior while moving in reverse (moving in a negative direction): whether reverse is allowed, only allowed when the master is reversed or not allowed in general.

The corresponding limitation is always valid, i.e. for drive synchronization (SYNCP, SYNCV, SYNCM, SYNCC, etc.), for positioning commands (CVEL), and even for *Testrun*.

In order to prevent automatic reversing during the *Testrun* adjust the value to 1 or 2.

### 32-69 Sampling Time for PID control

TIMER

**Range [unit]**

| | |
|---|---|
| 1 – 1000 [ms] | ∗ 1 |

**Function**

The TIMER parameter determines the sampling time of the control algorithm. For example, increase the value of the factory settings

- for very low pulse frequency, such as from 1 to 2 qc per sampling time. You need at least 10 to 20 qc per sampling time.
- Or for very slow systems with a long dead time. If 1 ms is used here for control, large motors will vibrate.

Accordingly, the value should not be set higher than 1000 (= 1 s). This would be a very slow control.

**NB!:**
Note that is has a direct effect on the PID loop e.g. if you double the TIMER the par. 32-60 *Proportional Factor* has twice the effect.

### 32-70 Scan Time for Profile Generator

PROFTIME

**Option**

| | | |
|---|---|---|
| * | 1 ms | [1] |
| | 2 ms | [2] |
| | 3 ms | [3] |
| | 4 ms | [4] |
| | 5 ms | [5] |

**Function**

The Parameter gives the possibility to set the sample time for the profile generator, which is independent of the sample time for the PID controller.

For demanding control tasks in the background (SYNCP, SYNCM, SYNCC), the execution time of the APOSS program may rise drastically. In such cases, the scan time of the profile generator can be increased to [2] in order to have more time available for the APOSS program. Values higher than 2 ms provide hardly any benefits.

**NB!:**
The VEL, ACC, and DEC has to be set after a SET PROFTIME command.

### 32-71 Size of the Control Window (Activation)

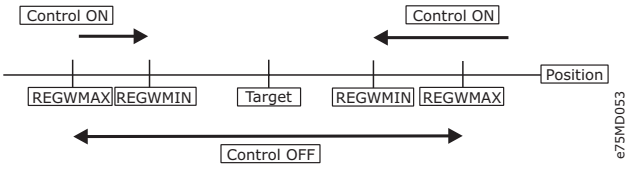
REGWMAX

**Range**

0 – MLONG [qc] * 0

**Function**

The parameters REGWMAX and REGWMIN are used to turn the position control within defined areas (*control windows*) on and off: REGWMAX indicates the size of the window outside of which the control should begin again.




### 32-72 Size of the Control Window (Deactiv.)

REGWMIN

**Range**

0 – MLONG [qc] * 0

**Function**

REGWMIN indicates the size of the window inside of which the control is to be deactivated until the par. 32-71 *Size of the Control Window (Activation)* is reached again.

## □ 32-8* Velocity & Acceleration

Use the following parameters to specify velocity, acceleration, and ramp.

### 32-80 Maximum Velocity (Encoder)

VELMAX

**Range [Unit]**

1 – 100000 [RPM] * 1500

**Function**

VELMAX defines the rated speed of the drive. This value is listed in RPM and is needed for the calculation of ramps and actual velocities.

**NB!:**
The nominal speed refers to the speed of the encoder.

### 32-81 Shortest Ramp

RAMPMIN

**Range [unit]**

0.001 – 3600.000 [s] * 1.000

**Function**

The RAMPMIN parameter determines the *Shortest Ramp* (maximum acceleration). It indicates how long the acceleration phase lasts at the very least in order to achieve the rated velocity.

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 300 ramps must always be set to minimum.

## 32-82 Ramp Type

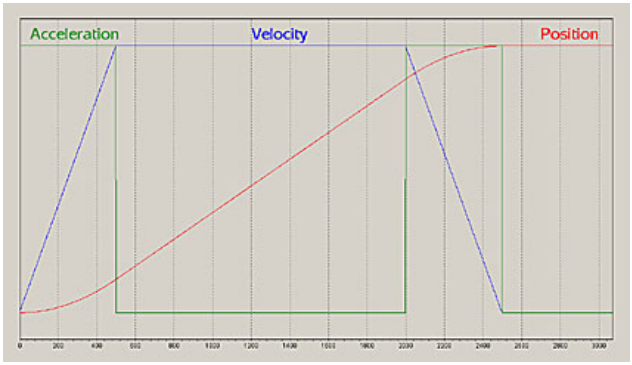
RAMPTYPE

### Option

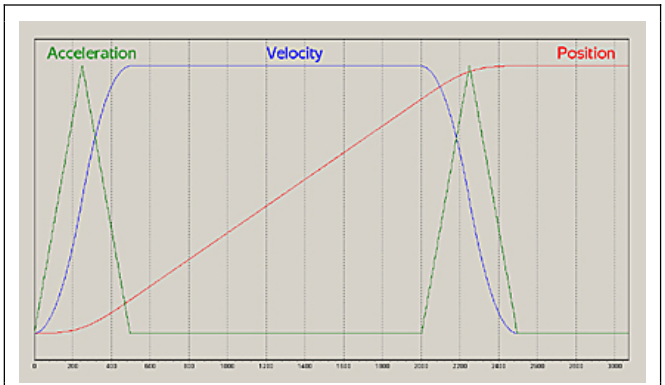
| | | |
|---|---|---|
| * | Linear | [0] |
| | S-ramp | [1] |
| | Movements with limited jerk | [2] |

### Function

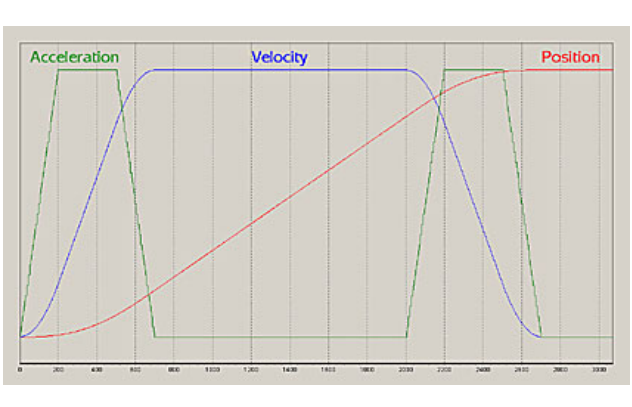
This parameter defines the ramp type: trapeze, sinusoidal, or limited jerk. These ramp type are relevant for all movements (POSA, POSR, CVEL, and MOTOR STOP), but not with SYNCx.




**Ramp type 0: trapeze**




**Ramp type 1: S- ramp**




**Ramp type 2: limited jerk**

Movements with limited jerk start with acceleration zero and increase acceleration by maximum Jerk until the maximum acceleration which is defined by par. 32-81 *Shortest Ramp* is reached. Then the movement continues with maximum acceleration. At the end the acceleration will be decreased by maximum jerk until acceleration is zero again. The maximum jerk is calculated by the internal parameter *Jerk Duration* JERKMIN.

### Function of JERKMIN

JERKMIN defines the minimum time in [ms] required before reaching the maximum acceleration.

There are four different JERKMIN options, see "Limited Jerk" in chapter „Functions and Examples": JERKMIN, JERKMIN2, JERKMIN3, JERKMIN4.

The maximum Jerk used with par. 32-82 *Ramp Type* = 2 is calculated by JERKMIN using following formulas:

$$\text{Max. Acceleration} = \frac{\text{Maximum Velocity}}{\text{Par. 32 - 81 Shortest Ramp}}$$

$$\text{Maximum Jerk} = \frac{\text{Maximum Acceleration}}{\text{Par. Jerk Duration JERKMIN}}$$

Note, that par. 32-81 *Shortest Ramp* and *Jerk Duration* are time values in milliseconds.

Calculation sample:

Par. 32-80 VELMAX = 3000 [RPM]
Par. 32-01 ENCODER = 500 counts/rev [PPR]
Par. 32-81 RAMPMIN = 500 ms
Parrameter JERKMIN = 200 ms

This results in:

$$\text{VELMAX} = 3000 * 500 * \frac{4}{60} = 100{,}000 \text{ qc/s}$$

$$= 100 \text{ qc/ms}$$

$$\text{MaxAcc} = 200{,}000 \text{ qc/s}^2 = 0.2 \text{ qc/ms}^2$$

$$\text{MaxJerk} = 1{,}000{,}000 \text{ qc/s}^3 = 0.001 \text{ qc/ms}^3$$

**NB!:**
A changed setting of *Jerk Duration* is executed after the next movement command (POSA, POSR, CVEL, or MOTOR STOP) and used for the calculation of the acceleration. Thus it is possible in NOWAIT ON mode with a repeated POSA command to adapt during a movement online (i.e. without stop) the movement to new definition.

**NB!:**
If JERKMIN2, JERKMIN3 and/or JERKMIN4 is set to "0" this will default to the same value as JERKMIN.

**NB!:**
Parameter *Jerk Duration* JERKMIN it is not implemented in the velocity axis parameter dialog field and must be set therefore in the program, e.g.

```
SET JERKMIN 100      // time in ms until the
   // maximum acceleration is reached
SET RAMPTYPE 2
      // movement profile with limited jerk
```

**32-83 Velocity Resolution**

VELRES

**Range**

1 – 10000 ∗ 100

**Function**

The *Velocity Resolution* VELRES defines a relative size for the velocity values of the motion commands and parameters. The information concerning speed and acceleration can then be made in whole numbers in relation to this scaling. The value 100 means that the information in the commands are related to 100, thus in percent.

**32-84 Default Velocity**

DFLTVEL

**Range**

1 – p. 32-83 VELRES ∗ 50

**Function**

*Default Velocity* indicates the default velocity which is always used when no velocity is defined in the process set. This value refers to the *Velocity Resolution* VELRES.

**32-85 Default Acceleration**

DFLTACC

**Range**

1 – p. 32-83 VELRES ∗ 50

**Function**

*Default Acceleration* indicates the acceleration used when no explicit statements have been made. This statement is made in relation to par. 32-81 *Shortest Ramp* and refers to the par. 32-83 *Velocity Resolution*.

## □ MCO Advanced Settings



### □ 33-0* Home Motion

Use following parameters to specify behavior for home run and home motion:

**33-00 Force HOME**

HOME_FORCE

**Option**

| | | |
|---|---|---|
| ∗ | Home run is not forced | [0] |
| | Home forced | [1] |

**Function**

0 = After being turned on the current position is valid as the real zero point.

1 = After turning on the FC300 and after changing axis parameters a forced tracking of the home position must be made before a motion command is executed directly or by the program.

If this parameter is set to [1], then movement to the home position must be completed before any other positioning movement can be completed.

For a motion command that is not executed with a terminated home run the error 106 is triggered.

**NB!:**
For safety reasons and to avoid false positioning the parameter should always be set to [1] and thus forcing tracking of the home position. However, in this case it is necessary to consider that all programs must complete a HOME command before the first motion command in order to receive perfect functioning.

**33-01 Zero Point Offset from Home Position**

HOME_OFFSET

**Range [unit]**

-MLONG – MLONG [qc] ∗ 0

**Function**

HOME_OFFSET is used to introduce an offset compared to the reference switch or index pulse. After homing, the drive is positioned to HOME_OFFSET. At this point the machine zero point or index is set.

### 33-02 Ramp for Home Motion

HOME_RAMP

**Range**

1 – par. 32-83 VELRES ∗ 10

**Function**

Acceleration to be used during movement to home position. This statement refers to the minimum ramp, which is defined under the par. 32-81 *Shortest Ramp*. This unit results from the par. 32-83 *Velocity Resolution* usually in % of the minimal ramp; 50% means half as fast, i.e. twice as long.

The following cohesion for HOME_RAMP results:

$$\text{Ramp for Home Motion [ms]} = \frac{\text{p. 32 - 83 Velocity Resolution}}{\text{p. 33 - 02 HOME\_RAMP}} * \text{p. 32 - 81 Shortest Ramp [ms]}$$

**NB!:**
*Ramp for Home Motion* can never have a higher value than par. 32-85 *Default Acceleration*.

### 33-03 Velocity of Home Motion

HOME_VEL

**Range**

- p. 32-83 – p. 32-83 ∗ 10

**Function**

HOME_VEL determines the *Velocity of Home Motion*, with which the movement to the reference switch is made. The velocity statement refers to the rated speed and depends on the parameter 32-83.

A negative sign means the search will be made in the other direction.

$$\text{Home Velocity [RPM]} = \text{par. 33 - 03 Home Velocity} * \frac{\text{p. 32 - 80 Maximum Velocity}}{\text{p. 32 - 83 Velocity Resolution}}$$

**NB!:**
Since the program always searches for the reference switch in the same direction of rotation (depending on sign) this should be set at the limits of the motion area. Only in this manner is it possible to guarantee that the drive actually moves towards the reference switch when moving home and not away from it.

In order to maintain a good repeatability of the reference motion no more than 10% of the maximum speed should be used.

### 33-04 Behavior during Home Motion

HOME_TYPE

**Option**

| | | |
|---|---|---|
| ∗ | Reverse and index | [0] |
| | Reverse no index | [1] |
| | Forward and index | [2] |
| | Forward no index | [3] |

**Function**

0 = Moves to reference switch with *Velocity of Home Motion* and direction, then reverses and slowly leaves the switch, subsequently moves to the next index impulse.

1 = like 0, but does not search for index impulse

2 = like 0 but without reversing, rather continues movement in the same direction out of the switch

3 = like 1 but without reversing

## □ 33-1* Synchronization

Position, velocity and angle/position synchronization, with or without marker and more is possible with following parameters:

### 33-10 Synchronization Factor Master (M:S)

SYNCFACTM

**Range**

–MLONG – MLONG ∗ 1

–MLONG to –1 = turns the direction of synchronization (ratio to the master)

**Function**

The synchronization is described with a ratio of qc (Master : Slave); SYNCFACTM determines the synchronization factor for the master.

*Syncfactor Master* and par. 33-11 *Syncfactor Slave* make the compensation of different drive factors possible or the adaptation of the slave speed in relation to the master speed set.

$$\text{Slave Velocity} = \text{Master Velocity} * \frac{\text{par. 33 - 11 Syncfactor Slave}}{\text{par. 33 - 10 Syncfactor Master}}$$

In conjunction with curve synchronization the parameters SYNCFACTM and SYNCFACTS are used to transform qc into MU units.

This allows the user to work with meaningful units in the CAM-Editor. See example 2 below.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{qc} = 1 \text{ MU}$$

provided that:

$$\text{Gearing Factor} = \frac{\text{Motor Revolutions}}{\text{Revolutions on Output}}$$

Encoder = Incremental encoder (the multiplier 4 is omitted in the case of absolute encoders)

Scaling factor = Number of user units UU (qc) that correspond to one revolution at the drive.

**Example 1**

If the master is to run twice as fast as the slave, then the ratio is 2 : 1

par. 33-10 *Syncfactor Master* = 2
par. 33-11 *Syncfactor Slave* = 1

**Example 2**

Conveyor belt:

The input should be possible in 1/10 mm resolution.

The drive is connected to the conveyor belt with a gearing of 25:11; this means that the motor makes 25 revolutions and the drive pulley 11.

Gear factor = 25/11

Incremental encoder directly on the master drive; encoder resolution = 4096

The drive pulley has 20 teeth/revolution, 2 teeth correspond to 10 mm; thus, 1 revolution = 100 mm conveyor belt feed.
Thus, the scaling factor is 1000

$$\frac{25/11 * 4096 * 4}{1000} \text{qc} = \frac{25 * 4096 \times * 4}{1000 \times 11} \text{qc} = 1 \text{ MU}$$

$$= \frac{2048}{55} \text{qc} = 1 \text{ MU} = \frac{\text{par. 33 - 10 Syncfactor Master}}{\text{par. 33 - 11 Syncfactor Slave}}$$

Set the following parameters in order to work with 1/10 degree division

par. 33-10 *Syncfactor Master* = 2048
par. 33-11 *Syncfactor Slave* = 55

**Example 3**

Calculation of the scaling factor for a friction drive:

Assume that the output is equipped with a friction wheel (radius 60 mm); we want to work with a resolution of 1/10 mm:

One revolution on the output is thus calculated as follows:

Scaling factor = 2 Π r * 10 = 2 Π * 60 * 10
= 3969.91

Scaling factor = 3970

Since an error will occur in any case due to the rounding, a marker adjustment must be performed after each full revolution.

## 33-11 Synchronization Factor Slave (M:S)

SYNCFACTS

**Range**

–MLONG – MLONG ∗ 1

**Function**

The synchronization is described with a ratio of qc (Master : Slave); *Syncfactor Slave* determines the synchronization factor for the slave.

Parameters 33-10 *Synchronizing Factor Master* and 33-11 *Syncfactor Slave* make the compensation of different drive factors possible or the adaptation of the slave speed in relation to the master speed set.

$$\text{Slave Velocity} = \text{Master Velocity} * \frac{\text{par. 33 - 11 Syncfactor Slave}}{\text{par. 33 - 10 Syncfactor Master}}$$

In conjunction with CAM synchronization the parameters *Synchronizing Factor Master* and *Slave* are used to transform qc into MU. This allows the user to work with meaningful units in the CAM-Editor. See example 2 in par. 33-10.

See prerequisites of the formula in par. 33-10 *Synchronizing Factor Master*.

$$\frac{\text{Gearing Factor} * \text{Encoder Resolution} * 4}{\text{Scaling Factor}} \text{qc} = 1 \text{ MU}$$

**Examples**

See par. 33-10 *Synchronizing Factor Master*.

## 33-12 Position Offset for Synchronization

SYNCPOSOFFS

**Range [Unit]**

–MLONG/p. 33-11 SYNCFACTS – MLONG/p. 33-11 SYNCFACTS [qc] ∗ 0

**Function**

Defines the offset for position synchronization (SYNCM, SYNCP). The offset is also valid for position synchronization with marker correction.

This position offset can be altered online at any time during the synchronization with a command.

$$\text{Slave Offset} = \text{p. 33 - 12 SYNCPOSOFFS} * \frac{\text{par. 33 - 11 } \textit{Syncfactor Slave}}{\text{par. 33 - 10 } \textit{Syncfactor Master}}$$

The offset will be executed immediately when the command SYNCP follows.

When SYNCM is started, however, the system waits for the first evaluation of the marker pulses. Only then is the offset applied.

To avoid compatibility problems you should determine the start-up behavior of SYNCM with par. 33-23 *Start Behavior for Sync*.

## 33-13 Accuracy Window for Position Sync.

SYNCACCURACY

**Range [Unit]**

–MLONG – MLONG [qc] ∗ 1000

0 – MLONG = A plus sign supplies the absolute value to SYNCERR.

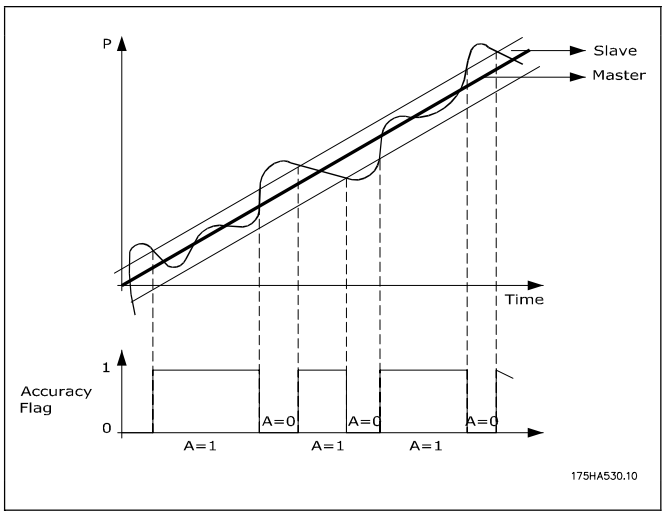
–MLONG to –1 = A minus sign supplies the synchronization error to SYNCERR with polarity sign. It is then possible to tell whether the synchronization is running ahead or behind.

**Function**

Defines how large the difference between the actual master and slave position can be during a position synchronization (SYNCP and SYNCM), so that the required accuracy is still fulfilled. In contrast SYNCERR provides the actual synchronization error of the slave in user units.

In the program you can query whether SYNCACCURACY will be fulfilled using SYNCSTAT.

SYNCACCURACY is important for the marker synchronization in order to be able to report READY, since otherwise SYNCERR would have to be queried and compared beforehand.




**Window set by SYNCACCURACY**

The dark line is the position followed by master and slave follows it. SYNCACCURACY sets the window as 100. So when the slave within the window the accuracy flag sets.

## 33-14 Relative Slave Velocity Limit

SYNCVELREL

**Range [Unit]**

0 – 100 [%] ∗ 0

0 = Off, i.e. no restriction

**Syntax**

SET SYNCVELREL value

value = percent value

**Function**

Tolerated deviance of the slave drive from the master velocity in %.

This parameter indicates by how many percent the slave drive can deviate from the velocity of the master while attempting re-synchronization.

For example: during changes in par. 33-12 *Position Offset for Synchronization,* or at the start of synchronization, or during the correction of deviation for marker evaluation. The following is valid:

If the slave need to catch up it runs with the maximum speed allowed; this is either the speed set with VEL or the master velocity calculated with MAVEL + (MAVEL * SYNCVELREL/100) depending which of the two is less. (MAVEL is the actual master velocity).

If the slave needs to slow down and wait for the master it will run with at least the following speed MAVEL – (MAVEL * SYNCVELREL/100).

That means, if SYNCVELREL is 50, for example, the slave will not run slower than MAVEL/2.

## 33-15 Marker Number for Master

SYNCMARKM

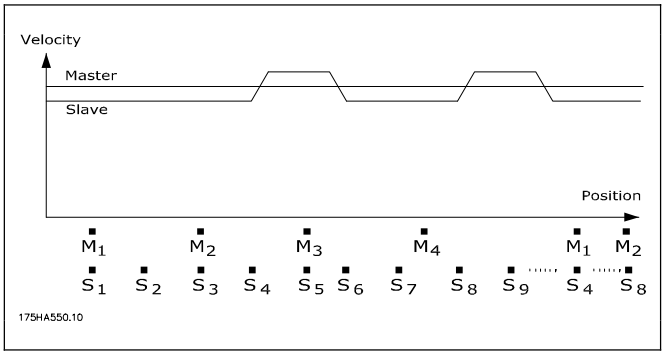
### Range

1 – 10000 ∗ 1

### Function

The *Marker Number for Master* and *Slave* must be set according to the ratio between the number of marker signals from master and slave.

A ration of 1:1 means that each slave marker will be aligned with each master marker. A ratio of 2:1 means that each slave marker will be aligned with each second master marker.




## 33-16 Marker Number for Slave

SYNCMARKS

### Range

1 – 10000 ∗ 1

### Function

The *Marker Number for Master* (par. 33-15) and *Slave* must be set according to the ratio between the number of marker signals from master and slave.

A ration of 1:1 means that each slave marker will be aligned with each master marker. A ratio of 2:1 means that each slave marker will be aligned with each second master marker.

### Example

The master marker is an external signal which reports when a transport article arrives; the corresponding slave marker is the index impulse of the motor. If the motor always requires 3 rotations until the article arrives, then this means that 3 index impulses must elapse before a marker comes. Thus, this results in a ratio of 3:1; only every 3rd slave pulse is evaluated.

## 33-17 Master Marker Distance

SYNCMPULSM

### Range [Unit]

0 – MLONG ∗ 4096
[qc] or in CAM mode [MU]

### Function

The *Master Marker Distance* indicates how many qc (master) lie between two master markers or in CAM-Mode the distance between sensor and working position.

If you use the encoder index impulse as a marker signal, then the distance between two markers is the resolution (qc) of the encoder.

If external marker signals are used, then it is possible to measure the marker distance with the program "marker count" (refer to sample program), if it is unknown.

*Master Marker Distance* is only valid for synchronization with marker correction (SYNCM and SYNCCMM).

In a CAM synchronization, the distance of the sensor to the working position in MU will be indicated instead of the distance between two master markers. (The distance is brought about automatically by the master cycle length (Mt).)

If the parameter is larger than one master cycle length (Mt), a marker-FIFO register will be created automatically for he marker correction handling.

### 33-18 Slave Marker Distance

SYNCMPULSS

**Range [Unit]**

0 – MLONG ∗ 4096
[qc] or in CAM mode [UU]

**Function**

The *Slave Marker Distance* defines how many qc (slave) lie between two markers (slave) or in CAM-Mode the distance of the sensor to the working position in UU.

*Slave Marker Distance* is only valid for synchronization with slave marker correction (SYNCM and SYNCCMS).

### 33-19 Master Marker Type

SYNCMTYPM

**Option**

| | | |
|---|---|---|
| ∗ | Encoder Z positive flank | [0] |
| | Encoder Z negative flank | [1] |
| | External marker positive flank | [2] |
| | External marker negative flank | [3] |

**Function**

Defines the signal type for the master marker: index pulse of the encoder or external marker.

*Master Marker Type* is only valid for synchronizations with marker correction (SYNCM and SYNCCMM) or if the command MIPOS should be used in the program.

External master marker signal: Input 5.

### 33-20 Slave Marker Type

SYNCMTYPS

**Option**

| | | |
|---|---|---|
| ∗ | Encoder Z positive flank | [0] |
| | Encoder Z negative flank | [1] |
| | External marker positive flank | [2] |
| | External marker negative flank | [3] |

**Function**

Defines the signal type for the slave marker: index pulse of the encoder or external marker.

*Slave Marker Type* is only valid for synchronizations with marker correction (SYNCM and SYNCCMS) or if the command MIPOS should be used in the program.

External slave marker signal: Input 6

### 33-21 Master Marker Tolerance Window

SYNCMWINM

**Range [Unit]**

0 – par. 33-17 ∗ 0
*Master Marker Distance*

0 = Off

[qc] or in CAM mode [MU]

**Function**

The *Master Marker Tolerance Window* shows how large the permitted tolerance for the occurrence of the markers is.

With the default setting [0] the window is not monitored, which means that it is always synchronized to the next marker even if this has a considerably larger interval.

At every other setting only those markers are accepted which are within the window. If there is no marker within the tolerance window the corresponding flag (SYNCSTAT) is set and no marker correction takes place. The corresponding other marker is also ignored and only corrected the next time, i.e. no catching up to the next marker.

When SYNCM (or SYNCCSTART) is started the monitoring only begins when the first marker has been found.

**NB!:**
Changes of the parameter will become active immediately – not only after the next SYNCM command.

**Example**

Par. 33-17 Master Marker Distance = 30000
Master Marker Tolerance Window = 1000

Only one marker within an interval of 29000 to 31000 is accepted.

## 33-22 Slave Marker Tolerance Window

SYNCMWINS

**Range [Unit]**

0 – par. 33-18 *Slave Marker Distance* ∗ 0

0 = Off

[qc] or in CAM mode [UU]

**Function**

The *Slave Marker Tolerance Window* SYNCMWINS shows how large the permitted tolerance for the occurrence of the markers is.

With the default setting [0] the window is not monitored, which means that it is always synchronized to the next marker even if this has a considerably larger interval.

At every other setting only those markers are accepted which are within the window. If there is no marker within the tolerance window the corresponding flag (SYNCSTAT) is set and no marker correction takes place. The corresponding other marker is also ignored and only corrected the next time – i.e. no catching up to the next marker.

When SYNCM (or SYNCCSTART) is started the monitoring only begins when the first marker has been found.

**NB!:**
Changes of the parameter will become active immediately – not only after the next SYNCM command.

## 33-23 Start Behavior for Sync.

SYNCMSTART (with Marker Correction)

**Option**

| | | |
|---|---|---|
| ∗ | Start Function 1 | [0] |
| | Start Function 2 | [1] |
| | Start Function 3 | [2] |
| | Start Function 4 | [3] |
| | Start Function 5 | [4] |
| | Start Function 6 | [5] |
| | Start Function 7 | [6] |
| | Start Function 8 | [1000] |
| | Start Function 9 | [1001] |
| | Start Function 10 | [1002] |
| | Start Function 11 | [1003] |
| | Start Function 12 | [1004] |
| | Start Function 13 | [1005] |
| | Start Function 14 | [1006] |
| | CAM Master Start | [2000] |

**Function**

SYNCMSTART defines whether at start the synchronization should be made to the leading, subsequent or closest marker impulse of the master.

SYNCMSTART is only valid for synchronization with marker correction (SYNCM and SYNCCMM).

0 = The slave marker following the first master marker (after SYNCM) is aligned with the first master marker.

1 = The first slave marker (after SYNCM) is aligned with the following master marker.

2 = After reaching the master velocity the next 2 markers will be aligned (correction can be forward or backward).

3 = After reaching the master velocity the next slave marker will be aligned with the master marker in front (correction is forward).

4 = After reaching the master velocity the next slave marker will be aligned with the marker behind (correction is backward).

5 = After reaching the master velocity the next slave marker will be aligned with the closest master marker (correction can be forward or backward, always the shortest distance).

6 = After the command SYNCM the first two markers are taken and the program synchronizes to these markers.

1000 = as [0], but an offset (par. 33-12 SYNCPOSOFFS) is not active before the first marker correction is done.

1001 = as [1], but an offset is not active before the first marker correction is done.

1002 = as [2], an offset is not active before the first marker correction is done.

1003 = as [3], an offset is not active before the first marker correction is done.

1004 = as [4], an offset is not active before the first marker correction is done.

1005 = as [5], an offset is not active before the first marker correction is done.

1006 = as [6], an offset is not active before the first marker correction is done.

2000 = Counting of the master pulses in MU begins with the master marker.

**NB!:**
Only the parameter 2000 is effective in curve synchronizations.

### 33-24 Marker Number for Fault

SYNCFAULT

**Range**

0 – 10000 ∗ 10

**Function**

Defines how often during a marker synchronization (SYNCM and SYNCCMM) an inaccuracy may occur during a synchronization evaluation before a FAULT is registered.

In the program this condition can be queried using SYNCSTAT.

### 33-25 Marker Number for Ready

SYNCREADY

**Range**

0 – 10000 ∗ 1

**Function**

SYNCREADY defines how often during a marker synchronization (SYNCM and SYNCCMM) a synchronization evaluation with ACCURACY must be completed with accuracy so that ready is fulfilled.

ACCURACY is checked during every correction. If ACCURACY is fulfilled then 1 is added until the set marker number has been achieved.

Synchronization evaluation is always executed after m marker pulses by the master par. 33-15 *Marker Number for Master*.

ACCURAY and READY can be queried using SYNCSTAT.

### 33-26 Velocity Filter

SYNCVFTIME

**Range [Unit]**

–MLONG – MLONG [µs] ∗ 0

–999 – 999 = standard table

**Standard Table**

| encoder resolution | τ_filt [µs] |
|---|---|
| 250 | 39500 |
| 256 | 38600 |
| 500 | 19500 |
| 512 | 19000 |
| 1000 | 9500 |
| 1024 | 9300 |
| 2000 | 4500 |
| 2048 | 4400 |
| 2500 | 3500 |
| 4096 | 1900 |
| 5000 | 1400 |

**Function**

This parameter configures the velocity filter which is used for the velocity synchronization. Since the velocity synchronization only uses the currently active master velocity and the values can decrease to very small values (e.g. 2 qc/ms), a small fluctuation in velocity can have dramatic effects. In order to even this out the following filter function is applied:

Cmdvel =
Old_Cmdvel + (Actvel – Old_Cmdvel) * ms/τ_filt

With the following:

| | |
|---|---|
| Cmdvel | = set velocity |
| Old_Cmdvel | = last set velocity |
| Actvel | = actual velocity of the master |
| ms | = sampling time (fixed 1ms) |
| τ_filt | = filter time constants |

Generally the value for τ_filt is taken from a table, depending on the Encoder counts per revolution of the master. This value can be overwritten by the parameter *Velocity Filter* and is always used when *Velocity Filter* is not equal zero.

If the speed filter is defined with a negative number, the corresponding value also applies for angle/position synchronization SYNCP and for marker correction SYNCM.

In this case filtering takes place as described above, but the errors made are summed up. This error sum is taken into the calculation with 1000/(τ*10) in each case, so that no position deviation can occur over prolonged periods.

The value returned by SYNCERR always contains the error made so that this is also used for the evaluation of the synchronicity. In the case of marker correction the correction value is balanced more slowly and with the same factor as the error sums.

If, for example, a filter factor of –100000 (100 ms) is used, a marker correction is balanced within 1 second (100 ms * 10). This allows a 'taming' of the synchronization without restricting the acceleration.

### 33-27 Offset Filter Time

SYNCOFFTIME

**Range [Unit]**

0 – MLONG [ms] ∗ 0

**Function**

Compensation velocity of an offset (1. synchronize; 2. new offset)

The *Offset Filter Time* also influences the way, how a new *Position Offset for Synchronization* (par. 33-12) value is handled. The offset which has to be realized will be done step by step. The step which is realized every sample time (ms) is calculated as follows:

$$\text{step size} = \frac{\text{par. 33 - 17 Master Marker Distance}}{\text{par. 33 - 27 Offset Filter Time (integer part)}}$$

So it will take *Offset Filter Time* to realize an offset of par. 33-17 *Master Marker Distance*. The *Offset Filter Time* also has influence on the marker start correction and on the correction of marker error (see par. 33-29 *Filter Time for Marker Correction*).

### 33-28 Marker Filter Configuration

SYNCMFPAR

**Option**

| | | |
|---|---|---|
| ∗ | Normal SYNCMFTIME filter function | [0] |
| | Constant value for the Marker Filter | [1] |
| | No correction of SYNCFACT | [2] |
| | Time constant based on *Filter Time for Marker Correction* SYNCMFTIME | [4] |
| | Only smoothing of the correction value, time constant as [4] | [16] |
| | Execute always marker distance averaging. | [64] |

**Function**

SYNCMFPAR is used to influence the behavior of marker filtering, see SYNCMFTIME.

The following values are bit valences and can be combined with each other:

0 = Normal filter function, see par. 33-29 *Filter Time for Marker Correction* SYNCMFTIME

1 = Instead of the dynamic marker filter constant a constant value of SYNCMFTIME / 300 is used.

2 = Gear correction is not executed.

4 = The *Filter Time for Marker Correction* (par. 33-29) is used instead of the *Offset Filter Time* (par. 33-27) to calculate the time constant for the correction value filter (G_Korrektur)

16 = Instead of calculation of the filtered marker distance and deviation, only the correction value is smoothed by a PT filter. Time constant for that filter on bit valence 4 basis (see above).

64 = Marker averaging and marker check are also executed when SYNCM is inactive.

### 33-29 Filter Time for Marker Correction

SYNCMFTIME

**Range [Unit]**

0 – MLONG [ms] ∗ 0

0 = Off;
if par. 33-26 *Velocity Filte*r is negative, the marker correction is spread by SYNCVFTIME /100

**Application Example**

Newspaper manufacturing needed this sort of filtering to synchronizing a chain to the newspaper stream coming from a printing machine. Because the newspaper stream is not quite constant, the problem is that if synchronized without filter the movements of the chain are very hard and dynamic. With all other sort of filters the system starts swinging in sinusoidal waves.

When using this complex filter the synchronizing work very well and solve the problem.

**Function**

SYNCMFTIME is given in ms and is used as follows:

**NB!:**
Master velocity filtering par. 33-26 *Velocity Filte*r is given in 1/1000 ms for a better resolution, but the marker filtering (SYNCMFTIME) is given in units of 1 ms.

Example:

SET SYNCVFTIME –50000
SET SYNCMFTIME 2000

This means, that the master velocity is filtered over a period of 50 ms. A marker error is corrected within a period of 2000 ms.

The actual filtered marker distance can be read out with SYSVAR 4238 indices if this filter is activated by setting SYNCMFTIME.

The *Filter Time for Marker Correction* and parameters 33-27 *Offset Filter Time* and 33-28 *Marker Filter Configuration* are used to influence the behavior of Marker Filtering (see below).

**Filtering Description**

Filtering is handled as follows:

Calculation of Marker Filter only if SYNCMFTIME > 0

If SYNCMFPAR = 1,

Every time when a real master marker is found, the Marker Filter constant can be calculated as SYNCMFTIME/300,

if SYNCMFPAR = 0

Every time when a real master marker is found, the Marker Filter constant can be calculated as the Filtered Old Master Velocity * SYNCMFTIME / (SYNCMPULSM * 3)

which means, that the Marker Filter constant is used as time constant for filtering. Then the time which is needed to get an output corresponding to a steady input should be nearly SYNCMFTIME.

The calculation is necessary, because the filter is executed every marker and not every ms.

This Marker Filter constant is now used when filtering the marker distance. The result is then used to calculate the necessary gear correction.

The Gear Correction can be calculated as follow:

Gear correction = (SYNCMPULSM – Filtered marker distance) / Filtered marker distance

Filtering master velocity and gear correction

Every sample time when the filtered master_velocity (difference of actual and last master position) is calculated

IF (SYNCVFTIME < 0)

then Filtered Old Master Velocity is calculated with a filter time constant equal SYNCVFTIME / 1000

Else the Filtered Old master velocity is set equal to the actual master velocity.

In case where

SYNCMFTIME > 0 and

SYNCMFPAR = 2

the gear correction is made by taking the current gear ratio and add the master velocity multiplied by the Gear correction.

Start correction only if SYNCMFTIME > 0

Start correction is the correction which must be realized after start condition is fulfilled. That means either the first two markers (par. 33-23 SYNCMSTART 1,6) were observed or the master velocity is reached and the first two markers (SYNCMSTART 2,3,4,5) has been observed.

This start correction is split in such a way, that it will be eliminated after par. 33-27 SYNCOFFTIME. (Actually it is divided by the amount of markers, which will be passed in SYNCOFFTIME at the actual master velocity and that value is added to the normal marker correction.)

If SYNCOFFTIME == 0,

Start correction will be eliminated at once, which means that the correction will be done in between two markers.

Marker correction SYNCMFTIME > 0

First the remaining start correction is subtracted from the marker error. Then correction filtering time corresponding to the par. 33-27 SYNCOFFTIME (master velocity dependant see start correction number of markers) is set.

Now the sum of all marker distance errors is used for a marker filter to calculate the filtered sum. Then the filtered error sum is subtracted from the unfiltered one. This result is then used to correct the marker correction.

This corrected correction is then given into the correction filter. The result of this correction filter is then stored (plus start correction part if necessary).

Then this correction is spread over one marker distance. This is done by dividing the correction by the number of samples which will be necessary to pass one marker distance at actual master speed. This value is stored and will be used every sample time to correct the calculated slave position.

Following par. 33-28 SYNCMFPAR settings modify behavior:

| | |
|---|---|
| SYNCMFPAR & 4 → | Correction Time Is used instead of using par. 33-27 SYNCOFFTIME. |
| SYNCMFPAR & 16 → | No correction concerning the error of marker distances will be done. |

Marker correction SYNCMFTIME == 0

In the first case where marker correction > 0, the correction is spread over a time of (-SYNCVFTIME / 100) ms.

In the second case the correction is added to the demand position at once.

The reaction of course is limited by the actual acceleration and deceleration in every case.

### 33-30 Maximum Marker Correction

SYNCMMAXCORR

**Range [Unit]**

0 – MLONG [qc] ∗ 0

0 = Off, i.e. no limit

**Function**

SYNCMMAXCORR is used to limit the maximum correction done by marker correction. The value is given in qc (slave). This is working with SYNCM and SYNCC.

After the PFG_G_KORREKTUR is calculated, the PFG_G_KORREST is set to the minimum of G_Korrektur or SYNCMMAXCORR. This value will then be used for correction.

**NB!:**
If you have set par. 33-29 *Filter Time for Marker Correction* or par. 33-26 *Velocity Filte*r (negative), this correction will be spread over a certain time, depending on these factors.

### 33-31 Synchronization Type

SYNCTYPE

**Option**

| | | |
|---|---|---|
| ∗ | Standard | [0] |
| | Look ahead | [1] |

**Function**

The way how synchronization is done can be changed:

0 = The actual master position is compared with the future slave position (where slave will be in 1 ms).

1 = The actual master position is compared with actual commanded position.

In the standard case (SYNCTYPE=0) the position difference is eliminated.

That means, that the actual master position (where master is now) is compared with the future slave position (where slave will be in 1 ms). Thus it will always drive behind the master, as long as no INTEGRAL is used.

If SYNCTYPE = 1 is selected the system compares actual master position with actual commanded position. That means, that the system will try to make this zero, no matter how PID is set.

**NB!:**
Be aware, that SYNCERR equals to actual (new) master command position minus actual position of the slave plus pending errors of filtering and pending corrections.

### □ 33-4* Limit Handling

Parameters for determination the limit switches behavior.

#### 33-40 Behavior at End Limit Switch

ENDSWMOD

**Option**

| | | |
|---|---|---|
| * | Call error handler | [0] |
| | Controlled stop | [1] |

**Function**

This parameter defines the behavior when a positive or negative hardware end limit is activated.

*Behavior after Error* see par. 33-83 (ERRCOND).

#### 33-41 Negative Software End Limit

NEGLIMIT

**Range [Unit]**

| | |
|---|---|
| -MLONG – MLONG [qc] | * -500000 |

**Function**

NEGLIMIT indicates the negative position limit for all movements. If this value is exceeded then an error is triggered. NEGLIMIT is only active if par. 33-43 SWNEGLIMACT has been set.

If a positioning command is entered which exceeds the limits set, then it is not executed.

**NB!:**
When using the command DEF ORIGIN the path limitation is automatically adapted so that the original position of the positioning range is maintained.

**NB!**:
The path limitation is always given in quadcounts.

#### 33-42 Positive Software End Limit

POSLIMIT

**Range [Unit]**

| | |
|---|---|
| -MLONG – MLONG [qc] | * 500000 |

**Function**

POSLIMIT indicates the Positive position limit for all movements. If this value is exceeded then an error is triggered.

POSLIMIT is only active if par. 33-44 SWPOSLIMACT is set. If a positioning command is entered which exceeds the limits set, then it is not executed.

**NB!:**
When using the command DEF ORIGIN the path limitation is automatically adapted so that the original position of the positioning range is maintained.

**NB!**:
The path limitation is always given in quadcounts.

#### 33-43 Negative Software End Limit Active

SWNEGLIMACT

**Option**

| | | |
|---|---|---|
| * | Inactive | [0] |
| | Active | [1] |

**Function**

By setting this parameter to [1] the FC 300 is informed that the negative software end limit should be monitored. Then it is checked whether the target position is located outside of the permissible movement range during every movement. In this case an error message is issued and the drive control is switched off.

In the positioning mode this means that the corresponding positioning process is not started and the error can be cleared with the ERRCLR command.

In synchronizing and speed mode an error can only be recognized after the limit has been exceeded, thus when the error message is issued the drive is already outside of the permissible area of movement. In this case, it is necessary to move the drive by hand back to the admissible area and to erase the error, or to temporarily turn off the corresponding *Software End Limit* and then delete the error.

#### 33-44 Positive Software End Limit Active

SWPOSLIMACT

**Option**

| | | |
|---|---|---|
| * | Inactive | [0] |
| | Active | [1] |

**Function**

By setting this parameter to [1] the FC300 is informed that the *Positive Software End Limit* is to be monitored. In this case it is checked whether the target position is located outside of the permissible movement range during every movement. If necessary an error message is issued and the drive control is switched off.

In the positioning mode this means that the corresponding positioning process is not started and the error can be cleared with the ERRCLR command.

In the synchronizing and speed mode an error can only be recognized after the limit has been exceeded, thus when the error message is issued the drive is already outside of the permissible area of movement. In this case, it is necessary to move the drive, by hand, back to the admissible area, and to erase the error to temporarily turn off the corresponding *Software End Limit* and then delete the error.

## 33-45 Time in Target Window

TESTTIM

**Range [Unit]**

0 – 10 [ms] ∗ 0

**Function**

Once the target window has been reached the position is read twice and compared with the par. 33-46 TESTVAL. If the result is less than TESTVAL, then the position has been reached, otherwise a new reading is taken. TESTTIM indicates the time interval between the measurements.

**NB!:**
The reason for the limitation to 10 ms is to be seen, that the function *diffval* waits really and in this time the monitoring of the limit switch and the position error is not active. For this reason the value should be not too long.

## 33-46 Target Window Limit Value

TESTVAL

**Range [Unit]**

1 – 10000 [qc] ∗ 1

**Function**

Once the target window has been reached the position is read twice with an interval of par. 33-45 TESTTIM and the interval is compared with the *Target Window Limit Value*.

The result determines whether the position is viewed as having been reached or not.

**NB!:**
For longer time intervals it must be taken into consideration that reaching this target position will be delayed by this amount of time in any case.

## 33-47 Size of Target Window

TESTWIN

**Range [Unit]**

0 – 10000 [qc] ∗ 0
0 = Off

TESTWIN must always be less than par. 33-46 TESTVAL.

**Function**

TESTWIN indicates the size of the target window. A position is only viewed as reached when the reference-run (trapeze) is executed, the actual position is within the window and the velocity is less than par. 33-46 *Target Window Limit Value* (Precondition I: TESTWIN and TESTTIM are activated.)

In this content the velocity is given as TESTVAL in qc/TESTTIM.

The controller wait to execute the next command until the actual position is within the target window.

If TESTWIN is not active [0], then the target has been reached if the set position is the target position. However, this does not necessarily correspond with the actual position of the drive.

**NB!:**
If the target window surrounding the end position is selected to be too small, the drive could move in a very small area around the end position without reaching the target window. Thus the program would be 'stuck' after the corresponding positioning command.

A target window of [0] deactivates the monitoring of the actual position and only monitors the command position.

### □ 33-5* I/O Configuration

There is one parameter per input and output. One function assigned to each input and output via this parameter.

**Digital Input Functions**

| | Digital input function | Select | Terminal |
|---|---|---|---|
| * | No function | [0] | X57, X59/7,8* |
| | Home switch 'no' | [1] | X57, X59/7,8* |
| | Home switch 'nc' | [2] | X57, X59/7,8* |
| | Negative end switch 'no' | [3] | X57, X59/7,8* |
| | Negative end switch 'nc' | [4] | X57, X59/7,8* |
| | Positive end switch 'no' | [5] | X57, X59/7,8* |
| | Positive end switch 'nc' | [6] | X57, X59/7,8* |
| | Error clear 'no' | [7] | X57, X59/7,8* |
| | Error clear 'nc' | [8] | X57, X59/7,8* |
| | Break program exe 'no' | [9] | X57, X59/7,8* |
| | Break program exe 'nc' | [10] | X57, X59/7,8* |
| | Continue program exe 'no' | [11] | X57, X59/7,8* |
| | Continue program exe 'nc' | [12] | X57, X59/7,8* |
| | Start program exe 'no' | [13] | X57, X59/7,8* |
| | Start program exe 'nc' | [14] | X57, X59/7,8* |
| | Program select | [15] | X57, X59/7,8* |

X57 = all
*) X59/7,8 only if par. 33-60 IOMODE is set to [0].

You can program all digital inputs 1 – 10 (12) to these functions:

- **No function [0]:** No reaction on signals from the input_n.
- **Home switch no [1]:** Defines digital input_n of MCO 305 as home switch. Behavior after reaching see HOME_TYPE.
- **Home switch nc [2]:** Defines digital input_n as home switch inverse. Behavior after reaching, see HOME_TYPE.
- **Negative end switch no [3]:** Defines digital input_n as the negative end switch.
- **Negative end switch nc [4]:** Defines digital input_n as the negative end switch inverse.
- **Positive end switch no [5]:** Defines digital input_n as the positive limit switch.
- **Positive end switch nc [6]:** Defines digital input_n as the positive limit switch inverse.
- **Error clear no [7]:** Defines digital input_n to be used to clear an error.
- **Error clear nc [8]:** Defines digital input_n to be used to clear an error.
- **Break program exe no [9]:** Defines digital input_n to be used to react and abort a program immediately, when activated. Such a program can be continued with CONTINUE.
- **Break program exe nc [10]:** Defines digital input_n to be used to react and abort a program immediately, when activated. Such a program can be continued with CONTINUE.
- **Continue program exe no [11]:** Defines digital input_n to be used to continue aborted programs.
- **Continue program exe nc [12]:** Defines digital input_n to be used to continue aborted programs.
- **Start program exe no [13]:** Defines digital input_n to be used to define type of program start.
  If the input_n is set to [13], the Autostart program is executed at first and the program waits until input_n will be active. This is then evaluated relevant to the program selection in order to determine the number of the program to be run. If no input for the program start is set, the program with auto identification will be started.
- **Start program exe nc [14]:** Defines digital input_n to be used to define the type of program start. If the input_n is set to [14], the Autostart program is executed at first and the program waits until input_n will be active. This is then evaluated relevant to the program selection in order to determine the number of the program to be run.
  If no input for the program start is set, the program with auto identification will be started.
- **Program select [15]:** Defines digital input_n to be used for a program selection. If input_n is set to [15] then this parameter indicates the input number starting at which the inputs for the *Program Selection* are used. This includes all numbers up to I_FUNCTION_14.

Example

If I_FUNCTION_3_15 and I_FUNCTION_7_13, then upon activation of input 7 the inputs 3, 4, 5, and 6 will be evaluated binary and the result will be used as a program number.

| Input | Level | Binary value |
|---|---|---|
| 3 | low | 0 |
| 4 | high | 2 |
| 5 | high | $2^2$ |
| 6 | low | 0 |
| => program to be started: | | 6 |

Thus it is possible to choose between a maximum of 90 programs, identified with the numerals 0 to 89.

**NB!:**
The values are not automatically reset if a new input is defined. The user has to take care of it. That means if I_FUNCTION_1 is set to [1] (i.e. input 1 is the reference switch) and the user sets I_FUNCTION_3 to [1] (i.e. input 3 is reference switch) then two inputs are defined as reference switches. The software always takes the first one and the second one is ignored.

**33-50 Terminal X57/1 Digital Input**

I_FUNCTION_1

∗ No function [0]

**Function**

Defines function of digital input 1 of MCO 305.

**33-51 Terminal X57/2 Digital Input**

I_FUNCTION_2

∗ No function [0]

**Function**

Defines function of digital input 2 of MCO 305.

**33-52 Terminal X57/3 Digital Input**

I_FUNCTION_3

∗ No function [0]

**Function**

Defines function of digital input 3 of MCO 305.

**33-53 Terminal X57/4 Digital Input**

I_FUNCTION_4

∗ No function [0]

**Function**

Defines function of digital input 4 of MCO 305.

**33-54 Terminal X57/5 Digital Input**

I_FUNCTION_5

∗ No function [0]

**Function**

Defines function of digital input 5 of MCO 305.

**33-55 Terminal X57/6 Digital Input**

I_FUNCTION_6

∗ No function [0]

**Function**

Defines function of digital input 6 of MCO 305.

**33-56 Terminal X57/7 Digital Input**

I_FUNCTION_7

∗ No function [0]

**Function**

Defines function of digital input 7 of MCO 305.

**33-57 Terminal X57/8 Digital Input**

I_FUNCTION_8

∗ No function [0]

**Function**

Defines function of digital input 8 of MCO 305.

**33-58 Terminal X57/9 Digital Input**

I_FUNCTION_9

∗ No function [0]

**Function**

Defines function of digital input 9 of MCO 305.

### 33-59 Terminal X57/10 Digital Input

I_FUNCTION_10

∗ No function [0]

**Function**

Defines function of digital input 10 of MCO 305.

### 33-60 Terminal X59/1 and X59/2 Mode

IOMODE

**Option**

Input [0]
X59/1 = Input 11
X59/2 = Input 12

∗ Output [1]
X59/1 = Output 1
X59/2 = Output 2

**Function**

Two of the terminals (X59/1 and X59/2) can be configured as digital input or digital output.

### 33-61 Terminal X59/1 Digital Input

I_FUNCTION_11

∗ No function [0]

**Function**

Defines function of digital input 11 of MCO 305.

NOTE: This parameter is only visible when p. 33-60 IOMODE = [0], i.e. input_11 is used in standard mode.

### 33-62 Terminal X59/2 Digital Input

I_FUNCTION_12

∗ No function [0]

**Function**

Defines function of digital input 12 of MCO 305.

NOTE: This parameter is only visible when p. 33-60 IOMODE = 0, i.e. input 12 is used in standard mode.

**Digital Output Functions**

| Digital output function | | Select | Terminal |
|---|---|---|---|
| ∗ | No function | [0] | X59 |
| | Moving 'no' | [1] | X59 |
| | Moving 'nc' | [2] | X59 |
| | Error 'no' | [3] | X59 |
| | Error 'nc' | [4] | X59 |
| | Brake control 'no' | [5] | X59 |
| | Brake control 'nc' | [6] | X59 |

NOTE: 8 outputs are only available if par. 33-60 IOMODE is set to [1].

You can program all digital outputs 1 – 8 (6) to these functions:

- **No function [0]:** No reaction on signals from the output_n.
- **Moving no [1]:** Defines digital output_n of the MCO 305 for motion command active. The output is always activated (24 V) as soon as a motion command is active, regardless in which mode (position, velocity or synchronization command). This function is not suitable for monitoring the motor, since the motor could be standing still although the control is in motion.
- **Moving nc [2]:** Defines digital output_n for motion command active. The output is always activated (0 V) as soon as a motion command is active, regardless in which mode (position, velocity or synchronization command). This function is not suitable for monitoring the motor, since the motor could be standing still although the control is in motion.
- **Error no [3]:** Defines output_n for error. The output is set (24 V) when an error has occurred. When the error is cleared this output is re-set.

**NB!:**
The setting of this parameter does not influence the use of the OUT and OUTB commands. With these commands it is also possible to change the outputs which have pre-defined functions.

- **Error nc [4]:** Defines output_n for error. The output is set (0 V) when an error has occurred. When the error is cleared this output is re-set.

**NB!:**
The setting of this parameter does not influence the use of the OUT and OUTB commands. With these commands it is also possible to change the outputs which have pre-defined functions.

- **Brake control no [5]:** Defines digital output_n for brake. If an output is defined for the brake, this remains active even when the program is terminated with ESC.
  The output is activated (24 V) in the case of an abort or option error if par. 33-83 ERRCOND is set to [1] or [3].

**NB!:**
The brake output must always be reset by an OUT command in the program.

Example
```
ON ERROR GOSUB err_handle
// p. 33-66 O4 is set to [6]
SET ERRCOND 1
// Main program loop
SUBPROG err_handle
  WAITI 1
  ERRCLR
  OUT 4 1
RETURN
```

- **Brake control nc [6]:** Defines digital output_n for brake. If an output is defined for the brake, this remains active even when the program is terminated with ESC.
  The output is activated (0 V) in the case of an abort or option error if par. 33-83 ERRCOND is set to [1] or [3].

**NB!:**
The brake output must always be re-set by an OUT command in the program.

### 33-63 Terminal X59/1 Digital Output

O_FUNCTION_1

∗ No function [0]

**Function**

Defines function of digital output 1 of MCO 305.

NOTE: This parameter is only visible when par. 33-60 IOMODE = 1, i.e. output 1 is not used in standard mode.

### 33-64 Terminal X59/2 Digital Output

O_FUNCTION_2

∗ No function [0]

**Function**

Defines function of digital output 2 of MCO 305.

NOTE: This parameter is only visible when par. 33-60 = 1, i.e. output 2 is not used in standard mode.

### 33-65 Terminal X59/3 Digital Output

O_FUNCTION_3

∗ No function [0]

**Function**

Defines function of digital output 3 of MCO 305.

### 33-66 Terminal X59/4 Digital Output

O_FUNCTION_4

∗ No function [0]

**Function**

Defines function of digital output 4 of MCO 305.

### 33-67 Terminal X59/5 Digital Output

O_FUNCTION_5

∗ No function [0]

**Function**

Defines function of digital output 5 of MCO 305.

### 33-68 Terminal X59/6 Digital Output

O_FUNCTION_6

∗ No function [0]

**Function**

Defines function of digital output 6 of MCO 305.

### 33-69 Terminal X59/7 Digital Output

O_FUNCTION_7

∗ No function [0]

**Function**

Defines function of digital output 7 of MCO 305.

### 33-70 Terminal X59/8 Digital Output

O_FUNCTION_8

∗ No function [0]

**Function**

Defines function of digital output 8 of MCO 305.

## □ 33-8* Global Parameters

### 33-80 Activated Program Number

PRGPAR

**Range**

-1 – 127 ∗ -1

–1 = Program number is not activated, i.e. no program to start after autoexec

0 – 127 = activated program number is started after power up (and autoexec)

The power-up state will be defined with p. 33-81.

**Function**

With PRGPAR it is possible to set which program should be started after the conclusion of a program executed via *Autostart* (auto identification). This parameter can also be changed and stored with other programs or via the display.

If no program number is activated and no input for the program start I_FUNCTION_n [13] or [14] is set, then the program with auto identification will be started.

**NB!:**

If no autostart program is defined then it is not possible to start a program via PRGPAR; this always requires a terminated autostart program.

### 33-81 Power-up State

Power-up-State

**Option**

| | | |
|---|---|---|
| | Motor off | [0] |
| ∗ | Motor on | [1] |

**Function**

Controller state after power-up can be defined.

Select *Motor off* [0] if the motor must remain un-controlled (FC 300 is coasted) after power-up. FC 300 and position control must be enabled with the MOTOR ON command before movement can be started.

Select *Motor on* [1] if the motor must be controlled after power-up, positioning controller is active and keeps the actual position until another control command is given.

### 33-82 Drive Status Monitoring

STATUSMONITORING

**Option**

| | | |
|---|---|---|
| | Off | [0] |
| ∗ | On | [1] |

**Function**

Enable/disable monitoring of FC 300 status while position control from MCO 305 is active.

Select *Off* [0] if monitoring must be disabled i.e. MCO 305 will try to control the motor independent of FC 300 status. Will normally lead to a position error (error 108) when trying to start a movement while FC 300 is not enabled.

Select *On* [1] if monitoring must be enabled. Error 113 will be activated if FC 300 is not enabled (e.g. trip) while MCO 305 is in the MOTOR ON state (position control).

### 33-83 Behavior after Error

ERRCOND

**Option**

| | | |
|---|---|---|
| ∗ | Coast | [0] |
| | Coast and brake | [1] |
| | Controlled stop | [2] |
| | Controlled stop and brake | [3] |
| | Jumps to error routine without automatic MOTOR OFF | [5] |

**Function**

0 = Standard, i.e. drive moves in COASTING, control loop is interrupted.

1 = Like [0], but brake output (if defined) is activated.

2 = Motor stop with maximum deceleration (stop ramp), subsequently standstill control.

3 = Like [2], brake output (if defined) is activated in addition, but only after MOTOR STOP. All other activities such as MOTOR OFF etc. must be set in the ON_ERROR routine.

5 = Jumps to the error routine, but the control will not switch off automatically. This can or must be initiated in the application program with a MOTOR OFF in the error routine.

**NB!:**

A brake output has to be defined in parameter 33-63 through 33-70, O_FUNCTION_n Option 5 and 6.

**33-84 Behavior after Esc**

ESCCOND

**Option**

| | | |
|---|---|---|
| * | Controlled stop | [0] |
| | Controlled stop + outputs = 0 | [1] |
| | Controlled stop + outputs = 1 | [2] |

**Function**

ESCCOND defines how the FC300 will react to a program termination using [Esc].

0 = The motor is stopped with maximum deceleration, the brake output is activated (if defined), the master simulation is stopped.
The outputs remain in the current status.

1 = As [0], but all outputs including the FC 300 outputs (if controlled by MCO 305) are set at [0].
Exception: The brake output – if defined – is always activated.

2 = As [0], but all outputs including the FC 300 outputs (if controlled by MCO 305) are set at [1];
Exception: The brake output – if defined – is always activated.

**33-85 MCO Supplied by External 24VDC**

EXTERNAL24V

**Option**

| | | |
|---|---|---|
| * | No | [0] |
| | Yes | [1] |

**Function**

Defines if external 24 V supply is connected or not.

## □ MCO Data Readouts

To support the PCD[] array reading and writing and still be in accordance with the ProfiDrive profile there are the following 20 parameters in the 34-0* and 34-2* group:

### □ 34-0* PCD Write Parameters

**34-01...10 PCD n Write to MCO**

n = 1 – 10
- p. 34-01 = PCD 1 Write to MCO
- p. 34-02 = PCD 2 Write to MCO
- p. 34-03 = PCD 3 Write to MCO
- p. 34-04 = PCD 4 Write to MCO
- p. 34-05 = PCD 5 Write to MCO
- p. 34-06 = PCD 6 Write to MCO
- p. 34-07 = PCD 7 Write to MCO
- p. 34-08 = PCD 8 Write to MCO
- p. 34-09 = PCD 9 Write to MCO
- p. 34-10 = PCD 10 Write to MCO

**Function**

All 10 parameters are selectable as display line parameters in par. 0-20 to 0-24.

But at index [n] only MCO PCD[n] Write can be selected. This selection defines that corresponding sub indices will be consumed by the MCO 305.

This also makes it possible to set the indices 0 and 1 (CTW/STW and REF/MAV) to MCO. This might however be in conflict with the ProfiDrive profile.

### □ 34-2* PCD Read Parameters

**34-21...31 PCD n Read from MCO**

n = 1 – 10
- p. 34-21 = PCD 1 Read from MCO
- p. 34-22 = PCD 2 Read from MCO
- p. 34-23 = PCD 3 Read from MCO
- p. 34-24 = PCD 4 Read from MCO
- p. 34-25 = PCD 5 Read from MCO
- p. 34-26 = PCD 6 Read from MCO
- p. 34-27 = PCD 7 Read from MCO
- p. 34-28 = PCD 8 Read from MCO
- p. 34-29 = PCD 9 Read from MCO
- p. 34-30 = PCD 10 Read from MCO

**Function**

All 10 read parameters are selectable as display line parameters in par. 0-20 to 0-24.

But at index [n] only "MCO PCD[n] Read" can be selected. This selection defines that the corresponding sub indices will be produced by the MCO 305.

## ☐ 34-4* Inputs & Outputs

### 34-40 Digital Inputs

**Function**

Read out status of the digital inputs.

### 34-41 Digital Outputs

**Function**

Read out status of the digital outputs.

## ☐ 34-5* Process Data

In most of the standard cases the 34-xx display parameters which are handled automatically can be used instead of LINKSYSVAR command.

### 34-50 Actual Position

**Function**

Current slave position in UU; corresponds to APOS command.

### 34-51 Commanded Position

**Function**

Set slave position in UU; corresponds to CPOS command.

### 34-52 Actual Master Position

**Function**

Current master position in qc; corresponds to MAPOS command.

### 34-53 Slave Index Position

**Function**

Last slave index position in UU; corresponds to IPOS command.

### 34-54 Master Index Position

**Function**

Last Master index position in qc; corresponds to MIPOS command.

### 34-55 Curve Position

**Function**

Retrieve slave curve position that corresponds to the current master position of the curve; corresponds to CURVEPOS command.

### 34-56 Track Error

**Function**

Queries the actual position error of the axis in UU (in consideration of the signs); corresponds to TRACKERR command.

### 34-57 Synchronizing Error

**Function**

Queries the actual synchronization error of the slave. This is the distance between the actual master position (converted with drive factor and offset) and the actual position of the slave. The result is displayed 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.

The parameter corresponds to SYNCERR command.

### 34-58 Actual Velocity

**Function**

Actual velocity in UU/s; corresponds to AVEL command.

### 34-59 Actual Master Velocity

**Function**

Actual velocity master in qc/s; corresponds to MAVEL command.

### 34-60 Synchronizing Status

**Function**

Flag to query synchronization status. The parameter corresponds to SYNCSTAT command.

### 34-61 Axis Status

**Function**

Displays info on status of program execution. The parameter corresponds to AXEND command.

### 34-62 Program Status

**Function**

Displays axis and control status in 4-Byte values. The parameter corresponds to STAT command.

### □ 34-7* Diagnosis Readouts

Parameters for readout of MCO alarms.

#### 34-70 MCO Alarm Word 1

**Function**

Displays MCO 305 alarm word for readout of MCO errors in MCT 10.

Par. 34-70 cannot be readout while motor is running.

| Bit | Hex | Dec | Meaning |
|---|---|---|---|
| 0 | 00000001 | 1 | FC not enabled |
| 1 | 00000002 | 2 | Error not reset |
| 2 | 00000004 | 4 | HOME not done |
| 3 | 00000008 | 8 | Position error |
| 4 | 00000010 | 16 | Index not found |
| 5 | 00000020 | 32 | Hardware end limit exceeded |
| 6 | 00000040 | 64 | Software end limit exceeded |
| 7 | 00000080 | 128 | No external 24 V |
| 8 | 00000100 | 256 | Digial output overload |
| 9 | 00000200 | 512 | Encoder error |
| 10 | 00000400 | 1024 | Memory error |
| 11 | 00000800 | 2048 | Parameter memory corrupted |
| 12 | 00001000 | 4096 | Program memory corrupted |
| 13 | 00002000 | 8192 | Reset by CPU |
| 14 | 00004000 | 16384 | WAITNDX timeout |
| 15 | 00008000 | 32768 | Internal MCO fault |
| 16 | 00010000 | 65536 | Home vel zero |
| .. | .. | .. | not used |
| 31 | 80000000 | 2147483648 | MCO alarm word 2 |

#### 34-71 MCO Alarm Word 2

**Function**

Displays MCO 305 alarm word for readout of MCO errors in MCT 10.

Par. 34-71 cannot be readout while motor is running.

| Bit | Hex | Dec | Meaning |
|---|---|---|---|
| 0 | 00000001 | 1 | Illegal axis number |
| 1 | 00000002 | 2 | Unknown command |
| 2 | 00000004 | 4 | Unknown parameter |
| 3 | 00000008 | 8 | Too many loops |
| 4 | 00000010 | 16 | Too many interrupts |
| 5 | 00000020 | 32 | Too many GOSUB |
| 6 | 00000040 | 64 | Too many RETURN |
| 7 | 00000080 | 128 | Use abort |
| 8 | 00000100 | 256 | LINK failed |
| 9 | 00000200 | 512 | Wrong array size (DIM) |
| 10 | 00000400 | 1024 | Array too small |
| 11 | 00000800 | 2048 | Too many time interrupts |
| 12 | 00001000 | 4096 | Out of memory |
| 13 | 00002000 | 8192 | Memory locked |
| 14 | 00004000 | 16384 | Illegal cam array |
| 15 | 00008000 | 32768 | Parameter save failed |

.

## □ Parameter Lists

The parameters are determined by parameter numbers. We recommend using the alphabetical overview as a guide; then you will be able to find detailed information very quickly using the number.

Changes during operation

"TRUE" means that the parameter can be changed, while the frequency converter is in operation.

"FALSE" means that the frequency converter must be stopped before a change can be made.

4-Set-up

"1-Set-up": Data value will be the same in all set-ups.

Conversion index

This number refers to a conversion figure used when writing or reading by means of a frequency converter.

Please see for all conversion indices the FC 300 Design Guide.

| **Conversion index** | 0 |
|---|---|
| **Conversion factor** | 1 |

Data type

Please see for all data types the FC 300 Design Guide.

| Data type | Description | Type |
|---|---|---|
| 2 | Integer 8 | Int8 |
| 3 | Integer 16 | Int16 |
| 4 | Integer 32 | Int32 |
| 5 | Unsigned 8 | Uint8 |
| 6 | Unsigned 16 | Uint16 |
| 7 | Unsigned 32 | Uint32 |

### □ Application Parameters, Parameter List

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conversion index | Type |
|---|---|---|---|---|---|---|---|
| **19-0*** | **Application Parameters** | | | | | | |
| 19-00 | | Application parameters | 0 | 'TRUE' | | | |
| … | | | | TRUE | | | |
| 19-89 | | Application parameters | 0 | TRUE | | | |
| **19-9*** | **Read-only Application Parameters** | | | | | | |
| 19-90 | | Application parameter 90 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-91 | | Application parameter 91 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-92 | | Application parameter 92 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-93 | | Application parameter 93 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-94 | | Application parameter 94 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-95 | | Application parameter 95 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-96 | | Application parameter 96 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-97 | | Application parameter 97 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-98 | | Application parameter 98 | 0 | read only | 1 set-up | 0 | Int32 |
| 19-99 | | Application parameter 99 | 0 | read only | 1 set-up | 0 | Int32 |

**∗ default setting [ ] value for use in communication via serial communication port**

### □ MCO Basics Settings, Parameter List

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conver-sion index | Type |
|---|---|---|---|---|---|---|---|
| **32-0*** | **Encoder 2 - Slave** | | | | | | |
| 32-00 | ENCODERTYPE | Incremental Signal Type | [1] RS422 | 'TRUE' | 1 set-up | | Uint8 |
| 32-01 | ENCODER | Incremental Resolution | 1024 PPR | TRUE | 1 set-up | | Uint32 |
| 32-02 | ENCODER ABSTYPE | Absolute Protocol | [0] None | TRUE | 1 set-up | | Uint8 |
| 32-03 | ENCODER ABSRES | Absolute Resolution | 8192 PPR | TRUE | 1 set-up | | Uint32 |
| 32-05 | ENCODER ABSTYPE | Absolute Encoder Data Length | 25 Bit | TRUE | 1 set-up | | Uint8 |
| 32-06 | ENCODERFREQ | Absolute Encoder Clock Frequency | 262.000 kHz | TRUE | 1 set-up | | Uint32 |
| 32-07 | ENCODER CLOCK | Absolute Encoder Clock Generation | [1] On | TRUE | 1 set-up | | Uint8 |
| 32-08 | ENCODER DELAY | Absolute Encoder Cable Length | 0 | TRUE | 1 set-up | | Uint16 |
| 32-09 | ENCODER MONITORING | Encoder Monitoring | [0] Off | TRUE | 1 set-up | | Uint8 |
| 32-10 | POSDRCT | Rotational Direction | [1] No action | TRUE | 1 set-up | | Uint8 |
| 32-11 | POSFACT_N | User Unit Denominator | 1 | TRUE | 1 set-up | | Uint32 |
| 32-12 | POSFACT_Z | User Unit Numerator | 1 | TRUE | 1 set-up | | Uint32 |
| 32-3* | **Encoder 1 - Master** | | | | | | |
| 32-30 | MENCODER TYPE | Incremental Signal Type | [1] RS422 | TRUE | 1 set-up | | Uint8 |
| 32-31 | MENCODER | Incremental Resolution | 1024 PPR | TRUE | 1 set-up | | Uint32 |
| 32-32 | MENCODER ABSTYPE | Absolute Protocol | [0] None | TRUE | 1 set-up | | Uint8 |
| 32-33 | MENCODER ABSRES | Absolute Resolution | 8192 PPR | TRUE | 1 set-up | | Uint32 |
| 32-35 | MENCODER DATLEN | Absolute Encoder Data Length | 25 Bit | TRUE | 1 set-up | | Uint8 |
| 32-36 | MENCODER FREQ | Absolute Encoder Clock Frequency | 262.000 kHz | TRUE | 1 set-up | | Uint32 |
| 32-37 | MENCODER CLOCK | Absolute Encoder Clock Generation | [1] On | TRUE | 1 set-up | | Uint8 |
| 32-38 | MENCODER DELAY | Absolute Encoder Cable Length | 0 | TRUE | 1 set-up | | Uint16 |
| 32-39 | MENCODER MONITORING | Encoder Monitoring | [0] Off | TRUE | 1 set-up | | Uint8 |
| 32-40 | MENCODER TERM | Encoder Termination | [1] On | TRUE | 1 set-up | | Uint8 |
| **32-5*** | **Feedback Source** | | | | | | |
| 32-50 | Source Slave | | [2] Enc2 | TRUE | 1-set-up | | |
| **32-6*** | **PID-Controller** | | | | | | |
| 32-60 | KPROP | Proportional Factor | 30 | TRUE | 1 set-up | 0 | Uint32 |
| 32-61 | KDER | Derivative Value for PID Control | 0 | TRUE | 1 set-up | 0 | Uint32 |

**∗ default setting  [ ] value for use in communication via serial communication port**

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conver-sion index | Type |
|---|---|---|---|---|---|---|---|
| 32-62 | KINT | Integral Factor | 0 | TRUE | 1 set-up | 0 | Uint32 |
| 32-63 | KILIM | Limit Value for Integral Sum | 1000 | TRUE | 1 set-up | 0 | Uint16 |
| 32-64 | BANDWIDTH | PID Bandwidth | 1000 | TRUE | 1 set-up | 0 | Uint16 |
| 32-65 | FFVEL | Velocity Feed-forward | 0 | TRUE | 1 set-up | 0 | Uint32 |
| 32-66 | FFACC | Acceleration Feed-forward | 0 % | TRUE | 1 set-up | 0 | Uint32 |
| 32-67 | POSERR | Maximum Tolerated Position Error | 20000 qc | TRUE | 1 set-up | | Uint32 |
| 32-68 | REVERS | Reverse Behavior for Slave | [0] Reversing | TRUE | 1 set-up | | Uint8 |
| 32-69 | TIMER | Sampling Time for PID control | 1 ms | TRUE | 1 set-up | | Uint16 |
| 32-70 | PROFTIME | Scan Time for Profile Generator | [1] 1 ms | TRUE | 1 set-up | | Uint8 |
| 32-71 | REGWINMAX | Size of the Control Window (Activation) | 0 qc | TRUE | 1 set-up | | Uint32 |
| 32-72 | REGWINMIN | Size of the Control Window (Deactivation) | 0 qc | TRUE | 1 set-up | | Uint32 |
| **32-8*** | **Velocity & Acceleration** | | | | | | |
| 32-80 | VELMAX | Maximum Velocity (Encoder) | 1500 RPM | TRUE | 1 set-up | | Uint32 |
| 32-81 | RAMPMIN | Shortest Ramp | 1 s | TRUE | 1 set-up | | Uint32 |
| 32-82 | RAMPTYPE | Ramp Type | 0 | TRUE | 1 set-up | | Uint8 |
| 32-83 | VELRES | Velocity Resolution | 100 | TRUE | 1 set-up | | Uint16 |
| 32-84 | DFLTVEL | Default Velocity | 50 | TRUE | 1 set-up | | Uint16 |
| 32-85 | DFLTACC | Default Acceleration | 50 | TRUE | 1 set-up | | Uint16 |

### □ MCO Advanced Settings, Parameter List

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conver-sion index | Type |
|---|---|---|---|---|---|---|---|
| **33-0*** | **Home Motion** | | | | | | |
| 33-00 | HOME_FORCE | Force HOME | [0] not forced | 'TRUE' | 1 set-up | | Uint8 |
| 33-01 | HOME_OFFSET | Zero Point Offset from Home Position | 0 qc | TRUE | 1 set-up | | Int32 |
| 33-02 | HOME_RAMP | Ramp for Home Motion | 10 | TRUE | 1 set-up | | Uint16 |
| 33-03 | HOME_VEL | Velocity of Home Motion | 10 | TRUE | 1 set-up | | Int16 |
| 33-04 | HOME_TYPE | Behavior during Home Motion | [0] Reverse + Index | TRUE | 1 set-up | | Uint8 |
| **33-1*** | **Synchronization** | | | | | | |
| 33-10 | SYNCFACTM | Synchronization Factor Master (M:S) | 1 | TRUE | 1 set-up | | Int32 |
| 33-11 | SYNCFACTS | Synchronization Factor Slave (M:S) | 1 | TRUE | 1 set-up | | Int32 |
| 33-12 | SYNCPOSOFFS | Position Offset for Synchronization | 0 qc | TRUE | 1 set-up | | Int32 |
| 33-13 | SYNC ACCURACY | Accuracy Window for Position Synchronization | 1000 qc | TRUE | 1 set-up | | Int32 |
| 33-14 | SYNCVELREL | Relative Slave Velocity Limit | 0 % | TRUE | 1 set-up | | Uint8 |
| 33-15 | SYNCMARKM | Marker Number for Master | 1 | TRUE | 1 set-up | | Uint16 |
| 33-16 | SYNCMARKS | Marker Number for Slave | 1 | TRUE | 1 set-up | | Uint16 |
| 33-17 | SYNCMPULSM | Master Marker Distance | 4096 | TRUE | 1 set-up | | Uint32 |
| 33-18 | SYNCMPULSS | Slave Marker Distance | 4096 | TRUE | 1 set-up | | Uint32 |
| 33-19 | SYNCMTYPM | Master Marker Type | [0] Enc. Z pos. | TRUE | 1 set-up | | Uint8 |
| 33-20 | SYNCMTYPS | Slave Marker Type | [0] Enc. Z pos. | TRUE | 1 set-up | | Uint8 |
| 33-21 | SYNCMWINM | Master Marker Tolerance Window | 0 | TRUE | 1 set-up | | Uint32 |
| 33-22 | SYNCMWINS | Slave Marker Tolerance Window | 0 | TRUE | 1 set-up | | Uint32 |
| 33-23 | SYNCMSTART | Start Behavior for Synchronization | [0] Start Funct. 1 | TRUE | 1 set-up | | Uint16 |
| 33-24 | SYNCFAULT | Marker Number for Fault | 10 | TRUE | 1 set-up | | Uint16 |
| 33-25 | SYNCREADY | Marker Number for Ready | 1 | TRUE | 1 set-up | | Uint16 |
| 33-26 | SYNCVFTIME | Velocity Filter | 0 µs | TRUE | 1 set-up | 0 | Int32 |
| 33-27 | SYNCOFFTIME | Offset Filter Time | 0 ms | TRUE | 1 set-up | | Uint32 |
| 33-28 | SYNCMFPAR | Marker Filter Configuration | [0] Marker Filter 1 | TRUE | 1 set-up | | Uint8 |
| 33-29 | SYNCMFTIME | Filter Time for Marker Correction | 0 ms | TRUE | 1 set-up | | Int32 |
| 33-30 | SYNCM MAXCORR | Maximum Marker Correction | [0] Off | TRUE | 1 set-up | | Uint32 |
| 33-31 | SYNCTYPE | Synchronization Type | [0] Standard | TRUE | 1 set-up | | Uint8 |

**∗ default setting  [ ] value for use in communication via serial communication port**

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conver-sion index | Type |
|---|---|---|---|---|---|---|---|
| **33-4*** | **Limit Handling** | | | | | | |
| 33-40 | ENDSWMOD | Behavior at End Limit Switch | [0] Call error handler | TRUE | 1 set-up | | Uint8 |
| 33-41 | NEGLIMIT | Negative Software End Limit | -500000 qc | TRUE | 1 set-up | | Int32 |
| 33-42 | POSLIMIT | Positive Software End Limit | 500000 qc | TRUE | 1 set-up | | Int32 |
| 33-43 | SWNEGLIMACT | Negative Software End Limit Active | [0] Inactive | TRUE | 1 set-up | | Uint8 |
| 33-44 | SWPOSLIMACT | Positive Software End Limit Active | [0] Inactive | TRUE | 1 set-up | | Uint8 |
| 33-45 | TESTTIM | Time in Target Window | 0 ms | TRUE | 1 set-up | | Uint8 |
| 33-46 | TESTVAL | Target Window Limit Value | 1 qc | TRUE | 1 set-up | | Uint16 |
| 33-47 | TESTWIN | Size of Target Window | 0 qc | TRUE | 1 set-up | | Uint16 |
| **33-5*** | **I/O Configuration** | | | | | | |
| 33-50 | I_FUNCTION_1 | Terminal X57/1 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-51 | I_FUNCTION_2 | Terminal X57/2 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-52 | I_FUNCTION_3 | Terminal X57/3 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-53 | I_FUNCTION_4 | Terminal X57/4 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-54 | I_FUNCTION_5 | Terminal X57/5 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-55 | I_FUNCTION_6 | Terminal X57/6 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-56 | I_FUNCTION_7 | Terminal X57/7 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-57 | I_FUNCTION_8 | Terminal X57/8 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-58 | I_FUNCTION_9 | Terminal X57/9 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-59 | I_FUNCTION_10 | Terminal X57/10 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-60 | IOMODE | Terminal X59/1 and X59/2 Mode | [0] Output | 'FALSE' | 1 set-up | | Uint8 |
| 33-61 | I_FUNCTION_11 | Terminal X57/11 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-62 | I_FUNCTION_12 | Terminal X57/12 Digital Input | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-63 | O_FUNCTION_1 | Terminal X59/1 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-64 | O_FUNCTION_2 | Terminal X59/2 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-65 | O_FUNCTION_3 | Terminal X59/3 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |

**∗ default setting** **[ ] value for use in communication via serial communication port**

| Par. No. # | Parameter name | Parameter description | Default setting | Changes during operation | 4-set-up | Conver-sion index | Type |
|---|---|---|---|---|---|---|---|
| 33-66 | O_FUNCTION_4 | Terminal X59/4 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-67 | O_FUNCTION_5 | Terminal X59/5 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-68 | O_FUNCTION_6 | Terminal X59/6 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-69 | O_FUNCTION_7 | Terminal X59/7 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| 33-70 | O_FUNCTION_8 | Terminal X59/8 Digital Output | [0] no function | TRUE | 1 set-up | | Uint8 |
| **33-8*** | **Global Parameters** | | | | | | |
| 33-80 | PRGPAR | Activated Program Number | -1 | TRUE | 1 set-up | 0 | Uint8 |
| 33-81 | Power-up State | Power-up State | [1] Motor on | TRUE | 1 set-up | | Uint8 |
| 33-82 | STATUS MONITORING | Drive Status Monitoring | [1] On | TRUE | 1 set-up | 0 | Uint8 |
| 33-83 | ERRCOND | Behavior after Error | [0] Coast | TRUE | 1 set-up | | Uint8 |
| 33-84 | ESCCOND | Behavior after Escape | [0] Contr. Stop | TRUE | 1 set-up | | Uint8 |
| 33-85 | EXTERNAL24V | MCO Supplied by External 24VDC | [0] No | TRUE | 1 set-up | | Uint8 |

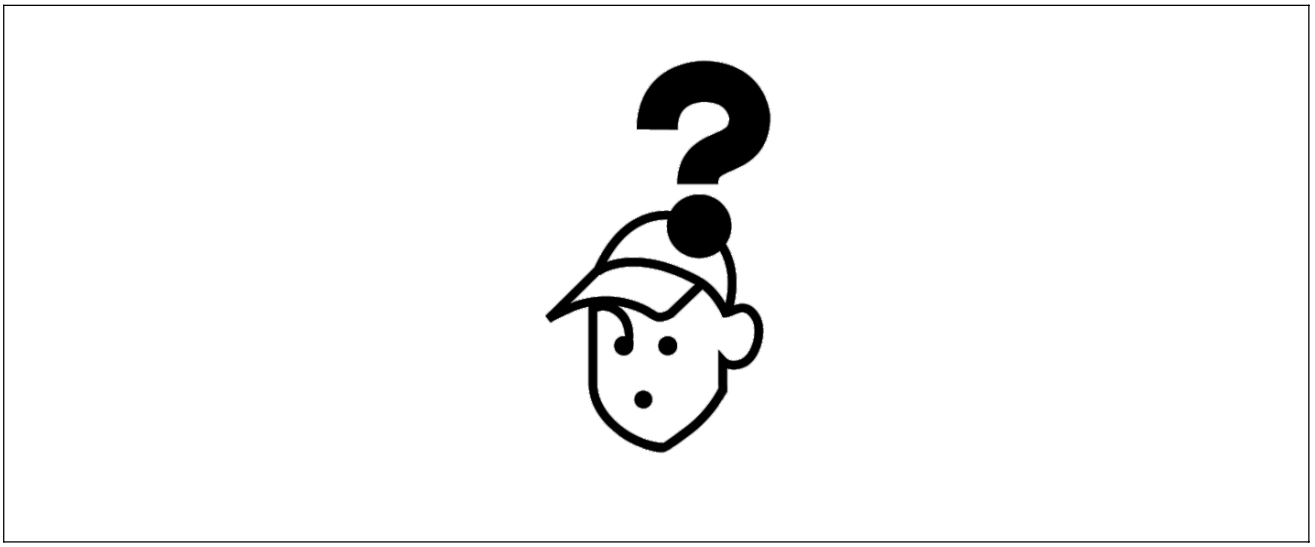
### □ MCO Data Readouts, Parameter List

| Par. No. # | Parameter Name | Parameter description | Default setting | Changes during operation | 4-Setup | Conver-sion Index | Type |
|---|---|---|---|---|---|---|---|
| **34-0*** | **PCD Write Parameters** | | | | | | |
| 34-01 | | PCD 1 Write to MCO | | | 1 set-up | | Uint16 |
| 34-02 | | PCD 2 Write to MCO | | | 1 set-up | | Uint16 |
| 34-03 | | PCD 3 Write to MCO | | | 1 set-up | | Uint16 |
| 34-04 | | PCD 4 Write to MCO | | | 1 set-up | | Uint16 |
| 34-05 | | PCD 5 Write to MCO | | | 1 set-up | | Uint16 |
| 34-06 | | PCD 6 Write to MCO | | | 1 set-up | | Uint16 |
| 34-07 | | PCD 7 Write to MCO | | | 1 set-up | | Uint16 |
| 34-08 | | PCD 8 Write to MCO | | | 1 set-up | | Uint16 |
| 34-09 | | PCD 9 Write to MCO | | | 1 set-up | | Uint16 |
| 34-10 | | PCD 10 Write to MCO | | | 1 set-up | | Uint16 |
| **34-2*** | **PCD Read Parameters** | | | | | | |
| 34-21 | | PCD 1 Read from MCO | | | 1 set-up | | Uint16 |
| 34-22 | | PCD 2 Read from MCO | | | 1 set-up | | Uint16 |
| 34-23 | | PCD 3 Read from MCO | | | 1 set-up | | Uint16 |
| 34-24 | | PCD 4 Read from MCO | | | 1 set-up | | Uint16 |

**∗ default setting** **[ ] value for use in communication via serial communication port**

| Par. No. # | Parameter Name | Parameter description | Default setting | Changes during operation | 4-Setup | Conver-sion Index | Type |
|---|---|---|---|---|---|---|---|
| 34-26 | | PCD 6 Read from MCO | | | 1 set-up | | Uint16 |
| 34-27 | | PCD 7 Read from MCO | | | 1 set-up | | Uint16 |
| 34-28 | | PCD 8 Read from MCO | | | 1 set-up | | Uint16 |
| 34-29 | | PCD 9 Read from MCO | | | 1 set-up | | Uint16 |
| 34-30 | | PCD 10 Read from MCO | | | 1 set-up | | Uint16 |
| **34-4*** | **Inputs & Outputs** | | | | | | |
| 34-40 | | Digital Inputs | | | 1 set-up | | Uint16 |
| 34-41 | | Digital Outputs | | | 1 set-up | | Uint16 |
| **34-5*** | **Process Data** | | **Unit** | | | | |
| 34-50 | | Actual Position | UU | | 1 set-up | | Int32 |
| 34-51 | | Commanded Position | UU | | 1 set-up | | Int32 |
| 34-52 | | Actual Master Position | qc | | 1 set-up | | Int32 |
| 34-53 | | Slave Index Position | UU | | 1 set-up | | Int32 |
| 34-54 | | Master Index Position | qc | | 1 set-up | | Int32 |
| 34-55 | | Curve Position | | | 1 set-up | | Int32 |
| 34-56 | | Track Error | UU | | 1 set-up | | Int32 |
| 34-57 | | Synchronizing Error | UU | | 1 set-up | | Int32 |
| 34-58 | | Actual Velocity | UU/s | | 1 set-up | | Int32 |
| 34-59 | | Actual Master Velocity | qc/s | | 1 set-up | | Int32 |
| 34-60 | | Synchronizing Status | | | 1 set-up | | Int32 |
| 34-61 | | Axis Status | | | 1 set-up | | Int32 |
| 34-62 | | Program Status | | | 1 set-up | | Int32 |
| **34-7*** | **Diagnosis Readouts** | | | | | | |
| 34-70 | | MCO Alarm Word 1 | | 'FALSE' | 1 set-up | | Uint32 |
| 34-71 | | MCO Alarm Word 2 | | FALSE | 1 set-up | | Uint32 |

# Troubleshooting



## Warnings and Error Messages

All messages are shown in the LCP display of the FC 300 in short and in the APOSS software in plain text.

You can find brief information on the error messages in the table or detailed information in the following section.

The tables contain the messages in numerical order. Letters following a % sign represent variables which can be used in plain text at the corresponding locations.

| Error no. | Error text | Description |
|---|---|---|
| 103 | Illegal axis num. | Axes not in system. |
| 105 | Error not reset | Error not cleared. |
| 106 | Home not done | Failed to move to HOME position. |
| 107 | Home vel. zero | Home was executed with Home Velocity set to zero. |
| 108 | Position error | Position error. |
| 109 | Index not found | Index pulse (encoder) not found. |
| 110 | Unknown cmm. | Unknown command. |
| 111 | SW end limit | Software end limit activated. |
| 112 | Unknown param. | Illegal parameter number. |
| 113 | FC not enabled | FC 300 is not ready but the PID controller is active. |
| 114 | Too many loops | Too many nested loops. |
| 115 | Par. save failed | Parameters save failed. |
| 116 | Param. memory | Parameters in memory are corrupted. |
| 117 | Progr. Memory | Programs in memory are corrupted. |
| 118 | Reset by CPU | Reset by CPU. |
| 119 | User abort | User abort. |
| 125 | HW end limit | HW end limit activated. |

| Error no. | Error text | Description |
|---|---|---|
| 149 | Too many inter. | Too many interrupt functions. |
| 150 | No ext. 24V | External supply is missing. |
| 151 | Too many gosub | Too many nested GOSUB commands |
| 152 | Too many return | Too many RETURN commands. |
| 154 | D. out overload | Digital Output overloaded. |
| 155 | LINK failed | LINKGPAR command failed. |
| 162 | Memory error | Error in verifying; EEPROM: address % defect. |
| 170 | Array size (DIM) | Error in DIM command. |
| 171 | Array too small | Attempt was made to cross array bounds. |
| 179 | Waitndx timeout | Timeout while waiting for index. |
| 184 | Too many ontime | Too many time interrupts. |
| 187 | Out of memory | No more room for variables. |
| 190 | Memory locked | The program memory is write-protected. |
| 191 | Illegal cam array | Curve array wrong. |
| 192 | Encoder error | Encoder error |
| 199 | Internal MCO fault | Internal MCO fault |

**Error 103**

**Illegal axis num.**

An attempt has been made to find an axis which does not exist in the controller.

Check to see if the program axis command has an invalid number or a general axis command (...X(*)).

**Error 105**

**Error not reset**

An attempt has been made to execute a motion command, although a momentary error message has not been cleared.

**Error 106**

**Home not done**

Failed to move to HOME position. According to the axis par. 33-00 *Force HOME*, a forced move to the machine zero-point is demanded, before other motion commands can be executed. This move to the machine zero-point has not been executed.

**Error 107**

**Home vel. zero**

An attempt was made to execute the HOME command but the motor is set to 0 in par. 33-03 *Velocity of Home Motion*.

**Error 108**

**Position error**

The distance between the set and the real position was greater than the *Maximum Tolerated Position Error* defined in par. 32-67.

Causes:

- Mechanically blocked or overloaded drive,
- par. 32-67 *Max. Tolerated Position Error* too small,
- commanded speed greater than FC 300 parameters 4-13 *Motor Speed High Limit* and 3-03 *Maximum Reference*,
- commanded acceleration too great,
- par. 32-60 *Proportional Factor* too small, or
- FC 300 not enabled.

**Error 109**

**Index not found**

At reference or index search, the encoder index pulse could not be found within a motor rotation.

Causes

- An encoder without an index pulse has been used,
- index pulse not connected,
- index pulse incorrect (all three channels must have a simultaneous low), or
- the par. 32-01 *Incremental Resolution* (ENCODER) is set too low.

**Error 110**

**Unknown comm.**

Cause: A communication or program error. The program must be re-compiled and re-loaded.

**Error 111**

**SW end limit**

A motion command will cause or has caused the software end limit to be activated.

Identification of attainment of software limit at a motion in the speed mode will only be made after the current position is identical to the software limit switch.

The control unit will be switched off and the drive must be manually moved back to within the admissible area, or the monitoring of the software limit switch must be temporarily de-activated via the *Negative and Positive Software End Limit* in parameters 33-43 and 33-44. Only then is it possible to clear the error.
In positioning mode, it will be known before motion start that the target position lies outside the path. In this case, the movement will not be executed and the error message can be cleared.

**Error 112**

**Unknown param.**

An attempt has been made to change a parameter (SET or SETVLT command), which does not exist.

**Error 113**

**FC not enabled**

FC 300 is not ready but the PID controller is active. The FC status word (Bit 09 and Bit 11) is monitored every 20 ms when the PID controller is active. The FC 300 is in the "Not ready" state when:

- it has an alarm,
- it is in local mode,
- local LCP stop is activated.

**Error 114**

**Too many loops**

Too many nested loops exist in the executed program.

**Error 115**

**Par. Save failed**

Saving of the option parameter failed.

**Error 116**

**Param. memory**

The parameters in EEPROM are no longer correct because of

- EEPROM defective or
- power outage while saving.

**NB!:**
You have to re-initialize the parameter with a 14-22 *Reset* and then overwrite these parameters again with your own application parameters.

Otherwise motion programs which require application parameters will no longer function correctly.

**Error 117**

**Progr. Memory**

The program data stored in EEPROM cannot be found or are no longer correct because of

- EEPROM defective or
- power outage while saving.

You have to do a 3-finger reset to reset all parameters to their defaults (ex factory) and to delete all user programs, arrays, and application parameters.

Afterwards re-load the programs and parameters.

This corresponds to a → *Reset complete* in the APOSS menu.

**Error 118**

**Reset by CPU**

The processor has been stopped and a re-set has automatically been executed (watchdog).

Causes could be

- Short term voltage drop,
- voltage peak, or
- short circuit.

**Error 119**

**User abort**

The *Autostart* program has been aborted by the user.

Or the [CANCEL] key was pressed during switching on and a Master Reset triggered.

**Error 125**

**HW end limit**

A motion command has caused an axis limit switch to be activated.

Through activation of an end limit switch, the controller (depending on the par. 33-40 *Behavior at End Limit Switch*) is automatically switched off and the drive must be manually moved out of this position, before the error message can be cleared.

**Error 149**

**Too many inter.**

More interrupt functions than the maximum possible number were used. Permitted are:

32 ON INT
32 ON STATBIT
32 ON COMBIT
10 ON PARAM
20 ON APOS, ON MAPOS, ON MCPOS

**Error 150**

**No ext. 24V**

External supply is missing.

**Error 151**

**Too many gosub**

In the program exists too many calls from one subroutine to another subroutine.

The error usually occurs when there is a recurrent reference to one of the sub-programs in a sub-program.

Avoid too many (10 is maximum) opposing subroutine calls, and avoid subroutines which call themselves (re-cursive subroutine procedures).

**Error 152**

**Too many return**

There are either more RETURN than corresponding GOSUB commands in the program, or there is a direct jump from a subroutine with a GOTO command.

Only one RETURN is allowed per sub-program.

It is always better to jump to the beginning of a sub-program and then to jump with IF… to a previously defined label.

**Error 154**

**D. out overload**

Digital output overloaded.

**Error 155**

**LINK failed**

LINKGPAR command failed.

**Error 162**

**Memory error**

After saving something in the EEPROM (a program or parameters) an error was detected during verification.

Delete the EEPROM with a 3-finger reset and try to save the program or parameters again.

If this is not successful please call the technical service department.

**Error 170**

**Array size (DIM)**

The definition of an array in a DIM command does not correspond to an already existing array in the MCO 305.

Cause might be that the fields are from older SYNCPOS/APOSS programs. The current program has other definitions.

Either adapt the APOSS program to the correct array size or delete the old arrays, e.g. in Stand-alone Mode with *Controller* → *Memory* → *Delete EEPROM* or use the command *Controller* → *Reset* → Arrys.

**NB!:**
Remember to follow the recommendations concerning saving programs and parameters before deleting the EEPROM.

**Error 171**

**Array too small**

An attempt was made to describe an array element that is located outside of the defined array limits.

Cause might be an error in the APOSS program. Array sizing does not agree with the space required (e.g. due to an incorrectly programmed loop).

Or the array is too small for the number of test drives triggered by TESTSTART.

Check loop variables.

**Error 179**

**Waitndx timeout**

The command WAITNDX was executed and the timeout listed was exceeded.

The timeout is probably too short or the index impulse could not found (see also Error 109).

**Error 184**

**Too many ontime**

Too many interrupts (ON TIME or ON PERIOD commands) were used within the program.

A maximum of 12 of these ON TIME and/or ON PERIOD commands are allowed within one program.

**Error 187**

**Out of memory**

No more space for variables.

When the APOSS program is started the space for the necessary variables is reserved dynamically. This space is now no longer available.

You may have selected a maximum number of variables which is too high. Reduce the maximum number in *Settings* → *Compiler* (Standard = 92).

Or the memory available is occupied with programs or arrays. Delete the programs or delete both the programs and arrays, i.e. by deleting the entire memory.

**NB!:**
Remember to follow the recommendations concerning saving programs and parameters before deleting the EEPROM.

**Error 190**

**Memory locked**

The program memory is write-protected and cannot be altered.

This means that auto recognition can neither be set nor deleted and programs can neither be saved nor deleted. Equally, → *RAM save* and → *EEPROM delete* will not be executed.

**Error 191**

**Illegal cam array**

An incorrect or old array is defined in the DIM instruction for SETCURVE.

An old array may exist if the CNF file with all parameters and arrays has not been loaded into the *CAM-Editor*.

An incorrect array could be caused by the following:

- It was not created by the curve editor.
- Previous version of a curve editor. Such an array must first be converted by the current CAM-Editor (→ *load* and *save*).
- Or the order of the arrays in the DIM instruction does not match the order in the cnf file. Refer to the number of the array in the title bar of the *CAM-Editor* in this respect.

**Error 192**

**Encoder error**

Error from encoder monitoring: open or short circuit in accordance with the displayed LED.

**NB!:**
An error will be indicated even if no encoder is connected.

**Error 199**

**Internal MCO fault**

If such an error should occur, please contact your dealer and report the error number displayed to the technical service department.

## □ APOSS Software Messages

The APOSS software messages are arranged in alphabetical order. Letters following a % sign represent variables which can be used in plain text at the corresponding locations.

| Error text |
|---|
| Compilation error(s): program not saved! |
| Connection to %d already exists [%s] - change to new Window |
| Connector %d pin %d is not valid in line %d column %d |
| Controller is executing a program or command! |
| Error in array part of file. |
| Error in axis parameter part of file. |
| Error in global parameter part of file. |
| Lost connection to #%d! |
| Timeout: no reply from FC |

### Compilation error(s): program not saved!

A file is always compiled first and then saved. If you want to save the program, for example in the menu *Controller* → *Save program* and a syntax error is found during compilation this message will be displayed.

Start the → *Syntax Check* in the menu *Development*, correct the syntax error and then save the program.

### Connection to ... already exists ...

### Connection to %d already exists [%s] - change to Window?

When opening a new window, or when trying to connect a window with a controller that is already linked to a window.

Yes: The controller is disconnected from the old window and linked to the new window.

No: The controller stays connected to the old window. The new window is not linked to a controller.

### Connector pin is not valid

### Connector %d pin %d is not valid in line %d column %d

An illegal combination or a pin number which cannot be set is used with the OUT command.

### Controller is executing a program or command!

When the controller is executing a command or program it is not available for additional commands. You have to *Break* the new command and re-start it once the previous command has been completely executed.

### Error in array part of file

When re-saving a configuration (e.g. with *Controller* → *Parameters* → *Restore from file*) the computer recognizes that the data in the array area is formatted incorrectly.

In order to be able to save a file, the following conditions must be fulfilled:

- Identical software versions,
- same configuration (e.g. same number of axes),
- in the case that arrays have already been inputted, these must match the ones that are to be saved in terms of type and size.

### Error in axis parameter part of file

When re-saving a configuration (e.g. with → *Restore from file*) the computer recognizes that the data in the area of the axis parameters is formatted incorrectly. The parameter number and the sequence be correct and numbering must be continuous.

In order to be able to save a file, the following conditions must be fulfilled:

- Identical software versions, that provides same number and order of the parameters,
- same configuration (e.g. same number of axes).

### Error in global parameter part of file.

When re-saving a configuration (e.g. with → *Restore from file*) the computer recognizes that the data in the area of the global parameters is formatted incorrectly. In order to be able to save a file, the following conditions must be fulfilled:

- Identical software versions, that provides same number and order of the parameters,
- same configuration (e.g. same number of axes).

### Lost connection to ...

If the FC 300 is turned off or the plug is pulled, etc. the window is disconnected from the FC 300 and the lost connection is registered.

### Timeout: no reply from FC

The FC 300 does not answer; check the connection.

# Appendix


## Get a General Idea of Program Samples

Some samples are delivered with the software. Double click on the file name opens the sample in APOSS software.

Some samples are described in the manual "Operation Instructions" or in the chapter "Functions and Samples" in this Design Guide. Please use the online help for more details.

All samples can be copied and pasted into the APOSS software. Please read the safety instructions before using the samples!

| File name or topic | Where to find? | | Description |
|---|---|---|---|
| | Online help | Manual or Software | |
| Absolute Positioning | x | Design Guide page 20 | Absolute Positioning for Palletizer Application. |
| ACC_01.M | x | | Comparison of some position controlled moves with different acceleration (but the same velocity). |
| APOS_01.M | x | | Displays the current motor position during a move. |
| AXEND_01.M | x | | Displays the status of an axis during various conditions of movement. |
| CAM Box | x | Design Guide page 44 | CAM Box |
| CHR_01.M | x | | Demonstrates the use of ASCII characters |
| CMODE_01.M | x | | Carries out different speed controlled continuous movements. |
| COM_OPT | x | | Program for sending and receiving 8 bytes of data via a communication option using PPO type 2. |
| CPOS_01.M | x | | Displays the internal calculated command position of an axis during a speed or position controlled motion. |
| DELAY_01.M | x | | Demonstration of a program delay and the influence of an interrupt during the delay. Definition of the interrupt sources and the corresponding subroutines (interrupt handlers). |

| File name or topic | Where to find? | | Description |
|---|---|---|---|
| | Online help | Manual or Software | |
| DIM_01.M | x | | Records and displays of a speed-diagram. |
| DORIG_01.M | x | | Defines different current positions as the real zero point. |
| Enc-S.m | x | APOSS Software | Encoder test program. |
| ERROR_01.M | x | | Evaluates an error status by the error number. |
| EXIT_01.M | x | | Panel game: You have to guess a digit. |
| Feed-forward Calculation | x | MCO 305 Operating Instructions | Feed-forward Calculation |
| GETP_01.M | x | | Displays the current PID control parameters. |
| GOSUB_01.M | x | | Displays time and position information during a movement. |
| GOTO_01.M | x | | The current time since program start is displayed every five seconds. |
| HOME_01.M | x | | Moves drives 1 to the device zero point (reference switch). |
| IF_01.M | x | | Evaluates an error status by the error number. |
| IN_01.M | x | | Displays the status of all digital inputs. |
| INB_01.M | x | | Displays the status of all digital inputs. |
| INB_02.M | x | | Displays the signal levels of all digital inputs and transfers them to the outputs. The display and the outputs are updated every time one of the inputs changes its status |
| INCL_01.M | x | | Displays some system information. |
| INCIN01.M | x | | Include file:<br>Displays the status of inputs (graphic) |
| INCPOS01.M | x | | Include file:<br>Display of the current and command position. |
| INCSTA01.M | x | | Include file:<br>Subprogram to display the system status in plain text. |
| INDEX_01.M | x | | Searches the index pulse of the encoder. |
| INKEY_01.M | x | | Intelligent input of a max. nine digit number and calculation of the cross sum. |
| LOOP_01.M | x | | Displays random horizontal bars. |
| Marker.m | x | APOSS Software | CAM Control: Printing of Cardboard Boxes with Marker Correction |
| Marker count | x | | Measurement of marker distance in connection with SYNCM |
| Marker Synchronization | x | Design Guide page 33 | Marker Synchronization for "Packaging with Varying Product Distance and Slip" application sample. |
| Mechanical Brake Control | x | Design Guide page 45 | Relative Positioning with Mechanical Brake |
| MOTOR_01.M | x | | Displays position information while the drive is moved by hand (axis control disabled). |
| Move-S.m | x | APOSS Software | Test run program. |
| MSTOP_01.M | x | | Aborts a position controlled movement and continues it after a short delay. |
| NOWAI_01.M | x | | Displays information during a motion. |
| ONINT_01.M | x | | Reaction to interrupts during a positioning motion. |

| File name or topic | Where to find? | | Description |
|---|---|---|---|
| | Online help | Manual or Software | |
| ORIG_01.M | x | | Searching the real zero point after a continuous motion in speed mode. |
| OUT_01.M | x | | Sets some outputs according to the current position. |
| OUTB_01.M | x | | Copy the status of every input to the corresponding output. |
| POS_01.M | x | | Relative and absolute positioning. |
| Position Synchronizing | x | Design Guide page 28 | Position Synchronizing for "Packaging with Fixed Product Distances" application sample. |
| Relative Positioning | x | Design Guide page 21 | Relative Positioning for "Palletizer" application sample. |
| REPEA_01.M, | x | | Counts down 10 seconds and starts a continuous motion. |
| slavesync.m | x | APOSS Software | CAM Control: Slave Synchronization with Marker |
| stamping.m | x | APOSS Software | CAM Control: Stamping of Boxes with Use-by Date |
| STAT_01.M | x | | Reads in and displays some system information. |
| syncc_msim.m | x | | Simulation of a master via software command. |
| TORIG_01.M | x | | Defines different temporary zero points and shows what it means to the current position information. |
| Touch Probe Positioning | x | Design Guide page 23 | Touch Probe Positioning for "Palletizer" application |
| Velocity Synchronizing | x | Design Guide page 25 | Velocity Synchronizing for Suit Case Conveyor Belt application. |
| VEL_01.M | x | | Varies the velocity during a position-controlled motion. |
| WAIT_01.M | x | | Demonstrates some types of waiting: time delay wait until target position is reached wait for a defined input condition. |
| WHILE_01.M | x | | Waits for an high signal on input 4 and displays all memorized inputs, made via the COM-interface during this time. |

## SYNCPOS > MCO 305 Parameters

There are some new parameters in this version. Because of the redesign of the parameter numbers, all parameter are listed with the new and SYNCPOS parameter number. The new parameters are marked "new".

Some parameters are deleted, e.g. I_BREAK or O_BRAKE. Instead there are the new parameters I_FUNCTION_n and O_FUNCTION_n which hold a number. This number defines which function is connected to the input or output. The deleted parameters are marked with "del", even they are compatible furthermore. You will find the corresponding new parameter numbers in the column "new par."

### New Parameters and all Parameters in Alphabetical Order

| Parameter Code | Parameter Name | Par. No. | SYNC-POS par. no. | new par. | Unit | Default setting |
|---|---|---|---|---|---|---|
| BANDWIDTH | PID Bandwidth | 32-64 | (35) 706 | | % | 100 |
| DFLTACC | Default acceleration | 32-85 | (34) | | % | 50 |
| DFLTVEL | Default velocity | 32-84 | (33) | | % | 50 |
| ENCODER | Incremental Resolution | 32-01 | (2) | | PPR | 1024 |
| ENCODERCLOCK | Absolute Encoder Clock Generation | 32-07 | (73) | new | | |
| ENCODER DATLEN | Absolute Encoder Data Length | 32-05 | (71) | new | Bit | 25 |
| ENCODERDELAY | Absolute Encoder Cable Length | 32-08 | (80) | new | | 0 |
| ENCODERFREQ | Absolute Encoder Clock Frequency | 32-06 | (74) | new | kHz | 262.000 |
| ENCODERTYPE | Incremental Signal Type (Slave) | 32-00 | (27) | | – | 0 |
| ENDSWMOD | Behavior at End Limit Switch | 33-40 | (44) | | – | 0 |
| ERRCOND | Behavior after error | 33-83 | (43) | | – | 0 |
| ESCCOND | Condition on program termination | 33-84 | (70) | | – | 0 |
| EXTERNAL24V | MCO supplied by External 24VDC | 33-85 | (110) | new | – | 0 |
| FFACC | Acceleration Feed-forward | 32-66 | (37) 708 | | % | 0 |
| FFVEL | Velocity Feed-forward | 32-65 | (36) 707 | | % | 0 |
| HOME_FORCE | Force HOME? | 33-00 | (3) | | – | 0 |
| HOME_OFFSET | Zero Point Offset from Home Position | 33-01 | (42) | | qc | 0 |
| HOME_RAMP | Ramp for Home Motion | 33-02 | (41) | | % | 10 |
| HOME_VEL | Velocity of Home Motion | 33-03 | (7) | | % | 10 |
| HOME_TYPE | Behavior during Home Motion | 33-04 | (40) | | – | 0 |
| I_BREAK | Input for abort | del | (105) | 33-50 … | – | 0 |
| I_CONTINUE | Continue program | del | (106) | 33-50 … | – | 0 |
| I_ERRCLR | Clear error | del | (107) | 33-50 … | – | 0 |
| I_FUNCTION_n | Terminal X57/n Digital Input | 33-50...59, 33-61, 33-62 | (45-47, 103-107) | new | – | 0 |
| I_NEGLIMITSW | Negative limit switch | del | (47) | 33-50 … | – | 0 |
| I_POSLIMITSW | Positive limit switch | del | (46) | 33-50 … | – | 0 |
| I_PRGCHOICE | Input for beginning program choice | del | (104) | 33-50 … | – | 0 |

| Parameter Code | Parameter Name | Par. No. | SYNC-POS par. no. | new par. | Unit | Default setting |
|---|---|---|---|---|---|---|
| I_PRGSTART | Input for program start | del | (103) | 33-50 … | – | 0 |
| I_REFSWITCH | Input for reference switch | del | (45) | 33-50 … | – | 0 |
| IOMODE | Terminal X59/1 and X59/2 Mode | 33-60 | (113) | new | | 0 |
| KDER | Derivative Value for PID Control | 32-61 | (12) 703 | | – | 1 |
| KILIM | Limit value for integral sum | 32-63 | (21) 705 | | – | 0 |
| KINT | Integral Factor | 32-62 | (13) 704 | | – | 0 |
| KPROP | Proportional Factor | 32-60 | (11) 702 | | – | 30 |
| MENCODER CLOCK | Absolute Encoder Clock Generation | 32-37 | (77) | new | – | [1] on |
| MENCODER | Incremental Resolution | 32-31 | (30) | | PPR | 1024 |
| MENCODER DATLEN | Absolute Encoder Data Length | 32-35 | (75) | new | Bit | 25 |
| MENCODER DELAY | Absolute Encoder Cable Length | 32-38 | (81) | new | | 0 |
| MENCODERFREQ | Absolute Encoder Clock Frequency | 32-36 | (78) | new | kHz | 262.000 |
| MENCODERTERM | Encoder Termination | 32-40 | (76) | new | – | 1 |
| MENCODERTYPE | Incremental Signal Type (Master) | 32-30 | (67) | | – | 1 |
| NEGLIMIT | Negative Software End Limit | 33-41 | (4) | | qc | –500000 |
| O_AXMOVE | Output for motion command active | del | (64) | 33-63… | – | 0 |
| O_BRAKE | Output for mechanical brake | del | (48) | 33-63… | – | 0 |
| O_ERROR | Output for error | del | (108) | 33-63… | – | 0 |
| O_FUNCTION_n | Terminal X59/n Digital Output | 33-63…70 | (48, 64, 108) | new | – | 0 |
| POSDRCT | Rotational Direction | 32-10 | (28) | | – | 1 |
| POSERR | Maximum Tolerated Position Error | 32-67 | (15) | | qc | 20000 |
| POSFACT_N | User Unit Denominator | 32-11 | (26) | | – | 1000 |
| POSFACT_Z | User Unit Numerator | 32-12 | (23) | | – | 1000 |
| POSLIMIT | Positive Software End Limit | 33-42 | (5) | | qc | 500000 |
| PRGPAR | Activated program number | 33-80 | (102) 701 | | – | –1 |
| PROFTIME | Scan Time for Profile Generator | 32-70 | (29) | | 1 | ms |
| RAMPMIN | Shortest Ramp | 32-81 | (31) | | s | 1 |
| RAMPTYPE | Ramp Type | 32-82 | (32) | | | 0 |
| REGWMAX | Size of the Control Window (Activation) | 32-71 | (38) | | qc | 0 |
| REGWMIN | Size of the Control Window (deactivation) | 32-72 | (39) | | qc | 0 |
| REVERS | Reverse Behavior for Slave | 32-68 | (63) | | – | 0 |
| STATUS MONITORING | Drive Status Monitoring | 33-82 | (79) 700 | new | – | 1 |
| SWNEGLIMACT | Negative Software End Limit Active | 33-43 | (19) | | – | 0 |
| SWPOSLIMACT | Positive Software End Limit Active | 33-44 | (20) | | – | 0 |

| Parameter Code | Parameter Name | Par. No. | SYNC-POS par. no. | new par. | Unit | Default setting |
|---|---|---|---|---|---|---|
| SYNCACCURACY | Size of the precision window for position synchronization | 33-13 | (55) | | qc | 1000 |
| SYNCFACTM | Synchronization factor master (M:S) | 33-10 | (49) | | qc | 1 |
| SYNCFACTS | Synchronization factor slave (M:S) | 33-11 | (50) | | qc | 1 |
| SYNCFAULT | Marker number for fault | 33-24 | (57) | | – | 10 |
| SYNCMARKM | Marker number for master | 33-15 | (52) | | – | 1 |
| SYNCMARKS | Marker number for slave | 33-16 | (53) | | – | 1 |
| SYNCMFTIME | Filter Time for Marker Correction | 33-29 | (18) | | 1 ms | 0 |
| SYNCMFPAR | Marker Filter Configuration | 33-28 | (17) | | – | 1 |
| SYNCMMAXCORR | Maximum Marker Correction | 33-30 | (6) | | qc | 0 |
| SYNCMPULSM | Master Marker Distance | 33-17 | (58) | | qc | 500 |
| SYNCMPULSS | Slave Marker Distance | 33-18 | (59) | | qc | 500 |
| SYNCMSTART | Start behavior for synchronization | 33-23 | (62) | | – | 0 |
| SYNCMTYPM | Master Marker Type | 33-19 | (60) | | – | 0 |
| SYNCMTYPS | Slave Marker Type | 33-20 | (61) | | – | 0 |
| SYNCMWINM | Master Marker Tolerance Window | 33-21 | (68) | | qc | 0 |
| SYNCMWINS | Slave Marker Tolerance Window | 33-22 | (69) | | qc | 0 |
| SYNCOFFTIME | Offset Filter Time | 33-27 | (16) | | ms | 0 |
| SYNCPOSOFFS | Position Offset for Synchronization | 33-12 | (54) | | qc | 0 |
| SYNCREADY | Marker Number for Ready | 33-25 | (56) | | – | 1 |
| SYNCTYPE | Synchronization Type | 33-31 | (51) | | – | 0 |
| SYNCVELREL | Relative Slave Velocity Limit | 33-14 | (66) | | % | 0 |
| SYNCVFTIME | Velocity Filter | 33-26 | (65) 709 | | $\tau$_filt (µs) | 0 |
| TESTTIM | Time in Target Window | 33-45 | (24) | | ms | 0 |
| TESTVAL | Target Window Limit Value | 33-46 | (25) | | qc | 1 |
| TESTWIN | Size of Target Window | 33-47 | (8) | | qc | 0 |
| TIMER | Sampling time for PID control | 32-69 | (14) | | ms | 1 |
| VELMAX | Maximum Velocity (Encoder) | 32-80 | (1) | | RPM | 3000 |
| VELRES | Velocity Resolution | 32-83 | (22) | | | 100 |

## □ What's New in the Update Version

This release is mainly caused by the new ramp type for movements with "limited jerk".

### □ New and Extended Parameters

| | |
|---|---|
| Par. 32-50 Source Slave | Choose Feedback Source for Slave. |
| Par. 32-82 Ramp Type | Another ramp type '2' for all movements with limited jerk. |
| | Jerk Duration JERKMIN<br>Defines the minimum time [ms] required before reaching the maximum acceleration. Four different JERKMIN parameters are available, see also "Limited Jerk" in chapter "Functions and Samples". |

### □ New SYSVAR Indices

| | | |
|---|---|---|
| 4275 | PFG_G_JSTATE | Contains the state of the Limited Jerk Movement state machine. |
| 4277 | PFG_G_JERKSTOPPATH | Delivers the length of the stop path. |

## □ 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 | VersioNumber | (dec) | 100 | Version as decimal (1.00 = 100) |
| 3 | CurveType | (dec) | 0 | 0 = symmetrical;<br>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 |

| Index | Name | Unit | Value | Description |
|---|---|---|---|---|
| 10 | IndexFIP | (dec) | IndexSTP + STPno*2 | Index to Fix point Part |
| 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 | | | | |

# □ Index

## D



## E



## F



## G



## H



## I

## K



## L



## M



## N



## O

## P



## Q



## R



## S

## T



## U



## V



## W