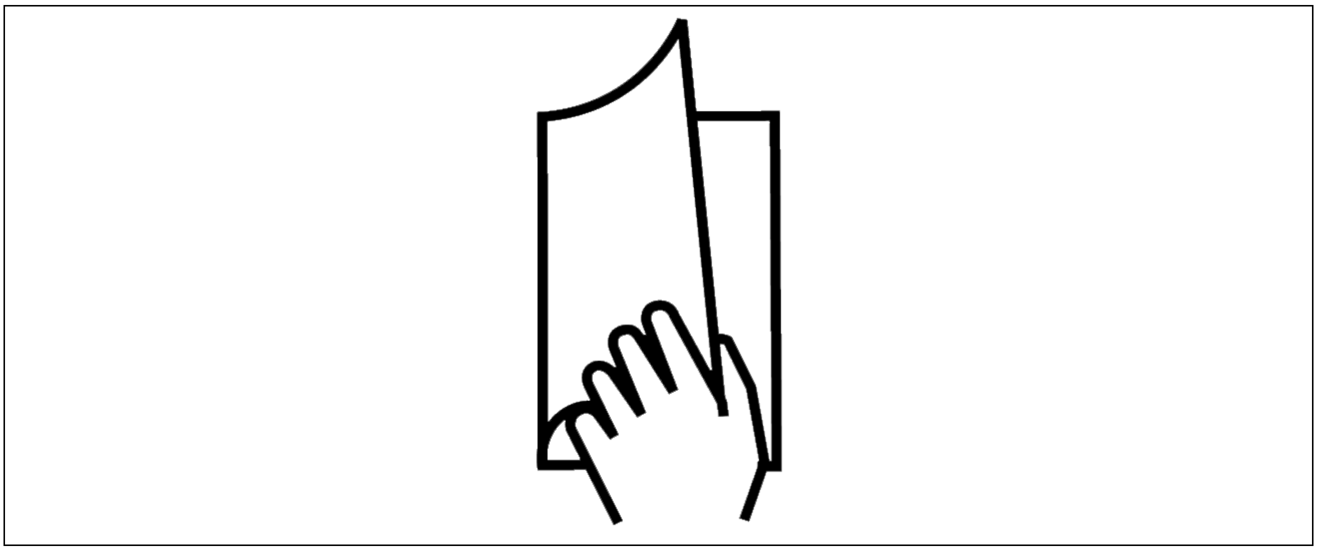
Contents

__ How to Read this MCO 305 Design Guide __

How to Read this MCO 305 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'.

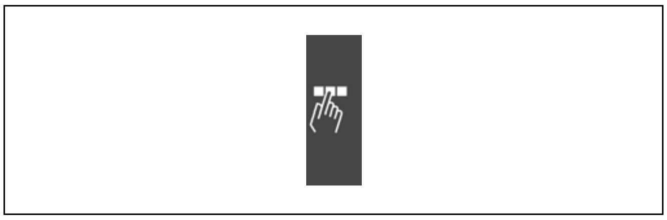
Chapter **Introduction to MCO 305** informs you about the functionality and features of the MCO 305, gives a system overview including configuration examples, and informs you about some basic topics like encoder and program execution.

Page divider for 'Introduction'.

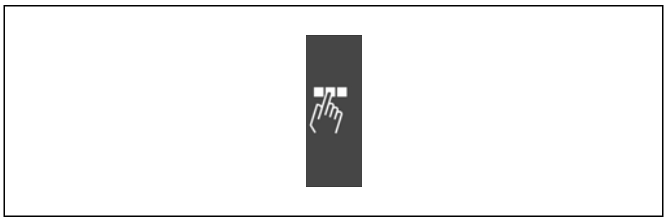
Chapter **Functions and Examples** guides you through some applications examples from simple positioning to different synchronizations as well as CAM controls. Setting the parameters, programming of controls, and editing of curves can be reconstructed in detail with these examples.

Page divider for 'Functions and Examples'.

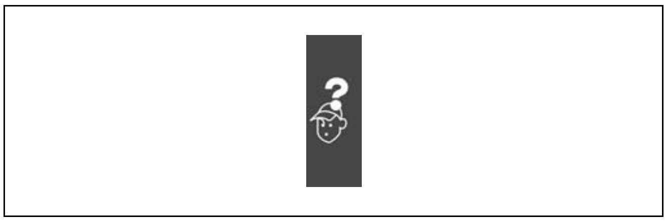
Chapter **PC Software Interface** informs you about the APOSS specific menus and functions. Click on → *Help* in the APOSS menu bar for more details. Chapter **APOSS Tools** provides detailed information about the CAM-Editor, Array-Editor as well as the APOSS Oscilloscope.


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 description of all commands arranged in groups and all parameters in the 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'.

The manual ends with an index.

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

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

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

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

□ Symbols and Conventions

Symbols used in this manual:

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

Indicates a general warning.

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

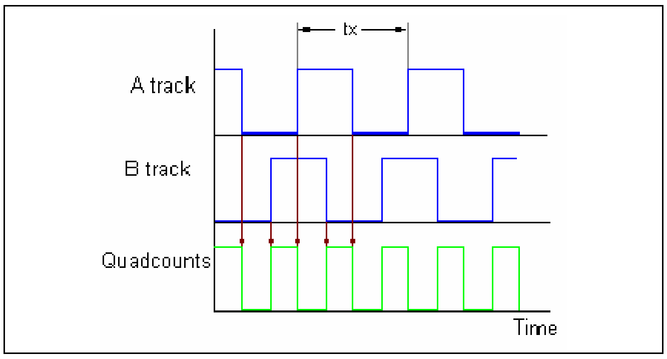
Changes to online parameters are activated immediately after the data value is changed. Changes to offline parameters are not activated until you enter [OK] on the LCP.

□ 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

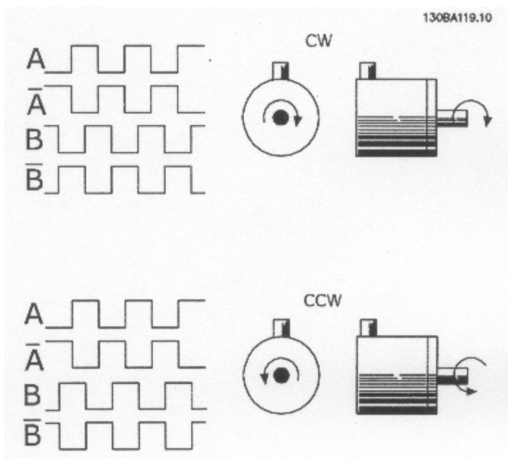
□ Encoder Direction

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.

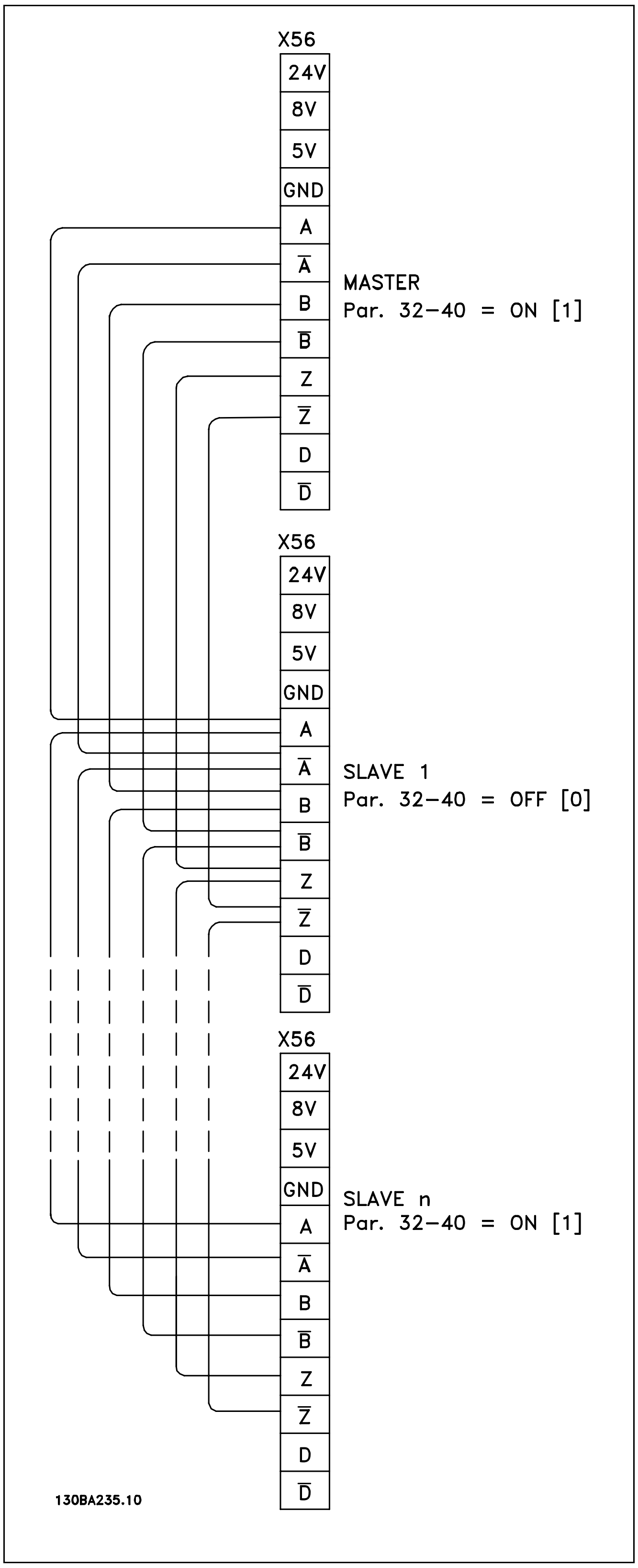
Counter Clockwise direction means channel B is 90 electrical degrees before A.

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.

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

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

SYNCFACTS/SYNCFACTM to multiply the delta_master.

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

Concerning the error we are dealing with, this means:

Normal case

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

Big factors

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

This should not be relevant in most applications.

User Units [UU]

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

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

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

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:

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

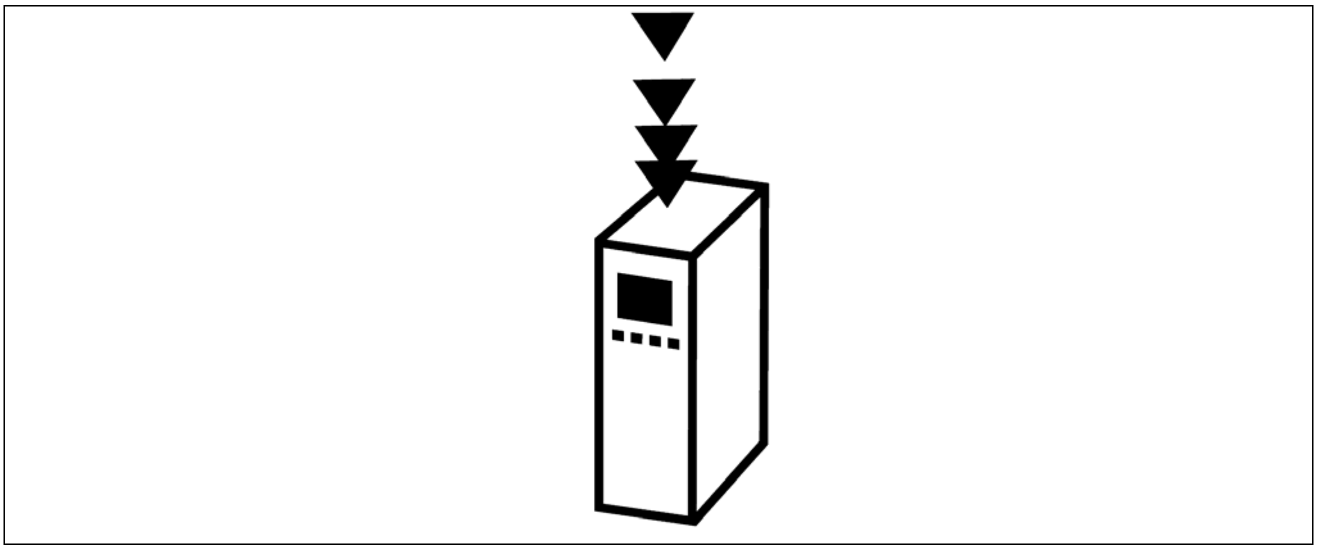
par. 33-10 *Synchronization Factor Master* SYNCFACTM
par. 33-11 *Synchronization Factor Slave* SYNCFACTS

□ 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 312 and FC 312; it adds functionality and flexibility to the already very comprehensive standard functionality of these drives.

FC 312 and FC 312 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 312 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.

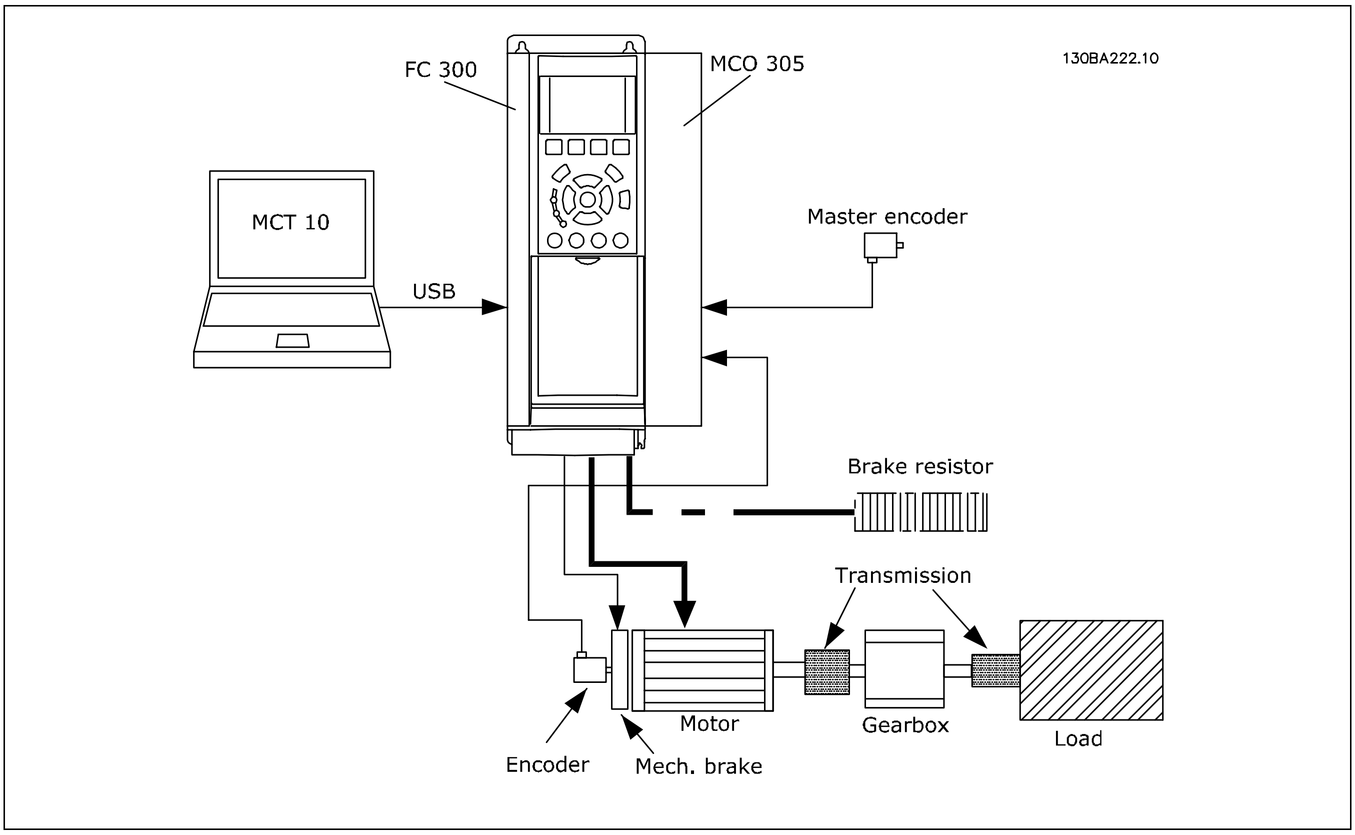
□ System Overview

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.

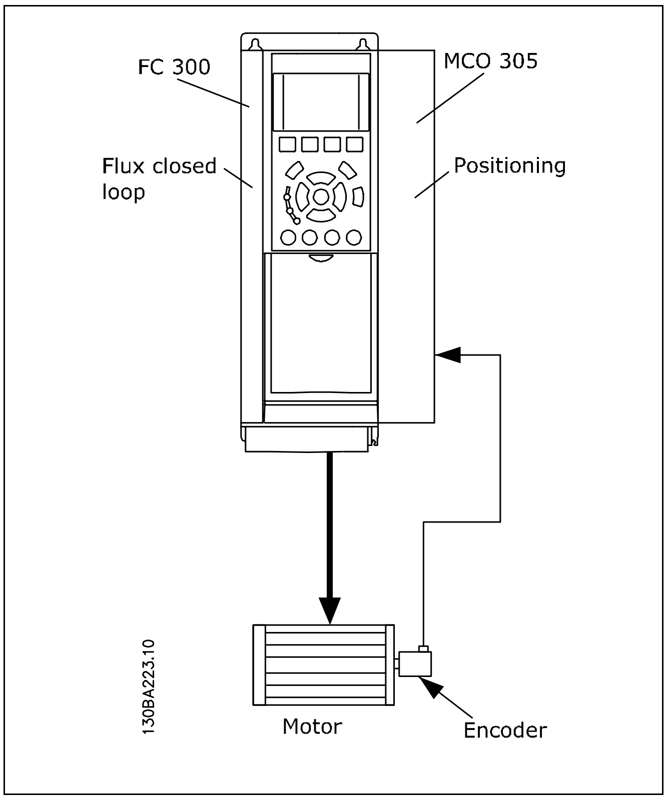
The following might also be required:

- Brake resistor for dynamic braking.
- Mechanical brake.

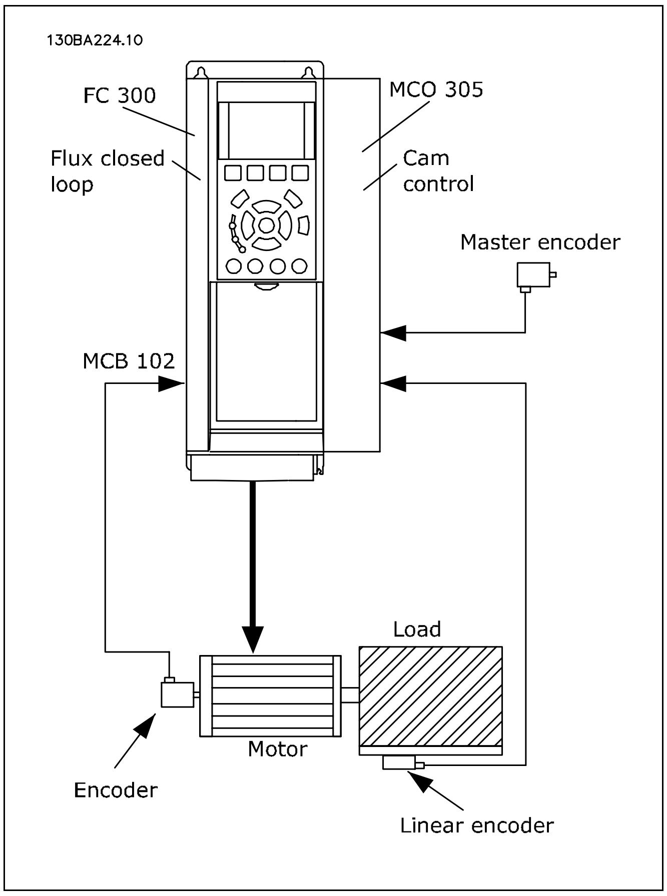


□ Configuration Examples

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

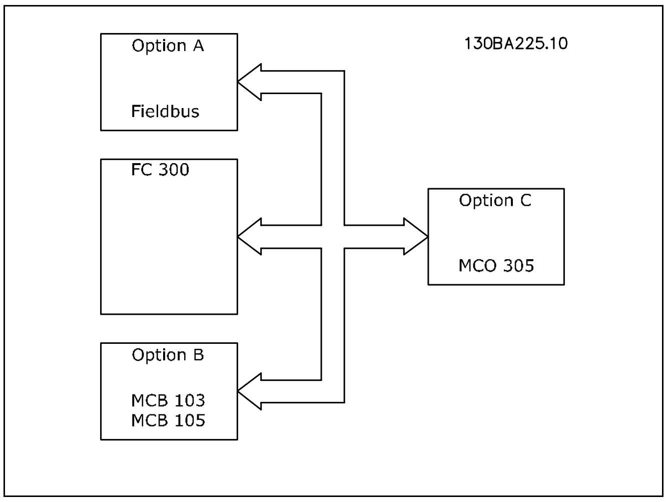
The Interface between MCO 305 and the FC 300 control card provides read/write access to all parameters as well as reading status of all inputs and the possibility to control all outputs. In addition various process data such as status word and actual motor current can be read by the MCO 305 application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

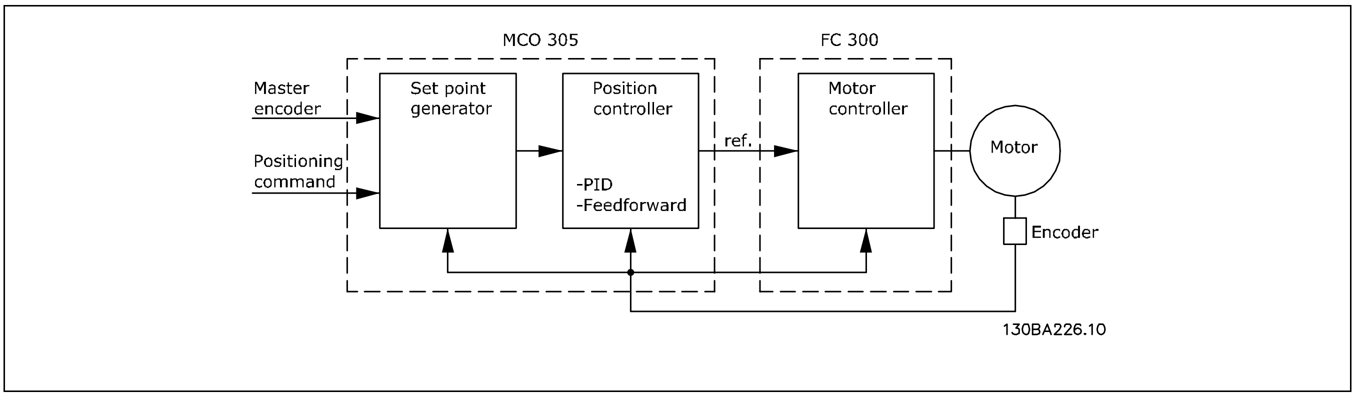
General purpose I/O option MCB 103: Status of inputs can be read and outputs can be controlled via the MCO 305 application program.



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).

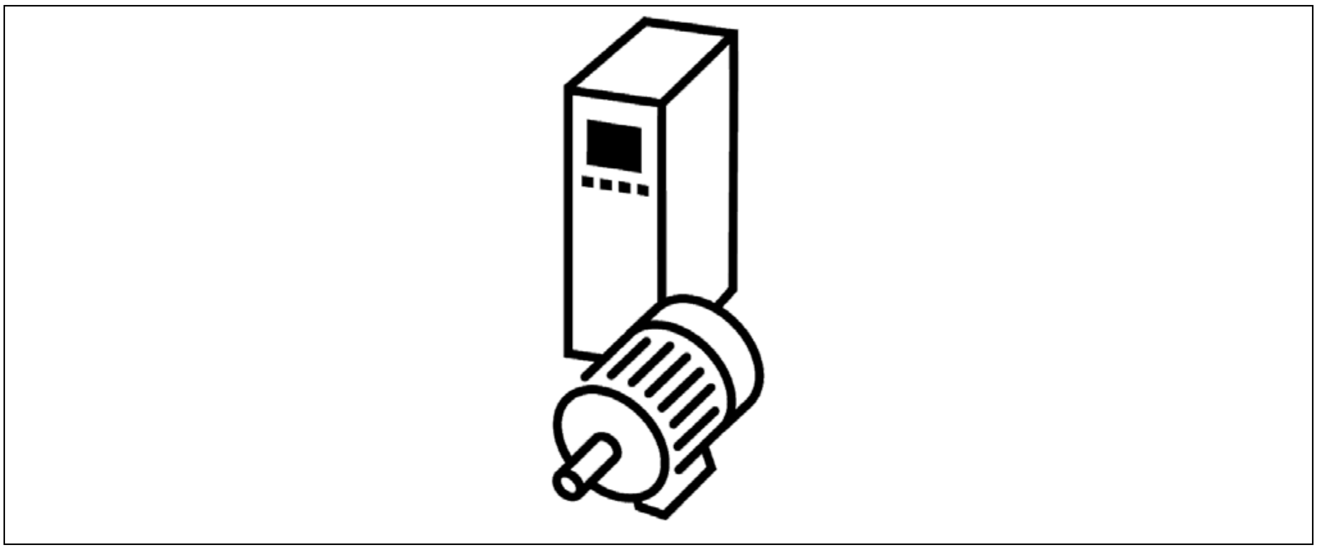
Starting the Autostart program after power-up can be avoided by pressing the [Cancel] key of the FC 300 LCP during power-up. The key must be pressed until the message User abort (error 119) appears in the display.

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.

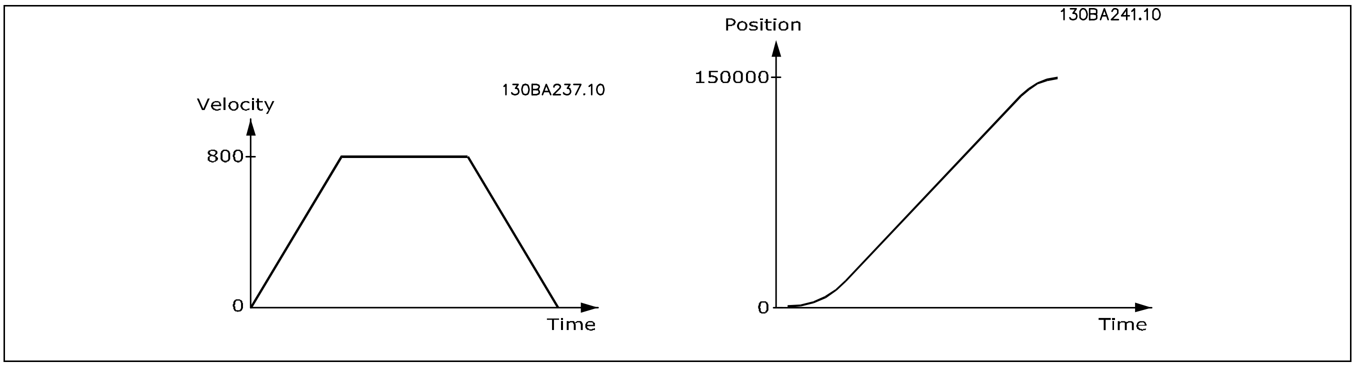
MCO 305 offers three main positioning types

- Absolute
- Relative
- Touch Probe

<u>Absolute Positioning</u>

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.

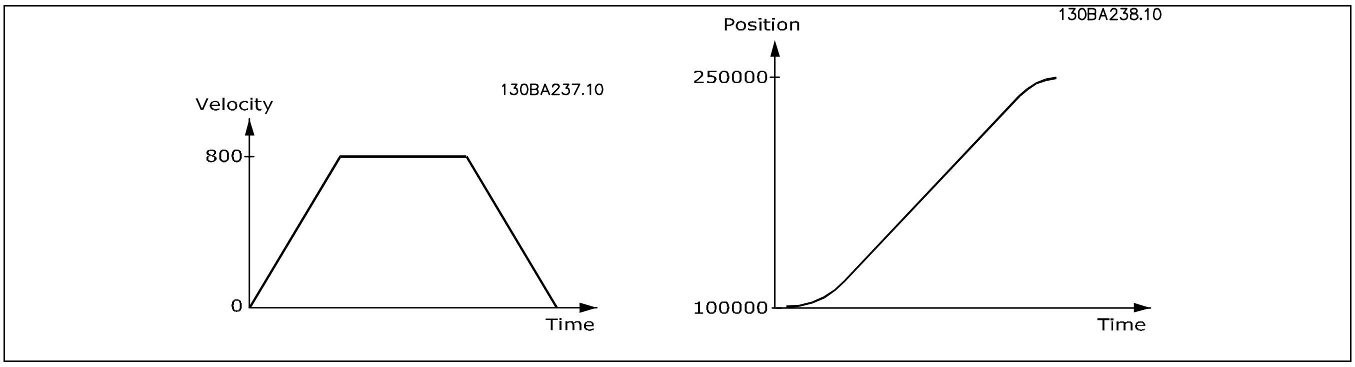
If the starting position is 0 and with an absolute positioning to 150.000 the target position is 150.000, the drive will thus move a distance of 150.000. If on the other hand the starting position is 100.000 and with an absolute positioning to 150.000 the target position is still 150.000 but the drive 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.

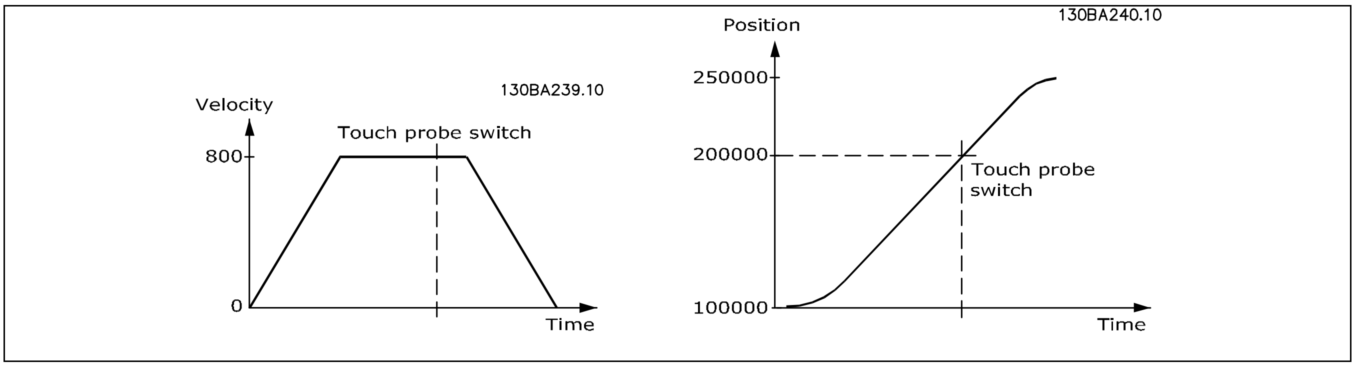


Touch Probe Positioning

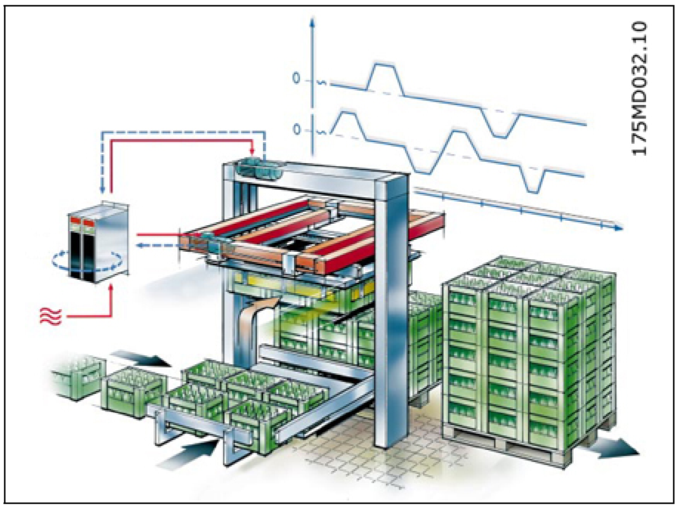
With touch probe positioning, the positioning is related to the actual position when the touch probe input is activated, that means the target position is the position of the touch probe + the positioning distance. Touch probe positioning is thus relative positioning relating to a touch probe instead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

32-0*	Encoder 2 – Slave	page 201
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213
33-0*	Home Motion	page 217
33-4*	Limit Handling	page 229

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 201
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213

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 201
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213
33-4*	Limit Handling	page 229

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 1 - 8 = reaction to the rising edge –1 - 8 = reaction to the falling edge 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

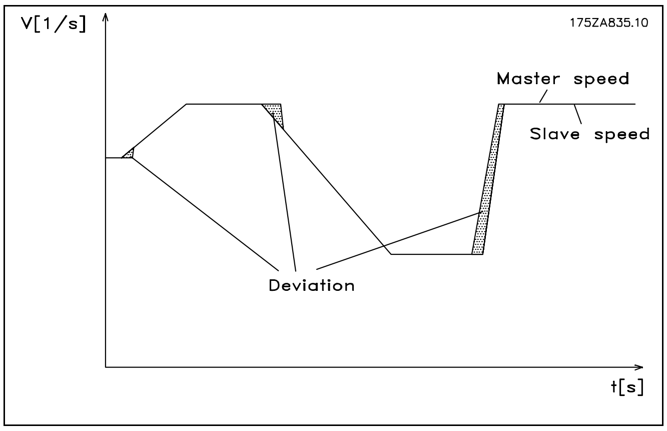
□ Velocity Synchronization (SYNCV)

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

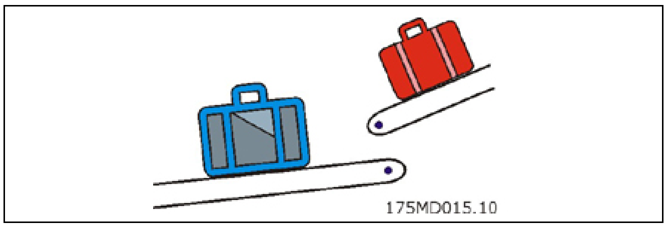


Control behavior with velocity synchronization.

□ 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 201
32-3*	Encoder 1 – Master	page 205
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213
33-1*	Synchronization	page 218

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)

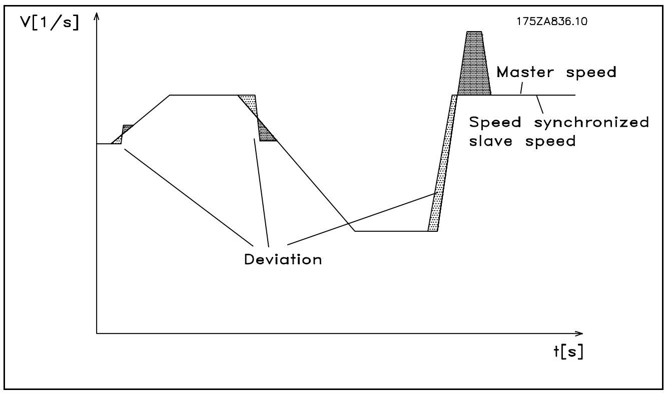
Position synchronization (SYNCP) is closed loop position control with a moving target where the set-point (commanded position) is the master position multiplied by the gear-ratio also considering any position off-set. The slave position is controlled based on this set-point and the actual position feedback 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 dyna-mic shaft the master as this shaft will anyway be the limiting factor for the system performance.

Typical applications are:

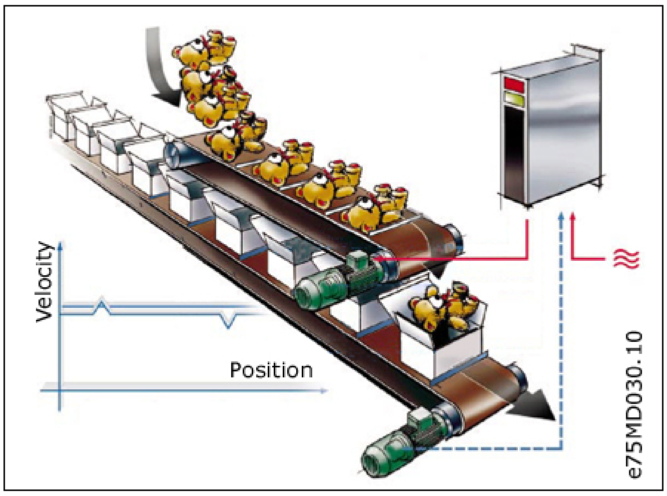
- Bottle washing machines.
- Foil wrapping.
- Packaging machines.
- Conveyors.
- Multi axis hoists.
- Filling.
- Printing.
- Cut on the fly.



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:

32-0*	Encoder 2 – Slave	page 201
32-3*	Encoder 1 – Master	page 205
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213
33-1*	Synchronization	page 218

Command	Description	Syntax	Parameter
DEFSYNCORIGIN	Defines master-slave relation for the next SYNCP or SYNCM command	DEFSYNCORIGIN master slave	master = reference position in qc 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.

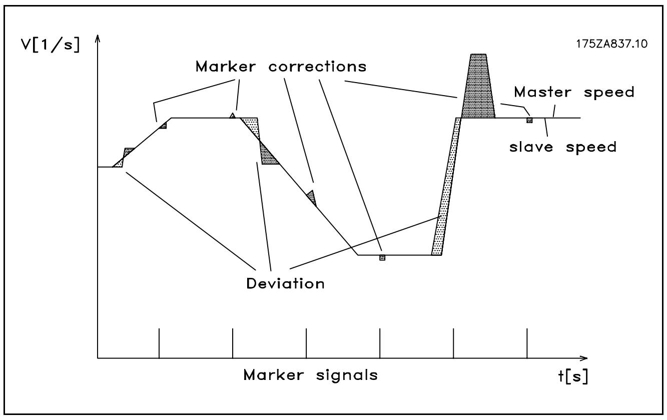
When starting marker synchronization the initial behavior will be like position synchronization but when the first set of markers has been detected marker correction will start. Which markers to use for the first marker correction can be defined by parameter 33-23, by defining the start behavior it is 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

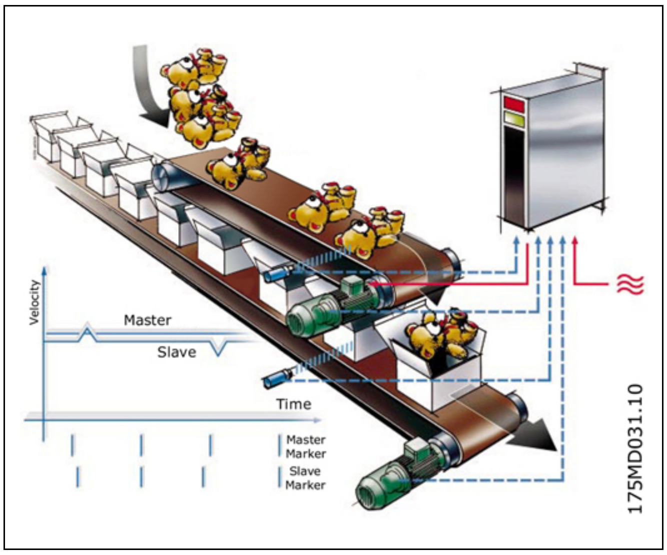
□ Application Example: Packaging with Varying Product Distance and Slip

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:

32-0*	Encoder 2 – Slave	page 201
32-3*	Encoder 1 – Master	page 205
32-6*	PID-Controller	page 210
32-8*	Velocity & Acceleration	page 213
33-1*	Synchronization	page 218

Command	Description	Syntax	Parameter
DEFSYNCORIGIN	Defines master-slave relation for the next SYNCP or SYNCM command.	DEFSYNCORIGIN master slave	master = reference position in qc 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)
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
```

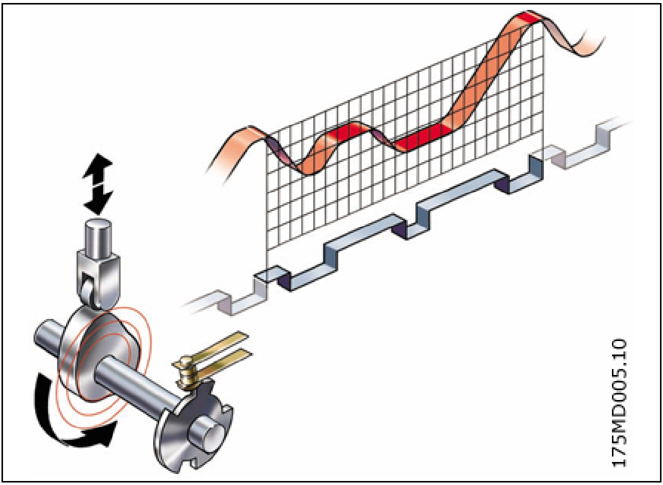
```
            IF (skip_first == 0) THEN        // Do not use first value as it might be invalid
                skip_first = 1
            ELSE
                marker_number = marker_number + 1   // Increment counter
                total_dist = total_dist + marker_distance // Summarize marker distances
            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*. into which you first load the controller parameters that have already been set. 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

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 either 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 the gradient of the curve at the beginning and the end is set to zero.

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.*

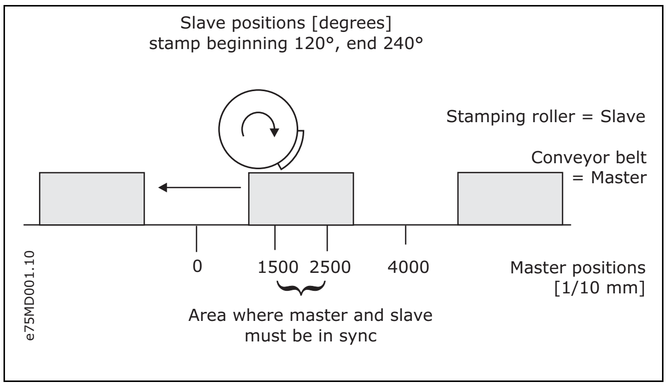
Internal Realization as Array

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:


Slave positions [degrees]
stamp beginning 120°, end 240°
Stamping roller = Slave
Conveyor belt = Master
0
1500
2500
4000
Master positions [1/10 mm]
Area where master and slave must be in sync
e75MD001.10

How to Edit the Curve Step by Step

1. Set the FC 300 with the required parameters.
2. Select this .zbc (or .cnf) file, APOSS will then automatically open the selected file as well as the *CAM-Editor*.

   <u>Stand-alone APOSS</u>:
   Start → *CAM-Editor* and load this .zbc (or .cnf) file.
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.

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

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

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

   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

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

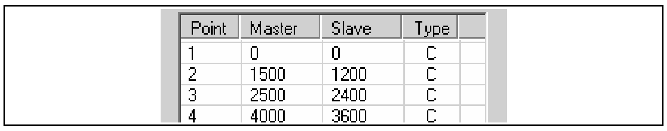
$$\frac{5/1 * 500 * 4}{3600}\text{qc} = \frac{5 * 500 * 4}{3600}\text{qc} = 1\,\text{UU}$$

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

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

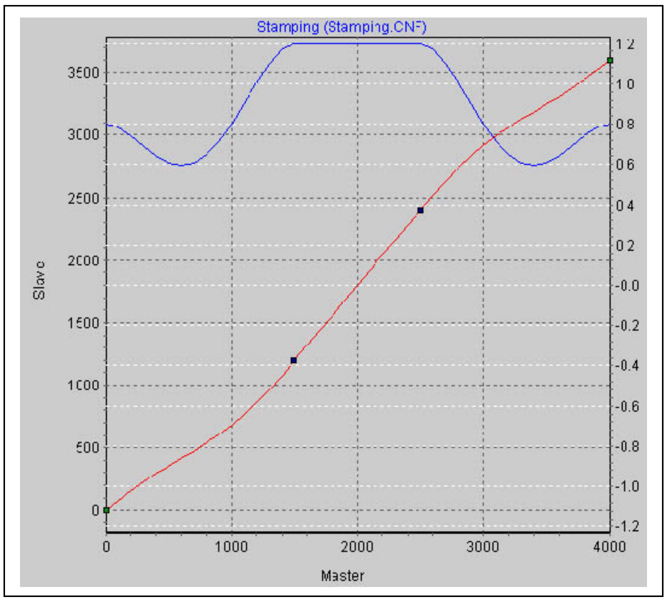
par. 32-12 *User Unit Numerator*	= 25
par. 32-11 *User Unit Denominator*	= 9

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.



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

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 .zbc file by clicking the "Save 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 .zbc 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:

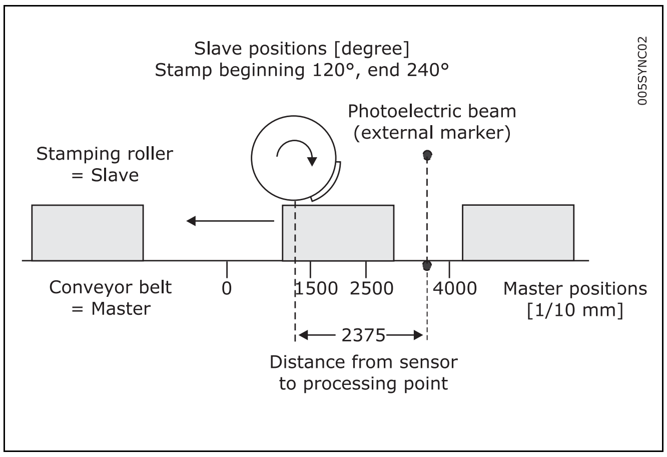
```
DIM stamping[92]          // Number of elements from the title bar of the CAM-Editor
HOME                      // Slave axis performs a home run (switch for zero position on top)
   // Afterwards, the slave will be in the zero position (0 degrees)
   // (Omitted if an absolute encoder is used)
SETCURVE 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.

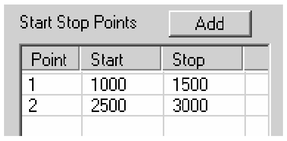
The following section describes how you can adapt the curve of the previous example to this application.

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.



Point	Start	Stop
1	1000	1500
2	2500	3000

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.

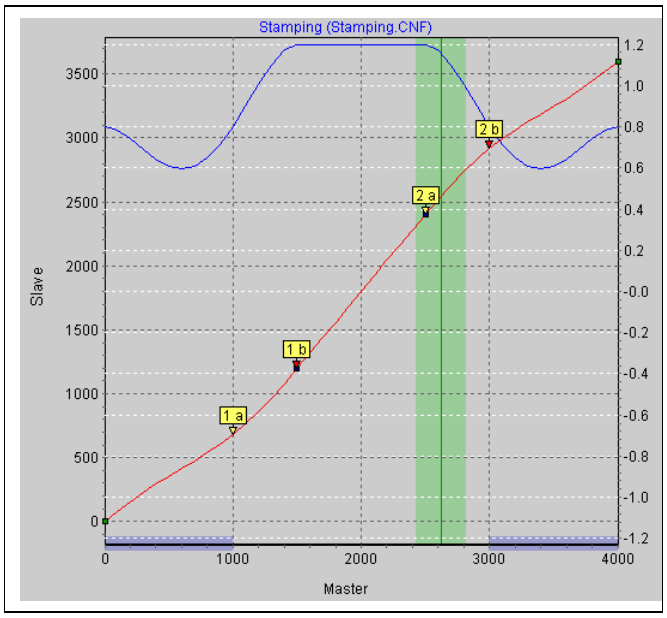
par. 33-17	*Master Marker Distance*	= 2375
par. 33-21	*Master Marker Tolerance Window*	= 200
par. 33-19	*Master Marker Type*	= 2

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 = 1000Enter 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 as" button to save the .zbc file, for example "marker".
16. Load the .zbc 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

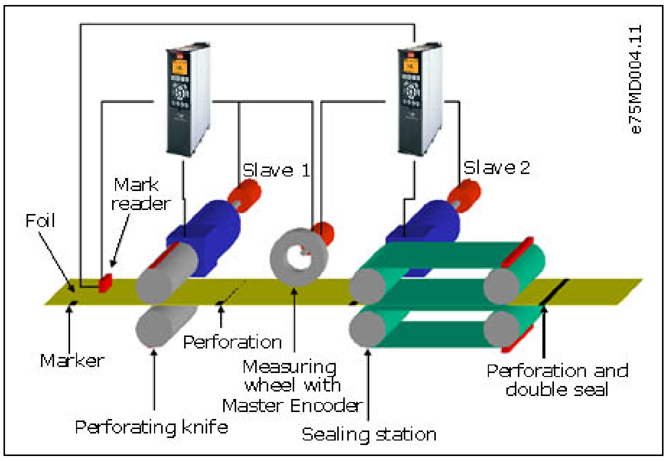
Since the curve is stored internally as an array, the DIM instruction must appear first in your program:

```
/******** 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
```

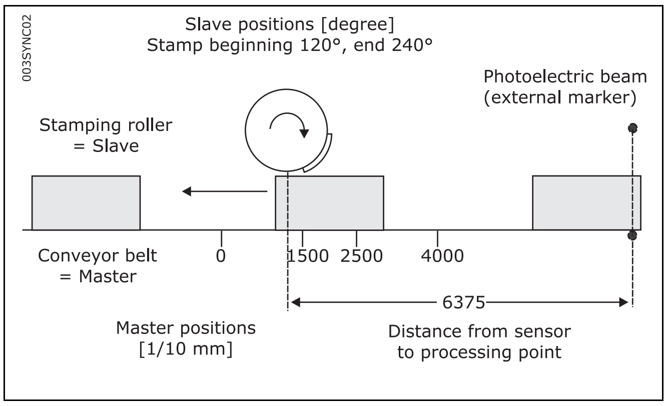
☐ If the Sensor Distance is Larger than one Master Cycle Length

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.



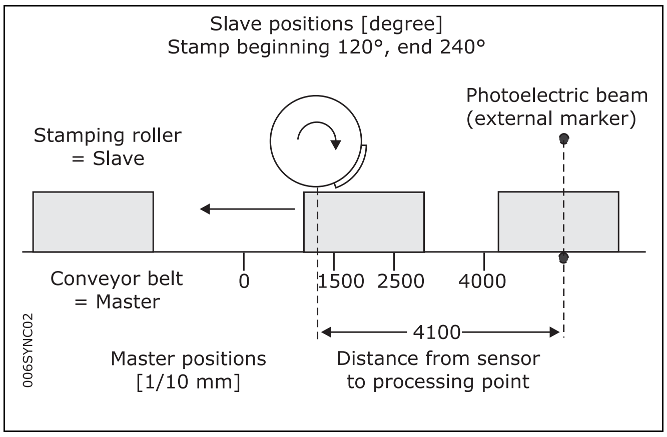
In order to assess in what area corrections may be made, you should subtract the master cycle length as often as necessary until the value is < 1 master cycle length. This is the maximum permitted distance for making corrections. In this example, it is 6375 – 4000 = 2375 and thus the same correction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


003SYNC02
Slave positions [degree]
Stamp beginning 120°, end 240°
Photoelectric beam (external marker)
Stamping roller = Slave
Conveyor belt = Master
0
1500
2500
4000
6375
Master positions [1/10 mm]
Distance from sensor to processing point

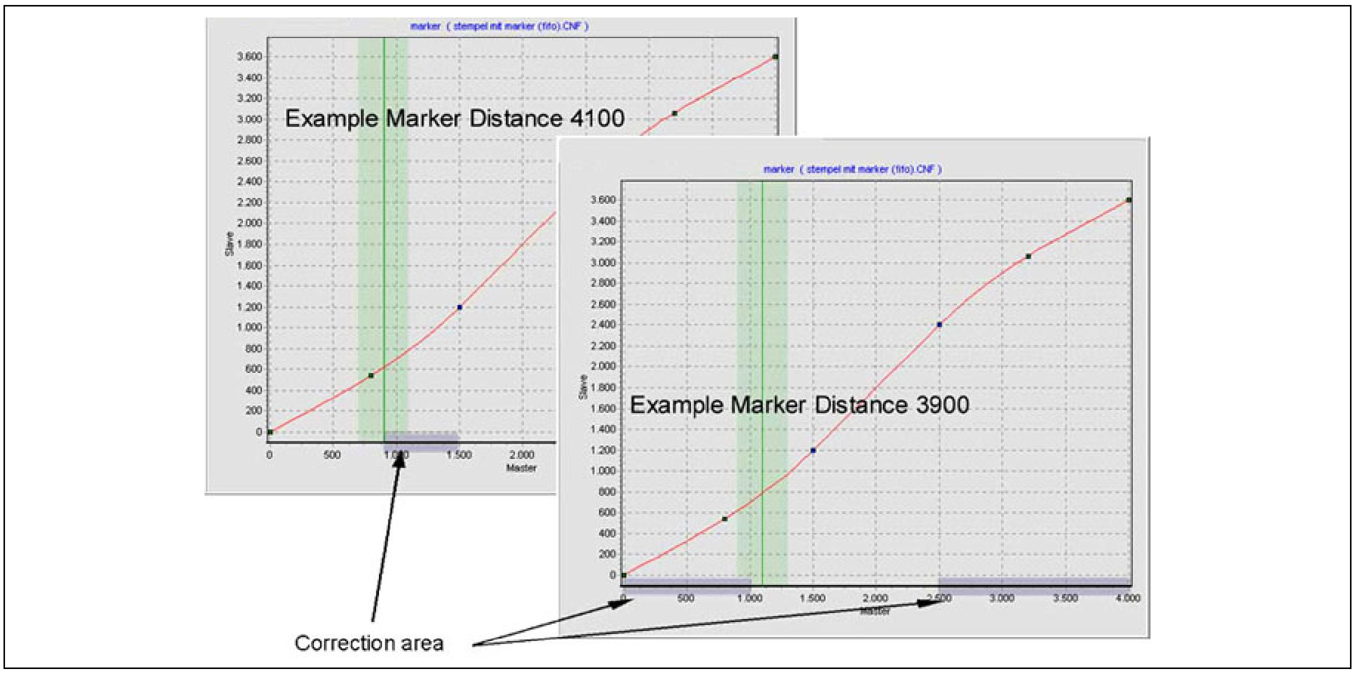
Problematic Situations in the Determination of the Marker Distance

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:


Slave positions [degree]
Stamp beginning 120°, end 240°
Photoelectric beam (external marker)
Stamping roller = Slave
Conveyor belt = Master
0
1500
2500
4000
4100
006SYNC02
Master positions [1/10 mm]
Distance from sensor to processing point

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. See the effects (correction area) by means of the two screen shots.



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*:

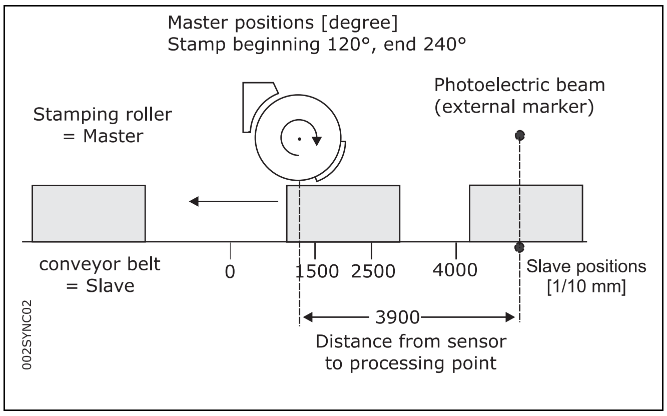
par. 33-17 *Master Marker Distance*	= 3900
Master Marker Position	= 800

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.


Master positions [degree]
Stamp beginning 120°, end 240°
Stamping roller = Master
Photoelectric beam (external marker)
conveyor belt = Slave
0
1500
2500
4000
Slave positions [1/10 mm]
3900
Distance from sensor to processing point
002SYNC02

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 "zbc".
2. This .zbc 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 Data:

   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.

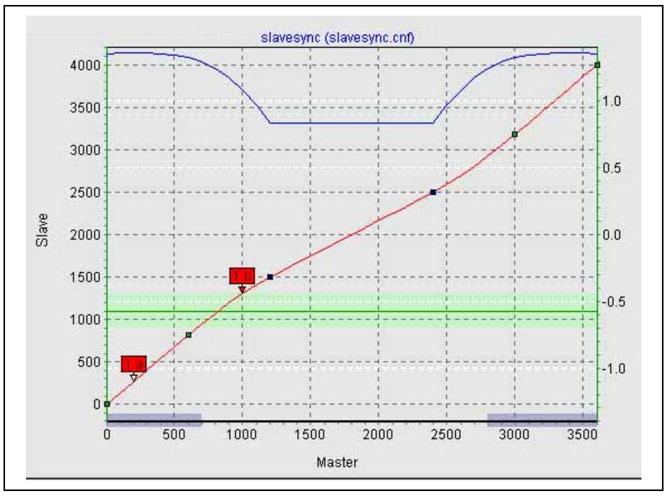
   A complete cycle length of the master is 360 degrees; this corresponds to 3600 MU.
   For a master cycle length of 3600, the → *Number of Intervals* = 36 produces a reasonable interval time of 27.7 ms. Set these two values using the "Set" button in the *Curve Data* index card.
6. Define → *Fix points* for the roller (slave) and the conveyor belt (master). The function → *Snap on Grid* should be activated.

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

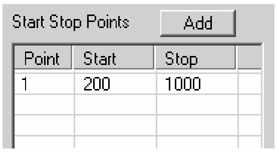
7. Master and slave must run synchronously with the same velocity between the master positions 1200 to 2400. For this, you need to have a straight line that is determined with two tangent points. Double-click in the column → *Type* for the fixpoint at position 2400.

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.


Start Stop Points

Point	Start	Stop
1	200	1000

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 as" button to save the .zbc file.
16. Load the .zbc 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

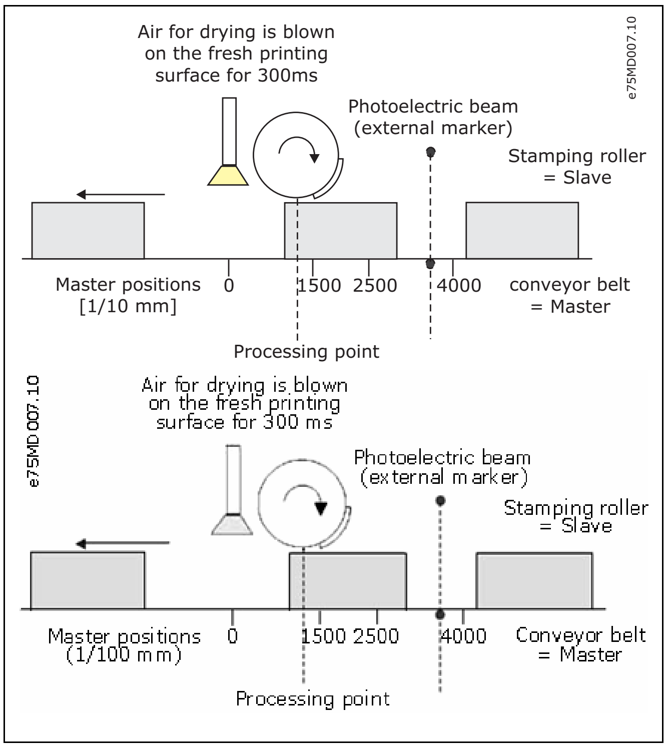
The mechanical camshaft is also reproduced by one (or more) curves. In order to realize a CAM box, it must be possible to engage and disengage the slave at specific master positions over and over again.

This is possible with APOSS with the interrupt command ON MAPOS 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:

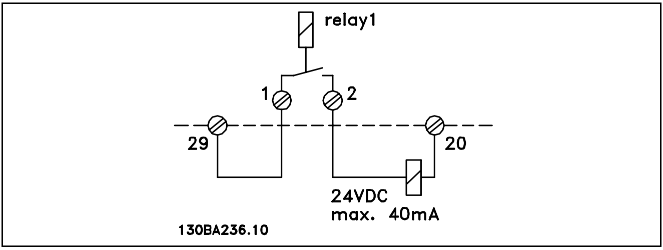

Air for drying is blown on the fresh printing surface for 300ms
Photoelectric beam (external marker)
Stamping roller = Slave
Master positions [1/10 mm]
0
1500
2500
4000
conveyor belt = Master
Processing point
e75MD007.10
Air for drying is blown on the fresh printing surface for 300 ms
Photoelectric beam (external marker)
Stamping roller = Slave
Master positions (1/100 mm)
0
1500 2500
4000
Conveyor belt = Master
Processing point
e75MD007.10

```
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

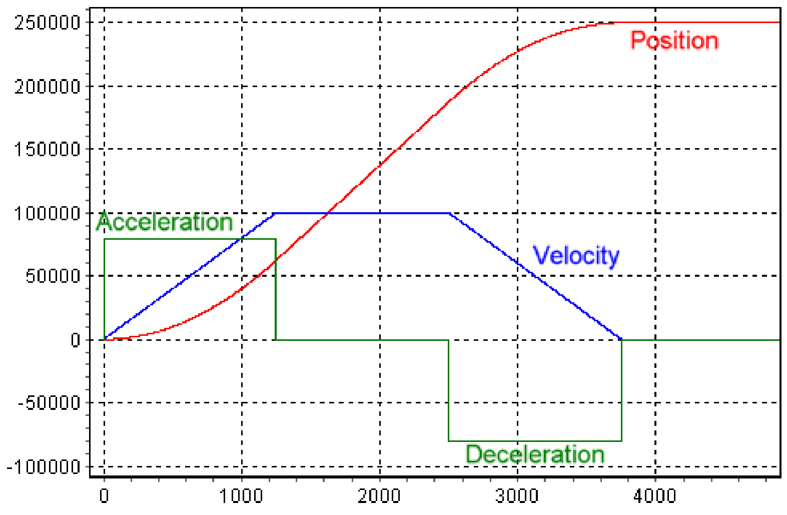
□ Understanding Limited-Jerk Movements

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.

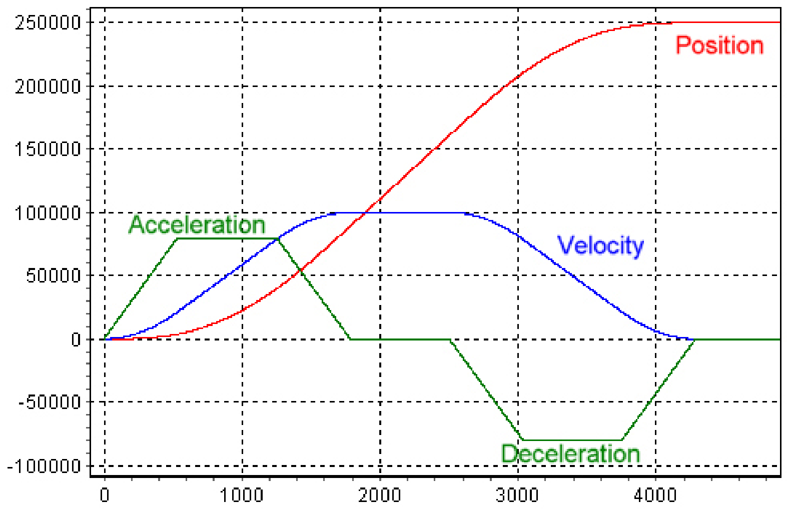
Typical applications where limited jerk is required:

- 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.

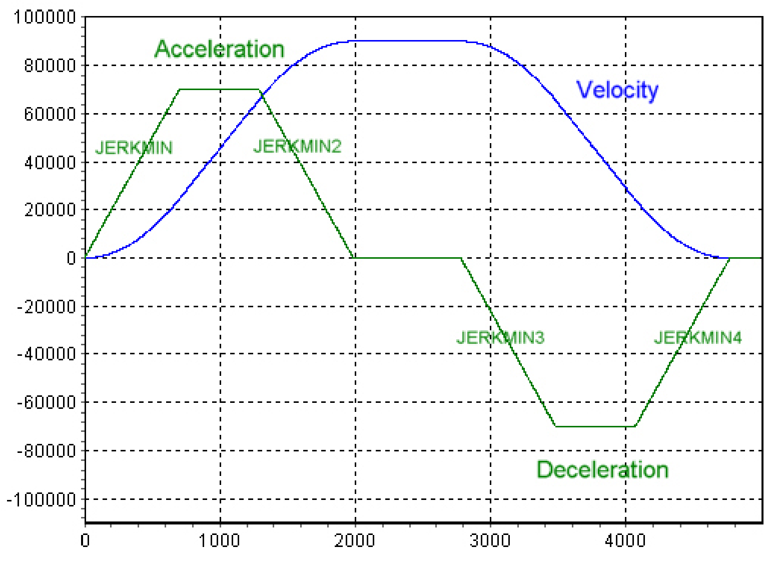


The user can control the “gentleness” of the acceleration ramp using 4 parameters:

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 consequen-ces 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:

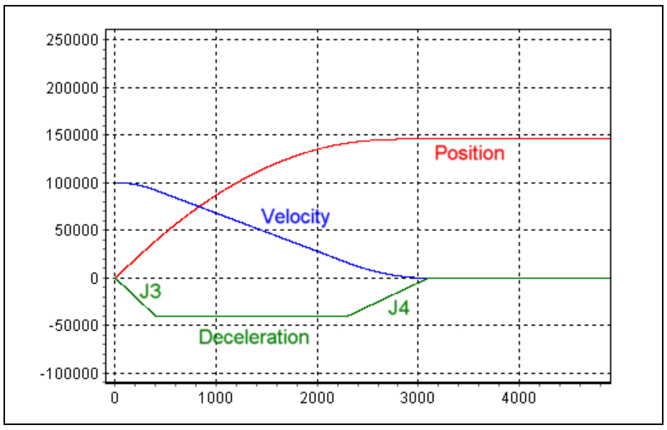
1. Stopping from the current velocity and acceleration (where the final position is not important).
2. Changing from the current velocity and acceleration to a specified constant velocity (where the positions are not important).
3. Moving from the current position (and the current velocity and acceleration)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

□ Examples

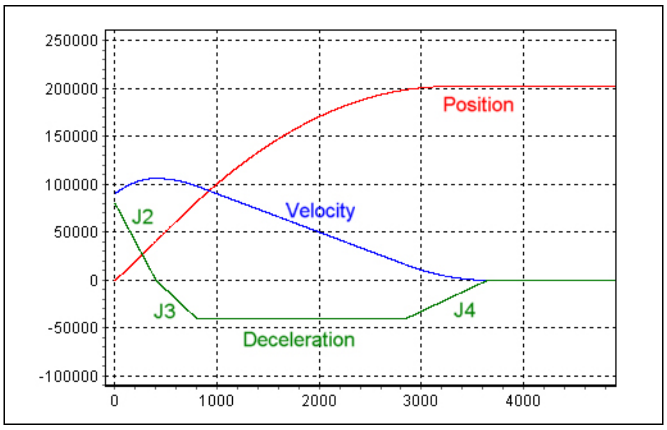
In the following examples, maximum acceleration has been set to a value higher than maximum deceleration so the motor can speed up faster than it can slow down. As well, JERKMIN has been set smaller than JERKMIN2, JERKMIN2 smaller than JERKMIN3, and JERKMIN3 smaller than JERKMIN4. This was done so that 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).

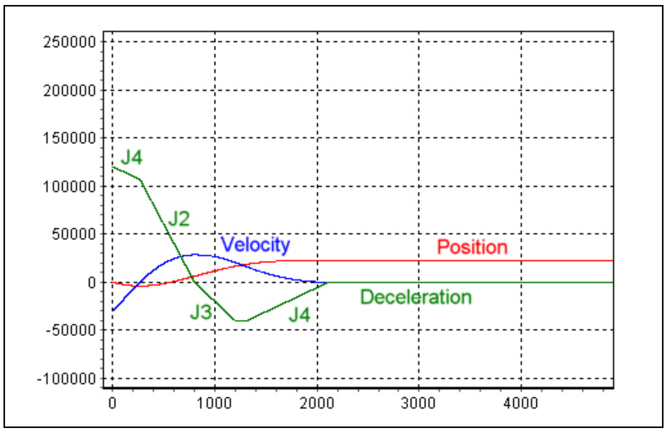


This chart shows a stopping movement that begins from a positive velocity and a positive acceleration. Since the initial acceleration is positive, the curve must start with an acceleration ramp-down segment to 0 acceleration (using JERKMIN2). This will then be followed by a deceleration ramp-up segment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



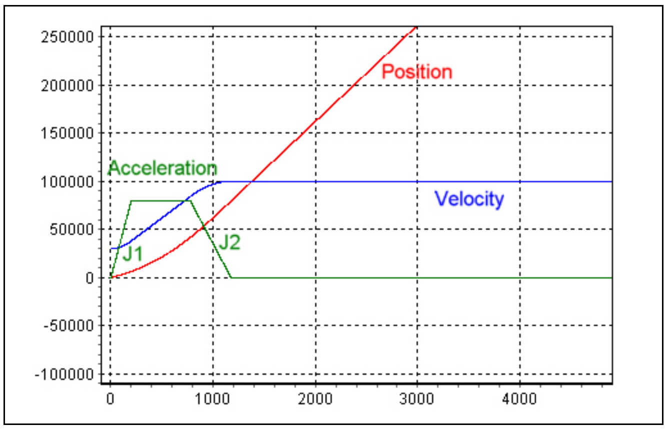
The chart below shows a stopping movement that begins from a *negative* velocity and a very high deceleration. (It is a deceleration because the speed is slowing down.) However, because the initial deceleration is so high, the motor is unable to stop without overshooting 0 velocity and 'coming back'.

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).

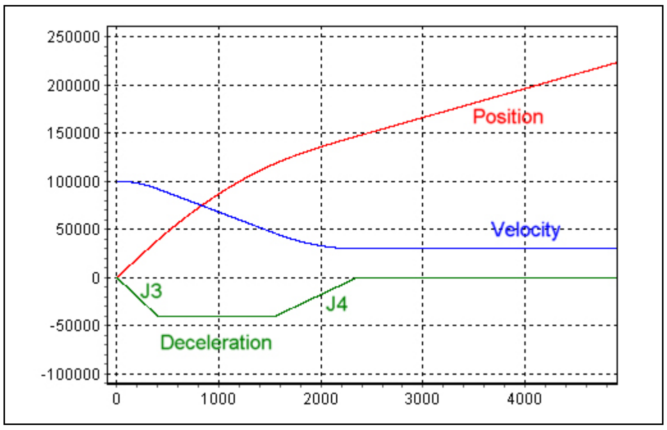


Changing to a Constant Velocity

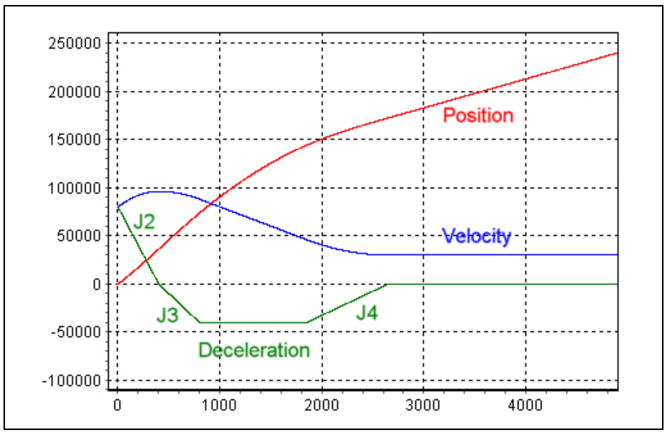
This chart shows a movement that begins from a positive constant velocity and increases its speed to a higher positive constant velocity. This curve consists of an acceleration ramp-up segment (using JERKMIN), followed by a constant acceleration segment (at maximum acceleration), and finally an acceleration 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.



Next 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.

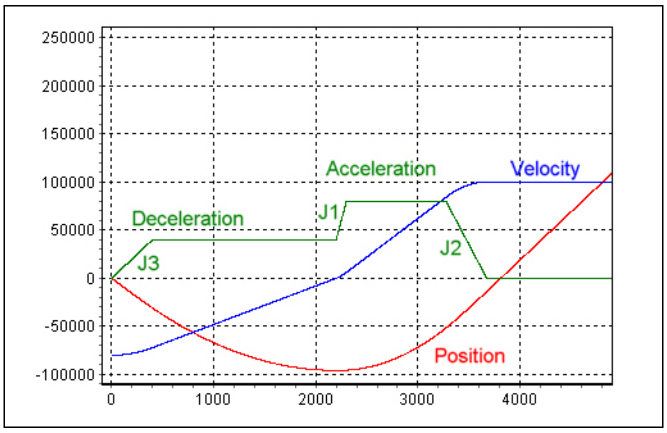


Next chart is similar to the previous example except that it begins with a positive acceleration. In this case, the curve must start with an acceleration ramp-down segment (using JERKMIN2). Once the acceleration reaches 0, then the curve can continue as before.



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).

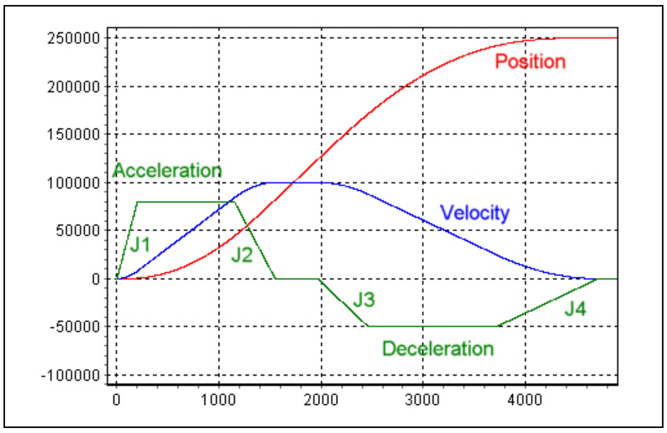


Moving to a Specified Position

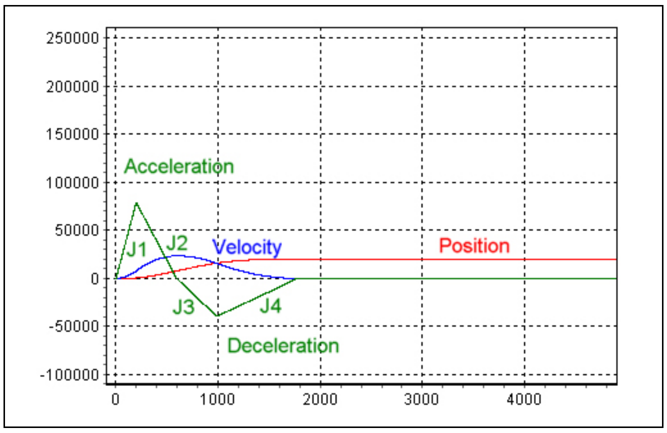
The chart below shows a 'normal' movement from being stopped at one position and then moving forward to stop at another position. The curve starts with an acceleration ramp to maximum velocity. This portion of the curve will be similar to the first of the examples shown in "Changing to a Constant Velocity".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



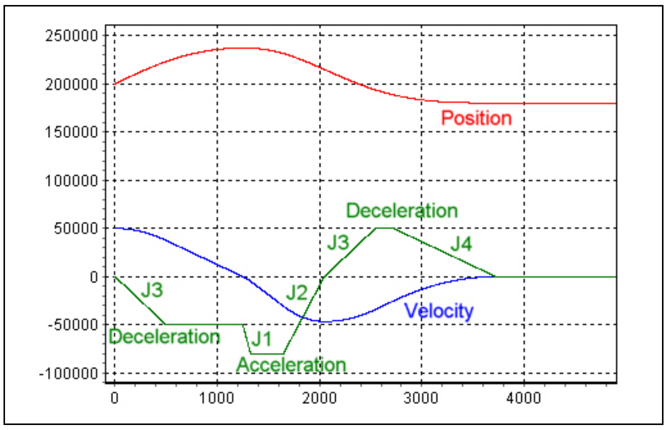
Next chart shows a typical 'short' movement where maximum velocity cannot be reached. In this case, the curve ramps-up the acceleration (using JERKMIN) for as long as possible. This may or may not reach maximum acceleration, depending on how far away the target position is. There will then be an acceleration 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.



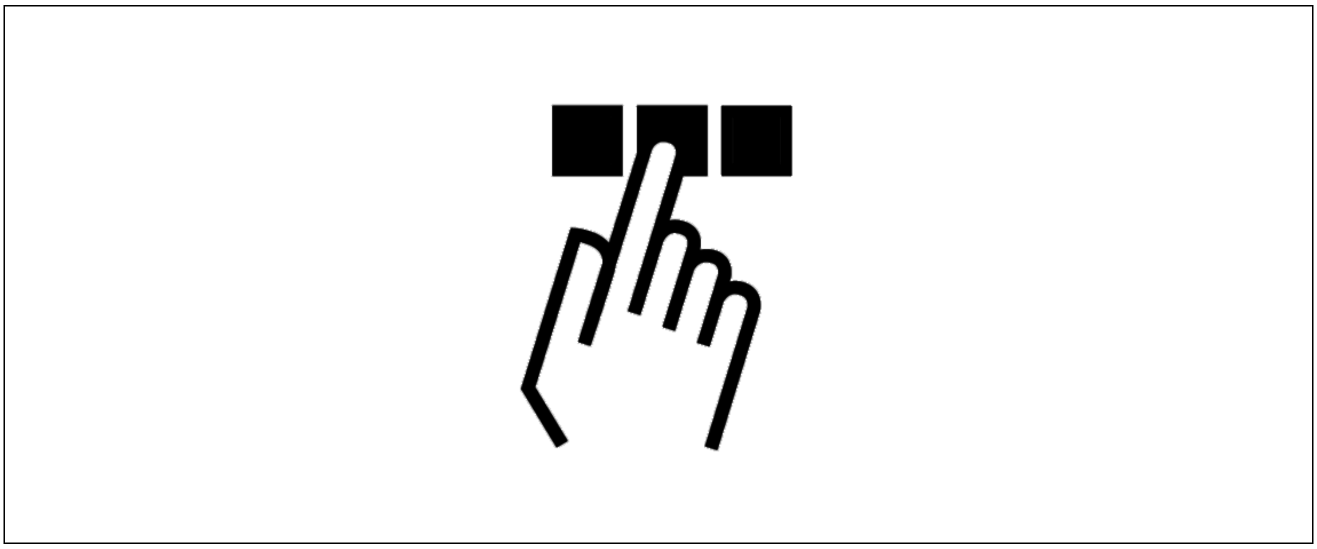
The chart below shows an example where the motor is initially moving in the "wrong" direction and must turn around and 'go back' to the target position. Since it must "turn around", the curve starts with a deceleration ramp-up segment (using JERKMIN3) to maximum deceleration.

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 <u>other</u> 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.

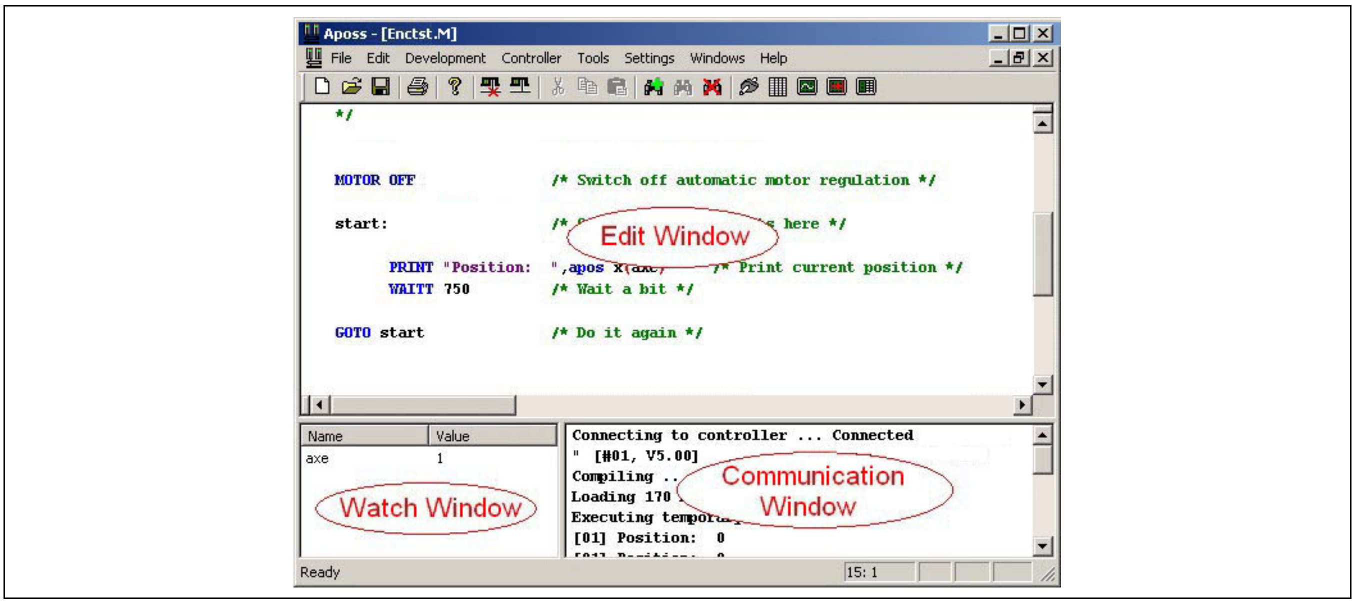
The selection of operating mode is done by MCT 10 at start-up of APOSS and can not be changed while APOSS is running.

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

The APOSS Window allows simultaneous access both to an APOSS program and to a controller. Multiple APOSS windows can be opened, each accessing a different APOSS program and a different controller.



```
    */


    MOTOR OFF                /* Switch off automatic motor regulation */

    start:                   /*                        here */

           PRINT "Position:  ",apos x                    /* Print current position */
           WAITT 750         /* Wait a bit */

    GOTO start               /* Do it again */
```

Name	Value
axe	1

```
Connecting to controller ... Connected
"  [#01, V5.00]
Compiling ..
Loading 170
Executing tempor
[01] Position:   0
```

The *Edit Window* displays the APOSS program being edited and provides all the usual text editing features. Different colors are used to distinguish between comments, program sections, operators, numbers, etc. You can alter the colors using *Settings* → *Editor*.

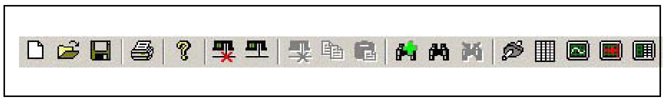
The *Communication Window* displays both messages from the APOSS-IDE (e.g. compiler messages) and messages from the controller (e.g. programmed PRINT commands). Messages from the controller will be prefixed by the controller ID number (e.g. "[01]" in the above example).

The *Watch Window* is useful during debugging. It allows the user to "watch" various aspects of both the controller and the executing program while the controller is operating. See *Development* → *Watch Add* for a description of how to use the Watch Window.

Title and Tool Bar

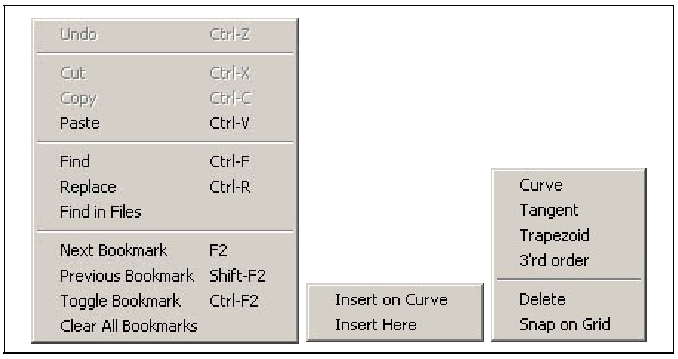
The *Title bar* displays the name of the APOSS program. If a controller is connected, then the controller ID number and the connection interface type will also be displayed. In the event of a controller error, the error is also shown in the title bar.

The *Tool bar* offers "single-click" access to several commonly used functions. These functions are described later in the manual.



Popup Menus

Popup menus are provided at certain program locations if the right mouse button is clicked. For example, the context menu in the Edit Window or a selection menu in *Tools → CAM Editor* for inserting or deleting Fix Points. The popup menus are closed automatically when the selected function is executed or if any other location on the screen is clicked with the left mouse button.



Function Keys

Frequently used functions are assigned to function keys:

[F1]	Access on-line help
[F2]	Skip to the next bookmark
[F3]	Find next (if "Find" has been used)
[F4]	Check the syntax of the program
[F5]	Execute the program (see also *Execute*)
[F9]	Single-step through the program (debug mode only)
[F11]	Access on-line help for system process information
[F12]	Open the "Command List" for simplified programming

The function keys will be described later in this chapter where appropriate.

[Esc] key

In standard Windows applications, the [Esc] key normally closes the currently active window. However, in the APOSS Window, the [Esc] key will automatically open a connection to the default controller if no controller has yet been connected. If a controller has already been connected, then the [Esc] key will abort any program currently executing on the controller.

NB!:
If an executing program is aborted while the drive is rotating, then the drive will slow down with the maximum allowed deceleration.

Reconnect a Lost Connection

When an active connection to a controller is lost (for example, if the controller is powered off or the communication line is disconnected), then the APOSS Window connected to that controller will display a "Lost connection" message in its title bar. However, the window will remain "associated" with that controller. Pressing [Esc] (or executing any command that needs to talk to the controller) will cause the APOSS Window to attempt to reconnect to the same controller. This behavior becomes relevant only when there are multiple controllers available on the communication line.

▫ The Edit Window

The Edit Window displays the current program being edited and allows it to be changed. Many functions are provided through the mouse and through keyboard shortcuts in order to assist the user in the editing process. These are described below.

Mouse functions

The Edit Window supports the following functions using the mouse buttons.

L-click	Change the cursor position and clear any current text selection.
R-click	Display the popup edit menu.
L-double click	Select the word under cursor.
L-down and drag	Select text.
[Alt]-L-down and drag	Select a column of text.
L-down on selection and drag	Move the selected text.
[Ctrl]-L-down on selection and drag	Copy the selected text.
L-click in left margin	Select the entire line.
L-down in left margin and drag	Select multiple lines.
Spin wheel	Scroll window vertically.
Single-click wheel	Select the word under cursor.
Double-click wheel	Select the entire line.
L-down on splitter and drag	Split the window into multiple views or adjust the current splitter view.
L-double click on splitter	Split the window in half or un-split the window if already split.

Keyboard functions

The Edit Window supports the following functions using keyboard shortcuts. Note that many of these functions are only available using keyboard shortcuts.

NB!:
Please note that some keyboard functions are vendor dependent and may not perform the desired function. If this is the case, please ask your system administrator.

Go to	
[Home]	Go to start of line.
[End]	Go to end of line.
[Ctrl]-[Home]	Go to start of program.
[Ctrl]-[End]	Go to end of program.
[Ctrl]-[←]	Go to start of word.
[Ctrl]-[ ]	Go to end of word.
[Ctrl]-[Alt]-[ ]	Go to start of next blank line.
[Ctrl]-[Alt]-[←]	Go to end of previous blank line.
[Ctrl]-[G]	Go to line (opens dialog).
[Ctrl]-[B]	Go to matching brace ("{([" or "})]").
[F2]	Go to next bookmark.
[Shift]-[F2]	Go to previous bookmark.
[Ctrl]-[F2]	Toggle bookmark on current line.

Text Selection	
[Shift]-[←]	Extend selection left.
[Shift]-[ ]	Extend selection right.
[Shift]-[↑]	Extend selection up.
[Shift]-[↓]	Extend selection down.
[Ctrl]-[Shift]-[←]	Extend selection to start of word.
[Ctrl]-[Shift]-[ ]	Extend selection to end of word.
[Shift]-[Home]	Extend selection to start of line.
[Shift]-[End]	Extend selection to end of line.
[Shift]-[PageUp]	Extend selection up one page.
[Shift]-[PageDown]	Extend selection down one page.
[Ctrl]-[Shift]-[Home]	Extend selection to start of program.
[Ctrl]-[Shift]-[End]	Extend selection to end of program.
[Ctrl]-[Alt]-[F8]	Select line under cursor.
Cut / Copy / Paste	
[Ctrl]-[C]	Copy selection to clipboard.
[Ctrl]-[Insert]	Copy selection to clipboard.
[Shift]-[Delete]	Cut selection to clipboard.
[Ctrl]-[X]	Cut selection to clipboard.
[Ctrl]-[Y]	Cut line to clipboard.
[Ctrl]-[V]	Paste from clipboard.
[Shift]-[Insert]	Paste from clipboard.
[Ctrl]-[Alt]-[K]	Cut all lines from previous blank line to next blank line to clipboard.
Find / Replace	
[Alt]-[F3]	Find.
[Ctrl]-[F]	Find.
[F3]	Find next.
[Shift]-[F3]	Find previous
[Ctrl]-[F3]	Find next word under cursor.
[Ctrl]-[Shift]-[F3]	Find previous word under cursor.
[Ctrl]-[R]	Find and replace.
Modify	
[Insert]	Toggle typing between "insert" and "overwrite".
[Ctrl]-[Z]	Undo last change.
[Alt]-[Backspace]	Undo last change.
[Tab] (with selection)	Indent selected lines.
[Shift]-[Tab]	Un-indent selected lines.
[Ctrl]-[Backspace]	Delete to start of word.
[Ctrl]-[Delete]	Delete to end of word.
[Ctrl]-[U]	Make selection lowercase.
[Ctrl]-[Shift]-[U]	Make selection uppercase.
[Ctrl]-[Shift]-[N]	Insert a new line above the current line.
[Ctrl]-[Alt]-[R]	Repeat the next command multiple times (opens dialog).
[Ctrl]-[Shift]-[R]	Start recording a macro.
Scroll Window	
[Ctrl]-[↑]	Scroll window up.
[Ctrl]-[↓]	Scroll window down.
[Ctrl]-[PageUp]	Scroll window left.
[Ctrl]-[PageDown]	Scroll window right.

Undo Function

You can use [Alt]-[Backspace], or [Ctrl]-[Z], or *Edit* → *Undo* to undo the last action in the Edit Window.

NB!:
Note that both *File* → *Save* and *File* → *Save* as clear the Undo memory.

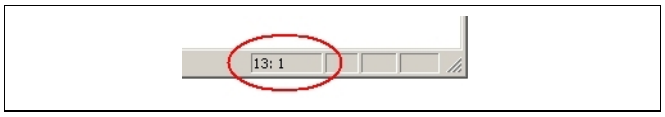
Tabs

Use tabs and indentation to visually structure your program. The tab size can be set in *Settings* → *Editor.*

Line numbers

Within your program you can use the line numbers for orientation purposes. For example, the "Syntax Check" command not only places the cursor at the first line corresponding to an error, but also displays the line numbers containing errors in the Communication Window.

The current line number can be found in the status bar at the bottom right of the APOSS Window as is shown below. For example "13:1" means that the cursor is located on line 13 at position 1 (i.e. before the first character).



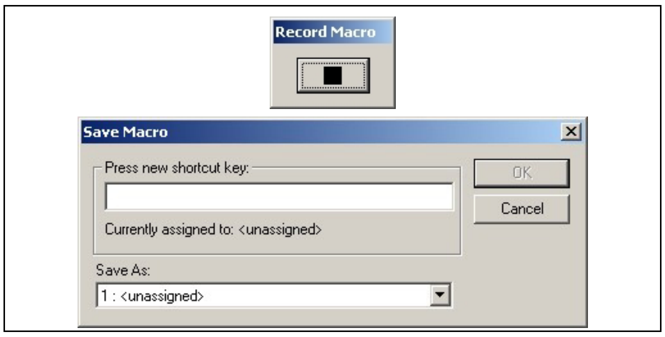
Recording and Executing Macros

Frequently used commands or command chains can be recorded as "macros" and then assigned to keyboard shortcuts. The shortcuts can then be used to repeat the commands whenever required. Up to 10 different macros can be defined. These macros are saved and restored the next time that the APOS-IDE is started.

To start recording a macro, press [Ctrl]-[Shift]-[R]. The following "Record Macro" dialog will indicate that the recording has started. Then type whatever you want recorded in the macro. This can include normal keyboard keys, keyboard shortcuts, and menu commands.

To end the recording, click on the black button in the "Record Macro" dialog.

This will display the dialog "Save Macro". Press the keyboard shortcut that you want to assign to the macro (e.g. [Ctrl]-[Shift]-[M]). Each shortcut can be up to two characters in length. Then select one of the available "Save As" alternatives.



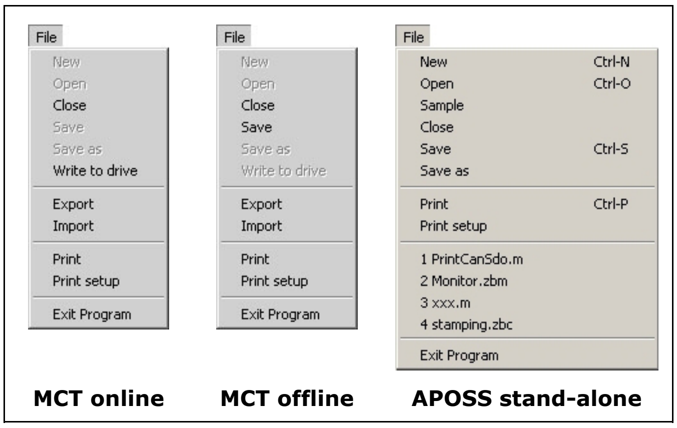
NB!:
When specifying the macro shortcut, almost any keyboard key or shortcut can be used. However, be careful not to use a keyboard key or shortcut that already has an existing meaning. Doing so will cause the original meaning to be lost and in some cases cause unpredictable results. For example, using the [↑] key will make it impossible to use that key anymore to move the cursor! Using the [Alt] key (e.g. [Alt]-[A], [Alt]-[X], etc.) is a good option since most of these will not conflict with previously defined shortcuts.

All currently defined macros can be cleared using *Edit* → *Clear Macros*.

☐ File Menu

The *File* menu contains all the commands necessary to create, open, save and print programs. *File → Close, Save, Save as, Print*, and *Print setup* are used in the usual way.

In depend on the operating mode – MCT online, MCT offline or APOSS stand-alone – different functions are enabled:



File → New

New files must be created via MCT10 or in APOSS stand-alone with *File → New*.

An APOSS Window is opened with the name “Untitled” and you can begin to write your program.

File → Open

Select the file via the MCT10. APOSS and with that the file is opened automatically. In APOSS stand-alone use *File → Open*.

File → Save as

Please use the features of the MCT 10 to rename a program file (*.m) or to copy it. Or use → *Save as* in APOSS stand-alone.

APOSS program files always have the extension “.m”.

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.

☐ Sample

APOSS contains several sample programs. Any of these sample programs may be freely used, modified, or incorporated into other programs by the user. *File → Sample* can be used to edit any of these sample programs. Please see Program Samples for a list of the available sample programs.

☐ Exit Program

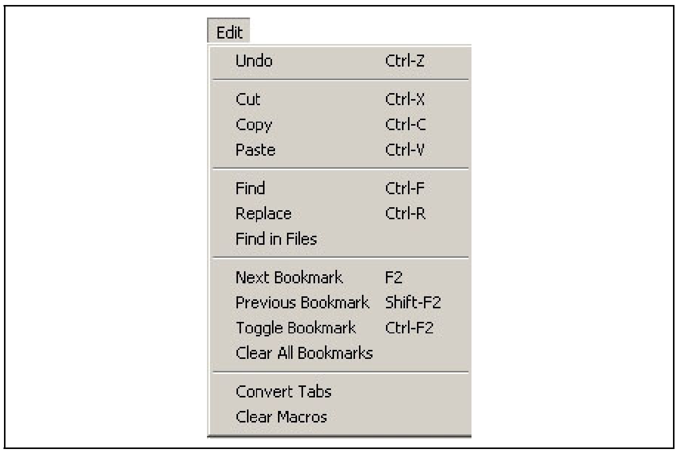
The APOSS application can be ended by clicking on *Exit Program* or on the ☒ icon in the far upper right corner of the window. If you have not yet saved a new file, or saved changes to an old file, then you will have the chance to do this.

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

NB!:
Disconnecting from very early versions of controllers (for example, as a result of *File → Exit Program*), may cause the controller to stop executing if the controller is using PRINT commands to send messages to the PC. This happens once the controller's internal print buffer becomes full.
This is not a problem for all newer controllers. Newer controllers will continue to execute even after the print buffer becomes full; print messages will simple be discarded once the buffer is full.

□ 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.



□ Find and Replace

Click on *Edit* → *Find* or press [Ctrl]-[F] and enter the search string into the following dialog field. Use [F3] to jump from one instance of the string to the next.

Click on → *Mark All* instead of → *Find* and all instances found are immediately "bookmarked" with a blue triangle in the left margin. [F2] can then be used to jump to successive bookmarks.

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

Wildcards	? for any single character + for one or more or 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	Sub expressions may be OR-ed together with the | pipe symbol.
Parenthesized sub expressions	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.

Examples:

10	Search for the string "10".
10+	Search for "1" followed by at least one "0" (e.g. 10, 100, 1000, etc.).
10*	Search for "1" followed by zero or more "0" characters (e.g. 1, 10, 100, 1000, etc.).
vel[xyz]	Search for "velx", "vely", or "velz".
vel[1-3]	Search for "vel1", "vel2", or "vel3".
(vel)|(acc)	Search for "vel" or "acc".
vel[\t]*=	Search for "vel", followed by any number of spaces or tabs, followed by "=" (i.e. search for assignments to the variable "vel").

□ Find in Files

The Find in Files function will display a dialog allowing the user to search for strings within files. All occurrences of the string which are found will be displayed in the dialog. Double-clicking on any occurrence in the list will open that file and position the cursor on that line. Note that Find in Files uses a "disk-based" search. It will only find occurrences of a string within files saved to disk. Any unsaved changes to files currently being edited, will not be found by Find in Files.

□ Bookmarks

Bookmarks allow the user to flag particular lines that are of interest to him and then quickly jump between them.

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

<u>Next Bookmark [F2]</u>

If bookmarks have been used in the editor, then this function or [F2] will scroll the program and position the edit cursor on the next line that contains a bookmark.

<u>Previous Bookmark</u>

If bookmarks have been used in the editor, then this function or [Shift]**+**[F2] will scroll the program and position the edit cursor on the previous line that contains a bookmark.

<u>Toggle Bookmark</u>

This function or [Ctrl]**-**[F2] will toggle the bookmark on the current line. If the line contains no bookmark, then a bookmark is placed on the line. If the line already contains a bookmark, then the bookmark is removed from the line.

<u>Clear All Bookmarks</u>

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

□ Convert Tabs

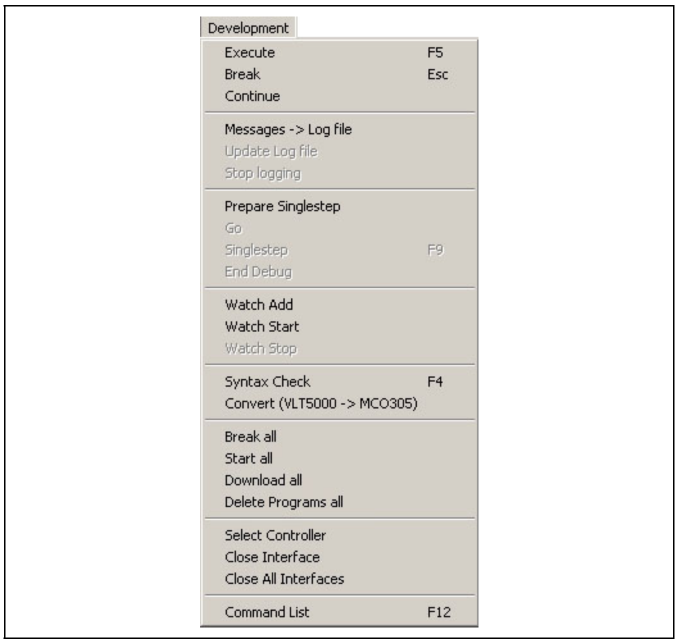
Clicking on this menu item will replace tab characters in the file currently being edited, with space characters.

The tab spacing set in the *Settings* → *Editor* dialog is used.

□ Clear Macros

This menu item will clear all editor macros defined using the [Ctrl]**-**[Shift]-[R] keyboard shortcut. Please see Recording and Executing Macros for a description of editor macros.

□ Development Menu



This menu provides various functions used during the program development phase of applications. This includes functions to compile, execute, break, and debug programs. It also includes functions that allow controllers to be connected and disconnected.

Many of these functions require a controller or the FC 300 to be connected before the function can be executed. A controller can be connected either by pressing [Esc] to connect to the default controller or by using *Development → Select Controller* to connect to a specific controller if more than one controller is present. If no controller has yet been connected when a function is executed, then most of these functions will automatically connect to the default controller.

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.

For information on debugging programs, please see Debugging Programs at the end of this chapter.

□ Execute [F5]

This will execute the program currently being edited. This involves the following steps:

1. If a controller is not currently open and connected to the APOSS Window, then an attempt is made to connect to the currently defined default controller. If no controller is connected, then the *Execute* is aborted.
2. A check is made to see if the controller is already executing a program. If so, then the user is asked if the existing program can be stopped. The controller can only execute one program at a time. If the controller is not "idle", then the *Execute* is aborted.
3. If the user has made changes to the program being edited, then the changes are automatically saved to the PC disk. If this is a "new" program, then the user is requested to enter a filename for the program. Note that it is not necessary to enter a filename at this time. If no filename is given, then a temporary file is used.
4. The program being edited is then "compiled". This produces a "machine code" version of the program that is customized for that particular controller. Note that this "machine code" is not human-readable and cannot be edited. If the compile fails for any reason, then the *Execute* is aborted.
5. The "machine code" (i.e. not the original text file being edited) is then downloaded to the controller and placed in the controller's temporary memory. Note that this temporary memory will be lost if the controller is powered off. To save a program permanently in the controller, use *Controller → Programs.*
6. After the download completes, the controller will start to execute the program.

Each subsequent time that *Execute* is used, the previously downloaded program will be overwritten with the newly downloaded program. This allows you to quickly and easily make changes and re-test during debugging.

<u>Run Programs in Several Controllers</u>

You can execute different programs on different controllers all at the same time simply by opening each program in a different APOSS Window. Then use *Development → Select Controller* in each window to select the desired controller for that program. Finally, use *Development → Execute* in each window to download and start the programs.

Note that you must use *Select* to open the controllers before using *Execute* since *Execute* will always connect to the default controller if no controller is yet connected.

Each APOSS Window can be connected to only one controller at a time and each APOSS program can be edited by only one APOSS Window at a time. Hence, if you want to execute the same program on several controllers, then use one of the following methods:

1. Use *Development* → *Select Controller* to select and connect to the first controller. Then use *Development* → *Execute* to download and start the program executing on the controller. Once the program is executing, then use *Development* → *Select Controller* a second time to select and connect to the second controller. This will disconnect from the first controller but leave the program executing. Then use *Development* → *Execute* to download and start the program on the second controller. Repeat the process for as many controllers as desired.
2. If the program is to be started on several controllers that all share the same network connection, (e.g. in a field bus) then *Development* → *Download all* can be used. This allows the program to be compiled, downloaded, saved in permanent controller memory, and started, all in a single user operation.

□ Break [Esc] and Continue

Click on *Development* → *Break* or press [Esc] in order to stop any program currently executing on the controller.

NB!:
If an executing program is aborted while the drive is rotating, then the drive will slow down with the maximum allowed deceleration.

Development → *Break all* can be used to stop the execution of multiple controllers at the same time.

Click on *Development* → *Continue*, in order to continue a program which was just aborted. In doing so any positioning 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 function can be used to start the logging of all messages displayed in the Communication Window to a file. Similarly, *Stop logging* will stop the logging of messages.

Messages are cached and may not always be written to the file immediately. As a result, if the file is edited or copied while logging is in progress, then the most recent messages will not always be included. To avoid this, use the *Update Log file* command immediately before editing or copying the file. This will force all messages cached up until this point to be written to the file.

Note that *Stop logging* and *Update Log file* will be enabled only if logging has been started.

□ Debugging Commands

The following commands are designed to be used to assist the user in debugging newly developed programs. For detailed information about debugging and the use of these commands, please see Debugging Programs.

<u>Prepare Singlestep</u>

This command will “prepare” both the APOSS-IDE and the controller for debugging. This includes compiling the program in “debug” mode, inserting breakpoints, and downloading the program.

<u>Go</u>

This command will start execution of the program at the current program line and continue executing until the next “user breakpoint” is reached.

<u>Singlestep</u>

This command will execute a single program line and stop at the next line. The next line is not executed.

End Debug

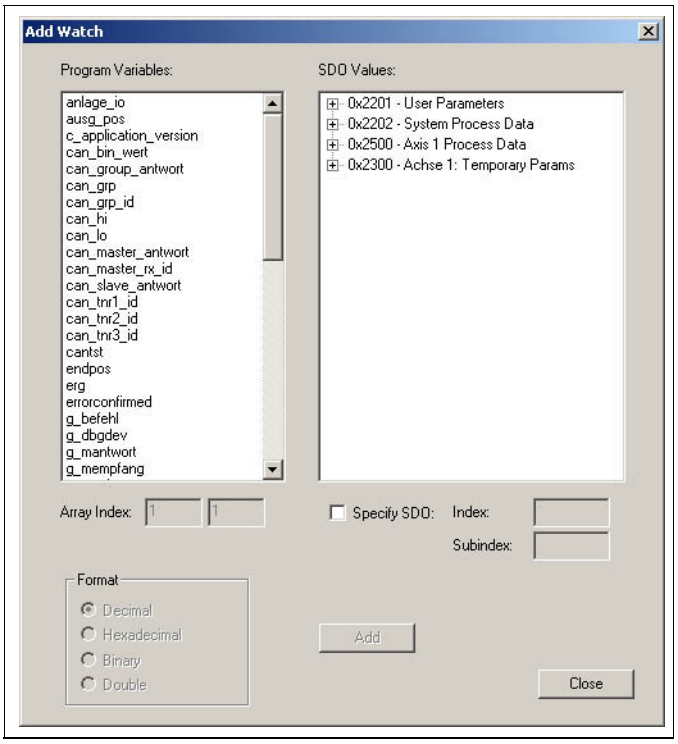
This command removes the APOSS-IDE and the controller from "debug" mode.

□ Watch Add / Start / Stop

This function enables online monitoring of the variables, arrays, system and axis processing data and axis parameters.

All values currently being watched are maintained in the Watch Window in the bottom-left corner of the APOSS Window. New values can be added to this list using *Development* → *Watch Add*. Existing values can be removed from the list by clicking on the value and then pressing the *Delete* keyboard key.

In the Watch Add dialog all program variables and arrays are shown on the left and all system values are shown (in a tree) on the right. Add a value to the Watch Window by clicking on it, selecting the format to be used to display the value, and then pressing the *Add* button. Values can also be added simply by double-clicking on the value. Multiple values can be added before closing the dialog.



```
anlage_io
ausg_pos
c_application_version
can_bin_wert
can_group_antwort
can_grp
can_grp_id
can_hi
can_lo
can_master_antwort
can_master_rx_id
can_slave_antwort
can_tnr1_id
can_tnr2_id
can_tnr3_id
cantst
endpos
erg
errorconfirmed
g_befehl
g_dbgdev
g_mantwort
g_mempfang
```

```
0x2201 - User Parameters
0x2202 - System Process Data
0x2500 - Axis 1 Process Data
0x2300 - Achse 1: Temporary Params
```

If an array value is being added to the Watch Window, then the array indices must be specified. These fields will be disabled if the selected value is not an array. Note that only the first 250 elements of an array can be watched.

Activate and stop the monitoring with → *Watch Start,* → *Watch Stop* or click .

NB!:
When monitoring is active, the Watch Window is updated constantly. This consumes both controller resources and network connection resources. Hence, the number of values being watched should be limited to a reasonable number (usually, not more than 10-15 values, depending on network connection speed). If more values than this are required, then the *Oscilloscope* should be used.

□ Syntax Check [F4]

Clicking on *Development* → *Syntax Check* will do a "test compile" of the program being edited. This can be a useful function to use while creating new programs or modifying existing programs. It is a quick way to find syntax errors in the program without having to download the program to the controller and execute it. If a syntax error is found, then the line number and an error description are displayed in the Communications Window and the cursor is automatically placed at the position of the error.

The *Syntax Check* produces a debug file in addition to checking the syntax. This file will have that same name as the program but with the file extension ".ad$".

◻ Compile to file

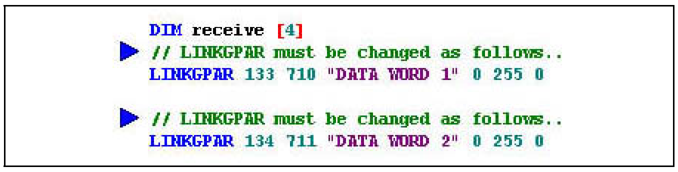
The *Execute* [F5] command will compile a program into a "machine-readable" binary version which is then downloaded to the controller and executed. The *Compile to file* command is similar except that the compiled binary version is not downloaded and executed. Instead, the binary version is saved in a ".bin" file on the PC hard drive. Binary ".bin" files can be managed using *Controller → Programs.*

When *Compile to file* is selected, a dialog will be displayed allowing the user to specify the exact controller hardware for which the program is to be compiled. Note that compiled binary versions are hardware-dependent and must be compiled for the hardware on which they will be executed. After this, a *Save As* dialog will be displayed allowing the user to select the file name to be used to save the file. The file name will default to the name of the program with ".bin" as a file extension. Only after this is the program compiled and saved on disk.

ACHTUNG!:
This feature is available only if *Binary file support in Settings → Options* is enabled.

◻ Convert VLT 5000 > 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.



```
DIM receive [4]
// LINKGPAR must be changed as follows..
LINKGPAR 133 710 "DATA WORD 1" 0 255 0

// LINKGPAR must be changed as follows..
LINKGPAR 134 711 "DATA WORD 2" 0 255 0
```

◻ Break all

If programs are executing in several controllers, then click on *Development → Break all* to abort all the executing programs. Note that this will only abort the programs running on controllers that share the same connection interface as the controller connected to this APOSS Window (e.g. multiple controllers on a field bus). Controllers connected using other interfaces are not affected.

Also note that programs will be aborted even on those controllers that are not currently connected to an APOSS Window. As long as the controller is using the same connection interface and is within the original connection ID scan range, then execution is stopped. For example, if controllers 1 and 2 are both on a field network but APOSS is currently only connected to controller 1, then Break all will still abort programs running on both controllers 1 and 2.

NB!:
If an executing program is aborted while the drive is rotating, then the drive will slow down with the maximum allowed deceleration.

◻ Start all

The *Start all* menu will send an 'Execute' command to all connected controllers that share the same connection interface as the controller connected to this APOSS Window. If you are testing software that runs on multiple controllers in a network, then this is a quick way to start all controllers at the same time.

◻ Download all

Development → Download all displays the *APOSS Download Mode* dialog. This allows the .m file currently being edited to be downloaded to multiple controllers.

Please read *Download → Programs* for more details and options.

□ Delete Programs all

Development → *Delete Programs all* will delete all programs saved on all controllers that share the same connection interface as the controller connected to this APOSS Window (e.g. multiple controllers on a field bus). Controllers connected using other interfaces are not affected.

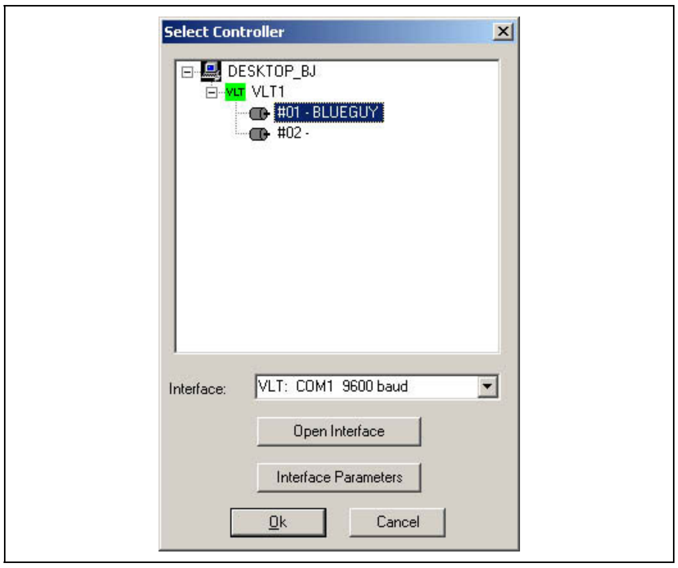
Also note that programs will be deleted even on those controllers that are not currently connected to an APOSS Window. As long as the controller is using the same connection interface and is within the original connection ID scan range, then programs will be deleted. For example, if controllers 1 and 2 are both on a field network but APOSS is currently only connected to controller 1, then *Delete Programs all* will still delete programs on both controller 1 and 2.

This command is intended to be used in conjunction with *Development* → *Download all.*

□ Select Controller

If more than one controller is accessible from the PC, then *Development* → *Select controller* can be used to select the specific controller to which the APOSS Window is to be connected. 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*. The desired interface can then be selected. Note that choosing an interface in this way does not change the default interface specified using *Settings* → *Interface*.



If a controller is already connected to the APOSS Window, then *Development* → *Select controller* can be used to switch the APOSS Window to another controller. Simply use the Select Controller dialog to select the new controller. The existing connection to the previous controller will then be closed and a connection to the new controller will be opened.

Since any specific controller can be connected to only one APOSS Window at a time, selecting a controller that is already connected to another APOSS Window will cause that controller to be switched from the other APOSS Window to this APOSS Window.

□ Close Interface / Close All Interfaces

If there is a controller currently connected in this APOSS Window, then → *Close Interface* can be used to close the connections to all controllers that share the same connection interface as this controller. A message will be displayed indicating which controller connections will be closed.

Close All Interfaces is similar to the *Close Interface* function except that all controllers on all connection interfaces are closed.

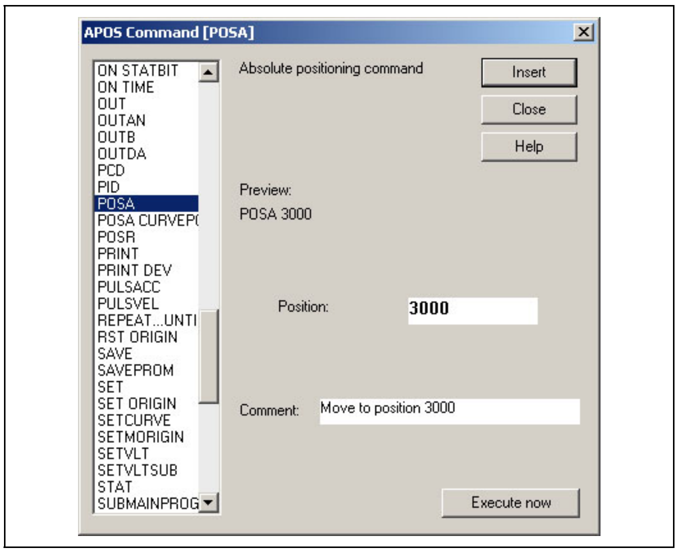
☐ Command List

The *Command List* offers a list all commands with their syntax, allowing them to be either inserted into the existing program or executed directly. This can be useful for those more infrequently used commands that may not be so well remembered by the user.

You can find detailed information on all commands simply by selecting the command and clicking on *Help* or pressing [F1].

For example POSA: Enter the position for the axis in the field. The preview shows you the exact syntax of the command.

The direct executing of a command by means of → *Execute now* is in the offline operating mode not possible.


APOS Command [POSA]

Absolute positioning command

Preview:
POSA 3000

Position: 3000

Comment: Move to position 3000

NB!:
During programming the input values will be either tested nor checked if the input range is valid. Because of the wide variety of possible applications and the large differences in performance of the motors this is either possible nor desired. Therefore, it is the responsibility of the programmer and the user to observe the performance ranges of the drive and of the system.

Insert or Execute now

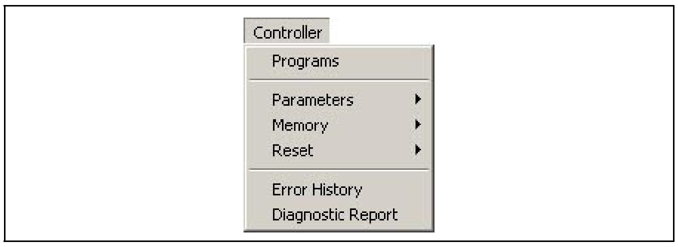
Move the text cursor (e.g. by clicking with the mouse) to the position in the Edit Window where you want to insert one or more new commands. Then click on *Development* → *Command List* and select the desired command (e.g. POSA). Enter any necessary parameter values and an optional comment and then click on → *Insert*. The completed command will then be inserted into the APOSS program at that position.

Click on → *Execute now* to test this command before inserting it into your program.

NB!:
Depending on the command, *Execute now* may cause drives to start up, stop, etc.

□ Controller Menu

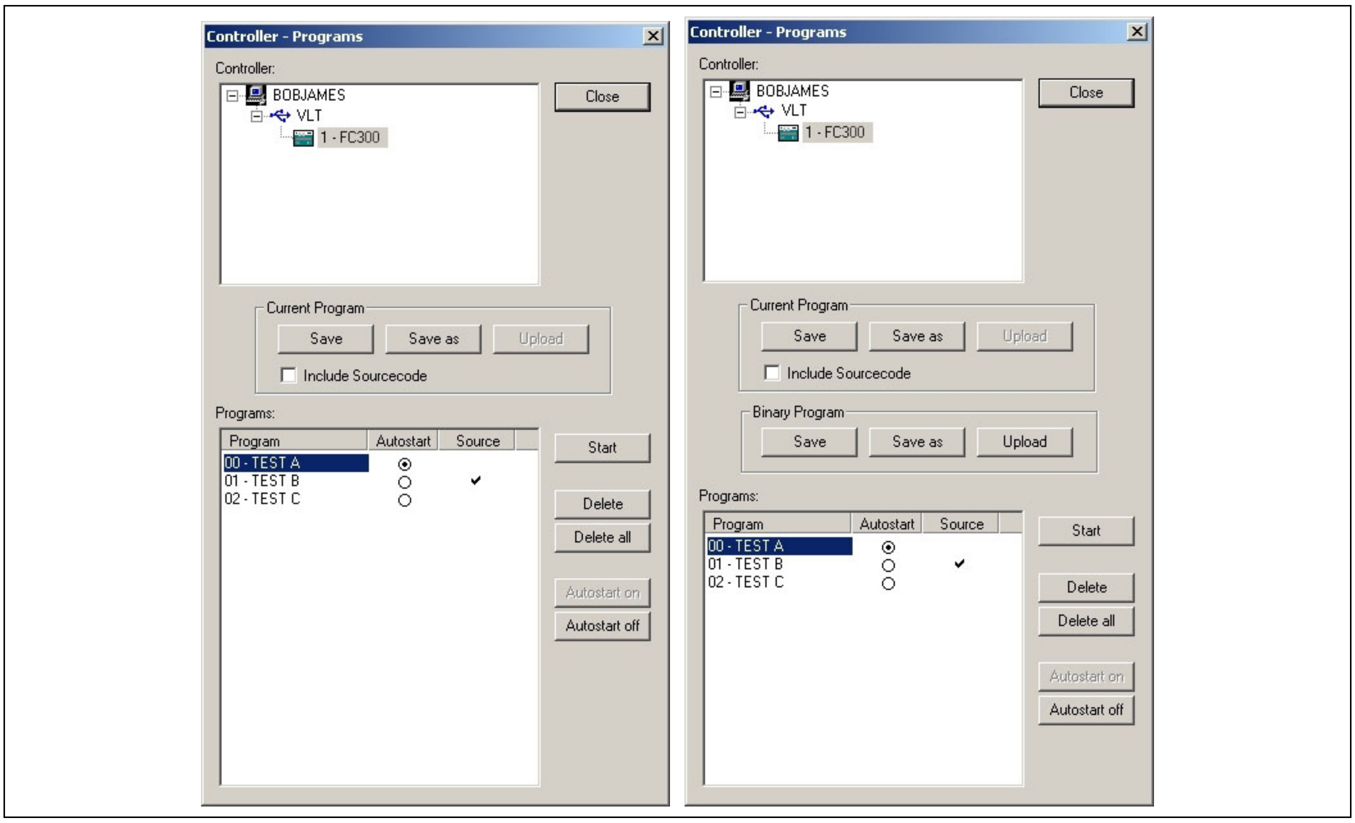
This menu provides various functions used to view and set the controller configuration and to manage various aspects of the controller's onboard memory. This includes functions to manage saved programs (and mark programs for *Autostart*), view and set global and axis parameters, and save and reset the controller EPROM memory.



□ Programs

Click on *Controller* → *Programs* to see all the programs currently stored on the controller in permanent memory. You can also select other currently connected controllers to see (and manage) the programs on those controllers.

NB!:
If "Binary file support" in *Settings* → *Options* is enabled, then the window shown on the right is displayed. Otherwise, the window on the left is displayed.



Under *Programs* is a list of all programs currently saved on the controller. Each program will be identified by a program number and a program name. Depending on available memory and the size of the saved programs, up to 91 programs may be saved at any one time. If a program has been designated as the *Autostart* program, then a ⦿ will appear in the Autostart column. If source code for a program has been included on the controller, then a checkmark will appear in the Source column.

Save a Current Program

Whenever you execute a program (i.e. with *Development* → *Execute*), it is compiled and downloaded into a temporary block of memory in the controller. This is overwritten with each subsequent *Execute* and it is lost if the controller is powered off. However, using *Controller* → *Programs*, you can permanently → *Save* the current program in the APOSS Window. The program is compiled and downloaded into a permanent block of memory in the controller. The new program will be appended to the end of the list of existing programs.

NB!:
Note that saving a program on the controller does not automatically save the original source code for the program. You should always use the normal *File* → *Save* menu command to save the program source to the PC hard drive.

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

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

Include Sourcecode

When this checkbox is enabled, the original source code for the program being saved will also be downloaded and saved on the FC 300. This source code can later be uploaded back from the controller and edited again on the PC.

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

All programs that have saved source code associated with them will have a checkmark in the Source column.

NB!:
If insufficient memory is available on the controller for saving the source code, then a message is displayed and other programs must be erased before saving the new program.

Upload a Current Program

All programs with a checkmark in the Source column have the associated source code for them saved on the controller. This source code can be uploaded back from the controller to the PC for subsequent use.

Select the desired program and click on *Current Program* → *Upload*. The default program name will contain the date and time when the original source code was saved on the controller. This is done to help avoid potential problems caused by overwriting existing files on the PC.

Save or Upload a Binary Program

These functions are displayed only if the *Binary file support* checkbox in the *Settings* → *Options* dialog is checked.

The *Save* and *Save as* buttons will display a dialog allowing a compiled .bin file to be selected. Input a program name and number and then download and save the binary file on the FC 300.

The *Upload* button will read the currently selected binary program from the FC 300.

NB!:
Motion Controllers older than MCO 5.00 do not support the uploading of binary files. In this case, the *Upload* button will be disabled.

Start Programs

Using the Programs window, you can start any saved program directly. Simply select the program to be started and click on → *Start*.

Autostart

In order to support the use of the controller in "stand-alone" or "turnkey" applications (i.e. without user input), the controller can be configured so that an APOSS program is started automatically when the controller is powered on. Typically, in these types of applications, the controller must also never be "stopped". So the controller can also be configured to restart an APOSS program in the event that the executing program terminates.

The first part of this mechanism is the "Autostart program". The Autostart program (if one is defined) is normally <u>always</u> started automatically when the controller is powered on. If no Autostart program has been defined, then <u>no</u> program is started when the controller is powered on. Note that when the controller is powered on, it will run various self-test checks. If any of these fail, then the controller will not attempt to start the Autostart program.

The second part of the Autostart mechanism is the "Restart program". With some exceptions, the Restart program is started automatically when the currently executing program terminates. In particular, this includes the termination of the Autostart program.

NB!:
If the Autostart program, Restart program, or any other program <u>is aborted by the user</u>, then the Autostart mechanism will be <u>deactivated</u>. The controller will not automatically start the Autostart program or any Restart program again until the controller is powered off and on.

NB!:
Note that a program designated as an Autostart program can always be <u>manually started</u> by the user. However, in this case, the program will execute as a normal program, <u>not</u> as an Autostart program, and no Restart program will be started when the program terminates.

<u>Autostart program</u>

The controller determines the Autostart program as follows:

1. If a program has been explicitly marked as the "Autostart program" (see below), then this program will be started when the controller is powered on.
2. Otherwise, if par. 33-5* *I_Function_n_13 or 14* is set to a digital input pin, then 33-5* *I_Function_n_15* will determine the program number of the Autostart program. The controller will wait until the input pin is activated and the Autostart program will then be started.
   Note that the pin may <u>already</u> be activated when the controller is powered on. These parameters are designed to be used when the Autostart program is being selected by some external source (such as a PLC).
3. Otherwise, there is no Autostart program.

Any saved program can be designated as the Autostart program by selecting it and clicking → *Autostart on*. The Autostart program will be marked with a ⦿ in the Autostart column. The Autostart designation can be cleared again simply by selecting the program and clicking → *Autostart off*. Only one program at a time can be designated as the Autostart program. So if a second program is chosen as the Autostart program, then the Autostart designation is cleared from the previous program.

One of the main purposes of an Autostart program is to select which of multiple programs is to be executed in applications that require different programs in different situations. The Autostart program will identify (usually based either on configuration parameters or on digital input settings) which program is to be executed. The Autostart program then designates that program as the Restart program and terminates. The controller will then automatically start the designated program.

NB!:
The *Start program* mechanism (33-5* *I_Function_n_13 or 14*) does not function properly in some Motion Controllers prior to MCO 5.00. If this affects the application, then please contact your supplier for updated firmware.

Restart program

If the controller executes an Autostart program when it is powered on, then the controller's Autostart mechanism will be activated. When the Autostart mechanism is active, then the controller will automatically start the designated Restart program whenever the currently executing program terminates. This will continue until the Autostart mechanism is deactivated.

The Autostart mechanism will be deactivated if any of the following happen:

1. The user has executed a "break" (i.e. [Esc]) command.
2. A program has terminated with an error (other than one of the errors listed below).

Once deactivated, the Autostart mechanism cannot be reactivated again with powering the controller off and on again.

The Autostart mechanism will not be deactivated if any of the following errors occur:

1. Position error (error 108)
2. Limit switch error (error 125)
3. Software limit switch error (error 111)

The controller determines which program is the Restart program as follows:

1. If 33-80 *Activated Program Number* is set (i.e. not -1), then this is the program number of the Restart program.
2. Otherwise, if 33-5* *I_Function_n_13 or 14* is set to a digital input pin, then the controller will wait until that input pin is activated. Once activated, then 33-5* *I_Function_n_15* will determine the program number of the Restart program.
   Note that the pin may not already be activated when the currently executing program terminates; activation is triggered only on a rising signal.
3. Otherwise, if the Autostart program is the only program to have been executed, then the Autostart program will be the Restart program (i.e. the Autostart program will be executed repeatedly).
4. Otherwise, there is no Restart program and the controller will not Autostart any program.

The par. 33-80 *Activated Program Number* is designed to be used for "linking" programs together. The currently executing program simply needs to set *Activated Program Number* to specify which program is to be executed next. Note that if *Activated Program Number* is not reset, then the current program will be executed repeatedly.

The parameters 33-5* I*_Function Start program* and *Program select* are designed to be used when the next program to be executed is being selected by some external source (such as a PLC).

NB!:
Please note that the originally designated Autostart program is restarted only if 33-80 *Activated Program Number* and 33-5* I*_Function Start program* have not been used. If either *Activated Program Number* or *Start program* have been used, then the Autostart program will only be executed once when the controller is powered on. This "one-time" execution can be useful for executing functions such as HOME. Note that the PRGPAR parameter can still be used to explicitly restart the Autostart program.

NB!:
The *Start program* mechanism (33-5* *I_Function_n_13 or 14*) does not function properly in some Motion Controllers prior to MCO 5.00. If this affects the application, then please contact your supplier for updated firmware.

Delete programs

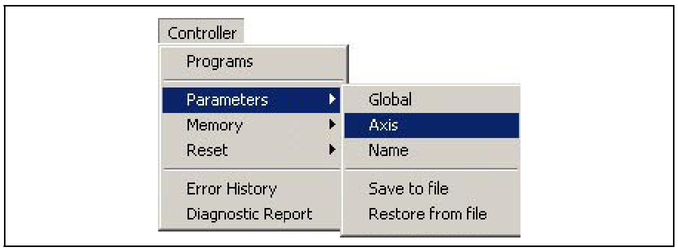
Select any program and click on → *Delete* if you want to delete that individual program from the controller. This permanently deletes the program from the controller so you should ensure that you have previously saved the program on the PC.

Click on → *Delete all* if you want to delete all the programs in the controller. This permanently deletes the programs from the controller so you should ensure that you have previously saved the programs on the PC.

□ 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 text cursor is in one of the input fields and information about the corresponding parameter will be displayed.

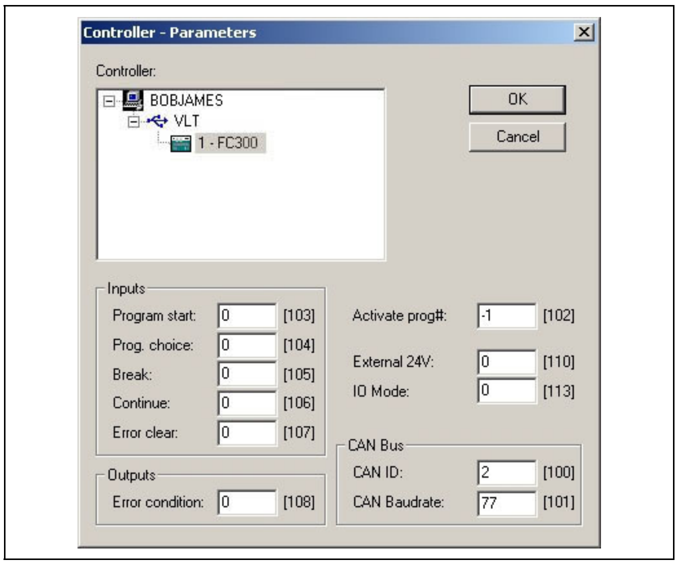


Parameters > Global

The global parameters include the functions of the inputs and outputs and the standard parameters which are group in 33-5* and 33-8*.

Select the controller you wish to edit. Then change the desired parameter values. Click on *OK* to load the changes into the controller.

NB!:
Note that you can select a different controller after changing values but before clicking *OK*. If you do this, then the changed values are written to the previous controller and the new controller's values are read and displayed in the fields.



Inputs
Program start: 0 [103]
Prog. choice: 0 [104]
Break: 0 [105]
Continue: 0 [106]
Error clear: 0 [107]

Activate prog#: -1 [102]
External 24V: 0 [110]
IO Mode: 0 [113]

Outputs
Error condition: 0 [108]

CAN Bus
CAN ID: 2 [100]
CAN Baudrate: 77 [101]

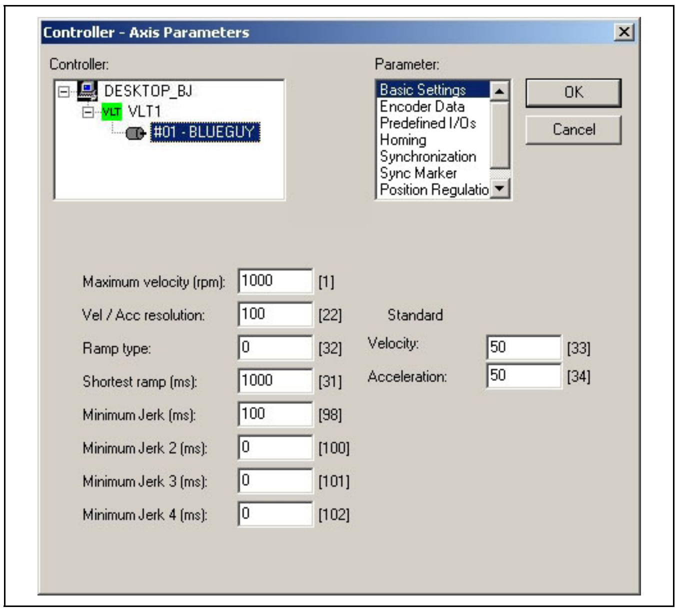
Global Parameters dialog box.

Parameters > Axis

Select the controller you wish to edit and, select the group of axis parameters from the following list:

– Basic Settings	– par. Group 32-8*
– Encoder Data	– par. Group 32-0*, 32-3*
– Predefined I/Os	– par. Group 33-4*, 33-5*
– Homing	– par. Group 33-0*
– Synchronization	– par. Group 33-1*
– Sync Marker	– par. Group 33-1*
– Position Regulation	– par. Group 32-6*

Change the desired parameter values. You can select a different parameter group and change those parameters as well. Then click on *OK* to load the changes into the FC 300.



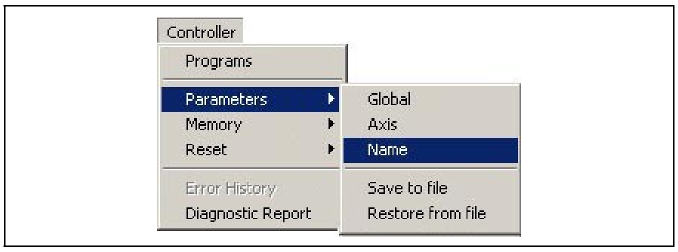
Maximum velocity (rpm): 1000 [1]
Vel / Acc resolution: 100 [22]
Ramp type: 0 [32]
Shortest ramp (ms): 1000 [31]
Minimum Jerk (ms): 100 [98]
Minimum Jerk 2 (ms): 0 [100]
Minimum Jerk 3 (ms): 0 [101]
Minimum Jerk 4 (ms): 0 [102]

Standard
Velocity: 50 [33]
Acceleration: 50 [34]

NB!:
Note that you can select a different axis after changing values but before clicking *OK*. If you do this, then the changed values for that axis are written to the controller and the values for the new axis are read and displayed in the fields. The same is true if you select a different controller: The current values are written and then the new values are read and displayed.

Parameters > Name

This function allows you to assign a name to a controller or to change the existing name of a controller. The name is displayed in various dialogs in the APOSS user interface, allowing you to more easily differentiate between multiple controllers.



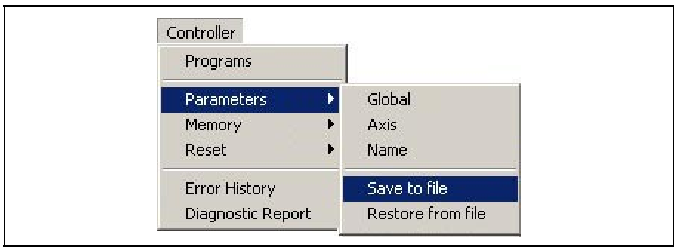
Click on *Controller* → *Parameters* → *Name* and select the controller that you want to name. Enter a name in the field and click on *OK*. Names cannot be longer than 8 characters and must be uppercase.

NB!:
Note that you can select a different controller after changing the name but before clicking *OK*.
If you do this, then the changed name is written to the previous controller and the new controller's name is read and displayed.

Parameters > Save to file

For backup purposes, the entire controller configuration (i.e. global parameters, axis parameters, user parameters, and arrays) can be uploaded and saved as a file on the PC.

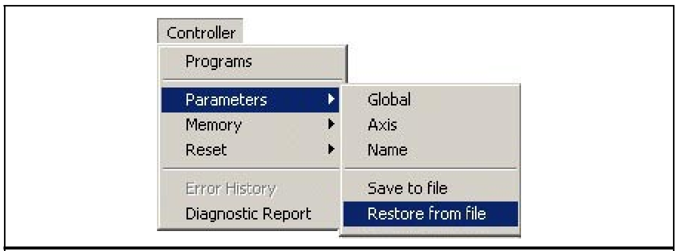
This file can then be downloaded again to the controller at a later time if it becomes necessary to restore the configuration



Select the controller, click on → *Save*, and then enter a name in the subsequent "Save As" dialog. If the parameters of multiple controllers are to be saved, then simply select another controller and → *Save* again.

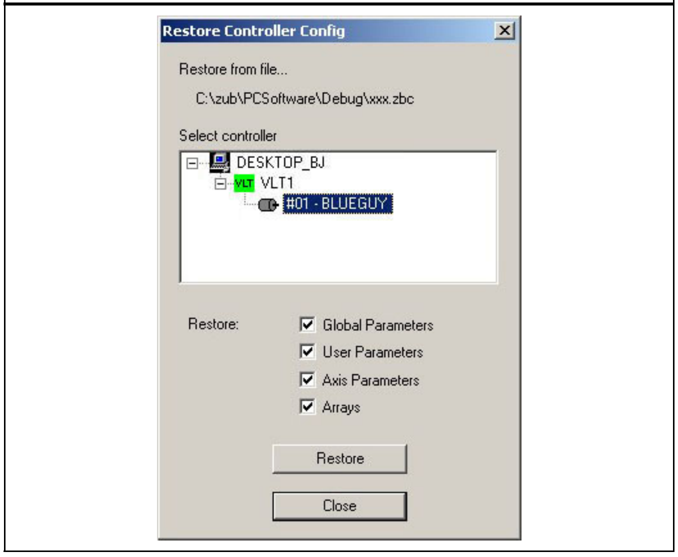
Parameters > Restore from file

If a controller's configuration has been backed up as a file on the PC, then the configuration can be restored to the controller again by downloading the configuration file.



Click on *Parameters* → *Restore from file*. This will display an "Open" dialog. Select the file containing the data to be loaded. In the subsequent dialog, select the FC 300 into which the data should be loaded. Use the checkboxes to specify which sets of parameters are to be loaded. Then click the → *Restore* button. The specified parameters are then loaded into the FC 300. Any previous parameter values are overwritten.

If the <u>same</u> data is to be loaded into more than one FC 300, then simply select another FC 300 and click → *Restore from file* again.


Restore Controller Config

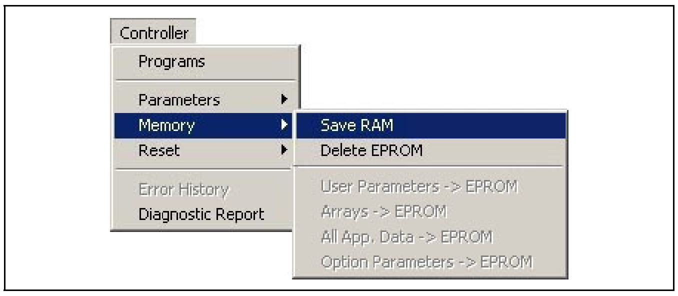
Restore from file...
C:\zub\PCSoftware\Debug\xxx.zbc

Select controller
DESKTOP_BJ
VLT1
#01 - BLUEGUY

Restore: [✓] Global Parameters
[✓] User Parameters
[✓] Axis Parameters
[✓] Arrays

□ Memory

As the case maybe which controller is connected, the function in the first or second group are enabled. For example if a prior version of a VLT 5000 is connected, then the first two menu commands are used. If in contrast a MCO 305 Option is connected, then the four commands of the second group are used.



<u>Save RAM</u>

The → *Save RAM* function saves all programs, parameters, and arrays from RAM into EPROM. This corresponds to the SAVEPROM command. This function is usually needed only to save arrays if necessary since programs and parameters are automatically saved. Any data that have not been saved from RAM into EPROM will be lost when the controller is powered off.

NB!:
Some very old versions of controllers did not automatically save programs in EPROM. For these controllers, use this function to save the programs into EPROM.

<u>Delete EPROM</u>

The → *Delete EPROM* function will reset all parameters to their initialization values and delete all arrays and programs. The controller is reset to the original factory setting. However, this is done <u>only after the controller has been turned off and on again</u>.

NB!:
Remember the following when you delete the EPROM:

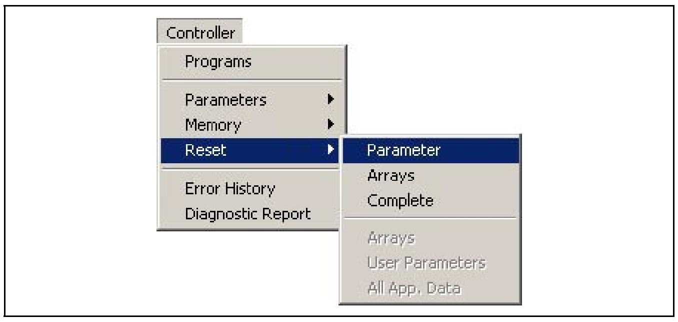
1. Check whether you have saved all the necessary APOSS programs on the computer. These will need to be reloaded into the controller again if necessary after the controller has been turned back on.
2. Check whether you have saved the controller configuration into a backup file on the computer (using *Parameters* → *Save to file*). With this file, you can reload the previous parameters and arrays if necessary.
3. Click on *Memory* → *Delete EPROM*.
4. Re-load the necessary configuration parameters and APOSS programs in the controller.

<u>User Parameters, Arrays, All Appl.Data, Option Parameters -> EPROM</u>

Use the corresponding function to save → *User Parameters*, → *Arrays*, → *All Application-Data* (which also contains the application program in addition to the user parameters) or → *Options Parameter*s(MCO 305 parameters) individually in the LCP-EEPROM.

□ Reset

As the case maybe which controller is connected, the function in the first or second group are enabled. For example if a prior version of a VLT 5000 is connected, then the first two menu commands are used. If in contrast a MCO 305 Option is connected, then the four commands of the second group are used.



Parameter

The → *Parameter* function will reset all global and axis parameters in the controller to their initialization values.

Arrays

The → *Arrays* function will delete all arrays in RAM. This function has the same effect as the DELETE ARRAYS command.

NB!:
Note that if *Memory* → *Save RAM* is subsequently used, or an APOSS program executes the SAVE ARRAYS command, then the arrays in the EPROM are also deleted!

Complete

The → *Complete* function will reset all parameters and delete all arrays and programs. The MCO 305 Option is reset to the original factory setting.

NB!:
Reset → *Complete* happens immediately. This differs from *Memory* → *Delete EPROM* which happens only after the controller has been turned off and on again.

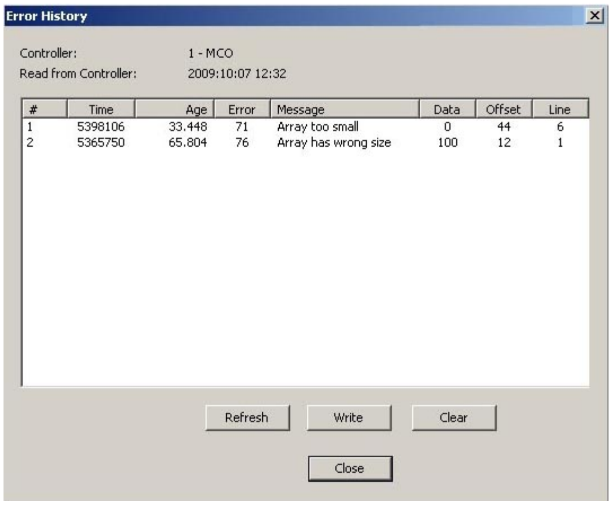
Arrays, User Parameters, All App.data

Use the corresponding function to reset → *Arrays,* → *User Parameters* or → *All Application data* (which also contains the application program in addition to the user parameters and arrays) individually in the LCP-EEPROM. The reset is executed at once!

□ Error History

Click on *Controller* → *Error History* to see a history of all the errors that have occurred on the controller.

NB!:
Some older versions of controller do not support this function. The function will be disabled in this case.



Controller: 1 - MCO
Read from Controller: 2009:10:07 12:32

#	Time	Age	Error	Message	Data	Offset	Line
1	5398106	33.448	71	Array too small	0	44	6
2	5365750	65.804	76	Array has wrong size	100	12	1

Errors are listed from most recent at the top of the list to oldest at the bottom of the list. The list is cleared each time that the controller is powered on. The list holds a maximum of 50 errors; the oldest error will be discarded when a new error is added to the list.

The following information is listed:

Time	Controller system time when error occurred. See "sys_clock" in System Process Data.
Age	Age (in seconds) of the error when the Error History dialog was opened (i.e. how long ago the error occurred). This will be updated each time the dialog is refreshed.
Error	Error number. See chapter "Troubleshooting".
Message	Error message text.
Data	Optional value saved with error. The meaning of this value depends on the error.
Offset	Binary offset within the compiled program where the error occurred.
Line	Line number within the program being edited where the error occurred. Double-clicking on an error in the list will close the Error History dialog and highlight this line in the Edit Window.

NB!:
The APOSS-IDE cannot verify that the program currently being edited matches the program executing in the controller at the time the error occurred. This is the user's responsibility. If the programs do not match, then the wrong line will be highlighted in the Edit Window.

The following functions are available:

Refresh	Update the list with the current error history from the controller. Note that this will update the "age" of the existing errors in the history.
Write	Write the error history into an ASCII text file so that the errors can be analyzed at a later time. This file can be edited by any text editor or imported into a spreadsheet application.
Clear	Clear the error history.

□ Diagnostics Report

Click on *Controller* → *Diagnostics Report* to create a text file containing the entire current state of the controller. This file is often required by the support personnel when assistance is required to help diagnose problems.

□ Tools Menu

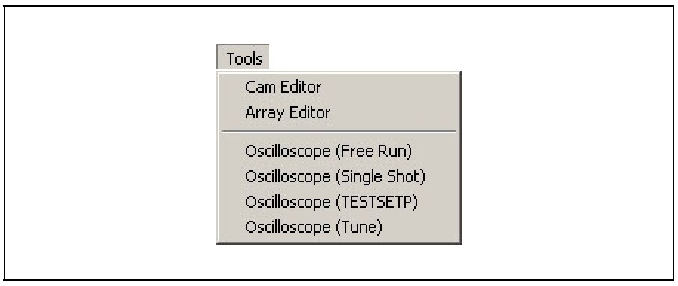
The Tools menu offers the following tools:

CAM-Editor

Array Editor

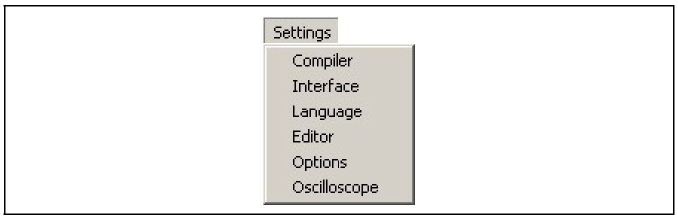
Oscilloscope

Please see chapter "APOSS Tools" for a detailed description.



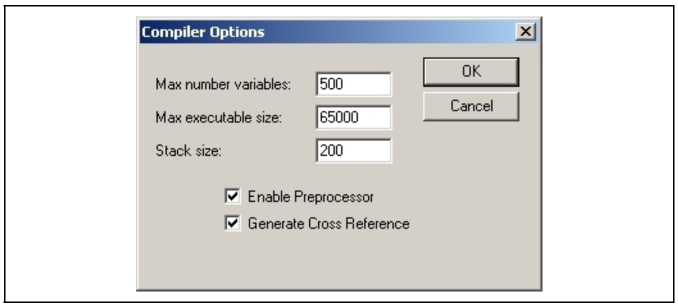
□ Settings Menu

This menu offers various options and settings.



□ Compiler

The default values for the compiler options are set appropriately for most applications.



Max number variables: 500
Max executable size: 65000
Stack size: 200
☑ Enable Preprocessor
☑ Generate Cross Reference

Max number variables

The Maximum number of variables has a direct effect on the amount of memory available in the controller. It is important to remember that an array also occupies the space of a variable.

Max 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 65000 bytes.

Stack size

Stack size specifies how much memory is to be reserved for the internal procedure call stack. Unless deeply nested procedure calls are being used, this value should be left unchanged.

Enable Preprocessor

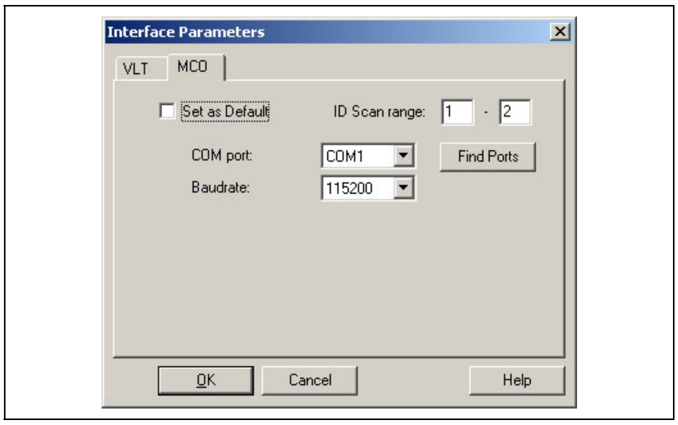
Enabling the preprocessor allows preprocessor statements such as #define to be used within programs. It also allows nested #include files to be used. For a complete description of the preprocessor capabilities, please see Preprocessor in chapter "Programming with APOSS".

Generate Cross Reference

Cross references should always be enabled. They are necessary for the functioning of the Watch Window and for debugging within APOSS.

□ Interface Parameters

Normally, connection interface parameters will have been set correctly when the APOSS-IDE is first installed and used (see chapter "How to Get Started" in the „MCO 305 Operating Instructions"). However, if they need to be changed for any reason (e.g. the baud rate or COM port has changed, the controller ID scan range needs to be extended, a new interface type has been added to the system, etc.), then *Settings* → *Interface* can be used to update the parameters. A dialog similar to the following will be displayed.



☐ Set as Default
ID Scan range: 1 - 2
COM port: COM1
Baudrate: 115200

Simply select the appropriate interface type and change the necessary parameters.

NB!:
Note that only one interface type can be selected as the default interface. The default interface is used to establish connections to controllers when the [Esc] key is pressed.

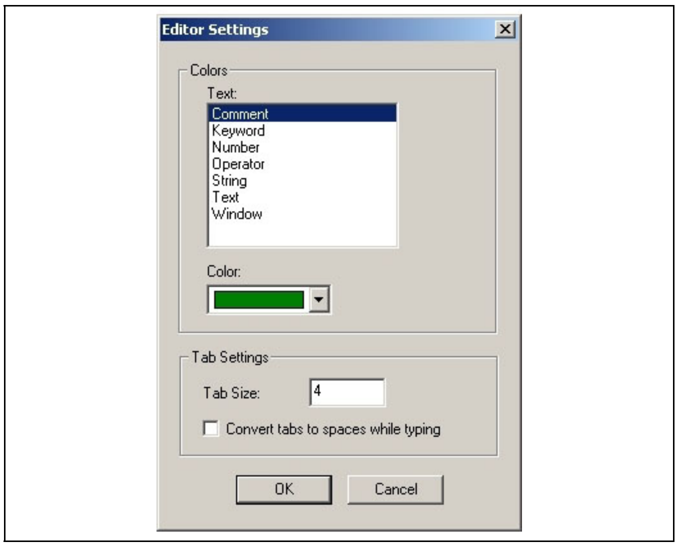
NB!:
Setting the interface parameters will affect only subsequent connections to controllers. Any controller to which there is already an established connection, will remain connected and will not be changed. If an established connection to a controller needs to be changed, then it will be necessary to close the connection to that controller, change the interface settings, and then re-connect to the controller.

□ Language

If you desire another language, click on *Settings* *Language* and choose from the available languages in the subsequent dialog field. → *Exit program* and start APOSS again. It is necessary to exit the program and then restart again before the language change will take effect.

□ Editor Settings

In order to provide greater clarity, different colors can be assigned to the various program elements such as comments, key words, numbers, etc. Tab settings can also be chosen to make the program more readable.


Editor Settings
Colors
Text:
Comment
Keyword
Number
Operator
String
Text
Window
Color:
Tab Settings
Tab Size: 4
☐ Convert tabs to spaces while typing

<u>Colors</u>

Select the type of program element (e.g. Comment) and select the desired color.

<u>Tab Settings</u>

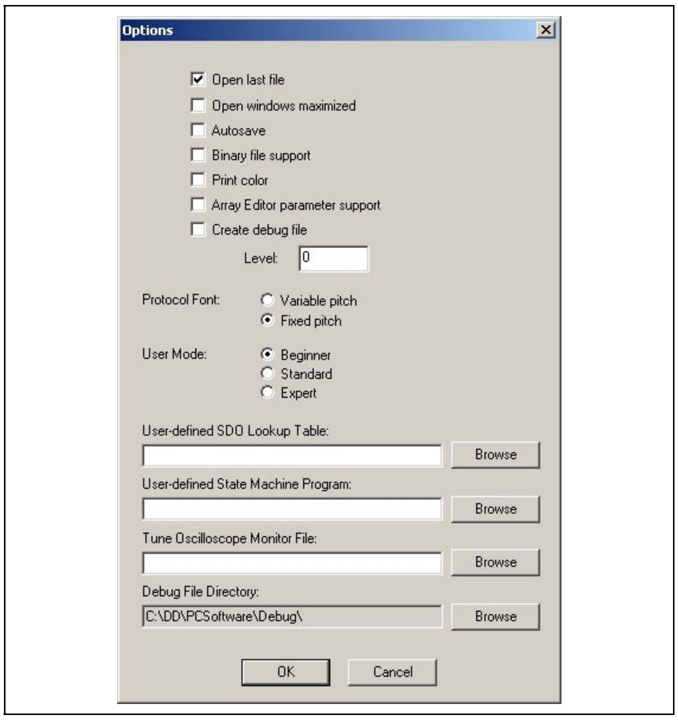
Choose the preferred tab size. This can be set to 2 or more.

<u>Convert Tabs</u>

Click on → *Convert Tabs to spaces while typing* to convert all tab characters to space characters while typing. Note that this does not affect any tab characters that already exist in the program. Use *Edit* → *Convert Tabs* if you want to convert all existing tabs in a program to spaces.

□ Options

Only in APOSS stand-alone: This dialog allows you to select various options that control the behavior of APOSS. When settings are changed in the → *Options* dialog, the settings will take effect the next time that they are used.


Options

☑ Open last file
☐ Open windows maximized
☐ Autosave
☐ Binary file support
☐ Print color
☐ Array Editor parameter support
☐ Create debug file

Level: 0

Protocol Font: ○ Variable pitch [✓ ● Fixed pitch]

User Mode: [✓ ● Beginner] ○ Standard ○ Expert

User-defined SDO Lookup Table:

User-defined State Machine Program:

Tune Oscilloscope Monitor File:

Debug File Directory: C:\DD\PCSoftware\Debug\

Open last file

If this option is enabled, then when APOSS starts, it will automatically try to reopen the most recently used file.

Open windows maximized

If this option is enabled, then all windows opened in APOSS will be opened full-size. They can subsequently be reduced in size, if desired; they will simply be opened full-size. If this option is not enabled, then only the first window opened will be opened full-size. If a second window is opened, then all windows will be made smaller so that all windows can be seen simultaneously.

Autosave

Enabling this option will cause APOSS to automatically save the current file being edited to disk before compiling and executing it. If not enabled, then APOSS will use a temporary file for compiling the current file.

Binary file support

If enabled, then APOSS will allow the direct handling of binary compiled program images in the *Controller → Programs* window. This includes uploading compiled programs from the controller so that they can be saved on the PC and downloading previously saved binary images back to the controller. Note that these binary images cannot be edited; they can only be saved and restored. Also note that some older controllers do not support this feature.

If enabled, this option will also cause the item "Compile to file" to be added to the *Development* menu (once APOSS has been re-started the next time). This item allows to user to manually compile and save the current program file to a binary file.

See also BinFileMap in illustrations in chapter Appendix in the MCO 305 Command Reference.

Print color

If this is enabled, then programs printed from the Edit Window will be printed in color using the same colors as the Edit Window. Otherwise, programs will be printed in black and white.

Array Editor parameter support

Normally, the Array Editor will read and display all user parameters and arrays. However, if this setting is enabled, then the Array Editor will also read and display global parameters, temporary and permanent axis parameters, axis process parameters, and system process parameters.

Create debug file

Enabling this option will cause APOSS to generate debug log files. This is normally useful only to product support personal. This option should normally be disabled.

Protocol font

This is the type font that is used to display messages in the Communications Window. Changing this value will only affect newly opened windows.

User Mode

The user mode allows APOSS to be tailored to the experience level of the user. Currently, this mostly applies to the *Oscilloscope* function only.

Customized Sections

The *SDO Lookup Tables* allow more experienced users to customize the contents of the SDO selection lists used in APOSS. For example, these are used in the Watch Window and the *Oscilloscope.*

The *State Machine Program* setting allows experienced users to use a customized state machine program in the *Tune Oscilloscope*.

The *Tune Oscilloscope Monitor File* setting allows experienced users to use a customized test window in the *Tune Oscilloscope*.

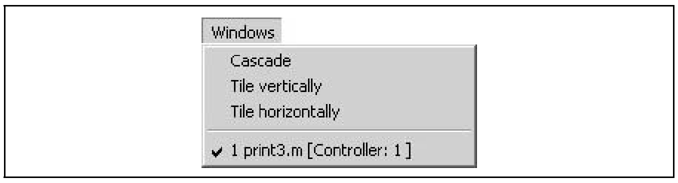
The *Debug File Directory* allows users to specify where debug log files are to be created.

□ Oscilloscope

This dialog allows you to select various options that control the behavior of APOSS oscilloscope tool. Please see *Oscilloscope* for more detail.

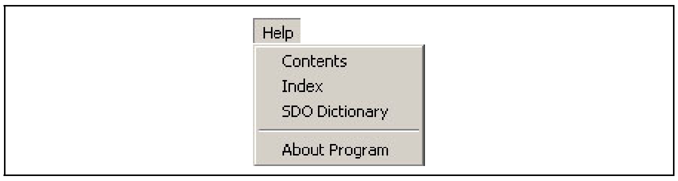
☐ Window Menu

The commands on the *Window* menu follow the Windows standards (i.e. *Cascade* for overlapping windows arranged down and across, *Tile vertically* for vertical windows arranged beside each other, or *Tile horizontally* for horizontal windows arranged above/below each other).



☐ Help Menu

The *Command list* [F12] and the parameter dialog fields in the *Controller* menu and *CAM-Editor*, offer direct access to help. Select a command in the *Command list* or one of the parameter input fields and press [F1]. Help for that specific item will be displayed.



The → *Contents* function will start the online help subsystem and display the “Contents” tab.

The → *Index* function will start the online help subsystem and display the “Index” tab.

The → *SDO Dictionary* function will start the online help subsystem and immediately jump to the “SDO Object Dictionary” page.

The → *About Program* function will display the version numbers of the APOSS-IDE, the low level interface driver, and the compiler.

The look and feel of the Help subsystem will depend on the version of the Windows operating system being used.

▫ Download Menu

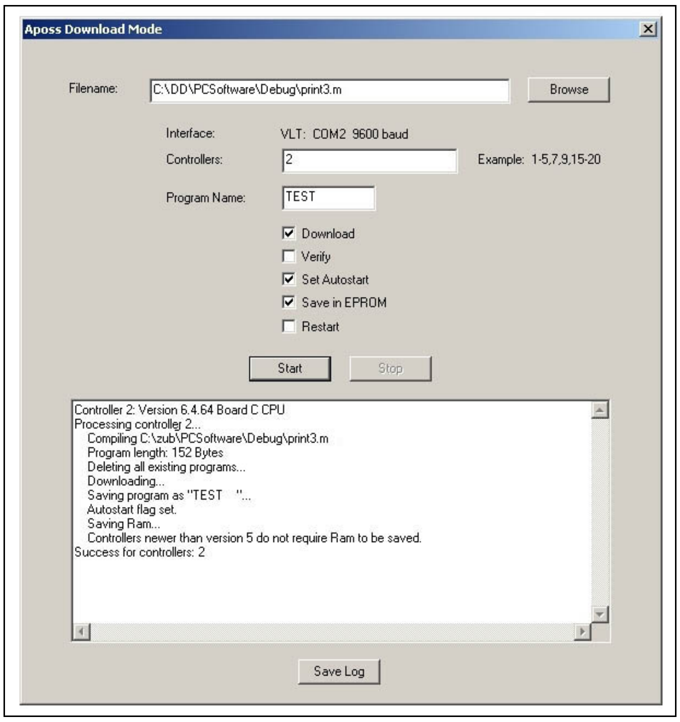
The *Download* menu provides a simple interface to support the downloading of programs to multiple controllers.

Note that all APOSS Windows, with the exception of the main APOSS-IDE application window, must be closed before this menu will be available.

Select → *Programs* and use the *Browse* button to select the APOSS program that should be down-loaded. If the controller interface supports multiple controllers, then enter the range of controller ID's to which the program should be downloaded.

Enter a program name (up to 8 characters) to be used to identify the program on the controller.

Select the desired operations using the check boxes *Download, Verify, Set Autostart, Save in EPROM*, and/or *Restart*. For example, you might check *Verify* but not *Download*. In that case the Downloader will only verify that the specified file is the one that is already on the controller.



Filename: C:\DD\PCSoftware\Debug\print3.m

Interface: VLT: COM2 9600 baud

Controllers: 2 Example: 1-5,7,9,15-20

Program Name: TEST

- [✓] Download
- [ ] Verify
- [✓] Set Autostart
- [✓] Save in EPROM
- [ ] Restart

```
Controller 2: Version 6.4.64 Board C CPU
Processing controller 2...
   Compiling C:\zub\PCSoftware\Debug\print3.m
   Program length: 152 Bytes
   Deleting all existing programs...
   Downloading...
   Saving program as "TEST    "...
   Autostart flag set.
   Saving Ram...
   Controllers newer than version 5 do not require Ram to be saved.
Success for controllers: 2
```

<u>Start Download Programs</u>

Click the *Start* button. A connection is established to the controllers using the default APOSS interface. Any controller that is currently executing will be stopped. The following steps are performed for each accessible controller. All steps are performed for one controller before going on to the next controller.

- The program is compiled using compiler settings appropriate to the controller.
- All existing programs on the controller are deleted.
- The program is downloaded.
- The program is saved as program number 0 with the specified program name. If no program name was specified, then the first 8 characters of the filename are used.
- If the *Verify* flag is set, then the downloaded program is uploaded again and the downloaded and uploaded versions are compared. If they differ, then processing is halted for this controller. Note that not all older controllers contain firmware that will support uploading the program. This flag is ignored if the controller does not support uploading.
- The *Autostart* flag is set for the program if requested.
- All data is *saved in EPROM* if requested. Note that this will be done automatically for newer controllers.
- The controller is *restarted* with the new program if requested.
- The connection to the controllers is closed.

If any error occurs during the processing of a controller, then processing is halted for that controller. However, processing will continue for subsequent controllers in the ID range. When the entire download operation is complete, a summary is displayed in the text box listing the ID's of those controllers that encountered no errors and those controllers that encountered errors. Use the scroll bars to scroll up in the text box in order to determine the nature of any problems encountered.

Save Log

The progress of the download is displayed in the large text box. If desired, this information can be saved to a text file by clicking the *Save Log* button.

If the download fails for one or more controllers, then you will get a "Failed for controllers" message. Saving the log will let you go back and review the failures at a later time.

☐ Debugging Programs

The APOSS-IDE contains a powerful built-in debugger. This provides common debug features such as single-stepping, breakpoints, and the ability to read and set program variable values.

NB!:
The debugger cannot be used in all cases. For example, <u>it may not be possible to use the debugger with programs that are actively controlling a motor</u>. Stopping program execution at a breakpoint is equivalent to pressing the [Esc] key to break program execution. This will cause the motor to slow down and stop with the maximum allowed deceleration. In many cases, stopping the motor like this will invalidate the test procedure and make the debug result meaningless. As well, if program execution is continued after a breakpoint, then it is unlikely that the motor can be restarted correctly to put the system into the state it was in prior to the breakpoint.

NB!:
<u>It may also not be possible to use the debugger with programs that rely on ON PERIOD functions</u>. The internal timer that triggers calls to ON PERIOD functions does <u>not</u> stop when program execution stops at a breakpoint. This may leave a pending interrupt which will then trigger an ON PERIOD call as soon as program execution is continued.

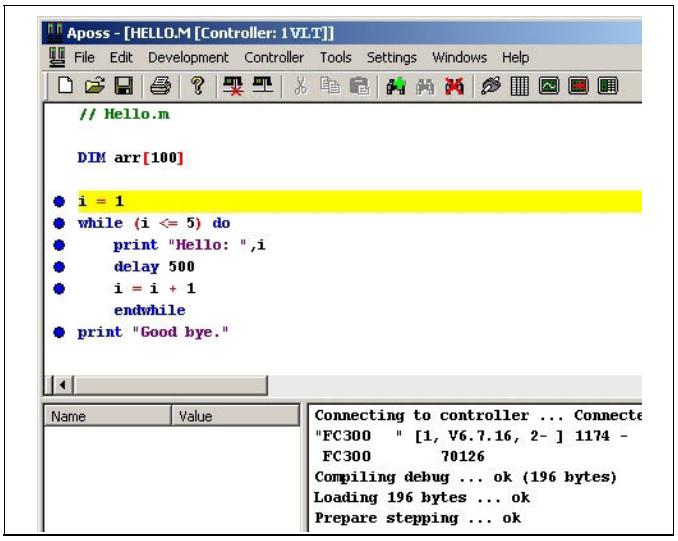
For situations like the above where the debugger cannot be used, the Oscilloscope provides excellent debugging capabilities. It can watch and record program variables and system states without having to break program execution. These can then be reviewed afterwards to identify problems. For more information on the Oscilloscope, please see APOSS Oscilloscope in chapter “APOSS Tools”.

☐ Starting the Debugger

To start the debugger, edit the program to be debugged in the normal way so that it is displayed in the Edit Window. Then click on *Development* → *Prepare Singlestep*. This will take the following actions:

1. The program is compiled in debug mode and downloaded to the controller. However, program execution is <u>not</u> started at this time.
2. A blue dot is placed before each executable statement in the program. These are the positions where the user may insert breakpoints.
3. The <u>next</u> statement to be executed (i.e. when execution is started or continued) will be highlighted in yellow.

The diagram shows what the Edit Window might look like:


Aposs - [HELLO.M [Controller: 1VLT]]
File Edit Development Controller Tools Settings Windows Help
Name
Value

```
  // Hello.m

  DIM arr[100]

● i = 1
● while (i <= 5) do
●     print "Hello: ",i
●     delay 500
●     i = i + 1
      endwhile
● print "Good bye."
```

```
Connecting to controller ... Connecte
"FC300   " [1, V6.7.16, 2- ] 1174 -
 FC300       70126
Compiling debug ... ok (196 bytes)
Loading 196 bytes ... ok
Prepare stepping ... ok
```

NB!:
While the debugger is active, the program should not be modified. Doing so will cause the APOSS-IDE to become out-of-sync with the version of the program executing on the controller and the debugger may no longer be able to properly follow the program execution. If the program must be changed, then the debugger should be stopped and the test restarted from the beginning.

☐ Stopping the Debugger

To stop the debugger and end the debugging session, click on *Development* → *End Debug*. This will remove the blue dots marking executable statements. Note that this will not remove any breakpoints (marked with red dots). Breakpoints will remain and can be used the next time that the debugger is started. See “Using Breakpoints” for more information on breakpoints.

☐ Single-Stepping

To single-step through the program, use *Development → Singlestep* or press [F9]. This will execute the next statement (i.e. the statement currently highlighted in yellow) and automatically stop before the next executable statement is executed. This next statement will then be highlighted in yellow.

While execution is stopped, the user is free to examine and modify the value of any program variable, set and clear breakpoints, etc.

At any time, program execute can be continued without single stepping, by using *Development → Go*.

☐ Using Breakpoints

Breakpoints are set by double-clicking anywhere on the statement in the program (except on the blue dot) where the breakpoint is to be set. The dot will change from blue to red to indicate that the breakpoint has been set.

Double-clicking on a statement that already contains a breakpoint will clear the breakpoint and the dot will change back from red to blue.

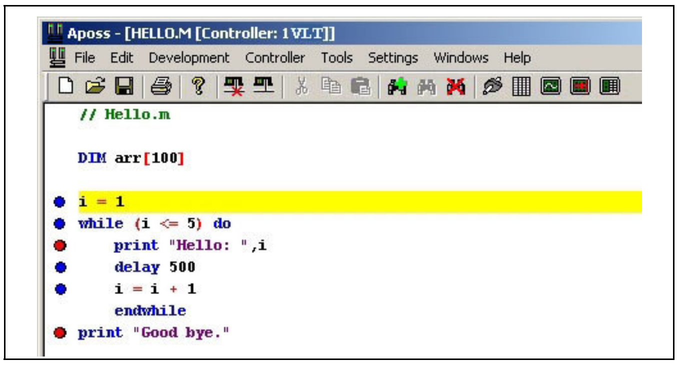
The user can set a "breakpoint" at any executable statement in the program. These are the statements with blue dots. When program execution encounters a breakpoint, execution is immediately stopped prior to executing the statement with the breakpoint. The statement will then be highlighted in yellow since this is the next statement to be executed.

While execution is stopped, the user is free to examine and modify the value of any program variable, set and clear breakpoints, etc.

When ready, the user can click on *Development → Go* to continue program execution. The program will then execute up until it encounters another breakpoint. Note that it is also possible to single-step (using *Development → Singlestep* or [F9]) after stopping at a breakpoint.

At any time during program execution, the user may also press the [Esc] key to stop execution. Execution will then stop immediately rather than continuing to the next breakpoint. The next statement to be executed will be highlighted in yellow. Again, the user can continue with either *Development → Go* or *Development → Singlestep* [F9].

The diagram shows what the Edit Window might look like when breakpoints have been set before each "print" statement. Note the red dot on these statements.


Aposs - [HELLO.M [Controller: 1 VLT]]
File Edit Development Controller Tools Settings Windows Help

```
// Hello.m

DIM arr[100]

i = 1
while (i <= 5) do
    print "Hello: ",i
    delay 500
    i = i + 1
    endwhile
print "Good bye."
```

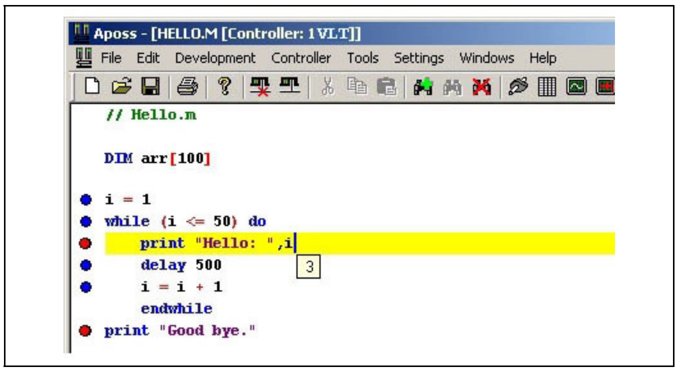
NB!:
A maximum of 10 breakpoints are allowed.

NB!:
#DEBUG commands have been replaced by breakpoints and are no longer necessary. Any existing #DEBUG commands need not be removed from the program; they will simply be ignored.

◻ Displaying and Modifying Variables

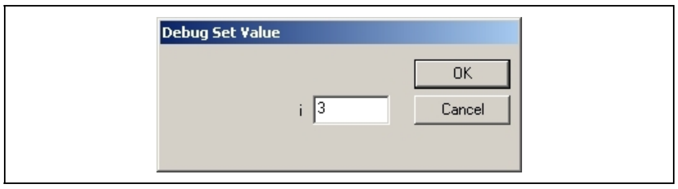
At any time while the debugger is active (i.e. whether the program is executing or stopped), the current value of any program variable can be displayed.

This is done by clicking on (or immediately after) any instance of the variable with the left mouse button.The current value will be displayed in a yellow popup box. The value will be displayed until the mouse is moved away from the variable. An example is shown where the mouse button was clicked near the variable "i" in the "print" statement.


Aposs - [HELLO.M [Controller: 1VLT]]
File Edit Development Controller Tools Settings Windows Help
3

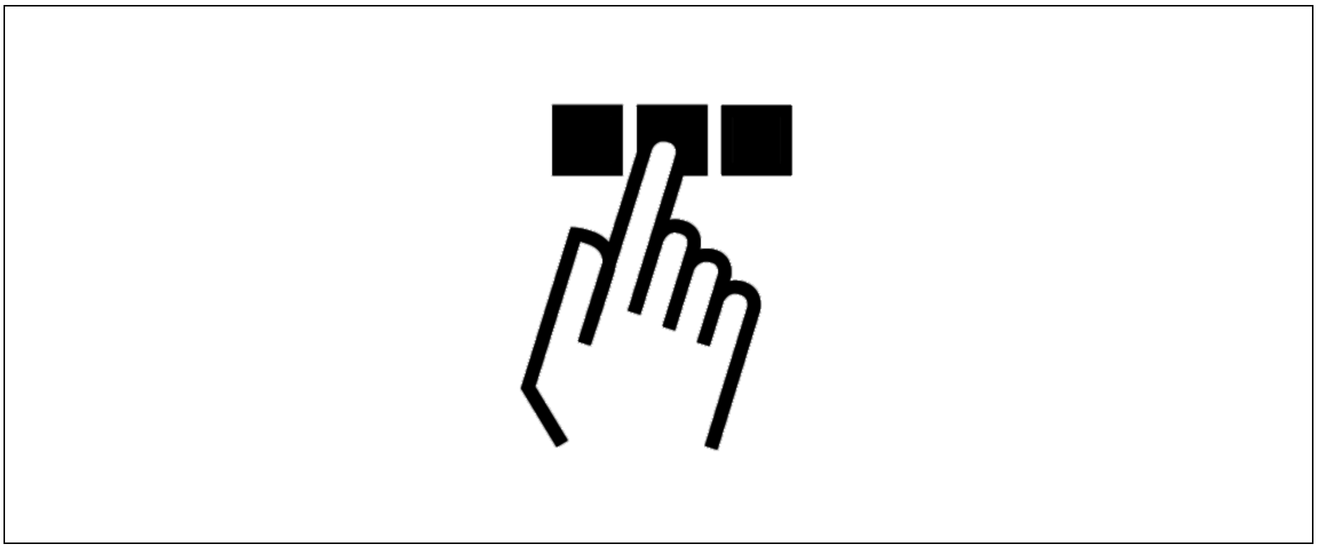
```
// Hello.m

DIM arr[100]

i = 1
while (i <= 50) do
    print "Hello: ",i
    delay 500
    i = i + 1
    endwhile
print "Good bye."
```

At any time while the debugger is active, the current value of any program variable can also be modified. This is done by clicking on (or immediately after) any instance of the variable with the right mouse button.



The dialog shown will be displayed allowing the user to modify the variable value.

APOSS Tools



▫ CAM-Editor

Curve profiles for CAM controls are created using the CAM-Editor. Curve profiles are defined by Fix points, parameters for engage and disengage motion, as well as parameters for synchronization with markers. The curves and parameters are displayed graphically and can be edited either graphically or textually.

You will find application samples of CAM controls and CAM box in chapter "Functions and Examples".

Each curve profile is stored in a single Global array on the controller. Multiple curves can be created for a CAM control simply by using multiple arrays.

The CAM-Editor will save the curve profile definitions and all parameter values in a "configuration file" with the extension ".zbc". They will be identified in Windows folders with the icon shown to the left. As with other types of files, configuration files may be opened, modified, and saved by the CAM-Editor at any later time.

NB!:
Previous versions of the APOSS-IDE used the file extension ".cnf" for configuration files. This was changed from ".cnf" to ".zbc" to avoid conflicts with the usage of the ".cnf" file extension by Microsoft Windows. However, existing ".cnf" files are still supported and can be read by the CAM-Editor simply by opening them in the normal way. So you are free to save it as a ".zbc" or ".cnf" file.

The CAM-Editor is designed to use exactly the same ".zbc" configuration file format as is used by the *Controller* → *Parameters* → *Save to file* and *Restore from file* commands to save and restore complete controller configurations. Hence, the configuration files created by the CAM-Editor will store all Global, User, and Axis parameters <u>in addition</u> to the curve profiles. As a result, you should always start the CAM-Editor with parameters <u>from the controller to which the curve profiles are intended</u>. Otherwise, the <u>wrong</u> Global, User, and Axis parameters may be downloaded to the controller when the curve profiles are downloaded.

Once curve profiles have been created and saved in a ".zbc" file, they can be downloaded to the controller using *Controller* → *Parameters* → *Restore from file*.

□ How to Start the CAM-Editor

MCT 10 Online and Offline Mode

The configuration files (*.zbc and *.cnf) can be opened using MCT 10. Doing so, the APOSS CAM editor is also started. Depending on the mode some menu functions are disabled. Use the functions of the MCT 10 Motion Control Tools for *New*, *Open* and *Save as*.

The file must contain minimum one curve. If the file does not contain a curve, you will be requested to add a curve to this file. In opposite you cannot delete the last curve of a file.

If you are starting with new curve profiles and a configuration file does not yet exist, then the recommended procedure is always to start with parameters <u>from the controller to which the curve profiles are intended</u>.

NB!:
If parameters are not read from the controller, then default factory settings will be used. In this case, the existing controller parameters will be <u>overwritten</u> with the factory defaults when this configuration file is downloaded to the controller.

When the CAM editor is closed, then APOSS is exit also and the program returns to the MCT 10.

Click on → *Write to Drive,* to download the new zbc values (especially the CAM arrays) to the drive. The new values will also be save in the MCT 10 data base and will overwrite a prior version of the file (i.e. it restore it into the MCT 10 data base).

APOSS stand-alone

Using APOSS stand-alone click on *Tools* → *CAM-Editor* to start the editor or click the CAM-Editor toolbar icon. An "Open file" dialog will be displayed allowing the user to select the configuration file to be edited. The CAM-Editor Window will then be opened.

The CAM-Editor can also be started by using *File* → *Open* and selecting the configuration file to be opened.

Simply double-clicking on a configuration file in Windows will also start the CAM-Editor. Note that double-clicking a ".cnf" file will <u>not</u> start the CAM-Editor; the file must be a ".zbc" file.

If you are starting with new curve profiles and a configuration file does not yet exist, then the recommended procedure is always to start with parameters <u>from the controller to which the curve profiles are intended</u>. This can be done in the following ways:

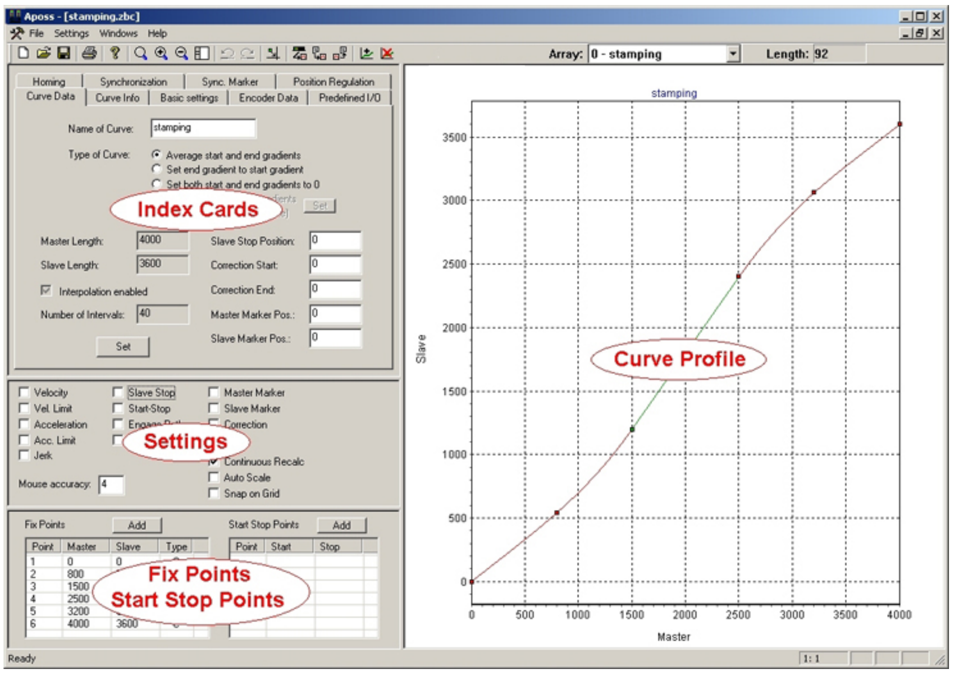
1. First connect to the controller and then create a configuration file using *Controller* → *Parameters* → *Save to file*. The newly created configuration file can then be opened with the CAM-Editor.
2. Open a "new" configuration file by either using *File* → *New* or by starting the CAM-Editor and pressing → *Create empty file* in the subsequent "Open file" dialog. Then use the *Read from Controller* toolbar button to read the current parameters from the controller.

NB!:
If parameters are not read from the controller, then default factory settings will be used. In this case, the existing controller parameters will be <u>overwritten</u> with the factory defaults when this configuration file is downloaded to the controller.

New configuration files will not contain any curve profiles. Use the → *Add CAM Array* toolbar button to insert a new curve profile.

Several sample configuration files containing curve profiles can be found using *File* → *Sample*.

☐ CAM-Editor Window



Name of Curve: stamping

Type of Curve: (●) Average start and end gradients ( ) Set end gradient to start gradient ( ) Set both start and end gradients to 0

Master Length: 4000 | Slave Stop Position: 0

Slave Length: 3600 | Correction Start: 0

[✓] Interpolation enabled | Correction End: 0

Number of Intervals: 40 | Master Marker Pos.: 0

Slave Marker Pos.: 0

☐ Velocity ☐ Vel. Limit ☐ Acceleration ☐ Acc. Limit ☐ Jerk ☐ Slave Stop ☐ Start-Stop ☐ Master Marker ☐ Slave Marker ☐ Correction [✓] Continuous Recalc ☐ Auto Scale ☐ Snap on Grid

Mouse accuracy: 4

Fix Points

Point	Master	Slave	Type
1	0	0	[illegible]
2	800	[illegible]	[illegible]
3	1500	[illegible]	[illegible]
4	2500	[illegible]	[illegible]
5	3200	[illegible]	[illegible]
6	4000	3600	[illegible]

Start Stop Points

Point	Start	Stop

The *CAM-Editor* Window is divided into four sections:

- Curve Profile diagram.
- Index Cards: *Curve Data*, *Curve Info*, and all the axis parameter index cards normally available using *Controller* → *Parameters* → *Axis*.
- *Settings* for displaying and hiding various attributes of the curve profile diagram.
- *Fix Points* and *Start Stop Points* tables.

The CAM-Editor window can be resized in the standard way. As well, the splitter bars between the four window sections can be adjusted using the mouse in the standard way. The splitter bars can be restored to their default positions by using the → ◫ *Reset splitters* toolbar button.

CAM-Editor Toolbar

The CAM-Editor toolbar contains the following special fields:

Array – This field lists all the arrays currently defined and allows the user to select the array to be edited. Each item in the list will consist of the array number followed by the curve profile name. If the array does not correspond to a curve profile, then “Array” will be displayed as the name. Non-curve arrays can be selected but they cannot be edited as curves.

Length – This displays the minimum length required for the array. This is the minimum value needed for the DIM statement in the APOSS program.

The non-standard toolbar buttons that are specific to the CAM-Editor are as follows:

Zoom Reset – Redisplay the diagram so that the entire curve is visible.

Zoom In by a factor of 2 and display only the central portion of the diagram.

Zoom Out by a factor of 2.

Reset splitters – Restore the splitters in the window to their default positions.

Undo the last change to either a Fixpoint or a Start Stop point. If multiple changes were made at a single time (for example adding multiple points, or by clicking the → *Snap on Grid* button), then these changes will be undone as a set. If no changes are available to be undone, then this button will be disabled and grayed.

Note that if multiple curve arrays exist, then a separate undo list is maintained for each curve.

Redo a change that was undone using the Undo button.

Snap on Grid – Snap all Fix points and Start Stop points to the interpolation grid.

Read from controller – Read the existing configuration parameters and all arrays and curves from a controller. This will replace all existing parameters and arrays currently in the CAM-Editor.

Import an ACSII text file that has been exported with the → *Export* toolbar button. The imported curve will replace the currently selected curve. The format of the file must exactly match an export file or the import will not succeed.

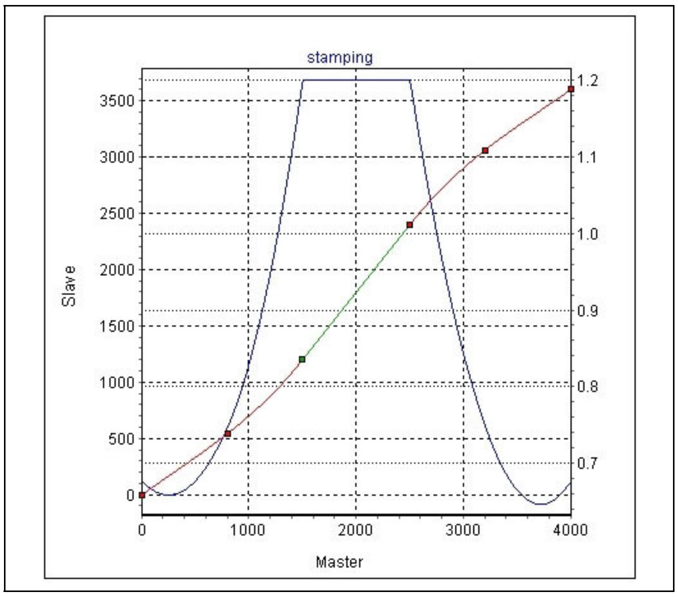
Export the curve data for the currently selected curve to an ASCII text file with the extension “.dat”. This is useful if you want to process the curve outside of APOSS (for example, using a spreadsheet). Please see “Array Structure of CAM Profiles” in the chapter “Technical Reference” for a description of the exact format of the array.

Add CAM array – A new curve profile array is added after the last existing array. A dialog is displayed allowing the user to set the initial attributes of the curve. The new curve will then be automatically selected and displayed. The initial curve will contain only two Fix points and will be a straight line. Additional Fix points should then be added to define the actual shape of the curve.

Delete array – The currently selected array will be deleted. Note that any array may be deleted; the array need not be a curve profile array.

◻ Curve Profile Diagram

The curve profile diagram graphically displays the curve along with other information about curve synchronization. It also allows Fix points to be added, deleted, and moved using the mouse.



Please note the following about the curve profile diagram:

- The name of the curve, if any, is shown at the top of the chart.
- The master distance is shown horizontally across the <u>bottom</u> of the diagram. The slave position is shown vertically up the <u>left</u> side of the diagram. Various other scales (e.g. velocity) will be shown vertically up the <u>right</u> side of the diagram when necessary.
- Fix points are displayed as small red or green squares. "Curve" points (i.e. Fix points that begin a curved segment of the profile) will be red and "tangent" points (i.e. Fix points that begin a tangent segment of the profile) will be green.
- Curved segments of the profile will be drawn red and tangent segments will be drawn green. Note that segments that are straight will be drawn green even though they may be "curve segments" (i.e. segments that begin with a "curve" point).

Zoom and Pan

The mouse buttons can be used in the curve profile diagram for the following functions:

<u>Zoom</u>

Zoom in to an area of interest in the diagram by pressing the left mouse button, moving the mouse across the area of interest, and then releasing the mouse button.

NB!:
Note that the selected rectangle will be stretched so that it is exactly displayed in the window. If stretching is not desired, then press and hold the [Ctrl] key before pressing the left mouse button.

Use the *Zoom Out* or *Zoom Reset* toolbar buttons to zoom back out again.

Zooming the diagram will automatically clear the → ☑ *Auto Scale* setting.

<u>Pan</u>

Pan across the diagram (i.e. move the diagram in the window) by pressing the right mouse button, moving the mouse, and then releasing the mouse button.

Panning the diagram will automatically clear the → ☑ *Auto Scale* setting.

Use *Zoom Reset* to reset both the pan and zoom so that the entire diagram is displayed again.

Using the Mouse to Edit Fix Points

Fix points can be edited interactively using the mouse within the diagram. But they can also be input directly in the Fix points table, please see Fix points**.**

Identify Fix Point

Fix points can be identified simply by moving the mouse and pausing over the Fix point in the diagram. The mouse cursor icon will change from the normal arrow icon to a hand icon and the corresponding Fix point will automatically be highlighted in the Fix Points table. The required "nearness" of the mouse to the Fix point can be adjusted using → *Mouse accuracy* in the window Curve profile settings**.**

Move Fix Point

A Fix point can be moved by dragging it with the left mouse button. Make sure that the hand icon is displayed before trying to drag the Fix point. The point's position will be automatically updated in the Fix Points table as the point is being dragged.

If → ☑ *Continuous Recalc* is enabled, then the curve profile will also be automatically recalculated and the Curve Info index card will be updated as the point is dragged.

Note that if → ☑ *Snap on Grid* is enabled, then the point will always snap onto the nearest interpolation grid line and will not follow the mouse exactly.

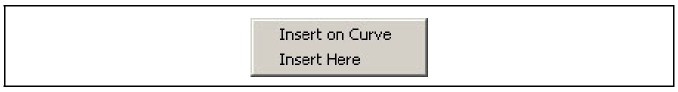
If necessary, use the *Undo* toolbar button to restore the point to its previous position.

Add Fix Point

A new Fix point can be added by moving the mouse to the point where the Fix point is to be placed and then double-clicking the left mouse button. If necessary, use the *Undo* toolbar button to delete the point. Please see Fix points for more information on adding Fix points, e.g. Edit Fix points and Fix points Types.

Using CAM Popup Menus

If the right mouse button is clicked in an empty part of the curve profile diagram, then the following popup menu will be displayed:



Insert on Curve

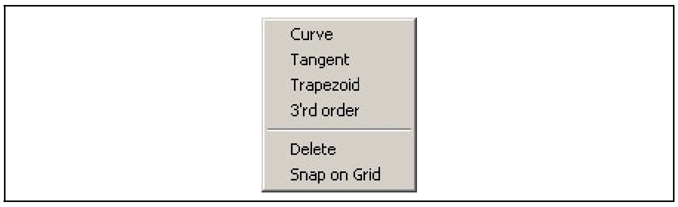
A new Fix point is inserted at the specified master position but on the curve rather than where the mouse button was clicked.

Insert Here

A new Fix point is inserted at the position where the mouse was clicked. This is the same as double-clicking with the left mouse button. The curve profile will be recalculated.

Change Fix point Type

If the right mouse button is clicked on a Fix point, then the following popup menu will be displayed to change this Fix point type:



Select the type to which the identified point should be changed: *Curve*, *Tangent*, *Trapezoid*, or *3'rd order***.**

Delete

Delete the identified Fix point.

Snap on Grid

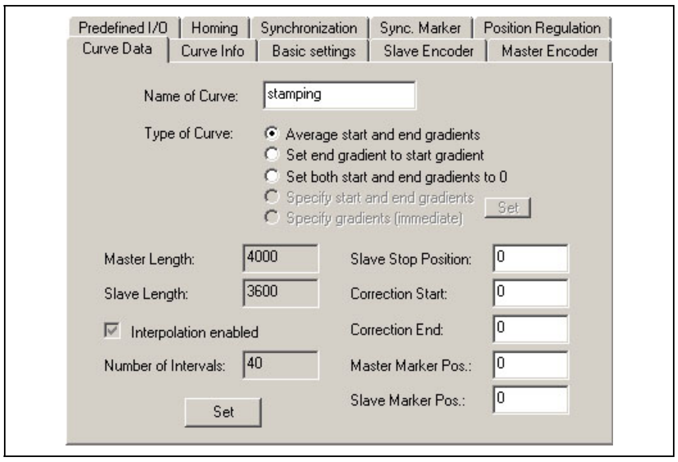
Snap this Fix point to the nearest interpolation grid line.

◻ CAM Index cards

The CAM-Editor index cards allow various parameters related to curve profiles to be changed. This is done using the *Curve Data* and *Curve Info* index cards. In addition, other index cards allow all axis parameters to be changed. These axis parameters will be downloaded to the controller along with the curve profiles when the configuration file is downloaded. Please see Parameter Reference for a detailed description of the axis parameters. The axis *Synchronization* parameters are of particular interest when working when with curve profiles. These will be described below.

A detailed help description of any parameter in the index cards may be displayed by placing the text cursor in the parameter field (e.g. by clicking on the field) and then pressing [F1].

Index card Curve Data



Name of Curve: stamping

Type of Curve: [✓ Average start and end gradients] Set end gradient to start gradient / Set both start and end gradients to 0 / Specify start and end gradients / Specify gradients (immediate)

Master Length:	4000	Slave Stop Position:	0
Slave Length:	3600	Correction Start:	0
[✓] Interpolation enabled		Correction End:	0
Number of Intervals:	40	Master Marker Pos.:	0
		Slave Marker Pos.:	0

<u>Name of Curve</u>

If you edit several curves, you can give meaningful names to the curves. Names may be up to 16 characters long and will be displayed in the “Array” field in the CAM-Editor toolbar. These names are for your own information; the controller does not use them when working with curve profiles.

<u>Type of Curve</u>

Typically, curve profiles are cyclic; when the controller reaches the end of a curve profile, it will restart the curve profile from the beginning again. As a result, it is extremely important that the gradients (i.e. velocity) of the curve profile matches at the start and end of the profile. This allows the slave motor to run smoothly from the end of the profile back onto the start of the profile as the master cycles around the profile. A discontinuity in the gradient would cause a hard velocity leap which could potentially damage the motor or connected machinery.

The *Type of curve* allows the user to select how the CAM-Editor will match the start and end gradients. The following curve types are available:

– *Average start and end gradients* – The actual slave velocity is not important at either the start or end of the profile. The CAM-Editor will choose an average gradient simply to ensure smoothness.

– *Set end gradient to start gradient* – The actual slave velocity at the start of the profile is important and should not be changed. The CAM-Editor will adjust only the end gradient so that it matches the start gradient. The start gradient is not changed.

– *Set both start and end gradient to 0* – The slave motor must be stopped when the master cycles. The CAM-Editor will set both the start and end gradients to 0.

– *Specify start and end gradients* – The actual slave velocity is important at <u>both</u> the start and end of the profile. The user must explicitly set the start and end velocities using the “Set” button. The CAM-Editor will warn the user if the start and end gradients do not match but it is the user’s responsibility to ensure than any discontinuity does not cause problems.

– *Specify gradients (immediate)* – The user must explicitly set the start and end velocities using the "Set" button. However, the start gradient will be replaced with the actual current velocity when the curve is activated in the controller. This type of curve is intended to be used when an existing curve must be replaced by a different curve before the first curve has completed its cycle.

NB!:
Note that not all versions of controllers can support all types of curve profiles.

Master Length and Slave Length

The master length is the distance that the master encoder must travel in order to complete one cycle; it is the horizontal length of the curve profile. As the master completes one cycle, the slave motor will also complete one cycle by following the curve profile from the start point to the end point. The slave length will be the difference in the slave position from the beginning to end of the curve. Note that this is not the same as the vertical height of the curve. If the slave has returned to its starting position when the master completes one cycle, then the slave length will be 0 even though the slave has moved during the cycle.

The master length and slave length can be set in three ways:

1. Changing the last Fix point 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 and slave cycle lengths directly. 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 (described below) to be changed simultaneously. The slave length cannot be changed using this dialog.

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

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

If the last Fix point must be moved beyond these limits, then it will be necessary to first change the limiting points or positions before moving the last Fix point. For example, if the last Fix point is being blocked by the master marker position, then move the master marker position first and then move the last Fix point.

Slave Stop Position

This specifies the position to which the slave should run and stop if no "SYNCCSTOP pnum slavepos" command with the variable "slavepos" was executed in the APOSS program.

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

If → ☑ *Slave Stop* is enabled, then a horizontal grey bar will indicate this position in the curve profile diagram.

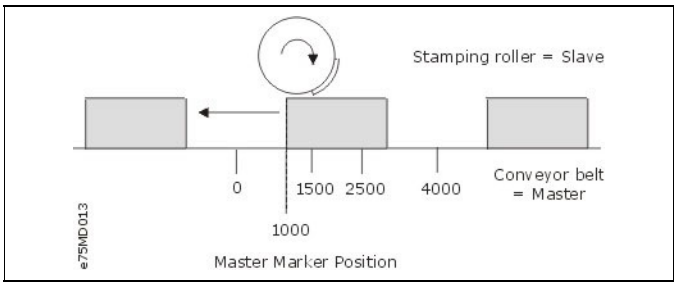
Correction Start and End

This specifies the master positions where the master correction should begin and end. Be careful to leave enough time to correct the synchronization before the processing point is reached.

If → ☑ *Correction* is enabled, then the correction area is shown as a blue-grey bar along the X-axis in the curve profile diagram.

Master and Slave Marker Positions

Enter the master position (or the slave position in the case of slave synchronization with marker) for which the marker has been set. For example, the beginning of the box in the illustration below is the master marker position.



The position of the curve where the marker is detected is calculated from the master marker position and the marker distance. This allows you to fix the correction area.

If → ☑ *Master Marker* is enabled, then a vertical green line will indicate the master marker position in the curve profile diagram. If → ☑ *Slave Marker* is enabled, then a horizontal green line will indicate the slave marker position.

Interpolation and Number of Intervals

Controllers containing firmware older than MCO 5.00, require "interpolation points" when processing curve profiles. For newer controllers, these are no longer necessary and this topic can be skipped entirely. The firmware version of the controller is displayed in the APOSS Communication Window when the controller is first connected in the APOSS-IDE.

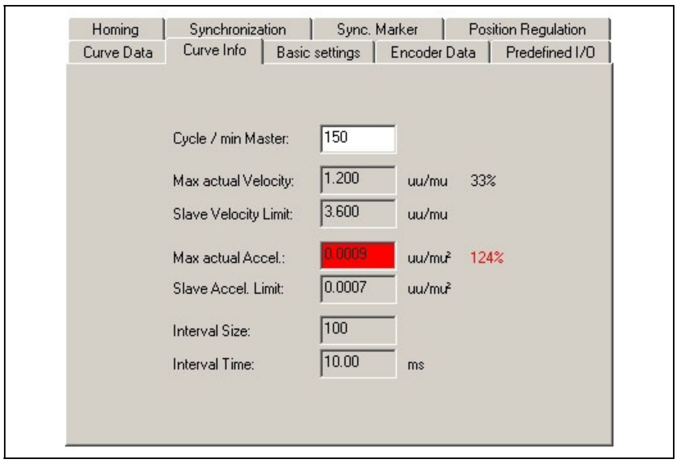
If the curve profile is intended for an older controller, then the *Interpolation enabled* checkbox should be selected. This can be done either when a new curve profile is added or by using the "Set" button. It cannot be done directly from the *Curve Data* index card.

Older controllers require many more Fix points than newer controllers. If interpolation is enabled, then the CAM-Editor will do this automatically for the user by adding interpolation points (i.e. "virtual" Fix points) on a uniformly spaced "interpolation grid". The size of the interpolation interval (i.e. the distance between interpolation points) determines the accuracy with which the controller can follow the curve profile and must be chosen by the user based on the accuracy requirements of the application. Smaller interpolation intervals will give higher accuracy but will result in more interpolation points and larger arrays in the controller. Very small intervals can also cause the controller to have performance problems as a result of the large number of short segments. The "Interval Time" (found on the *Curve Info* index card) should not be smaller than 20-30 ms.

In order to avoid a fractional number of interpolation points, the *Master Length* is required to be an integral multiple of the interpolation interval. To ensure this, the CAM-Editor will always snap the last Fix point to the interpolation grid. When the "Set" button is used, the CAM-Editor will also ensure that the new *Master Length* is a multiple of the interpolation interval. As well, when the *Master Length* is changed with the "Set" button, the number of interpolation points is changed automatically so that the interpolation interval (i.e. the curve accuracy) remains unchanged. It is recommended that the master cycle length be at least 1000 in order to allow for an adequate resolution of the interpolation grid.

Although not strictly necessary, it is recommended that user Fix points also be placed exactly on the interpolation grid. The *Snap all to grid* toolbar button and the ☑ *Snap on Grid* checkbox option can be used to assist the user in placing Fix points on the interpolation grid. If Fix points are not placed on the interpolation grid, then the actual curve profile followed by the controller will pass near the off-grid Fix points but may not pass exactly through them.

Index Card Curve Info



Cycle / min Master: 150

Max actual Velocity: 1.200 uu/mu 33%

Slave Velocity Limit: 3.600 uu/mu

Max actual Accel.: 0.0009 uu/mu² 124%

Slave Accel. Limit: 0.0007 uu/mu²

Interval Size: 100

Interval Time: 10.00 ms

Cycles/min Master

This specifies 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.

Maximum Actual Velocity

Max actual Velocity indicates the maximum velocity of the slave required by this curve profile in user units per master unit (UU/MU). This is also shown as a percentage of the *Slave Velocity Limit*. If the velocity exceeds the limit, then it is shown in red.

You can graphically display the velocity in the curve profile diagram by enabling → ☑ *Velocity***.** The velocity curve of the slave will be shown in blue and the axis will be displayed on the right side of the chart (as long as acceleration and/or jerk are not also being displayed). Note that it may be necessary to press the *Reset Zoom* toolbar button for the velocity curve to be displayed with the proper scale within the existing chart boundaries.

Slave Velocity Limit

Slave Velocity Limit is the maximum velocity of which the slave is capable, given the current value of *Cycle/min Master* and the maximum speed rating of the slave motor. As *Cycle/min Master* increases, *Slave Velocity Limit* will decrease so that the actual velocity of the slave does not exceed the maximum speed rating of the slave motor. If the velocity required by this curve (*Max actual Velocity*) exceeds this limit, then the slave will not be able to follow the master.

If → ☑ *Vel. Limit* is enabled, then the velocity limit will be displayed as horizontal dark-blue lines in the curve profile diagram. Note that both positive and negative velocity limits are displayed.

Maximum Actual Acceleration

Max actual Accel. indicates the maximum acceleration of the slave required by this curve profile in user units per master unit squared (UU/MU²). This is also shown as a percentage of the *Slave Accel. Limit*. If the acceleration exceeds the limit, then it is shown in red.

You can graphically display the acceleration in the curve profile diagram, by enabling → ☑ *Acceleration***.** The curve of the slave will be shown in magenta and the axis will be displayed on the right side of the chart (as long as velocity and/or jerk are not also being displayed). Note that it may be necessary to press the 🔍 *Reset Zoom* toolbar button for the acceleration curve to be displayed with the proper scale within the existing chart boundaries.

Slave Acceleration Limit

Slave Accel. Limit is the maximum acceleration of which the slave is capable, given the current value of *Cycle/min Master* and the maximum acceleration rating of the slave motor. As *Cycle/min Master* increases, *Slave Accel. Limit* will decrease so that the actual acceleration of the slave does not exceed the maximum acceleration rating of the slave motor. If the acceleration required by this curve (*Max actual Accel.*) exceeds this limit, then the slave will not be able to follow the master.

If → ☑ *Accel. Limit* is enabled, then the acceleration limit will be displayed as horizontal dark-magenta lines in the curve profile diagram. Note that both positive and negative acceleration limits are displayed.

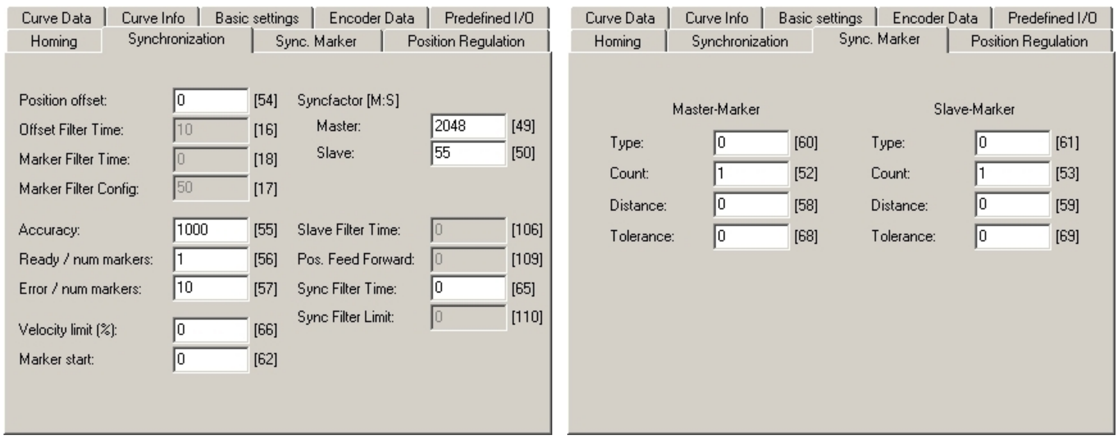
<u>Interval Size and Interval Time (ms)</u>

These values are only important if interpolation has been enabled. The interval size is the length of a single interpolation interval in master units. It is derived from the number of intervals and the length of the master cycle.

The interval time (in milliseconds) is derived from the interval size and the number of cycles of the master per minute. This time should not be smaller than 30 milliseconds. 30 to 100 ms are suitable values. Changes should be made in *Curve Data* → *Number of Intervals* in order to ensure a reasonable value.

Index Card Synchronization and Sync Marker

The axis synchronization parameters are of particular interest when working with curve profiles since they affect the synchronization of the slave to the master. The relevant parameters are described below.



Position offset:	0	[54]	Syncfactor [M:S]		
Offset Filter Time:	10	[16]	Master:	2048	[49]
Marker Filter Time:	0	[18]	Slave:	55	[50]
Marker Filter Config:	50	[17]			
Accuracy:	1000	[55]	Slave Filter Time:	0	[106]
Ready / num markers:	1	[56]	Pos. Feed Forward:	0	[109]
Error / num markers:	10	[57]	Sync Filter Time:	0	[65]
			Sync Filter Limit:	0	[110]
Velocity limit (%):	0	[66]			
Marker start:	0	[62]			

Master-Marker			Slave-Marker		
Type:	0	[60]	Type:	0	[61]
Count:	1	[52]	Count:	1	[53]
Distance:	0	[58]	Distance:	0	[59]
Tolerance:	0	[68]	Tolerance:	0	[69]

<u>Syncfactor Master and Slave</u>

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

<u>Marker Distance</u>

Enter the distance of the sensor to the processing point: Parameter 33-17 *Master Marker Distance* and 33-18 *Salve Marker Distance*. The position of the curve where the marker is detected is calculated from the master marker position and the marker distance.

If → ☑ *Master Marker* or *Slave Marker* is enabled, then these positions will be shown as a green line in the curve profile diagram.

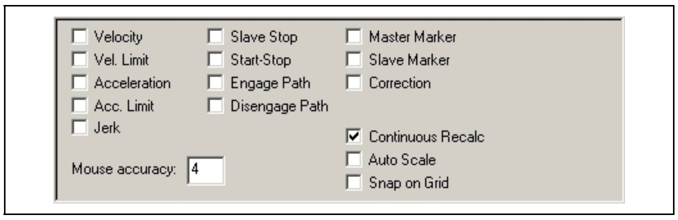
<u>Tolerance</u>

This is the tolerance window for the appearance of the master marker 33-21 SYNCMWINM or the slave marker 33-22 SYNCMWINS (Marker monitoring).

If → ☑ *Master Marker* or *Slave Marker* is enabled, then the tolerance window is shown as a green area in the curve profile diagram.

☐ Curve profile settings

The Settings section of the CAM-Editor Window contains checkboxes for displaying and hiding various attributes of the curve profile diagram.


☐ Velocity ☐ Slave Stop ☐ Master Marker
☐ Vel. Limit ☐ Start-Stop ☐ Slave Marker
☐ Acceleration ☐ Engage Path ☐ Correction
☐ Acc. Limit ☐ Disengage Path
☐ Jerk
☑ Continuous Recalc
Mouse accuracy: 4
☐ Auto Scale
☐ Snap on Grid

<u>Velocity</u>

Display the velocity curve corresponding to the curve profile. This is displayed as a blue line. The grid scale for this curve is displayed down the right side of the curve profile diagram (as long as this is the only grid to be displayed on the right side of the chart).

<u>Vel. Limit</u>

Display the velocity limits as two horizontal dark blue lines.

<u>Acceleration</u>

Display the acceleration curve corresponding to the curve profile. This is displayed as a magenta line. The grid scale for this curve is displayed down the right side of the curve profile diagram (as long as this is the only grid to be displayed on the right side of the chart).

<u>Acc. Limit</u>

Display the acceleration limits as two horizontal dark magenta lines.

<u>Jerk</u>

Display the “jerk” curve corresponding to the curve profile. The “jerk” is the impulse on the drive as a result of acceleration changes. This is displayed as a cyan (light blue) line. The grid scale for this curve is displayed down the right side of the curve profile diagram (as long as this is the only grid to be displayed on the right side of the chart).

<u>Slave Stop</u>

Display the slave stop position (defined in the *Curve Data* index card) as a horizontal grey bar.

<u>Start-Stop</u>

Display the positions of the Start Stop points as yellow or red flags on the curve profile.

<u>Engage Path</u>

Display the engage path for the currently selected Start Stop pair. This is displayed as a dark blue curve from the slave stop position to the curve profile between the Start Stop pair.
Note that a Start Stop pair must be selected in the Start Stop Points table before anything is displayed.

<u>Disengage Path</u>

Display the disengage path for the currently selected Start Stop pair. This is also displayed as a dark blue curve.

<u>Master Marker</u>

Display the master marker position (derived from the data in the *Curve Data*, Synchronization, and Sync Marker index cards). This is displayed as a vertical green line. If a tolerance has been specified, then the tolerance area is displayed as a pale green rectangle underneath the master marker position.

<u>Slave Marker</u>

Display the slave marker position (derived from the data in the Curve Data, Synchronization, and Sync Marker index cards). This is displayed as a horizontal green line. If a tolerance has been specified, then the tolerance area is displayed as a pale green rectangle underneath the slave marker position.

<u>Correction</u>

Display the positions where the master correction should begin and end (defined in the *Curve Data* index card). This is displayed as a blue-grey bar along the bottom axis of the curve profile diagram.

<u>Continuous Recalc</u>

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. This redrawing of the curve may require a significant amount of processing power. If this redrawing behaves erratically or creates a disruption on your computer, then you can deactivate it; the curve will then be recalculated and displayed as soon as you release the mouse button.

<u>Auto Scale</u>

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.

<u>Snap on Grid</u>

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 fixpoint is <u>always</u> 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.

<u>Mouse accuracy</u>

If the right mouse button is pressed while the mouse is pointing at a Fix point, then the Fix point popup menu is displayed. Otherwise, the normal popup menu is displayed. This value specifies how near to a Fix point the mouse must be before the CAM-Editor will recognize that the mouse is pointing at that Fix point. Note that a "hand" mouse icon (rather than the normal arrow mouse icon) will be displayed when the CAM-Editor has determined that the mouse is pointing at a Fix point. That Fix point will also be highlighted in the Fix Point Table so that the user can positively identify the point.

□ 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. Note that you should not define more Fix points than are necessary to adequately produce the final desired curve profile. Adding more points than necessary will consume controller resources without a corresponding increase in benefit.

Note that any change made to Fix points using the Fix Points table, can be undone using the *Undo* toolbar button.

Edit Fix points

Please see Using the Mouse in Curve Profile Diagram for a description of how to add, delete, and modify Fix points using the mouse in the curve profile diagram.

Adding Fix points

To add a new Fix point, click 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. If → ☑ *Snap on Grid* is enabled, then the master values will be snapped onto the interpolation grid as they are being added.

If a new Fix point is added after the last Fix point, then this will change the master cycle length. When this happens, the number of interpolation points is changed automatically so that the interpolation interval (i.e. the distance between interpolation points) remains unchanged. The last Fix point is always snapped to the interpolation grid in order to avoid a fractional number of interpolation points.

Deleting Fix points

To delete a Fix point, click on the Fix point in the Fix Points table and then press the [Del] key.

Modifying Fix points

To modify a Fix point, double-click on the Fix point in the Fix Points table. If the "Master" or "Slave" column is clicked, then the Fix point position can be changed. If the "Type" column is clicked, then the Fix point type can be changed.

Note that a Fix point cannot be moved beyond either of the two adjacent Fix points. The last Fix point can be moved as far as desired towards the right. If this is done, then the master cycle length is automatically changed. The number of interpolation points is also changed automatically so that the interpolation interval (i.e. the distance between interpolation points) remains unchanged. The last Fix point is always snapped to the interpolation grid in order to avoid a fractional number of interpolation points.

Fix points Types

The Fix point type determines the shape of the single curve profile segment that follows the Fix point. Segments can either be straight (i.e. tangent) or curved. The *CAM-Editor* will always try to make a smooth transition from one segment to the next. For this reason, at least one curve segment is required between adjacent tangent segments.

In the curve profile diagram, tangent points are displayed as small green squares and all other types of points are displayed as small red squares.

The following Fix point types are supported:

Curve (C)

The following segment is curved. If it connects adjacent tangent segments, then it will be calculated using a single 5'th order curve. This provides the smoothest possible transition between the segments since both the velocity and acceleration will match exactly at both ends of the segment. However, it may require higher velocities and accelerations than other curve types. If there is more than one consecutive curve segment, then the series of curve segments will be calculated together using a single cubic spline.

Tangent (T)

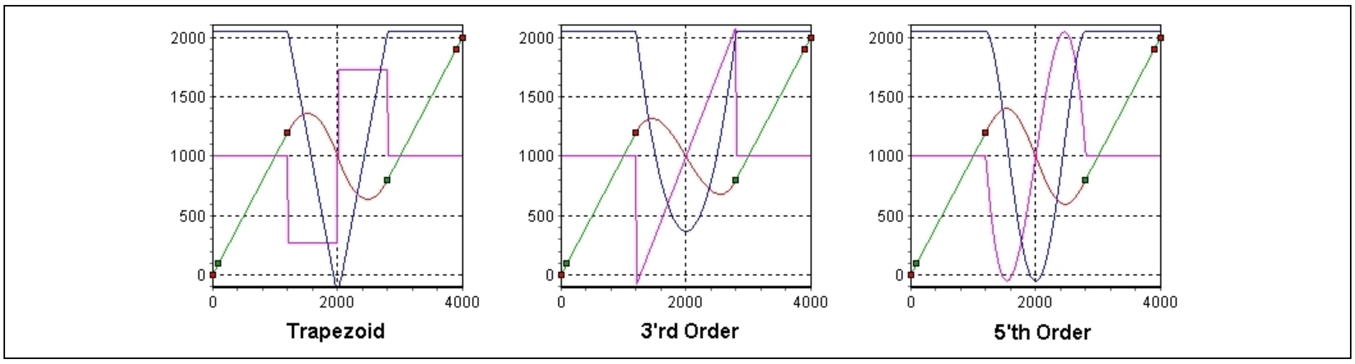
The following segment is straight. The velocity will be constant and the acceleration will be zero.

Trapezoid (CZ)

The following segment will be calculated using a trapezoidal velocity profile (i.e. two segments of constant acceleration). This curve type can only be used between adjacent tangent segments.

3'rd order (C3)

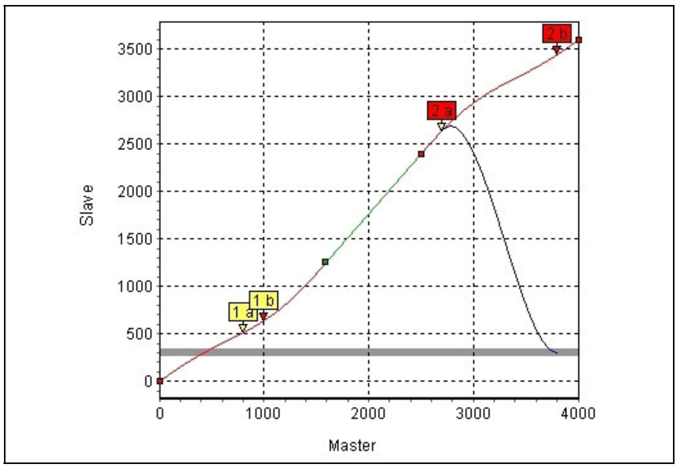
The following segment will be calculated using a single 3'rd order curve. This curve type can only be used between adjacent tangent segments.



In the trapezoid curve, the two segments of constant acceleration (magenta line) can be seen. This produces a sharp change in velocity (blue line) midway through the curve. In the 3'rd order curve, the velocity change is smoother but there are still sharp changes in the acceleration. In the 5'th order curve, the changes in both the velocity and acceleration are smoother.

◻ Start Stop Points

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



If → ☑ *Start Stop* is enabled, then Start Stop points will be displayed with yellow flags. 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. Selecting a Start Stop pair by clicking on it in the Start Stop Points table will cause the flags to be displayed in red. This will allow you to identify which pair is which.

Enable → ☑ *Engage Path* and/or → ☑ *Disengage Path* to display the engage and/or disengage curves. Then select the Start Stop pair of interest by clicking on it in the Start Stop Points table. Only one pair of engage and disengage curves can be displayed at a time.

Enable → ☑ *Slave Stop* to display the slave stop position. This is the position where the engage curve starts and the disengage curve ends. It will be displayed as a horizontal gray bar.

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!

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 end at the Stop point. When moving backward, it begins at the Stop point and ends at the Start point.

A maximum of 25 Start Stop point pairs can be defined. This is useful, for example, to define multiple starts and stops in a cycle in order to account for different situations during the start. Using the commands "SYNCCSTART pnum" and "SYNCCSTOP pnum slavepos", you can determine in your program which point pair is to be used.

If *Start points* **=** *Stop points*, the slave will be engaged with the set maximum velocity (i.e. without a curve) as soon as the master has reached this point.

If *Start point* **>** *Stop point*, then the engaging/disengaging operation will "wrap around" the master cycle.

If no *Start Stop points* have been defined, then SYNCCSTART will cause the slave will be engaged with the set maximum velocity.

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

Edit Start Stop points

Start Stop points are edited in the Start Stop table:

Adding Start Stop points

To add a new pair of Start Stop points, click the → *Add* button in the *Start Stop Points* table. This will display the dialog for adding point pairs. 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.

Deleting Start Stop points

To delete a pair of Start Stop points, click on the Start Stop point pair in the Start Stop Points table and then press the [Del] key.

Modifying Start Stop points

To modify a pair of Start Stop points, double-click on the Start Stop point pair in the Start Stop Points table. This will display a dialog allowing the pair to be changed.

◻ Array Editor

The APOSS *Array Editor* provides a list-oriented interface that allows all controller arrays and parameters to be viewed and edited. This includes user parameters, global parameters, axis parameters, axis state variables, and system state variables.

In addition, an optional "*Array Definition file*" may be created as a template specifying such things as element names, minimum and maximum values, default values, etc. Using an array definition file allows much more user-friendly editing of arrays. The Array Editor behaves slightly differently depending on whether or not an array definition file is being used. This is described later. Array definition files will always use ".zba" as the file extension.

The Array Editor supports two modes: an Expert mode and a User mode. Expert mode allows full access to all arrays and parameters and to all editing features. Expert mode is intended to be used to create the array definition files that will subsequently be used in User mode. In User mode, access to arrays and parameters is restricted to only those elements pre-defined in an array definition file. An array definition file may be used but cannot be edited in User mode. The main intent of User mode is to provide a "configuration tool" for field personnel. In this case, the array definition file can be used to limit access to only those pre-defined items required by the configuration.

The "Array Editor parameter support" affects the Array Editor. It can be accessed using the *Settings* → *Options* command.
If ☑ *Array Editor Parameter Support* is enabled, then all arrays and parameters (i.e. user, global, axis, etc.) are loaded into the Array Editor.
If ☐ *Array Editor Parameter Suppor*t is disabled, then only arrays and user parameters are loaded. Other parameters are not accessible even if they are specified in an array definition file.

◻ Starting the Array Editor

The Array Editor can be started either from within the APOSS IDE or as a stand-alone application.

When started from within the APOSS IDE, the Array Editor always starts in Expert mode and allows full access to all arrays and parameters. The Array Editor window will behave in a similar way to the other APOSS windows (i.e. edit window, CAM Editor, oscilloscope, etc.).

When started as a stand-alone application, the Array Editor can be started in either Expert mode or User mode. The stand-alone Array Editor window will be a Windows dialog.

Only one instance of APOSS may be running at a time so if APOSS is already running, then a message is displayed and the stand-alone Array Editor instance will terminate.

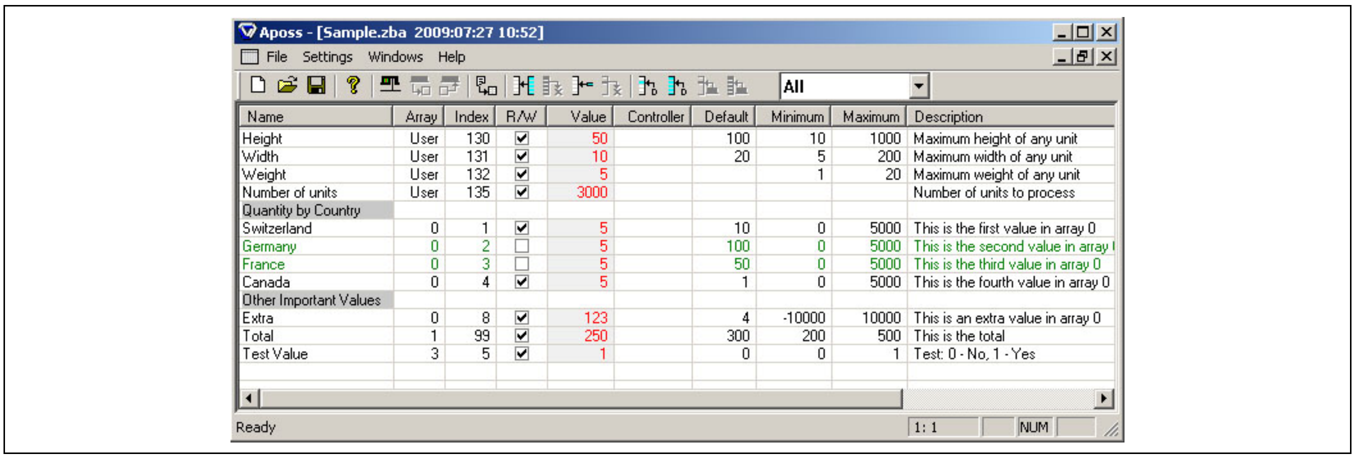
Within APOSS

The Array Editor can be started from within the APOSS IDE using any of the following methods:

1. Using the *Tools* → *Array Editor* menu item or the Array Editor toolbar button.
2. Using the *File* → *New* menu item or the File New toolbar button and then selecting "Create Array .zba file".
3. Using the *File* → *Open* menu item and then selecting an existing ".zba" file.
4. Using any of the other Windows file open mechanisms such as double-clicking on a ".zba" file or dragging a ".zba" file onto an open APOSS window.

In all of these cases, if a controller is already connected in the currently active window, then the arrays and parameters are automatically loaded from that controller.

The window title will include the controller ID and name (if a controller is connected) and the date when the values were read from the controller.


Aposs - [Sample.zba 2009:07:27 10:52]
File Settings Windows Help
All
Ready
1: 1
NUM

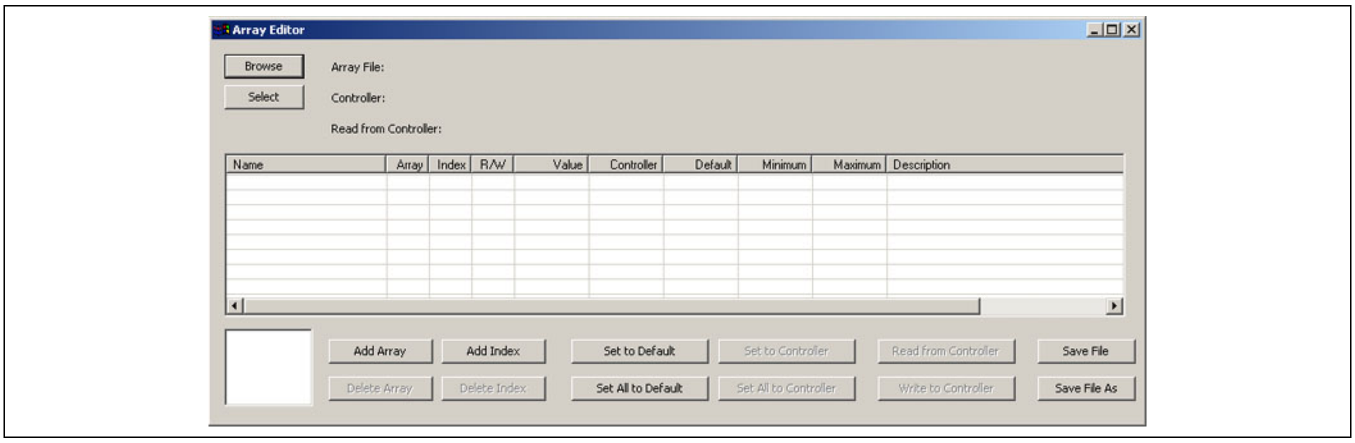
Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
Height	User	130	✓	50		100	10	1000	Maximum height of any unit
Width	User	131	✓	10		20	5	200	Maximum width of any unit
Weight	User	132	✓	5			1	20	Maximum weight of any unit
Number of units	User	135	✓	3000					Number of units to process
Quantity by Country									
Switzerland	0	1	✓	5		10	0	5000	This is the first value in array 0
Germany	0	2	☐	5		100	0	5000	This is the second value in array (
France	0	3	☐	5		50	0	5000	This is the third value in array 0
Canada	0	4	✓	5		1	0	5000	This is the fourth value in array 0
Other Important Values									
Extra	0	8	✓	123		4	-10000	10000	This is an extra value in array 0
Total	1	99	✓	250		300	200	500	This is the total
Test Value	3	5	✓	1		0	0	1	Test: 0 - No, 1 - Yes

Stand-alone application

The Array Editor can be started stand-alone by adding either an argument to the Windows command line.

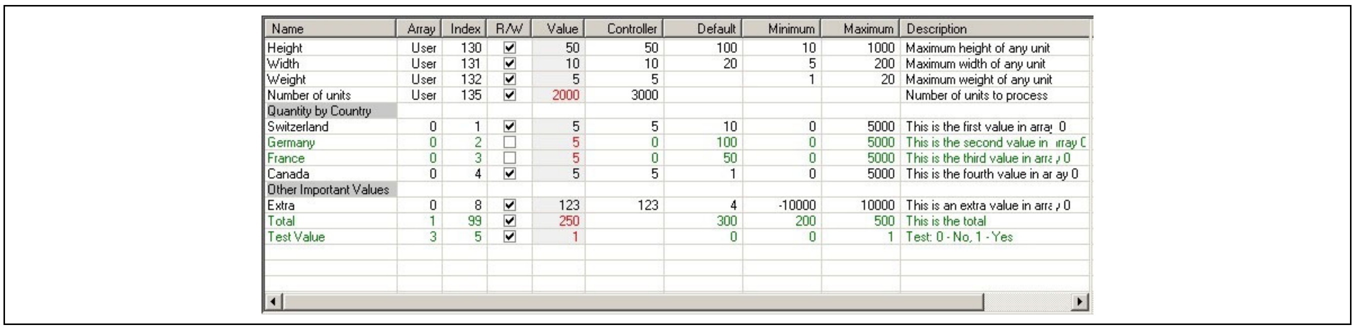
"/a" will start the Array Editor in User mode and
"/ae" will start the Array Editor in Expert mode.

In stand-alone application, the Array Editor will always start without a controller being connected and without an array definition file being loaded. The user must explicitly connect to a controller and explicitly load an array definition file.



☐ Array Editor List

Controller arrays and parameters are displayed in a list similar to that shown below.



Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
Height	User	130	✓	50	50	100	10	1000	Maximum height of any unit
Width	User	131	✓	10	10	20	5	200	Maximum width of any unit
Weight	User	132	✓	5	5		1	20	Maximum weight of any unit
Number of units	User	135	✓	2000	3000				Number of units to process
Quantity by Country									
Switzerland	0	1	✓	5	5	10	0	5000	This is the first value in array 0
Germany	0	2	☐	5	0	100	0	5000	This is the second value in array (
France	0	3	☐	5	0	50	0	5000	This is the third value in array 0
Canada	0	4	✓	5	5	1	0	5000	This is the fourth value in array 0
Other Important Values									
Extra	0	8	✓	123	123	4	-10000	10000	This is an extra value in array 0
Total	1	99	✓	250		300	200	500	This is the total
Test Value	3	5	✓	1		0	0	1	Test: 0 - No, 1 - Yes

One or more rows in the list can be selected using any of the normal Windows list selection methods: Clicking, [Shift]-clicking, or [Ctrl]-clicking.

Values can be edited (as far as allowed) by double-clicking on them.

Values that have been changed by the user will be displayed using a bold font. Values that are invalid for any reason will be displayed with a red background. If the user pauses the mouse over an invalid value, then a popup window will display a message describing the error.

Name

If the array index or parameter has been given a name, then it is displayed in this column. It will be blank if there is no name. Items in the list can be organized into "groups" and given a title (select the type "Group heading" using → *Add Index*). These titles will be displayed with a dark background. In Expert mode, this field can be edited by double-clicking on it.

Array

This column identifies the array number or parameter type.

Index

This column identifies the array index or parameter number.

R/W

If a list item can be written to the controller, then this column will be checked. If the item is read-only, then the column will not be checked. The entire row in the list will be displayed in green if the item is read-only. This column can only be modified in Expert mode.

Value

This column contains the editable "user" values for items. Values will be displayed with a light background so that the column is clearly identifiable. If a value differs from the value currently stored on the controller, then the value will be display in red. The value may be blank if there is no known value.

Users can change these values simply by clicking on the value and entering a new value or by typing [Ctrl]-[V] to paste a value from the Windows clipboard.

Press the [Enter] key to accept the new value. Press the [Esc] key to discard the new value and keep the previous value. Clicking with the mouse anywhere other than within the small edit box will automatically be treated as if the [Enter] key had been pressed.

NB!:
Changed values are not written to the controller until the user explicitly requests that they be written.

Controller

If the Array Editor is currently connected to a controller, then this column will display the current controller values. If there is no controller, then this column will be blank. If a controller is connected but the value does not exist on the controller (for example, an array may not exist), then this column will be blank and the entire row will be displayed in green to indicate that the item is "read-only" and cannot be written to the controller.

Default

The default value is the value that the controller will assign to global and axis parameters when the controller is reset to the factory settings. The Array Editor extends the concept to the other parameters (i.e. user parameters and arrays) for the template .zba file that is being created.

If a default value is defined for the list item, then it is displayed here. Otherwise, this column is blank. In Expert mode, this field can be edited by double-clicking on it.

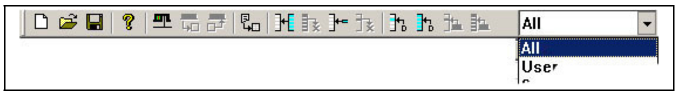
Minimum / Maximum

If a minimum and/or a maximum value is defined for the list item, then it is displayed here. Otherwise, this column is blank. In Expert mode, this fields can be edited by double-clicking on it.

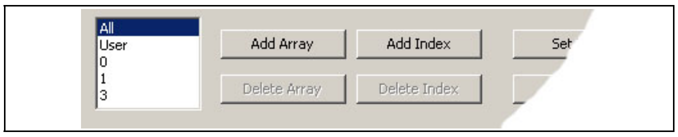
Description

Each list item may be given an optional description which is displayed in this column. In Expert mode, this field can be edited by double-clicking on it.

□ Array Editor Functions



If the Array Editor has been started from with the APOSS IDE, then all Array Editor functions will be available on the toolbar. The *Array Selection Box* is displayed rightmost.



If the Array Editor has been started stand-alone, then all functions will be available as normal dialog buttons. The *Array Selection Box* is displayed in the bottom left corner:

Connect / Select

Connect to a controller or switch an existing connection to another controller. This will cause the arrays and parameters to be automatically loaded from the new controller. The new values will be displayed in the "Controller" column.

In the stand-alone Array Editor, this function is provided by the → *Select* button.

Read from controller

This button will re-read all arrays and parameters from the controller. The new values will be displayed in the "Controller" column.

Write to controller

This button will cause all the values from the "Value" column in all arrays (even values in arrays that are not currently being displayed), to be written to the controller. Items that have no value (i.e. a blank in the "Value" column) will retain their existing values. Once the values have been written to the controller, the "Controller" column will be updated to reflect the new values on the controller.

Overlay descriptions

This button will allow list item names, descriptions, minimum values, maximum values, and default values to be copied from an existing array definition file into the Array Editor.

This function is not available in the stand-alone Array Editor.

Add array

add a new array. A dialog provides the definition of array number and size.

This function is not available in User mode. Arrays can only be added in Expert mode.

Delete array

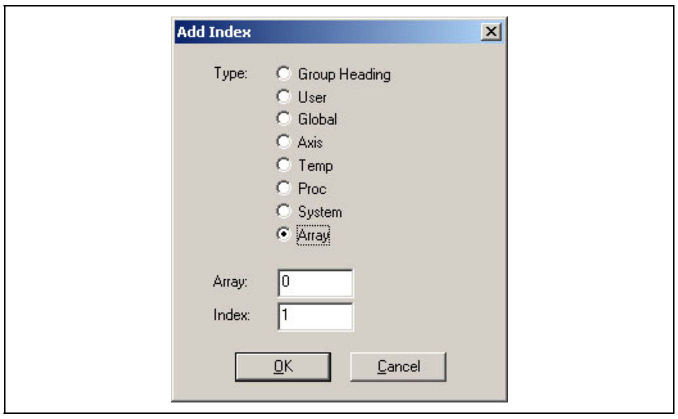
or pressing [Esc] delete the currently selected array. The array must be selected in the "Array selection list".

This function is not available in User mode. Arrays can only be deleted in Expert mode.

Add index

insert a new item into the list. The item will be inserted before the currently selected list item. A dialog provides the types.

This function is not available in User mode. New items can only be added in Expert mode.



Type: [✓ Array]
Array: 0
Index: 1

Delete index

or pressing [Esc] delete the currently selected list item, or click in stand-alone Array-Editor on → *Delete Index*.

This function is not available in User mode. Items can only be deleted in Expert mode.

Set to default

Highlight one or more array elements in the list and click → . This set the "value" of all currently selected list items to the corresponding default values. Any previous values are overwritten.

Note that values are not changed for those items which have a blank default value.

Set all to default

set the "value" of all currently visible list items to the corresponding default values. Any previous values are overwritten. Values are also not changed for any array not selected in the "Array selection box".

Note that values are not changed for those items which have a blank default value.

Set to controller

set the "value" of all currently selected list items to the current controller values.

Note that values are not changed for those items which have a blank controller value.

Set all to controller

set the "value" of all list items to the current controller values. Note that values are not changed for those items which have a blank controller value. Values are also not changed for any array not selected in the "Array selection box".

Array selection box

The array selection box is displayed either in the Array Editor toolbar or in the bottom left corner of the stand-alone Array Editor dialog. This allows the user to select either all arrays or individual arrays. It may be easier for the user to select individual arrays in order to work with a smaller list.

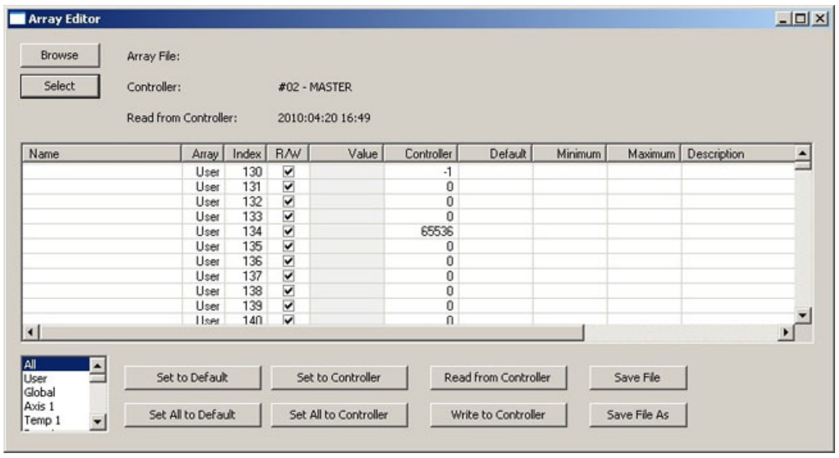
Save File / Save File As

Click on *File* → *Save* to save the current contents of the Array Editor as an array definition file; click on *File* → *Save as*, to save this file with a different name.

In the stand-alone Array Editor the → *Save File* or → *Save File As* button allows the current contents of the Array Editor to be saved as an array definition file.

☐ User mode WITHOUT an Array Definition File

The stand-alone Array Editor User mode dialog is shown below.



Array Editor

Array File:

Controller: #02 - MASTER

Read from Controller: 2010:04:20 16:49

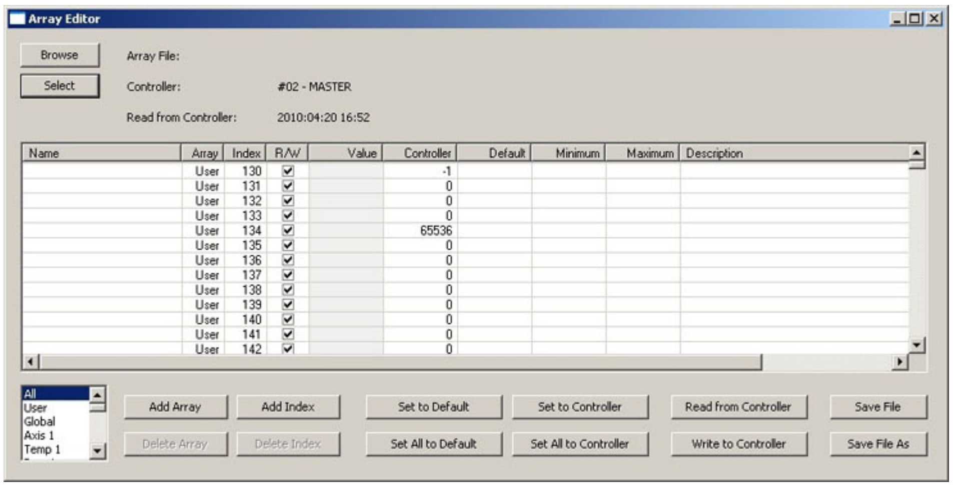
Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
	User	130	✓		-1				
	User	131	✓		0				
	User	132	✓		0				
	User	133	✓		0				
	User	134	✓		65536				
	User	135	✓		0				
	User	136	✓		0				
	User	137	✓		0				
	User	138	✓		0				
	User	139	✓		0				
	User	140	✓		0				

To use it WITHOUT an Array Definition File, follow these steps.

1. Press the → *Select* button. This will display the normal APOSS "*Select Controller*" dialog that will allow the user to set interface parameters, open interfaces, and choose a controller. Once the connection to the controller has been opened, the controller name will appear in the Array Editor dialog. The "Read from Controller" string will show the date and time when the arrays were read from the controller.
   All existing arrays in the controller (including the User array, Global parameter array, permanent and temporary Axis parameter arrays, Axis Process data arrays, etc.) will be read automatically and displayed in the dialog. The array number (or name for User, Global, and Axis arrays) is displayed in the "Array" column, the array index is displayed in the "Index" column, and the value is displayed in the "Controller" column. The array selection box in the bottom-left of the dialog will list all the arrays that were found on the controller.
2. At any time once a controller is open, pressing the → *Read from Controller* button will re-read all the arrays from the controller and display the current values in the dialog. The "Read from Controller" string will be updated to show the date and time when the arrays were read from the controller.
3. Click on any array in the array selection box to select that array for editing.
4. To change the value of an array element, click on the "Value" column of that array element. A small edit box will then be opened allowing the user to enter a new value. A new value may be entered either with the keyboard or by typing [Ctrl]-[V] to paste a value from the Windows clipboard.
   Note that some array elements may be read-only. These will be displayed green and cannot be changed by the user.
   Press the [Enter] key to accept the new value. Press the [Esc] key to discard the new value and keep the previous value. Clicking with the mouse anywhere *other* than within the small edit box will automatically be treated as if the [Enter] key had been pressed.
   Note that the "Value" column will contain only those values that have been changed. The values will be displayed in red if they are different from the values currently stored on the controller.
5. When editing is complete, press → *Write to Controller*. This will cause all the values from the "Value" column in all arrays, to be written to the controller. Array elements that have no value (i.e. a blank in the "Value" column) will retain their existing values. Once the arrays have been written to the controller, the "Controller" column will be updated to reflect the new values on the controller.
6. If desired, the *Select* button can be pressed again and a different controller selected.
7. When the Array Editor dialog is closed and any changes have been made, then the user will be prompted whether or not to save the changes in a file. Saving the changes will allow the changed values to be retrieved and reviewed at a later time (by using the *Browse* button).

□ Expert mode WITHOUT an Array Definition File

The stand-alone Array Editor Expert mode dialog is shown below. Use the toolbar icons, if the Array-Editor has been started within the APOSS-IDE.



Array File:

Controller: #02 - MASTER

Read from Controller: 2010:04:20 16:52

Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
	User	130	✓		-1				
	User	131	✓		0				
	User	132	✓		0				
	User	133	✓		0				
	User	134	✓		65536				
	User	135	✓		0				
	User	136	✓		0				
	User	137	✓		0				
	User	138	✓		0				
	User	139	✓		0				
	User	140	✓		0				
	User	141	✓		0				
	User	142	✓		0				

This dialog is used in the same way as the User mode dialog. The only difference is that *Add* and *Delete* buttons are now available. However, since no Array Definition File has been loaded, the Array Editor expects to work directly with the controller. As a result, the add and delete functions behave slightly differently than they would if a file was loaded:

Add Array

will allow a new array to be added after the last existing array. The user will be prompted for the array size.

NB!:
This new array will not be added to the controller until → *Write to Controller* is pressed.

Delete Array

will delete the currently selected array. Since "holes" (i.e. missing array numbers) are not allowed, all arrays following the deleted array will be "renumbered" in order to close the gap left by the deleted array. The array will not actually be deleted on the controller until → *Write to Controller* is pressed.

Note that only user arrays can be deleted; parameters cannot be deleted.

Add Index

will insert a single array index before the currently selected array index. If more than one array index is selected, then the new index is inserted before the first selected index. If no index is selected, then the new index is added at the end of the array. All array elements following the new index will be "renumbered" to make room for the new index.

Note that new indices can only be added to user arrays.

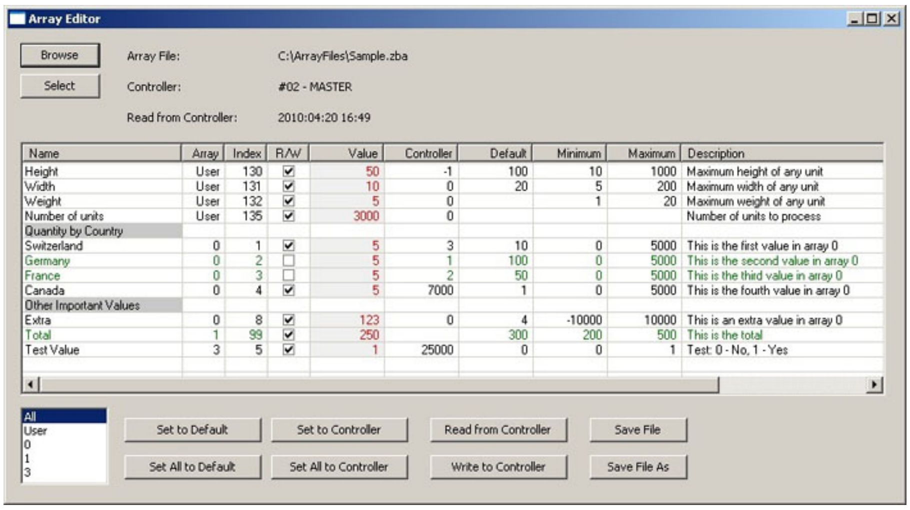
Delete Index

will delete all of the currently selected array elements. If all array elements are selected, then the array itself will be deleted. Since "holes" (i.e. missing array element numbers) are not allowed, all array elements following the deleted elements will be "renumbered" in order to close the gaps left by the deleted elements. The array elements will not actually be deleted on the controller until → *Write to Controller* is pressed.

Note that only user array elements can be deleted.

☐ User mode WITH an Array Definition File

The stand-alone Array Editor User mode dialog when an array definition file is being used is shown below.



Array File: C:\ArrayFiles\Sample.zba

Controller: #02 - MASTER

Read from Controller: 2010:04:20 16:49

Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
Height	User	130	✓	50	-1	100	10	1000	Maximum height of any unit
Width	User	131	✓	10	0	20	5	200	Maximum width of any unit
Weight	User	132	✓	5	0		1	20	Maximum weight of any unit
Number of units	User	135	✓	3000	0				Number of units to process
Quantity by Country									
Switzerland	0	1	✓	5	3	10	0	5000	This is the first value in array 0
Germany	0	2	☐	5	1	100	0	5000	This is the second value in array 0
France	0	3	☐	5	2	50	0	5000	This is the third value in array 0
Canada	0	4	✓	5	7000	1	0	5000	This is the fourth value in array 0
Other Important Values									
Extra	0	8	✓	123	0	4	-10000	10000	This is an extra value in array 0
Total	1	99	✓	250		300	200	500	This is the total
Test Value	3	5	✓	1	25000	0	0	1	Test: 0 - No, 1 - Yes

To use the Array Editor with an array definition file, proceed as follows:

1. Press the → *Browse* button. This will display the normal Windows "*Open File*" dialog and allow the user to select an array definition file. Array definition files will have the file extension ".zba". The contents of the file will then be displayed in the dialog. At this point, the "Controller" column will still be empty (since no arrays have been read from a controller) but all other columns may or may not contain values.
2. Press the → *Select* button and choose a controller in the same way as before. All arrays will then be read from the controller and the values will be "merged" into the list along with the existing values from the array definition file.
3. At any time once a controller is open, pressing → *Read from Controller* button will re-read all the arrays from the controller and display the current values in the dialog. The "Read from Controller" string will be updated to show the date and time when the arrays were read from the controller.

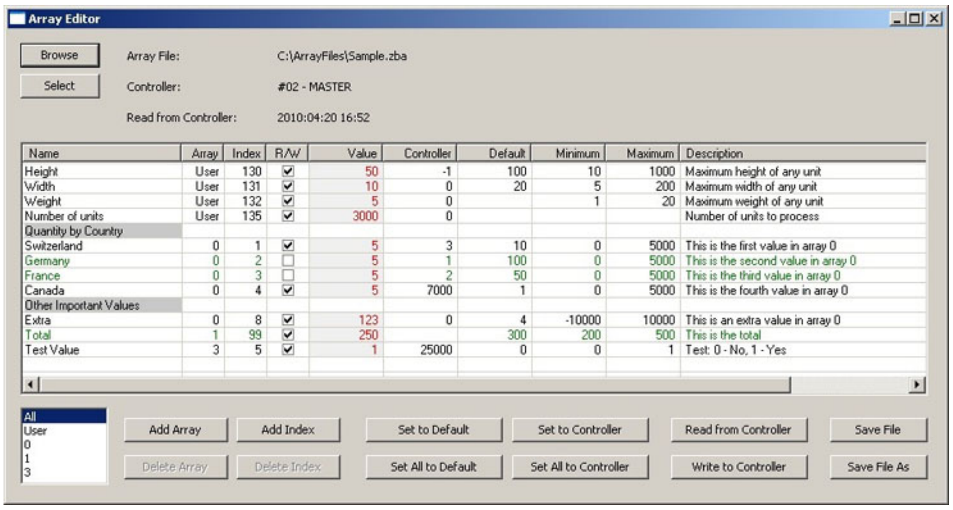
When the Array Editor dialog is closed and any changes have been made, then the user will be prompted whether or not to save the changes in a file. Saving the changes will allow the changed values to be retrieved and reviewed at a later time (by using the "Browse" button).

Please note the following points when using the Array Editor with an array definition file:

- Any array element (or any entire array) that does not have a definition within the array definition file, is not displayed and may not be accessed, even if it exists on the controller. The array definition file will allow access only to those array elements defined within the file.
- Any array element that is defined within the array definition file but that does not exist within the controller, is displayed in the list but it is disabled and will be displayed in green. It cannot be selected or changed.
- Array elements may be flagged as "read-only" in the array definition file. In this case, the check box in the "R/W" (i.e. Read/Write) column will be clear and the array element will be displayed in green. Read-only array elements cannot be changed.
- Group headings will be displayed if they have been defined within the file. These are displayed with a gray background in the "Name" column. These rows in the list cannot be selected. The array element definitions within a group may correspond to different arrays; all definitions in a group need not come from the same array.

☐ Expert mode WITH an Array Definition File

The stand-alone Array Editor Expert mode dialog when an array definition file is being used is shown below. Use the toolbar icons, if the Array-Editor has been started within the APOSS-IDE.



Array File: C:\ArrayFiles\Sample.zba

Controller: #02 - MASTER

Read from Controller: 2010:04:20 16:52

Name	Array	Index	R/W	Value	Controller	Default	Minimum	Maximum	Description
Height	User	130	✓	50	-1	100	10	1000	Maximum height of any unit
Width	User	131	✓	10	0	20	5	200	Maximum width of any unit
Weight	User	132	✓	5	0		1	20	Maximum weight of any unit
Number of units	User	135	✓	3000	0				Number of units to process
Quantity by Country									
Switzerland	0	1	✓	5	3	10	0	5000	This is the first value in array 0
Germany	0	2	☐	5	1	100	0	5000	This is the second value in array 0
France	0	3	☐	5	2	50	0	5000	This is the third value in array 0
Canada	0	4	✓	5	7000	1	0	5000	This is the fourth value in array 0
Other Important Values									
Extra	0	8	✓	123	0	4	-10000	10000	This is an extra value in array 0
Total	1	99	✓	250		300	200	500	This is the total
Test Value	3	5	✓	1	25000	0	0	1	Test: 0 - No, 1 - Yes

This dialog is used in the same way as the User mode dialog. The difference is that *Add* and *Delete* buttons are now available and that columns other than the "Value" column may be edited. As well, group headings and disabled list items (i.e. items that do not exist on the controller) may be selected. This allows those lines to be selected for the purposes of adding and deleting index lines. However, since an Array Definition File has been loaded, the Array Editor expects to work with the file rather than with the controller. As a result, the add and delete functions behave slightly differently than they would if a file was not loaded:

Add Array

will allow a new array to be added. The user will be prompted for the array number and array size. If no arrays have been read from the controller, then the new array elements will all be enabled. If arrays have been read from the controller, then the new arrays elements will be enabled or disabled according to whether or not they exist in the arrays read from the controller.

Note that adding arrays will add the arrays to the array definition file only; the arrays will not be added to the controller.

Delete Array

will delete the currently selected array from the array definition file. The array will not be deleted from the controller. Deleting an array in this case, is simply equivalent to removing user access to the array.

NB!:
When the array definition file is being edited, User, Global, Axis, etc., arrays can be deleted.

Add Index

will insert a single array element before the currently selected array element. If more than one array element is selected, then the new element is inserted before the first selected element. If no element is selected, then the new element is added at the end of the entire list. A dialog will be displayed allowing the user to specify exactly what kind of element is to be added.

Delete Index

will delete all of the currently selected array elements from the array definition file. The array elements on the controller will not be changed. Deleting array elements in this case, is simply equivalent to removing user access to those elements.

NB!:
Note that when the array definition file is being edited, User, Global, Axis, etc., array elements can be deleted.

□ Creating an Array Definition File

Array definition files are simple text files with the file extension ".zba". Array definition files can be created directly using the Array Edit Expert mode. However, they are designed so that they may also be created using common spreadsheet programs. As well, they may be created and edited, with care, using simple text-editor programs.

To create a file using the Array Editor, follow these steps:

1. Start the Array Editor.
2. Click the → *Select* button and connect to a controller. All arrays will then be automatically read from the controller.
3. Click the → *Save As* button and write the array definition file to disk. This is important since it switches the Array Editor into "WITH File" mode.
4. Edit the array elements as required to add names, descriptions, minimum values, maximum values, etc.
5. Click the → *Save* button when complete.

An example array definition file is shown below. The file will contain one line per definition. Each definition will consist of one or more "items", separated by either comma characters or semicolon characters. The first item in each line will be a "type" and can be one of the following values:

Date	The following item is the date that will be displayed in the dialog's "Read from Controller" heading.
Widths	The following items are the column widths to be used in the display.
User	Defines a user array index.
Global	Defines a global parameter.
System	Defines a system process data parameter.
Axis	Defines a permanent axis parameter.
Temp	Defines a temporary axis parameter.
Proc	Defines an axis process data parameter.
Array	Defines a normal array index.
Group	Defines a group heading to be displayed in the list.

Any line in the file that does not contain a recognized item type will be treated as a comment line and ignored.

Following the item type, there should be one item for each column in the Array Editor list with the exception of the Controller column. The Controller column is not saved in the file. These items must be ordered in exactly the following order:

Array, Index, Name, Value, Default, Minimum, Maximum, Description, and Read/Write flag.

The Array column should be empty for User and Global items, it should contain the 1-based axis number for permanent and temporary Axis items and for Axis Process data items, and it should contain the 0-based array number for Array items.

For Group items, the Array column determines when the heading is to be displayed and should contain one of the following:

-1 - Always displayed.

-2 - Displayed only when the "User" array (or "All") is selected.

-3 - Displayed only when the "Global" array (or "All") is selected.

-4 - Displayed only when the "System" array (or "All") is selected.

-100-n - Displayed only when "Axis n" (or "All") is selected (e.g. -101 for axis 1).

-200-n - Displayed only when "Temp n" (or "All") is selected (e.g. -201 for axis 1).

-300-n - Displayed only when "Proc n" (or "All") is selected (e.g. -301 for axis 1).

0-based array number - Displayed only when the corresponding array (or "All") is selected.

The Index column should be 0 for Group items.

The Name and Description strings may contain commas, semicolons, and blanks but, if so, then the string must be enclosed in double quotes. These strings must not contain double quote characters.

The Read/Write flag may be either 0 (for Read-only) or 1 (for Read/Write).

Example of an Array Definition File

```
Date,2009:07:27 10:52
Widths,150,40,40,40,70,70,70,70,70,365
This is a comment line.
User,,130,Height,50,100,10,1000,"Maximum height of any unit",1
User,,131,Width,10,20,5,200,"Maximum width of any unit",1
User,,132,Weight,5,,1,20,"Maximum weight of any unit",1
User,,135,"Number of units",3000,,,,"Number of units to process",1
***** Another comment line *****
Group,0,0,"Quantity by Country",,,,,,0
Array,0,1,Switzerland,5,10,0,5000,"This is the first value in array 0",1
Array,0,2,Germany,5,100,0,5000,"This is the second value in array 0",0
Array,0,3,France,5,50,0,5000,"This is the third value in array 0",0
Array,0,4,Canada,5,1,0,5000,"This is the fourth value in array 0",1
Group,-1,0,"Other Important Values",,,,,,0
Array,0,8,Extra,123,4,-10000,10000,"This is an extra value in array 0",1
Array,1,99,Total,250,300,200,500,"This is the total",1
Array,3,5,"Test Value",1,0,0,1,"Test: 0 - No, 1 - Yes",1
```

If an array definition file is being created using a spreadsheet, then the spreadsheet columns must be set up exactly as described above. The spreadsheet can then be saved as a "CVS (Comma delimited)" file type. This will generate the correct comma-separated or semicolon-separated file. The file should be then be renamed to give it the ".zba" file extension.

□ APOSS Oscilloscope

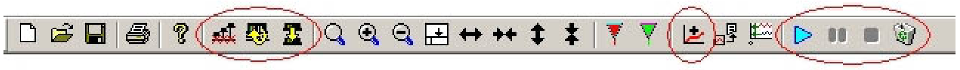
The APOSS Oscilloscope is a tool for recording and graphically displaying values from one or more controllers while those controllers are operational and executing programs. It is intended to be used as an aid in debugging prototype applications, diagnosing operational problems, and tuning the internal controller parameters for optimal system performance.

Operating the Oscilloscope basically involves the following steps:

1. Deciding what type of Oscilloscope is to be used.
2. Starting the Oscilloscope (except the TESTSETP Oscilloscope, q. v.).
3. Specifying the "curves" (i.e. the values, variables, states, etc.) that are of interest.
4. Starting the test (i.e. executing the application on the controller). The data for those curves will then be collected automatically.
5. Analyzing the resulting curves.
6. Making whatever changes are appropriate (i.e. to the curve definitions, to the controller parameters, to the application program, etc.) and repeating the test.

The exact details of steps 3 and 4 depend on the type of oscilloscope being used. These are described below. The last two steps are the responsibility of the user.

The following generic toolbar buttons are provided to assist the user in steps 3 and 4 (defining curves and starting tests).



Close all - Close the connections to all controllers on all interfaces.

Note that this will also close the connections to controllers in other windows (e.g. in any open APOSS Windows).

Reconnect all curves defined in the Oscilloscope to the currently defined default controller. (See the *Settings*→ *Interface* menu command in Chapter "PC Software Interface".)

If the curves were previously associated with different controllers, then the connection will be changed to the new controller.

Switch all - Select a new controller and switch all curves in the Oscilloscope to this new controller.

Add curve - Add a new curve to the Oscilloscope. This is described in detail later.

Start an oscilloscope "run". This will start the recording of curve data.

Pause an oscilloscope "run". This will pause the recording of curve data. The run can be started again using the "Start" toolbar button.

Stop an oscilloscope "run". This will stop the recording of curve data. If curve data is being saved on the controller, then it will be uploaded into the Oscilloscope at this time.

Delete all the data saved for all the curves. This includes curves that are not currently being displayed.

□ Oscilloscope Versions

There are four versions of the Oscilloscope available to users. Each version is designed for a different use but all versions basically operate in a similar manner and have a similar appearance. The *Free Run, Single Shot,* and *TESTSETP Oscilloscopes* are designed for debugging and diagnostics. The *Tune Oscilloscope* is designed for optimizing system performance.

Free Run Oscilloscope

The Free Run Oscilloscope is a tool designed specifically for diagnosing operational problems in running systems. However, it can also be used for debugging prototype applications. It is referred to as “Free Run” because the application on the controller can run freely without regard to how the Oscilloscope is being used or that it is being used at all. The impact of the existence and usage of the Oscilloscope on the running application has been minimized as much as possible. This allows the controller to behave as it would in actual operational situations.

Values are recorded and displayed in real time during the actual test. This information is normally collected using PDO messages from the controller.

NB!:
Note that PDO messages are not supported by some older controllers and by some connection interface types.

Free Run Oscilloscope in Polling Mode

In certain circumstances, the user may be unable or unwilling to allow the Oscilloscope to use PDO messages. Reasons for this include the following:

- The running application may already be using all available PDO message types. In this case, there will be no free PDO message type that the Oscilloscope can use to receive curve data from the controller.
- The controller may have an older version of firmware installed that does not support the dynamic configuration of PDO messages. The Oscilloscope will then be unable to configure the required PDO messages.
- The controller may be connected using a connection interface type that does not support PDO messages.

In these cases, the Free Run Oscilloscope either cannot be used or must be used in the slower “polling” mode. Note that using polling can have a significant impact on the performance and response of the controller and the communication network.

If the Free Run Oscilloscope cannot be used, then it may be possible to use the Single Shot Oscilloscope as an alternative.

Single Shot Oscilloscope

The Single Shot Oscilloscope is designed more for debugging prototype applications than it is for diagnosing operational problems in running systems. However, it can be used for both. It can also be used in some situations where the Free Run Oscilloscope cannot be used.

It is referred to as “Single Shot” because the curve data is recorded on the controller itself and then uploaded to the Oscilloscope after the test is complete.

Values are recorded on the controller in a TESTSETP array. The allocation of this array can be done either manually by the user (i.e. by defining a user array for the purpose) or automatically using available free memory in the controller. In either case, the *Single Shot Oscilloscope* may affect the memory management of the application program. The TESTSETP array will be automatically configured by the *Single Shot Oscilloscope* when recording begins.

NB!:
Some older controllers do not support the automatic configuration of TESTSETP arrays. In these cases, the Single Shot Oscilloscope cannot be used; the *TESTSETP Oscilloscope* should be used instead.

TESTSETP Oscilloscope

The TESTSETP Oscilloscope is designed to display user-defined TESTSETP arrays created on the controller (i.e. with the TESTSETP command). As with the Single Shot Oscilloscope, values are recorded using a TESTSETP array and uploaded to the Oscilloscope after the test is complete. However, the TESTSETP Oscilloscope relies on users configuring their own TESTSETP arrays from within their application programs. The arrays are not automatically configured by the Oscilloscope; the Oscilloscope simply reads already existing TESTSETP arrays that have been created by the user's application.

The TESTSETP Oscilloscope is useful in the following situations:

- Applications where the user has created a customized TESTSETP array from within the application program. This includes the use of multiple recordings within a single TESTSETP array.
- Testing controllers that contain firmware that is too old to support the automatic configuration of TESTSETP arrays. In this case, the Single Shot Oscilloscope cannot be used.

Since the TESTSETP Oscilloscope simply reads existing arrays, there is no need to start and stop recording. Hence, the TESTSETP Oscilloscope operates slightly differently from the Single Shot and Free Run Oscilloscopes. The steps are as follows:

1. Before the TESTSETP Oscilloscope is started, the user must execute an APOSS program that records the desired data using the various TESTSET commands. This program can be started in the normal way by using *Development* → *Execute* [F5].
2. After the test has been completed, the TESTSETP Oscilloscope can be started. The TESTSETP Oscilloscope will then automatically attempt to read the TESTSETP array from the controller.

NB!:
In previous APOSS versions this functionality was available using the *Testrun* → *Display recording* menu command. Starting with MCO 5.00, "*Display recording*" has been replaced by the TESTSETP Oscilloscope.

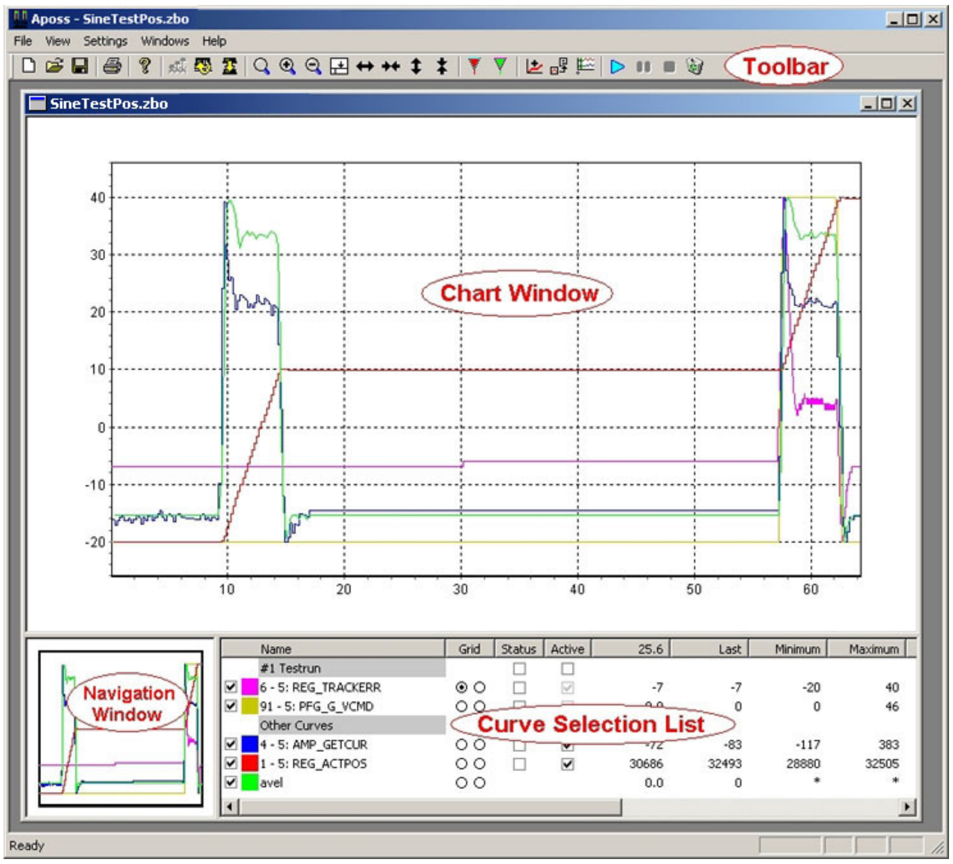
The TESTRUN functionality that existed in previous APOSS versions, allowed the results of *Execute testrun* and *Display recording* commands to be saved in ".txt" files. These old ".txt" files can still be read and displayed by newer versions of APOSS. See Reading Old Testrun Files.

Tune Oscilloscope

This Oscilloscope is designed to assist the user in "tuning", or optimizing, the controller's internal parameters. This is a necessary step for ensuring the best and smoothest possible performance of the controller in each specific application.

NB!:
In previous APOSS versions this functionality was available using the *Testrun* → *Execute testrun* menu command. Starting with MCO 5.00, "*Execute testrun*" has been replaced by the "Tune Oscilloscope"*.*

□ Oscilloscope Window



	Name	Grid	Status	Active	25.6	Last	Minimum	Maximum
	#1 Testrun		☐	☐				
✓ 🟪	6 - 5: REG_TRACKERR	● ○	☐	✓	-7	-7	-20	40
✓ 🟨	91 - 5: PFG_G_VCMD	○ ○	☐		0.0	0	0	46
	Other Curves							
✓ 🟦	4 - 5: AMP_GETCUR	○ ○	☐	✓	-72	-83	-117	383
✓ 🟥	1 - 5: REG_ACTPOS	○ ○	☐	✓	30686	32493	28880	32505
✓ 🟩	avel	○ ○			0.0	0	*	*

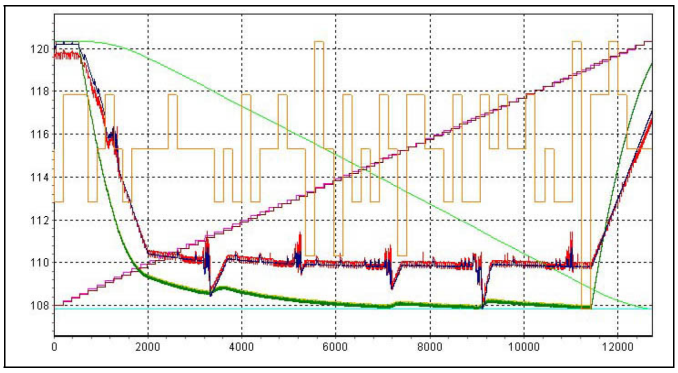
The layout of the Oscilloscope Window is shown above. Note that when an Oscilloscope Window is active, then the normal APOSS menu and toolbar are replaced by a custom menu and toolbar for the Oscilloscope.

The Oscilloscope Window as a whole can be resized using the standard Microsoft Windows mechanisms. As well, the individual sub-windows can be resized by dragging the "splitter" bars that separate the windows, using the left mouse button.

Note that if the windows are made too small, then the Oscilloscope will stop displaying the charts rather than continue to "shrink" the charts past the point where they become unusable.

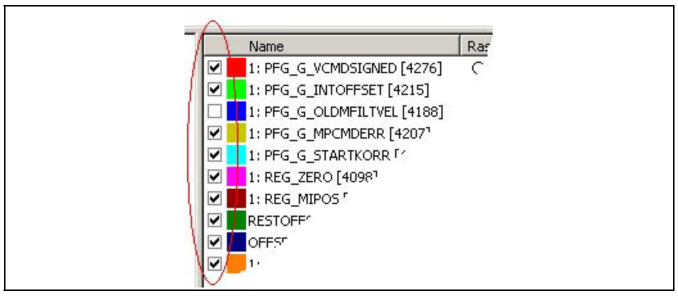
Chart Window

The *Chart Window* is the main graphical display window for the Oscilloscope. The curves representing the collected data values are displayed here. The chart displayed in this window can be zoomed in and out, panned, and modified in various ways in order to see more or less detail in the curves.



The figure above shows a sample chart. Each colored curve in the chart represents one value, measured over time, during the execution of a "test run". The X-axis of the chart represents time and is usually measured in milliseconds. Time increases from left to right in the chart. Note that the time is derived from the controller's internal clock. As a result, test runs will not start at time 0. The "Shift to 0" toolbar button can be used to reset the start time to 0, making the chart more readable.

Individual curves can be shown or hidden simply by clicking on the checkboxes in the "Curve Selection List" (see the figure). Other properties of the curves (e.g. color, scale, etc.) can also be set using the "Curve Selection List".



		Name	Ra
✓	🟥	1: PFG_G_VCMDSIGNED [4276]	
✓	🟩	1: PFG_G_INTOFFSET [4215]	
	🟦	1: PFG_G_OLDMFILTVEL [4188]	
✓	🟨	1: PFG_G_MPCMDERR [4207	
✓	🟦	1: PFG_G_STARTKORR	
✓	🟪	1: REG_ZERO [4098	
✓	🟥	1: REG_MIPOS	
✓	🟩	RESTOFF	
✓	🟦	OFFS	
✓	🟧	1:	

Using the Mouse in the Chart Window

The mouse buttons can be used as follows in the Chart Window:

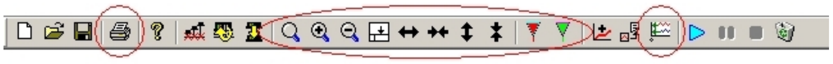
Left button (Zoom) - Zoom in to an area of interest in the chart by pressing the left mouse button, moving the mouse across the area of interest, and then releasing the mouse button.

NB!:
Note that the selected rectangle will be stretched so that it is exactly displayed in the window. If stretching is not desired, then press and hold the [Ctrl] key before pressing the left mouse button. Use the *Zoom Out* or *Zoom Reset* toolbar buttons to zoom back out again.

Right button (Pan) - Pan across the chart (i.e. move the chart in the window) by pressing the right mouse button, moving the mouse, and then releasing the mouse button.

Toolbar Buttons

The toolbar buttons associated with the Chart Window are shown in the following figure:



Print the chart.

Zoom Reset - Redisplay the entire chart.

Zoom In so that less of the chart (and more detail) is displayed.

Zoom Out so that more of the chart is displayed.

Splitter adjust – Adjust the splitter so that all of the curves are visible in the "Curve Selection List".

Stretch the time axis (X-axis) so that less time (and more detail) is displayed.

Shrink the time axis (X-axis) so that more time is displayed.

Stretch the vertical axis (Y-axis) so that more detail is displayed.

Shrink the vertical axis (Y-axis) so that less detail is displayed.

Cursor A - Display or hide cursor "A".

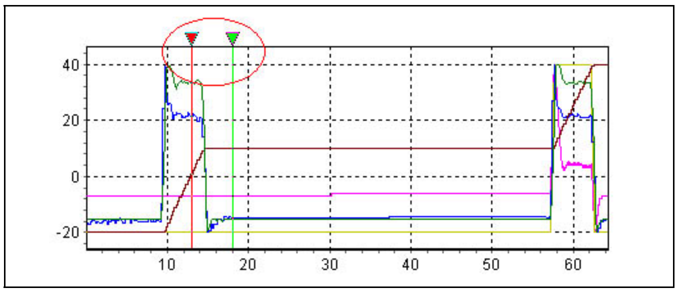
Cursor B - Display or hide cursor "B".

Shift to 0 - Shift the time axis (X-axis) so that curves start at time 0. Clicking again will toggle the start time back to its original value.

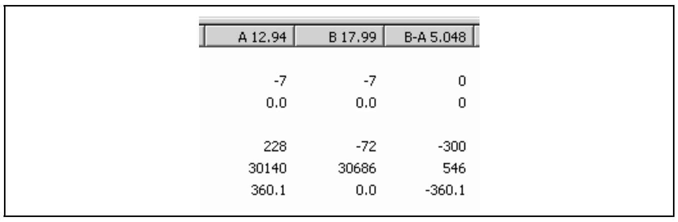
<u>Using A and B Cursors</u>

The Oscilloscope provides two "cursors": A and B. These can be used as a convenience by the user to mark particular times along the X-axis. They can be shown or hidden by clicking on the Cursor A and Cursor B toolbar buttons or by using the *View* → *Cursor A* and → *Cursor B* menu commands.

The cursors will appear as vertical lines through the chart and each will have a small triangular "handle" at the top end. This is shown in the figure below. The cursors can be moved left or right by clicking on the triangular handle with the <u>right</u> mouse button and dragging the cursor left or right.



When the cursors are used, the value of each curve at the position of the cursor is shown in the "Curve Selection List". The column headings will contain the X coordinate (time) of the position of each cursor. As well, there will be a column containing the difference between the curve values at the cursor positions (i.e. the value at cursor "B" minus the value at cursor "A"). This is shown in this figure:

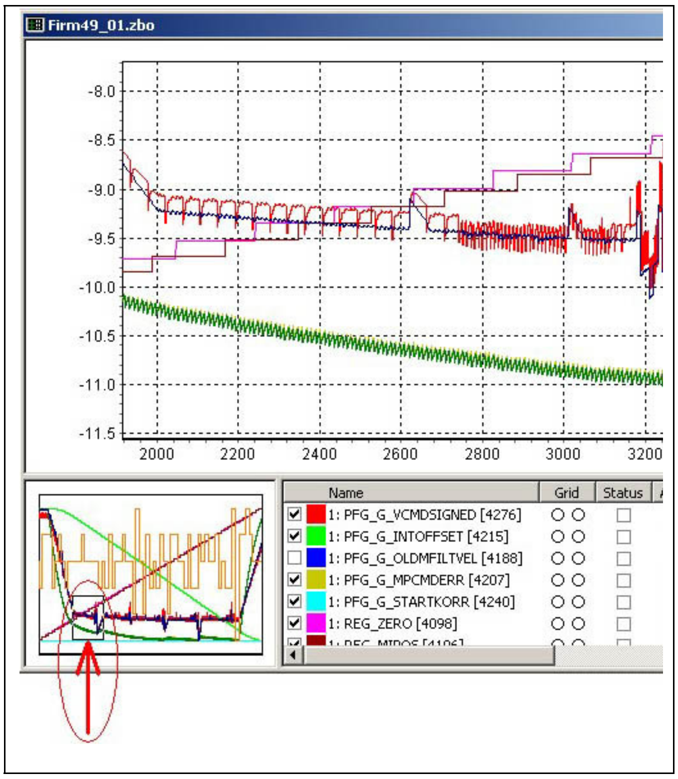


A 12.94	B 17.99	B-A 5.048
-7	-7	0
0.0	0.0	0
228	-72	-300
30140	30686	546
360.1	0.0	-360.1

Navigation Window

The *Navigation Window* displays an overview of the entire chart. If the main Chart Window has been zoomed, then this window allows the user to see and move the position of the zoomed area relative to the entire chart.

The zoomed area of the main chart will appear as a black outline rectangle in the *Navigation Window*. This is shown in the figure below. If Cursor A and Cursor B are shown, then they will also be visible in the Navigation Window.



Firm49_01.zbo

	Name	Grid	Status
☑ 🟥	1: PFG_G_VCMDSIGNED [4276]	○ ○	☐
☑ 🟩	1: PFG_G_INTOFFSET [4215]	○ ○	☐
☐ 🟦	1: PFG_G_OLDMFILTVEL [4188]	○ ○	☐
☑ 🟨	1: PFG_G_MPCMDERR [4207]	○ ○	☐
☑ 🟦	1: PFG_G_STARTKORR [4240]	○ ○	☐
☑ 🟪	1: REG_ZERO [4098]	○ ○	☐
☑ 🟥	1: REC_MIPOS [4106]	○ ○	☐

Using the Mouse in the Navigation Window

Left button (Zoom) - Zoom in to an area of interest in the chart by pressing the left mouse button, moving the mouse across the area of interest, and then releasing the mouse button. The selected area will then be displayed in the Chart Window.

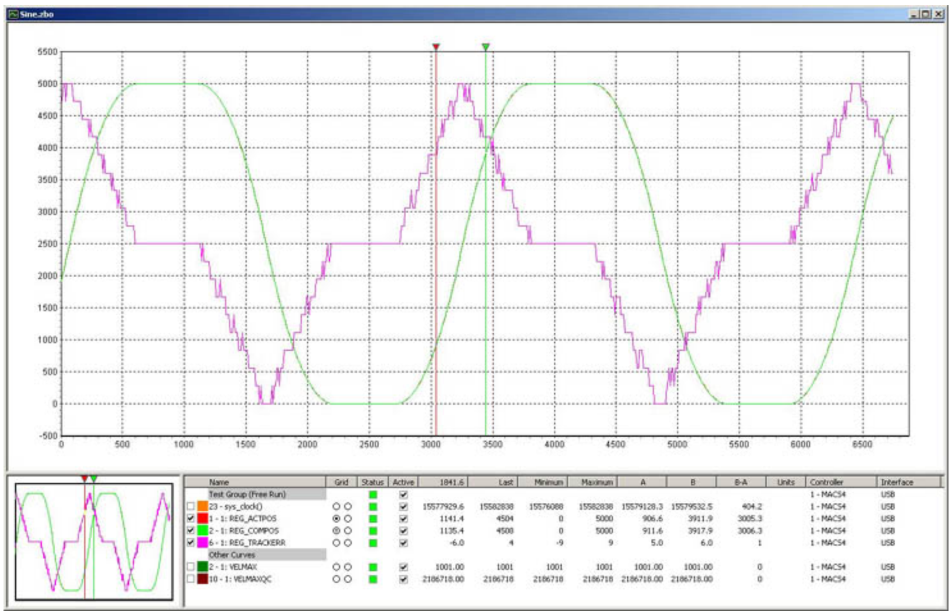
NB!:
Note that the selected rectangle will be stretched so that it is exactly displayed in the window. If stretching is not desired, then press and hold the [Ctrl] key before pressing the left mouse button.

Right button (Pan) - Pan across the chart (i.e. move the chart in the window) by placing the mouse anywhere within the black outline rectangle, pressing the right mouse button, moving the mouse, and then releasing the mouse button. The chart in the Chart Window will follow the movement of the black outline rectangle.

□ Curve Selection List

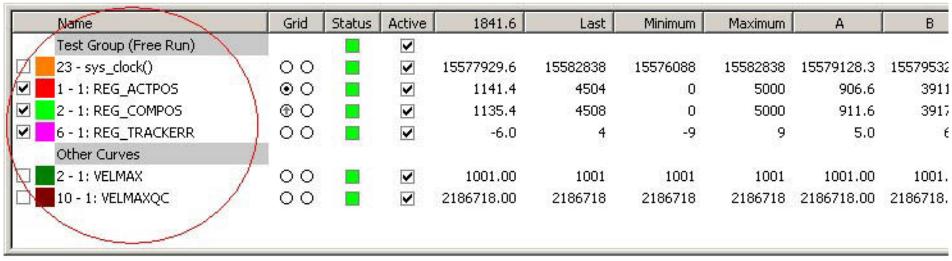
The "Curve Selection List" contains a list of all the curves that the user has defined and for which data have been collected, along with various details (i.e. values, variables, states, etc.) about each curve. All the properties of each individual curve, such as displaying, hiding, enabling, disabling, scaling, color, etc., are controlled from this window.

The setting of all display properties, data collection properties, etc., is done using this list. If the list is too long or too wide to fit within the window, then the associated scrollbars can be used to view other parts of the list. Column widths can also be changed by dragging the column header separators with the left mouse button. The figure below shows an example of the entire list.



	Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B	B-A	Units	Controller	Interface
	Test Group (Free Run)		🟩	✓									1 - MACS4	USB
☐	23 - sys_clock()	○ ○	🟩	✓	15577929.6	15582838	15576088	15582838	15579128.3	15579532.5	404.2		1 - MACS4	USB
✓	1 - 1: REG_ACTPOS	● ○	🟩	✓	1141.4	4504	0	5000	906.6	3911.9	3005.3		1 - MACS4	USB
✓	2 - 1: REG_COMPOS	● ○	🟩	✓	1135.4	4508	0	5000	911.6	3917.9	3006.3		1 - MACS4	USB
✓	6 - 1: REG_TRACKERR	○ ○	🟩	✓	-6.0	4	-9	9	5.0	6.0	1		1 - MACS4	USB
	Other Curves													
☐	2 - 1: VELMAX	○ ○	🟩	✓	1001.00	1001	1001	1001	1001.00	1001.00	0		1 - MACS4	USB
☐	10 - 1: VELMAXQC	○ ○	🟩	✓	2186718.00	2186718	2186718	2186718	2186718.00	2186718.00	0		1 - MACS4	USB

Name Column



	Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
	Test Group (Free Run)		🟩	✓						
☐	23 - sys_clock()	○ ○	🟩	✓	15577929.6	15582838	15576088	15582838	15579128.3	15579532
✓	1 - 1: REG_ACTPOS	● ○	🟩	✓	1141.4	4504	0	5000	906.6	3911
✓	2 - 1: REG_COMPOS	● ○	🟩	✓	1135.4	4508	0	5000	911.6	3917
✓	6 - 1: REG_TRACKERR	○ ○	🟩	✓	-6.0	4	-9	9	5.0	6
	Other Curves									
☐	2 - 1: VELMAX	○ ○	🟩	✓	1001.00	1001	1001	1001	1001.00	1001.
☐	10 - 1: VELMAXQC	○ ○	🟩	✓	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

The “Name” column consists of three parts:

Checkbox – If this is checked, then this curve will be displayed in the Chart and Navigation Windows. Otherwise, the curve is hidden. Left-clicking anywhere within the Name column will toggle the checkbox.

Color patch – This shows the color being used for the curve in the Chart and Navigation Windows. The color is one of the curve properties that can be changed by the user.

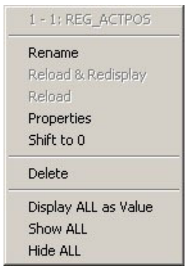
Name – The name of the curve. This can also be changed by the user. Names with a grey background will be headings that group together the following curves in the list.
Note that “Other Curves” is a special group and its name cannot be changed.

Popup Menu Functions for Name Column

Right-clicking anywhere in the Name column will display the popup menu:

The top item in the list is simply the curve name for reference only. It will always be grayed.

The other commands are as follows:



Rename – This command allows the user to rename the curve.

Reload & Redisplay – This command will “reload” the curve data from the controller and also redisplay the data in the Chart Window.
Note that this will not always be possible, depending on the type of curve. The command will be grayed if reloading is not possible.

Reload – This command will “reload” the curve data from the controller. Note that this will not always be possible, depending on the type of curve. The command will be grayed if reloading is not possible.

Properties – This command is used to view and/or modify the properties of a curve. See Modifying a Curve's Properties.

Shift to 0 – Curve data is collected based on the controller's system time. Since tests do not always start at time 0, this command allows the start time to be toggled between 0 and the actual start time. This is similar to the *Shift to 0* toolbar button except that it is limited to the single selected curve.

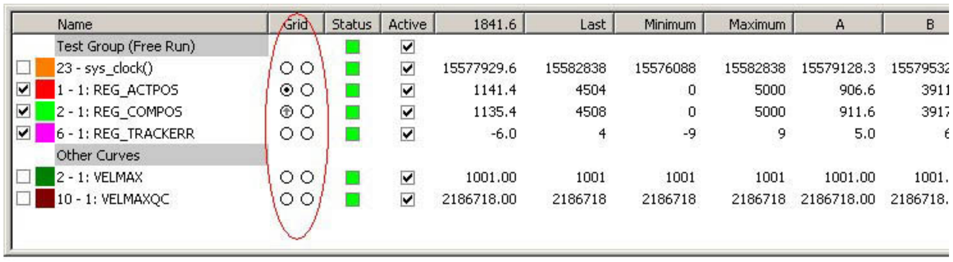
Delete – This command will delete the selected curve.

Display ALL as Value – This command will change the display interpolation style of all curves to “Value”. See Modifying a Curve's Properties.

Show ALL – This command will display all curves in the chart. It can be used to avoid displaying all curves one-by-one.

Hide ALL – This command will hide all curves in the chart. It can be used to avoid hiding all curves one-by-one.

Grid Column



Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
Test Group (Free Run)		🟩	✓						
☐ 🟧 23 - sys_clock()	○ ○	🟩	✓	15577929.6	15582838	15576088	15582838	15579128.3	1557953
✓ 🟥 1 - 1: REG_ACTPOS	● ○	🟩	✓	1141.4	4504	0	5000	906.6	3911
✓ 🟩 2 - 1: REG_COMPOS	● ○	🟩	✓	1135.4	4508	0	5000	911.6	391
✓ 🟪 6 - 1: REG_TRACKERR	○ ○	🟩	✓	-6.0	4	-9	9	5.0	6
Other Curves									
☐ 🟩 2 - 1: VELMAX	○ ○	🟩	✓	1001.00	1001	1001	1001	1001.00	1001.
☐ 🟥 10 - 1: VELMAXQC	○ ○	🟩	✓	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

The “Grid” column consists of two columns of radio buttons.

Clicking on the left radio button will cause the Y-axis of the Chart Window to be labeled on the left side with a scale corresponding to the selected curve. In the above example, the scale for "REG_ACTPOS" will be displayed. Clicking on the right radio button will be identical except that the Y-axis of the Chart Window will be labeled on the right side. The chart can be labeled on both the left and right sides at the same time.

NB!:
Note that all the other curves will still be displayed using a generic "top-to-bottom" scale so that the entire curve will be visible. It is important to remember that the scale of these other curves will NOT correspond to the scale drawn on the Y-axis.

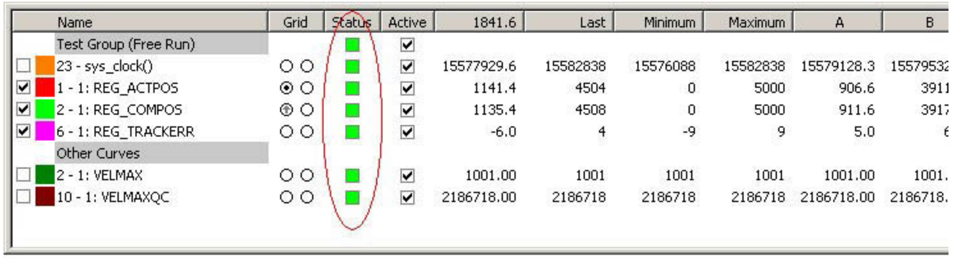
Since only one scale at a time can be drawn on either side of the chart, selecting a curve will cause any curve previously selected for that side to be deselected.

Sharing the Y-axis

In some cases, it is desirable to have two or more curves "share" the same Y-axis. For example, an "actual position" curve and a "commanded position" curve will be measured using the same units and should be displayed using the same scale in the Chart Window.

This can be done by right-clicking on the radio button. This will cause the radio button to display a small grey upward arrow. The selected curve will then share the same Y-axis as the curve that is currently being displayed on the chart's Y-axis. In the above example, "REG_COMPOS" shares the same Y-axis as "REG_ACTPOS". Right-clicking again will toggle the "share" attribute.

Status Column



Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
Test Group (Free Run)		🟩	✓						
☐ 23 - sys_clock()	○ ○	🟩	✓	15577929.6	15582838	15576088	15582838	15579128.3	1557953
✓ 1 - 1: REG_ACTPOS	● ○	🟩	✓	1141.4	4504	0	5000	906.6	3911
✓ 2 - 1: REG_COMPOS	○ ○	🟩	✓	1135.4	4508	0	5000	911.6	3917
✓ 6 - 1: REG_TRACKERR	○ ○	🟩	✓	-6.0	4	-9	9	5.0	6
Other Curves									
☐ 2 - 1: VELMAX	○ ○	🟩	✓	1001.00	1001	1001	1001	1001.00	1001.
☐ 10 - 1: VELMAXQC	○ ○	🟩	✓	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

The "Status" column shows the current status of the controller associated with the curve.

Note that not all curves need to be associated with the same controller; different curves can be associated with different controllers.

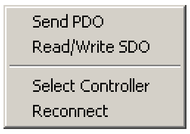
The Status column can have any of the following colors:

🟩	Controller is executing a program.
🟩	Controller is not executing a program.
🟥	Controller is in an error state.
■	Controller connection has been lost.
☐	Controller has been associated with this curve but is not currently connected.

This column will be empty for any curve that is not explicitly associated with a controller. For example, "Derived" curves are associated with other curves rather than with controllers.

Popup Menu Functions for Status Column

Right-clicking in this column will display the following popup menu of commands relating to the controller associated with the selected curve:



Send PDO – This command will allow the user to send a PDO message to the controller.

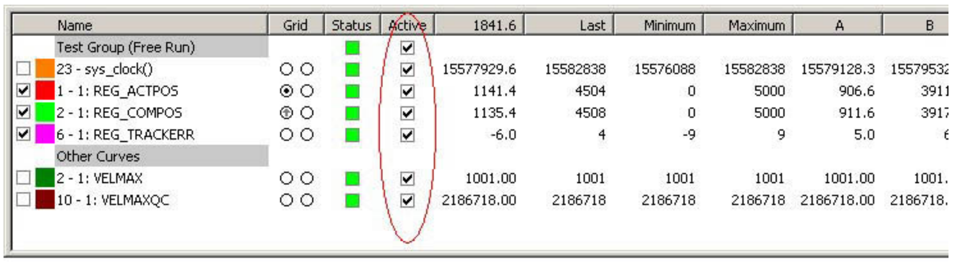
Read/Write SDO – This command will allow the user to read or write an SDO on the controller.

Select Controller – This command allows the user to select a different controller to be associated with the selected curve. Other curves that are associated with this controller, or other controllers, are not affected.

NB!:
In some cases, multiple curves <u>must</u> be connected to the same controller. For example, all curves that originate from the same TESTSETP array <u>must</u> be associated with the same controller since the array comes from a single controller! In this case, the entire group of curves will be changed when one curve is changed. The Controller and Interface columns will show the controller with which the curve is currently associated.

Reconnect – If the controller is not currently connected or the connection has been lost, then this command will try to reconnect to the controller.

Active Column



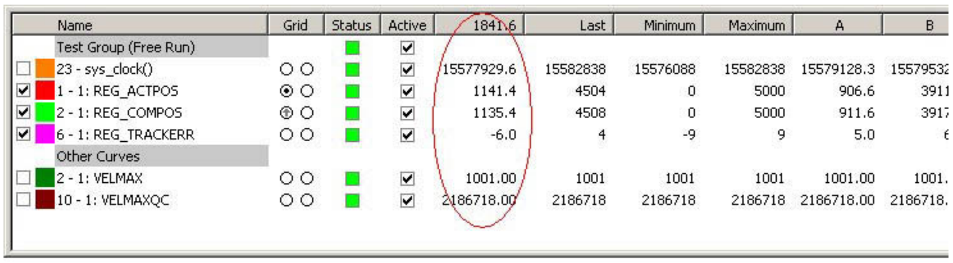
		Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
		Test Group (Free Run)		🟩	☑						
☐	🟧	23 - sys_clock()	○ ○	🟩	☑	15577929.6	15582838	15576088	15582838	15579128.3	15579532
☑	🟥	1 - 1: REG_ACTPOS	◉ ○	🟩	☑	1141.4	4504	0	5000	906.6	3911
☑	🟩	2 - 1: REG_COMPOS	◉ ○	🟩	☑	1135.4	4508	0	5000	911.6	3917
☑	🟪	6 - 1: REG_TRACKERR	○ ○	🟩	☑	-6.0	4	-9	9	5.0	6
		Other Curves									
☐	🟩	2 - 1: VELMAX	○ ○	🟩	☑	1001.00	1001	1001	1001	1001.00	1001.
☐	🟥	10 - 1: VELMAXQC	○ ○	🟩	☑	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

The “Active” column will contain a checkbox for those curves that are explicitly associated with a controller.

When the box is checked, then the Oscilloscope will enable data collection for that curve. Otherwise, data will not be collected for that curve. Clicking in the column will toggle the status of the checkbox.

Note that data is only collected by the Oscilloscope when an Oscilloscope “run” has been started. See “Starting the Oscilloscope”.

Mouse Position Column

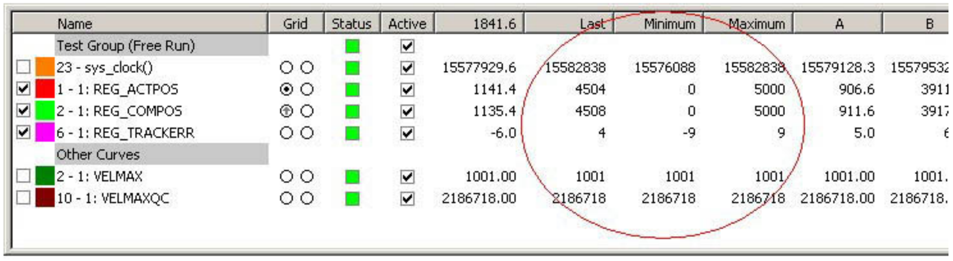


		Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
		Test Group (Free Run)		🟩	☑						
☐	🟧	23 - sys_clock()	○ ○	🟩	☑	15577929.6	15582838	15576088	15582838	15579128.3	15579532
☑	🟥	1 - 1: REG_ACTPOS	◉ ○	🟩	☑	1141.4	4504	0	5000	906.6	3911
☑	🟩	2 - 1: REG_COMPOS	◉ ○	🟩	☑	1135.4	4508	0	5000	911.6	3917
☑	🟪	6 - 1: REG_TRACKERR	○ ○	🟩	☑	-6.0	4	-9	9	5.0	6
		Other Curves									
☐	🟩	2 - 1: VELMAX	○ ○	🟩	☑	1001.00	1001	1001	1001	1001.00	1001.
☐	🟥	10 - 1: VELMAXQC	○ ○	🟩	☑	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

When the mouse is moved across the Chart Window, the value of each curve at the position of the mouse is shown in the Mouse Position column.

The column heading will contain the X coordinate (time) of the position of the mouse. If a value is shown as “*”, then the column is too narrow to display the entire value. In this case, the column can be widened by dragging the column separator in the header with the left mouse button.

Last, Minimum, and Maximum Columns



		Name	Grid	Status	Active	1841.6	Last	Minimum	Maximum	A	B
		Test Group (Free Run)		🟩	✓						
☐	🟧	23 - sys_clock()	○ ○	🟩	✓	15577929.6	15582838	15576088	15582838	15579128.3	15579532
✓	🟥	1 - 1: REG_ACTPOS	◉ ○	🟩	✓	1141.4	4504	0	5000	906.6	3911
✓	🟩	2 - 1: REG_COMPOS	◉ ○	🟩	✓	1135.4	4508	0	5000	911.6	3917
✓	🟪	6 - 1: REG_TRACKERR	○ ○	🟩	✓	-6.0	4	-9	9	5.0	6
		Other Curves									
☐	🟩	2 - 1: VELMAX	○ ○	🟩	✓	1001.00	1001	1001	1001	1001.00	1001.
☐	🟥	10 - 1: VELMAXQC	○ ○	🟩	✓	2186718.00	2186718	2186718	2186718	2186718.00	2186718.

These columns will show the values listed below.

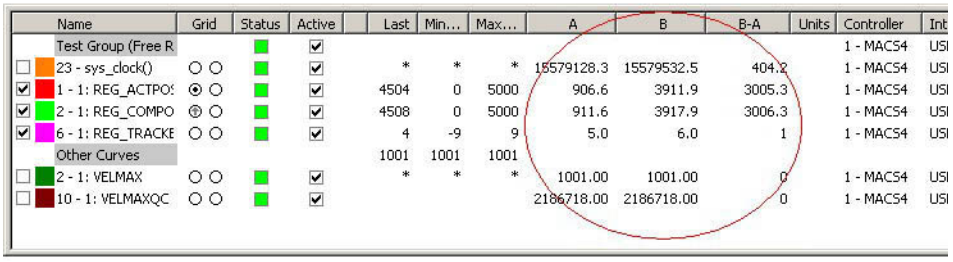
If a value is shown as "*", then the column is too narrow to display the entire value. In this case, the column can be widened by dragging the column separator in the header with the left mouse button.

"Last" - The last recorded value of each curve.

"Minimum" - The minimum recorded value of each curve.

"Maximum" - The maximum recorded value of each curve.

Cursor A/B Columns

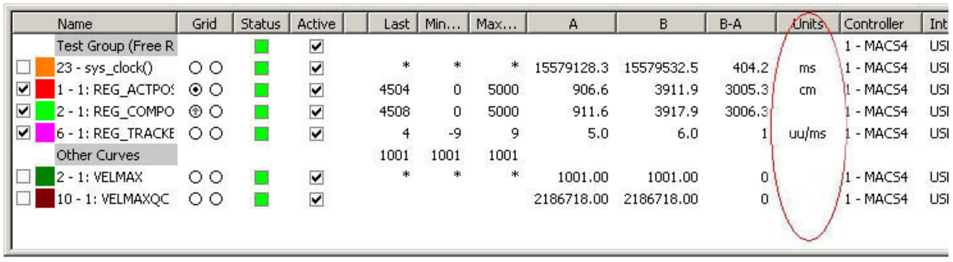


		Name	Grid	Status	Active		Last	Min...	Max...	A	B	B-A	Units	Controller	Int
		Test Group (Free R		🟩	✓									1 - MACS4	USI
☐	🟧	23 - sys_clock()	○ ○	🟩	✓		*	*	*	15579128.3	15579532.5	404.2		1 - MACS4	USI
✓	🟥	1 - 1: REG_ACTPOS	◉ ○	🟩	✓		4504	0	5000	906.6	3911.9	3005.3		1 - MACS4	USI
✓	🟩	2 - 1: REG_COMPO	◉ ○	🟩	✓		4508	0	5000	911.6	3917.9	3006.3		1 - MACS4	USI
✓	🟪	6 - 1: REG_TRACKE	○ ○	🟩	✓		4	-9	9	5.0	6.0	1		1 - MACS4	USI
		Other Curves					1001	1001	1001						
☐	🟩	2 - 1: VELMAX	○ ○	🟩	✓		*	*	*	1001.00	1001.00	0		1 - MACS4	USI
☐	🟥	10 - 1: VELMAXQC	○ ○	🟩	✓					2186718.00	2186718.00	0		1 - MACS4	USI

The column headings will contain the X coordinate (time) of the position of each cursor. When the cursors are used, the value of each curve at the position of the cursor is shown in the columns "A" an "B": As well, there will be a column containing the difference between the curve values at the cursor positions (i.e. the value at cursor "B" minus the value at cursor "A").

As well, the If a value is shown as "*", then the column is too narrow to display the entire value. In this case, the column can be widened by dragging the column separator in the header with the left mouse button.

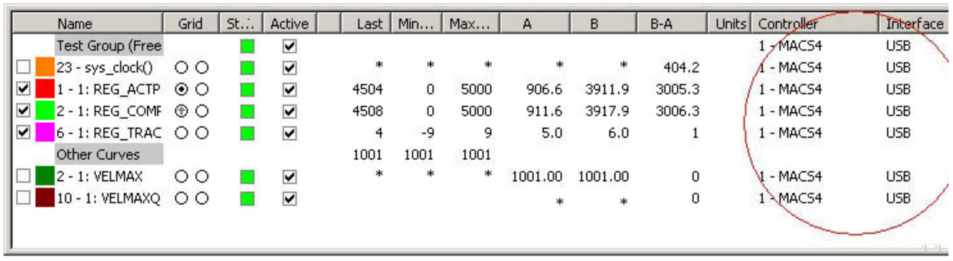
Units Column



		Name	Grid	Status	Active		Last	Min...	Max...	A	B	B-A	Units	Controller	Int
		Test Group (Free R		🟩	✓									1 - MACS4	USI
☐	🟧	23 - sys_clock()	○ ○	🟩	✓		*	*	*	15579128.3	15579532.5	404.2	ms	1 - MACS4	USI
✓	🟥	1 - 1: REG_ACTPOS	◉ ○	🟩	✓		4504	0	5000	906.6	3911.9	3005.3	cm	1 - MACS4	USI
✓	🟩	2 - 1: REG_COMPO	◉ ○	🟩	✓		4508	0	5000	911.6	3917.9	3006.3		1 - MACS4	USI
✓	🟪	6 - 1: REG_TRACKE	○ ○	🟩	✓		4	-9	9	5.0	6.0	1	uu/ms	1 - MACS4	USI
		Other Curves					1001	1001	1001						
☐	🟩	2 - 1: VELMAX	○ ○	🟩	✓		*	*	*	1001.00	1001.00	0		1 - MACS4	USI
☐	🟥	10 - 1: VELMAXQC	○ ○	🟩	✓					2186718.00	2186718.00	0		1 - MACS4	USI

An optional "Units" string can be defined by the user to each curve. This is described in Modifying a Curve's Properties. If defined, then the Units string is displayed in this column. This string is solely for information; it is not used by the Oscilloscope.

Controller and Interface Columns



	Name	Grid	St.:.	Active		Last	Min...	Max...	A	B	B-A	Units	Controller	Interface
	Test Group (Free		🟩	✓									1 - MACS4	USB
☐ 🟧	23 - sys_clock()	○ ○	🟩	✓		*	*	*	*	*	404.2		1 - MACS4	USB
✓ 🟥	1 - 1: REG_ACTP	● ○	🟩	✓		4504	0	5000	906.6	3911.9	3005.3		1 - MACS4	USB
✓ 🟩	2 - 1: REG_COMF	● ○	🟩	✓		4508	0	5000	911.6	3917.9	3006.3		1 - MACS4	USB
✓ 🟪	6 - 1: REG_TRAC	○ ○	🟩	✓		4	-9	9	5.0	6.0	1		1 - MACS4	USB
	Other Curves					1001	1001	1001						
☐ 🟩	2 - 1: VELMAX	○ ○	🟩	✓		*	*	*	1001.00	1001.00	0		1 - MACS4	USB
☐ 🟥	10 - 1: VELMAXQ	○ ○	🟩	✓					*	*	0		1 - MACS4	USB

If a controller is connected, then the "Controller" column will list the ID number (usually the CAN ID) and user-assigned name of the controller associated with each curve. If a controller is not connected, then only the ID number is shown.

The "Interface" column will list the connection interface description associated with each curve.

Modifying a Curve's Properties

Curve Properties can be viewed and modified by right-clicking on the name of the curve in the "Name" column of the "Curve Selection List" and then selecting the Properties command. This will display a dialog containing the following "tabs":

Display - Shows the various attributes affecting the display of the curve in the chart and the display of values in the list. This tab will not be present if the curve is not graphical (i.e. it cannot be drawn in the chart). For example, group headings contain no actual curve data and are never drawn.

Axis - Shows the various attributes affecting the Y-axis of the curve. This tab will also not be present if the curve is not graphical (e.g. group headings).

Source - Shows details of where and how the data for the curve is collected.

Curve Properties: Display

Color

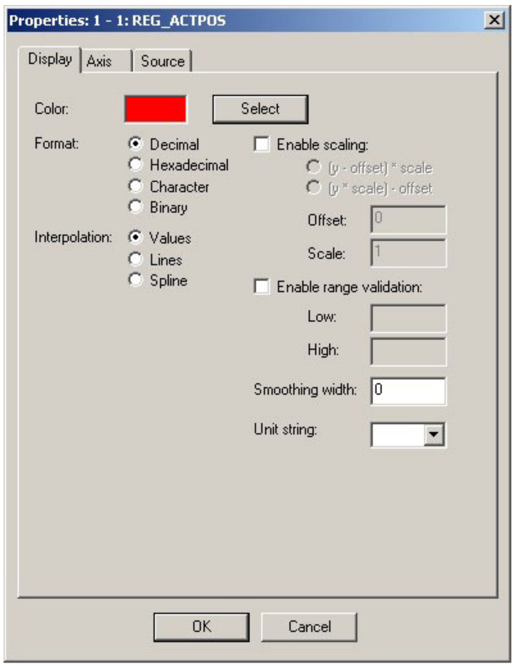
Click on the *Select* button to change the color of a curve in the Chart Window.

Format

This setting specifies how the curve values (i.e. mouse position, last, minimum, maximum, etc.) are to be displayed in the "Curve Selection List".

NB!:
Note that values should not be displayed as characters unless the controller is actually generating character values for that curve. If not, then blanks will usually be displayed.

Please note that the setting of *Interpolation* will also affect how curve values are displayed.



Properties: 1 - 1: REG_ACTPOS

Display | Axis | Source

Color: 🟥 Select

Format: [✓ Decimal] Hexadecimal Character Binary

☐ Enable scaling:
(y - offset) * scale
(y * scale) - offset
Offset: 0
Scale: 1

Interpolation: [✓ Values] Lines Spline

☐ Enable range validation:
Low:
High:

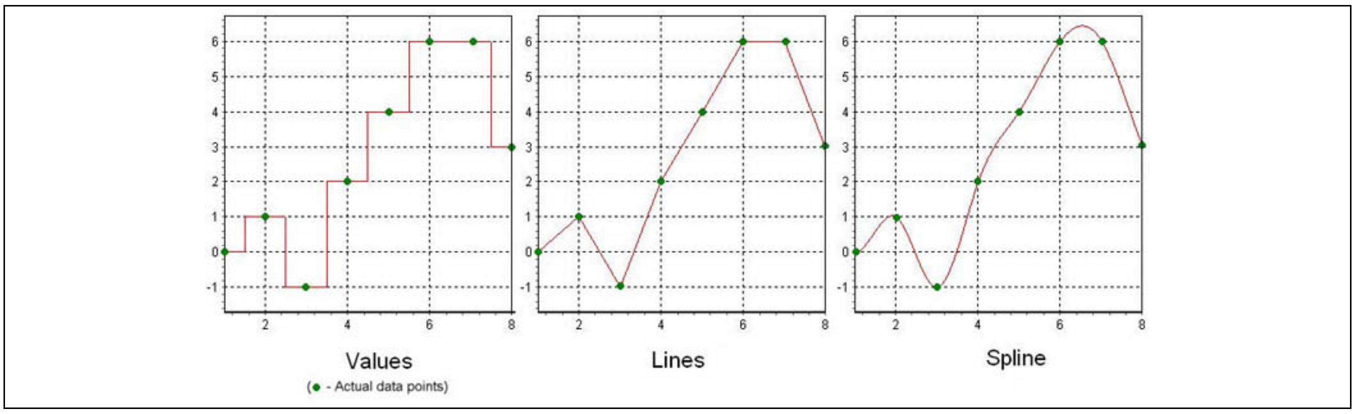
Smoothing width: 0

Unit string:

OK Cancel

Interpolation

Data are recorded for a curve as a series of discrete sample points. This setting specifies how the Oscilloscope will interpolate values between the actual sample points. The settings are illustrated below:



Typically, the choice of interpolation depends on the type of value being recorded. For example, any sort of "state" value will normally be displayed using *Values* since states are usually discrete in nature (i.e. they are one value or another and never have "in between" values). Values that are continuous in nature, such as position, velocity, etc., will normally be displayed using *Lines*.

Spline interpolation should only be used in very exceptional cases where a curve contains only a very limited number of sample points and is specifically intended to represent a spline (e.g. CAM Editor curves). In general, *Spline* should not be used when doing normal time-based sampling of data values.

NB!:
This setting may override the *Display* setting for some columns in the "Curve Selection List". If *Values* is selected, then the values in the "Curve Selection List" will be displayed according to the *Display* setting. However, if *Lines* or *Spline* is selected, then the values for the mouse position column and all the cursor columns will be interpolated values and will be displayed as *Decimal* regardless of the *Display* setting.

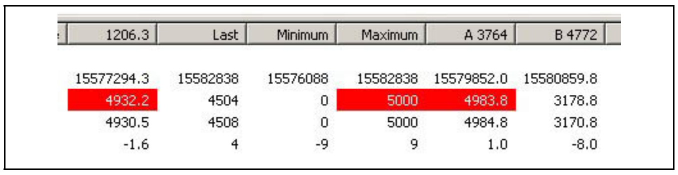
Enable scaling

For convenience, values recorded from the controller can be scaled in the Oscilloscope. For example, the user might want a value that is recorded in millimeters to be displayed in centimeters. This affects both the curve in the Chart Window and the displayed values in the "Curve Selection List". Two choices of scaling algorithms are available:
"(y-offset)*scale" and "(y*scale)-offset". Usually, one or the other will be more natural in any particular application. In the millimeter-to-centimeter example, *Offset* would be set to 0 and *Scale* would be set to 0.1.

Enable range validation

Often, it is useful to know if a curve value has gone "out of bounds". This can be done by enabling range validation and setting the low and high bounds of the valid range of values. Any value that is not within the valid range will then be displayed in the "Curve Selection List" with a red background.

Note that if only one bound is appropriate, then the other bound can be left blank. An example is shown below where the maximum value for the second curve has been set to 4900.



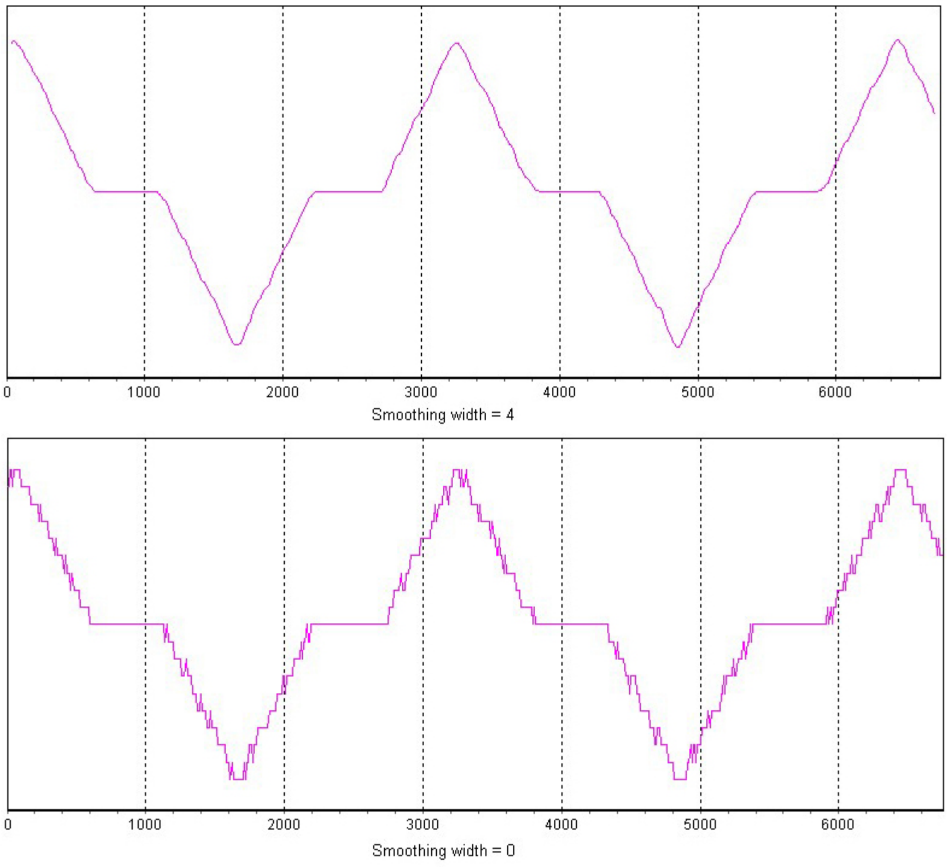
1206.3	Last	Minimum	Maximum	A 3764	B 4772
15577294.3	15582838	15576088	15582838	15579852.0	15580859.8
🟥 4932.2	4504	0	🟥 5000	🟥 4983.8	3178.8
4930.5	4508	0	5000	4984.8	3170.8
-1.6	4	-9	9	1.0	-8.0

Smoothing width

In some cases, measured values can contain a certain amount of "jitter" (i.e. oscillations that are fast, somewhat random, and usually small). This can be distracting in the chart. An example is shown below. Setting *Smoothing width* will reduce the appearance of this jitter by averaging the values of adjacent sample points.

Note that this is visual in the Oscilloscope only; it does NOT affect any of the actual values in the controller.

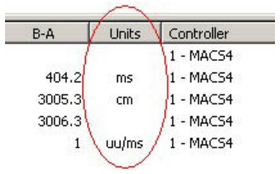
The width will specify the number of points on either side of each point that will be used in the averaging. For example, a width of 2 means 2 points on the left and 2 points on the right of each point for a total of 5 points in each averaged point.



NB!:
Warning: Care must be taken to avoid setting the smoothing width any higher than necessary. For example, if a width of 2 gives satisfactory results, then do not use a width of 4. As the width is set higher and higher, "real" information in the curve will begin to be "averaged away" and lost.

<u>Unit string</u>

An optional "Units" string can be set for each curve. Users can either enter their own string or use the drop-down box to select one of the predefined strings. The string will appear in the "Units" column in the "Curve Selection List".

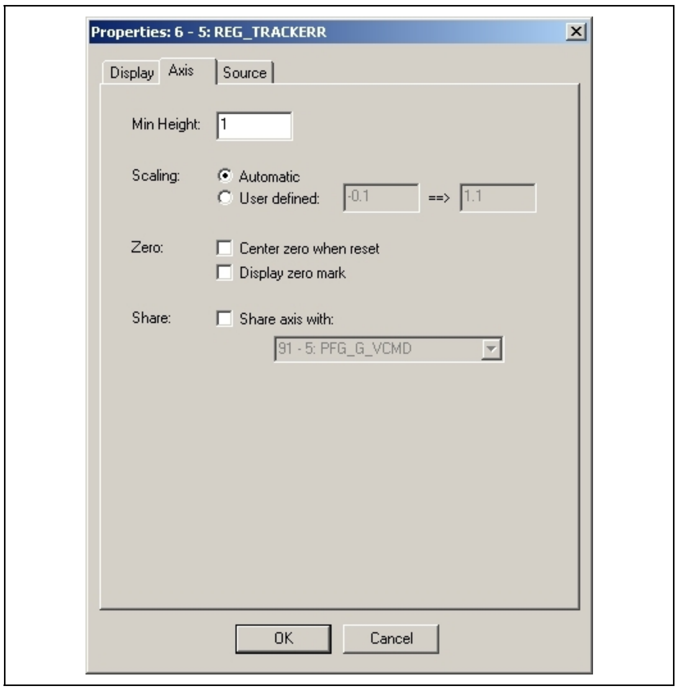


B-A	Units	Controller
		1 - MACS4
404.2	ms	1 - MACS4
3005.3	cm	1 - MACS4
3006.3		1 - MACS4
1	uu/ms	1 - MACS4

Curve Properties: Axis

Min Height

The Y-axis of the chart is normally scaled so that the entire curve can be drawn on the chart. However, this scaling cannot be determined in a way that will produce easily readable grid labels when the curve has a constant value or changes by only a very small amount. In these cases, the *Min Height* value is used as the minimum height of the Y-axis (i.e. the magnitude of the range of the Y-axis).



Properties: 6 - 5: REG_TRACKERR

Display | Axis | Source

Min Height: 1

Scaling: [✓ Automatic] User defined: -0.1 ==> 1.1

Zero: ☐ Center zero when reset ☐ Display zero mark

Share: ☐ Share axis with: 91 - 5: PFG_G_VCMD

OK Cancel

Scaling

If scaling is set to *Automatic*, then the Y-axis of the chart will be automatically scaled so that the entire curve can be drawn on the chart with maximum detail. The Y-axis range will be set to approximately 10% higher than the maximum value of the curve and 10% lower than the minimum value of the curve (and adjusted for the *Minimum Height* described above).

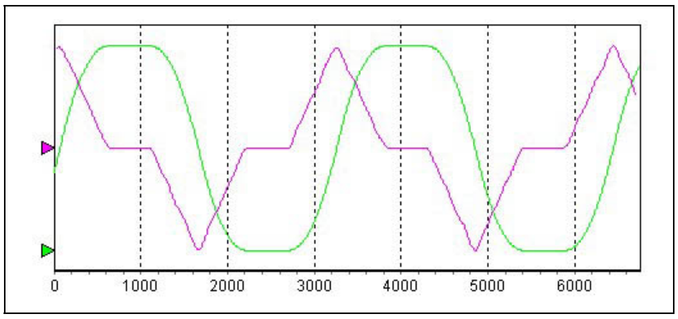
If scaling is set to *User defined*, then the user must specify the range of the Y-axis in the supplied boxes. Note that this will be automatically adjusted if the *Center zero when reset* option is enabled (see below).

The Y-axis scale defined here (i.e. either Automatic or User defined) will be the scale used for the curve when *Reset zoom* toolbar button is pressed. Note that if the scaling is changed, then the new scaling will not take effect until the next time that the user presses the *Reset zoom* toolbar button.

Zero

If *Center zero when reset* is enabled, then the Y-axis range for the curve will be adjusted so that it is symmetric about 0. This will result in the "0" grid line being drawn through the center of the chart.

If *Display zero mark* is enabled, then a small triangular mark will be drawn on the left side of the chart at the position of the 0 grid line for that curve. It will be drawn in the same color as the curve so that marks can be distinguished if marks are being drawn for multiple curves. The following figure illustrates these zero marks.



Share

In some cases, it is desirable to have two or more curves "share" the same Y-axis. For example, an "actual position" curve and a "commanded position" curve will be measured using the same units and should be displayed using the same scale in the Chart Window.

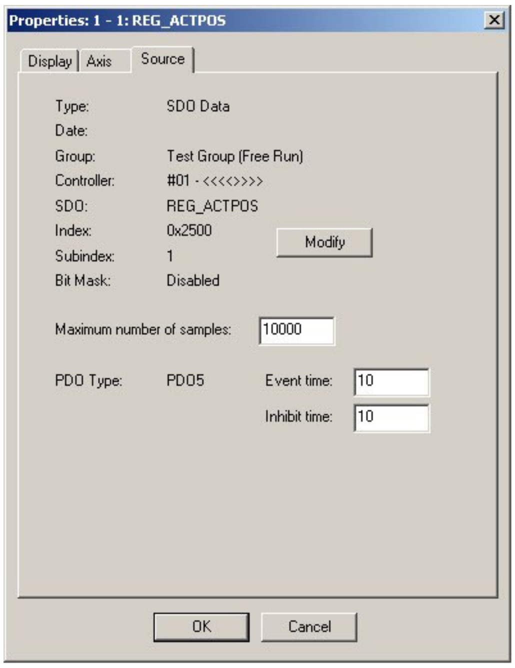
This can be done by enabling *Share* and choosing the "other" curve from the list of curves provided. This curve will then share the same Y-axis as the chosen "other" curve.

Sharing a Y-axis can also be done using the radio buttons in the Grid column of the "Curve Selection List".

NB!:
As well as explicit sharing (e.g. curve B shares the Y-axis of curve A and curve C also shares the Y-axis of curve A), implicit chains can also be created (e.g. curve C shares the Y-axis of curve B and curve B shares the Y-axis of curve A … implying that curve C also shares the Y-axis of curve A).
Changing how a curve shares the Y-axis can make or break these implicit chains.

Curve Properties: Source

The Source tab will display various information about the type of the curve and where the data for the curve came from. The exact content of this dialog will vary depending on the type of the curve. In some cases, attributes related to the collection of the data can also be modified.


Properties: 1 - 1: REG_ACTPOS

Type:	SDO Data
Date:	
Group:	Test Group (Free Run)
Controller:	#01 - <<<<>>>>
SDO:	REG_ACTPOS
Index:	0x2500
Subindex:	1
Bit Mask:	Disabled

Maximum number of samples: 10000

PDO Type: PDO5

Event time: 10

Inhibit time: 10

☐ Starting the Oscilloscope

Starting the Oscilloscope involves following steps:

1. Opening an APOSS program in an APOSS Window.
2. Connecting to the controller.
3. If the TESTSETP Oscilloscope is being used, then before starting the Oscilloscope, execute the APOSS program which creates the TESTSETP array.
4. Starting the selected Oscilloscope.

Opening an APOSS program

Begin by opening the APOSS program (.m) corresponding to the program currently loaded or executing in the controller. This will open a normal APOSS Window. Note that it is not strictly necessary to open the corresponding program. Any program, or even a new program, can be opened. For example, the user may not have the actual program readily available. However, opening the correct program can make the diagnostics effort easier since the user can refer to the program while trying to understand the results of the test.

Connecting to the controller

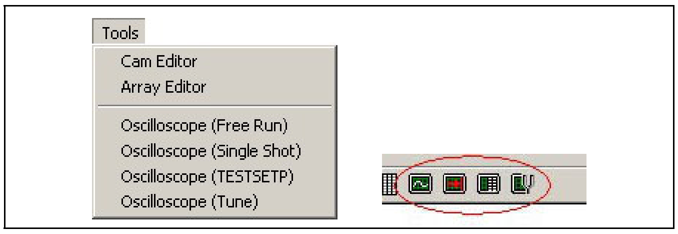
In the APOSS Window, connect to the appropriate controller. Use any of the available methods (e.g. the [Esc] key to connect to the default controller, the *Development* → *Select Controller* menu command, the *Connect* toolbar button, etc.).

Before using the TESTSETP Oscilloscope

The TESTSETP Oscilloscope expects a TESTSETP array to exist prior to the Oscilloscope being started. The Oscilloscope will automatically read this array when it is started. Hence, the user must first execute an APOSS program that records the desired data using the various TESTSET commands. This program can be started in the normal way by using *Development* → *Execute* [F5].

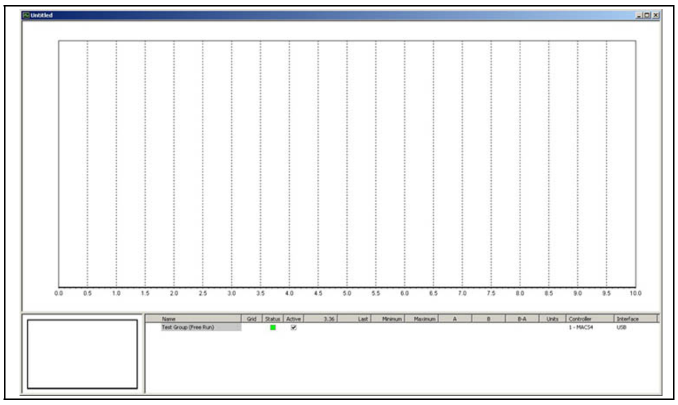
Starting the Oscilloscope

Each of the different Oscilloscope versions can be started from the *Tools* menu or by clicking on one of the Oscilloscope toolbar buttons. The Oscilloscope will also be started if a previously saved "oscilloscope file" is re-opened. Oscilloscope files are described below.



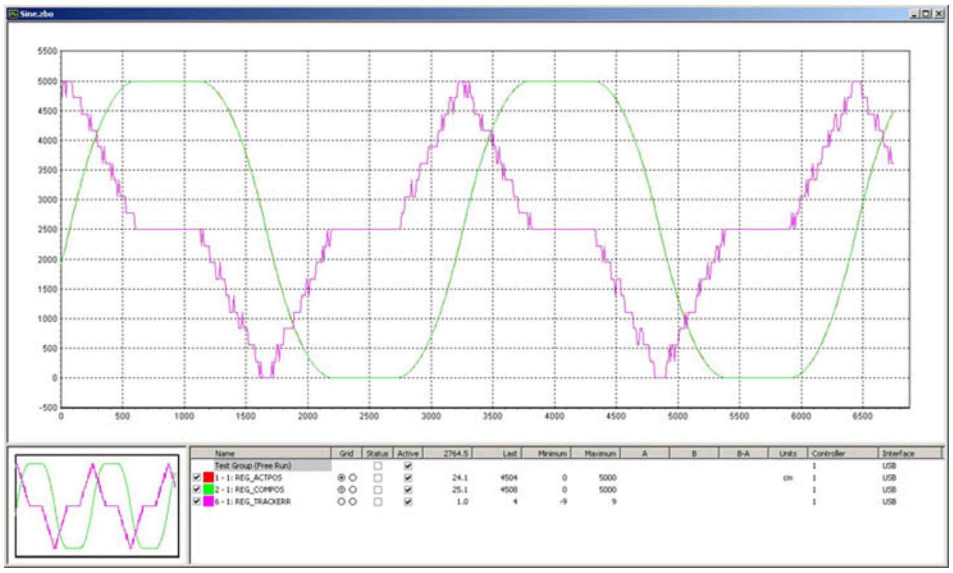
The Oscilloscope will automatically be connected to the controller that was connected in the APOSS Window. Note that if a controller was not already connected in the APOSS Window, then starting the Oscilloscope will cause the default controller to be connected.

An Oscilloscope window similar to the following will be opened. The connected controller can be seen at the far right side of the "Curve Selection List". The green box in the "Status" column in this example indicates that the controller is connected, is executing a program, and is not in an error state.



Name	Grid	Status	Active	3.36	Last	Minimum	Maximum	A	B	B-A	Units	Controller	Interface
Test Group (Free Run)		🟩	✓									1 - MAC54	USB

An Oscilloscope can also be started by opening an Oscilloscope file that was previously saved from an Oscilloscope. This is done simply by opening the Oscilloscope ".zbo" file using any of the normal methods for opening files. A Free Run Oscilloscope window similar to the following will be opened.



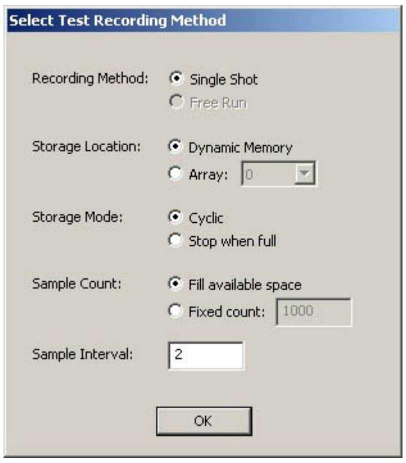
	Name	Grid	Status	Active	2764.5	Last	Minimum	Maximum	A	B	B-A	Units	Controller	Interface
	Test Group (Free Run)		☐	✓									1	USB
✓ 🟥	1 - 1: REG_ACTPOS	● ○	☐	✓	24.1	4504	0	5000				cm	1	USB
✓ 🟩	2 - 1: REG_COMPOS	● ○	☐	✓	25.1	4508	0	5000					1	USB
✓ 🟪	6 - 1: REG_TRACKERR	○ ○	☐	✓	1.0	4	-9	9					1	USB

The white boxes in the "Status" column indicate that controllers are not connected. However, the controller ID and connection interface can be seen at the far right side of the "Curve Selection List".

When Oscilloscope files are opened, controllers are never connected automatically. This allows Oscilloscope files to be opened and analyzed offline. Controllers can be reconnected either using the toolbar buttons or using the right mouse button in the "Curve Selection List" (see Status Column in "Curve Selection List").

Usage Notes for the Single Shot Oscilloscope

When the *Single Shot Oscilloscope* starts, it will automatically add a single group in the same way that the Free Run Oscilloscope does. However, the user is required to specify some characteristics of the TESTSETP array that will be used to record the curve data. The following window will be displayed:



Select Test Recording Method

Recording Method: [✓ Single Shot] Free Run

Storage Location: [✓ Dynamic Memory] Array: 0

Storage Mode: [✓ Cyclic] Stop when full

Sample Count: [✓ Fill available space] Fixed count: 1000

Sample Interval: 2

OK

<u>Recording Method</u>

For the Single Shot Oscilloscope, the recording method must be *Single Shot*.

<u>Storage Location</u>

This specifies the location where the TESTSETP array will be stored. If set the *Dynamic Memory*, then the TESTSETP array will be placed in available free memory in the controller. Note that the amount of free memory will depend on the size and number of programs and other data already stored in the controller. If set to *Array*, then the TESTSETP array will be placed in the specified user-defined array in the controller. This array must already have been defined by the user's application program.

<u>Storage Mode</u>

This determines what happens when the TESTSETP array becomes full. If it is set to *Cyclic*, then new data will start to overwrite old data and the old data will be lost. If set to *Stop when full*, then recording stops once the TESTSETP array is full and new data will be lost.

Sample Count

This specifies the maximum number of points that will be saved in the TESTSETP array. If set to *Fill available space*, then as many points as possible will be saved in the array. If set to *Fixed count*, then not more than that many sets of points (i.e. one point per curve) will be saved.

Sample Interval

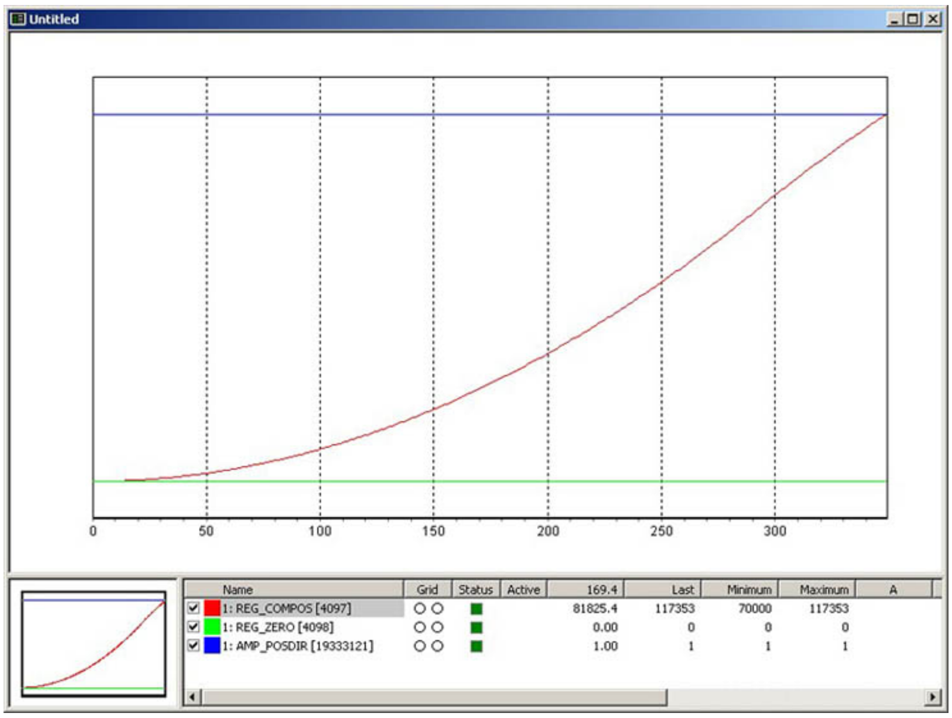
This specifies the time interval (in milliseconds) between samples.

The group added will have the name "Test Group (Single Shot)".

Usage Notes for the TESTSETP Oscilloscope

When the TESTSETP Oscilloscope starts, it will automatically try to locate TESTSETP arrays on the controller. If only one TESTSETP array is found, then that array will be automatically uploaded and displayed.
If more than one TESTSETP array is found, then the user is requested to identify which array is to be uploaded. The other TESTSETP arrays can be uploaded once the Oscilloscope starts.

The TESTSETP Oscilloscope window will appear similar to that shown below.



	Name	Grid	Status	Active	169.4	Last	Minimum	Maximum	A
✓ 🟥	1: REG_COMPOS [4097]	○ ○	🟩		81825.4	117353	70000	117353	
✓ 🟩	1: REG_ZERO [4098]	○ ○	🟩		0.00	0	0	0	
✓ 🟦	1: AMP_POSDIR [19333121]	○ ○	🟩		1.00	1	1	1	

This example was created with the APOSS command:

```
TESTSETP 1 4097 4098 0x01270001 test
```

Please note the following with regard to the "Curve Selection List":

- The TESTSETP array defined three values so three curves are automatically added to the Oscilloscope. The Oscilloscope will always add one curve for each curve defined in the array.
- The name of first curve in the set of TESTSETP curves will always be drawn with a grey background. This will let the user identify the sets of TESTSETP curves if multiple TESTSETP arrays have been added to the Oscilloscope.
- The Oscilloscope will try to derive curve names based on the indices specified in the TESTSETP command. These names will have the format "Axis: Name [Index]" where "Index" is the index from the command.
- The Oscilloscope can upload the TESTSETP array when the controller is in almost any state. The controller need not be executing a test program. So the "Status" column will not necessarily be bright green.

- Since the Oscilloscope does not control which values are recorded or the starting and stopping of the recording, the Active column will always be empty for TESTSETP curves.
- Also note that the *Start* toolbar button will be disabled in the TESTSETP Oscilloscope since starting and stopping of the recording is not done by the Oscilloscope.

Save and Open Oscilloscope Files

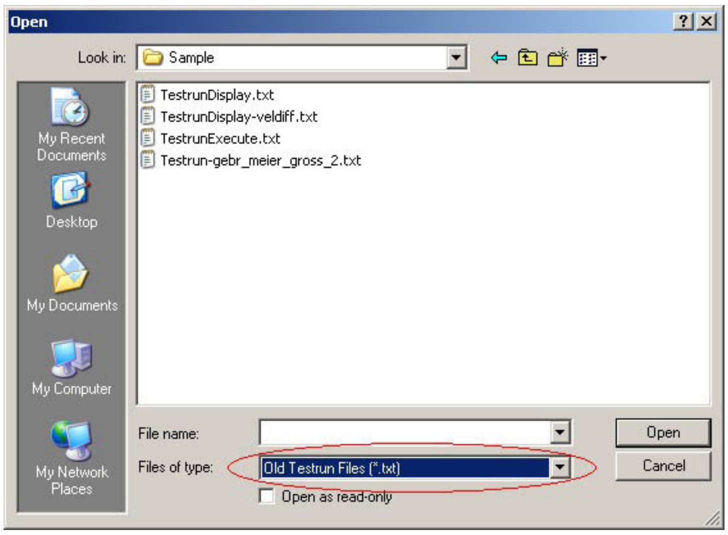
At any time, the complete set of curves, their data and properties, and all the oscilloscope settings, can be saved as an "oscilloscope file" on the PC. This is done in the standard way by using the *File* → *Save* and → *Save As* commands or by using the corresponding toolbar buttons. Oscilloscope files will always have a ".zbo" file extension. They will be identified in Windows folders with the icon .

Any of the standard methods of opening files (i.e. using the *File* → *Open* menu command or toolbar button, double-clicking the file, dragging the file onto the APOSS application icon or onto an open APOSS window, etc.) can be used to re-open an oscilloscope file. When an oscilloscope file is re-opened, then the Oscilloscope will be started and the curves re-displayed.

The Oscilloscope also has a special → *Save without Data* command in the File menu. This command can be used to save the complete oscilloscope configuration (i.e. all curve definitions and settings, etc.) but without including any of the actual recorded curve data. The intent of this is to provide the user with a method of creating "template" oscilloscope setups that can be opened and re-used at a later time.

Reading Old Testrun Files

The TESTRUN functionality that existed in previous APOSS versions, allowed the results of *Execute testrun* and *Display recording* commands to be saved in ".txt" files. These old ".txt" files can still be read and displayed by newer versions of APOSS. To do this, simply open the ".txt" file using the standard file open mechanism. To make this easier, the Files of type field in the Open window can be set to "Old Testrun Files" as is shown below.

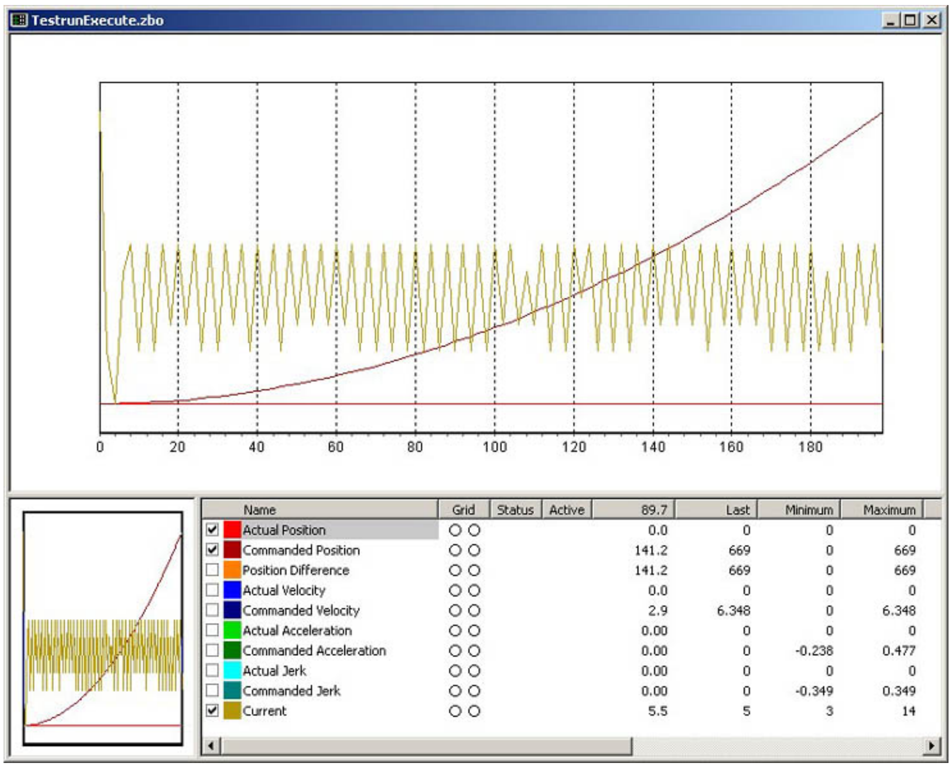


TestrunDisplay.txt
TestrunDisplay-veldiff.txt
TestrunExecute.txt
Testrun-gebr_meier_gross_2.txt

File name:
Files of type: Old Testrun Files (*.txt)

When the file is opened, then a TESTSETP Oscilloscope window will be opened to display the curves. An example is shown below. The file can then be resaved as a normal Oscilloscope ".zbo" file if desired.

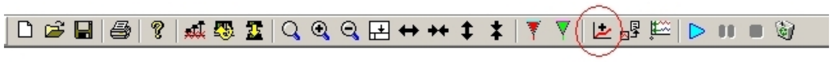
NB!:
If APOSS does not recognize the file as containing old Testrun data, then the file will be opened in a normal APOSS Window rather than in an Oscilloscope Window.



Name	Grid	Status	Active	89.7	Last	Minimum	Maximum
✓ 🟥 Actual Position	○ ○			0.0	0	0	0
✓ 🟥 Commanded Position	○ ○			141.2	669	0	669
☐ 🟧 Position Difference	○ ○			141.2	669	0	669
☐ 🟦 Actual Velocity	○ ○			0.0	0	0	0
☐ 🟦 Commanded Velocity	○ ○			2.9	6.348	0	6.348
☐ 🟩 Actual Acceleration	○ ○			0.00	0	0	0
☐ 🟩 Commanded Acceleration	○ ○			0.00	0	-0.238	0.477
☐ 🟦 Actual Jerk	○ ○			0.00	0	0	0
☐ 🟦 Commanded Jerk	○ ○			0.00	0	-0.349	0.349
✓ 🟨 Current	○ ○			5.5	5	3	14

Old ".txt" files do not contain enough information about the original controller to reliably reconnect to the controller. Hence, the Status, Controller, and Interface columns are left empty. However, information about the original test parameters can be viewed using the curve *Properties* → *Source* window.

□ Adding Oscilloscope Curves



Curves are added to the Oscilloscope using the *Add curve* toolbar button. The types of curves that can be added will depend of the type of Oscilloscope being used.

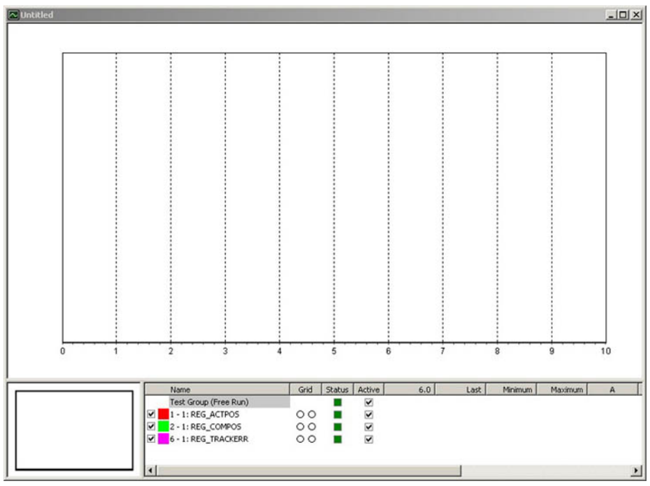
The Oscilloscope uses "Groups" to manage sets of curves. When a test is started, the Oscilloscope will configure all the curves in a group into a PDO mapping on the controller. Then each time the controller generates a PDO message, the entire set of curve values will be transmitted from the controller to the Oscilloscope and displayed.

When the Oscilloscope is started, a single Group is added automatically. The user is then free to add any curves of interest to this group. The following types of curves may be of particular interest when diagnosing problems:

- Axis process data (SDO 0x2500). For example, axis position, velocity, tracking error, etc.
- System process data (SDO 0x2202). For example, states of the digital I/O pins.
- User parameters (SDO 0x2201) … if used by the application.
- Program variables. If the correct APOSS program is known, then any variable used in the program can be recorded.

A single group is normally adequate for most purposes. However, if the User Mode is set to "Expert" in *Settings* → *Options*) menu, then the Oscilloscope will also support multiple groups. Multiple groups may be necessary, for example, if multiple controllers are being tested simultaneously. Since all curves in a group must be associated with the same controller, if two controllers are being used, then two groups must be defined, one for each controller.

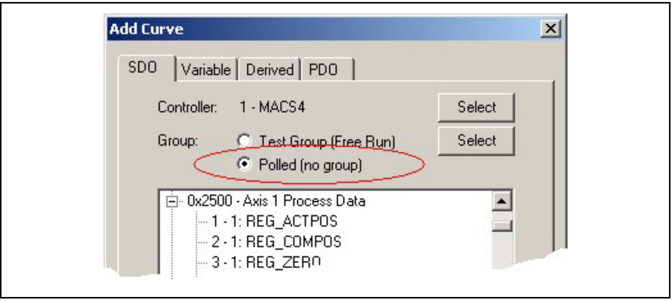
Once the curves are added, then the Oscilloscope Window will look something like that shown. In this example, three curves have been added but no data has yet been recorded so the chart is empty. Also, the dark green "Status" column indicates that the controller is connected but that no program is currently executing.



	Name	Grid	Status	Active	6.0	Last	Minimum	Maximum	A
	Test Group (Free Run)		🟩	✓					
✓ 🟥	1 - 1: REG_ACTPOS	○ ○	🟩	✓					
✓ 🟩	2 - 1: REG_COMPOS	○ ○	🟩	✓					
✓ 🟪	6 - 1: REG_TRACKERR	○ ○	🟩	✓					

Usage Notes for Polling Mode

If the Free Run Oscilloscope is being used in Polling Mode, then add the curves as usual but instead of selecting a group for the curve, the *Polled (no group)* option should be selected.



Add Curve

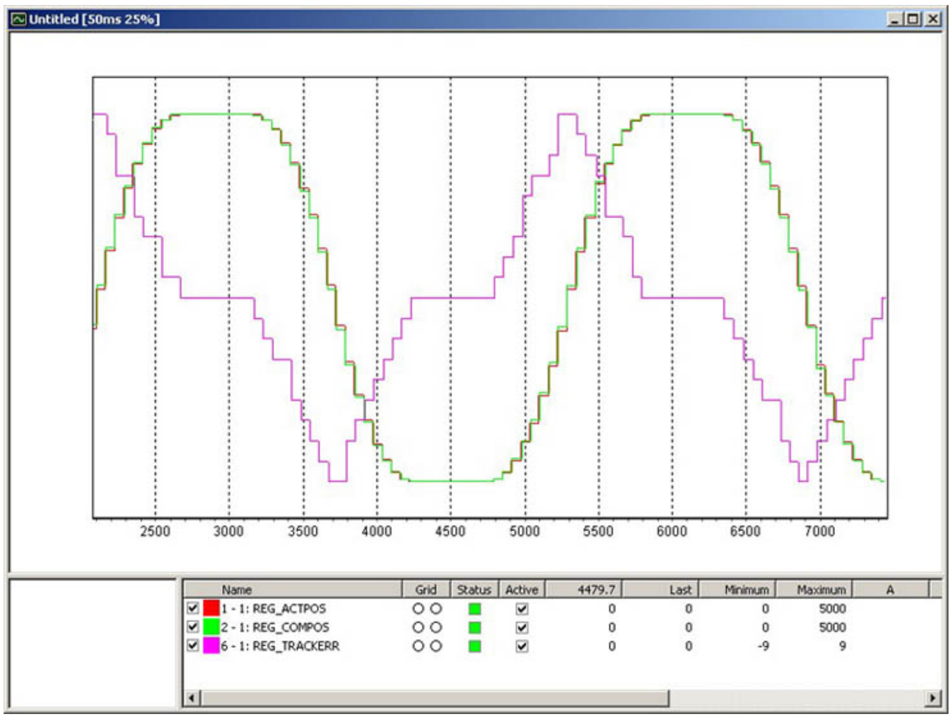
SDO | Variable | Derived | PDO

Controller: 1 - MACS4 — Select

Group: Test Group (Free Run) [✓ Polled (no group)] — Select

0x2500 - Axis 1 Process Data
- 1 - 1: REG_ACTPOS
- 2 - 1: REG_COMPOS
- 3 - 1: REG_ZERO

Recording is then started in the normal way using the *Start* ▷ toolbar button. The Oscilloscope will retrieve curve data by “polling” each curve on a regular time interval. Each curve is polled by sending an explicit SDO request for its value. The user can set the polling time interval (i.e. the polling rate) using the *Settings* → *Oscilloscope* menu command. The Oscilloscope Window will look similar to the following:



Name	Grid	Status	Active	4479.7	Last	Minimum	Maximum	A
✓ 🟥 1 - 1: REG_ACTPOS	○ ○	🟩	✓	0	0	0	5000	
✓ 🟩 2 - 1: REG_COMPOS	○ ○	🟩	✓	0	0	0	5000	
✓ 🟪 6 - 1: REG_TRACKERR	○ ○	🟩	✓	0	0	-9	9	

When the Oscilloscope is actively recording data, the window title (top left in the above example) will contain a string similar to “[50ms 25%]”. The “50ms” is the polling rate. In this example, each curve will be polled once every 50 milliseconds. The “25%” gives an estimate of how much of the communications network is being consumed. This percentage will go up as more and more curves are being polled. A percentage higher than 100% indicates that the Oscilloscope can no longer poll all curves in each time interval. If this happens, the user should either reduce the number of curves being polled or increase the poll time interval.

The number of curves that can be polled and the rate at which they can be polled, cause a significant limitation to the Oscilloscope when running in polling mode. Many fewer curves can be polled, and much less often, than is the case if PDO messages are used. The result is that the recorded data is much less detailed.

SDO Curve

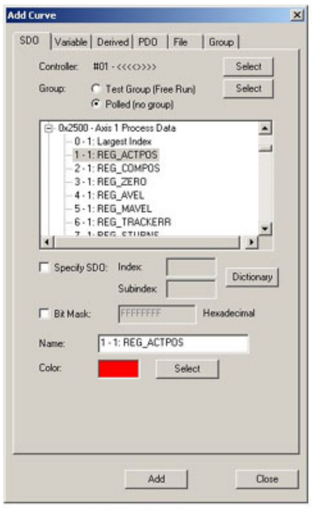
A curve can be added for any SDO that can be read from the controller.

There are three methods of recording the data for the curve: as a "polled" value, through a "Free Run" group curve, and through a "Single Shot" group curve. "Free Run" groups can only be used in the Free Run Oscilloscope and "Single Shot" groups can only be used in the Single Shot Oscilloscope. Recording the data through a group curve is the preferred method since this greatly reduces the utilization of both the controller processor and the communications network.

Polled value - When an oscilloscope run is started, the Oscilloscope will regularly "poll" the SDO by sending an "SDO read request" to the controller. Polled SDO curves will be added to the "Curve Selection List" under the "Other Curves" heading.

Free Run group - When an oscilloscope run is started, all SDO's in the group will be configured into a single PDO message. The controller will then automatically assemble and send the PDO message to the Oscilloscope on a regular cycle. These SDO curves will be added to the "Curve Selection List" under the associated group heading.

Single Shot group - When an oscilloscope run is started, all SDO's in the group will be configured into a TESTSETP array. When the run ends, the Oscilloscope will upload the TESTSETP array and extract the SDO data. These SDO curves will be added to the "Curve Selection List" under the associated group heading.



Add Curve

Controller: #01 - <<<<>>>>

Group: ( ) Test Group (Free Run) [✓ Polled (no group)]

0x2500 - Axis 1 Process Data
- 0 - 1: Largest Index
- 1 - 1: REG_ACTPOS
- 2 - 1: REG_COMPOS
- 3 - 1: REG_ZERO
- 4 - 1: REG_AVEL
- 5 - 1: REG_MAVEL
- 6 - 1: REG_TRACKERR

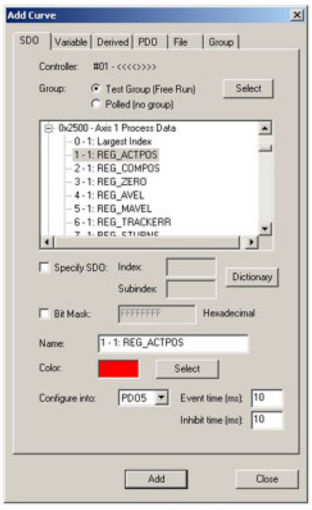
Specify SDO: ☐ Index: Subindex:

Bit Mask: ☐ FFFFFFFF Hexadecimal

Name: 1 - 1: REG_ACTPOS

Color: 🟥

Polled value



Add Curve

Controller: #01 - <<<<>>>>

Group: [✓ Test Group (Free Run)] ( ) Polled (no group)

0x2500 - Axis 1 Process Data
- 0 - 1: Largest Index
- 1 - 1: REG_ACTPOS
- 2 - 1: REG_COMPOS
- 3 - 1: REG_ZERO
- 4 - 1: REG_AVEL
- 5 - 1: REG_MAVEL
- 6 - 1: REG_TRACKERR

Specify SDO: ☐ Index: Subindex:

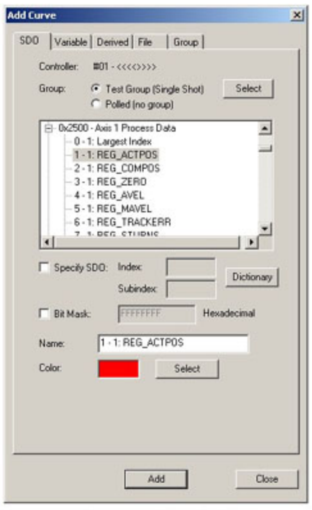
Bit Mask: ☐ FFFFFFFF Hexadecimal

Name: 1 - 1: REG_ACTPOS

Color: 🟥

Configure into: PDO5 Event time (ms): 10 Inhibit time (ms): 10

Free Run group



Add Curve

Controller: #01 - <<<<>>>>

Group: [✓ Test Group (Single Shot)] ( ) Polled (no group)

0x2500 - Axis 1 Process Data
- 0 - 1: Largest Index
- 1 - 1: REG_ACTPOS
- 2 - 1: REG_COMPOS
- 3 - 1: REG_ZERO
- 4 - 1: REG_AVEL
- 5 - 1: REG_MAVEL
- 6 - 1: REG_TRACKERR

Specify SDO: ☐ Index: Subindex:

Bit Mask: ☐ FFFFFFFF Hexadecimal

Name: 1 - 1: REG_ACTPOS

Color: 🟥

Single Shot group

<u>Controller</u>

This is the controller that will be associated with the curve. If a polled curve is being added, then the user will be able to select a different controller for the curve. Each polled curve can be associated with a different controller. If the curve is being added to a group, then the curve must be associated with the controller that the group is associated with.

<u>Group</u>

The SDO can either be polled or added to an existing group. If more than one group exists, then the user will be able to select the group to which the SDO is to be added.

<u>SDO list</u>

Use this list to → *Select* the SDO of interest.

<u>Specify SDO</u>

If the SDO cannot be found in the SDO list, then clicking this checkbox will allow the user to specify the exact SDO index and subindex. The SDO index can be specified in either decimal (e.g. 9472) or hexadecimal (e.g. 0x2500) notation.

<u>Bit Mask</u>

If the SDO value has an internal coding (e.g. multiple values packed into a single SDO value), then this mask can be used to extract the appropriate bits. For example, a value of 000000FF will extract the low-order 8 bits from the 32-bit SDO value.

<u>Name</u>

This is the name that will appear in the “Curve Selection List”. A default name will be generated but the user is free to change this to any other desired name. The default name will be formed as “Subindex - Axis: Name”.

<u>Color</u>

This is the color that will be used to draw the curve. The Select button can be used to choose a different color.

<u>Configure into</u>

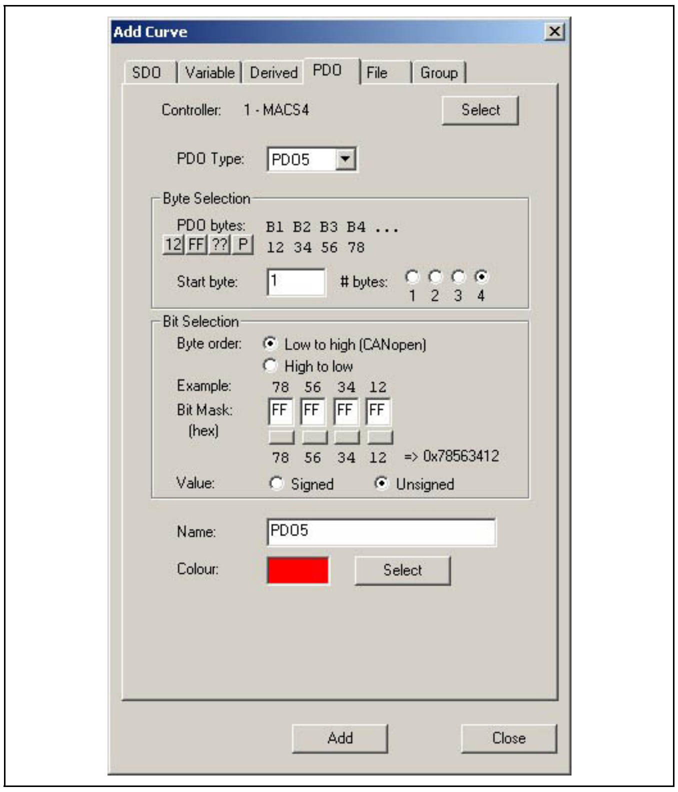
If the SDO is being added to a Free Run group, then the user is able to select the PDO into which the SDO will be configured. The PDO event and inhibit times can also be set. (See SDO 1800 in SDO Object Dictionary CANopen Basic Data in the manual „MCO 305 Command Reference”.)

NB!:
The *Specify SDO, Bit Mask*, and *Inhibit time* fields are only available if the APOSS “User Mode” is set to “Expert”. The *User Mode* is set using the *Settings* → *Options*) menu command.

PDO Curve

If the application running on the controller is sending PDO messages, then a PDO curve can be used to extract data from the PDO message. The data can be extracted from any byte position within the PDO.

Note that PDO curves are not polled and cannot be added to groups. Data is recorded asynchronously by the Oscilloscope whenever a PDO message is received. PDO curves will be added to the "Curve Selection List" under the "Other Curves" heading.



Add Curve

SDO | Variable | Derived | PDO | File | Group

Controller: 1 - MACS4 [Select]

PDO Type: PDO5

Byte Selection

PDO bytes: B1 B2 B3 B4 ...
12 | FF | ?? | P 12 34 56 78

Start byte: 1 # bytes: 1 2 3 [✓ 4]

Bit Selection

Byte order: [✓ Low to high (CANopen)] High to low

Example: 78 56 34 12

Bit Mask: (hex) FF FF FF FF

78 56 34 12 => 0x78563412

Value: Signed [✓ Unsigned]

Name: PDO5

Colour: 🟥 [Select]

[Add] [Close]

Controller

This is the controller that will be associated with the curve. The user will be able to select a different controller for the PDO. Each PDO curve can be associated with a different controller.

PDO Type

This is the type of the PDO.

Note that not all connection interface types support all PDO types and some interface types do not support any PDO messages at all.

Byte Selection

This section is used to select which bytes in the PDO message will contain the data of interest. Bytes are numbered starting at 1, which is the first data byte in the PDO message. The number of bytes required can be from 1 to 4. The "B1 B2..." string with the "12 34..." string underneath it, will give the user an indication of which bytes have currently been selected.

The four small buttons can be used to generate test PDO data that the user can then use to help verify the correct byte and bit specification.

12 - The byte string 0x12 0x34 0x56 0x78 will be used.
FF - The byte string 0xFF 0xFF 0xFF 0xFF will be used.
?? - A string of random bytes will be generated.
P - The Oscilloscope will listen for an actual incoming PDO. If found, the bytes will be extracted from the PDO.

Bit Selection

This section is used to specify how the value is to be extracted from the selected bytes. *Byte order* can be either *Low to high* (which is the standard CANopen byte order and is used for commands such as PDO and LINKSDO) or *High to low* (which should be used for low-level commands such as CANIN and CANOUT).

The *Example* bytes are exactly the test PDO bytes from the Byte Selection section above. The example bytes in this section are always displayed in the standard high-to-low byte order for a single multi-byte value.

Changing *Byte order* will also change the order of the example so that the user can tell exactly how the incoming PDO bytes will be handled.

The *Bit Mask* fields contain hexadecimal values that will be used as a mask to extract the curve data from the selected bytes. The small button under each field can be used to toggle the mask between 0xFF and 0x00. Under these buttons, the same example bytes are displayed again, this time incorporating the bit mask. And finally, the user can specify whether the value is to be treated as a signed or unsigned value.

<u>Name</u>

This is the name that will appear in the “Curve Selection List”. A default name will be generated but the user is free to change this to any other desired name.

<u>Color</u>

This is the color that will be used to draw the curve. The Select button can be used to choose a different color.

Variable Curve

If the application currently executing on the controller is known and has been compiled by APOSS, then curves can be added to the Oscilloscope for any variable used within the program. As with SDO curves, there are three methods of recording the data for the curve:

Polled value – When an oscilloscope run is started, the Oscilloscope will regularly "poll" the SDO by sending an "SDO read request" to the controller. Polled SDO curves will be added to the "Curve Selection List" under the "Other Curves" heading.

Free Run group – When an oscilloscope run is started, all SDO's in the group will be configured into a single PDO message. The controller will then automatically assemble and send the PDO message to the Oscilloscope on a regular cycle. These SDO curves will be added to the "Curve Selection List" under the associated group heading.

Single Shot group – When an oscilloscope run is started, all SDO's in the group will be configured into a TESTSETP array. When the run ends, the Oscilloscope will upload the TESTSETP array and extract the SDO data. These SDO curves will be added to the "Curve Selection List" under the associated group heading.

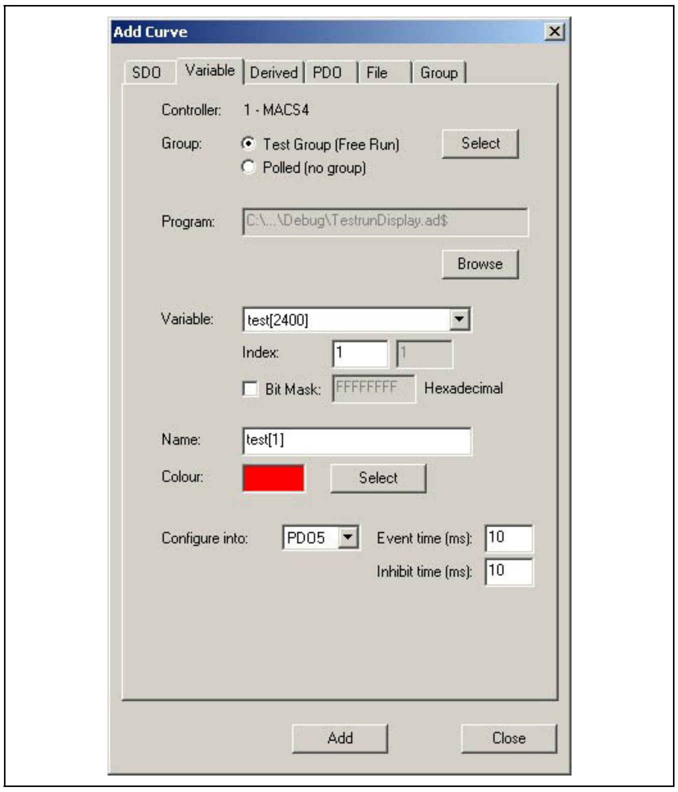
NB!:
The Oscilloscope cannot verify that the program running on the controller is identical to the program selected by the user when adding a *Variable curve*. It is the user's responsibility to ensure that this is true. A mismatch will not cause the Oscilloscope to fail but it also will not generate the correct results for the curve.

Controller

This is the controller that will be associated with the curve. If a polled curve is being added, then the user will be able to select a different controller for the curve. Each polled curve can be associated with a different controller. If the curve is being added to a group, then the curve must be associated with the controller that the group is associated with.

Group

The variable can either be polled or added to an existing group. If more than one group exists, then the user will be able to select the group to which the variable is to be added.


Add Curve

SDO | Variable | Derived | PDO | File | Group

Controller: 1 - MACS4

Group: (●) Test Group (Free Run) ( ) Polled (no group)

Program: C:\...\Debug\TestrunDisplay.ad$

Variable: test[2400]

Index: 1 1

☐ Bit Mask: FFFFFFFF Hexadecimal

Name: test[1]

Colour: 🟥

Configure into: PDO5

Event time (ms): 10

Inhibit time (ms): 10

Program

This is the program containing the variable of interest. It is the responsibility of the user to ensure that this is the program that is executing on the controller.

NB!:
Note that the file extension of the program file will be ".ad$". This is the cross reference file generated by the APOSS compiler. (See *Settings* → *Compiler*)

The Browse button can be used to select a different program file.

Variable

This will be a list of variables found in the program. Use this list to select the variable of interest. If an array (either one or two dimensional) is selected, then → *Index* must be used to specify exactly which array element is to be recorded by the curve.

If the variable value has an internal coding (e.g. multiple values packed into a single value), then the → *Bit Mask* can be used to extract the appropriate bits. For example, a value of 000000FF will extract the low-order 8 bits from a 32-bit variable value.

NB!:
Note that *Bit Mask* cannot be used with "double" variables.

Name

This is the name that will appear in the "Curve Selection List". A default name will be generated but the user is free to change this to any other desired name.

Color

This is the color that will be used to draw the curve. The Select button can be used to choose a different color.

Configure into

If the variable is being added to a Free Run group, then the user is able to select the PDO into which the variable will be configured. The PDO event and inhibit times can also be set. (See SDO 1800 in SDO Object Dictionary CANopen Basic Data in the manual "MCO 305 Command Reference".)

The *Inhibit time* field is only available if the APOSS "User Mode" is set to "Expert". The User Mode is set using the *Settings* → *Option* menu command.

Derived Curve

If other curves have already been added, then curves can be added that are derived from those curves. The derived curve will always be added to the "Curve Selection List" after the curve from which it is derived.

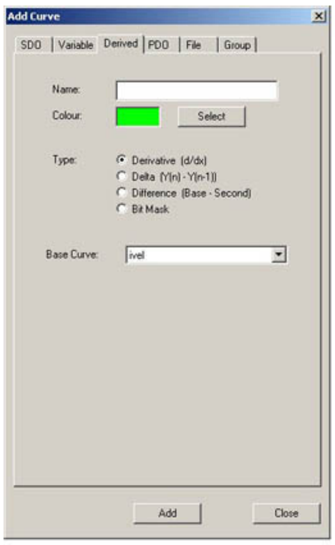
The following forms of derived curves can be added:

Derivative – This is the mathematical derivative (i.e. d/dx) of the original curve.

Delta – A "delta" curve will show the incremental differences along the original curve. Each point on the curve will be the difference between the corresponding point on the original curve and the previous point on the original curve.

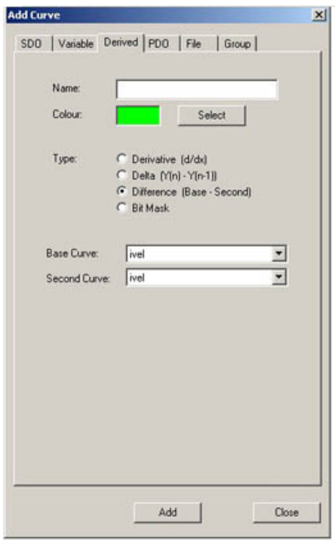
Difference – A "difference" curve will be the simple difference (i.e. subtraction) between the values of two original curves.

Bit Mask – If a curve value has an internal coding (e.g. multiple values packed into a single value), then this curve will derive its data from bits extracted from the values of the original curve. For example, a mask of 000000FF will extract the low-order 8 bits from a 32-bit value.



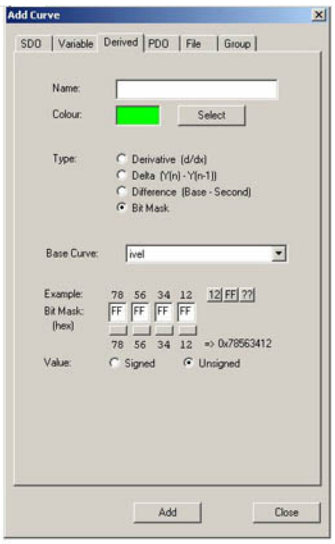
Name:
Colour: 🟩
Type: [✓ Derivative (d/dx)] Delta (Y(n) - Y(n-1)) Difference (Base - Second) Bit Mask
Base Curve: ivel

Derivative / Delta



Name:
Colour: 🟩
Type: Derivative (d/dx) Delta (Y(n) - Y(n-1)) [✓ Difference (Base - Second)] Bit Mask
Base Curve: ivel
Second Curve: ivel

Difference



Name:
Colour: 🟩
Type: Derivative (d/dx) Delta (Y(n) - Y(n-1)) Difference (Base - Second) [✓ Bit Mask]
Base Curve: ivel
Example: 78 56 34 12 12|FF|??
Bit Mask: (hex) FF FF FF FF
78 56 34 12 => 0x78563412
Value: Signed [✓ Unsigned]

Bit Mask

Name

This is the name that will appear in the "Curve Selection List". The user should set this to some appropriate name.

Color

This is the color that will be used to draw the curve. Use → *Select* to choose a different color.

Type

This is the type of derived curve to be added.

Base Curve(s)

Base Curve is the curve from which the new curve will be derived.

Second Curve - If the new curve is derived from two curves (e.g. a "difference" curve), then this will be the second curve.

Bit Mask

This is used to specify how a Bit Mask value is to be extracted from the original value (which is always a 32-bit value). The Example bytes can be used by the user to help verify the correct bit mask specification. The three small buttons can be used to generate different test values for the example.

12	The value 0x78563412 will be used.
FF	The value 0xFFFFFFFF will be used.
??	A random value will be generated.

The *Bit Mask* fields contain hexadecimal values that will be used as a mask to extract the curve data from the selected bytes. The small button under each field can be used to toggle the mask between 0xFF and 0x00. Under these buttons, the same example bytes are displayed again, this time incorporating the bit mask. And finally, the user can specify whether the value is to be treated as a signed or unsigned value.

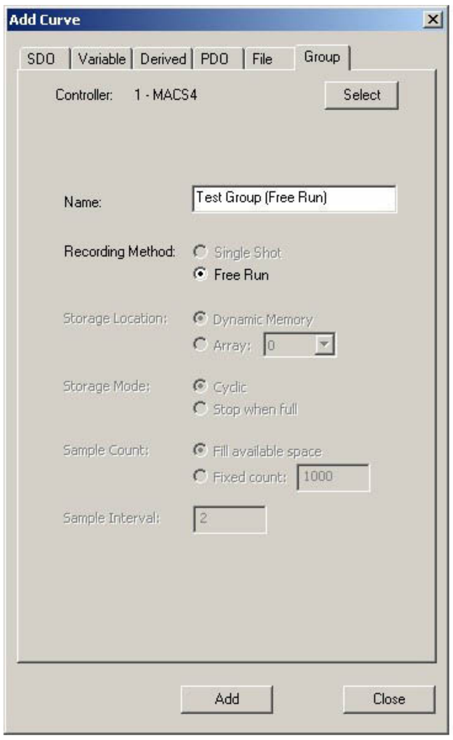
Group Curve

A Group curve is not an actual curve and it contains no recorded information. Instead, it simply contains a collection of other curves and allows that entire set of curves to be handled together as a unit. For example, all the curves in a group will be configured by the Free Run Oscilloscope into a single PDO message for transferring values from the controller to the Oscilloscope. Or all the curves in a group will be configured by the Single Shot Oscilloscope into a single TESTSETP array for data collection.

NB!:
Note that one consequence of this is that all curves in a group must be associated with the same controller.

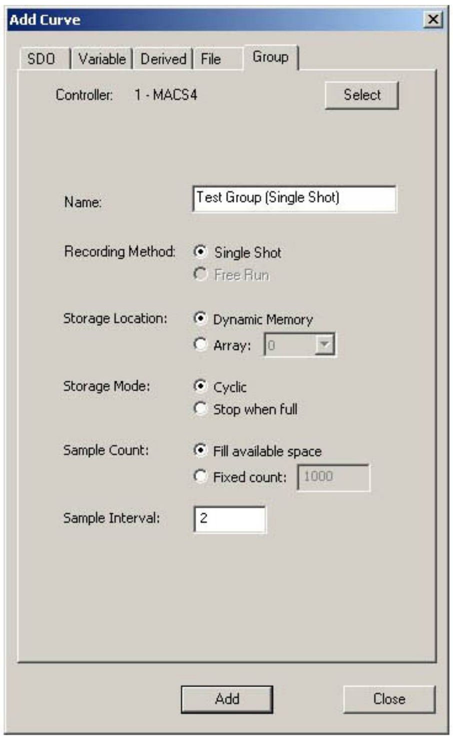
The preferred method of using the Oscilloscope is to add a Group curve and then add other curves to that group. A single Group curve is adequate for most uses of the Oscilloscope. However, in some cases (e.g. testing with multiple controllers), adding multiple Group curves can be useful. This can be done if "User Mode" is set to "Expert" (see *Settings* → *Options* menu command).

Group curves will be displayed in the "Curve Selection List" with a grey background. Groups will always be added to the list after all previously existing groups and before the "Other Curves" heading if it exists.



Controller: 1 - MACS4
Name: Test Group (Free Run)
Recording Method: Single Shot [✓ Free Run]
Storage Location: [✓ Dynamic Memory] Array: 0
Storage Mode: [✓ Cyclic] Stop when full
Sample Count: [✓ Fill available space] Fixed count: 1000
Sample Interval: 2

Free Run Group



Controller: 1 - MACS4
Name: Test Group (Single Shot)
Recording Method: [✓ Single Shot] Free Run
Storage Location: [✓ Dynamic Memory] Array: 0
Storage Mode: [✓ Cyclic] Stop when full
Sample Count: [✓ Fill available space] Fixed count: 1000
Sample Interval: 2

Single Shot Group

Name

This is the name that will appear in the "Curve Selection List". A default name will be given but the user should set this to some appropriate name.

Recording Method

If the Free Run Oscilloscope is being used, then this will be set to *Free Run*. In this case, sample data is transferred from the controller to the Oscilloscope in real time using PDO messages. If the Single Shot Oscilloscope is being used, then this will be set to *Single Shot*. In this case, sample data is recorded in a TESTSETP array and transferred to the Oscilloscope at the end of the test.

All the following attributes apply only to *Single Shot Groups*. They affect various aspects of the TESTSETP array that is configured to record the curve data during the test.

Storage Location

This specifies where the TESTSETP array will be placed. If set to *Dynamic Memory*, then the TESTSETP array will be saved in the remaining unused memory in the controller. Note that the amount of unused memory will vary depending on such things as the size of the loaded program, the number and sizes of any pre-allocated user arrays, the number of saved programs, etc. If *Array* is selected, then the TESTSETP array will be placed in the specified array. This array must be pre-allocated in the controller (e.g. by defining it in an application program or by using the Array Editor, etc.).

Storage Mode

This specifies what happens when the TESTSETP array becomes full.

If set to *Cyclic*, then the oldest existing data point will be lost in order to make room for the newest data point (i.e. the TESTSETP array will contain only the *most recent* set of points recorded).

If set to *Stop when full*, then new data points are discarded (i.e. the TESTSETP array will contain only the first and oldest set of points recorded).

Sample Count

This specifies the number of sets of sample points that can be saved (i.e. it determines the size of the TESTSETP array).

Note that it refers to the number of sets of points (i.e. one sample point for each curve in the group) and not to the total number of individual sample points.

If set to *Fill available space*, then the TESTSETP array will be made as large as possible while still fitting within either the dynamic memory or the predefined user array. Alternatively, *Fixed count* can be used if the user wants a specific number of points only.

Sample Interval

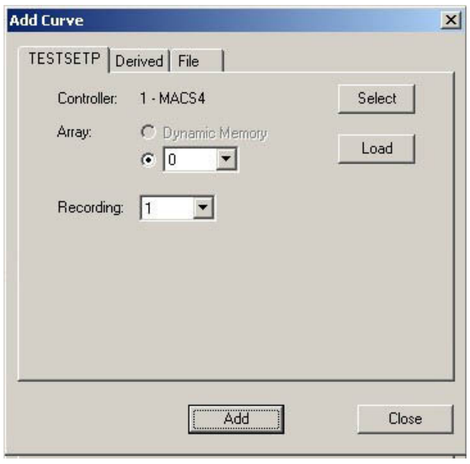
This specifies the time interval, in milliseconds, between each saved set of sample points.

TESTSETP Curve

If a user configures a custom TESTSETP array in an APOSS program (i.e. rather than using the Single Shot Oscilloscope to configure the array automatically then a TESTSETP curve must be added to the Oscilloscope to store the data. This will be necessary, for example, with older controllers that do not support the automatic configuration of TESTSETP arrays. When the TESTSETP curve is added, the Oscilloscope will immediately try to read the TESTSETP array from the controller. Hence, the test must already have been run and completed and the TESTSETP array created on the controller *before* the TESTSETP curve is added to the Oscilloscope.

When a TESTSETP "curve" is added, the Oscilloscope will actually add a separate curve for each curve defined in the TESTSETP array. The first of these curves will always be drawn with a grey background in the "Curve Selection List".

TESTSETP curves can only be added when the TESTSETP Oscilloscope is being used.



Controller: 1 - MACS4
Array: ( ) Dynamic Memory [✓ 0]
Recording: 1

Controller

This is the controller containing the TESTSETP array. The → *Select* button can be used to choose a different controller.

Array

The array can be loaded from either dynamic memory or from a predefined user array. Once the array location has been specified, then the *Load* button must be pressed. This will cause the Oscilloscope to upload the TESTSETP array from the controller.

NB!:
Note that *Dynamic Memory* will be disabled if the controller does not support dynamic memory or if *Dynamic memory* does not contain a TESTSETP array.

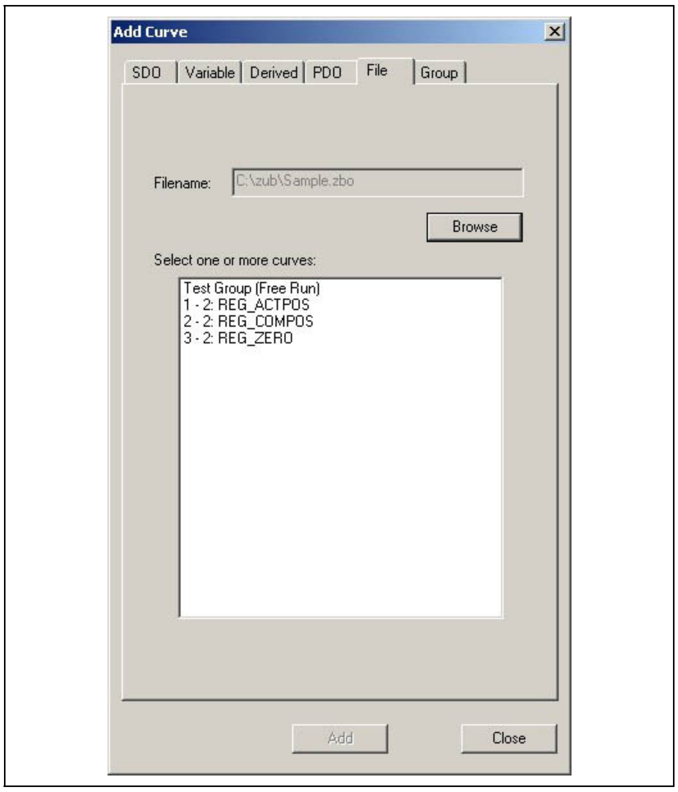
Recording

If the TESTSETP array contains multiple recordings, then one of the recordings must be chosen. More than one recording can be added to the Oscilloscope but a separate TESTSETP "curve" must be added for each one.

Adding Curves from other Oscilloscope Files

Curves defined in other Oscilloscope ".zbo" files can be copied into the current Oscilloscope by selecting the File tab.

Note: This feature is only available if the "User Mode" is set to "Expert" (see *Settings → Options*).



Filename: C:\zub\Sample.zbo

Select one or more curves:

```
Test Group (Free Run)
1 - 2: REG_ACTPOS
2 - 2: REG_COMPOS
3 - 2: REG_ZERO
```

<u>Filename</u>

This is the Oscilloscope file containing the curves that will be copied. The *Browse* button can be used to select the Oscilloscope file.

<u>Curve list</u>

This is a list of the curves found in the selected Oscilloscope file.

Select the curve to be copied by clicking on it with the left mouse button. Multiple curves can be selected by holding the [Ctrl] key down while clicking on the curve. Some curves rely on the presence of other curves and cannot be copied as stand-alone curves. For example, a derivative curve cannot be copied without also copying the curve from which it is derived. Similarly, all curves in a group must be copied together. If the user selects one of these curves, then the other required curves will be automatically selected as well.

NB!:
Note that the list will contain only those curves in the selected Oscilloscope file that are compatible with the current Oscilloscope. For example, if the selected Oscilloscope file contains Single Shot groups, then these groups will not be in the list if the current Oscilloscope is a Free Run Oscilloscope.

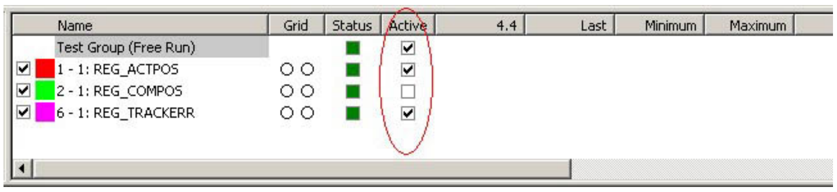
☐ Starting and Ending the Test

Once all the curves have been added, then the test itself can be started. This involves three steps:

1. Selecting which curves are to be recorded.
2. Starting the application program running on the controller.
3. Starting the Oscilloscope recording.

<u>Selecting which curves are to be recorded</u>

The user can define many curves in the Oscilloscope but it is not necessary to record data for all curves during each test. Use the checkbox in the "Active" column in the "Curve Selection List" to select which curves will be recorded. In the example below, data will be recorded only for the "REG_ACTPOS" (actual position) and "REG_TRACKERR" (tracking error) curves. Data for "REG_COMPOS" will not be recorded.



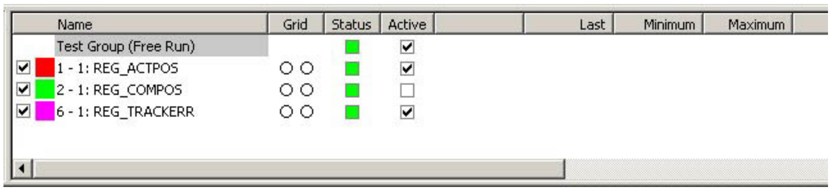
Name	Grid	Status	Active	4.4	Last	Minimum	Maximum
Test Group (Free Run)		🟩	✓				
✓ 🟥 1 - 1: REG_ACTPOS	○ ○	🟩	✓				
✓ 🟩 2 - 1: REG_COMPOS	○ ○	🟩	☐				
✓ 🟪 6 - 1: REG_TRACKERR	○ ○	🟩	✓				

Note that groups as a whole can be selected or deselected. If selected, then data for the selected curves within that group will be recorded. If deselected, then the checkboxes for all curves within that group will be grey and no data will be recorded for any of those curves.

When multiple groups have been defined, then there is a restriction as to which groups may be selected at the same time. No two groups may be selected at the same time if they are associated with the same controller <u>and</u> use the same PDO message. If groups have been defined in this way, then selecting one group will automatically cause the other group to be deselected.

<u>Starting the application program</u>

The next step is to start the application program running on the controller if it is not already running. Do this by switching to the original APOSS Window and starting the application in any of the normal ways. Then switch back to the Oscilloscope Window. The "Curve Selection List" should now look something like that below. Note that the bright green "Status" column indicates that the controller is now executing a program.

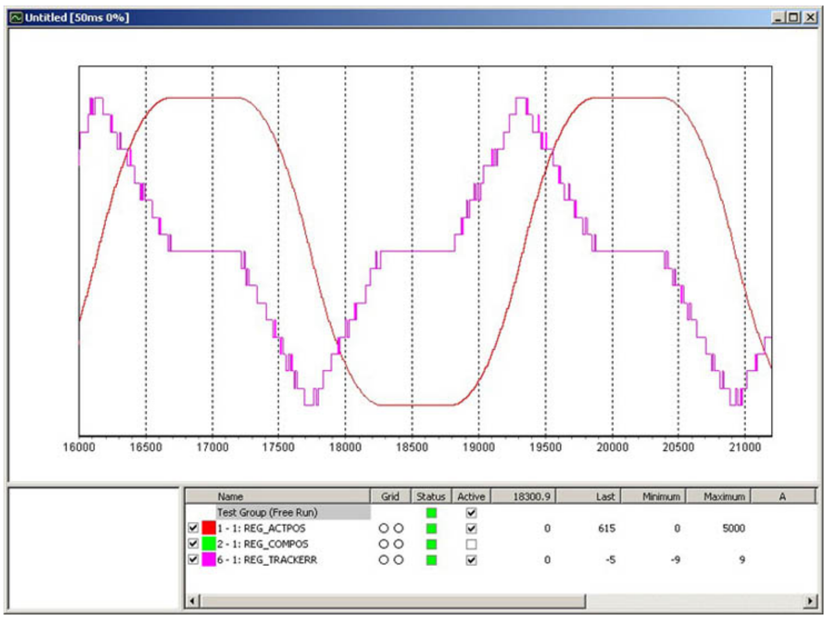


Name	Grid	Status	Active		Last	Minimum	Maximum
Test Group (Free Run)		🟩	✓				
✓ 🟥 1 - 1: REG_ACTPOS	○ ○	🟩	✓				
✓ 🟩 2 - 1: REG_COMPOS	○ ○	🟩	☐				
✓ 🟪 6 - 1: REG_TRACKERR	○ ○	🟩	✓				

<u>Starting the Oscilloscope recording</u>

The final step is to start the Oscilloscope recording. This is done used the ▷ *Start* toolbar button.

At this point, curve data should start to be recorded and displayed in the Chart Window. This will be done in real time as the test progresses (for the Free Run Oscilloscope) or after the end of the test (for the Single Shot Oscilloscope). The Oscilloscope Window should now look similar to that shown below.



	Name	Grid	Status	Active	18300.9	Last	Minimum	Maximum	A
	Test Group (Free Run)		🟩	✓					
✓ 🟥	1 - 1: REG_ACTPOS	○ ○	🟩	✓	0	615	0	5000	
✓ 🟩	2 - 1: REG_COMPOS	○ ○	🟩	☐					
✓ 🟪	6 - 1: REG_TRACKERR	○ ○	🟩	✓	0	-5	-9	9	

The order in which the test program execution is started and the Oscilloscope recording is started, is not fixed. The user is free to start the program execution and then the recording or start the recording and then the program execution, whichever is appropriate.

At any time during the recording, any of the **Zoom** toolbar buttons can be used, any of the curves can be hidden or shown in the Chart Window using the checkbox in the "Name" column, and the Chart Y-axis can be set using the Grid column. Note that the Navigation window is not drawn and cannot be used while recording is in progress.

Each curve has a fixed maximum number of points that can be recorded. Once this number has been reached, then the oldest points are discarded to make room for new points. The user can change this maximum number in the *Properties* → *Source* window.

The recording will continue until the user stops it by pressing the *Stop* toolbar button.

NB!:
Note that this does <u>not</u> abort the executing program; the executing program will continue to run. The *Stop* button only stops the recording of new data.

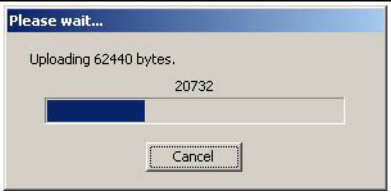
NB!:
Note that the user can abort the executing program at any time simply by switching to the APOSS Window and pressing the [Esc] key to break the program. Aborting the program does <u>not</u> stop the Oscilloscope from recording. The Oscilloscope will continue to listen for incoming PDO messages until the *Stop* toolbar button is pressed.

Usage Notes for the Single Shot Oscilloscope

If multiple groups are used, then only one group per controller can be selected for recording. If a second group is selected for the same controller, then the first group will be automatically deselected.

Start the recording as described above with the *Start* toolbar button. However, while data is being recorded, no curves will be displayed in the Chart Window.

Once the recording is stopped, then a window similar to the following will be displayed as the TESTSETP array is uploaded to the Oscilloscope. The curves will be displayed in the Chart Window once the upload completes.



Uploading 62440 bytes.

20732

□ Analyzing the Results

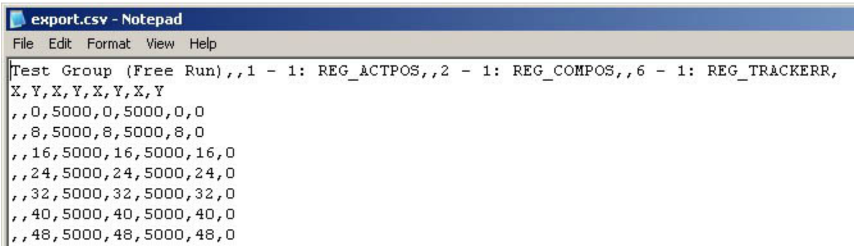
Any of the following functions and features of the Oscilloscope can be used to help analyze the recorded data:

1. The curves in the Chart Window may show correlations or anomalies. Use the *Zoom* toolbar buttons and the Navigation Window to display more detailed portions of the curves.
2. The Last, Minimum, and Maximum columns in the "Curve Selection List" may provide useful information.
3. Use the mouse and the Cursors to scan back and forth across the curves. Curve values will be displayed in the "Curve Selection List".
4. Use the *Properties* → *Display* window to enable *range validation* for curves. This may make it easier to locate "out-of-bounds" problems.
5. Create "*derived*" Curves. For example, a derived curve can be added that shows the difference between two curves or the rate at which a curve is changing.

Export Curve Data into a txt file

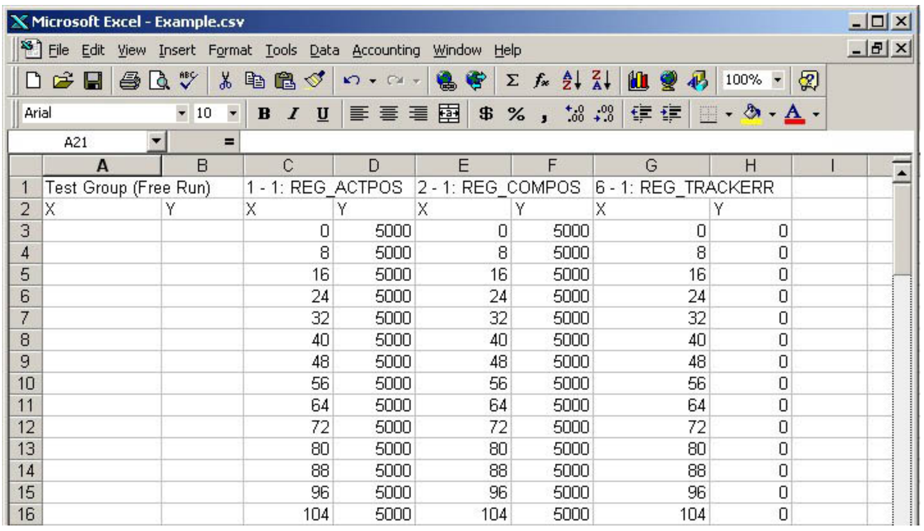
Use the *Export* toolbar button to export the curve data to a standard text file. This file can then be used by other programs to analyze the curve.

This will export all the curve data to a "comma-separated value" (.csv) file. This is a standard text file that can be imported into spreadsheet programs and opened with normal text editors. The file will appear similar to the following example:



```
Test Group (Free Run),,1 - 1: REG_ACTPOS,,2 - 1: REG_COMPOS,,6 - 1: REG_TRACKERR,
X,Y,X,Y,X,Y,X,Y
,,0,5000,0,5000,0,0
,,8,5000,8,5000,8,0
,,16,5000,16,5000,16,0
,,24,5000,24,5000,24,0
,,32,5000,32,5000,32,0
,,40,5000,40,5000,40,0
,,48,5000,48,5000,48,0
```

Once imported into Microsoft Excel, it will appear similar to this:


Microsoft Excel - Example.csv
File Edit View Insert Format Tools Data Accounting Window Help
100%
Arial
10
A21

	A	B	C	D	E	F	G	H	I
1	Test Group (Free Run)		1 - 1: REG_ACTPOS		2 - 1: REG_COMPOS		6 - 1: REG_TRACKERR		
2	X	Y	X	Y	X	Y	X	Y	
3			0	5000	0	5000	0	0	
4			8	5000	8	5000	8	0	
5			16	5000	16	5000	16	0	
6			24	5000	24	5000	24	0	
7			32	5000	32	5000	32	0	
8			40	5000	40	5000	40	0	
9			48	5000	48	5000	48	0	
10			56	5000	56	5000	56	0	
11			64	5000	64	5000	64	0	
12			72	5000	72	5000	72	0	
13			80	5000	80	5000	80	0	
14			88	5000	88	5000	88	0	
15			96	5000	96	5000	96	0	
16			104	5000	104	5000	104	0	

☐ Repeating the Test

Once Oscilloscope recording has been stopped, then the user is free to make any necessary changes, additions, and deletions to the curves. Recording can be restarted simply by pressing the ▷ *Start* button again.

NB!:
Note that this will delete any previously recorded data. Previously recorded data will not be deleted if the *Pause* toolbar button is used instead to the *Stop* button. In this case, a vertical red bar is drawn in the chart at the position where the recording was paused.

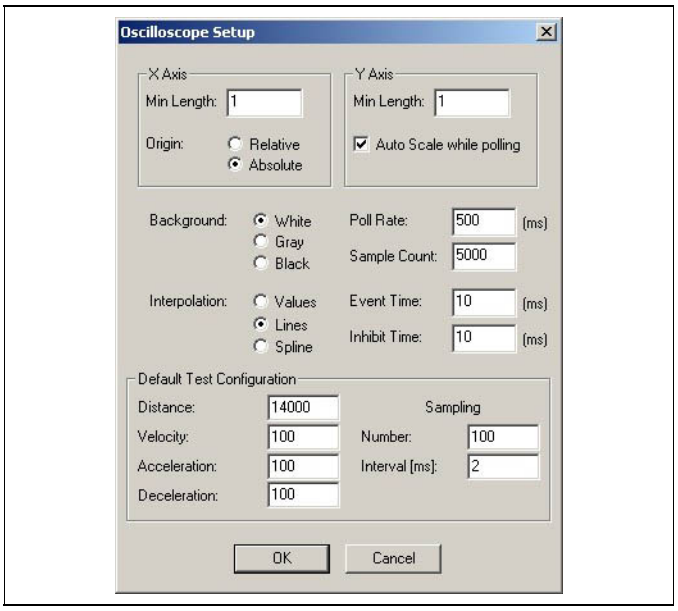
Usage Notes for the TESTSETP Oscilloscope

Repeating a TESTSETP test is done entirely by the user outside the TESTSETP Oscilloscope. Once completed, then the user has three options:

1. Open an entirely new TESTSETP Oscilloscope.
2. If the previous values are no longer needed, then the existing curves can be "reloaded". Do this by right-clicking on the curve name in the "Curve Selection List" and selecting the *Reload* or *Reload & Redisplay* command. The old values will be discarded and the new values will be loaded.
3. Add new TESTSETP curves using the *Add curve* toolbar button. This has the advantage that it allows the new curves to be compared with the old curves. This can be done either visually or by adding "derived" curves.

☐ Oscilloscope Settings

Various global Oscilloscope attributes and defaults can be set using the *Settings* → *Oscilloscope* menu command.



Oscilloscope Setup

X Axis
- Min Length: 1
- Origin: Relative [✓ Absolute]

Y Axis
- Min Length: 1
- ✓ Auto Scale while polling

Background: [✓ White] Gray Black

Poll Rate: 500 (ms)

Sample Count: 5000

Interpolation: Values [✓ Lines] Spline

Event Time: 10 (ms)

Inhibit Time: 10 (ms)

Default Test Configuration

Distance:	14000	Sampling	
Velocity:	100	Number:	100
Acceleration:	100	Interval [ms]:	2
Deceleration:	100		

OK Cancel

X axis

Min Length specifies the minimum length of the X-axis in the Chart Window. This value is used when the chart is empty. In normal situations, the X-axis length is based on the range required by the curves being displayed.

The *Origin* setting determines the initial state of the *Shift to 0* toggle button. If set to *Relative*, then by default the X-axis (time) will be shifted so that curves start at time 0. If set to *Absolute*, then the X-axis is not shifted. This setting will affect all curves.

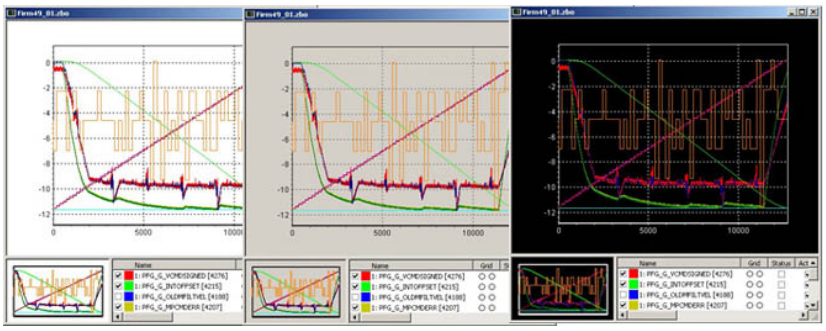
Y axis

Min Length specifies the default minimum length to be used for the Y-axis when new curves are being added. After a curve has been added, then this value can be changed by setting the curve *Properties* → *Axis* in the "Curve Selection List".

If "Auto Scale while polling" is enabled, then the Y-axis used in the Chart Window will be recalculated while an Oscilloscope "run" is active. This prevents curves from being drawn "outside" the Chart Window if Y values go outside the range of the current Y-axis. If this happens, then the Y-axis of the Chart Window will be scaled so that the curve always remains entirely visible in the Chart Window.

Background

This selects the background color of the Chart and Navigation Windows. The different backgrounds are shown below.



Interpolation

This is the default display style to be used when new curves are added. After a curve has been added, its display style can be changed using *Properties* → *Display* in the "Curve Selection List".

<u>Poll Rate</u>

The poll rate specifies how often the Oscilloscope will try to read data for polled curves from the controller when an Oscilloscope “run” start been started. Care should be taken not to set this time shorter than necessary as it can significantly increase demands on the PC processor, the controller processor, and the communication network.

<u>Sample Count</u>

This is the default sample count to be used when new curves are added. The sample count is the maximum number of points that the Oscilloscope will save for a curve before old points are discarded to make room for new points. See also *Properties* → *Source*.

<u>Event Time / Inhibit Time</u>

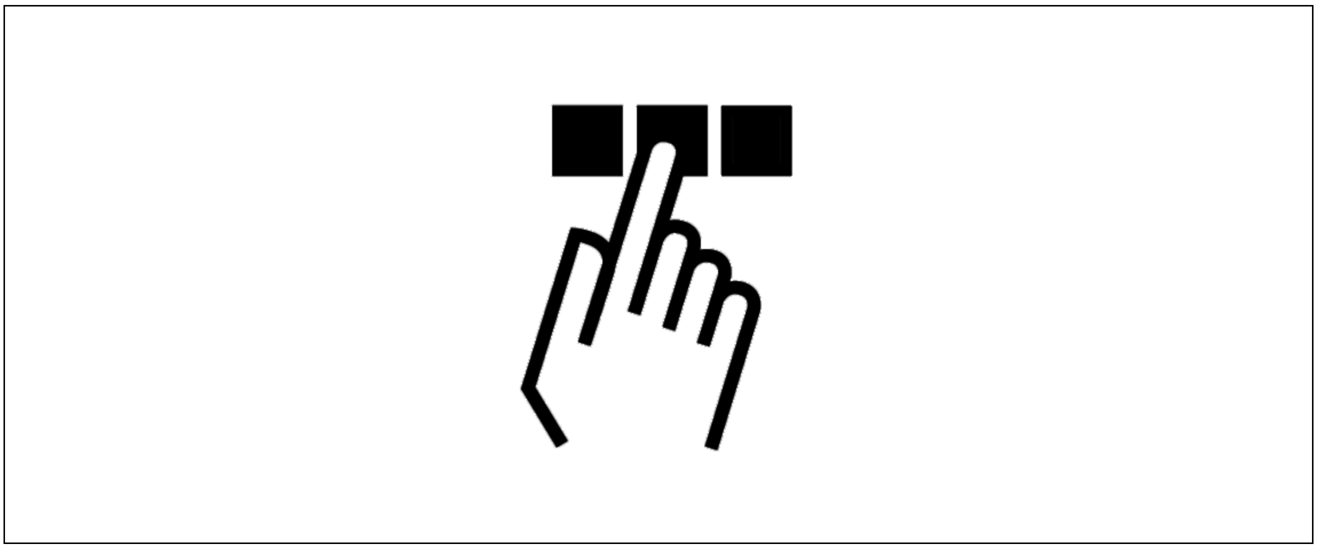
This is the default PDO event time and the default PDO inhibit time to be used when new curves are being added.

After a curve has been added, these values can be changed by using *Properties* → *Source* in the “Curve Selection List”.

<u>Default Test Configuration</u>

This set of parameters specifies the default values to be used by the Tune Oscilloscope.

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.

All commands are described in command groups including its syntax and parameters. Please see the manual MCO 305 Command Reference for a detailed description as well as short examples. You can reconstruct many programs with assistance of the information in the Program Samples in the online help.

See PC Software Interface for all programming tools from *File, Edit, Development, etc.* menu. 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 presented. After entering the values, the syntax is automatically formed and you can insert the entire command in your program. And in chapter Parameter Reference all the parameters are described, first in general and then in detail.

Basics

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
```

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

```
SET POSERR 100000000      // Parameters
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
```

Functions, Declarations, and Pragma directives

Function declarations and function definitions can be anywhere in the program code except the subprogram section. Pragma directives, if present, must be directly after the DIM statement.

That means a program can look like:

```
DIM statement
pragmas
...
variable declarations
...
function declarations
...
program code
...
function definitions
...
SUBMAINPROG
subprogram definitions
...
ENDPROG
```

Or it could also look like:

```
DIM statement
pragmas
...
SUBMAINPROG
subprogram definitions
...
ENDPROG
variable declarations
...
function declarations
...
program code
...
function definitions
...
```

Or it could look like:

```
DIM statement
pragmas
...
variable declarations
...
function definitions
...
program code
...
SUBMAINPROG
subprogram definitions
...
ENDPROG
```

Function definitions can also be anywhere in the program code as well as variable declarations or function declarations.

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 POSA pos

<u>Input Values</u>

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 downloaded to the controller but execution of the program is not started. Breakpoints are added to every executable line and are displayed as blue dots at the beginning of each line in the program. The "next" line to be executed, is highlighted in yellow. The user can then "single step" through the program. Please see the section Debugging Programs on page 86 for more details.

□ Interrupts

In general there are four types of interrupts:

ON INT	Interrupt at the edges of an input
ON posint	Position interrupt (ON APOS, …)
ON posint SETOUT (TOIN)	Simulates a cam box (all types of posints)
ON PERIOD / ON TIME	Interrupt after a certain period of time
ON COMBIT / ON STATBIT	Interrupt when Bit n is set
ON KEYPRESSED	Interrupt when a LCP key is pressed or released
ON PARAM	Interrupt when a parameter n is changed
ON CANMSG	Interrupt after receiving a CAN message

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 posint (ON APOS, ON MAPOS, ON MCPOS, ON IPOS, ON MIPOS) before
    ON COMBIT before
      ON STATBIT before
        ON PARAM before
          ON CANMSG before
            ON KEYPRESSED before
              ON TIME / ON PERIOD
```

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

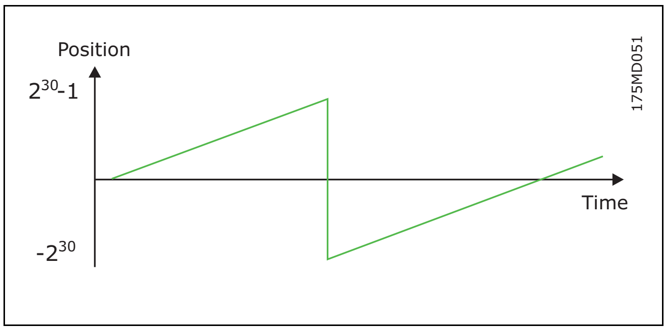
Word (16 bit): 2^{16} - 1 = 65535

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]
```

2-Dimensional Arrays

Starting with MCO 5.00 two dimensional arrays are supported

```
DIM test[10][100]
```

will create an array with a total of 1000 elements. (Note that in the Array Editor and in the zbc file, it will appear as a one dimensional array of size 1000.) This array can be addressed as follows:

```
lin = 1
col = 1
value = 1
WHILE(lin <= 10) do
    WHILE(col <= 100) do
     test[lin][col++] = value++
    ENDWHILE
    lin++
ENDWHILE
```

Array Copy

APOSS allows copying whole arrays in a single command.

```
DIM test[100], field[200]
...
test[] = field[]
```

In the case where the array sizes differ, the smaller of the two arrays limits the copying process. So in the above case, only the first 100 longs of field will be copied into test. This works for local arrays (LONG, DOUBLE) as well.

Variable Arrays

In addition to DIM arrays, which exist outside of APOSS programs and can be stored in flash and read by APOSS, it is now also possible to define arrays in the same way as variables. These arrays only exist inside the application program and are freed again after exiting the program. Those arrays are defined by using the normal variable declaration statements.

```
long aTest[100], bTest[10]
```

Those arrays are stored like variables. They can also be defined locally inside of functions, but then they reside on stack. So you must be aware of overflowing the stack (see Compiler Options).

NB!:
Double arrays are not supported yet.

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*.

Switch Statement

Switch statements are supported starting with MCO 5.00:

The following expressions are valid

```
#define WITH 1
#define WITHOUT 2
LONG val

val = WITH

switch (val)
    case  WITH:  var = var + 1
                 break
    case WITHOUT: var = var - 1
                  // fall through
    default :     var = var % 2
endswitch
```

Break Statement

Break statements are supported starting with MCO 5.00:

The new break statement is not only useful for switch statements (see above) but also to leave WHILE and REPEAT loops. If used in such a loop, a break leaves the loop. In nested loops only the inner loop is exited.

Pragmas

At the moment there are only two pragma directives:

    #pragma NOIMPLICIT
    #pragma IMPLICIT

NOIMPLICIT tells the compiler that no implicit variable declarations are allowed any more. That means that every variable must be declared before it is used. IMPLICIT tells the compiler that implicit variable declarations are allowed again. Declaration statements look as follows:

```
long a,b,c,d
double u,v,y,z
     // x is not allowed, because it is used for axes notification x(n)
```

It is recommended that NOIMPLICIT be used since this avoids the creation of unwanted definitions of new LONG variables by mistake (e.g. a misspelled variable name).

NB!:
Pragmas are supported starting with MCO 5.00.

Variable Types

Variable types are available starting with MCO 5.00.

APOSS supports variables of type LONG and DOUBLE. LONG variables can be defined implicitly by just using them (unless the NOIMPLICIT pragma is used). DOUBLE variables must be declared before they are used.

Declaration of those variables looks like:

```
long a,b,c,d
double u,v,y,z
     // x is not allowed, because it is used for axes notification x(n)
```

Implementation of DOUBLE variables is hardware and firmware dependent. MCO305 (firmware > 5.00) – uses 64 bit precision (some double functions like sqrt, sin, cos .. are only 32 bit precision).

Additionally, the following DOUBLE functions are supported:

sqrt, sin, cos, tan, acos, asin, atan, grad, rad, ln, exp, as well as the constant PI. These functions are only 32-bit precision for some hardware (MCO305). They can be used as follows:

```
y = sqrt(x)  // returns the square root of x
y = sin(x)   // returns the sine of x  (this also works for cos, tan, acos, asin, atan)
y = grad(x) // converts x (radians) into degrees (0..360)
y = rad(x)   // converts degrees into radians (0.. 2 *pi)
x = pi * 0.5 // Pi is 3.1415….
y = ln (x)    // returns the natural logarithm of x
y = x exp u // returns x raised to u  (x^u)
```

The double functions can be used: FABS returns the floating point absolute value of a floating point argument.

```
   double dx, dy
   dy = -100.00 % 15.0  // this will be -6.6666..
   dx = fabs(dx)  // this gives back 6.6666..
```

ROUND rounds the floating point argument to the closest integer. This function still returns a double.

```
   dx = round(dy)  // this will be -7.0000
```

The RND command for integers also works for negative numbers.

```
   a = -100
   b = a rnd 15  // now correctly delivers –7
```

Furthermore, a cast operator is supported. So you can explicit cast expressions like:

```
      a = ((long) y) – 4
      y = (double) a
```

If necessary, the compiler will internally cast expressions. If

```
      y = a * 2.0
```

where a is a long variable and y is a double variable, then the a is cast to double before the multiplication is executed. But you must be careful. If

y = a % 2

Then the integer a is first divided by 2 (integer arithmetic) and then the result is converted to double before it is assigned to y.

Functions

APOSS supports functions and local variables. Functions can have the type LONG or DOUBLE. A function can have maximum 100 parameters. Functions must be declared before they are used. This can either be done by defining them first or by using a declaration statement. (See examples below).

Functions must always have a return type of either LONG or DOUBLE. No VOID functions are supported at the moment.

The body of the function must be enclosed in braces.

All variables defined inside a function are local. That means they are not visible outside the function.

Local variables are stored temporarily on the stack. So defining a lot of local variables or local arrays may overflow the stack (see stack size in compiler options).

```
long test_func (long a, long b, double c) ;   // declaration statement
...
wert = test_func(a,b,c)                       // using the function
...
long test_func(long a, long b, double c)      // function definition
{
    long result, tmpval                       // local variables
    ....
    return(result)                            // here we return the value of the function
}
```

Functions can handle LONG or DOUBLE variables (or expressions) as parameters. This is done “by value”. This means that inside the functions, those parameters are locally defined.

Arrays can also be passed and this is done “by reference”. This allows the original array to be modified inside the function.

```
DIM test[100]
long modi(long[] myarray, long index)
{
    myarray[index] = myarray[index] * 10
    return(index)
}
...
modi(test,2)    // element 2 of test will be multiplied by 10
```

NB!:
Functions are supported starting with MCO 5.00.

□ Arithmetic

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, right-shift, 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
++	plus 1	expr++ = expr + 13	Post-increment (= Addition + 1)
–	minus	9 – 3 = 6	Subtraction
––	minus 1	expr–– = expr – 13	Post-decrement (= Subtraction – 1)
*	times	2 * 3 = 6	Multiplication
%	divided by	19 % 3 = 6	Division (result is truncated)
^	XOR	expr1 ^ expr2 127 ^ 255 = 128	Exclusive Or (binary operation)
mod	Modulo	expr1 mod expr2 250 mod 16 = 10	Mathematic modulo (rest of an integer division)
rnd	Division	expr1 rnd expr2 250 rnd 16 = 16	Division with round-off (opposite to truncating)
abs	Absolute amount	Abs(expr) abs (-5) = 5	Absolute amount of the expression

Post-increment

The compiler supports post increment and decrement for both long and double variables. So the following expressions are allowed. (This support depends only on the compiler version; the compiled program will run on both new and old firmware versions.)

```
while(index < 100) do
   test[index] = index++
endwhile
value-
```

This also works with arrays as in the following example.

```
DIM testarray[20]
...
var = testarray[5]++
testarray[10]
```

Here, the value of testarray[5] is copied into var and then the value of testarray[5] is incremented by one.

NB!:
The Post-increment operator is supported starting with MCO 5.00.

Double precision Operators

All operators (+,-,*,%,++,--) are available as double operators starting with MCO 5.00.

NB!:
Up to these compiler and firmware versions all arithmetical operations had been integer number operations.

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 7.1 = 1 7.3 = 1 7.4 = 0	Returns the Bit expr2 from expr1	1 - 32
.i	Bit	expr1.i expr2 7.i1 = 1	same as above, but syntax at full length	1 … 32
.b	Byte	expr1.b expr2 0x027F.b1 = 127 0x027F.b2 = 2	Returns the Byte expr2 from expr1	1 - 4

.ub		value = 0x1FFFE
		b = value.ub1
		// b is 254
			Returns "Byte" unsigned	

.sb		value = 0x1FFFE
		b = value.sb1
		// b is -2
			Returns "Byte" signed	

.w	Word	expr1.w expr2 0x0010FFFF.w2 = 16	Returns the Word expr2 from expr1	1 - 2
.uw			Returns "Word" designed	
.sw			Returns "Word" signed	
.l	Long	expr1.l expr2	Returns the Long expr2 from expr1 (standard)	

New Bit Operators

The bit operators .b and .w always delivered unsigned results. With the following bit operators.

.ub, .uw, .sb, .sw

unsigned or signed results can be explicitly choose, if desired.

Examples:

```
value = 0x1FFFE
b = value.ub1  // b is 254
b = value.sb1  // b is -2
```

NB!:
The new Bit operators are supported starting with MCO 5.00.

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 Value = 72	1 - 4
Value.b2	= 128	Second byte of value is set at 128 Value = 0x00008048	1 - 4
Value.w2	= 15	Second word of value is set at the value 15. Value = 0x000F8048	1 - 2

Priority of the Operators and the Operations

Operators in the same line have the same priority, thus they are completed from left to right.

The priorities are described in decreasing order:

exp						(exponential)
*	%	mod	rnd	.	BITP	(multiplicative) BITP = bit expression, e.g. .b or .u
+	–					(additive)
>>	<<					(bit-by-bit shifting)
>=	<=	>	<			(relation)
==	!=					(equality)
&						(bit-by-bit and)
^						(exclusive or, binary operation)
|						(bit-by-bit inclusive or)
AND						(logical and)
OR						(logical or)

☐ Preprocessor

The Preprocessor is designed to make various modifications to the program before it is actually compiled. Its most important functions are:

1. Handling continuation lines.
2. Including "header files" or other common blocks of code that have been saved in other files.
3. Text substitution (macro expansion).
4. Conditional compilation.

NB!:
Please note, that the preprocessor commands must be lower case only.

☐ Continuation lines

Any line that ends with a "\" will be continued on the next line.

Example:

```
PRINT " apos = ",apos,\
      " avel = ",avel
```

is the same as:

```
PRINT " apos = ",apos," avel = ",avel
```

☐ #include

Files can be embedded within a file by using the following statement:

`#include` "*file*"

`#include` <*file*>

where "*file*" is the name of the file to be embedded and may include directories. Either "\" or "/" may be used for directories.

If the quote syntax is used, then the Preprocessor will search for the include file in the same directory in which the file that included it was found. This is normally the same directory in which the original .m file was found. However, it could be another directory if nested include files are used that come from different directories. If the include file is not found in this directory, then the APOSS "system include" directory is searched. The APOSS system include directory is a subdirectory under the APOSS install directory.

If the angle bracket syntax is used, then the Preprocessor will search for the include file only in the APOSS "system include" directory.

Examples:

```
#include "defines.m"
#include "Library/Structures.m"
#include "Library\Structures.m"
#include <System.m>
```

NB!:
Please note that the syntax of the #include statement for versions of the APOSS IDE prior to MCO 305 is different. Before the statement contained no quotes. If an "old" program is opened using a new version, then quotes will be automatically added.

☐ #define, #undef

Simple text replacement can be done using the following statement:

`#define` name string

All instances of "*name"* that follow this statement will be replaced with "*string"*. Note that "*string*" will contain all characters (including spaces) up to the end of the line. The line may be continued using the "\" line continuation character if necessary.

Definitions can be cleared again using the following statement:

`#undef` *name*

The following names are predefined:

`__DATE__` - The date when the program was compiled (as a character string).
`__TIME__` - The time when the program was compiled (as a character string).
`__FILE__` - The filename of the program (as a character string).
`__LINE__` - The current line number in the program (as a number).

Example:

```
#define ARRAY_LEN 100
#define PRINT_MSG print " apos = ",apos," avel = ",avel
DIM prm[ARRAY_LEN]
PRINT_MSG
PRINT "This is line ",__LINE__
```

is the same as:

```
DIM prm[100]
PRINT " apos = ",apos," avel = ",avel
PRINT "This is line ",5
```

Text replacement is done recursively. Once the original string has been replaced, then the Preprocessor will look for other replacements within the new string. Note that text replacement is <u>not</u> done within the **#define** statement itself and is <u>not</u> done within strings enclosed in quotes.

Example:

```
#define VALUE 10
#define PRINT_MSG print "VALUE = ",VALUE
PRINT_MSG
#define VALUE 20
PRINT_MSG
```

is the same as:

```
PRINT "VALUE = ",10
PRINT "VALUE = ",20
```

Defines may also include one or more arguments separated by commas as follows:

`#define` name(arg1,arg2) string

In this case, instances of "*arg*" will be replaced in "*string*".

Example:

```
#define SQUARE(x) ((x)*(x))
#define SUM_SQUARE(x,y) (SQUARE(x) + SQUARE(y))
i = SUM_SQUARE(3,4)
i = SUM_SQUARE(a,b)
```

is the same as:

```
i = (((3)*(3)) + ((4)*(4)))
i = (((a)*(a)) + ((b)*(b)))
```

## □ #ifdef, #ifndef, #if

Program code can be conditionally compiled using any of the following statements:

`#ifdef` *name* - If "*name*" has been defined using **#define**.
`#ifndef` *name* - If "*name*" has not been defined using **#define**.
`#if` *integer-expression* - If "*integer-expression*" evaluates to non-0.

These can be followed by any number of statements and then **#endif**.
#else can also be used as is shown in the following example.

Example:

```
#ifdef INCLUDE_DEBUG
PRINT "Position = ",apos
PRINT "Velocity = ",avel
#endif

#ifdef START_AT_0
start = 0
#else
start = 1
#endif

#if 0
This will never be compiled.
#endif

#if 1
This will always be compiled.
#endif

#define OPTION 1
#if OPTION > 2
This will be compiled only if OPTION is greater than 2.
#endif
```

If **#if** is used, then **#elif** (for “else if”) can be used.

Example:

```
#define CUSTOMER_A 1
#define CUSTOMER_B 2

#define CUSTOMER CUSTOMER_B

#if CUSTOMER == CUSTOMER_A
Compile this for customer A.
#elif CUSTOMER == CUSTOMER_B
Compile this for customer B.
#else
Otherwise compile this.
#endif
```

□ APOSS Command Groups

All commands are described in detail and alphabetically ordered as well as complemented with short examples in the manual MCO 305 Command Reference.

□ Initialization Commands

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

Command	Description	Syntax	Parameter
DEFCORIGIN	sets the command position as real zero point.	DEFCORIGIN	–
DEFMORIGIN	sets the current master position as the zero point for the master	DEFMORIGIN	–
DEFORIGIN	sets the actual position as zero point	DEFORIGIN	–
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	–
RSTORIGIN	resets temporary zero point	RSTORIGIN	–
SAVE ARRAYS	save arrays in the EPROM	SAVE ARRAYS	–
SAVE AXPARS	save current axis parameters in the EPROM	SAVE AXPARS	–
SAVE GLBPARS	save current global parameters in the EPROM	SAVE GLBPARS	–
SAVEPROM	saves memory in EPROM	SAVEPROM	
SETMORIGIN	set the current position as the zero point for the master	SETMORIGIN value	value = absolute position
SETORIGIN	sets temporary zero point	SETORIGIN p	p = absolute position

□ 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
			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	

Command	Description	Syntax	Parameter
… ENDIF	end of the conditional program execution	ENDIF	

LOOP	repeats loop	LOOP n label	n = number of repetitions
			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
VLTALARMSTAT	returns if an alarm is active or not	VLTALARMSTAT	–
VLTCONTROL	sets the VLT control word in MOTOR OFF state	VLTCONTROL control word value	value
VLTERRCLR	clears a VLT-alarm	VLTERRCLR	–
WAITAX	waits until target position is reached	WAITAX	–

WAITI	waits for input	WAITI n s	n = input number
			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 (Group I/O)

Command	Description	Syntax	Parameter
IN	reads input bit-by-bit (individually)	res = IN n	n = input number
INAD	reads analog input	res = INAD n	n = number of analog input
INB	reads input by bytes (units of 8)	res = INB n	n = input number
INKEY	reads key codes of FC 300	INKEY p	p = 0 (wait for key code) p > 0 (wait for max. p ms) p < 0 (no wait for key code)

OUT	sets digital outputs bit-by-bit (single)	OUT n s	n = output number
			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
			v = value

OUTDA	sets FC 300 analog outputs	OUTDA n v	n = output number
			v = value

□ System State Functions

Commands for querying system status information such as drive position and velocity, system clock, error status, etc.. Commands for configuring and using arrays for recording test results. (Group SYS)

Command	Description	Syntax	Parameter
APOS	reads actual position	res = APOS	–
APOSDIFF	overflow handling of incremental encoders in applications	res = APOSDIFF oldpos	oldpos = APOS at a previous time
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	–
CPOSDIFF	overflow handling of incremental encoders in applications	res = CPOSDIFF oldpos	oldpos = CPOS at a previous time
ENCPOSOFFS	syncs the incremental position counter with the absolute counter in the encoder	res = ENCPOSOFFS offset	offset = returns the difference between absolute and incremental position
ENCTGREAD	reads a RS485 telegram from the encoder	res = ENCTGREAD array	array = user array where the received payload data should be put.
ENCTGWRITE	sends a RS485 telegram to the encoder	res = ENCTGWRITE length array	length = number of bytes (in the user array) to be sent array = user array containing the payload data to send to the encoder.
ERRNO	reads error number	res = ERRNO	–
IPOS	queries last index or marker position of the slave	res = IPOS	–

Command	Description	Syntax	Parameter
IPOSDIFF	overflow handling of incremental encoders in applications	res = IPOSDIFF oldpos	oldpos = IPOS at a previous time
JERKFINVEL	calculates the final velocity for a jerk-limited stop with maximum acc/dec.	res = JERKFINVEL	–
JERKSTOPDIST	calculates the necessary distance for a jerk-limited stop with max deceleration	res = JERKSTOPDIST dec	dec = sets deceleration in % or VELRES units
MAPOS	queries actual position of the master	res = MAPOS	–
MAPOSDIFF	overflow handling of incremental encoders in applications.	res = MAPOSDIFF oldpos	oldpos = MAPOS at a previous time
MAVEL	queries actual velocity of the master	res = MAVEL	–
MENCPOSOFFS	syncs the incremental position counter with the absolute counter in the encoder	res = MENCPOSOFFS offset	offset = returns the difference between absolute and incremental position
MENCTGREAD	reads a RS485 telegram from the encoder	res = MENCTGREAD array	array = user array where the received payload data should be put.
MENCTGWRITE	sends a RS485 telegram to the encoder	res = MENCTGWRITE length array	length = number of bytes (in the user array) to be sent array = the user array containing the payload data to send to the encoder.
MIPOS	queries last index or marker position of the master	res = MIPOS	–
MIPOSDIFF	overflow handling of incremental encoders in applications.	res = MIPOSDIFF x(n) oldpos	oldpos = MIPOS x(n) at a previous time
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
PRINTDEV	stops information output	PRINTDEV nn printlist	nn = no for the device: 0 = standard -1 = no output after that printlist = argument
STAT	reads axis status	res = STAT	–
SYNCERR	queries actual synchronization error of the slave	res = SYNCERR	–
TESTSETDEST	defines the memory section for saving recorded data	TESTSETDEST arrayname	arrayname = name of the array used for the recording or keyword DYNMEM for recor-ding data into free memory.
TESTSETINDEX	specifies system variables for data recording	TESTSETINDEX svi1, svi2, svi3, ..., svin	svi1…svin = indices of maximum 20 system variables (sysvar[...]) to be recorded.
TESTSETP	specifies recording data for test run	TESTSETP ms vi1 vi2 vi3 arrayname	ms = interval in ms vi 1 … 3 = indices of the values to be recorded arrayname = array used for recording

Command	Description	Syntax	Parameter
TESTSETTIME	defines the sampling period for data recording	TESTSETTIME ms	ms = sampling period in ms
TESTSETTYPE	defines "one time" or "cyclic" recording	TESTSETTYPE type	type: 0 = one time recording 1 = cyclic recording
TESTSTART	starts the recording of a test run	TESTSTART n	n = 0 The max. number of samples is recorded, which fits into the defined memory section. n > 0 No. of samples to be recorded
TESTSTOP	stops the data recording	TESTSTOP method	method: 0 = stops immediately
TIME	reads internal controller time	res = TIME	–
TRACKERR	queries actual position error of an axis	res = TRACKERR	–
_GETVEL	changes sample time for AVEL and MAVEL	var = _GETVEL t	t = sample time in ms

Interrupt Functions

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 CANINPUT id GOSUB	calls up a subprogram when a CAN telegram type 'id' arrives	ON CANINPUT id GOSUB name	id = 0
			name = subroutine

ON CANMSG GOSUB	verifies arrival of a buffered CAN message	ON CANMSG GOSUB name	name = name of the subroutine
ON COMBIT .. GOSUB	call up a subprogram when Bit n of the communication buffer is set	ON COMBIT n GOSUB name	n = bit n of communication buffer name = subprogram

ON DELETE .. GOSUB	deletes a position interrupt: ON posint GOSUB	ON DELETE pos GOSUB name	pos = value
			name = subprogram name

ON ERROR GOSUB	calls subroutine in the event of an error	ON ERROR GOSUB name	name = subprogram name
ON INT .. GOSUB	calls subroutine depending on input signal	ON INT n GOSUB name	n = input to be monitored name = subprogram name
ON KEYPRESSED GOSUB	interrupts when a LCP key is pressed or released	ON KEYPRESSED GOSUB name	name = subprogram name

ON PARAM .. GOSUB	calls up a subprogram when a parameter is altered	ON PARAM n GOSUB name	n = parameter number
			name = subprogram name

ON PERIOD GOSUB	calls subroutine at regular intervals	ON PERIOD n GOSUB name	n =
			n > 20 ms (time for re-call)
			n = 0 (switch off function)
			name = subprogram name

ON posint .. GOSUB	calls up a subprogram when a position interrupt occurs: ON APOS = when the slave position xxx is passed ON IPOS = distance between the last marker position and the actual position ON MAPOS = when the master position xxx [qc] is passed ON MCPOS = when the master position xxx [MU] is passed ON MIPOS = distance between two markers is reached.	ON sign postype position GOSUB name	sign= + = rising edge or when the position is passed in positive direction – = falling edge or when the position is passed in negative direction postype = APOS,IPOS, MAPOS MCPOS, MIPOS position = depending on the command in user units [UU], or master user units [MU], or curve units [CU] name = subprogram name

Command	**Description**	**Syntax**	**Parameter**
ON posint SETOUT	Simulates a cam box (all types of position interrupts: APOS, IPOS, MAPOS, MCPOS, MIPOS)	ON +/- type position SETOUT outno	type = any POSINT position = depending on the command in user units [UU], or master user units [MU], or curve units [CU]

ON posint SETOUT (TOIN)		ON +/- type position SETOUT outno TOIN inno	outno = kann jede gültige Nummer eines Ausgangs sein (oder negative Nummer des Ausgangs)
			inno = kann jede gültige Nummer eines Eingangs sein (oder neg. Nummer des Eingangs)

ON STATBIT .. GOSUB	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 name = subprogram

ON TIME .. GOSUB	calls subroutine after single timing	ON TIME n GOSUB name	n = time for re-call
			name = subprogram

□ Commands for Parameter Handling

All parameters with a parameter code can be set and read with the following commands. (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
			indxno = index number

LINKGPAR	links global parameter with LCP display	LINKGPAR parno "text" min max type	parno = LCP par. Number
			text = ASCII text
			min = min. value
			max = max. value
			type = online / offline par.

LINKPDO	link the system variable with PDO and copy in the internal parameters	LINKPDO no len indx pdo	no = rank order in RxPDO
			len = number of the bits to be imported
			indx = index of the system variable SYSVAR
			pdo= values between 1 .. 4 or 5 (serial)

LINKSDO	copying an internal object in the PDO	LINKSDO indx len no "text" pdo	indx = index of the system variable SYSVAR len = length of the bits to be imported no = rank order in the PDO text = " " (comment) pdo = values between 1 .. 4 -pdo = dummy entry of the link object into the mapping list

LINKSYSVAR	links system variable with LCP display	LINKSYSVAR indx parno "text"	indx = SYSVAR index
			parno = LCP par. Number
			text = display text

SET	sets parameter values (MCO 305 and application parameter)	SET par v	par = par. Identification
			v = parameter value

SETVLT	sets FC 300 parameter values	SETVLT par v	par = parameter number
			v = parameter value

SETVLTSUB	sets FC 300 parameter values with index number	SETVLTSUB par indxno v	par = parameter number
			indxno = index number
			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 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 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
DEFSYNCORIGIN	Defines master-slave relation for the next SYNCP or SYNCM command.	DEFSYNCORIGIN master slave	master = reference position in qc slave = reference position
MOVESYNCORIGIN	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)
SYNCM	Synchronization of angle/position with marker correction.	SYNCM	–
SYNCMARKERSTART	Resets a marker or resets marker handling.	SYNCMARKERSTART restarttype	restarttype = 0 or 1
SYNCP	Synchronization of angle/position.	SYNCP	–
SYNCSTAT	Queries flag for synchronization status.	res = SYNCSTAT	
SYNCSTATCLR	Resetting of the flags MERR and MHIT.	SYNCSTATCLR value	value = 8 = SYNCMMHIT 16 = SYNCSMHIT 32 = SYNCMMERR 64 = SYNCSMERR
SYNCV	Synchronization of velocity.	SYNCV	–

☐ 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 slavepos = slave position after disengaging

□ CAN Basic library

Raw Telegram mode

This library contains all the functions necessary to do raw CAN telegram transfer. So you can define mailboxes with the commands DEFCANIN and DEFCANOUT for reading and writing telegrams. Those commands allow the length and telegram id (standard 11 bit identifiers) to be chosen. Once defined, you can use those mailboxes to receive and send those telegrams using the commands CANOUT and CANIN.

The CANIN command allows the user to use remote frames and to define if he wants to wait for an answer or not. You can also define the time out for the reading.

The command CANDEL can be used to delete already defined mailboxes.

Command	Description	Syntax	Parameter
CANDEL	erases all or single CAN objects	CANDEL objno	objno = object number, which is returned during object definition = –1 deletes all objects (except standard objects)
CANIN	reads an object	status = CANIN objno timeout control varhi varlo	objno = object number, which is returned during object definition timeout = -1 = no wait for data 0 = waits until data arrives >0 = waits for data in timeout control = 0 = copies new arrived data to the variables 1 = sends a remote frame and waits for data in dependence on *timeout* varhi = bytes 0 to 3 of the CAN object data varlo = bytes 4 to 7 of the CAN object data

CANOUT	sends message (active)	CANOUT no valhi vallo	valhi = bytes 0 to 3 of the CAN object data
			vallo = Bytes 4 to 7 of the CAN object data

DEFCANIN	defines a receive object	objno = DEFCANIN id dlen	id = CAN-id dlen = data length of the object in bytes (max. 8 bytes)
DEFCANOUT	defines a transmit object in the CAN controller	objno = DEFCANOUT id dlen	id = CAN-identification number dlen = data length of the object in bytes (max. 8 bytes)

Buffered Message Transfer

This message transfer is used by APOSS. But if APOSS is not active, it could also be used for application purposes. But you must be aware that buffered message handling expects that all telegrams are really read. Otherwise, a full buffer can stop program execution because, for example, an OUTMSG has to wait until buffer allows writing again.

There are three telegrams used for this sort of communication.

GlobalMessage	telegram ID = 0	8 Byte length
RxMessage	telegram ID = CANNR * 2 + 1	8 Byte length
TxMessage	telegram ID = CANNR * 2	

GlobalMessages are read by all controllers. The first two bytes may not be used by applications. These bytes must always be 0 or you may have unwanted side effects.

The following functions handle buffered transfer:

Command	Description	Syntax	Parameter
INGLB	reads Glb CAN message	res = INGLB (p)	p = time-out, defined ... 0 = waits until a message arrives > 0 = waits a maximum of p milliseconds 0 = no wait for a message
INMSG	reads CAN message (polling)	intval = INMSG timeout	timeout = < 0 = does not wait for data 0 = waits until data arrives > 0 = waits for data in timeout [ms]
MSGVAL	supplies the long value of the last message	longval = MSGVAL	–
ON CANINPUT	calls up a subprogram when a CAN telegram type 'id' arrives	ON CANINPUT id GOSUB name	id = 0 name = name of the subroutine

OUTMSG	sends CAN message (polling)	OUTMSG intval longval	intval = bytes 2 and 3 of the CAN-message
			longval = bytes 4 and 7 of the CAN-message

USRSTAT	sets user status	USRSTAT val	val = value to be set

SDO2

A second SDO is supported to communicate with debug devices like SDO monitors.

This second SDO uses the following CobId:

SDO2-Tx = 0x681 – 0x6FF
SDO2-Rx = 0x101 – 0x1FF

To enable the use of SDO2, you must first write the above addresses to the Objects

ServerSdo2Parameters - Index 0x1201 Subindex 0x01 (CobId Rx) and 0x02 (CobId Tx).

If you read these parameters, then you will get the above mentioned CobIds with the highest bit set. This bit is the NOT_VALID bit.

It is not possible to change these CobIds; you only can overwrite them with the same value with the NOT_VALID bit set to zero.

It is possible to disable these SDO's again by resetting the NOT_VALID bit.

Using SDO2 does not need extra objects in CAN-Memory. Receiving is done by the universal Rx-Mailbox. Transmitting is also done by the universal Tx-Mailbox.

This enabling or disabling could also be done from within the APOSS program:

```
SDO2_txid = 0x680 + (get CANNR)
SDO2_rxid = 0x100 + (get CANNR)
COB_NOT_VALID = 0x80000000
      // enable SDO2
      sysvar[0x01120101] = SDO2_txid
      sysvar[0x01120102] = SDO2_rxid

      // disable SDO2
      sysvar[0x01120101] = (SDO2_txid | COB_NOT_VALID)
      sysvar[0x01120102] = (SDO2_rxid | COB_NOT_VALID)
```

☐ CANopen Master library

This library has functions to support the usage of CANopen Slaves like I/O modules or encoders or even drives. Those functions do not only allow the configuration with SDOs but do allow the usage of process data through PDOs.

Functions SDOREAD and SDOWRITE are used to read or write the object dictionary of any slave. Even an SDOREADSEG and SDOREADSEGP is supported which allows segmented reads (either unpacked or packed) to be done.

Those commands allow the user to configure and command CAN slaves on the bus. So the user could, for example, set another drive into a special mode or start the drive with a given speed (if the drive has a CAN option).

To allow easy usage of external I/O over CAN bus, the normal IN and OUT commands are extended. If you specify a number bigger than 256, then automatically a CAN I/O is assumed. So by using the command OUT 257 1 you can set the output number 1 on the CAN I/O module with the Node ID 1.

The output number equals (node_Id * 256 + output_no). The same is true for INAD and OUTDA. So any kind of analogue input and output modules can also be used.

Furthermore, you have the possibility to register a CANopen module with the command CANINI. Doing this, the module is automatically guarded in the background. This also makes the IO traffic faster and you can even use an ON INT for a CANopen input.

The following commands and parameters realize master functionality:

Command	Description	Syntax	Parameter
CANIN (>100)	Gathers all Transmit-PDOs of digital input modules or CAN drive status by using only one CAN telegram.	result = CANIN (id * 100) timeout control varhi varlo	id = CAN id timeout = -1 = no wait for data 0 = waits until data arrives >0 = waits for data in timeout control = 0 = copies new arrived data to the variables 1 = sends a remote frame and waits for data in dependence on *timeout* varhi = bytes 0 to 3 of the CAN object data varlo = bytes 4 to 7 of the CAN object data

CANINI,	Initializes the necessary objects (PDOs) for data exchange of CANopen nodes, or enables extended CANINI, CANIN function.	CANINI no, no, …,
		CANINI 999
		CANINI no, 999, …
			no = guard * (busoffset * 1000 + id)
			guard = -1, +1 (without / with guarding)
			busoffset = 100000 , 0 (slave bus, master bus)

SDOREAD	Reads SDO of a connected CANopen device.	val = SDOREAD id index sub	id = CAN id (1…127)
			-id = executes the command without waiting for the answer
			index = index of object
			sub = sub index (0x00 – 0xFF)

SDOREADSEG	Segmented read of SDOs (unpacked).	res = SDOREADSEG id, index, subindex, arrayname	id = CAN id number
			-id = executes the command without waiting for the answer
			index = 0x2000
			subindex = parameter number
			arrayname = name of a existing array

Command	Description	Syntax	Parameter

SDOREADSEGP	Segmented read of SDOs (packed).	res = SDOREADSEGP id, index, subindex, arrayname	id = CAN id number
			-id = executes the command without waiting for the answer
			index = 0x2000
			subindex = parameter number
			arrayname = name of a existing array

SDOSTATE	Checks the result of an active communication.	res = SDOSTATE id value	id = CAN id
			value = parameter value

SDOWRITE	Sets SDO of a connected CANopen device.	SDOWRITE id index sub val	id = CAN id (1…127)
			-id = executes the command without waiting for the answer
			index = index of object
			sub = sub index
			val = parameter value

The following functions support master functionality:

Command	Description	Syntax	Parameter
IN	Reads status of digital input.	res = IN n	n = input number or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + input number (or input byte)
INB	Reads one byte from digital inputs.	res = INB n	n = input byte or with CAN open I/O modules: as above
OUT	Set or re-set digital outputs.	OUT n s or OUT X/n s	n = output number or with CANopen I/O modules: CAN-Bus + (Module-CAN-ID * 256) + output number X/n = terminal / relay number s = condition (0 = OFF, 1 = ON)
OUTB	Alteration of the condition of a digital output byte.	OUTB n v	n = output byte, or with CAN open I/O modules: CAN-Bus + (Module-CAN-ID * 256) + output CAN-Bus: 0 = Master bus v = value (0 ... 255)
SET ENCODERTYPE	SET par. 32-00 *Incremental Signal Type.*		

☐ CANopen Slave library

This library contains all functions which are necessary to act as a CANopen Slave. With special application software, it would even be possible to behave like a DS402 drive.

To make that possible, a complete SDO dictionary is implemented, all functions necessary for supporting PDO1..4, dynamic mapping of PDOs and transmission types supporting SYNC transfer, as well as inhibit time and event time.

To use those functions, it is of course necessary that there is a unique node ID (par. 33-90 CANNR) on the bus.

This functionality, for example, allows very easy communication between two controllers. So one controller could easily write into the others object dictionary and vice versa. Or the PDO command could be used to transmit PDO's which can be easily read by all other controllers using, for example, the extended CANINI 999 function. This gives you nearly unlimited possibilities to make fast and efficient CAN communication between multiple controllers.

The slave library also supports a SDO2 object which allows APOSS to communicate over a second SDO while a PLC (or other controller) is communicating over SDO. This is very useful for debugging and development of applications.

The very powerful commands LINKSDO and LINKPDO allow to define which SDO should be mapped into an outgoing PDO and which part of an incoming PDO is automatically mapped into which object in the dictionary. Because the object dictionary also contains for example the User Parameters, it is very easy to update the user parameters in the background when a PDO is received without any action of the application.

The following commands realize the CANopen slave functionality:

Command	**Description**	**Syntax**	**Parameter**
LINKPDO,	Link the system variable with RxPDO and copy in the internal parameters, or link part of an array into the PDO.	LINKPDO no len indx pdo	no = rank order in the RxPDO len = number of the bits to be imported indx = index of the system variable SYSVAR pdo = values between 1 .. 4 or 5 (serial)
LINKSDO,	Link TxPDO with internal system variable, or link part of an array out of the PDO.	LINKSDO indx len no "text" pdo	indx = index of the system variable SYSVAR len = length of the bits to be imported no = rank order in the PDO text = " " (comment) pdo = values between 1 .. 4 -pdo = dummy entry of the link object into the mapping list
PDO[ ]	Pseudo array for direct access to the CANopen PDOs.	PDO[n]	n = 1001. for 1. PDO 2001. for 2. PDO … 5001. for 5. PDO (serial PDO)
SYSVAR[0x01…]	System variable (Pseudo array) reads system values; see SDO-Dictionary		

□ Get a General Idea of Program Samples

Some samples are described in the manual "Operation Instructions" or in the chapter "Functions and Samples" in this Design Guide. All other samples are you will find in the online help. Many of the samples are delivered with the APOSS software, too. Open it with *File → Sample*.

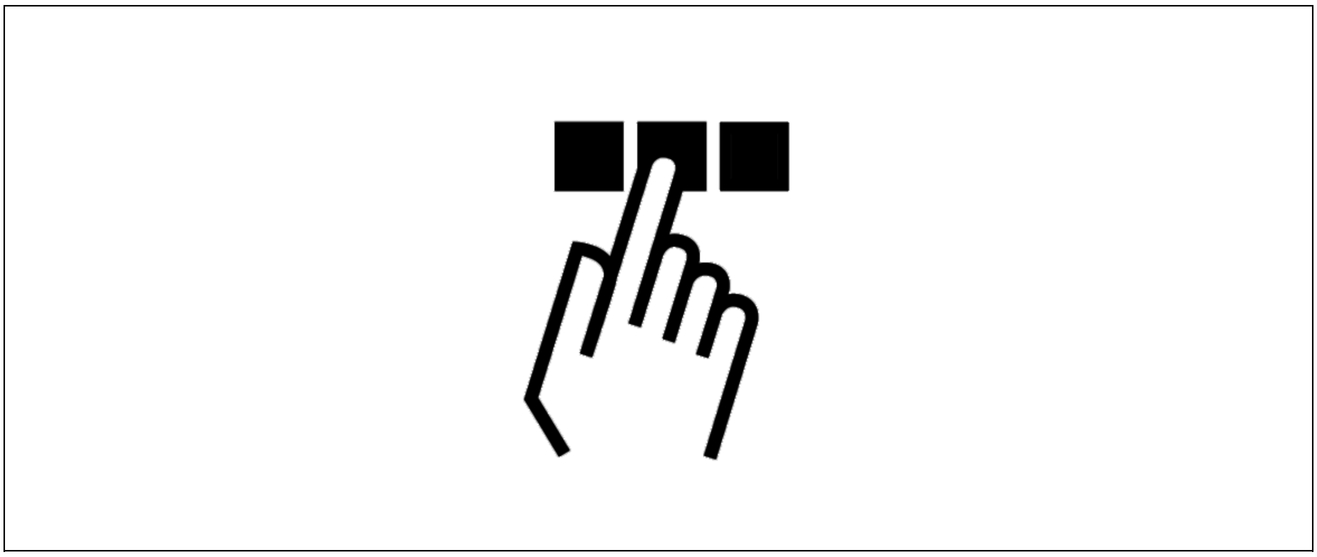
All samples can be copied and pasted into the APOSS software. Please read the safety instructions before using the samples!

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 46	CAM Box
CANIN.M	x		Defines an interrupt procedure which is called up when a CAN message is received.
CANOUT.M	x		Defines an object to be sent.
CAN_sample.M	x		Communication between 2 controllers.
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).
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.

File name or topic	Where to find?		Description
	Online help	Manual or Software	
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: Displays the status of inputs (graphic)
INCPOS01.M	x		Include file: Display of the current and command position.
INCSTA01.M	x		Include file: 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.
JerkMinTest.M	x		Test program for recording of movements with different ramp types (-> parameter RAMPTYPE).
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 47	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.
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.

File name or topic	Where to find?		Description
	Online help	Manual or Software	
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.

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 EPROM* 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
- CAN 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 EPROM 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 EPROM 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 EPROM 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 0-21 0-22 0-23 0-24	[1990] - [1999] [3400] - [3410] [3421] - [3430] [3440] - [3441] [3450] - [3462]	-
1-02	[4] MCO Encoder 1 [5] MCO Encoder 2	-
1-62	-	0%
3-15 3-16 3-17	-	[0] No function

Par. number	New selections	New default
3-41 3-42 3-51 3-52 3-61 3-62 3-71 3-72	-	0.01 s
4-10	-	"Both Directions"
5-10 5-11 5-12 5-13	-	[0] No operation
5-30 5-31 5-32 5-33	[51] MCO Controlled	[51] MCO Controlled
5-40	[51] MCO Controlled	[51] MCO Controlled
5-60 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 [5] MCO Encoder 2	-
8-02	[5] Option C0	[5] Option C0
9-15	[3401] - [3410]	Index [0] 3401 Index [1] 3402 Index [2] 3403 Index [3] 3404 Index [4] 3405 Index [5] 3406 Index [6] 3407 Index [7] 3408 Index [8] 3409 Index [9] 3410
9-16	[3421] - [3430]	Index [0] 3401 Index [1] 3402 Index [2] 3403 Index [3] 3404 Index [4] 3405 Index [5] 3406 Index [6] 3407 Index [7] 3408 Index [8] 3409 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 201
32-3*	Encoder 1 - Master	page 202
32-5*	Feedback Source	page 209
32-6*	PID controller	page 210
32-8*	Velocity & Acceleration	page 213

33-**	**MCO Advanced Settings**	
33-0*	Home Motion	page 217
33-1*	Synchronization	page 218
33-4*	Limit Handling	page 229
33-5*	I/O Configuration	page 231
33-8*	Global Parameters	page 235
33-9*	MCO Port Settings	page 237

34-**	**MCO Data Readouts**	page 238
34-0*	PCD Write Parameters	page 238
34-2*	PCD Read Parameters	page 238
34-4*	Inputs & Outputs	page 239
34-5*	Process Data	page 239
34-7*	Diagnosis Readouts	page 240

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]
	CAN encoder	[3]

Function

Specifies type of the incremental encoder connected to Encoder 2 interface (X55 and X62 if CAN encoder is used).

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.

Select *CAN encoder* [3] if a MCO CAN encoder is used.

ENCODERTYPE == 9 is supported starting with MCO 5.00. This slave simulation can be used for testing. This parameter cannot be selected in the menu; it is set in the APOSS program.

Using 2-digits parameters allows assigning the encoder, if one wishes to alternately use two motors with one control (instead of the previous SWAPMENC command).

SET ENCODERTYPE 1n or
SET ENCODERTYPE 2n

NB!:
Prior to the commands to change the encoder 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.

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.

This parameter setting cannot be selected in the menu; it is set in the APOSS program.

Sample

```
OUT 1 1             // switch motor leads
SET KPROP ...       // change axis parameters
SET ENCOERTYPE 11      // assign encoder 1
MOTOR ON            // turn on control again
POSA 10000
   // move the motor which is connected
   // to the master encoder
MOTOR OFF
OUT 1 0             // Switch motor leads
SET KPROP ...       // Change axis parameters
SET ENCOERTYPE 21      // assign encoder 2
MOTOR ON            // Turn on control again
POSA 0
   // Move the motor again which is connected
   // to the slave encoder
```

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). Encoder resolution can be found on encoder nameplate or datasheet.

– 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.
– CAN encoder (32-00 = [3]):
Incremental encoder: Pulses per revolution
Absolut encoder: (Pulses per revolution)/4

NB!:
The parameters for the incremental resolution (32-01 or 32-31) are always used, even if the CAN encoder is an absolute encoder. But <u>a quarter of the encoder resolution must be set</u> for a CAN absolute encoder! The reason is the internal calculation, which uses the four times of the number of counts, because an incremental encoder returns 4 times more quad-counts than its counts. An absolut encoder returns “only” this real resolution as maximum value.

When *Motor Control* [3] is selected in par. 32-50 *Source Slave*, then the resolution can be set with this parameter.

NB!:
When using *Motor Control* (SPI Encoder) the resolution value must be a second power, otherwise rounding errors lead to positioning drifts (e.g. SYNCP).

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]
Hiperface	[1]
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 *Hiperface* [1] if this type of an absolute encoder is connected. The selection includes the default settings encoder ID “1” and encoder parity “even”.

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 communi-cation/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: DEFORIGIN, HOME, INDEX, IPOS, and WAITNDX.

Note: Selection [1] *Hiperface* encoder is available with version 6.7.40.

32-03 Absolute Resolution

ENCODERABSRES

Range	
1 … MLONG	∗ 8192

Function

The encoder resolution is used to calculate velocity in RPM (rounds per minute).

∗ default setting **[ ] value for use in communication via serial communication port**

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

32-04 Absolute Encoder Baudrate X55

ENCODERBAUD

Option		
	600	[0]
	1200	[1]
	2400	[2]
	4800	[3]
∗	9600	[4]
	19200	[5]
	38400	[6]

Function

Select the baud rate of the attached encoder.

Note: This parameter is available with firmware version 6.7.40.

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 bits.

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

∗ default setting [ ] value for use in communication via serial communication port

32-09 Encoder Monitoring

ENCODERMONITORING

Option

∗	Off	[0]
	3 Channels	[1]
	2 Channels	[2]

Function

Monitoring of open- and short-circuit of the encoder inputs can be enabled or disabled.

Select *Off* [0] if hardware monitoring not is required.

Select [1] if all 3 channels A,B, and Index should be monitored.

Select [2] if 2 channels A,B should be monitored.

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.

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

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.

The factor is no longer limited to small values starting with version MCO 5.00, see also User Units in chapter „How to read the Design Guide".

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.

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

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.

The factor is no longer limited to small values starting with version MCO 5.00, see also User Units.

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

The factor is no longer limited to small values starting with version MCO 5.00, see also User Units in chapter „How to read the Design Guide".

32-13 Enc.2 Control

ENCCONTROL - 107

Option

*	No soft changing	[0]
	Encoder soft changing enable	[1]
	Soft zero setting enable	[2]
	Encoder soft changing and soft zero enable	[3]

Function

The Encoder Control Word configures the position evaluation after a change of encoder source.

Soft encoder changing is useful if encoders should be switched while running. If this is done without using this parameter, then setting the new encoder will typically cause a position error because the encoder values are not the same.

Select *No soft changing* [0] to switch directly to the position data of the new source.

Select *Encoder soft changing* [1] not to switch entirely to the value of the new encoder. Instead, the old value is kept and the differences from the new encoder is just added. This makes it possible to change encoders "on the run".

Select *Soft zero setting* [2] if the encoder is used by another axis or for some other purpose and it is not desired to really change the encoder value when a DEFORIGIN is used. If soft zero setting is on, then a DEFORIGIN can be used and the new reported actual position is zero afterwards. However, the encoder still has its old value (i.e. the value you can see if you read it by using the SYSVAR 0x0122022A or similar).

Selection *Encoder soft changing and soft zero* [3] gives possibility for bump less changing of feedback encoder in software while running and setting position to zero without loosing actual position.

32-14 Enc.2 node ID

Encoder node ID

Range

1 – 127	∗ 127

Function

Enter the feedback CAN encoder node ID.

32-15 Enc.2 CAN guard

Encoder CAN Guard

Option

∗	Off	[0]
	On	[1]

Function

Feedback CAN encoder guardians can be enabled or disabled.

Off [0] is the default setting. Select [1] if feedback CAN encoder should be monitored.

◻ 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]
	CAN encoder	[3]

Function

Specifies type of the incremental encoder connected to Encoder 1 interface (X56 and X62 if CAN encoder is used).

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 *CAN encoder* [3] if an encoder with a MCO CAN interface is connected.

Using 2-digits parameters allows to assign the encoder (instead of the previous SWAPMENC command):

SET MENCODERTYPE 11 or
SET MENCODERTYPE 21

This parameter cannot be selected in the menu; it is set in the APOSS program.
See sample par. 32-00.

<u>Handling of the automatically generated CAN objects</u>

The necessary CAN objects (PDOs, GUARD-, SYNC-object) for the information transfer of the actual position via the bus are generated automatically and the bus node is also initialized (NMTO). This object generation, which is done in background, corresponds basically to the command CANINI. In contrast to the application object generation using CANINI the automatic generated objects by MENCODERTYPE remain untouched even when an updated CANINI is done by the application later on. Even a CANDEL –1 command does not delete or stop the objects, which are automatically established by the MENCODERTYPE setting.

The automatic generated objects can just be deleted or reconfigured by a new differing SET MENCODERTYPE. Also in the case of program abort (by means of [Esc]) the objects remain existent up to the next program start. The start of a program deletes any formerly defined objects and initializes new objects according to the setting of the permanent parameters or at run-time according to any temporary parameter definition inside the program code.

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). Encoder resolution can be found on encoder nameplate or datasheet.

- Digital incremental encoder (32-00 = [1]): The resolution must be set in Pulses Per Revolution.
- CAN encoder (32-00 = [3]):
  Incremental encoder: Pulses per revolution
  Absolut encoder: (Pulses per revolution)/4

NB!:
The parameters for the incremental resolution (32-01 or 32-31) are always used, even if the CAN encoder is an absolute encoder. But a quarter of the encoder resolution must be set for a CAN absolute encoder!

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]
Hiperface	[1]
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 *Hiperface* [1] if this type of an absolute encoder is connected. The selection includes the default encoder ID “1” and encoder parity “even”.

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 not be used with absolute encoder.

Note: Selection [1] Hiperface encoder is available with version 6.7.40.

32-33 Absolute Resolution

MENCODERABSRES

Range [Unit]	
1 – MLONG [PPR]	∗ 8192

Function

Note: Parameter only visible when par. 32-32 ≠ 0

32-34 Absolute Encoder Baudrate X55

MENCODERBAUD

Option	
600	[0]
1200	[1]
2400	[2]
4800	[3]
∗ 9600	[4]
19200	[5]
38400	[6]

Function

Select the baud rate of the attached encoder.

Note: This parameter is available with firmware version 6.7.40.

∗ default setting [ ] value for use in communication via serial communication port

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]
	3 Channels	[1]
	2 Channels	[2]

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 [1] if all 3 channels A,B, and Index should be monitored.

Select [2] if 2 channels A,B should be monitored.

An encoder error will issue (error 192).

32-40 Encoder Termination

MENCODERTERM

Option

	Off	[0]
∗	On	[1]

∗ default setting   [ ] value for use in communication via serial communication port

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.

32-43 Enc.1 Control

MENCCONTROL

Option

∗	No soft changing	[0]
	Encoder soft changing enable	[1]
	Soft zero setting enable	[2]
	Encoder soft changing and soft zero enable	[3]

Function

Configuration of master position evaluation after a change of encoder source.

Soft encoder changing is useful if encoders should be switched while running. If this is done without using this parameter, then setting the new encoder will typically cause a position error, because the encoder values are not the same.

Select *No soft changing* [0] to switch directly to the position data of the new master source.

Select *Encoder soft changing* [1], then it does not switch entirely to the value of the new encoder, but it keeps the old value and just adds the differences from the new encoder. This makes it possible to change encoders "on the run".

Select *Soft zero setting* [2], if the encoder is used by another axis or for some other purpose and you don't want to really change the encoder value when you use a DEFMORIGIN. If soft zero setting is on, then you can use a DEFMORIGIN and the new reported actual position is zero afterwards. However, the encoder still has its old value (i.e. the value you can see if you read it by using the SYSVAR 0x0122022A or similar).

Selection *Encoder soft changing and soft zero* [3] gives possibility for bump less changing of master encoder in software while running and setting position to zero without loosing actual position.

32-44 Enc.1 node ID

Encoder 1 node ID

Range

1 – 127	∗ 127

Function

Enter the master CAN encoder node ID.

With 14-22 reset this setting must not be changed.

32-45 Enc.1 CAN guard

Encoder 1 CAN guardians

Option

∗	Off	[0]
	On	[1]

Function

On/off selection of master CAN encoder guardians.

Off [0] is the default setting. Select [1] if feedback CAN master encoder should be monitored.

□ 32-5* Feedback Source

32-50 Source Slave

Option

	Encoder 1 X56	[1]
∗	Encoder 2 X55	[2]
	Motor Control	[3]

Function

Choose the feedback source for MCO.

Choose [1] for Encoder 1.

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.
The resolution for *Motor Control* can be set in par. 32-01 *Incremental Resolution*.

Note: Option [2] is available with firmware version 6.7.40.

32-52 Source Master

Option

∗	Encoder 1 X56	[1]
	Encoder 2 X55	[2]
	Motor Control	[3]

Function

Set the parameter Source Master:

Choose [1] for Encoder 1 on X56.

Choose [2] for Encoder 2 on X55. 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.

Note: The parameter is available with firmware version 6.7.40.

□ 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.

∗ default setting [ ] value for use in communication via serial communication port

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 feed-back 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 restriction is always valid, i.e. for the drive synchronization (SYNCP, SYNCV, SYNCM, SYNCC, etc.), for positioning commands (POSA, POSR), the speed command (CVEL), and also for a test run with the *Oscilloscope.*

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 position 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)

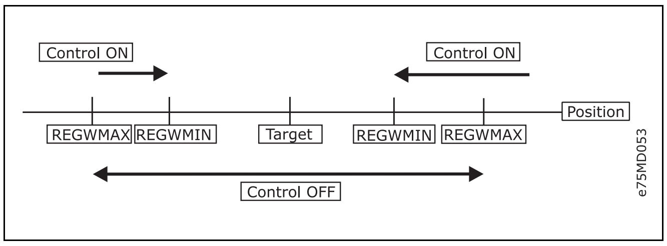
REGWMAX

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.



∗ default setting [ ] value for use in communication via serial communication port

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.

32-73 Integral limit filter time

KILIMTIME

Range [ms]

-10000 … 10000 ∗ 0

Option

∗	Integral part of PID position regulation always active.	[0]
	< 0 = Integral part of PID position regulation just active at standstill	[1]
	> 0 = Integral part of PID position regulation just active during movement	[2]

Function

Time in ms which is used to increase or decrease the integral limit of the position control loop up to KILIM.

The integral part of the PID *position control loop* can be active all the time or just during a movement or just at standstill. The value of this parameter determines this behavior.

Using value 0 activates the integral part of the PID position control loop all the time according to the parameter 32-62 KINT and the limitation parameter 32-63 KILIM.

Using a value > 0 activates the integral part of the PID position control loop just during a motor movement. If the motor is in standstill, the integral part is reduced to zero. If the motor starts to move, the integration limit is increased from zero to the defined KILIM value within the period of time given by the KILIMTIME setting. If the motor stops again the integral part is reduced again by decreasing the limit to zero within the defined period.

This handling of the integral part can be an advantage for synchronization applications, where low synchronization errors are requested, but no hard regulation at standstill is wanted.

Using a value < 0 activates the integral part of the PID position control loop just at standstill. If the motor starts to move, the integral part is reduced to zero by decreasing the integration limit within the period of time given by the absolute value of the KILIMTIME setting. If the motor stops again, the integration limit increased from zero to the defined KILIM within the defined period of time. This handling of the integral part can be helpful to prevent nervous drive behavior during a movement, but still get a very accurate positioning result at standstill.

See also p. 32-60 KPROP and p. 32-61 KDER.

32-74 Position error filter time

POSERRTIME

Range [ms]

0 … 10000 ∗ 0

0 = no time frame active, i.e. error state is triggered immediately.

Function

Time frame [ms] for triggering position error state.

Too big tracking errors (cp. POSERR) will just trigger error state, if they exist longer than POSERRTIME..

This parameter has a default of 0. If this parameter is not 0, then a position error is only produced if the position error value (par. 32-67) is exceeded for a time longer than POSERRTIM. Internally the position error is checked every 20 ms, that means a detection of a position error can take up to NoOfAxes * 20 ms in the worst case. If you have for example the POSERRTIME set to 300 ms, then we check if a position error has been detected more than (300 / 1 * 20) = 15 times.

If this is true then an error is generated.

□ 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

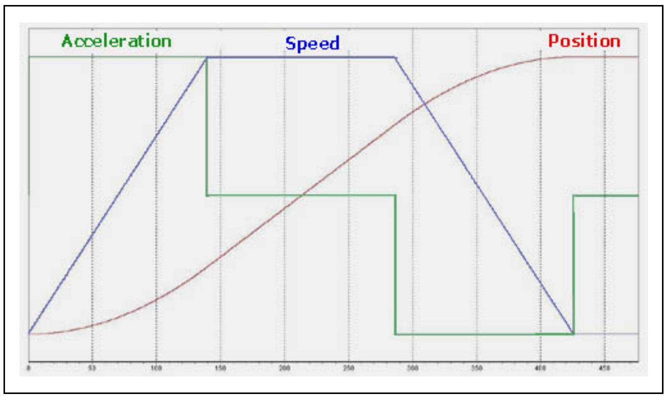
RAMPTYPE - 32

Option

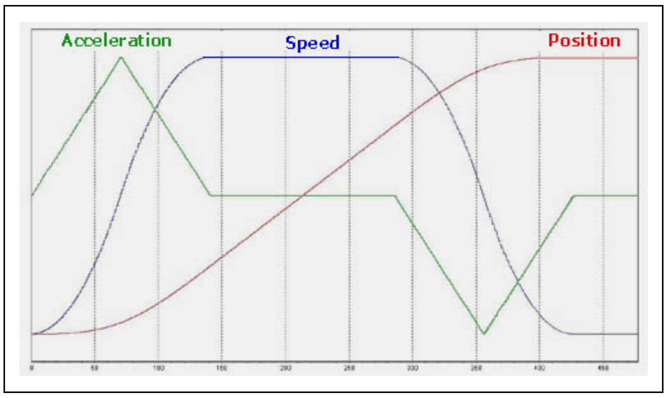
∗	Linear	[0]
	S-ramp	[1]
	Movements with limited jerk	[2]

Function

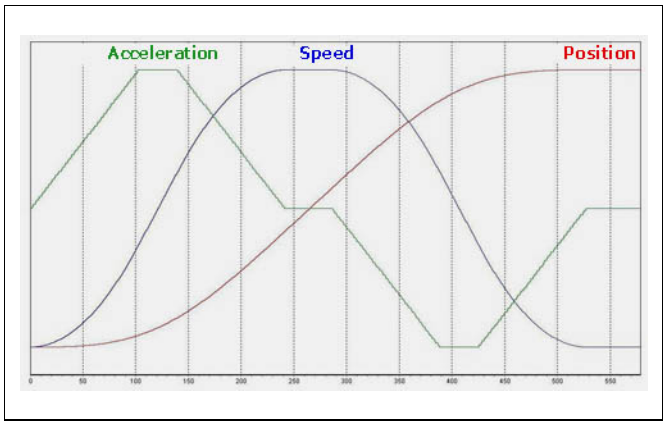
This parameter defines the ramp type: trapeze, sinusoidal, or limited jerk. These ramp type are relevant for all movements (POSA, POSR, CVEL, and MOTOR STOP), but not with SYNCx.



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 parameter *Jerk Duration* JERKMIN.

There are four different parameters for the RAMPTYPE = 2 with limited jerk, see Limited Jerk in chapter "Function and Samples".

See also program sample JerkMinTest.m.

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*.

32-86 Acc. up for limited jerk

JERKMIN

Range

0 … MLONG [ms]	∗ 100

Function

Acceleration ramp-up constant for limited jerk movements. This specifies the time in ms required to ramp the acceleration up from 0 to maximum acceleration.

There are four different JERKMIN options, see "Limited Jerk" in chapter "Function and Examples" and the parameter 32-87, 32-88, and 32-89.

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.

NB!:
A changed setting of JERKMIN 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!:
Starting with MCO 5.00 the Parameter *Jerk Duration* JERKMIN is available in the axis parameter field "Basic Settings".

Sample

Calculation sample:

VELMAX = 3000 [RPM]
ENCODER = 500 counts/rev [PPR]
RAMPMIN = 500 ms
JERKMIN = 200 ms

This results in:

VELMAX $= 3000 * 500 * \frac{4}{60} = 100{,}000$ qc/s
$= 100$ qc/ms
MaxAcc $= 200{,}000\ \text{qc/s}^2 = 0.2\ \text{qc/ms}^2$
MaxJerk $= 1{,}000{,}000\ \text{qc/s}^3 = 0.001\ \text{qc/ms}^3$

32-87 Acc. down for limited jerk

JERKMIN2

Range

0 … MLONG [ms]	∗ 0

Function

Acceleration ramp-down constant. This specifies the time in 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 par. 32-86.

NB!:
A changed setting of JERKMIN2 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.

See also “Limited Jerk” in chapter “Function and Examples” and par. 32-86.

32-88 Dec. up for limited jerk

JERKMIN3

Range

0 … MLONG [ms]	∗ 0

Function

Deceleration ramp-up constant. This specifies the number of milliseconds required to ramp the deceleration up from 0 to maximum deceleration.

NB!:
A changed setting of JERKMIN3 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 JERKMIN3 set to 0, this will default to the same value as par. 32-86.

See also “Limited Jerk” in chapter “Function and Examples” and par. 32-86.

32-89 Dec. down for limited jerk

JERKMIN4

Range

0 … MLONG [ms]	∗ 0

Function

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).

NB!:
A changed setting of JERKMIN4 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 JERKMIN4 set to 0, this will default to the same value as par. 32-86.

See also “Limited Jerk” in chapter “Function and Examples” and par. 32-86.

◻ 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:

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.

∗ default setting [ ] value for use in communication via serial communication port

$$\text{Home Velocity [RPM]} = \text{par. 33-03 Home Velocity} * \frac{\text{p. 32-80 Maximum Velocity}}{\text{p. 32-83 Velocity Resolution}}$$

ACHTUNG!:
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:

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.

Starting with MCO 5.00 the factor is no longer limited to small values, see also User Units in chapter „How to read the Design Guide".

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

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.

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*.

Starting with MCO 5.00 the factor is no longer limited to small values, see also User Units.

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.

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

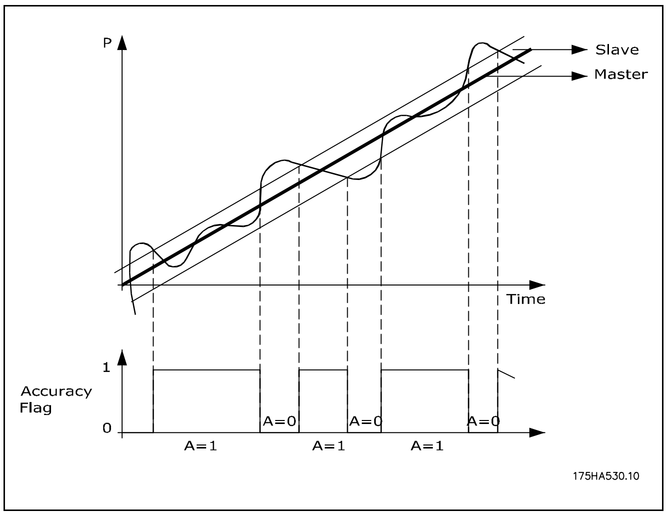
0 – MLONG =	A plus sign supplies the absolute value to SYNCERR.
–MLONG to –1 =	A minus sign supplies the synchronization error to SYNCERR with polarity sign. It is then possible to tell whether the synchronization is running ahead or behind.

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.

NB!:
The new size of the sync windows will only be active after a new SYNC-command. This does not only concern the activation but also changes in size of the windows.

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.

An extended marker type realizes a logical AND for marker input and a second input, e.g. to synchronize every second marker with a PLC control. This also influences the command MIPOS. MIPOS only delivers a new value, if the condition is true.

External master marker signal: Input 5.

∗ default setting [ ] value for use in communication via serial communication port

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

SYNCMTYPS supports the following values if ENCODERTYPE == 9

0	virtual markers are produced with a distance of ENCODERFREQ.
n2	At positive edge of input n the actual position is taken as marker position
n3	At negative edge of input n the actual position is taken as marker position

The accuracy is of course limited to 1 ms which is the internal update rate.

33-21 Master Marker Tolerance Window

SYNCMWINM

Range [Unit]

0 – par. 33-17 *Master Marker Distance* ∗ 0

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]
	SYNCMSTART 7	[7]
	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]
	SYNCMSTART 1007	[1007]
	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.

7 = Start with a polynomial 5th to reach the master exactly in a marker position.

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.

1007 = as [7], 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.

Parameter 7 and 1007 are supported starting with MCO 5.00. How the parameters operates is specified in the online help.

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*).

The *Offset Filter Time* defines the time which should be used to compensate one marker distance.

Starting with MCO 5.00 not only the offset over the given time is distributed, but also a trapezoidal movement is calculated which is superimposed onto the normal synchronizing movement.

This is used to calculate limiting accelerations and velocity with the assumption that normally 20 % for acceleration, 60 % with constant velocity, and another 20 % of the time to decelerate will be used. Based on that interpretation, it is calculated what acceleration and deceleration ramps should be used and how the maximum velocity needs to be limited to reach that.

This new distribution is not only used for offsets, but also for start corrections (first marker correction), normal marker corrections and accumulated errors caused by REVERS setting.

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 par. 33-29 *Filter Time for Marker Correction.*

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.

If par. 33-29 SYNCMFTIME is set, then the G_MMarkerDist is calculated a little differently. If the time set results in a number of markers which is smaller than 100, then at least 100 are taken to calculate the average marker distance (Filter). This only affects the calculation of the marker distance. The other filter calculations (deviation) are still done with the number of markers calculated by par. 33-29 SYNCMFTIME and actual velocity of master.

See also Marker Correction illustration in chapter "Technical Reference".

33-29 Filter Time for Marker Correction

SYNCMFTIME

Range [Unit]

0 – MLONG [ms]	∗ 0

0 = Off;
if par. 33-26 *Velocity Filter* 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 works very well and solves the problem.

Function

SYNCMFTIME is given in ms and is used as follows:

NB!:
Master velocity filtering par. 33-26 *Velocity Filter* 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.

Marker distance calculation

If SYNCMFTIME is set, then the G_MMarkerDist is calculated a little differently. If the time set results in a number of markers which is smaller than 100, then at least 100 are taken to calculate the average marker distance (Filter). This only affects the calculation of the marker distance. The other filter calculations (deviation) are still done with the number of markers calculated by par. 33-29 SYNCMFTIME and actual velocity of master.

Calculation of marker distance (if markers < 100) is changed starting with MCO 5.00.

See also Marker Correction illustration in chapter "Technical Reference" in "MCO 305 Command Reference".

33-30 Maximum Marker Correction

SYNCMMAXCORR

Range [Unit]

-MLONG – MLONG [qc]	∗ 0

0 = no limitation

Function

SYNCMMAXCORR is used to limit the maximum correction done by marker correction. This is working with SYNCM and SYNCC. The value is given in qc (slave) by the customer using the SET SYNCMMAXCORR command.

The value > 0 limits the marker correction by the given value. So if the correction would be bigger it is limited to the given value

The value < 0 sets the parameter so that no correction at all is done. The customer can switch off marker correction by this value.

Support of value < 0 is starting with MCO 5.00.

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.

33-32 Feed Forward Speed Adaptation

SYNCFFVEL

Range

0 … MLONG [per mill von VCMD]	∗ 0

0 = disabled

Function

This parameter supports velocity dependent position feed forward in synchronous modes (SYNCP/ SYNCM/ SYNCC).

This parameter is either 0 (disabled), which is default, or a value which gives the feed forward in 1/1000 of command velocity. That means a value of 1000 adds a feed forward to MPCMD (sync command position) of VCMD (command velocity). To determine the value to use, the sysvar NORMTRACKERR can be used. If you find out that your normal NORMTRACKERR is 100 percent, then you should set the SYNCFFVEL to 1000.

This allows to minimize SYNCERR. With such a feed forward the sync error could be nearly eliminated without having the disadvantages of using an INT part in the PID.

At the moment there is still the disadvantage that you have a bump if you are starting SYNCP while running. This occurs if you come up to velocity by a SYNCC or a CVEL and then start SYNCP.

New sysvar REG_NORMTRACKERR (4124), see also SDO dictionary, axis process data.

This SYSVAR gives back the track error in relation to command velocity in percent. For example, a value of 120 tells you that the track error is 1.2 times VCMD. This value is relatively constant if the conditions do not change (load, friction, …). This value typically is independent of the velocity.

The parameter is available starting with MCO 5.00.

33-33 Velocity Filter Window

SYNCVFLIMIT - 100

Range [qc]

0 … MLONG	∗ 0

0 = disabled

Function

Sync error window [qc] for automatic deactivation of 33-26 *Velocity Filter* SYNCVFTIME.

The *Velocity Filter* is deactivated, when sync error exceeds the value given by SYNCVFLIMIT.
The *Velocity Filter* is activated again, when sync error decreases below 1/5 of SYNCVFLIMIT.

This parameter helps to avoid big synchronization errors when master is accelerating or decelerating and a large value for SYNCVFTIME is used.

This parameter allows the deactivation of the par. 33-26 *Velocity Filter* if the error becomes bigger than SYNCVFLIMIT. If the filter error (PFG_G_MFILTERROR) exceeds the value of SYNCVFLIMIT (qc) then the filter is deactivated slowly. If the error becomes smaller than SYNCVFLIMIT/5, then the filter is activated again. In case of SYNCM, the internal filter error is reset every SYNCVFTIME to zero because it is only growing and never decreasing. In that case, after SYNCVFTIME it is again checked if it is small enough to reactivate it.

The filter is not disabled or enabled immediately, but it will be increased or decreased slowly by 1 ms per ms. And it is not disabled totally, but it is leaved at least at 5 ms. In the case of SYNCP, the original filter value is still used to calculate the actual filter error (PFG_G_MFILTERROR) to decide if the filter should be enabled again. In the case of SYNCM (no error correction), the filter error is reset to zero and is waited at least SYNCVFTIME to decide if the filter error stays below limit or not. If it stays below limit, then the filter is activated again.

The filter error can be observed using the sysvar PFG_G_MFILTERROR, see also SDO dictionary, axis process data.

The parameter is available starting with MCO 5.00.

□ 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 DEFORIGIN 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 DEFORIGIN 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.

It is possible to clear a software limit error and then drive in the opposite direction. If you try again to move in the wrong direction, then the error 198 occurs. In this case, it is necessary to move the drive by hand back to the admissible area and to erase the error, or in the menu *Controller* → *Parameters* → *Axis* to temporarily turn off the corresponding *Software End Limit* and then delete the error.

The behavior in case of limit switches is improved starting with MCO 5.00.

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.

It is possible to clear a software limit error and then drive in the opposite direction. If you try again to move in the wrong direction, then the error 198 occurs. In this case, it is necessary to move the drive by hand back to the admissible area and to erase the error, or in the menu *Controller* → *Parameters* → *Axis* to temporarily turn off the corresponding *Software End Limit* and then delete the error.

Improved behavior in case of limit switches is available starting with MCO 5.00.

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.

NB!:
Modified handling of TESTWIN to adapt to the needs of CANopen: If TESTTIM is set but TESTVAL is not set, then the CANopen case is assumed. In that case, it is checked if the time within the TESTWIN is longer than TESTTIM. If so, then the position has been reached. Otherwise, the position has not been reached.

TESTWIN handling is modified starting with version MCO 5.00.

□ 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.

NOTE: Limit switches and reference switches allow the usage of any input. That means also larger numbers are supported.

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.

∗ default setting [ ] value for use in communication via serial communication port

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 uncontrolled (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

The behavior in case of limit switches is changed starting with MCO 5.00:

In case of hard and software limit switches it is possible to clear a software limit error and then drive in the opposite direction. If you try again to move in the wrong direction, then a new error is generated.

Handling of hardware limit switches is the same as software limit switches. That means that you can clear the error and drive in the opposite direction. But if you try to move in the wrong direction, then the error 198 occurs.

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.

That way avoid in case of data transfer with CAN terminals that just a disturbed CAN communication stops the control. (E.g. when simple information data are transferred to a terminal and the correct display is not security relevant or the data will be updated cyclic anyway.)

Then you can check in the error routine at first, whether it is a CAN error (189) which can be deleted at once. In all other cases the control can be switched off in the error routine and the running process can be stopped corresponding the application requirements.

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.

□ 33-9* MCO Port Settings

33-90 X62 MCO CAN node ID

CANNR

Range

0 … 127 N/A ∗ 127

Function

Defines the CAN-Node ID for the MCO-Bus on the option board.

The CAN number is defined while the interface settings.

If the CANNR is set to 9999, no standard CAN objects are created. Standard CAN objects are necessary for the communication with the APOSS program when using the commands OUTMSG, INMSG, and INGLB.

33-91 X62 MCO CAN baud rate

CANBAUD

Option

	10 Kbps	[16]
	20 Kbps	[17]
	50 Kbps	[18]
	100 Kbps	[19]
∗	125 Kbps	[20]
	250 Kbps	[21]
	500 Kbps	[22]
	1000 Kbps	[24]

Function

CANBAUD defines the baud rate of the MCO bus.

The baud rate can also be set by using the APOSS command SET and the parameter CANBAUD.

Sample

```
SET CANBAUD 22    // set baud rate to 500 Kbaud
SAVE GLBPARS      // save global parameters
                  // restart the device
```

∗ default setting [ ] value for use in communication via serial communication port

33-94 X60 MCO RS485 Serial termination

RSTERMINATION

Option	
∗ Off	[0]
On	[1]

Function

Choose termination for the RS485 connection on X60.

33-95 X60 MCO RS485 serial baud rate

RSBAUDRATE

Option	
2400 Baud	[0]
4800 Baud	[1]
∗ 9600 Baud	[2]
19200 Baud	[3]
38400 Baud	[4]
57600 Baud	[5]
76800 Baud	[6]
115200 Baud	[7]

Function

Defines the baud rate of the MCO RS485 serial link.

□ 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 correspon-ding 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; corres-ponds 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	Digital 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	Conver-sion 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'	2 set-ups		Uint8
32-01	ENCODER	Incremental Resolution	1024 PPR	TRUE	2 set-ups		Uint32
32-02	ENCODER ABSTYPE	Absolute Protocol	[0] None	TRUE	2 set-ups		Uint8
32-03	ENCODER ABSRES	Absolute Resolution	8192 PPR	TRUE	2 set-ups		Uint32
32-04	ENCODERBAUD	Absolute Encoder Baudrate X55	[4] 9600	FALSE	2 set-ups		Uint8
32-05	ENCODER ABSTYPE	Absolute Encoder Data Length	25 Bit	TRUE	2 set-ups		Uint8
32-06	ENCODERFREQ	Absolute Encoder Clock Frequency	262.000 kHz	TRUE	2 set-ups		Uint32
32-07	ENCODER CLOCK	Absolute Encoder Clock Generation	[1] On	TRUE	2 set-ups		Uint8
32-08	ENCODER DELAY	Absolute Encoder Cable Length	0	TRUE	2 set-ups		Uint16
32-09	ENCODER MONITORING	Encoder Monitoring	[0] Off	TRUE	2 set-ups		Uint8
32-10	POSDRCT	Rotational Direction	[1] No action	TRUE	2 set-ups		Uint8
32-11	POSFACT_N	User Unit Denominator	1	TRUE	2 set-ups		Uint32
32-12	POSFACT_Z	User Unit Numerator	1	TRUE	2 set-ups		Uint32
32-13	ENCCONTROL	Enc.2 Control	0	TRUE	2 set-ups		Uint8
32-14		Enc.2 node ID	127	TRUE	2 set-ups		Uint8
32-15		Enc.2 CAN guard	[0] Off	TRUE	2 set-ups		Uint8
32-3*	**Encoder 1 - Master**						
32-30	MENCODER TYPE	Incremental Signal Type	[1] RS422	TRUE	2 set-ups		Uint8
32-31	MENCODER	Incremental Resolution	1024 PPR	TRUE	2 set-ups		Uint32
32-32	MENCODER ABSTYPE	Absolute Protocol	[0] None	TRUE	2 set-ups		Uint8
32-33	MENCODER ABSRES	Absolute Resolution	8192 PPR	TRUE	2 set-ups		Uint32
32-34	MENCODERBAUD	Absolute Encoder Baudrate X55	[4] 9600	FALSE	2 set-ups		Uint8
32-35	MENCODER DATLEN	Absolute Encoder Data Length	25 Bit	TRUE	2 set-ups		Uint8
32-36	MENCODER FREQ	Absolute Encoder Clock Frequency	262.000 kHz	TRUE	2 set-ups		Uint32
32-37	MENCODER CLOCK	Absolute Encoder Clock Generation	[1] On	TRUE	2 set-ups		Uint8
32-38	MENCODER DELAY	Absolute Encoder Cable Length	0	TRUE	2 set-ups		Uint16
32-39	MENCODER MONITORING	Encoder Monitoring	[0] Off	TRUE	2 set-ups		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
32-40	MENCODER TERM	Encoder Termination	[1] On	TRUE	2 set-ups		Uint8
32-43	MENCCONTROL	Enc.1 Control	0	TRUE	2 set-ups		Uint8
32-44		Enc.1 node ID	127	TRUE	2 set-ups		Uint8
32-45		Enc.1 CAN guard	[0] Off	TRUE	2 set-ups		Uint8
32-5*	**Feedback Source**						
32-50		Source Slave	[2] Enc2 X55	TRUE	2-set-ups		Uint8
32-52		Source Master	[1] Enc1 X56	TRUE	2-set-ups		Uint8
32-6*	**PID-Controller**						
32-60	KPROP	Proportional Factor	30	TRUE	2 set-ups	0	Uint32
32-61	KDER	Derivative Value for PID Control	0	TRUE	2 set-ups	0	Uint32
32-62	KINT	Integral Factor	0	TRUE	2 set-ups	0	Uint32
32-63	KILIM	Limit Value for Integral Sum	1000	TRUE	2 set-ups	0	Uint16
32-64	BANDWIDTH	PID Bandwidth	1000	TRUE	2 set-ups	0	Uint16
32-65	FFVEL	Velocity Feed-forward	0	TRUE	2 set-ups	0	Uint32
32-66	FFACC	Acceleration Feed-forward	0 %	TRUE	2 set-ups	0	Uint32
32-67	POSERR	Maximum Tolerated Position Error	20000 qc	TRUE	2 set-ups		Uint32
32-68	REVERS	Reverse Behavior for Slave	[0] Reversing	TRUE	2 set-ups		Uint8
32-69	TIMER	Sampling Time for PID control	1 ms	TRUE	2 set-ups		Uint16
32-70	PROFTIME	Scan Time for Profile Generator	[1] 1 ms	TRUE	2 set-ups		Uint8
32-71	REGWINMAX	Size of the Control Window (Activation)	0 qc	TRUE	2 set-ups		Uint32
32-72	REGWINMIN	Size of the Control Window (Deactivation)	0 qc	TRUE	2 set-ups		Uint32
32-73	KILIMTIME	Integral limit filter time	0 ms	TRUE	2 set-ups		int16
32-74	POSERRTIME	Position error filter time	0 ms	TRUE	2 set-ups		int16
32-8*	**Velocity & Acceleration**						
32-80	VELMAX	Maximum Velocity (Encoder)	1500 RPM	TRUE	2 set-ups		Uint32
32-81	RAMPMIN	Shortest Ramp	1 s	TRUE	2 set-ups		Uint32
32-82	RAMPTYPE	Ramp Type	0	TRUE	2 set-ups		Uint8
32-83	VELRES	Velocity Resolution	100	TRUE	2 set-ups		Uint16
32-84	DFLTVEL	Default Velocity	50	TRUE	2 set-ups		Uint16
32-85	DFLTACC	Default Acceleration	50	TRUE	2 set-ups		Uint16
32-86	JERKMIN	Acc. up for limited jerk	100 ms	TRUE	2 set-ups		Uint32
32-87	JERKMIN2	Acc. down for limited jerk	0 ms	TRUE	2 set-ups		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
32-88	JERKMIN3	Dec. up for limited jerk	0 ms	TRUE	2 set-ups		Uint32
32-89	JERKMIN4	Dec. down for limited jerk	0 ms	TRUE	2 set-ups		Uint32

□ 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'	2 set-ups		Uint8
33-01	HOME_OFFSET	Zero Point Offset from Home Position	0 qc	TRUE	2 set-ups		Int32
33-02	HOME_RAMP	Ramp for Home Motion	10	TRUE	2 set-ups		Uint16
33-03	HOME_VEL	Velocity of Home Motion	10	TRUE	2 set-ups		Int16
33-04	HOME_TYPE	Behavior during Home Motion	[0] Reverse + Index	TRUE	2 set-ups		Uint8
33-1*	**Synchronization**						
33-10	SYNCFACTM	Synchronization Factor Master (M:S)	1	TRUE	2 set-ups		Int32
33-11	SYNCFACTS	Synchronization Factor Slave (M:S)	1	TRUE	2 set-ups		Int32
33-12	SYNCPOSOFFS	Position Offset for Synchronization	0 qc	TRUE	2 set-ups		Int32
33-13	SYNC ACCURACY	Accuracy Window for Position Synchronization	1000 qc	TRUE	2 set-ups		Int32
33-14	SYNCVELREL	Relative Slave Velocity Limit	0 %	TRUE	2 set-ups		Uint8
33-15	SYNCMARKM	Marker Number for Master	1	TRUE	2 set-ups		Uint16
33-16	SYNCMARKS	Marker Number for Slave	1	TRUE	2 set-ups		Uint16
33-17	SYNCMPULSM	Master Marker Distance	4096	TRUE	2 set-ups		Uint32
33-18	SYNCMPULSS	Slave Marker Distance	4096	TRUE	2 set-ups		Uint32
33-19	SYNCMTYPM	Master Marker Type	[0] Enc. Z pos.	TRUE	2 set-ups		Uint8
33-20	SYNCMTYPS	Slave Marker Type	[0] Enc. Z pos.	TRUE	2 set-ups		Uint8
33-21	SYNCMWINM	Master Marker Tolerance Window	0	TRUE	2 set-ups		Uint32
33-22	SYNCMWINS	Slave Marker Tolerance Window	0	TRUE	2 set-ups		Uint32
33-23	SYNCMSTART	Start Behavior for Synchronization	[0] Start Funct. 1	TRUE	2 set-ups		Uint16
33-24	SYNCFAULT	Marker Number for Fault	10	TRUE	2 set-ups		Uint16
33-25	SYNCREADY	Marker Number for Ready	1	TRUE	2 set-ups		Uint16
33-26	SYNCVFTIME	Velocity Filter	0 μs	TRUE	2 set-ups	0	Int32
33-27	SYNCOFFTIME	Offset Filter Time	0 ms	TRUE	2 set-ups		Uint32
33-28	SYNCMFPAR	Marker Filter Configuration	[0] Marker Filter 1	TRUE	2 set-ups		Uint8
33-29	SYNCMFTIME	Filter Time for Marker Correction	0 ms	TRUE	2 set-ups		Int32
33-30	SYNCM MAXCORR	Maximum Marker Correction	[0] Off	TRUE	2 set-ups		Uint32
33-31	SYNCTYPE	Synchronization Type	[0] Standard	TRUE	2 set-ups		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
33-32	SYNCFFVEL	Feed Forward Speed Adaptation	0	TRUE	2 set-ups		Uint32
33-33	SYNCVFLIMIT	Velocity Filter Window	0 qc	TRUE	2 set-ups		Uint32
33-4*	**Limit Handling**						
33-40	ENDSWMOD	Behavior at End Limit Switch	[0] Call error handler	TRUE	2 set-ups		Uint8
33-41	NEGLIMIT	Negative Software End Limit	-500000 qc	TRUE	2 set-ups		Int32
33-42	POSLIMIT	Positive Software End Limit	500000 qc	TRUE	2 set-ups		Int32
33-43	SWNEGLIMACT	Negative Software End Limit Active	[0] Inactive	TRUE	2 set-ups		Uint8
33-44	SWPOSLIMACT	Positive Software End Limit Active	[0] Inactive	TRUE	2 set-ups		Uint8
33-45	TESTTIM	Time in Target Window	0 ms	TRUE	2 set-ups		Uint8
33-46	TESTVAL	Target Window Limit Value	1 qc	TRUE	2 set-ups		Uint16
33-47	TESTWIN	Size of Target Window	0 qc	TRUE	2 set-ups		Uint16
33-5*	**I/O Configuration**						
33-50	I_FUNCTION_1	Terminal X57/1 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-51	I_FUNCTION_2	Terminal X57/2 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-52	I_FUNCTION_3	Terminal X57/3 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-53	I_FUNCTION_4	Terminal X57/4 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-54	I_FUNCTION_5	Terminal X57/5 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-55	I_FUNCTION_6	Terminal X57/6 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-56	I_FUNCTION_7	Terminal X57/7 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-57	I_FUNCTION_8	Terminal X57/8 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-58	I_FUNCTION_9	Terminal X57/9 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-59	I_FUNCTION_10	Terminal X57/10 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-60	IOMODE	Terminal X59/1 and X59/2 Mode	[0] Output	'FALSE'	2 set-ups		Uint8
33-61	I_FUNCTION_11	Terminal X57/11 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-62	I_FUNCTION_12	Terminal X57/12 Digital Input	[0] no function	TRUE	2 set-ups		Uint8
33-63	O_FUNCTION_1	Terminal X59/1 Digital Output	[0] no function	TRUE	2 set-ups		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
33-64	O_FUNCTION_2	Terminal X59/2 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-65	O_FUNCTION_3	Terminal X59/3 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-66	O_FUNCTION_4	Terminal X59/4 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-67	O_FUNCTION_5	Terminal X59/5 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-68	O_FUNCTION_6	Terminal X59/6 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-69	O_FUNCTION_7	Terminal X59/7 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-70	O_FUNCTION_8	Terminal X59/8 Digital Output	[0] no function	TRUE	2 set-ups		Uint8
33-8*	**Global Parameters**						
33-80	PRGPAR	Activated Program Number	-1	TRUE	2 set-ups	0	Uint8
33-81	Power-up State	Power-up State	[1] Motor on	TRUE	2 set-ups		Uint8
33-82	STATUS MONITORING	Drive Status Monitoring	[1] On	TRUE	2 set-ups	0	Uint8
33-83	ERRCOND	Behavior after Error	[0] Coast	TRUE	2 set-ups		Uint8
33-84	ESCCOND	Behavior after Escape	[0] Contr. Stop	TRUE	2 set-ups		Uint8
33-85	EXTERNAL24V	MCO Supplied by External 24VDC	[0] No	TRUE	2 set-ups		Uint8
33-9*	**MCO Port Settings**						
33-90		X62 MCO CAN node ID	127 N/A	TRUE	2 set-ups		Uint8
33-91		X62 MCO CAN baud rate	[20] 125 Kbps	TRUE	2 set-ups		Uint8
33-94		X62 MCO RS485 serial termination	[0] Off	TRUE	2 set-ups		Uint8
33-95		X62 MCO RS485 serial baud rate	[2] 9600 Baud	TRUE	2 set-ups		Uint8

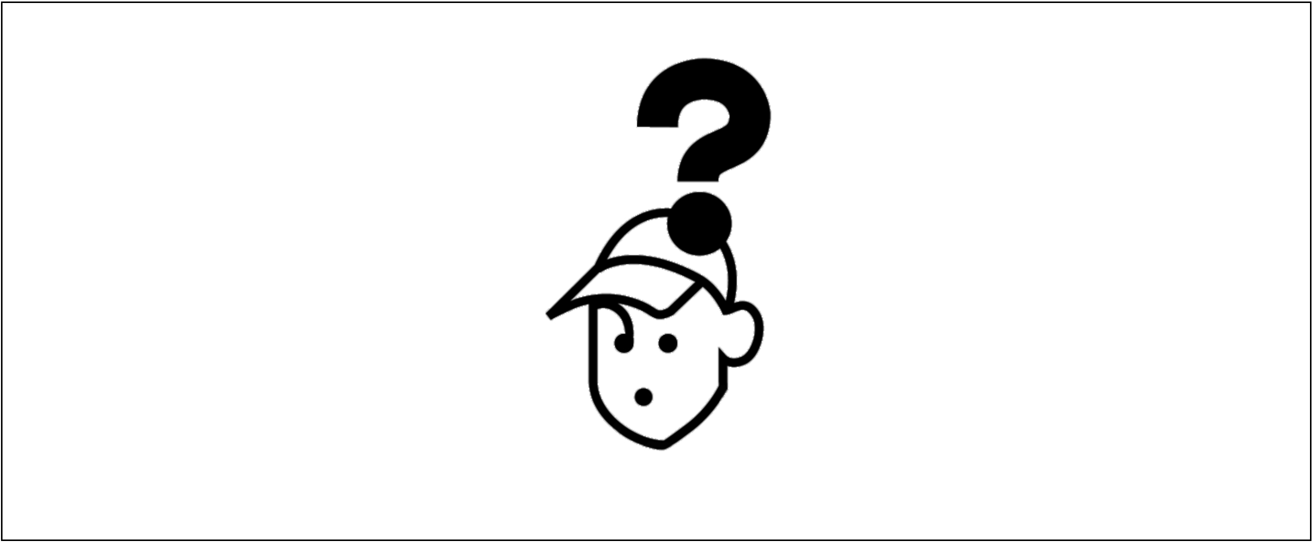
□ MCO Data Readouts, Parameter List

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			All set-ups		Uint16
34-02		PCD 2 Write to MCO			All set-ups		Uint16
34-03		PCD 3 Write to MCO			All set-ups		Uint16
34-04		PCD 4 Write to MCO			All set-ups		Uint16
34-05		PCD 5 Write to MCO			All set-ups		Uint16
34-06		PCD 6 Write to MCO			All set-ups		Uint16

∗ default setting **[ ] value for use in communication via serial communication port**

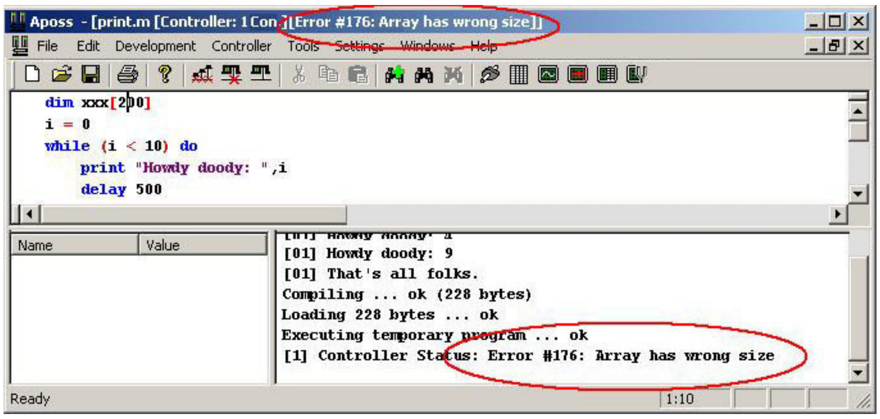
Par. No. #	Parameter Name	Parameter description	Default setting	Changes during operation	4-Setup	Conver-sion Index	Type
34-07		PCD 7 Write to MCO			All set-ups		Uint16
34-08		PCD 8 Write to MCO			All set-ups		Uint16
34-09		PCD 9 Write to MCO			All set-ups		Uint16
34-10		PCD 10 Write to MCO			All set-ups		Uint16
34-2*	**PCD Read Parameters**						
34-21		PCD 1 Read from MCO			All set-ups		Uint16
34-22		PCD 2 Read from MCO			All set-ups		Uint16
34-23		PCD 3 Read from MCO			All set-ups		Uint16
34-24		PCD 4 Read from MCO			All set-ups		Uint16
34-26		PCD 6 Read from MCO			All set-ups		Uint16
34-27		PCD 7 Read from MCO			All set-ups		Uint16
34-28		PCD 8 Read from MCO			All set-ups		Uint16
34-29		PCD 9 Read from MCO			All set-ups		Uint16
34-30		PCD 10 Read from MCO			All set-ups		Uint16
34-4*	**Inputs & Outputs**						
34-40		Digital Inputs			All set-ups		Uint16
34-41		Digital Outputs			All set-ups		Uint16
34-5*	**Process Data**		**Unit**				
34-50		Actual Position	UU		All set-ups		Int32
34-51		Commanded Position	UU		All set-ups		Int32
34-52		Actual Master Position	qc		All set-ups		Int32
34-53		Slave Index Position	UU		All set-ups		Int32
34-54		Master Index Position	qc		All set-ups		Int32
34-55		Curve Position			All set-ups		Int32
34-56		Track Error	UU		All set-ups		Int32
34-57		Synchronizing Error	UU		All set-ups		Int32
34-58		Actual Velocity	UU/s		All set-ups		Int32
34-59		Actual Master Velocity	qc/s		All set-ups		Int32
34-60		Synchronizing Status			All set-ups		Int32
34-61		Axis Status			All set-ups		Int32
34-62		Program Status			All set-ups		Int32
34-7*	**Diagnosis Readouts**						
34-70		MCO Alarm Word 1		'FALSE'	All set-ups		Uint32
34-71		MCO Alarm Word 2		FALSE	All set-ups		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 title bar and communication window including the error number and the meaning.



Aposs - [print.m [Controller: 1 Con][Error #176: Array has wrong size]]

```
dim xxx[200]
i = 0
while (i < 10) do
    print "Howdy doody: ",i
    delay 500
```

```
[01] Howdy doody: 9
[01] That's all folks.
Compiling ... ok (228 bytes)
Loading 228 bytes ... ok
Executing temporary program ... ok
[1] Controller Status: Error #176: Array has wrong size
```

Starting with MCO 305 the last 50 errors since power up are stored internally. This information is not stored in flash. It can be read out by APOSS (see Error History) or by SDO access. For every error, not only time and axis are stored, but also additional error information. This information is error specific.

Error no.	LCP display	APOSS Error message
102	Too many CAN objects	There are no more CAN objects available (CANINI).
103	Illegal axis num.	Axis 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 velocity 0
108	Position error	Position error.
109	Index not found	Index pulse (encoder) not found.
110	Unknown com.	Unknown command.
111	SW end limit	Software end limit activated.
112	Unknown param.	Illegal parameter number.
113	FC not enabled	VLT Error Status
114	Too many loops.	Too many nested loops.
115	Par. save failed	INLONG command got an illegal string
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.
121	No more SDO chn	Number of SDO channels exceeded.
125	HW end limit	Limit switch activated.
149	Too many inter.	Too many interrupt functions.
150	No ext. 24 V	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 failed.
156	Illegal double arg.	A floating point function was called with an invalid argument.
160	Internal Intr. error	Interrupt happened, but interrupt address is no longer valid.
162	Memory error	Error in verifying
170	Too many DIM arrays	Too many DIM arrays defined.
171	Array too small	Array too small
175	Out of array mem.	No more memory space for the new array defined by DIM.
176	Array size wrong	Array size does not correspond to the size of the existing array.
179	WAITNDX TO	Timeout while waiting for index.
184	Too many ontime	Too many ONTIME or ONPERIODS interrupts.
187	Out of memory	Not enough memory for variables
188	CAN guarding error	A guarding error happened.
189	CAN send-receive error	CAN send or receive error.
190	Memory locked	Memory locked
191	Illegal cam array	Illegal curve array in SETCURVE.

Error no.	LCP display	APOSS Error message
192	Encoder error	Encoder error
193	Stack overflow	Stack overflow: Too many local variables or nested function calls.
194	Out of dyn. mem	Out of dynamic memory.
195	Too many test indices	Too many test indices in data logging command.
196	Code too old	Code is too old for the current firmware.
198	Limit sw. violation	Wrong direction after limit switch tripped and error reset.
199	Internal MCO fault	Internal MCO fault

Error 102

Too many CAN objects

CANINI guarding reports an error, because there are no more CAN objects available. The optional error information (see Error History) is used as follows:

CN_TIMEOUT	-2	// timeout of CAN commands when sending or reading telegrams
NO_HARDWARE	-6	// no CAN hardware present
NO_MEMORY	-7	// no more entries available (mailboxes or lists)
NO_CANMEMORY	-10	// no more mailboxes available for define command
NO_MOBJ	-11	// the demanded mailbox is not available
CN_CANERROR	-12	// a CAN bus error is detected (low level bus error)
CN_MOBJ_DIRERR	-13	// direction of mailbox is wrong (try to read a write box or vice versa)
NO_USER	-33	// return value for sdo state (SDOSTATE).

SDO_ABORT	-50	// has to be higher than the CN_ error messages
SDO_ID_NOT_IN_USE	-33	// return value for sdo state (SDOSTATE)
SDO_SEG_ARRAY_TOO_SMALL	-51	
SDO_SEG_TOGGLE_ERROR	-52	
SDO_SEG_TOO_MUCH_DATA	-53	
SDO_SEG_NOT_ENOUGH_DATA	-54	
SDO_SEG_ARRAY_WRITE_ERROR	-55	

GUARD_ERROR_NOT_OPERATIONAL	-101
GUARD_ERROR_TOGGLE	-102
GUARD_ERROR_MODULE_NOT_RESPONSE	-103
GUARD_ERROR_NO_MODULE	-104

Error 103

Illegal axis number

A motion command is addressing an axis number greater than 1.

Error 105

Error not reset

An attempt has been made to execute a motion command, although an actual error message has not been cleared.

First clear the error message and then execute the motion command.

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 velocity 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 command

A communication or program error. The program must be re-compiled and re-loaded.

Error 111

Software end limit

A motion command will cause or has caused the software end limit to be activated. These limit are activated if actual or target position is outside software limits specified by parameters *Negative and Positive Software End Limit* 33-43 and 33-44.

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.

Starting with MCO 5.00 it is possible to clear a software limit error and then drive in the opposite direction. If you try again to move in the wrong direction, then error 198 occurs.

Handling of hardware limit switches is the same.

In positioning mode, it is 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.

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.

Limit switches and reference switches allow the usage of any input, that means also larger numbers

Software-Limit error handling a up MCO 5.00: A Software-Limit error can not be cleared: 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 the *Negative and Positive Software End Limit* in parameters 33-43 and 33-44. Only then is it possible to clear the error.

Error 112

Unknown parameter

An attempt has been made to change a parameter (SET or SETVLT command), which does not exist.

Error 113

FC not enabled

VLT Error Status: FC 300 is not enabled but the PID controller MCO 305is 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,
– or par. 8-02 *Control word source* is not set to Option C0.

Error 114

Too many loops

Too many nested LOOP commands exist in the executed program.

Error 115

Parameter save failed

Saving of the option parameter failed.

The INLONG command got an illegal string. It has been used to read a long value from serial line. If the string which arrives does not represent a valid number then this error will be released.

Error 116

Parameters memory are corrupted

The parameters in EPROM are no longer correct because of

– EPROM defective or
– power outage while saving.

NB!:
You must 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.

In APOSS stand-alone you also could use *Controller → Parameters → Reset.*

Error 117

Programs in memory are corrupted

The program data stored in EPROM cannot be found or are no longer correct because of

– EPROM 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 *Controller*.

In APOSS stand-alone delete the EPROM with *Controller → Memory → Delete EPROM* and then re-load the programs and parameters.

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 121

No more SDO channels

If a SDOREAD or SDOWRITE is used with a negative index, then the command returns immediately and stores the running SDO in a channel. It is freed when the result is read.

There is a maximum of 5 channels.

Error 125

Limit switch activated

A motion command has caused a hardware 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 (up to versions < MCO 5.00).

The behavior in case of hard and software limit switches has been improved starting with MCO 5.00: It is possible to clear a Limit error and then drive in the opposite direction. But if you try to move in the wrong direction, then the error 198 occurs.

Limit switches and reference switches allow the usage of any input, that means also larger numbers are supported.

Error 149

Too many interrupt functions

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 posint GOSUB: ON APOS, ON IPOS, ON MAPOS, ON MCPOS, ON MIPOS

Error 150

No external 24 V

External 24 V supply is missing.

Error 151

Too many nested GOSUB commands

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 commands

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 156

Illegal double argument

Mathematical error: Illegal arguments for one of the “double” function, that means, a floating point function was called with an invalid argument. For example sqrt got a negative number or asin or acos were called with an argument bigger than 1.

Double functions are available starting with MCO 5.00.

Error 160

Internal interrupt error

Interrupt happened, but interrupt address is no longer valid. (Internal error which should not ever happen.)

Error 162

Memory error

Error in verifying: After saving something in the EPROM (a program or parameters) an error was detected during verification.

Delete the EPROM 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

Too many DIM arrays

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 with *Controller → Memory → Delete EPROM* or use the command *Controller → Reset → Arrays*.

NB!:
Remember to follow the recommendations concerning saving programs and parameters before deleting the EPROM.

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 175

Out of array memory

There is no more memory space for the new array defined by a DIM command.

Error 176

Array size wrong

The size in a DIM command does not correspond to the size of the existing array.

Cause might be that the fields in the controller are from older APOSS programs and the current program has other definitions.

Either delete existing arrays or correct the DIM command with *Controller → Memory → Delete EPROM* or use the command *Controller → Reset → Arrays*.

NB!:
Remember to follow the recommendations concerning saving programs and parameters before deleting the EPROM.

Error 179

Waitndx timeout

Timeout while waiting for index: 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 183

Invalid argument

This command error signals that an TESTSTOP command contained an invalid argument.

Or a compiler error in other commands like invalid parameter value, format, or range. (Internal error, which should not occurred.)

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

Not enough memory 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 with *Controller → Programs → Delete all*

or delete both the programs and arrays, i.e. by deleting the entire memory with *Controller → Memory → Delete EPROM.*.

NB!:
Remember to follow the recommendations concerning saving programs and parameters before deleting the EPROM.

Error 188

CAN guarding error

A guarding error happened. This happens either when requesting guarding messages from slaves or if guarding is done by a master. In both cases it is caused by a time out. In the additional error information it could be seen if it was an error caused by a master guarding.

The additional error information (see Error History) is used as it is shown in error 102.

Error 189

CAN send or receive error.

This error is a send or receive error caused by a SDOREAD or SDOWRITE, by a CANIN or CANOUT or by an IN or OUT command using CAN I/O.

The optional error information either contains the CAN ID which produced the error (IN, OUT, SDO, …) or the object number (handle) which was used (CANIN, CANOUT).

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 → *Delete EPROM* 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 zbc (or 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* (→ *open* and *save*).
- Or the order of the arrays in the DIM instruction does not match the order in the zbc (or 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 193

Stack overflow

Internal error. Dynamic stack overflow caused by too many local variables or to many nested function calls.

Increase the stack size in *Settings* → *Compiler.*

Error 194

Out of dynamic memory

There is not enough dynamic memory for the requested data log (TESTSETP). Either TESTSTART demands too much dynamic memory or is called repeatedly.

Error 195

Too many test indices

The data logging command (TESTSETP) contained too many indices. There is an actual limit of 20.

Error 196

Code too old

The compiler which produced the program code was too old for this firmware.

Please use a newer APOSS.

Error 198

Limit sw. violation

After reaching the limit switch and clearing the error there was the attempt to move again into wrong direction.

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.

Compilation error ..

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* → *Programs* → *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 already exists

Connection to .. already exists - change to new 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 pin is not valid in line .. column ..

An illegal combination or a pin number which cannot be set is used with the OUT command.

Controller is executing

Controller is executing a program or command!

When the controller is executing a command or a program it is not available for additional commands. You must → *Break [Esc]* the new command and re-start it once the previous command has been completely executed.

Error in .. part of file

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.

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

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

Timeout: no reply from controller

The FC 300 does not answer; check the connection.

Index